Draft version May 21, 2024
Preprint typeset using LaTeX style emulateapj v. 11/12/01

# MANUAL FOR THE *GALAXY* *N*-BODY SIMULATION CODE

J. A. Sellwood[1]
Department of Physics & Astronomy, Rutgers University,
136 Frelinghuysen Road, Piscataway, NJ 08854, USA

Contents



[1] present address: Steward Observatory, University of Arizona, 933 N Cherry Avenue, Tucson, AZ 85721, USA

# Part I
# Preamble

## 1. INTRODUCTION

This (incomplete and evolving) document presents a description of the *GALAXY* software package[2] written almost exclusively by the author for collisionless simulations of galaxies. I have developed the code over a period of almost five decades, and continue to make refinements; it has now reached version 16.11. Pieces of the code have been described in a number of papers in the literature over the years, but here I attempt to provide a comprehensive description. The full source code is publicly available, and may be downloaded and installed as described in §3. The code may be copied freely (see §1.5), but people publishing results that use the software should acknowledge the source, and cite Sellwood (2014) where the code was announced.

The quality of any collisionless $N$-body simulation is almost entirely limited by the number of particles employed, which places a high premium on efficiency. For many applications of interest, the 3D methods described here are tens to hundreds of times faster than the popular tree codes (see §1.3), and the 2D methods have a still greater performance advantage. However, speed comes at the cost of reduced flexibility and increased danger of compromised results from inappropriate use. While the code is neither as versatile nor as easy to use as other publicly available packages, the careful user should find the increased quality of the results obtainable is ample compensation for the effort required to learn how to use it.

### 1.1. **Contents of this package**

To make use of an $N$-body code, one must first set up the model to be evolved and afterwards analyze the behavior of the simulation in order to draw conclusions. The *GALAXY* package therefore supplies three integrated pieces of code:

A The most obvious, and in many ways the easiest part to develop and describe, is the code to compute the evolution of an $N$-body system. The program `galaxy` (see §§2 & 6) advances a pre-created file of particles in time for as long as is desired. This is the core of the code, where efficiency is key. At the end, and at intermediate times if desired, it creates a file with the full information about the state of the system, from which the evolution can be continued.

B Analysis of the results is both more difficult to develop and to describe, largely because almost every project has a different scientific purpose, and therefore requires some piece of analysis that could be unique. However, most projects also make use of standard analyses of the system state and the code provides many such options. Unlike most other publicly available codes, these standard analyses in the *GALAXY* package are computed "on-the-fly" (see §13), and the results at each analysis step are saved in a file for later examination. The huge advantage of analysis on-the-fly is that the results are much more compact and can therefore be saved at frequent intervals without the inconvenience of large data storage and time-consuming analysis programs. The principal disadvantage is, of course, that a simulation must be rerun if a quantity was not measured in the first run. However, the analysis options are now well enough developed that this happens but rarely. Furthermore, the code is structured in such a way that it should be easy to add additional analysis software if needed, as also described in §13.

C By far the most difficult part of a successful $N$-body project is the creation of the desired initial set of particles for the simulation. Assigning initial positions and velocities of a specified system that is to be in equilibrium is generally hard, and it becomes especially difficult for a system, such as a disk galaxy, with multiple mass components such as a disk, bulge, and halo. The code package includes sophisticated software (Sellwood 2024, see also §15.9) that can be used in some cases, but naturally not every possible option is anticipated. A selection of possible mass components is available that are listed in §§16.1.1 – 16.1.3 and more can be added if desired by the user, as described in §16.2.

While this description of the *GALAXY* package is intended to be self-contained, there are numerous references to the excellent textbook by Binney & Tremaine (2008, hereafter BT08) where the dynamics background is explained.

This document describes the current release, v16.11, of the software. Descriptions of the differences from previous releases are given in the Appendix.

### 1.2. **Source code**

The code is written in standard Fortran (f90, but see footnote[10] on p26) and should compile without difficulty (but see footnote[8] on p13). Pieces of the code use the following **open access** software packages:

- graphics routines from *PGPLOT*,[3]
- FFTs from the *FFTPACK* (Schwarztrauber 1982),
- routines for numerical quadrature from *QUADPACK* (Piessens *et al.* 1983),
- a few routines for matrix manipulation from *LAPACK* (LAPACK User's guide 1999),
- a few routines from the NIST Core Math Library *CMLIB* (NIST Core Math Library 20xx), and
- various routines from the substantial software portal maintained by Burkardt (2017).

The source code of all these utilities is included in the *GALAXY* source code bundle except for *PGPLOT*, which will have to be downloaded and installed separately. The few *LAPACK* routines that are supplied are extracted from the whole package, as are the needed *BLAS* routines. If this package is already installed on the user's system, then it should be preferred over the sub-directory containing the part of the source code needed here.

The main programs `pcs2dmp`, `galaxy`, and `dmp2pcs` do not make use of *MPI* calls and can therefore be used only for running on a single processor. There is a separate set of executables `pcs2dmp_mpi`, `galaxy_mpi`, and `dmp2pcs_mpi` that enable calculations on multiple processors (see §6.5), which therefore require a system with some version of *MPI*. Even though these executables must be linked with *MPI*, they can also be used for running on a single processor if desired. The names of these programs used without the extension `_mpi` in this document generally imply both single processor and parallel versions.

Almost all variables in the $N$-body `galaxy` code are single precision (`real*4`), and higher precision is required in only one part (§8.11). I have verified, on a number of occasions, that the quantitative measures from a simulation are changed by insignificant amounts when all variables and arithmetic are converted to double precision (`real*8`), even in a study of stochasticity (Sellwood & Debattista 2006). The reasons that single precision is adequate are given in §10.3.

### 1.3. **Benchmark tests**

Sellwood & Debattista (2006) reported, for just one test case, that *GALAXY* ran $\sim 37$ times faster than the non-public tree code *PKDGRAV* (Stadel 2001) on a single processor. Here I report more informative benchmark tests against the public tree code *GADGET-2* (Springel 2005). While *GADGET-2* has been used quite extensively to simulate isolated galaxies, it was designed for cosmic structure formation calculations with gas dynamics that cannot be simulated with *GALAXY*. This comparison is simply to highlight the relative efficiency of *GALAXY* for stellar dynamics of isolated galaxies, and is in no way intended as a criticism of *GADGET-2*.

Fig. 1 shows the measured cpu (wallclock) time for identical files of equal-mass star particles in simulations with `galaxy` and `gadget`. The timings shown are cpu seconds per step averaged over 100 time steps. The upper panels show scaling with $N$ for 8 cores (left) and 16 cores (right), while the lower panels show scaling with the number of cores for $N = 10^7$ (left) and $N = 10^8$ (right). All other numerical parameters were held fixed as the particle number and number of cores were varied. The opening angle used in `gadget` was 0.5 radians and the particle softening length was $0.05\ell$.

The left-hand panels are for an equilibrium Plummer sphere using the spherical grid option of `galaxy` with surface harmonics $0 \le l \le 4$ included in the force determination. There were 4 time-step zones, and the grid was recentered every 8 time steps. The right-hand panels are for a thickened KT disk run using a cylindrical polar grid of size $N_R \times N_a \times N_z = 86 \times 128 \times 125$, with sectoral harmonics $0 \le m \le 8$ included in the force determination. In this case, there were 3 time-step zones, and the grid was not recentered.

Fig. 2 shows that the forces determined in the two codes are almost identical. The error bars show the dispersions, a large part of which arise from shot noise in the 1M particle distributions used in each case. The non-radial parts of the forces, shown by error bars about zero, are similar near the center, but the azimuthal smoothing in `galaxy` has the desirable property of yielding smaller non-radial forces farther out in the model. Thus for some applications, *GALAXY* is to *GADGET-2* as an FFT is to a naïve Fourier transform: it takes a fraction of the time to obtain the same result.

As may be seen from the Fig. 1, `galaxy` runs $\sim 50$ times faster than `gadget` for spherical models consistently for all $N$ and regardless of the number of processors employed. For the disk model, the speed advantage of `galaxy` over `gadget`

[3] available from http://www.astro.caltech.edu/~tjp/pgplot/ See footnote 7 for advice on installing PGPLOT.

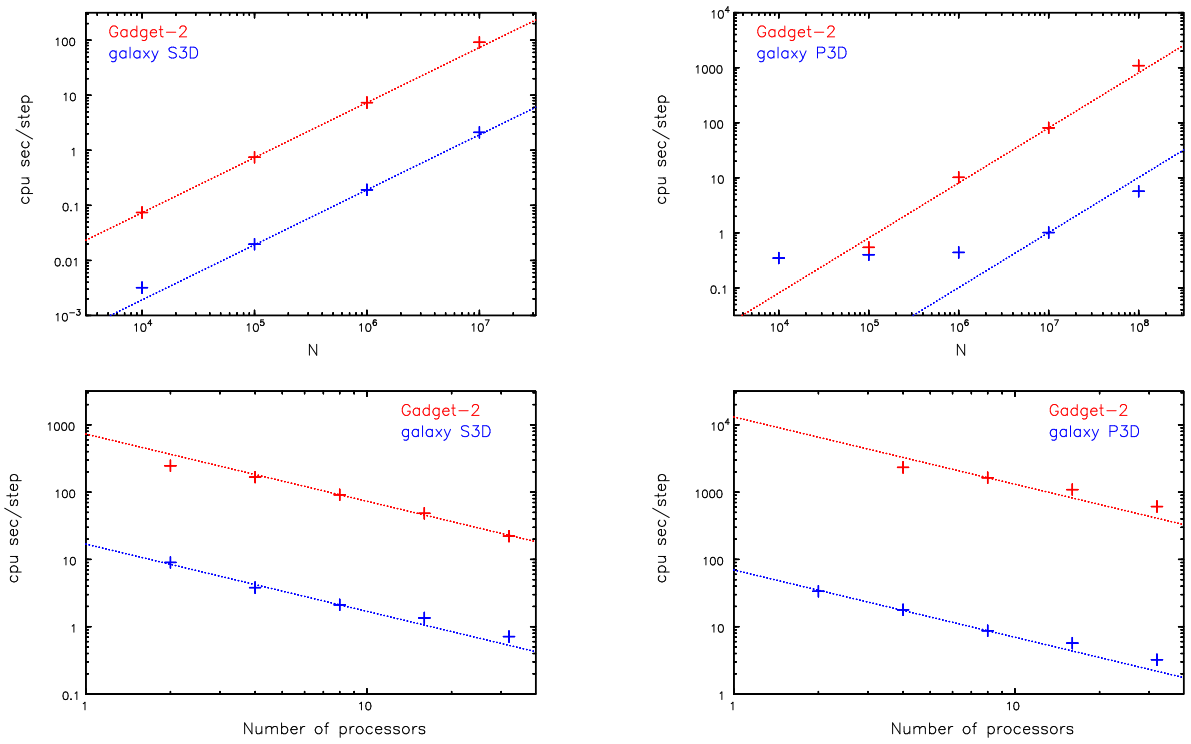


FIG. 1.— CPU timings for `galaxy` (blue) and `gadget2` (red). The left panels are for a spherical model, the right panels for a disk. The upper left panel shows the scaling with particle number $N$ for 8 cores while the upper right uses 16 cores. The lower left panel shows scaling with the number of processors for $N = 10^7$, while the right is for $N = 10^8$. The dotted lines indicate slopes $= \pm 1$, and are not fits to the data.

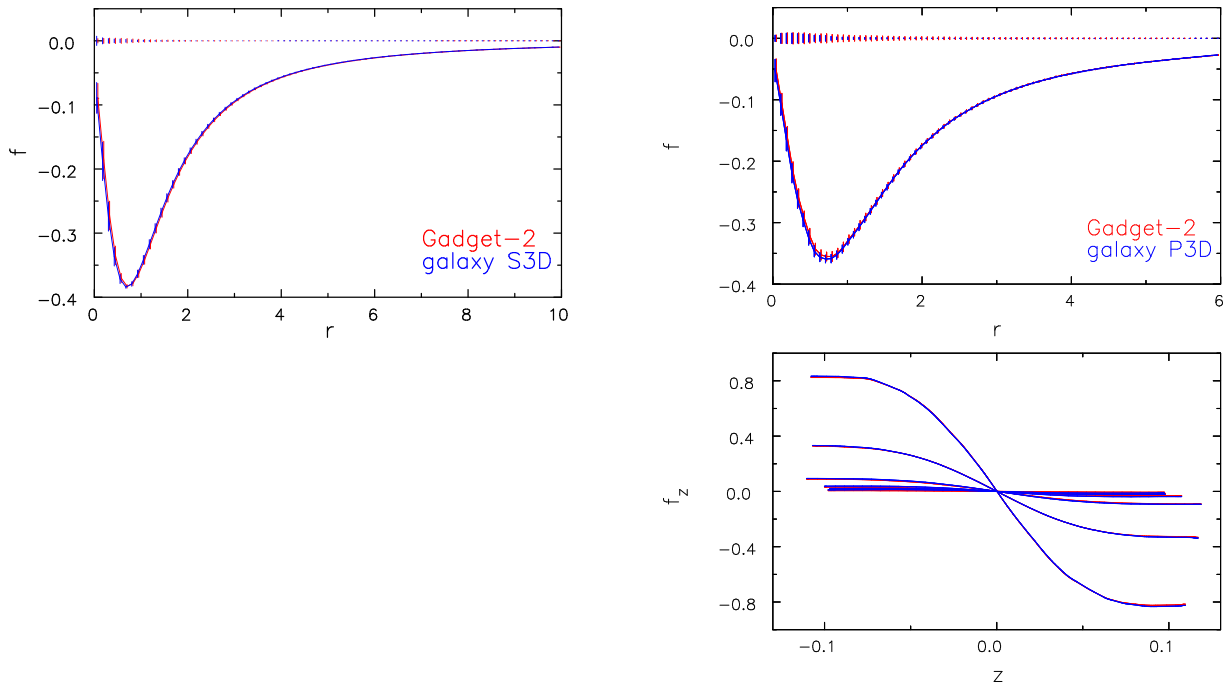


FIG. 2.— Comparisons of accelerations, in units of $GM/a^2$, determined by `galaxy` (blue) and `gadget2` (red) for identical sets of one million particles. The left panel is for a spherical Plummer model, the right panels for a Kuzmin-Toomre disk. The joined points with error bars in the left panel show the spherically averaged radial accelerations in a set of bins and the error bars give the $\pm\sigma$ variations. The upper set of error bars show the $\pm\sigma$ variations of the accelerations perpendicular to the radial direction. The plotted values from each code have been shifted horizontally in opposite directions by small equal amounts so that the differences are visible. The upper right panel shows azimuthally averaged radial accelerations in a Kuzmin-Toomre disk, while the lower panel compares the vertical accelerations at a set of radii.

gradually improves as $N$ is increased. Because there is a fixed overhead for the grid force determination, the running time for `galaxy` barely increases until $N \gtrsim 10^6$, and for very large $N$, `galaxy` can be $\gtrsim 200$ times faster than `gadget`.

### 1.4. **Disclaimers**

This free software comes with ABSOLUTELY NO WARRANTY. Users are welcome to make use of it, change it, and to redistribute it under certain conditions – see `http://www.gnu.org/licenses/` for details.

It is not expected that a would-be user could download the software provided and quickly run his/her desired simulation using it. The basic `galaxy` program should prove relatively easy to use as a "black box" and I provide a few test cases (see §14) whose results the user should be able to reproduce without additional coding. But many possible pitfalls could vitiate results from hasty use in a new application and I anticipate that a capable prospective user would need to gain many weeks of experience using the package before feeling sufficiently confident that results are publishable.

While the author welcomes reports of bugs in the distributed software, which he will attempt to fix, he is unable to provide help to modify it for some other application. The user should run the standard tests (§14) to ensure that the package has been properly installed, and report to the author if any of these tests fail on the user's system. But he cannot offer help beyond that point.

### 1.5. **License**

*GALAXY* is a package of free software, distributed under the GNU General Public License.[4] This implies that the software may be freely copied and distributed. It may also be modified as desired, and the modified versions distributed as long as any changes made to the original code are indicated prominently and the original copyright and no-warranty notices are left intact. Please read the General Public License for more details. Note that the author retains the copyright to the code.

[4] see http://www.gnu.org/copyleft/gpl.html

# 2. OVERVIEW OF THE SIMULATION CODE *GALAXY*

## 2.1. **Code structure**

The basic structure of this, and any other, $N$-body code is conceptually very simple:

1. Read a short ASCII data file to set up the simulation and select numerical parameters. Read the initial positions, velocities, and (optionally) masses of the particles from a second, previously created, file.
2. Compute the gravitational accelerations acting on each particle (§8).
3. Integrate forward the velocities and positions of the particles for a short time step (§11).
4. Repeat operations 2 and 3 as desired, possibly saving intermediate results.
5. Save the final moment in a state from which the integration could be resumed.

Step 1 presupposes that a file of particles as been prepared for use, as described in §4.

The code allows particles to have individual masses, for their motion to be integrated using a range of time steps for greater efficiency, and inserted message-passing-interface (*MPI*) calls enable its use on parallel machines with high efficiency.

## 2.2. **Computing the gravitational field of the particles**

The user of the code may select any of eleven different methods to compute the gravitational field of the particles. It is even possible to compute the field of different populations of particles in one model galaxy by different methods, or to use two different grids for interacting (spheroidal, but not disk) galaxies, in a single simulation. Even with two grids, all particles exert gravitational forces on all others.

### 2.2.1. *Grid methods*

The fastest methods to estimate the field from the particles employ a grid, or mesh, of (generally) regularly spaced points that spans a fixed volume, or area in 2D. They are particularly efficient because the number of particles employed is typically greater than the number of grid points at which the field is determined. Convenient grids can be Cartesian, cylindrical polar, or spherical. It is possible, especially with Cartesian grids, to refine the grid spacing, known as adaptive mesh refinement (AMR), but I have not developed this option. Instead, I prefer polar coordinate geometries that concentrate spatial resolution at the geometric center of the grid, which coincides with the highest density of particles in simulations of galaxies.

Grid methods can be highly optimized to compute the gravitational field at the mesh points from a distribution of masses assigned to the same set of points. Since the particles move continuously, a grid method requires schemes to assign the masses of the particles to the grid points and, subsequently, to interpolate from the field at the grid points to form an estimate of the force acting on each particle, as described in §10.

A significant limitation of grid methods is that self-consistent forces cannot be computed for particles outside the grid volume. However, their motion can be advanced using one of a number of different approximations, described in §11.6.

**Available grids**

For full 3D motion:

1. Cartesian §8.7
2. cylindrical polar §8.5
3. spherical with an expansion in surface harmonics §8.10

When motion of the particles is confined to a plane:

4. Cartesian §8.3
5. polar §8.4.

When forces acting on the particles are constrained to be axisymmetric:

6. polar axisymmetric §8.9

### 2.2.2. *Field methods*

It is also highly efficient to compute the field from an expansion in some suitably chosen set of basis functions, which I name the "smooth-field particle" method (SFP).[5] When the mass distribution is well-represented by the first few functions of the basis, and a moderate number of additional functions provide an adequate representation of the evolution, these codes can be extremely efficient and are perfectly parallelizable. However, the moment the gross mass distribution changes to a significant degree, the number of functions needed to evaluate the field increases rapidly, and code ceases to be competitive. Thus these methods are ideal for checking the stability of equilibrium models, or for measuring relaxation rates in equilibrium models, but for little else.

**Available methods**

7. For near spherical models: Hernquist SCF §8.11.5
8. For 2D disks §8.11.3, and
9. For 3D disks §8.11.4.

### 2.2.3. *Direct methods*

The least efficient, but most flexible, methods determine the gravitational forces directly from the particles. The most naïve approach is direct summation of the forces over all particle pairs, for which the computation time clearly scales as $N^2$ and rapidly becomes prohibitively expensive as $N$ is increased. Tree codes approximate the forces from distant groupings of particles, and therefore do not yield quite such accurate forces, but the approximation leads to a substantial speed up over the direct-$N$ approach, though they are still far slower than other methods described above.

**Available methods**

10. Direct-$N$ §8.12
11. Tree method §8.13

Although neither is competitive with grid or expansion techniques, I include the direct-$N$ option because it can be useful to introduce a set of very heavy particles whose motion and interactions with the more numerous lighter particles can be computed directly.

The tree code offered here is not well optimized and is included merely to make it easy for the user to compare results obtained by this popular technique with those from the better methods provided in this package. Potential users with a research project that cannot be undertaken using the grid-based methods offered here are advised to look elsewhere for a state-of-the-art tree code.

### 2.2.4. *Hybrid or 2-grid methods*

For some purposes, it can be useful to solve for the field of one population of particles on one grid and to use a different method for another. Of course it is necessary, in all such cases, that all particles in the simulation feel the attraction of all others at all times, so at least one grid must fully overlap with the other. The package offers three distinct options:

1. Logical `hybrid` = T enables two concentric grids. For example, the cylindrical polar grid, which is ideal for a disk population, and the spherical grid, which is better suited to compute the field of bulge/halo particles. Such a method was described in Appendix B of Sellwood (2003).
2. Logical `twogrd` = T allows two grids having different centers to compute the internal dynamics of two interacting, but initially separate systems. I do not have a publication describing an application to multiple, interacting systems, but I have run a few test cases of two interacting spheroidal systems only. A disk galaxy model interacting with a smaller spheroidal satellite could be computed with the current code, but the general case of two interacting disk systems with the disk planes not in the orbit plane would require some significant additional development of the code from its present version.
3. Logical `heavies` = T is useful to compute the motion of a sea of light particles using a grid method, for example, while including a small number of heavy particles whose attractions are computed directly, as described in Jardel & Sellwood (2009) and §12.4.

[5] Hernquist & Ostriker (1992) named this technique the "self-consistent field" method, but any method to compute the gravitational field from the particles is self-consistent.

### 2.3. **System of units**

Most mass models (§16) have a natural mass $M$, not always the total mass, and length scale $\ell$. The natural unit of velocity is therefore $v_{\rm dyn} \equiv (GM/\ell)^{1/2}$ and the dynamical time is $\tau_{\rm dyn} \equiv (\ell^3/GM)^{1/2} = \ell/v_{\rm dyn}$.

In a single component model, it makes sense to work in units where $G = M = \ell = 1$ so that $\tau_{\rm dyn} = v_{\rm dyn} = 1$ also.

But we need to be able to set masses and length scales for each separate component in composite models. In this case, the user must choose one component to define the length and mass scales, and then supply relative values of these quantities for the other components. Note that the only significance of the notional mass $M$ is to set the mass scaling, it need not be the total mass of one, or of all, components, or of all the particles. Initial coordinates of the particles must be supplied in these units, and all analysis output is given in the same units.

Many people prefer kpc for distance and km/s for velocity, *etc.* Conversion between sets of units should not be hard. All the user need do is adopt physical values for $M$ and $\ell$ and all other conversions follow. Input distances must be divided by $\ell$, input velocities by $v_{\rm dyn}$, input masses by $M$, *etc.* A convenient example of scaling to physical units is to choose $\ell = 3$ kpc and a time unit $\tau_{\rm dyn} = 10$ Myr, which yields $M = 6 \times 10^{10}$ $\rm M_\odot$ and velocity unit $v_{\rm dyn} \simeq 293$ km/s.

### 2.4. **Scaling the simulation**

There are two key variables that the user must specify to determine the spatial and time resolution of the simulation.

- `lscale` specifies the value of $\ell$ in grid units – *e.g.* the number of mesh spaces on a Cartesian grid. Spatial resolution is improved as the value of `lscale` is increased, but it cannot be arbitrarily large since the model must still fit within the grid. Furthermore, some particles may move outwards as the simulation evolves, and the grid should be large enough that no more than a small fraction leave the active grid – see §11.6.
- `ts` specifies the length of the basic time step in units of $\tau_{\rm dyn}$. The smaller the value of this parameter, the more accurate the time integration, but the trade-off is that the simulation will require more time steps to compute the desired evolution. Before v16.0, the input value was that of the shortest step, with a recommended value that particles near the center took $\sim 200$ steps to complete an orbit. This changed in v16.0 and the input value is now the time step that would be appropriate for particles in the perifery of the model, and most particles are stepped forward in integer fractions of this step.

Clearly, `lscale` has dimensions of inverse length and `ts` of time. All quantities within the code are scaled and stored in internal units in order to avoid multiplications and/or divisions by constants when advancing the motion, locating the nearest mesh point, *etc.* Conversion factors from internal to external units are given in Table 3 after the Appendix.

### 2.5. **Adaptive time steps**

It is usually necessary to subdivide the time step where the acceleration is high, and the code allows for a hierarchy of block time steps that are reduced as needed by successive factors of two. The maximum number of possible levels is specified by the user.

Before v16, the selection of the appropriate time step depended on the position of the particle within a predefined set of spatial zones, which was the only option. A new option in v16 is to allow the time step to depend on the acceleration to be applied to the particle. The scaling from acceleration to time step is controlled by a user-input dimensionless parameter, with a recommended value of 0.05 – larger values allow longer time steps and conversely. However, there are circumstances where spatial time step zones are preferred. See §11.2 for more detail.

### 2.6. **Files**

All I/O files associated with the *GALAXY* code use the naming convention: runX.yyyT where X is an integer whose value must lie in the range $10 \le$ X $\le 9999$. The string yyy is a three character extension; some recognized extensions are listed in Table 1, not all of which are needed in every simulation, and others can be created. If a T is appended to the extension, it is an integer giving the evolution stage, in dynamical times, that the simulation had reached when the file was created.

A user wishing to open a file for their own I/O purpose may obtain a logical unit number for the file by a call to subroutine `new_unit`. It will return, via the calling argument [*e.g.* `call new_unit`(iunit)] a logical unit number that is, and will remain, unique during execution.

## 2.7. **Memory organization**

The main arrays are allocatable to avoid the need to reset parameter variables and recompile whenever the simulation is resized. This is the principal feature of the f90 extension to f77 that the code embodies. Other variables are held in common blocks that are declared in "`include`" files.

The particle coordinates and additional information are stored in the 2D array `ptcls`. The first dimension is `nwpp` elements arranged since version 15[6] as follows:

1. the first 6 (4 in 2D) elements are the current phase space coordinates $(x, y, z, v_x, v_y, v_z)$
2. if particles do not all have the same mass (§15.9.2), the next element is the particle mass
3. if the BH tree option is selected (§8.13), the next element is the particle's index in the tree
4. the next element is an integer flag specifying the mass component to which it belongs
5. the next, and final, element is the link to the next particle in the list (see §11.5.1)

Note that items 1 & 2 are floating point quantities (`real*4`), while 3 through 5 are type `integer`. Since the array `ptcls` is declared to be type `real*4`, the `integer` elements must be accessed through two local variables, one type `real*4` the other type `integer`, that are `equivalenced` to each other. (In Fortran, this means the two variables share the same memory location.) Subroutine `chklst` contains a clear example.

[6] In versions 14.4 and earlier, the arrangement differed slightly: 1, 4, 2, 3 & 5.

Table 1
Principal file types

| Extn | Description |
|---|---|
| .dat | a short ASCII text file to input the options and parameters |
| .lis | an ASCII text file of output to which progress is logged |
| .res | a binary data file which contains the results of the on-the-fly analysis |
| .dmp | a binary file that contains the coordinates of all the particles, ancilliary information needed for a seamless restart of the simulation |
| .dfn (.df1, df2, ...) | a binary file of initial radii, velocities, and (optionally) masses of a single population of particles selected from a DF for use in `mkpcs` |
| .pcs | a binary file containing a header line together with all the particle coordinates and masses that can be used as input to `pcs2dmp` and/or is output by `dmp2pcs` |
| .flg | a very short ASCII file that enables the duration of a simulation to be altered during execution (see §6.2) |
| .grt | an ancilliary binary file needed by some grid-based Poisson solvers |
| .grd | a second ancilliary binary file needed only when checking some grid-based Poisson solvers |
| .pft | a binary data file with the spline coefficients of the total potential fitted to a composite model |
| .cmp | a binary data file with the total potential and density distributions of models that have been compressed adiabatically |
| .stb | a binary data file with a table of values at different radii giving the supplemental central attraction to correct for softening *etc.* |

# Part II
# Using the code

## 3. DOWNLOADING AND COMPILING THE CODE

The installation assumes *PGPLOT*[3,7] is installed. If the user wishes to run in parallel mode, a version of *MPI* will also be needed. Otherwise, the software bundle supplied should be self-contained, and none of its programs require linking with commercial software packages. Proceed with the following steps:

1. To download the software, go to the website

   `http://www.physics.rutgers.edu/galaxy`

   Enter your email address and the file `galaxy.tgz` will download to your system automatically. (The email address will enable you to be notified when updates are available.)

2. This gzipped $\sim$ 2MB tar file can be moved from your Download directory to some convenient location in your directory tree and then unpacked with the command:

   `tar xzf galaxy.tgz`

   and all the files of the package will be created in the directory `GALAXY16` and subdirectories thereof. The scripts and Makefile assume that the user will be using `gfortran`[8], and will need to be edited if a different compiler is desired.

3. Build libraries from the source code. (All `rebuild` scripts use `gfortran`[8] by default.)

   ```
   cd GALAXY16/SRC/lib16
   ./rebuild
   cd ../RAJ
   ./rebuild
   cd ../utils
   ./rebuild
   ```

   These scripts should create the three object libraries `main16.a`, `RAJ.a`, and `libutils.a` in the directory `GALAXY16/lib`.

4. Users not wishing to run simulations in parallel should skip to step 5. To run in parallel, users will need an implementation of *MPI*. The scripts and Makefile assume *OpenMPI*[9] and they will need to be edited if a different installation of *MPI* is to be used. The mpi compiler is assumed to be `mpif90`

   ```
   cd ../lib16_mpi
   ./rebuild
   ```

   This script should create the additional object library `main16_mpi.a` in the directory `GALAXY16/lib`.

5. ```
   cd ../progs
   ```

   To inform the compiler of the location of the *PGPLOT* object library, edit the line that begins `pgplot_lib =` in the Makefile (in the `progs` subdirectory).

6. Compile the code:

   ```
   make
   make clean
   ```

[7] Advice on installing PGPLOT: The package is well documented, and the instructions are generally easy to follow. However, some additional information is required. First, to be compatible with the *GALAXY* package, the following device drivers need to be selected: /GIF, /PS, /VPS, /CPS, /VCPS, and /XWINDOW. Second, on 64-bit machines only, the declaration statement for the variables `PIXMAP` and `WORK` in the subroutine `gidriv.f`, which can be found in the `/drivers` sub-directory of the package, needs to be changed to type INTEGER*8. Third, the `makemake` command has many different compiler options that, unfortunately, do not include `gfortran`, which is the recommended compiler for the *GALAXY* package. The easiest way to overcome this limitation is to select the `g77-gcc` option as the last parameter of the `makemake` command and then use an editor to substitute `gfortran` for `g77` in the resulting `Makefile`.

[8] Compilations with some recent versions of `gfortran` may fail with the error message "Argument mismatch". This is not a real error, and can be overcome by adding the compiler option `-fallow-argument-mismatch`.

[9] available from http://www.open-mpi.org/software/ompi/v1.8/
   ```

The `make clean` will not only tidy up the directory, but will move all the executables to the directory `GALAXY16/bin`, which should be added to your `PATH` through commands such as (or equivalents for your shell):

```
export PATH=.:$HOME/GALAXY16/bin:${PATH}:
```

7. To activate the option of running in parallel

```
cd ../progs_mpi
make
make clean
```

The `make clean` will add four `xxx_mpi` executables to the directory `GALAXY16/bin`

The following executables of the *GALAXY* package should now be ready for use:

`analys, cold, compress, corrplt, dfiter, dflook, dmp2pcs, estfreq, gadget2pcs, galaxy, genplt, isotropy, merge, mkpcs, modefit, pcs2dmp, plotpft, ptest, smooth, tipsy2pcs, weed` and (optionally) `dmp2pcs_mpi, galaxy_mpi, pcs2dmp_mpi, ptest_mpi`

Executables without the `_mpi` suffix are for single processors only. The use of all these programs is described in the remainder of this document. From v16.11, most programs can be invoked with the run number as a parameter on the command line, but the program will prompt for this datum if the command line parameter is omitted, as in earlier releases.

## 4. CREATING A FILE OF PARTICLES

All simulations require a `.pcs` file of initial particle coordinates that are to be advanced in time. This section describes three ways in which the file can be created.

### 4.1. **Running** `mkpcs`

The author has invested considerable effort to create equilibrium models from which simulations can be started. Program `mkpcs` sets up such models from the ASCII `.dat` file and, possibly, auxiliary files that result from the various pieces of code described in §15.

#### 4.1.1. *Using* `.dfn` *files*

The program `mkpcs` can either generate particles at random from specified density distributions and assign them velocities so that the model is in approximate equilibrium, or it can input information about a particle population that was previously created by one of the sophisticated methods described in §15. For example, it can load a precalculated `.dfn` file to set up a good equilibrium halo in the presence of a disk, while the disk particles are generated randomly from the desired radial and vertical density profiles. The velocities of the disk particles are then selected in the manner described in §15.6. A `.dfn` file for just one population can have the name `runX.dfn`, but if multiple files for different components are needed, they should be named `.df1`, `.df2`, *etc.*, each numbered for the corresponding mass component given in the `.dat` file.

#### 4.1.2. *Quiet starts*

The user can specify whether a quiet start is to be created. This is achieved by arranging mirror particles, with identical radii and velocities in polar coordinates, that are equally spaced around rings and, if 3D, copied into a reflection symmetric ring on the opposite side of the mid-plane. This procedure suppresses shot noise in the low-order terms of the density distribution, at the expense of a greatly enhanced signal from the order of rotational symmetry in the ring. If no terms higher than half that adopted symmetry contribute to the gravitational field, then each ring of particles behaves as a smooth wire that can support low-order distortions. Since the azimuthal variations of the field are tiny, instabilities will grow from very low amplitude, allowing a reasonably accurate growth rate to be estimated.

#### 4.1.3. *Initial gravitational field*

The desired model is specified in the ASCII `.dat` file, which is usually the same file that is read by `galaxy`. The gravitational attraction of the initial density distribution is computed by the selected method in order to set initial orbital velocities. Where possible, the density used will be a smooth distribution that is either the sum of the analytic density functions of each component or it is computed by integrating the DF over all velocities; this latter method is preferred because it takes full account of energy bounds, *etc.*, that may be needed to limit the radial extent of the component. If no smooth density can be created, then the code will use the actual particles; note that azimuthal averages reduce, but do not eliminate, shot noise in the resulting attraction.

If requested, the code will also determine and save the corrective central attraction that may be needed to ensure virial equilibrium, as described in §15.10.

#### 4.1.4. *Setting initial orbital velocities*

Once the full central attraction of the model is available, the program assigns velocities to those particles for which a DF has not been specified, as described in §15.6.

#### 4.1.5. *Eliminating drift*

The user has the option to make small adjustments to the positions and/or velocities in order to force the center of mass to coincide with the coordinate origin and/or to eliminate any net momentum of the model, that might have arisen during particle selection. This option will never be needed if particles are arranged on rings in a quiet start.

#### 4.1.6. *Setting bulk motions*

If the particles belong to two spatially separate mass distributions, or one plus a rigid perturber, that are in orbit around each other, the program will apply the appropriate spatial and velocity shifts to each set of particles to create the desired initial conditions.

#### 4.1.7. *Writing the file*

After all these steps are completed, the program creates the `.pcs` file.

Since any numerical parameter can be selected afresh when loading a `.pcs` file for evolution, it is possible, but not recommended, to use a different method, grid scaling, *etc.* to compute the forces from the particles in `mkpcs` from those to be used in `galaxy`.

### 4.2. **Creating your own `.pcs` file**

The user may, if he or she wishes, use their own software to create a file of particles at the initial moment that is to be evolved using `galaxy`. This file of *binary* (or unformatted) data must contain one or two short header records followed by many separate records each containing no more than (typically) 5000 particles. The format of these data is described below. It may seem a little arcane that the particles must be divided into groups in this manner, but a single Fortran record of binary data containing more than $10^8$ particles, say, may be too large to be read all at once, and Fortran does not offer a convenient way to read parts of a very long record.

The first header record must contain, eight 4-byte words: $\mathtt{n}_1$, $\mathtt{n}_2$, $\mathtt{n}_3$, `ncoor, np, time, pm, pertbn`.

- The first three integers specify the number of particles, which could be zero, in each of three mass components; the total $\mathtt{n}_1 + \mathtt{n}_2 + \mathtt{n}_3 = \mathtt{nbod}$, the number of particles in the simulation. The ASCII `.dat` file that is to be used by `pcs2dmp` and by `galaxy` must list each active mass component in the same order as they appear in the `.pcs` file.
- `ncoor` is type integer and specifies the number of words of information for each particle. This is equal to the number of phase space dimensions, normally 6 for the 3 positions and 3 velocities but could be 4 if the particles are all confined to the plane $z = 0$. If the particles have individual masses, the value should be increased by one to 7 (or 5).
- `np` is type integer and generally has the value 5000, which is the number of particles in each group. Files created with $\mathtt{np} \neq 5\,000$ can usually be read, but files output by `dmp2pcs` will always have $\mathtt{np} = 5\,000$.
- `time` is type real and should normally have the value 0 at the start, and in files output by `dmp2pcs` this variable will record the number of dynamical times (see §2.3) since the start.
- `pm` is type real, and gives the nominal mass of a single particle in external units (see §2.3). If particles have equal masses, then `pm` specifies their mass. If particles have unequal masses, then `pm` is a scale factor, which could be unity, for all the individual masses specified later in the file (see eq. 1).
- `pertbn` is type logical. If T, then the simulation will include one of the external perturbing fields described in §12.1. If F, no such external perturber is included (the usual case).

A second header record is required only if the last variable of the first header, `pertbn`, has the value T. The perturber could be an orbiting rigid mass or a central softened point mass, or a rigid rotating bar, *etc.* (see §12.1). If present, this record must supply initial values for `xptrb, vptrb, mptbr, eptbr`.

- `xptrb` is real vector of length 3 which specifies the initial $x, y, z$ position of the perturber center.

- `vptrb` is real vector of length 3 which specifies the initial $v_x, v_y, v_z$ velocity of the perturber.
- `mptrb` is type real and gives the nominal mass of the perturber in external units (see §2.3).
- `eptbr` is type real, and gives the core radius of the perturbing mass in external units (see §2.3).

The remaining records give the particle coordinates and, optionally, masses. There are `ncoor` words giving the $(\boldsymbol{x}_j, \boldsymbol{v}_j, [w_j])$ coordinates of the $j$th particle, plus optionally the scaled mass of that particle, its actual mass is $\mu_j = \mathtt{pm} \times w_j$. These values should be in the external units described in §2.3. A complete record therefore contains

$$\mathtt{ncoor} \times \mathtt{np} \quad \text{real*4 words}$$

The file should contain multiple records of this size, and probably conclude with a final record of shorter length to make up the exact number of particles: $\mathtt{n}_1$ + $\mathtt{n}_2$ + $\mathtt{n}_3$. If unequal particle masses are supplied, then the total mass of the $i$th component is

$$M_i = \sum_{j=1}^{\mathtt{n}_i} \mu_j = \mathtt{pm} \sum_{j=1}^{\mathtt{n}_i} w_j. \tag{1}$$

### 4.3. **Converting a *GADGET-2* or *TIPSY* file**

Program `gadget2pcs` attempts to convert a "restart-file" of particles that has been created for `gadget` into a `.pcs` file that can be used for `galaxy`. The *GADGET-2* file can contain only halo, disk, and bulge particles that do not require co-moving integration, and there can be no gas, star, or boundary particles.

The default assumption is that physical quantities are given in the system of units recommended in the *GADGET-2* documentation: *i.e.*, position coordinates are in kpc from the center, velocities are in km/s, and masses are in units of $10^{10}$ solar masses. If a different scaling is used, the user will have to recode the unit conversion.

This interactive program asks for the name (max 80 characters) of the *GADGET-2* restart-file to be converted. If reading and rescaling the data it contains is successful, then the program asks for the name to be used for the output file, and creates it.

To make use of this file in a run of `galaxy`, it should be renamed to `runX.pcsT`, as described in §2.6. It is simplest for the `runX.dat` to specify the types of any halo, disk, and/or bulge components as 'UNKN' and DF types to 'NONE' to avoid crashes due to "unrecognized component" in a number of subroutines. Running `pcs2dmp` will set up the `.dmp` file to be used by `galaxy`.

A similar utility program `tipsy2pcs` is also supplied to convert files from the U Washington $N$-body shop, but is not user-friendly!

## 5. THE ASCII INPUT FILE

The programs `mkpcs, pcs2dmp, galaxy`, and `dmp2pcs` read a short ASCII file of input instructions for the run which *must* be named `runX.dat`. An example is provided on the next page, with line numbers prepended for reference only – *i.e.* the line numbers are *not* part of the file.

The # character is used to separate data from comments in a line, so a line beginning with # is ignored. Information before a # on any line is read. The first 10 characters is a string, of which the first few are interpreted as a keyword that is *not* case sensitive. The remaining information from column 11 to the # is parsed for the information expected to follow the keyword.

The program outputs a message and terminates if an expected keyword is not found. This can happen if a line is missing, for example. It also outputs the line and terminates if the syntax of the parameters is incorrect – *nb.* from version 15.22, a tab character is accepted in place of multiple spaces.

The purpose of, and values on, each line are explained after the example. The lines in the file fall into groups that are read by different subroutines as follows:

A Subroutine `getset` reads lines A1 & A2

B Subroutine `msetup` reads lines B. The first specifies the number of mass components, which can be disks, bulges or halos, each of which might be composed of particles or a rigid mass. This routine then expects a few lines of data per mass component explaining what type it is and giving values of mass and length scale in units of $M$ and $\ell$ as explained in §2.3. A list of available mass componets and the information needed for each is given in §16.1.

C Subroutine `setgrd` reads the next group of lines, which gives numerical details such as the numbers of grid points in each dimension, the softening length and type of kernel, the number of sectoral harmonics to include, *etc.* It should be clear that the precise keywords and values on each line, and the number of lines needed varies with the type of grid, as explained further in §5.2.

D Subroutine `setrun` reads most of the remaining lines which relate to scaling to grid units, time-stepping, the number of particles and their properties, and the types of analysis to be performed, as explained further in §5.3.

E The last line is read by program `galaxy`

In a little more detail:

Line A1: the run number of the simulation. The numerical value given here *must* correspond to the "`X`" field in the file name (see §2.6). Allowed values are $11 \leq$ `X` $\leq 9999$.

Line A2: the type of field solution method that will be used. Valid names are: c2d, p2d, p3a, p3d, c3d, s3d, sf2d, sf3d, scf, dr3d, bht, hybrid, and twogrid (see §2.2). If it is either of the last two, then two more lines must be supplied with the actual field solution methods

Line B1: the number of different mass components in the model. Currently this is restricted to the range $1 \leq n \leq 5$

Lines B2 – B6: properties of one indivdual mass component as described briefly below (§5.1) and in full detail in §16.1. This group of lines must be repeated as many times as the number of mass components given in line B1.

Lines C1 – C5: information for the selected field determination method. The number of these lines and their content depends on the field determination method selected, as described below (§5.2). Lines C1 – C5 (or their equivalents) must be repeated for as many field determination methods as were indicated in line A2. *NB.*, the softening length is input in **grid** units, *i.e.* the value of `lscale` $\times(\epsilon/\ell)$. It has to be this way because the value of $\ell$ in grid units has not yet been set. I should try to remove this blemish one day.

Lines D1 – D20 are read by subroutine `setrun` and are described in §5.3 below.

Line E1: read by program `galaxy`: the desired end time of the simulation

| | Key word | par | par | Comment |
|---|---|---|---|---|
| A 1 | run no | 1000 | | |
| A 2 | grid type | p2d | | |
| | # | | | |
| B 1 | ncom | 2 | | # number of distinct mass components |
| | # | | | |
| B 2 | disc | t | | # toggle for a disk |
| B 3 | type | exp | | # type keyword |
| B 4 | mass | 0.5 | | # mass in units of $M$ (§2.3) in this component |
| B 5 | scale | 1.0 | 5.0 | # length scale and truncation radius in units of $\ell$ (§2.3) of this component |
| B 6 | dftype | none | 1.5 | # DF type and parameter or initial $Q$ |
| B 7 | taper | t | 4.0 | # toggle for outer taper of a truncated disk and its start radius |
| | # | | | |
| B 2 | disc | f | | # toggle for a disk |
| B 3 | type | asdi | | # type keyword |
| B 4 | mass | 0.5 | | # mass in units of $M$ in this component |
| B 5 | scale | 1.0 | 20.0 | # length scale and truncation radius in units of $\ell$ of this component |
| B 6 | dftype | none | | # DF type and optional parameter(s) |
| | # instructions for p2d | | | |
| C 1 | grid size | 106 | 128 | # mesh dimensions (*i.e.* $N_R, N_a$ for this grid type) |
| C 2 | HASH | 2 | | # highest active sectoral harmonic |
| C 3 | softl | 1.0 | | # softening length $\epsilon$ (see §9) in **grid** units |
| C 4 | sect | 2 | | # order of rotational symmetry imposed |
| C 5 | skip | 0 | | # turn off any selected sectoral harmonics |
| | # instructions for setrun | | | |
| D 1 | time step | 0.05 | | # longest step in units of $\tau_{\rm dyn}$ (§2.3) |
| D 2 | zones | 2 | | # number of levels of subdivision of the basic time step (see §11.2) |
| D 3 | zonacc | t | 0.05 | # toggle for acc. dep. time steps and, if 't', a dimensionless scale factor |
| D 4 | # | 2 | 1.0 | # step ratio and radius inside of which the shorter step is used |
| D 5 | guard rads | 1 | | # number of guard radii (see §11.4) |
| D 6 | | 2 | 0.1 | # step fraction and radius inside of which the shorter step is used |
| D 7 | lscale | 8 | | # value of $\ell$ (§2.4) in grid units |
| D 8 | offgrid | t f | f f | # toggles to select options for off-grid particles (see §11.6) |
| D 9 | uqmass | f | | # toggle for individual particle masses - 'f' is default when this line is omitted |
| D10 | #cntd | 0.4 | 1 | # recenter grid at this time interval using scheme 1 (*NB.* instruction is commented out) |
| D11 | supplement | f | | # toggle for supplementary forces - 'f' is default when this line is omitted |
| | # | | | |
| D12 | icmp | 1 | | |
| D13 | npar | 500000 | | # number of particles in this component |
| D14 | start | t | f f | # logicals for dist, smr & quiet (see §4.1.2) |
| D15 | end data for component 1 | | | |
| | # | | | |
| D11 | icmp | 2 | | |
| D12 | npar | 0 | | # number of particles in this component – 0 indicates it is a rigid component |
| D14 | end data for component 2 | | | |
| | # | | | |
| D15 | analysis | 2.0 | | # time interval (in units of $\tau_{\rm dyn}$) between analyses |
| D16 | # moni | 100 | | # monitor 1 particle in every 100 (*NB.* this instruction is commented out) |
| D16 | plot | 20.0 | | # time interval for pictures |
| D16 | dvel fld | 5 | 12 | # vel moment bins - radial bins per `lscale` then azimuthal |
| D16 | lgsp | 61 | 1 | # number of different pitch angles and sectoral harmonics |
| D16 | hbins | 100 | | # number of angular momentum bins |
| D16 | frqs | 20 | | # number of values per `lscale` of $\Omega_c$, $\kappa$, and (if 3D) $\nu$ |
| D17 | save | | | # further instructions for results file |
| D18 | intg | | | # global integrals ($E$, $\boldsymbol{L}$, *etc.*) |
| D18 | danl | | | # sectoral harmonic analysis of the density on grid rings |
| D19 | end of "save" instructions | | | |
| D20 | end | | | # last data card for `setrun` |
| | # | | | |
| E 1 | end time | 100.0 | | # in units of $\tau_{\rm dyn}$ |

### 5.1. **Inputs for the different mass components**

The number of separate mass components, disks, bulges, halos, *etc.* should be given in line B1. Full details of all available mass components and how to specify them are given in §16.1. Briefly, each indivdual mass component should be specified as follows:

Line B2: a simple T or F toggle to indicate whether it is a disk

Line B3: a 4 character keyword (not case sensitive) to indicate the type, which must have one of the values given in §16.1.1 or §16.1.2. A numerical parameter should follow in some cases.

Line B4: (not always needed) the mass unit of this component in units of the nominal mass $M$ (see §2.3)

Line B5: the length scale of this component in units of the nominal length scale $\ell$ (see §2.3) and its outer edge radius in units of $\ell$.

Line B6: a 4 character keyword (not case sensitive) to indicate the type of DF, which must have one of the values given in §16.1.3 and, if needed, the parameter value(s) for that DF. No value is required if the DF keyword is "NONE", except if the component is a disk when the parameter must give the initial $Q$.

line B7: for a truncated disk, a logical variable to turn on an optional outer taper (quite desirable) and the inner radius, in units of $\ell$, at which the taper begins. A reasonable value is $\sim 80\%$ of the disk truncation radius entered in line B5.

Lines B2 – B6/7 must be repeated as a group as many times as there are mass components indicated in line B1

### 5.2. **Inputs for the different grid types**

1. **C2D** 2D Cartesian grid

```
C 1   grid size    256   256      # mesh dimensions N_x, N_y
      # powers of 2 recommended
C 2   softl        2.0   1        # softening length ε in grid units units and softening rule
      # The softening length is in units of h_x (§8.3). A value less than unity here makes no numerical sense
      # Softening rules are: 1 for Plummer and 2 for cubic spline (§9)
      # If the second parameter is absent, 1 is set as default
```

2. **P2D** 2D polar grid

```
C 1   grid size        106   128      # mesh dimensions (N_R, N_a)
      # N_R can have any value, but FFTs in azimuth require that N_a has no large prime factors
C 2   HASH             2              # highest active sectoral harmonic
      # Highest Active Sectoral Harmonic, or m_max in §8.4
C[2a  POWEr law  0.5                  # optional input: index γ (see §8.4) – usually omitted
      # ring spacings are exponential by default, i.e. γ = 1]
C[2b  UOFFset    -20                  # optional input: grid ring for inner boundary – usually omitted
      # 0 by default so that the grid has no hole – see §8.4]
C 3   softl            1.0   1        # softening length in grid units units and softening rule
      # The softening length is in units of h_R (§8.4). A value less than unity causes excessive grid noise
      # Softening rules are: 1 for Plummer and 2 for cubic spline (§9)
      # If the second parameter is absent, 1 is set as default
C 4   sect             2              # order of rotational symmetry imposed
      # e.g. a value 2 filters out all odd sectoral harmonics
C 5   skip             0              # a list of sectoral harmonic to exclude
      # the list could be empty, but any enumerated harmonics will be excluded from the self-consistent field.
      # The entry 0 turns off radial forces
```

3. **P3D** 3D cylindrical polar grid

```
C 1   grid size      100   128   225   # mesh dimensions (N_R, N_a, N_z)
      # as for P2D, but N_z can have factors of only 3 and/or 5
C 2   z spacing      0.1               # in grid units
      # the spacing of the grid planes in units of h_R (§8.4). Restriction Δz < ε
C 3   HASH           8                 # highest active sectoral harmonic
      # Highest Active Sectoral Harmonic, or m_max in §8.4
C[3a  POWEr law      0.5               # optional input: index γ (see §8.4) – usually omitted
      # γ = 1 by default, and ring spacings are exponential]
C[3b  UOFFset        -20               # optional input: grid ring for inner boundary – usually omitted
      # 0 by default so that the grid has no hole §8.4]
C 4   softl          0.5   2           # softening length in grid units and softening rule
      # The softening length is in units of h_R (§8.4). Softening rules are: 1 for Plummer and 2 for cubic spline (§9)
      # If the second parameter is absent, 2 is set as default
C 5   sect           1                 # order of rotational symmetry imposed
      # as line C 4 for P2D above – the value 1 leaves all sectoral harmonics m ≤ m_max active
C 6   skip                             # list of any additional sectoral harmonics to skip
      # as line C 5 for P2D above, a blank field means none is skipped
```

4. **C3D** 3D Cartesian grid

```
C 1   mesh size   129   129   129   # mesh dimensions (N_x, N_y, N_z)
      # values of 2^n + 1 are required
C 2   shape       1.    1.    1.    # cubic cells
      # this line can be used to change the aspect ratio of the grid cells. Cubic cells are strongly recommended,
      # however.
```

5. **P3A** 3D axisymmetric grid

```
C 1   grid size   111   375         # mesh dimensions (N_R, N_z)
      # as for P3D, N_z can have factors of only 3 and/or 5
C 2   power law   0.0               # index γ (0 gives equal spacing)
      # the rule is R_u = h_R u^{1/(1-γ)}
C 3   z spacing   0.1               # in grid units
      # any value is allowed
C 3   skip                          # enter 0 here for active vertical forces only
```

6. **S3D** Spherical grid method

```
C 1   grid size   301   110.        # number of radial grid shells (N_r) and the radius of the grid outer boundary
      # No restrictions on either number
C 2   lmax        10                # value of l_max
      # the highest order of surface harmonic contributing to the field
C 3   sect        2                 # order of symmetry imposed
      # a value 2 causes all odd-l values to be skipped
C 4   skip                          # list of any additional surface harmonics to skip
      # as line C 5 for P2D above
```

7. **SF2D** Smooth field expansion in 2D

```
C 1   Abel-Jacobi      7                    # basis choice: the number is k for this basis
C 2   minm      maxm   minn   maxn          # basis functions to be used
C 3     2         2      0     20           # m, n = selected sectoral harmonics and radial functions
C 4   minmaxr     1.0                       # min & max allowed value of MAXR (function edge)
C 5   maxmaxr     1.0                       # as a fraction of the initial disc edge
C 6   sect          2                       # order of rotational symmetry imposed
      # a value 2 causes all odd-m values to be skipped
C 7   skip                                  # list of any additional sectoral harmonics to skip
      # as line C 5 for P2D above
```

8. **SF3D** Smooth field expansion for a thickend disk

   Warning: the selection of $k$ values in the following lines has not been tested in a meaningful simulation!

   ```
   C 1  Bessel     0.5                  # basis choice and δk – only Bessel fns allowed
   C 2  softl      0.0                  # Plummer softening length, usually zero
   C 3  minm   maxm   minn   maxn       # Bessel functions to be used
   C 4     2      2      0     20       # m,n = selected sectoral harmonics and number of k values
   C 5  ntrab    101                   # number of radii to tabulate functions
   C 6  planes    21    0.1            # number of planes and z-spacing
   C 7  sect       2                    # order of rotational symmetry imposed
        # A value 2 causes all odd-m values to be skipped
   ```

9. **SCF** Smooth field expansion for a spherical system (as Hernquist & Ostriker 1992)

   ```
   C 1  Plummer   10.          # basis set and outer radius
   C 2  lmax       8           # highest spherical haronic
   C 3  nmax      20           # highest radial function
   C 4  sectors    2           # order of spherical symmetry imposed
        # a value 2 causes all odd-l values to be skipped
   C 5  skip                   # list of any additional surface harmonics to skip
        # as line C 5 for P2D above
   ```

10. **DR3D** A simple direct-$N$ method in 3D

    ```
    C 1  softl   0.05   2      # softening length and softening rule
         # The unit of length is defined later in the input stream
         # Rules: 1 for Plummer and 2 for cubic spline (§9) with 2 as default if the second parameter is absent
    ```

11. **BHT** Barnes-Hut tree method

    ```
    C 1  softl      0.05   2      # softening length and softening rule
         # The unit of length is defined later in the input stream
         # Rules: 1 for Plummer and 2 for cubic spline (§9) with 2 as default if the second parameter is absent
    C 2  tolerance   0.5           # maximum angle before opening boxes – i.e. the value of θ_max in §8.13
    ```

### 5.3. **Inputs to set up time stepping, analysis,** *etc.*

Many parameters for a run of `galaxy` are set in the ASCII input lines read by subroutine `setrun`. This routine reads and parses the lines by a 4-character keyword (not case sensitive) that must be the first 4 characters of most lines. Not all keywords need be present and the order of the key-worded lines is unimportant, although some keyworded lines may require subsequent lines to be read before the next keyword. The use of each of the 26 valid keywords is as follows.

#### 5.3.1. *Required keywords related to the evolution*

1. LSCAle: Required keyword plus a single type real value, which is the basic length scale $\ell$ in grid units. See §2.3 for the definition of $\ell$ and §2.4 for advice on choosing the value of `lscale`.

2. TIME step: Required keyword plus a single type real value, which is the basic time step in units of $\tau_{\rm dyn}$. See §2.3 for the definition of $\tau_{\rm dyn}$ and §2.4 for advice on the choice of time step.

3. OFFGrid: Required keyword plus four type logical values, which are instructions for what to do with particles leaving the grid (see §11.6)

4. ICMP: Required keyword plus a single type integer value giving the mass component number to which the subsequent input lines apply. A group of these lines is required for each separate mass component.

   (a) NPARticles: Required keyword plus a single type integer value giving the number of particles to be used to represent this mass component. A zero value signifies that the mass component is rigid.

   (b) STARt type instructions: This keyword is required only if the mass component is represented by mobile particles. It should be followed by three T/F toggles: (i) T means the initial coordinates will be supplied in a `.dfn` file (§4.1.1), (ii) T means the radial mass profile should be as close as possible to that of the mass component (rarely useful), and (iii) whether the particles will be replicated around rings – a "quiet start". Values (i) can (ii) cannot both be T, but both can be F.

   (c) PGRD (the grid for this mass component). This keyword is required only if the simulation employs more than a single grid and if the mass component is composed of particles. The type integer value specifies the initial grid (*e.g.* 1 or 2 for the 1st or 2nd specified in part C) for particles of this mass component.

(d) Z0IN (the initial thickness $z_0$ of this population.) This keyword is required for 3D disks only. One or two values must also be supplied. The first (type real), specifies the scale height $z_0$ in units of $\ell$, while the optional second (type integer) specifies the vertical mass profile type. Possible values of this integer are:

1 Spitzer analytic isothermal sheet, with the theoretical vertical dispersion. This choice is not recommended for the reasons given in §15.8

2 Gaussian vertical profile (default)

3 $\mathrm{sech}^2$ vertical profile, but not Spitzer's dispersion

4 rounded exponential vertical profile (Solway *et al.* 2012)

5 exponential vertical profile

If the second parameter is omitted, the value 2 is assumed by default. For options 2 thru 5, the vertical balance is set as described in §15.8.

(e) CPOP (the initial center of mass of this component.) Optional keyword followed by three real values giving the $(x, y, z)$ coordinates of the center of mass of the component. The default is that the center of mass of each component is at the coordinate origin.

(f) VPOP (the initial motion of the center-of-mass of this component.) Optional keyword followed by three real values giving the $(v_x, v_y, v_z)$ components of the velocity of the center of mass of the mass component – this bulk velocity is added to the internal velocities of each particle in the component. The default is $(0, 0, 0)$.

(g) END Required keyword to end this short sequence for each mass component.

### 5.3.2. *Optional keywords related to the evolution*

5. ZONEs: Optional keyword plus a single type integer value, which is the number of levels (`nzones`) of subdivision of the basic time step (see §2.5): $1 \leq$ `nzones` $\leq 10$. If this keyworded line is omitted `nzones` is set equal to 1 and all particles move with the same time step. If `nzones` $> 1$, the shortest time step is a fraction $2^{1-nzones}$ of the basic step.
If `nzones` $> 1$, the next line must give the value of the logical variable `zonacc`. If `zonacc` = `.T.`, changes to the time step depend on the acceleration (as of v16), but `zonacc` = `.F.` reverts to spatially defined zones, which was the only option in earlier versions. If `zonacc` = `.T.`, then the value of the dimensionless constant that scales the time step criterion is required as the second parameter.
If `zonacc` = `.F.`, the second parameter is ignored, and the next `nzones` $- 1$ lines are read to find the radii (in units of $\ell$) of the boundary inside of which the next shorter step is implemented. The zone boundaries must be sequenced in increasing radii so that longer steps are taken farther from the center.

6. GUARd radii: Optional keyword plus a single type integer value, which is the number of different guard radii `nguard` (see §2): $0 \leq$ `nguard` $\leq 10$. If this keyworded line is omitted `nguard` is set equal to 0.
If `nguard` $> 0$, the next `nguard` lines must give the radius in units of $\ell$ of the boundary inside of which the next shorter time step is used. The boundaries must be sequenced in decreasing radius such that longer steps are taken farther from the center.

7. CNTD (or centered): Optional keyword plus a type real and a type integer value. The first specifies the time interval, in units of $\tau_{\rm dyn}$, between grid recenterings, which must be a multiple of the basic step, and the second indicates which rule is to be used to find the center (see §8.14). The default is not to recenter the grid, if the line is omitted or commented out. If the line is present, but with only the time interval specified while the second parameter is missing or zero, then rule 1 is adopted as the default.

8. PERTurbation: Optional keyword plus a character string as the first value. The default is no perturbation, if the line is omitted or commented out.
If the line is present, then the character string must be one of the following:

(a) BAR: A rigid bar that is a homogeneous, prolate spheroid assumed centered on the grid and rotating about its minor axis. The 4 further values are all type real: (i) the bar mass, (ii) the bar semi-major axis, (iii) its initial value of $\Omega_p$, and (iv) the bar ellipticity.

(b) SPIRal: A rigid spiral pattern to perturb the disk. The two further values are of type real: (i) the spiral pattern speed $\Omega_p$ and (ii) and an amplitude scale.

(c) GNRC a generic perturber: the nature of this perturber is described in §12.1.1. As for other perturbers, the keyword line requires two extra type real values giving the mass and scale size (both in external units §2.3). The next two lines give, respectively, the $(x, y, z)$ position coordinates and the $(v_x, v_y, v_z)$ velocity components of the perturber center in the same units.

(d) One of the halo types listed in §16.1.2: A rigid external perturber which has the density profile of the named component. The keyword line requires two extra type real values giving the mass and scale size (both in external units §2.3).
The next two lines give, respectively, the $(x, y, z)$ position coordinates and the $(v_x, v_y, v_z)$ velocity components of the perturber center in the same units.

9. REST: optional keyword plus two parameters of type logical. If the first of the parameters is T, then all the coordinates of all the particles, and the perturber if present, are shifted to locate the initial center of mass of the system at the grid center. If the second logical is T, then any initial net momentum is subtracted from the velocities of all the particles. These options are inconsistent with quiet starts (§4.1.2), which achieve the same result by other means. This option affects only the start up procedure; dynamic recentering of the grid is much more useful and may be activated with the CNTD option.
If no line has this keyword, the default is to leave the particle coordinates unchanged from their initially generated values.

10. SUPPlementary forces: Optional keyword plus a single type logical value. T or F toggle to supplement the central attraction on each particle with the difference between the theoretical Newtonian attraction and the mean central attraction of the particles at the start (see §15.10 for a fuller description). The default value is F, if this line is absent.
Note that supplementary forces would be redundant when fixed radial forces are required, which is the case when the axisymmetric part of the forces from the particles is skipped, which is set by a zero in the SKIP line for the force determination method (see §5.2).

11. UQMAss: Optional keyword plus a single type logical value. `.T.` or `.F.` toggle to specify whether particles have unequal masses. If this line is absent, the default value is `.F.` and all particles have equal masses.

### 5.3.3. *Required keyword related to analysis*

12. ANALysis: Required keyword plus a single type real value giving the interval, in units of $\tau_{\rm dyn}$, between analyses. This interval MUST be an integral multiple of, or equal to, the basic time step.

13. SAVE (types of analysis that the user may wish to examine, but which do not require values). Required keyword with no value. Subsequent lines specify the requested types. Valid values, which are all optional except for the terminating instruction, are:

    (a) DANL sectoral harmonic coefficients on each grid ring (P2D and P3D only)
    (b) DNST the density of particles assigned to the grid, which is usually projected to the mid-plane
    (c) INTG totals of kinetic and potential energies, virial of Clausius, angular momenta, linear momenta, and center of mass, computed separately for each population of particles, as well as Ostriker's $t$ value.
    (d) LVAL the specific angular momentum of every particle (which makes the .res file very large)
    (e) MOIT compute the moment of inertia tensor for non-disk particle populations. The particles of each population are divided into 10 groups ranked by their binding energies and the MoI tensor computed for each.
    (f) PNTL the values of the potential at every grid point
    (g) S3DC the values of coefficients of the surface harmonic expansion used in the S3D method only
    (h) END : Required keyword to flag the end of this sub-list.

### 5.3.4. *Optional keywords related to analysis*

14. LGSP (logarithmic spiral transforms as described in §13.1.1.8). Optional keyword plus two type integer values giving (i) the number of different pitch angles, centered on $\alpha = 90^\circ$ (radial) and spaced by $\Delta \cot \alpha = 0.25$, and (ii) the number $n_m$ of sectoral harmonics for this analysis. The values computed will range $n_{\rm sect}(1 \le m \le n_m)$, where $n_{\rm sect}$ is the order of rotational symmetry imposed in the solution for the gravitational field. This analysis is computed separately for each disk population. The default is not to perform this analysis.

15. FRQS Optional keyword requesting that the values of $\Omega$, $\kappa$, and (if 3D) $\nu$ be evaluated at a set of radii from azimuthally averaged forces in the mid-plane of the simulation. The type integer parameter specifies the number of values per scale length $\ell$ to be computed, and the measurements extend beyond the grid edge.

16. DVEL (disk velocity field as described in §13.1.1.7). Optional keyword requesting that first and second moments of the disk particle velocities be computed in cylindrical polar bins. The two type integer values specify (i) the number of radial bins per `lscale` with measurements made to 1.2 times the outer initial radius of the largest disk, and (ii) the number of azimuthal bins. This analysis is computed separately for each disk population. The default is not to perform this analysis.

17. HBIN (angular momentum bins) Optional keyword requesting that the particles of each population be binned by the $z$-component of their angular momentum. The single type integer value specifies the number of bins. This analysis is computed separately for each population. The default is not to perform this analysis.

18. HVEL (halo velocity field as described in §13.1.1.7). Optional keyword requesting that first and second moments of the halo particle velocities be computed in cylindrical polar bins. The two type integer values specify (i) the number of radial bins per `lscale` with measurements made to the outer initial radius of the largest halo, and (ii) the number of vertical slices. This analysis is computed separately for each spheroidal population. The default is not to perform this analysis.

19. MONItor If particle motion is confined to a plane (2D), this option saves $E$, $L_z$, $\boldsymbol{x}$, and $\boldsymbol{v}$ for a fraction of the particles. In 3D, only $E$ and the three components of $\boldsymbol{L}$ are saved. Optional keyword plus a single type integer value that gives the stride through the particle array used to select particles. The default is not to perform this analysis.

20. PLOT (to save information to create snapshots of the particle distribution). Optional keyword plus a type real value giving the interval, in units of $\tau_{\rm dyn}$, between snapshots. Irrespective of the number of particles in the simulation, the maximum number saved for this analysis never exceeds $10^4$ in each component, and can be less. The default is not to save data for snapshots.

21. RHOR (compute the density profile as described in §13.1.1.2). Optional keyword plus a single type integer value that gives the stride through the sorted radii used to record the radii and mass enclosed. This analysis is computed separately for each population of particles. The default is not to perform this analysis.

22. RNGS (introduces rings of massless test particles, as described in §13.1.1.12). Optional keyword plus two or three type integer values giving (i) the number of rings, which are evenly spaced to the largest outer radius of the massive particle distribution, (ii) the number of particles per ring, and (iii) the grid number assigned to these test particles. This last number is needed only when multiple grids are used. The default is not to introduce rings of test particles.

23. SPHB (spherical Bessel function analysis as described in §13.1.1.10). Optional keyword plus two type integer parameters giving (i) the number of different radial functions $n_{\rm max}$ and (ii) the number of angular harmonics $l_{\rm max}$ for this analysis. This analysis is computed separately for each spheroidal population. The default is not to perform this analysis.

24. ZANL (vertical bending analysis as described in §13.1.1.9). Optional keyword plus two type integer parameters giving (i) the number of radial annuli, and (ii) the number $n_m$ of sectoral harmonics for this analysis. The values computed will range $n_{\rm sect}(1 \le m \le n_m)$, where $n_{\rm sect}$ is the order of rotational symmetry imposed in the solution for the gravitational field. This analysis is computed separately for each disk population. The default is not to perform this analysis.

25. ZPRF (vertical profile analysis as described in §13.1.1.11). Optional keyword plus two type integer values giving (i) the number of different radial annuli, and (ii) the number of vertical bins in which the density is computed. This analysis is computed separately for each disk population. The default is not to perform this analysis.

#### 5.3.5. *Required keyword to end input to* `setrun`

26. END : Required keyword to flag the end of data for subroutine `setrun`.

The program galaxy requires an ASCII .dat file and a **restart file** (.dmp) containing the state of the model at some instant, which it then evolves forward in time. As desribed above, the .dmp file contains not only the coordinates of the particles, but additional information, such as the mass distribution most recently assigned to the grid, and other details that enable galaxy to integrate forward from the given moment as if the simulation had not been interrupted. This is a desirable feature for the user, since it enables a rerun to perfectly replicate the previous evolution. However, the .dmp file has a structure that is intimately related to the galaxy code and is therefore highly obscure. Thus I supply the program pcs2dmp to create this arcane file from more user-friendly inputs.

## 6.1. **Creating a .dmp file**

Program pcs2dmp reads in a file of particles (.pcs) containing initial positions, velocities, and (optionally) masses. The structure of this file is described in §4. The program also scales the input coordinates to internal (program) units, divides particles into their various time step zones, time centers the initial velocities, and initializes the linked lists. The final step is to output all the information needed for galaxy into the restart file (.dmp).

When adaptive time steps depend on the acceleration, they cannot be chosen until the full gravitational field of all the mass components has been calculated. In this case, the program pcs2dmp initially assumes that all particles move with the largest (basic) time step, and only when the gravitational field is known can it call subroutine rezone to implement substeps. Once that is done, the particles masses are then reassigned to the copies of the grids for the apppropriate time steps.

As mentioned in §11.7, the .dmp file is designed to contain all the information needed by galaxy to evolve a model forward in time. The program galaxy creates a new version of the .dmp file from time to time, and always at the end, from which the evolution could be seamlessly continued.

As a precaution, program pcs2dmp first checks whether a .res file already exists. New data will be appended to this file, which may be what the user intends, but the file should first be deleted if it contains unwanted data from a previous false start. If no .res file exists, pcs2dmp creates one and writes at least the header (see §13.4).

## 6.2. **Computing evolution**

Once the .dmp file is created, galaxy will integrate the evolution forward, outputting the requested analysis information to the .res file at each analysis step. It is not necessary to interrupt execution in order to examine this file using programs described in §13. Indeed it is strongly recommended that the user use this capability at an early stage to check that the model was indeed set up as intended and is in equilibrium (if that was intended), as well as at later times to determine whether the model is evolving as expected.

The program galaxy will also output .dmp files at intermediate times, which are named runX.dmpT, with T being the stage, in dynamical times (§2.3) that evolution has reached. Since each file has a new time extension, they do not overwrite each other. These intermediate .dmp files are potentially useful for a number of reasons: (1) should the computer crash through a hardware fault or power outage, say, the simulation does not need to be restarted from the beginning, but can be resumed from the most recent complete .dmp file. (2) They could each be converted to .pcs files to enable the user to analyze them using his/her own software.

### 6.2.1. *Controlling the length of the integration*

The program galaxy will continue to execute until the end time, specified in the last line of the .dat file, has been reached. It will then output a new .dmp file and execution will terminate.

However, the user can arrange an earlier or later graceful exit, if desired, by manipulating a tiny ASCII file with extension .flg. At the start of execution, the final step number read from the .dat file is copied to this file, which is then closed. As the simulation progresses, galaxy periodically checks for the existence and contents of this file. If the file is no longer present, then the simulation will exit gracefully, saving the .dmp file so that execution can be seamlessly resumed. If the file is present, it will revise the end time to the value read from this file. Thus by changing the end time in this file, *e.g.* with a text editor, the user can request either that the evolution be continued for longer than originally foreseen, or can request a graceful exit at an earlier time. If the simulation has already passed the new end time, then this is equivalent to deleting the file, since the simulation will soon exit gracefully. (Of course, most operating systems offer ways to terminate execution abruptly, if a graceful exit is not needed!)

### 6.3. **Re-creating simple particle files**

The program `dmp2pcs` is designed to do the opposite of `pcs2dmp`: it will remove all extraneous information from a restart file (extension `.dmp`), convert particle coordinates to their time-centered values in external units, and create a simple file of particle coordinates (extension `.pcs`) at the new time. Such files can be useful for analysis that is more sophisticated than that which is performed "on-the-fly".

Note that the order of the particles in the later `.pcs` files is the same as that in the first. Particles are not rearranged as the evolution proceeds.

### 6.4. **Why** `pcs2dmp` and `dmp2pcs` **are not part of** `galaxy`

Obviously, running `pcs2dmp` on the new `.pcs` file would recreate a `.dmp` file from which the evolution could be continued. This is a useful option if one wishes to make changes to the numerical parameters: one can adjust the scaling between natural units and grid units, adjust the time step or zoning, use a different number of processors (see §6.5), or even change to a different grid type, *etc.*

All such changes will alter slightly the subsequent evolution from that which would have occured had the run been continued with the same numerical scheme and parameters. In fact, even continuing the run with the same scaling and numerical parameters will cause the subsequent evolution to deviate from that which would have occurred had the simulation not been interrupted, and because simulations are stochastic, the evolution could diverge macroscopically in some circumstances (see Sellwood & Debattista 2006, and §11.7). Thus a strategy to preserve only `.pcs` files from which evolution could be continued is deprecated.

### 6.5. **Running in parallel**

The principal advantage of parallel operations is that the particles are spread evenly between processors, which reduces the cpu time needed to process them through one time step in strict inverse proportion to the number of processors employed.

Of course, the mass arrays created by each processor must be combined, and the gravitational field of the whole model must be made available on each processor before the next step begins. The communication overhead for this part is tiny, when the spherical grid (§8.10) and basis methods (§8.11) are used, as the amount of data to be combined is very small. However, grid methods both require larger amounts of data to be exchanged and the calculation of the field is a fixed overhead. I have therefore created special versions of some subroutines that parallelize some parts of the field computation by some, but not all, grid methods. When the field determination method is not parallelized,[10] each cpu completes the entire solution for the field, which is faster than waiting for one cpu to do the job and then transferring the results. While this approach could perhaps be optimized, the code still benefits from sharing the particles between cpus, and performance is respectable when $N$ is large.

The only force determination methods that cannot (so far) be used when running in parallel are the direct-$N$ and tree code methods.

The package supplies a separate set of executables that include *MPI* instructions to enable the calculation to be distributed over multiple processors. These executables have the same names with the extension `_mpi` as `pcs2dmp_mpi`, `galaxy_mpi`, *etc.* The main programs without the `mpi` extension do not require *MPI* and can therefore be run in single processor mode only. Those with the `mpi` extension do require *MPI*, and can be run either in single processor mode with the command

```
galaxy_mpi
```

or in parallel, using 10 processors say, with the command

```
mpirun -np 10 galaxy_mpi
```

For any given run, the number of processors used by `galaxy_mpi` must be the same as for `pcs2dmp_mpi`, since the restart information is available in a number of files that is equal to the number of processors.

Since each processor advances the motion of a fraction of the particles, no one processor can create a single `.dmp` file that contains all the information needed for a restart. Thus each processor saves a partial `.dmp` file, and these must be combined, using program `dmp2pcs_mpi` to create a complete `.pcs` file. It has proven more efficient to do this in a separate stand-alone process than to hold up the evolution within `galaxy_mpi` while all the information is passed to one processor, simply in order to output a single `.dmp` file.

[10] The code written by Richard James to solve Poisson's equation on a Cartesian grid contains the Fortran 95 extension of OpenMP instructions. However, there does not appear to be an easy way to implement this capability within the current code architecture, since it would attempt to share the work to cpus that are already being utilized. I have tried using mpi instructions to parallelize the Fourier transforms by sharing the transforms on separate planes of the grid. Unfortunately, this strategy yielded a negligible gain in speed, because broadcasting the results from each plane to the other nodes took as much time as completing the transforms on each separate node!

## 7.1. General analysis

The interactive program `analys` enables the results to be displayed. You enter the run number, and have the option to enter additional run numbers if you wish to compare results from several simulations. Entering '0' will terminate the input of run numbers.

The default is to display results in the units described in §2.3. However, some (but not yet all) of the graphs produced by `analys` can be marked in physical units. If this option is selected, you must input the desired values of $\ell$, in kiloparsecs, and of $\tau_{\rm dyn}$, in millions of years. The consequent conversion factors for mass and velocity are then displayed, and you have the option to start over if unsatisfied.

You will then be asked to select an output device, which is most usefully your screen (xterm) but the program can also be used to create diagrams in the form of PostScript files. The gif file option is most useful for making movies; a sequence of gif files can be combined to make a movie using *GIMP*, which is a publicly available *GNU* product.

Program `analys` offers many options that may be selected by a 4 character keyword. Further inputs are needed within each option, and the user can navigate from each back to the main program to examine other aspects of the evolution. The program is reasonably user friendly, in that it prompts the user to re-enter inputs if the syntax is wrong, or the values are non-sensical, or if the requested data are not available, and there are a few help menus.

1. 'ampl': routine `amplot` plots the time variation of the amplitude of selected coefficients of any available transforms of the mass distribution (lgsp, danl, sphb, zanl, *etc.*). The user is asked to select the data type, if the `.res` file does not contain any data of the selected type, you will be returned to the same prompt, entering "end" will get you out of this loop. If data exists, you will be asked whether you wish the amplitude to be shown on a logarithmic or linear scale, which sectoral harmonic ($m$), and then some other inputs to select the radial range, pitch angle, *etc.*, depending on the type of data. The prompts give you an option to go back to the previous question, enabling you to exit the routine.

2. 'cntd': routine `mkdens` creates color images of the non-axisymmetric part of the density at selected times (see Fig. 3 of Sellwood & Carlberg 2014, for an example). This is resynthesized from the sectoral harmonic analysis, and options exist to apply a more restrictive filter and to show either the absolute or relative overdensity.

3. 'frqs': routine `frqplt` plots the radial variation of the principal frequencies, $\Omega$, $\kappa$, and $\nu$ measured in the mid-plane, at selected times

4. 'intg': routine `intplt` shows the time evolution of the total energy, virial ratio, total linear or angular momentum, or the number of escapers from the grid. When multiple populations are present, there is also an option to plot the evolution of the separate $L_z$ of each component, and the torque between them. Note this routine requires there be more than a single analysis step in the `.res` file, and it will simply return if only one (or none) data record is found. If just one record is found, it will report the initial virial ratio.

5. 'lgan': routine `loganl` displays coefficients of any available transforms of the mass distribution (lgsp, danl, sphb, zanl, *etc.*) at selected times. You select the type of data you wish to display, the sectoral harmonic ($m$), *etc.* After the data are displayed, you can ask to look at the time evolution of a particular coefficient, and these data are fitted with a growth rate and pattern speed. However, the separate program `modefit` is much better for estimating mode eigenfrequencies from simulated data.

6. 'lgav': routine `lgspav` plots time averages, with means, medians and dispersions, of any available transforms of the mass distribution (lgsp, danl, sphb, zanl, *etc.*). For the example, the magnitude of the leading/trailing bias of spirals can be visualized by averaging over many spiral episodes.

7. 'moit': routine `moiplt` plots the time evolution of the MoI data of each separate population and each energy range. The plots show the two axis ratios and the "triaxiality parameter".

8. 'moni': routine `monplt` multiple options to plot the heating rates *etc.* from the monitored particles

9. 'mrpl': routine `mrplot` plots the time evolution of the radii of various mass fractions of the selected active mass component. It works best with rhor data and shows either linearly spaced or logarithmically spaced mass fractions. When logarithmic spacing is selected, it starts from the smallest possible mass, increasing the fraction by factors of 2 at each new line.

10. 'pnts': routine `pntplt` produces dot plots of the positions (or velocities) of a fraction of the particles at selected times – this is the only option for which "square frames" should be selected when opening the plotting device. While color density plots are now vogue, the simple particle plots displayed here give you a very quick vizualization

of what happened as the model evolved. There are also options to plot the particle velocities, in cylindircal polar components, or in the meridional plane $(R, z)$.

11. 'prop': routine `prpplt` offers many options to display the radial variations of means or dispersions of the velocities at selected times. It also does the same for the surface density (although option 'rhor' is better) and the radial variation of Toomre's $Q$.

12. 'rhor': routine `rhoplt` plots the radial varitaion of the density or the mass of a selected component at intervals determined by the user.

13. 'rngs': routine `rngplt` displays the rings of particles at selected times

14. 'satl': routine `satanl` plots either

    (a) the motion of a rigid perturber, whether that may be a satellite, a bar or even a spiral, or

    (b) it can display the motion of the grid center(s) when it is (they are) allowed to move.

15. 'spct': routine `spctrm` produces power spectra of a selected sectoral harmonic of the density or some other transform over a selected time interval. You first select the type of data, then the sectoral harmonic. You then have options to select part of the data, say the radial range, and then part of the time evolution. The number of analysis moments should not have any large prime factors, which would cause the FFT to abort. You can display the selected data, if desired, and can go back and reselect if the selected part is not to your liking. Once ready, you are asked to select a fraction of the frequency range, it is geneally best to enter 1 here at the first time, which shows all possible frequencies but entering 4 say, which shows a quarter of the frequency range. The frequencies are generally assumed to be positive, *i.e.* not retrograde, but enterging 0 (zero) here will display some negative frequencies – of course, as for any Fourier transfor over a finite time interval, the frequency spectrum repeats outside the fundamental range. If the horizontal axis is radius in these plots, you will be asked whether to show the frequencies measured from the model or the theoretical frequencies of the original mass distibution; there are things to be said for either option - try them and you will see.

16. 'zprf': routine `zprplt` displays the vertical density profile of the disk at selected radii and times

Six other options are also available but are probably less useful.

More sophisticated analysis options, such as mode fitting, are described in §13.3.1.

# Part III
# Detailed description

## 8. COMPUTING THE GRAVITATIONAL FIELD

### 8.1. General principles

In an $N$-body experiment, a stellar system is modeled by point particles with masses $\mu_i$ at positions $\boldsymbol{x}_i(t)$, so the density at time $t$ is a sum of Dirac-$\delta$ functions:

$$\rho(\boldsymbol{x},t) = \sum_{i=1}^{N} \mu_i\, \delta(\boldsymbol{x} - \boldsymbol{x}_i(t)) \; . \tag{2}$$

A density distribution of this form is numerically inconvenient, and all collisionless methods attribute a finite density and size to each particle by one means or another that will soon become clear.

#### 8.1.1. *Gravitational potential*

The potential at an arbitrary field point $\Phi(\boldsymbol{x})$ is related to the density through Poisson's equation

$$\nabla^2\Phi(\boldsymbol{x}) = 4\pi G\rho(\boldsymbol{x}). \tag{3}$$

Numerical solutions on grids of this elliptic, second order PDE were first developed for plasma physics (*e.g.* Hockney & Eastwood 1981). The solution is straightforward if the electrostatic potential on the perimeter of the mesh, which could be grounded say, can be specified as a boundary condition.

For an isolated mass distribution, however, the gravitational potential on the boundary of a grid cannot be specified; all that is known is that $\Phi$ decays as $r^{-1}$ at large distances. James (1977) devised a scheme, described in §8.7, to overcome this difficulty, but all other methods evaluate the integral over a volume $V$ containing the mass:

$$\Phi(\boldsymbol{x}) \equiv \int_V \rho(\boldsymbol{x}')P\left(|\boldsymbol{x}' - \boldsymbol{x}|\right) d^3\boldsymbol{x}' = \sum_{i=1}^{N} \mu_i\, P\left(\boldsymbol{x} - \boldsymbol{x}_i\right), \tag{4}$$

with the second form applying to a system of particles. The potential at the point $\boldsymbol{x}$ is therefore the convolution of the spherically symmetric kernel $P(\xi)$ with the mass distribution $\rho(\boldsymbol{x}')$. Here $\xi = |\boldsymbol{x}' - \boldsymbol{x}|$, the geometric distance between the source and the field point. The choice of the Newtonian potential of a unit mass $P(\xi) = -G/\xi$ would yield the exact potential of point particles.

#### 8.1.2. *Finite size particles*

An immediate advantage of this approach is that one can **soften** the force law at short range by changing the kernel, $P(\xi)$, to have some slightly different form that asymptotes to $-G/\xi$ at large distances, but avoids the numerical difficulties created by the singularity as $\xi \to 0$. Further discussion and possible choices for the kernel are given in §9.

It has long been recognized that softening can be regarded as *smoothing the density* distribution (see Barnes 2012, for a clear explanation). We can apply the Poisson operator to the potential $\Phi_s$ of a system of softened point particles in order to determine the effective density distribution:

$$\rho_{\rm eff}(\boldsymbol{x}) = (4\pi G)^{-1}\nabla^2\Phi_s(\boldsymbol{x}) = (4\pi G)^{-1}\left[\sum_{i=1}^{N} \mu_i\, \nabla^2 P(\xi_i)\right] = \sum_{i=1}^{N} \mu_i\, \rho_{\rm kern}(\xi_i), \tag{5}$$

where

$$\rho_{\rm kern}(\xi) \equiv (4\pi G)^{-1}\nabla^2 P(\xi) = \frac{1}{4\pi G\xi^2}\frac{d}{d\xi}\left(\xi^2\frac{dP(\xi)}{d\xi}\right), \tag{6}$$

is the effective density of each particle. Thus the softened potential of the system of $N$ point particles is the Newtonian field of a smoothed density that is convolution of the kernel $\rho_{\rm kern}$ with the particles (eq. 2).

When using grid methods to solve for the field, the mass of a particle is spread over a few grid points, and the acclerations are interpolated between grid points, as described in §10. These procedures automatically attribute an effective size to the particle that is related to the grid spacing, leading to the alternative name of "finite size particle" methods (*e.g.* Hockney & Eastwood 1981). I usually apply additional explicit softening, making the particles larger than the inner grid cells on non-uniform polar grids.

### 8.1.3. *Accelerations (or force per unit mass)*

The acceleration to be applied to a particle is $\boldsymbol{a} = -\nabla\Phi$ and we therefore need to estimate the gradient of the potential. If we know the scalar gravitational potential $\Phi(i,j,k)$ at grid points $\{i,j,k\}$ only, we must approximate each component of the acceleration by differencing the grid values. The simplest central difference expression is:

$$a_x(i,j,k) \approx -\frac{\Phi(i+1,j,k) - \Phi(i-1,j,k)}{2h_x}, \tag{7}$$

which yields an approximation to the $x$-acceleration component to be applied to the particles. Here I have assumed uniform spacing, $h_x$, of the grid points. (An improved difference operator that is used for the 3D Cartesian grid, is given in eq. 57.) The disadvantages of this approach are:

1. the estimate of the gradient is good to 2nd order in $h_x$ only,
2. it is smoothed over a distance of $2h_x$,
3. the formula cannot be applied right at the grid edges, and
4. it is much harder to estimate the gradient when the mesh spacing is non-uniform, as occurs on most polar grids.

Of course, one could evaluate higher order differences to yield a better approximation to the gradient, but the computational cost rises rapidly and does not lead to improved resolution, while non-central differences introduce other numerical problems. I therefore find it advantageous to solve directly for the vector acceleration components[11]

$$\boldsymbol{a}(\boldsymbol{x}) = \int_V \rho(\boldsymbol{x}')\boldsymbol{S}(\boldsymbol{x}' - \boldsymbol{x})\, d^3\boldsymbol{x}', \tag{8}$$

where

$$\boldsymbol{S}(\boldsymbol{x}' - \boldsymbol{x}) = -\nabla_{\boldsymbol{x}} P(|\boldsymbol{x}' - \boldsymbol{x}|) = -\frac{dP}{d\xi}\frac{\boldsymbol{x}' - \boldsymbol{x}}{|\boldsymbol{x}' - \boldsymbol{x}|}. \tag{9}$$

Although this Coulomb integral form necessitates a separate convolution for each coordinate component, the extra work required is more than compensated by four benefits:

1. acclerations at grid points are determined exactly, instead of as an estimate through potential differences,
2. avoiding central differences (eq. 7) improves the effective spatial resolution of the grid by a factor of two,
3. the dependence of the forces acting on a particle on its sub-grid position, aka **grid noise** (§10.1.1), is substantially reduced, and
4. it avoids the difficulty of estimating gradients on non-uniformly spaced grids.

I solve for the acceleration components at a general time step, and evaluate the potential also at an analysis step only.

## 8.2. **Grids and FFTs**

As already noted, techniques inherited from plasma physics suggest that it is most efficient to invoke a grid, raster, or mesh of points at which masses are assigned and the field is calculated. Since particles move continuously, grid methods require schemes to interpolate between grid points, as discussed in §10.

The most efficient techniques employ fast Fourier transforms (FFTs). But a discrete Fourier transform over a finite region implies that the transformed region is replicated indefinitely in all coordinate directions. Thus for Cartesian grids, the field near the edge of the grid would be almost equally affected by the presence of adjacent image mass distributions implicitly created by the Fourier representation, as illustrated in 1D in Fig. 3. But the influence of the periodic images can be eliminated if one surrounds the active region containing the mass with empty regions of equal width. Fig. 3 makes it clear that grid doubling entirely eliminates the influence of image meshes, and does not merely reduce their influence by moving the images farther away. Unfortunately, grid doubling necessitates Fourier transforms on a grid of twice the size in 1D, four times the size in 2D, and eight times the size in 3D. Despite the overhead associated with grid doubling, FFT methods are substantially faster than direct convolution (eq. 4), and yield the exact same result, to machine precision. I supply test programs (§14) to verify that this is true, as a strong check of this part of the software.

[11] Methods based on the integral (8) should perhaps more properly be called Coulomb solvers than Poisson solvers!

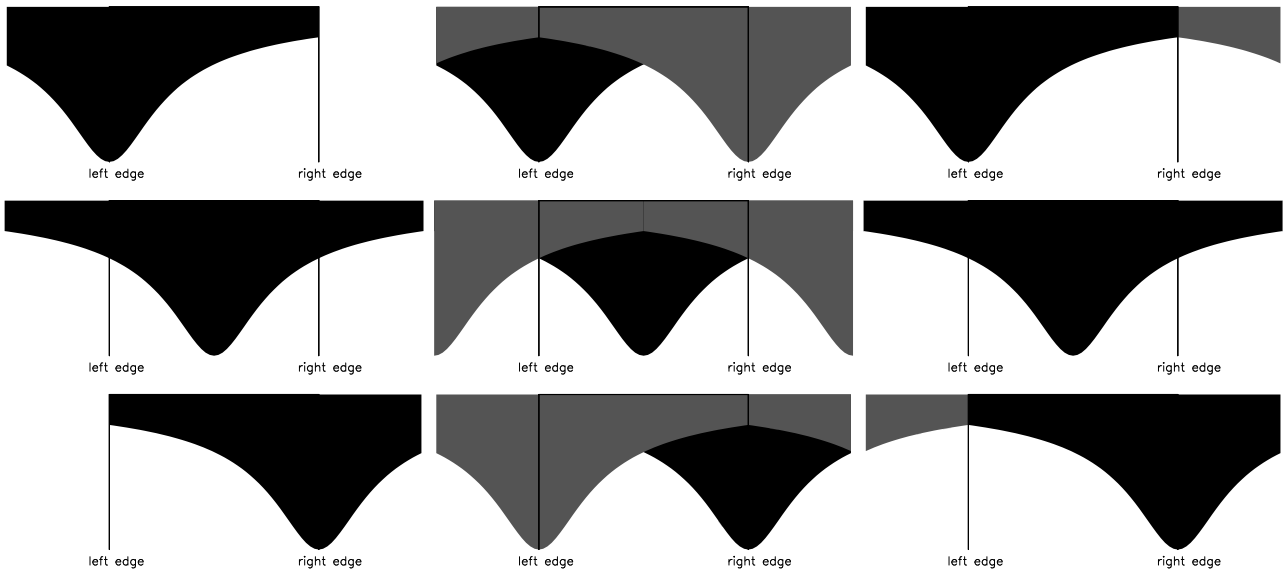


FIG. 3.— 1D illustration of the need to double the grid. The black function is the softened potential of a unit mass. The left panel shows the convolution of three masses, one on each grid boundary, marked by the vertical lines, and one in the grid center; the total potential is sum of all three. The middle panel illustrates in grey the influence of the periodic images, implicitly created by the discrete Fourier transformation of the mass distribution. The right panel shows that the influence of the periodic images is removed from the active grid region (between the left an right edges) when the period is doubled and the additional region contains no mass.

### 8.3. **2D Cartesian grid**

The fixed resolution of a uniform Cartesian mesh has the disadvantage that a dense center may not be well-resolved. This type of grid is therefore more suitable for rather uniform density disks, such as the Kalnajs (1972) disks, which were studied by Hohl (1972) in one of the earlier descriptions of this method. I am not aware of any attempt to implement adaptive resolution for a 2D mesh.

#### 8.3.1. *The solution for the potential*

We wish to solve for the gravitational field over a 2-D mesh of points that arises from masses lying at the same mesh points. In a Cartesian mesh, the grid points must be equally spaced in each coordinate direction. In principle, the $x$- and $y$-spacings could differ but it is strongly desirable that $h_x = h_y = h$, in order to minimize force anisotropies (§10.1.1). (The choice of `lscale`$= \ell/h$ is the key spatial resolution parameter §2.4). The number of grid points in each direction is $N_x$ and $N_y$, and $N_x = N_y$ for most applications.

The potential at the point $(i, j)$ arising from masses $w(u, v)$ is

$$\Phi(i,j) = \sum_{u=0}^{N_x-1} \sum_{v=0}^{N_y-1} w(u,v) P(i-u, j-v), \tag{10}$$

where the kernel $P$, aka Green's function, which is written here as a function of two variables is, in fact, a function only of the geometric distance between the source and field points $\xi = [h_x^2(i-u)^2 + h_y^2(j-v)^2]^{1/2}$ as defined in §9. This summation is a double convolution in $x$ and $y$ and can therefore be written as a multiplication in Fourier space. In order to eliminate the effects of the periodic images we must double the grid in each dimension. So in fact, we must evaluate a set of summations of the form

$$\Phi(i,j) = \sum_{u=0}^{2N_x-1} \sum_{v=0}^{2N_y-1} w(u,v) P(i-u, j-v), \tag{11}$$

which yields the same result, since $w(u, v) = 0$ for $N_x \le u \le 2N_x - 1$ and $N_y \le v \le 2N_y - 1$. Since the Green function, $P(a, b)$, is both real and even with a half-period of $N_x$ in $x$ and $N_y$ in $y$, it may be expanded as

$$P(a,b) = \sqrt{\frac{2}{N_x}}{\sum_{\alpha=0}^{N_x}}' \sqrt{\frac{2}{N_y}}{\sum_{\beta=0}^{N_y}}' P_{\alpha\beta} \cos\left(\frac{\pi a \alpha}{N_x}\right) \cos\left(\frac{\pi b \beta}{N_y}\right), \tag{12}$$

where the primes on the summation symbol indicate that the end values are halved. The Fourier coefficients $P_{\alpha\beta}$ are

given by

$$P_{\alpha\beta} = \sqrt{\frac{2}{N_x}} \sum_{k=0}^{N_x}{}' \sqrt{\frac{2}{N_y}} \sum_{n=0}^{N_y}{}' P(k,n) \cos\left(\frac{\pi k\alpha}{N_x}\right) \cos\left(\frac{\pi n\beta}{N_y}\right), \tag{13}$$

where the prime again indicates that the end values are halved.

Substituting the definition (12) into (11), we obtain

$$\begin{aligned}\Phi(i,j) &= \sqrt{\tfrac{4}{N_xN_y}} \textstyle\sum_{u=0}^{2N_x-1} \sum_{v=0}^{2N_y-1} w(u,v) \sum_{\alpha=0}^{N_x}{}' \sum_{\beta=0}^{N_y}{}' P_{\alpha\beta} \cos\left(\frac{\pi(i-u)\alpha}{N_x}\right) \cos\left(\frac{\pi(j-v)\beta}{N_y}\right) \\ &= \sqrt{\tfrac{4}{N_xN_y}} \textstyle\sum_{u=0}^{2N_x-1} \sum_{v=0}^{2N_y-1} w(u,v) \sum_{\alpha=0}^{N_x}{}' \sum_{\beta=0}^{N_y}{}' P_{\alpha\beta} \\ &\quad \left[\cos\left(\frac{\pi i\alpha}{N_x}\right) \cos\left(\frac{\pi u\alpha}{N_x}\right) + \sin\left(\frac{\pi i\alpha}{N_x}\right) \sin\left(\frac{\pi u\alpha}{N_x}\right)\right] \\ &\quad \left[\cos\left(\frac{\pi j\beta}{N_y}\right) \cos\left(\frac{\pi v\beta}{N_y}\right) + \sin\left(\frac{\pi j\beta}{N_y}\right) \sin\left(\frac{\pi v\beta}{N_y}\right)\right].\end{aligned} \tag{14}$$

Re-ordering the summations, we find

$$\begin{aligned}\Phi(i,j) &= \sqrt{\tfrac{4}{N_xN_y}} \textstyle\sum_{\alpha=0}^{N_x}{}' \sum_{\beta=0}^{N_y}{}' P_{\alpha\beta} \\ &\quad \left\{\cos\left(\frac{\pi i\alpha}{N_x}\right) \cos\left(\frac{\pi j\beta}{N_y}\right) \sum_{u=0}^{2N_x-1} \sum_{v=0}^{2N_y-1} w(u,v) \cos\left(\frac{2\pi u\alpha}{2N_x}\right) \cos\left(\frac{2\pi v\beta}{2N_y}\right)\right. \\ &\quad + \cos\left(\frac{\pi i\alpha}{N_x}\right) \sin\left(\frac{\pi j\beta}{N_y}\right) \sum_{u=0}^{2N_x-1} \sum_{v=0}^{2N_y-1} w(u,v) \cos\left(\frac{2\pi u\alpha}{2N_x}\right) \sin\left(\frac{2\pi v\beta}{2N_y}\right) \\ &\quad + \sin\left(\frac{\pi i\alpha}{N_x}\right) \cos\left(\frac{\pi j\beta}{N_y}\right) \sum_{u=0}^{2N_x-1} \sum_{v=0}^{2N_y-1} w(u,v) \sin\left(\frac{2\pi u\alpha}{2N_x}\right) \cos\left(\frac{2\pi v\beta}{2N_y}\right) \\ &\quad \left. + \sin\left(\frac{\pi i\alpha}{N_x}\right) \sin\left(\frac{\pi j\beta}{N_y}\right) \sum_{u=0}^{2N_x-1} \sum_{v=0}^{2N_y-1} w(u,v) \sin\left(\frac{2\pi u\alpha}{2N_x}\right) \sin\left(\frac{2\pi v\beta}{2N_y}\right)\right\}.\end{aligned} \tag{15}$$

Defining

$$\sum_{u=0}^{2N_x-1} \sum_{v=0}^{2N_y-1} w(u,v) \cos\left(\frac{2\pi u\alpha}{2N_x}\right) \cos\left(\frac{2\pi v\beta}{2N_y}\right) \equiv 2\sqrt{N_xN_y} w_{cc,\alpha\beta}, \tag{16}$$

*etc.*, (15) may be written as

$$\begin{aligned}\Phi(i,j) &= \tfrac{1}{2\sqrt{N_xN_y}} \textstyle\sum_{\alpha=0}^{N_x}{}' \sum_{\beta=0}^{N_y}{}' \\ &\quad \left\{\Phi_{cc,\alpha\beta} \cos\left(\frac{2\pi i\alpha}{2N_x}\right) \cos\left(\frac{2\pi j\beta}{2N_y}\right) + \Phi_{cs,\alpha\beta} \cos\left(\frac{2\pi i\alpha}{2N_x}\right) \sin\left(\frac{2\pi j\beta}{2N_y}\right)\right. \\ &\quad \left. + \Phi_{sc,\alpha\beta} \sin\left(\frac{2\pi i\alpha}{2N_x}\right) \cos\left(\frac{2\pi j\beta}{2N_y}\right) + \Phi_{ss,\alpha\beta} \sin\left(\frac{2\pi i\alpha}{2N_x}\right) \sin\left(\frac{2\pi j\beta}{2N_y}\right)\right\},\end{aligned} \tag{17}$$

where

$$\Phi_{cc,\alpha\beta} = 2\sqrt{N_xN_y} w_{cc,\alpha\beta} P_{\alpha\beta}, \tag{18}$$

*etc.* Thus the procedure is to form the double Fourier transform (16) in subroutine `c2manl`, multiply by the appropriate coefficient as (18) in subroutine `c2conv`, and resynthesize the potential using (17) in subroutine `c2fsyn`.

The values of $P_{\alpha\beta}$ are precalculated and tabulated by subroutine `c2grnm`.

### 8.3.2. *The solution for the acceleration components*

Unlike most other authors, I compute the acceleration components directly by convolution also.

The $x$ and $y$ acceleration components have Green functions that are anti-symmetric in one component and the analysis needs to be reworked. For the $x$-accelerations the Green function must be expanded as

$$S_x(a,b) = \sqrt{\frac{2}{N_x}} \sum_{\alpha=1}^{N_x-1} \sqrt{\frac{2}{N_y}} \sum_{\beta=0}^{N_y}{}' S_{x,\alpha\beta} \sin\left(\frac{\pi a\alpha}{N_x}\right) \cos\left(\frac{\pi b\beta}{N_y}\right). \tag{19}$$

The Fourier coefficients $S_{x,\alpha\beta}$ are given by

$$S_{x,\alpha\beta} = \sqrt{\frac{2}{N_x}} \sum_{j=1}^{N_x-1} \sqrt{\frac{2}{N_y}} \sum_{k=0}^{N_y}{}' S_x(j,k) \sin\left(\frac{\pi j\alpha}{N_x}\right) \cos\left(\frac{\pi k\beta}{N_y}\right). \tag{20}$$

Using the definition (19), the $x$-acceleration may be expanded as

$$\begin{aligned} a_x(i,j) &= \sqrt{\tfrac{4}{N_xN_y}} \sum_{u=0}^{N_x-1} \sum_{v=0}^{N_y-1} w(u,v) \sum_{\alpha=1}^{N_x-1} \sum_{\beta=0}^{N_y}{}' \, S_{x,\alpha\beta} \sin\left(\tfrac{\pi(i-u)\alpha}{N_x}\right) \cos\left(\tfrac{\pi(j-v)\beta}{N_y}\right) \\ &= \sqrt{\tfrac{4}{N_xN_y}} \sum_{u=0}^{N_x-1} \sum_{v=0}^{N_y-1} w(u,v) \sum_{\alpha=1}^{N_x-1} \sum_{\beta=0}^{N_y}{}' \, S_{x,\alpha\beta} \\ &\quad \left[\sin\left(\tfrac{\pi i\alpha}{N_x}\right) \cos\left(\tfrac{\pi u\alpha}{N_x}\right) - \cos\left(\tfrac{\pi i\alpha}{N_x}\right) \sin\left(\tfrac{\pi u\alpha}{N_x}\right)\right] \\ &\quad \left[\cos\left(\tfrac{\pi j\beta}{N_y}\right) \cos\left(\tfrac{\pi v\beta}{N_y}\right) + \sin\left(\tfrac{\pi j\beta}{N_y}\right) \sin\left(\tfrac{\pi v\beta}{N_y}\right)\right]. \end{aligned} \tag{21}$$

Proceeding as before, and defining $S_{x,0\beta} = S_{x,N_x\beta} = 0$, we get

$$\begin{aligned} a_x(i,j) &= \sqrt{\tfrac{4}{N_xN_y}} \sum_{\alpha=0}^{N_x}{}' \sum_{\beta=0}^{N_y}{}' \, S_{x,\alpha\beta} \\ &\quad \Big\{\sin\left(\tfrac{2\pi i\alpha}{2N_x}\right) \cos\left(\tfrac{2\pi j\beta}{2N_y}\right) \sum_{u=0}^{2N_x-1} \sum_{v=0}^{2N_y-1} w(u,v) \cos\left(\tfrac{2\pi u\alpha}{2N_x}\right) \cos\left(\tfrac{2\pi v\beta}{2N_y}\right) \\ &\quad + \sin\left(\tfrac{2\pi i\alpha}{2N_x}\right) \sin\left(\tfrac{2\pi j\beta}{2N_y}\right) \sum_{u=0}^{2N_x-1} \sum_{v=0}^{2N_y-1} w(u,v) \cos\left(\tfrac{2\pi u\alpha}{2N_x}\right) \sin\left(\tfrac{2\pi v\beta}{2N_y}\right) \\ &\quad - \cos\left(\tfrac{2\pi i\alpha}{2N_x}\right) \cos\left(\tfrac{2\pi j\beta}{2N_y}\right) \sum_{u=0}^{2N_x-1} \sum_{v=0}^{2N_y-1} w(u,v) \sin\left(\tfrac{2\pi u\alpha}{2N_x}\right) \cos\left(\tfrac{2\pi v\beta}{2N_y}\right) \\ &\quad - \cos\left(\tfrac{2\pi i\alpha}{2N_x}\right) \sin\left(\tfrac{2\pi j\beta}{2N_y}\right) \sum_{u=0}^{2N_x-1} \sum_{v=0}^{2N_y-1} w(u,v) \sin\left(\tfrac{2\pi u\alpha}{2N_x}\right) \sin\left(\tfrac{2\pi v\beta}{2N_y}\right)\Big\}. \end{aligned} \tag{22}$$

For the $x$-accelerations, we therefore have

$$\begin{aligned} a_x(i,j) &= \tfrac{2}{\sqrt{N_xN_y}} \sum_{\alpha=0}^{N_x}{}' \sum_{\beta=0}^{N_y}{}' \\ &\quad \Big\{a_{cc,\alpha\beta} \cos\left(\tfrac{2\pi i\alpha}{2N_x}\right) \cos\left(\tfrac{2\pi j\beta}{2N_y}\right) + a_{cs,\alpha\beta} \cos\left(\tfrac{2\pi i\alpha}{2N_x}\right) \sin\left(\tfrac{2\pi j\beta}{2N_y}\right) \\ &\quad + a_{sc,\alpha\beta} \sin\left(\tfrac{2\pi i\alpha}{2N_x}\right) \cos\left(\tfrac{2\pi j\beta}{2N_y}\right) + a_{ss,\alpha\beta} \sin\left(\tfrac{2\pi i\alpha}{2N_x}\right) \sin\left(\tfrac{2\pi j\beta}{2N_y}\right)\Big\}, \end{aligned} \tag{23}$$

but with

$$\begin{aligned} a_{cc,\alpha\beta} &= -\sqrt{2N_xN_y} w_{sc,\alpha\beta} S_{x,\alpha\beta} \\ a_{cs,\alpha\beta} &= -\sqrt{2N_xN_y} w_{ss,\alpha\beta} S_{x,\alpha\beta} \\ a_{sc,\alpha\beta} &= \sqrt{2N_xN_y} w_{cc,\alpha\beta} S_{x,\alpha\beta} \\ a_{ss,\alpha\beta} &= \sqrt{2N_xN_y} w_{cs,\alpha\beta} S_{x,\alpha\beta} \end{aligned} \tag{24}$$

Thus the procedure is almost identical to the solution for the potential. The solution for the $y$-accelerations is similar. The values of $S_{x,\alpha\beta}$ and $S_{y,\alpha\beta}$ are also precalculated and tabulated by subroutine `c2grnm`.

### 8.4. **2D polar grid**

A 2D polar grid was first employed by Miller (1976). A preliminary description of the version used here was first given in Sellwood (1981), but it has been refined since. Compared to a Cartesian grid, a polar grid has two obvious advantages:

1. the geometry is more naturally suited to the simulation of nearly round galaxies and
2. the spatial resolution of the grid is highest in the dense center of the galaxy being modeled.

But it also has some significant disadvantages:

3. the forces between particles have a grid-dependent part (see §10.2) and
4. it will be harder to compute the distance between particles for SPH.

The grid-dependent part of the force on a particle is negligible only when the gravity softening length, $\epsilon$, is much greater than even the largest cell sizes, but this is true only in the very center of most practical meshes.

The grid has $N_a$ spokes with even angular spacing $\alpha = 2\pi/N_a$ radians. The radial spacing of grid rings can be arbitrary, but it is clearly advantageous to space them more closely near the center. The rule adopted by Miller (1976) was to set the radius of the $u$-th ring (integer $u$) to be $R_u = h_R e^{\alpha u}$, which results in the radial spacing $R_{u+1} - R_u \equiv R_u(e^\alpha - 1)$ that,

for small $\alpha$, is very nearly equal to the azimuthal extent $\alpha R_u$, giving the grid cells a desirable aspect ratio everwhere.[12] However, $R_0$ would have radius $h_R$, and $R_{-\infty}$ would be needed to prevent the grid from having a central hole. The default rule I adopt therefore is

$$R_u = h_R\,(e^{\alpha u} - 1) \qquad \text{with} \qquad 0 \le u < N_R - 1, \tag{25}$$

so that $R_0 = 0$ and a grid with a finite number, $N_R$, of rings has no hole. Note that the code is structured such that a change to the rule (25) would require revision of only a single function (`rofu`). The length unit $h_R$ is defined as the "grid unit", which plays the role of $h_x$ on a Cartesian mesh. While formally the grid has $N_a$ separate points at $R = 0$, no numerical difficulties arise because the Green function is well-behaved at zero separation.

[The code has options to change the radial grid spacing in two ways: (i) to override the default $R_0 = 0$ for the innermost radius so that the grid does have an inner hole. This is achieved by entering an integer "offset" value $u_0$. If $u_0 \le 0$ the inner hole radius is $R_{\rm in} = h_R e^{-\alpha u_0}$ and the outer boundary is at $R_{\rm out} = h_R e^{\alpha(N_R - u_0)}$. If $u_0 > 0$, then grid has no hole, but has $u_0$ rings linearly spaced in the range $0 \le R \le h_R$. (ii) It is also possible to change the exponential rule (eq. 25) for the grid ring radii to a power-law

$$R_u = h_R(u + u_0)^\beta \qquad \text{with} \qquad 0 \le u < N_R - 1, \tag{26}$$

where $\beta = (1 - \gamma)^{-1}$, with $\gamma$ being the input parameter. Why did I do it that way? Neither of these options has been used by the author in recent years.]

Of course, the polar character of the grid leads to grid cells whose azimuthal and radial extents increase with distance from the center. This property has not given rise to any numerical dificulties, in my experience, even when the largest cell sizes are a few times larger than the softening length (see §9). However, it behoves the user to check that important results are unchanged when the grid cell sizes are halved in each dimension, say, by doubling the number of grid spokes and rings.

### 8.4.1. *The solution for the potential*

The potential at the point $(i, j)$ arising from masses $w(u, v)$ is

$$\Phi(i,j) = \sum_{u=0}^{N_R-1} \sum_{v=0}^{N_a-1} w(u,v) P(i,u,j-v), \tag{27}$$

where the Green's function $P$ depends only on the distance between the source and field points, as described in §9. The radial part of the convolution cannot be solved by Fourier transformation when the radial spacing is arbitrary and/or the kernel is not scale invariant. However, the azimuthal summation is a simple convolution in angle, which can be written as a multiplication in Fourier space. Since azimuthal variations are intrinsically periodic, it does not require surrounding blank areas and the Green function, $P(i, u, a)$, is both real and even in azimuth with a half-period of $N_a/2$. It may therefore be expanded as

$$P(i,u,a) = \frac{2}{\sqrt{N_a}} {\sum_{m=0}^{N_a/2}}' P_m(i,u) \cos\left(\frac{2\pi a m}{N_a}\right), \tag{28}$$

where the prime on the summation sign indicates the end values are halved. The Fourier coefficients $P_m(i, u)$ are given by

$$P_m(i,u) = \frac{2}{\sqrt{N_a}} {\sum_{j=0}^{N_a/2}}' P(i,u,j) \cos\left(\frac{2\pi j m}{N_a}\right), \tag{29}$$

where again the end values are halved.

Substituting the definition (28) into (27), we get

$$\begin{aligned} \Phi(i,j) &= \tfrac{2}{\sqrt{N_a}} \textstyle\sum_{u=0}^{N_R-1} \sum_{v=0}^{N_a-1} w(u,v) {\sum_{m=0}^{N_a/2}}' P_m(i,u) \cos\left(\tfrac{2\pi(j-u)m}{N_a}\right) \\ &= \tfrac{2}{\sqrt{N_a}} \textstyle\sum_{u=0}^{N_R-1} \sum_{v=0}^{N_a-1} w(u,v) {\sum_{m=0}^{N_a/2}}' P_m(i,u) \left[\cos\left(\tfrac{2\pi jm}{N_a}\right)\cos\left(\tfrac{2\pi vm}{N_a}\right) + \sin\left(\tfrac{2\pi jm}{N_a}\right)\sin\left(\tfrac{2\pi vm}{N_a}\right)\right]. \end{aligned} \tag{30}$$

Re-ordering the summations, we obtain

$$\begin{aligned} \Phi(i,j) = \frac{2}{\sqrt{N_a}} {\sum_{u=0}^{N_R-1}} {\sum_{m=0}^{N_a/2}}' P_m(i,v) \quad &\left\{\cos\left(\tfrac{2\pi jm}{N_a}\right) \textstyle\sum_{m=0}^{N_a-1} w(u,v) \cos\left(\tfrac{2\pi vm}{N_a}\right)\right. \\ &\left. + \sin\left(\tfrac{2\pi jm}{N_a}\right) \textstyle\sum_{v=0}^{N_a-1} w(u,v) \sin\left(\tfrac{2\pi vm}{N_a}\right)\right\}. \end{aligned} \tag{31}$$

[12] This choice may seem additionally tempting, since Fourier transformation of the potential integral (eq. 4) in logarithmic radial coordinates reduces to a simple multiplication for a scale-free kernel, such as the Newtonian potential. Unfortunately, the introduction of a softening length scale into the kernel destroys this simplicity.

Defining

$$w_{c,m}(u) = N_a^{-1/2} \sum_{v=0}^{N_a-1} w(u,v) \cos\left(\tfrac{2\pi vm}{N_a}\right)$$
$$w_{s,m}(u) = N_a^{-1/2} \sum_{v=0}^{N_a-1} w(u,v) \sin\left(\tfrac{2\pi vm}{N_a}\right), \tag{32}$$

*etc.*, eq. (31) may be written as

$$\Phi(i,j) = \frac{2}{\sqrt{N_a}} {\sum_{m=0}^{N_a/2}}' \left\{ \Phi_{c,m}(i) \cos\left(\frac{2\pi jm}{N_a}\right) + \Phi_{s,m}(i) \sin\left(\frac{2\pi jm}{N_a}\right) \right\}, \tag{33}$$

where

$$\Phi_{c,m}(i) = \sqrt{N_a} \sum_{u=0}^{N_R-1} w_{c,m}(u) P_m(i,u)$$
$$\Phi_{s,m}(i) = \sqrt{N_a} \sum_{u=0}^{N_R-1} w_{s,m}(u) P_m(i,u), \tag{34}$$

Thus the solution for the potential proceeds by azimuthal Fourier analysis of the masses $w(u,v)$ assigned to each ring $u$ using (32) in subroutine `p2manl`, followed by the radial convolution of each separate sectoral harmonic $m$ using (34) in subroutine `p2conv`, and is completed by Fourier synthesis of the potential coefficients on each ring $i$ using (33) in subroutine `p2fsyn`.

The radial convolution (34) part of the solution is the slowest, since it requires $2N_R^2(N_a/2+1)$ operations, but since the different sectoral harmonics $m$ are independent, it is readily parallelized. It also requires a large table of the azimuthally transformed Green function $P_m(i,v)$, which are precalculated in subroutine `p2grnm`.

### 8.4.2. *Filtering sectoral harmonics*

The above procedure yields the exact solution of eq. (27) at each grid point, and includes all sectoral hamonics $0 \le m \le N_a/2$. In most applications, azimuthal variations in the density for large $m$ are generally small and arise almost entirely from shot noise of the particles. They are therefore of no physical importance. Replacing the summations (34) with zeros for sectoral harmonics $m_{\max} < m \le N_a/2$ is both easy and time-saving, and results in a smoother potential from which no meaningful information has been lost. Furthermore, the table of Green function values can be reduced in size, since most terms will never be required. See also §8.8.

### 8.4.3. *The solution for the accelerations*

As for the potential, we wish to solve for the acceleration components

$$a_R(i,j) = \sum_{u=0}^{N_R-1} \sum_{v=0}^{N_a-1} w(u,v) S_R(i,u,j-v)$$
$$a_\phi(i,j) = \sum_{u=0}^{N_R-1} \sum_{v=0}^{N_a-1} w(u,v) S_\phi(i,u,j-v). \tag{35}$$

The radial acceleration Green function is also symmetric in azimuth, and the above analysis goes through with the substitution of $P(i,u,a) \to S_R(i,u,a)$. But the Green function for the azimuthal acceleration is anti-symmetric and requires the different expansion

$$S_\phi(i,u,a) = \frac{2}{\sqrt{N_a}} \sum_{m=1}^{N_a/2-1} S_{\phi,m}(i,u) \sin\left(\frac{2\pi am}{N_a}\right). \tag{36}$$

The Fourier coefficients $S_{\phi,m}$ are given by

$$S_{\phi,m}(i,u) = \frac{2}{\sqrt{N_a}} \sum_{j=1}^{N_a/2-1} S_\phi(j) \sin\left(\frac{2\pi jm}{N_a}\right). \tag{37}$$

The algebra that leads to the solution of eq. (35) follows along the same lines as above, and is straightforward.

The radial convolution part of the solution again requires large tables of values of both $S_{R,m}(i,u)$ and $S_{\phi,m}(i,u)$, which are also precalculated in subroutine `p2grnm`.

### 8.4.4. *Conversion to Cartesian components*

The formulae (35) yield the acceleration components in polar coordinates, but it is highly desirable to integrate the motion of particles in Cartesian coordinates. Integration in polar coordinates has three disadvantages:

1. it loses not only the simplicity of Cartesian leap frog (§11.1),
2. particle motion near $R = 0$ requires vanishingly short time steps since the azimuthal angle $\phi$ changes rapidly for even a slowly moving particle, and
3. interpolation of the radial accelerations between grid spokes gives rise to a self-force of the particle towards the grid center, essentially because the forces on the different spokes are not parallel. While this spurious acceleration can be calculated and subtracted for each particle, it adds a tiresome overhead.

These difficulties disappear when Cartesian leap frog is used. I therefore transform $a_R(i,j)$ and $a_\phi(i,j)$ to $a_x(i,j)$ and $a_y(i,j)$, which is readily accomplished by subroutine `polcat` at each grid point before any particles are advanced.

## 8.5. **3D polar grid**

The solution for the potential on a cylindrical polar grid was described by Pfenniger & Friedli (1991) and extended to solving directly for the accelerations by Sellwood & Valluri (1997). We wish to solve for the gravitational field over a 3D mesh of points which arises from masses lying at the same mesh points. Let the grid have $N_R$ arbitrarily spaced rings, $N_a$ equally spaced azimuthal divisions and $N_z$ planes spaced equally by the distance $h_z$. The number of planes is constrained to be odd, so that $z = 0$ the mid-plane is at the middle of the particle disk, but FFTs require that $N_z$ does not have any large prime number factors. In the current implementation, the number of planes can contain no factors other than 3 or 5, and therefore the only values (less than 1000) that can be used are:

| | | | | | |
|---|---|---|---|---|---|
| 3 | 5 | 9 | 15 | 25 | 27 |
| 45 | 75 | 81 | 125 | 135 | 225 |
| 243 | 375 | 405 | 625 | 675 | 729 |

### 8.5.1. *The solution for the potential*

The potential at the point $(i,j,k)$ arising from masses $w(t,u,v)$ is

$$\Phi(i,j,k) = \sum_{t=0}^{N_R-1}\sum_{u=0}^{N_a-1}\sum_{v=0}^{N_z-1} w(t,u,v)P(i,t,j-u,k-v), \tag{38}$$

where the kernel or Green's function, $P$, is defined elsewhere. The radial convolution is not amenable to Fourier methods for the reasons given in §8.4, but the remaining summation is a double convolution in $\phi$ and $z$ and can therefore be written as a multiplication in Fourier space. The azimuthal field is periodic, but in order to eliminate the effects of the periodic images in the $z$-direction, we must double the grid.

Since $w(t,u,v) = 0$ for $N_z \le v \le 2N_z - 1$, we can double the vertical summation without changing the result:

$$\Phi(i,j,k) = \sum_{t=0}^{N_R-1}\sum_{u=0}^{N_a-1}\sum_{v=0}^{2N_z-1} w(t,u,v)P(i,t,j-u,k-v). \tag{39}$$

Now the kernel, $P(i,t,a,b)$, is both real and even with a half-period of $N_a/2$ in $\phi$ and $N_z$ in $z$; it may therefore be expanded as

$$P(i,t,a,b) = \frac{2}{\sqrt{N_a}}{\sum_{m=0}^{N_a/2}}' \sqrt{\frac{2}{N_z}}{\sum_{n=0}^{N_z}}' P_{mn}(i,t)\cos\left(\frac{2\pi am}{N_a}\right)\cos\left(\frac{\pi bn}{N_z}\right), \tag{40}$$

where the prime indicates the end values are halved. The Fourier coefficients $P_{mn}(i,t)$ are given by

$$P_{mn}(i,t) = \frac{2}{\sqrt{N_a}}{\sum_{r=0}^{N_a/2}}' \sqrt{\frac{2}{N_z}}{\sum_{s=0}^{N_z}}' P(i,t,r,s)\cos\left(\frac{2\pi rm}{N_a}\right)\cos\left(\frac{\pi sn}{N_z}\right), \tag{41}$$

where the prime again indicates that the end values are halved.

Substituting the definition (40) into (39), we get

$$\begin{aligned}\Phi(i,j,k) &= \sqrt{\tfrac{8}{N_aN_z}}\sum_{t=0}^{N_R-1}\sum_{u=0}^{N_a-1}\sum_{v=0}^{N_z-1} w(t,u,v)\,{\sum_{m=0}^{N_a/2}}'\,{\sum_{n=0}^{N_z}}'\,P_{mn}(i,t)\cos\left(\tfrac{2\pi(j-u)m}{N_a}\right)\cos\left(\tfrac{\pi(k-v)n}{N_z}\right)\\
&= \sqrt{\tfrac{8}{N_aN_z}}\sum_{u=0}^{N_a-1}\sum_{v=0}^{N_z-1} w(t,u,v)\,{\sum_{m=0}^{N_a/2}}'\,{\sum_{n=0}^{N_z}}'\,P_{mn}(i,t)\\
&\qquad\left[\cos\left(\tfrac{2\pi jm}{N_a}\right)\cos\left(\tfrac{2\pi um}{N_a}\right)+\sin\left(\tfrac{2\pi jm}{N_a}\right)\sin\left(\tfrac{2\pi um}{N_a}\right)\right]\\
&\qquad\left[\cos\left(\tfrac{\pi kn}{N_z}\right)\cos\left(\tfrac{\pi vn}{N_z}\right)+\sin\left(\tfrac{\pi kn}{N_z}\right)\sin\left(\tfrac{\pi vn}{N_z}\right)\right].\end{aligned} \tag{42}$$

Re-ordering the summations, and doubling the vertical sequence, we get

$$\begin{aligned}\Phi(i,j,k) &= \sqrt{\tfrac{8}{N_a N_z}} \textstyle\sum_{t=0}^{N_R-1} \sum_{m=0}^{N_a/2\,\prime} \sum_{n=0}^{N_z}{}' P_{mn}(i,t) \\ &\Big\{\cos\left(\tfrac{2\pi jm}{N_a}\right)\cos\left(\tfrac{2\pi kn}{2N_z}\right)\textstyle\sum_{u=0}^{N_a-1}\sum_{v=0}^{2N_z-1} w(t,u,v)\cos\left(\tfrac{2\pi um}{N_a}\right)\cos\left(\tfrac{2\pi vn}{2N_z}\right) \\ &+\cos\left(\tfrac{2\pi jm}{N_a}\right)\sin\left(\tfrac{2\pi kn}{2N_z}\right)\textstyle\sum_{u=0}^{N_a-1}\sum_{v=0}^{2N_z-1} w(t,u,v)\cos\left(\tfrac{2\pi um}{N_a}\right)\sin\left(\tfrac{2\pi vn}{2N_z}\right) \\ &+\sin\left(\tfrac{2\pi jm}{N_a}\right)\cos\left(\tfrac{2\pi kn}{2N_z}\right)\textstyle\sum_{u=0}^{N_a-1}\sum_{v=0}^{2N_z-1} w(t,u,v)\sin\left(\tfrac{2\pi um}{N_a}\right)\cos\left(\tfrac{2\pi vn}{2N_z}\right) \\ &+\sin\left(\tfrac{2\pi jm}{N_a}\right)\sin\left(\tfrac{2\pi kn}{2N_z}\right)\textstyle\sum_{u=0}^{N_a-1}\sum_{v=0}^{2N_z-1} w(t,u,v)\sin\left(\tfrac{2\pi um}{N_a}\right)\sin\left(\tfrac{2\pi vn}{2N_z}\right)\Big\}. \end{aligned} \tag{43}$$

Defining

$$w_{cc,mn}(t) \equiv (2N_aN_z)^{-1/2}\sum_{u=0}^{N_a-1}\sum_{v=0}^{2N_z-1} w(t,u,v)\cos\left(\frac{2\pi mu}{N_a}\right)\cos\left(\frac{2\pi nv}{2N_z}\right), \tag{44}$$

*etc.*, this may be written as

$$\begin{aligned}\Phi(i,j,k) &= (2N_aN_z)^{-1/2}\textstyle\sum_{m=0}^{N_a/2\,\prime}\sum_{n=0}^{N_z}{}' \\ &\Big\{\Phi_{cc,mn}(i)\cos\left(\tfrac{2\pi jm}{N_a}\right)\cos\left(\tfrac{2\pi kn}{2N_z}\right)+\Phi_{cs,mn}(i)\cos\left(\tfrac{2\pi jm}{N_a}\right)\sin\left(\tfrac{2\pi kn}{2N_z}\right) \\ &+\Phi_{sc,mn}(i)\sin\left(\tfrac{2\pi jm}{N_a}\right)\cos\left(\tfrac{2\pi kn}{2N_z}\right)+\Phi_{ss,mn}(i,t)\sin\left(\tfrac{2\pi jm}{N_a}\right)\sin\left(\tfrac{2\pi kn}{2N_z}\right)\Big\}, \end{aligned} \tag{45}$$

where

$$\Phi_{cc,mn}(i) = \sqrt{2N_aN_z}\,w_{cc,mn}(t)P_{mn}(i,t), \tag{46}$$

*etc.* Thus the solution for the potential proceeds by double Fourier analysis of the masses $w(t,u,v)$ assigned to each ring and plane $u,v$ using (44) in subroutine `p3manl`, followed by the radial convolution of each separate sectoral harmonic $m$ using (46) in subroutine `p3conv`, and is completed by Fourier synthesis of the potential coefficients on each ring and plane $j,k$ using (45) in subroutine `p3fsyn`.

The radial convolution (46) part of the solution is the slowest, since it requires $4N_R^2N_z(N_a/2+1)$ operations, but since the different planes $n$ are independent, it is readily parallelized. It also requires a large table of the azimuthally and vertically transformed Green function $P_{mn}(i,t)$, which are precalculated in subroutine `p3grnm`.

### 8.5.2. *Filtering sectoral harmonics*

The above procedure yields the exact solution of eq. (39) at each grid point, and includes all sectoral hamonics $0 \le m \le N_a/2$. As for the 2D polar grid, azimuthal variations in the density for large $m$ are generally small and arise almost entirely from shot noise of the particles. They are therefore of no physical importance. Replacing the summations (46) with zeros for sectoral harmonics $m_{\max} < m \le N_a/2$, as for the 2D polar grid (§8.4.2) is both easy and time-saving, and results in a smoother potential from which no meaningful information has been lost. Furthermore, the table of transformed Green function values, $P_{mn}(i,t)$ can be reduced in size, since most terms will never be required.

Note that one cannot extend this simplification to the vertical transform, since all terms $n$ are needed to represent the vertical variation of the potential.

### 8.5.3. *The solution for the accelerations*

The radial acceleration Green function is also symmetrical both vertically and azimuthally and the above analysis goes through without change. But the azimuthal and vertical accelerations have different symmetry properties for the Green function and the analysis needs to be reworked separately for each. For azimuthal accelerations, the Green function is anti-symmteric in angle and thefore must be expanded as

$$S_\phi(i,t,a,b) = \frac{2}{\sqrt{N_a}}\sum_{m=1}^{N_a/2-1}\sqrt{\frac{2}{N_z}}\sum_{n=0}^{N_z}{}' S_{\phi,mn}(i,t)\sin\left(\frac{2\pi am}{N_a}\right)\cos\left(\frac{\pi bn}{N_z}\right). \tag{47}$$

The Fourier coefficients $S_{\phi,mn}$ are given by

$$S_{\phi,mn}(i,t) = \frac{2}{\sqrt{N_a}}\sum_{r=1}^{N_a/2-1}\sqrt{\frac{2}{N_z}}\sum_{s=0}^{N_z}{}' S_\phi(r,s)\sin\left(\frac{2\pi rm}{N_a}\right)\cos\left(\frac{\pi sn}{N_z}\right). \tag{48}$$

Using the definition (47), the azimuthal acceleration may be expanded as

$$\begin{aligned} a_\phi(i,j,k) &= \sqrt{\tfrac{8}{N_a N_z}} \sum_{t=0}^{N_R-1} \sum_{u=0}^{N_a-1} \sum_{v=0}^{N_z-1} w(t,u,v) \sum_{m=1}^{N_a/2-1} {\sum_{n=0}^{N_z}}' S_{\phi,mn}(i,t) \sin\left(\tfrac{2\pi(j-u)m}{N_a}\right) \cos\left(\tfrac{\pi(k-v)n}{N_z}\right) \\ &= \sqrt{\tfrac{8}{N_a N_z}} \sum_{t=0}^{N_R-1} \sum_{m=0}^{N_a-1} \sum_{n=0}^{N_z-1} w(t,u,v) \sum_{m=1}^{N_a/2-1} {\sum_{n=0}^{N_z}}' S_{\phi,mn}(i,t) \\ &\qquad \left[\sin\left(\tfrac{2\pi jm}{N_a}\right)\cos\left(\tfrac{2\pi um}{N_a}\right) - \cos\left(\tfrac{2\pi jm}{N_a}\right)\sin\left(\tfrac{2\pi um}{N_a}\right)\right] \\ &\qquad \left[\cos\left(\tfrac{\pi kn}{N_z}\right)\cos\left(\tfrac{\pi un}{N_z}\right) + \sin\left(\tfrac{\pi kn}{N_z}\right)\sin\left(\tfrac{\pi un}{N_z}\right)\right]. \end{aligned} \tag{49}$$

Proceeding as before, and defining $S_{\phi,0n}(i,t) = S_{\phi,N_a/2n}(i,t) = 0$, we get

$$\begin{aligned} a_\phi(i,j,k) &= \sqrt{\tfrac{8}{N_a N_z}} \sum_{t=0}^{N_R-1} {\sum_{m=0}^{N_a/2}}' {\sum_{n=0}^{N_z}}' S_{\phi,mn}(i,t) \\ &\Big\{\sin\left(\tfrac{2\pi jm}{N_a}\right)\cos\left(\tfrac{2\pi kn}{2N_z}\right) \sum_{m=0}^{N_a-1} \sum_{n=0}^{2N_z-1} w(t,u,v)\cos\left(\tfrac{2\pi mu}{N_a}\right)\cos\left(\tfrac{2\pi nv}{2N_z}\right) \\ &+ \sin\left(\tfrac{2\pi jm}{N_a}\right)\sin\left(\tfrac{2\pi kn}{2N_z}\right) \sum_{m=0}^{N_a-1} \sum_{n=0}^{2N_z-1} w(t,u,v)\cos\left(\tfrac{2\pi mu}{N_a}\right)\sin\left(\tfrac{2\pi nv}{2N_z}\right) \\ &- \cos\left(\tfrac{2\pi jm}{N_a}\right)\cos\left(\tfrac{2\pi kn}{2N_z}\right) \sum_{m=0}^{N_a-1} \sum_{n=0}^{2N_z-1} w(t,u,v)\sin\left(\tfrac{2\pi mu}{N_a}\right)\cos\left(\tfrac{2\pi nv}{2N_z}\right) \\ &- \cos\left(\tfrac{2\pi jm}{N_a}\right)\sin\left(\tfrac{2\pi kn}{2N_z}\right) \sum_{m=0}^{N_a-1} \sum_{n=0}^{2N_z-1} w(t,u,v)\sin\left(\tfrac{2\pi mu}{N_a}\right)\sin\left(\tfrac{2\pi nv}{2N_z}\right)\Big\}. \end{aligned} \tag{50}$$

For the azimuthal accelerations, we therefore have

$$\begin{aligned} a_\phi(i,j,k) &= \tfrac{1}{\sqrt{2N_a N_z}} \sum_{t=0}^{N_R-1} {\sum_{m=0}^{N_a/2}}' {\sum_{n=0}^{N_z}}' \\ &\Big\{a_{cc,mn}(i)\cos\left(\tfrac{2\pi jm}{N_a}\right)\cos\left(\tfrac{2\pi kn}{2N_z}\right) + a_{cs,mn}(i)\cos\left(\tfrac{2\pi jm}{N_a}\right)\sin\left(\tfrac{2\pi kn}{2N_z}\right) \\ &+ a_{sc,mn}(i)\sin\left(\tfrac{2\pi jm}{N_a}\right)\cos\left(\tfrac{2\pi kn}{2N_z}\right) + a_{ss,mn}(i)\sin\left(\tfrac{2\pi jm}{N_a}\right)\sin\left(\tfrac{2\pi kn}{2N_z}\right)\Big\}, \end{aligned} \tag{51}$$

but with

$$\begin{aligned} a_{cc,mn}(i) &= -\sqrt{2N_a N_z}\, w_{sc,mn}(t) S_{\phi,mn}(i,t) \\ a_{cs,mn}(i) &= -\sqrt{2N_a N_z}\, w_{ss,mn}(t) S_{\phi,mn}(i,t) \\ a_{sc,mn}(i) &= \sqrt{2N_a N_z}\, w_{cc,mn}(t) S_{\phi,mn}(i,t) \\ a_{ss,mn}(i) &= \sqrt{2N_a N_z}\, w_{cs,mn}(t) S_{\phi,mn}(i,t) \end{aligned} \tag{52}$$

For vertical accelerations, the Green function is anti-symmteric in $z$ while symmetric in angle. It thefore must be expanded as

$$S_z(i,t,a,b) = \frac{2}{\sqrt{N_a}} {\sum_{m=0}^{N_a/2}}' \sqrt{\frac{2}{N_z}} \sum_{n=1}^{N_z-1} S_{z,mn}(i,t)\cos\left(\frac{2\pi am}{N_a}\right)\sin\left(\frac{\pi bn}{N_z}\right). \tag{53}$$

The Fourier coefficients $S_{z,mn}$ are given by

$$S_{z,mn}(i,t) = \frac{2}{\sqrt{N_a}} {\sum_{j=0}^{N_a/2}}' \sqrt{\frac{2}{N_z}} \sum_{k=1}^{N_z-1} S_z(i,t,j,k)\cos\left(\frac{2\pi jm}{N_a}\right)\sin\left(\frac{\pi kn}{N_z}\right). \tag{54}$$

The algebra to obtain the solution for $a_v(i,j,k)$ is similar but tedious.

The radial convolution part of the solutions for the accelerations again requires large tables of values of $S_{R,m}(i,k)$, $S_{\phi,m}(i,k)$, and $S_{v,m}(i,k)$, which are also precalculated in subroutine `p3grnm`.

As for the 2D polar grid, we immediately convert the radial $a_R(i,j,k)$ and azimuthal $a_\phi(i,j,k)$ acceleration components to Cartesian $a_x(i,j,k)$, $a_y(i,j,k)$.

### 8.6. **Green's function files**

Most of the above grid methods require a large table containing values of the transformed Green function. This table is created, and written to a `.grt` file, by a call to `greenm` early in the execution of `galaxy`. Since the contents of this file can readily be recreated, it may be regarded as scratch file that need not be preserved. However, the preparation of this table can be rather time-consuming for large grids, especially for the cylindrical polar grid, and it is recommended that the `.grt` file be preserved while it may still be useful. Subroutine `greenm` attempts to open the file and, if the file is found, it checks the contents of the header. If the header indicates the file is the appropriate one, with all the correct parameters, then the routine does not attempt to re-create it.

If a `.grt` file having the appropriate name exists, but the header information disagrees with that expected from the grid parameters in the `.dat` file, then every executable will terminate with an error message reporting the disagreement. In this case, the user should should simply delete the `.grt` file and rerun the executable, which will create the correct file.

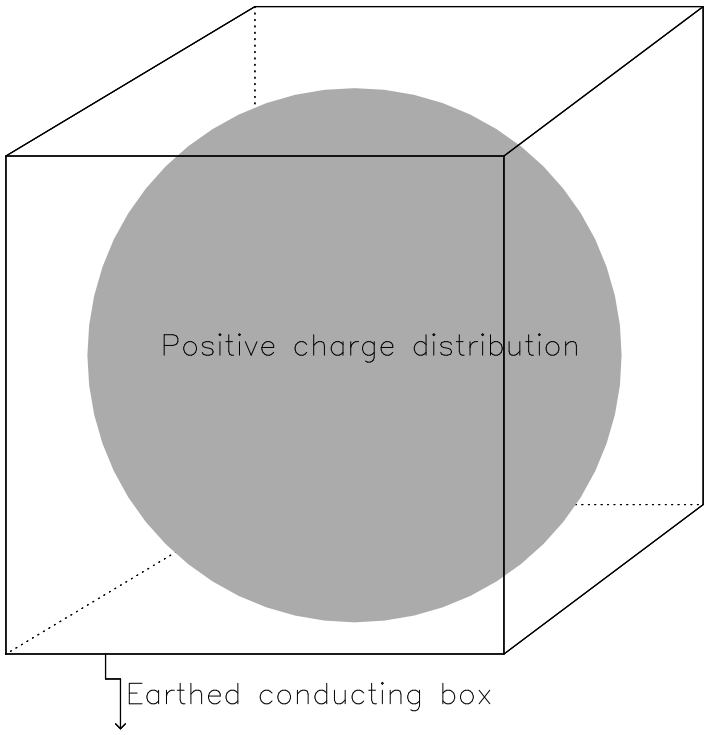


FIG. 4.— Schematic illustration of the solution for Poisson's equation described by James (1977).

### 8.7. **3D Cartesian grid**

The technique described by James (1977) differs fundamentally from that used in the three previously described methods since, instead of the potential integral (4), it reverts to a numerical solution of Poisson's equation (3).

The method works by analogy with electrostatics, as illustrated in Fig. 4. The potential on the boundary of the computation volume cannot be predicted when the mass (charge) distribution is isolated. However, it is easy to solve Poisson's equation for the interior potential $\Phi_g$, if the boundaries of the box are imagined to be grounded, such that $\Phi_g = 0$ on all boundary points and also everywhere outside the box. The interior potential $\Phi_g \neq 0$, because of the positive charge contained by the box and therefore a negative charge distribution must be induced on the walls of the box. The magnitude of the charge, $m$, at each point on the box wall can be determined using Gauss's law

$$4\pi G m = \int_S \nabla\Phi_g \cdot d\boldsymbol{A}, \tag{55}$$

where the integral is over the surface of a small cube containing that one grid point, and $d\boldsymbol{A}$ is the normal vector to the surface of this little cube. Only one surface contributes to the integral because $\Phi = 0$ everywhere except at interior points.

If the induced boundary charges were neutralized by adding a charge distribution of the opposite sign, the potential everywhere would be the desired solution for the potential $\Phi$ of the original isolated interior charge distribution. James therefore proceeds by determining the potential $\Phi_b$ on the boundary of the box by summing the potential contributions from all the neutralizing boundary charges, and then performs a second solution for the interior field by solving Laplace's equation $\nabla^2\Phi_b = 0$ for an empty box with a now-known boundary potential. The potential of the isolated charge distribution is then $\Phi = \Phi_g + \Phi_b$ at every point.

James (1977) describes at length how the solution for the boundary charges, and their potential $\Phi_b$ on the boundary, can all be computed in Fourier space, so that only two FFTs over the entire volume are needed.

The software written by James is accessed through a call to routine `c3fndf`. It requires that the number of mesh points in each dimension be $2^j + 1$, with $j$ being an integer. Non-cubic grid cells are allowed, but not recommended. As for other grid methods, it requires an auxilliary data file that is created in `c3grnm`, but which in this case is deleted when the program ends.

This method does not enable direct solutions for the acceleration components, and the potential has to be differenced using a finite-difference approximation to the gradient operator. Rather than the simple difference operator given in (7), the code uses the following finite difference operator, which was preferred by May & James (1984):

$$a_x = -\frac{\partial\Phi(i,j,k)}{\partial x} \approx -\frac{\Delta_x}{12h}, \tag{56}$$

$$\text{where} \quad \Delta_x = \Phi(i+1,j+1,k) - \Phi(i-1,j+1,k) \; + \; \Phi(i+1,j-1,k) - \Phi(i+1,j-1,k) \; + \\ \Phi(i+1,j,k+1) - \Phi(i-1,j,k+1) \; + \; \Phi(i+1,j,k-1) - \Phi(i-1,j,k-1) \; + \\ 2[\Phi(i+1,j,k) - \Phi(i-1,j,k)]$$

with analagous expressions for $a_y$ and $a_z$.

The inter-particle force law that results from a linear interpolation scheme (CIC, see§10) on this grid was displayed in the appendix of Sellwood & Merritt (1994) and recalculated for the right-hand panel of Figure 8 below.

### 8.8. **Dirty tricks**

Polar grids, in both 2- and 3D (§§8.4 & 8.5), as well as planar SFP (§8.11.3) methods, allow the user to impose various rotational symmetries on the derived forces. For example, it is desirable to filter out the $m = 1$ sectoral harmonic in order to prevent unbalanced forces from arising between the particles and the attraction of a fixed mass component, that may represent a central mass, bulge, and/or halo. Further, one could, if desired, filter out all odd sectoral harmonics or even restrict the forces from the particles to those from a single $m$. All these options can be implemented as described in §§8.4.2 & 8.5.2.

If the axisymmetric ($m = 0$) terms are zeroed out during the solution, then it is, of course, necessary to apply a fixed central attraction in addition to the non-axisymmetric forces from the particles.

It is also possible to impose reflection symmetry, about the $z = 0$ plane say, which will inhibit buckling instabilities, and/or other rotation or reflection symmetries, simply by performing the appropriate average of the density distribution assigned to the grid points before solving for the field. Naturally, the geometry of Cartesian grids allows only 2- or 4-fold rotation symmetries to be imposed (Sellwood 2020, showed how to eliminate $m = 4$ while preserving $m = 2$).

Similar games can be played on the spherical grid (§8.10) and also with the spherical SFP (§8.11.5) method, simply by excluding terms of specified $l$.

### 8.9. **Polar axisymmetric grid**

This method was described and employed by Sellwood (1996). In essence, the particles that move in 3D, each represent a uniform ring of mass having a small radial extent. Because forces from such mass elements can be computed without divergent values, even with no softening, this method yields the true Newtonian field of a finitely thick axisymmetric disk.

#### 8.9.1. *Potential theory*

The basic axisymmetric mass element is a uniform hoop. The gravitational potential at an arbitrary field point, $(R, z)$, of a thin wire ring of radius $R'$ and mass $\delta M$ lying in the $z = 0$ plane centered on $R = 0$ is

$$\delta\Phi(R,z) = -\frac{G\delta M}{\pi\sqrt{RR'}}\alpha\, K(\alpha), \tag{57}$$

where $K$ is a complete elliptic integral of the first kind and

$$\alpha^2 = \frac{4RR'}{(R+R')^2 + z^2}. \tag{58}$$

The radial acceleration is

$$\delta a_R(R,z) = \frac{G\delta M}{\pi\sqrt{RR'}}\left[\frac{E(\alpha)}{1-\alpha^2}\frac{\partial\alpha}{\partial R} - \frac{\alpha\, K(\alpha)}{2R}\right] \tag{59}$$

where $E$ is a complete elliptic integral of the second kind and

$$\frac{\partial\alpha}{\partial R} = \frac{\alpha^3}{8R}\left[\frac{R'}{R} - \frac{R}{R'} + \frac{z^2}{RR'}\right]. \tag{60}$$

The vertical acceleration is

$$\delta a_z(R,z) = \frac{G\delta M}{\pi\sqrt{RR'}}\;\frac{E(\alpha)}{1-\alpha^2}\;\frac{\partial\alpha}{\partial z}, \tag{61}$$

with

$$\frac{\partial\alpha}{\partial z} = -\frac{\alpha^3 z}{4RR'}. \tag{62}$$

These expressions are well defined everywhere except for the radial force and potential on the ring of mass itself, where $\alpha = 1$.

Since the central attraction of a massive ring on itself cannot be neglected, I give the rings a finite radial extent, imagining them to have uniform surface density (*i.e.*, $\delta M = 2\pi\Sigma R\delta R$) and to extend mid-way to the next grid point on either side, while having zero vertical thickness. The gravitational field of such a Saturn ring-like element of mass $M$ is

$$\mathcal{F}(R,z) = \frac{2M}{(R_2^2 - R_1^2)} \int_{R_1}^{R_2} \frac{\delta\mathcal{F}}{\delta M} R' dR', \tag{63}$$

with $\mathcal{F} = \Phi$, $a_R$ or $a_z$ with the integrands eqs. (57), (59), (61) respectively. These expressions are finite everywhere, but the Cauchy principal value of the integral is required for the potential and radial attraction when $z = 0$ and $R_1 < R < R_2$.

In order to ensure that the vertical forces between any pair of mass elements on the grid are equal and opposite, I evaluate the integral for each pair of grid points just once and use the negative of the first result in place of the other. (The difference between the two values arises from not evaluating the mass-weighted average of the force over the extended mass at the field point; the considerable extra expense of making this tiny adjustment seems unjustified.)

#### 8.9.2. *The solution for the field*

As for other methods, the solution for the field of the particles proceeeds by convolution of their mass distribution, assigned to grid points, with the expressions eq. (63). The vertical part can be solved by FFT methods, but the radial part must be done directly. The algebra can readily be adapted from the solution for the 3D polar grid (§8.5).

### 8.10. **Spherical grid**

This method is ideally suited to computing the attraction of near spherical distributions of particles. It was first described in Appendix A of Sellwood (2003), but an improved radial interpolation scheme (§8.10.4) was implemented in version 15 (Dec 2016).

#### 8.10.1. *Potential theory*

The potential at a field point $\boldsymbol{r}$ of a point mass $\mu$ at $\boldsymbol{r}'$ can be expressed as a multipole expansion (formula 3.70 of Jackson 1962):

$$\Phi(\boldsymbol{r}) = -G\mu \sum_{l=0}^{\infty} \frac{4\pi}{2l+1} \sum_{m=-l}^{l} Y_{lm}(\theta,\phi) Y_{lm}^*(\theta',\phi') \frac{r_<^l}{r_>^{l+1}}, \tag{64}$$

where $r_>$ ($r_<$) is the greater (lesser) of $r$ and $r'$. The surface harmonics $Y_{lm}$ are defined as

$$Y_{lm}(\theta,\phi) = \sqrt{\frac{2l+1}{4\pi}\frac{(l-|m|)!}{(l+|m|)!}} P_l^{|m|}(\cos\theta) e^{im\phi} \times \begin{cases} (-1)^m & \text{if } m \ge 0 \\ 1 & \text{if } m < 0 \end{cases}. \tag{65}$$

From this definition, we have

$$Y_{lm}^*(\theta,\phi) = (-1)^m Y_{l,-m}(\theta,\phi) \tag{66}$$

Thus

$$\Phi(\boldsymbol{r}) = -G\mu \sum_{l=0}^{\infty} \sum_{m=0}^{l} (2-\delta_{m0}) \frac{(l-m)!}{(l+m)!} P_l^m(\cos\theta) e^{im\phi} P_l^m(\cos\theta') e^{-im\phi'} \frac{r_<^l}{r_>^{l+1}} \tag{67}$$

and now $\Phi$ is just the real part.

The potential due to a number of point masses is therefore

$$\Phi(\boldsymbol{r}) = -G \sum_{lm} (2-\delta_{m0}) \frac{(l-m)!}{(l+m)!} P_l^m(\cos\theta) e^{im\phi} \left[ \tfrac{1}{r^{l+1}} \textstyle\sum_{\rm int} \mu_j P_l^m(\cos\theta_j') e^{-im\phi_j'} r_j'^l + r^l \sum_{\rm ext} \mu_j P_l^m(\cos\theta_j') e^{-im\phi_j'} \tfrac{1}{r_j'^{l+1}} \right], \tag{68}$$

where $(r_j', \theta_j', \phi_j')$ is the position of the $j$-th particle. The acceleration components can be derived from this by taking the negative gradient wrt the unprimed coordinates, *i.e.* holding the summations in the square brackets constant.

Expansion to $l = \infty$ is impractical, of course, and all the summations in eq. (68) are terminated at some adopted $l_{\rm max}$, which is a parameter of the code. Note that the number of terms in the expansion increases as $(l_{\rm max}+2)(l_{\rm max}+1)/2$.

### 8.10.2. *Limitations of the method*

The monopole term ($l = 0$) of eq. 64 approximates the source mass by a spherical shell of uniform surface density and radius $r'$. Adding the $l = 1$ (dipole) terms makes the surface density on the shell anti-symmetric with the highest density at the position of the source particle, and minimum density 180° away. As higher $l$ terms are included, the density on the shell becomes more localized to the position of the source, but the angular variations of the surface density never perfectly cancel for finite $l$. Consequently, as $l_{\max}$ is increased, the surface density on the shell varies more rapidly with angle, with some parts having negative density. This behavior is responsible for the two following, well-known features of the method:

- The forces acting between the pair of particles change discontinuously as they cross in radius – the problem known as "shell crossings". Discontinuous changes in acceleration, when combined with a finite time-step size, lead to a potentially serious violation of energy conservation. This problem becomes less severe, however, as more particles are employed, because the attraction between any pair decreases as a fraction of the total experienced by each particle, but some smoothing of the field remains desirable.
- Because of the angular variations in the effective surface density on the shell, the errors in the calculated forces between a pair of particles at similar radii can be very large. When the summation is truncated at a moderate $l_{\max}$, the effective mass of a point particle is spread over a wide angle, with the implication that the attraction, which asymptotes to that of the point mass at large or small relative distances from the shell, has significant errors at nearby radii. Increasing $l_{\max}$ reduces the radial extent of the region where the error is large, but also causes the errors over this shrinking region to worsen (just as Gibbs's phenomenon in Fourier series) and to vary more rapidly with angle between the source and field points. This behavior is illustrated in Figure 5.

The implication of the second point is that the expansion (68) can be very useful for models having only moderate flattening or elongation, but *it is ill-suited for systems with strong departures from spherical symmetry.* In mildly-distorted spherical systems, many particles in a given radial range are approximately uniformly distributed in angle, leading to near cancellation of the errors between each particle pair, and the net forces closely match that of a smooth mass distribution.

**Warning:** However, this happy situation degrades as the departures from sphericity become more extreme. In particular, using the method to compute the attraction from disks, or structures such as the tidal tails of an infalling satellite gives rise to large, non-cancelling force errors that compromise its use. Increasing $l_{\max}$ in order to try to "resolve" the disk or stream thickness makes matters worse, as the errors become larger!

### 8.10.3. *Adopted approach*

In order overcome the "shell crossings" issue, Villumsen (1982), White (1983), and perhaps others, introduced a softening parameter into the expressions for the potential. An alternative strategy, adopted for axial symmetry by van Albada & van Gorkom (1977) and generalized to 3-D by van Albada (1982), is to introduce a grid in all three coordinates.

Here we adopt the intermediate approach advocated by McGlynn (1984). We introduce a 1-D radial mesh on which the coefficients $A_{lm}$ and $B_{lm}$ (defined below, eq. 72) are tabulated, but retain the exact angular dependence. We therefore denote this method as PM+SH (standing for particle-mesh + surface harmonics) to indicate its somewhat composite character.

Concentric spheres at the set of radii $\{r_k\}$, $k = 0, N_r - 1$ with $r_0 = 0$, divide the computation volume into "shells." The radii of the grid spheres can be arbitrary, of course, but the rule adopted here is

$$r_k = 10^{s_d k} - 1 \qquad \text{with} \qquad 0 \le k \le N_r - 1, \qquad \text{where} \qquad s_d = \frac{\log_{10}(r_{\rm grid} + 1)}{N_r - 1}. \tag{69}$$

Here, $r_{\rm grid}$ is the outer boundary of the sphere inside of which masses can contribute to the gravitational field.

We improve on McGlynn (1984) by interpolating between grid radii in the spirit of the cloud-in-cell procedure widely used in PM methods (see §10). The CIC scheme implies that a particle has a finite size, or radial extent in this case, that is related to the grid spacing. For a particle at radius $r'_j$, we locate the nearest $r_k$ (which can be either interior or exterior to $r'_j$). We then give that particle the radial extent $\delta r'_j = \frac{1}{2}(r_{k+1} - r_{k-1})$, centered on $r'_j$. Thus fractions

$$w_{j,1} = \frac{1}{2} - \frac{r'_j - r_k}{\delta r'_j} \qquad \text{and} \qquad w_{j,2} = \frac{1}{2} + \frac{r'_j - r_k}{\delta r'_j} \tag{70}$$

of the mass of particle $j$ lie interior and exterior respectively to $r_k$. The sole exceptions to this rule occur at $r_0$, where the entire mass of the particle is deemed to be exterior to the centre, and at $r_{N_r}$, where the entire mass of the particle is deemed to be interior to the outer edge.

Following van Albada & van Gorkom (1977), we include the factors $r_k^{-(l+1)}$ and $r_k^l$ in the particle's contribution to the coefficients $A_{lm}(k)$ and $B_{lm}(k)$ (defined below) on sphere $k$, since it avoids multiplication (division) by large (small) numbers raised to high powers; clearly, the ratio $r_k/r'_j$ will always be close to unity, except for $r_0$. This exception causes no difficulties since only the monopole term contributes at the center – the angular terms have no meaning.

Thus the first step is to form the partial sum of the contributions from each particle fragment to the interior and exterior terms on each of its two neighbouring grid points, for each $(l, m)$:

$$\begin{aligned}
\alpha_{lm}(k) &= \alpha_{lm}(k) + \mu_j w_{j,1} P_l^m(\cos\theta'_j) e^{-im\phi'_j} \frac{1}{r_k} \left(\frac{r'_j}{r_k}\right)^l \\
\alpha_{lm}(k+1) &= \alpha_{lm}(k+1) + \mu_j w_{j,2} P_l^m(\cos\theta'_j) e^{-im\phi'_j} \frac{1}{r_{k+1}} \left(\frac{r'_j}{r_{k+1}}\right)^l \\
\beta_{lm}(k-1) &= \beta_{lm}(k-1) + \mu_j w_{j,1} P_l^m(\cos\theta'_j) e^{-im\phi'_j} \frac{1}{r'_j} \left(\frac{r_{k-1}}{r'_j}\right)^l , \\
\beta_{lm}(k) &= \beta_{lm}(k) + \mu_j w_{j,2} P_l^m(\cos\theta'_j) e^{-im\phi'_j} \frac{1}{r'_j} \left(\frac{r_k}{r'_j}\right)^l ,
\end{aligned} \tag{71}$$

where the summation for $\alpha_{lm}(k)$ $[\beta_{lm}(k)]$ is over particle fragments in the shell immediately interior [exterior] to $r_k$. (There are no contributions to $\alpha_{lm}(0)$ since no mass can be interior to $r = 0$). Every particle should contribute to these coefficients, which is accomplished in subroutine `s3dsum`.

The next stage is to sum over grid points so that a single coefficient on each grid point includes contributions from all the interior and exterior mass. Thus we work recursively both inwards and outwards combining the previously obtained coefficients as follows:

$$\begin{aligned}
A_{lm}(k) &= \alpha_{lm}(k) + A_{lm}(k-1) \left(\tfrac{r_{k-1}}{r_k}\right)^{l+1} \\
B_{lm}(k) &= \beta_{lm}(k) + B_{lm}(k+1) \left(\tfrac{r_k}{r_{k+1}}\right)^{l} .
\end{aligned} \tag{72}$$

This operation is performed in subroutine `s3dswp`. Note, it is the only step in the method for which the computation time depends on the number of radial grid points employed. Since there should be many fewer shells than particles, this single sweep in each direction represents a tiny overhead.

The potential on each grid sphere $r_k$ can now be evaluated as

$$\Phi(r_k, \theta, \phi) = -G \sum_{lm} (2 - \delta_{m0}) \frac{(l-m)!}{(l+m)!} P_l^m(\cos\theta) e^{im\phi} \left[A_{lm}(k) + B_{lm}(k)\right] . \tag{73}$$

### 8.10.4. *Radial interpolation*

In order to avoid discontinuous changes across grid shells, we interpolate the interior and exterior masses to a general field point, $\boldsymbol{r} = (R, \phi, z) = (x, y, z)$

$$\mathcal{A}_{lm}(r) = w_1 A_{lm}(k) + w_2 A_{lm}(k+1) \qquad \text{and} \qquad \mathcal{B}_{lm}(r) = w_1 B_{lm}(k) + w_2 B_{lm}(k+1) \tag{74}$$

and we choose the weight functions to vary linearly with $r$ in the usual way:

$$w_1 = \frac{r_{k+1} - r}{r_{k+1} - r_k} \qquad \text{and} \qquad w_2 = \frac{r - r_k}{r_{k+1} - r_k} . \tag{75}$$

The potential at a general field point is therefore

$$\Phi(r, \theta, \phi) = -G \sum_{lm} (2 - \delta_{m0}) \frac{(l-m)!}{(l+m)!} P_l^m(\cos\theta) e^{im\phi} \mathcal{Q}_{lm}(r) \tag{76}$$

where

$$\mathcal{Q}_{lm}(r) = \left(\frac{r_k}{r}\right)^{l+1} w_1 A_{lm}(k) + \left(\frac{r_{k+1}}{r}\right)^{l+1} w_2 A_{lm}(k+1) + \left(\frac{r}{r_k}\right)^{l} w_1 B_{lm}(k) + \left(\frac{r}{r_{k+1}}\right)^{l} w_2 B_{lm}(k+1). \tag{77}$$

Note that this procedure differs slightly from the strategy prior to Dec 2016, where the accelerations were determined at the grid radii and evaluated at a general point by interpolation. Now, the interior and exterior masses are interpoated to the field point, and the potential and accelerations are determined from these values. This revised strategy gives slightly more accurate accelerations, especially near the grid center.

### 8.10.5. *Acceleration components*

The acceleration vector is, as always, the negative gradient of the potential wrt the field coordinates. Differentiating term by term, we have

$$-\nabla\Phi(r,\theta,\phi) = G\sum_{lm}(2-\delta_{m0})\frac{(l-m)!}{(l+m)!}\ \left\{e^{im\phi}\mathcal{Q}_{lm}(r)\nabla P_l^m(\cos\theta)\ + \right.$$
$$\left. P_l^m(\cos\theta)\mathcal{Q}_{lm}(r)\nabla e^{im\phi} + P_l^m(\cos\theta)e^{im\phi}\nabla\left[\mathcal{Q}_{lm}(r)\right]\right\}. \tag{78}$$

Since

$$\cos\theta = \frac{z}{r},\quad \sin\theta = \frac{R}{r},\quad \cos\phi = \frac{x}{R}\quad\text{and}\quad \sin\phi = \frac{y}{R};\quad\text{with}\quad R = \sqrt{x^2+y^2} = \sqrt{r^2-z^2}, \tag{79}$$

the gradients are:

$$\begin{aligned}\nabla e^{im\phi} &= e^{im\phi}\left(-\frac{imy}{R^2},\frac{imx}{R^2},0\right)\\ \nabla P_l^m(\cos\theta) &= \frac{dP_l^m(\cos\theta)}{d\cos\theta}\left(-\frac{xz}{r^3},-\frac{yz}{r^3},\frac{x^2+y^2}{r^3}\right)\\ \nabla\left[\mathcal{Q}_{lm}(r)\right] &= \mathcal{G}_{lm}(r)\left(\frac{x}{r},\frac{y}{r},\frac{z}{r}\right),\end{aligned} \tag{80}$$

where

$$\begin{aligned}\mathcal{G}_{lm}(r)\equiv\frac{d\mathcal{Q}_{lm}}{dr} &\approx -\frac{(l+1)}{r}\left[w_1A_{lm}(k)\left(\frac{r_k}{r}\right)^{l+1} + w_2A_{lm}(k+1)\left(\frac{r_{k+1}}{r}\right)^{l+1}\right]\\ &\quad +\frac{l}{r}\left[w_1B_{lm}(k)\left(\frac{r}{r_k}\right)^{l} + w_2B_{lm}(k+1)\left(\frac{r}{r_{k+1}}\right)^{l}\right]\\ &\approx -\frac{(l+1)}{r}\left[w_1A_{lm}(k)+w_2A_{lm}(k+1)\right] + \frac{l}{r}\left[w_1B_{lm}(k)+w_2B_{lm}(k+1)\right].\end{aligned} \tag{81}$$

Note that the first line ignores the radial variation of $w_1$ and $w_2$ and the last drops small factors like $(r/r_k)^l$. I have found that including these extra terms adds considerable extra work without improving the accuracy of the result. In effect, neglect of these terms reverts to simple linear interpolation of the acceleration components between grid radii. To the same level of approximation we simply interpolate the potential through

$$\mathcal{Q}_{lm}(r) \approx w_1A_{lm}(k) + w_2A_{lm}(k+1) + w_1B_{lm}(k) + w_2B_{lm}(k+1). \tag{82}$$

Accelerations due to the mass distribution within the grid can still be computed for particles that are outside the grid. The $A_{lm}(N_r)$ terms only are needed, since the $B_{lm}$ terms are zero because no mass is exterior. However, the $(r_{N_r}/r)^{l+1}$ factors cannot be neglected when $r \gtrsim r_{N_r}$.

Thus the acceleration components at an arbitrary field point are

$$\begin{aligned}\boldsymbol{a}(\boldsymbol{r}) = -\nabla\Phi(r,\theta,\phi) = G\sum_{lm}(2-\delta_{m0})\frac{(l-m)!}{(l+m)!}\ e^{im\phi}\left\{\frac{dP_l^m(\cos\theta)}{d\cos\theta}\mathcal{Q}_{lm}(r)\left(-\frac{xz}{r^3},-\frac{yz}{r^3},\frac{x^2+y^2}{r^3}\right)\right.\\ \left. +\ P_l^m(\cos\theta)\mathcal{Q}_{lm}(r)\left(-\frac{imy}{R^2},\frac{imx}{R^2},0\right) +\ P_l^m(\cos\theta)\mathcal{G}_{lm}(k)\left(\frac{x}{r},\frac{y}{r},\frac{z}{r}\right)\right\}.\end{aligned} \tag{83}$$

These values are computed for each particle in subroutine `s3dacc`.

The grid point at the center, $r_0 = 0$, is not troublesome since the only non-zero coefficient is $B_{00}$; for this term the gradients wrt angles are zero and the derivative of $B_{lm}$ vanishes because of the $l$ factor. (Physically, there are no forces interior to a uniform hollow shell.)

### 8.10.6. *Legendre polynomials and their derivative*

Press *et al.* (1992) describe how to evaluate $P_l^m(x)$ using the recurrence relation

$$(l-m)P_l^m = x(2l-1)P_{l-1}^m - (l+m-1)P_{l-2}^m. \tag{84}$$

The coefficients can be very large for large $(l,m)$ and it is therefore expedient to work with

$$\Pi_l^m \equiv \sqrt{\frac{(l-m)!}{(l+m)!}}P_l^m, \tag{85}$$

which has a more moderate value. We need to rewrite the recurrence relation as

$$(l-m)\sqrt{\frac{(l+m)!}{(l-m)!}}\Pi_l^m = x(2l-1)\sqrt{\frac{(l-1+m)!}{(l-1-m)!}}\Pi_{l-1}^m - (l+m-1)\sqrt{\frac{(l-2+m)!}{(l-2-m)!}}\Pi_{l-2}^m. \tag{86}$$

Dividing through by $\sqrt{(l-1+m)!/(l-1-m)!}$ yields

$$\sqrt{(l+m)(l-m)}\Pi_l^m = x(2l-1)\Pi_{l-1}^m - \sqrt{\frac{(l-1-m)}{(l-1+m)}}(l+m-1)\Pi_{l-2}^m, \tag{87}$$

which is our new recurrence relation. For the case where $l = m+1$, we have

$$P_{m+1}^m = x(2m+1)P_m^m. \tag{88}$$

Thus

$$\Pi_{m+1}^m = x\sqrt{(2m+1)}\Pi_m^m. \tag{89}$$

Formula (8.5.4) from Abramowitz & Stegun (1962) gives

$$\frac{dP_l^m(\cos\theta)}{d\cos\theta} = \frac{l\cos\theta P_l^m(\cos\theta) - (l+m)P_{l-1}^m(\cos\theta)}{-\sin^2\theta}, \tag{90}$$

with $P_{l-1}^m(\cos\theta) = 0$ when $l = m$. This can be rewritten as

$$\frac{d\Pi_l^m(\cos\theta)}{d\cos\theta} = \frac{l\cos\theta\Pi_l^m(\cos\theta) - (l+m)\sqrt{(l-m)/(l+m)}\Pi_{l-1}^m(\cos\theta)}{-\sin^2\theta}. \tag{91}$$

These values are computed in subroutine `s3tplm`.

### 8.10.7. *Discussion*

The source mass distribution is smoothed because the point mass particles are blurred both radially, by the grid, and azimuthally, by the truncations at $l_{\max}$. The density distribution that gives rise to the field is therefore smooth. In our application, the amount of radial smoothing is linked to the radial grid spacing, although this could be changed.

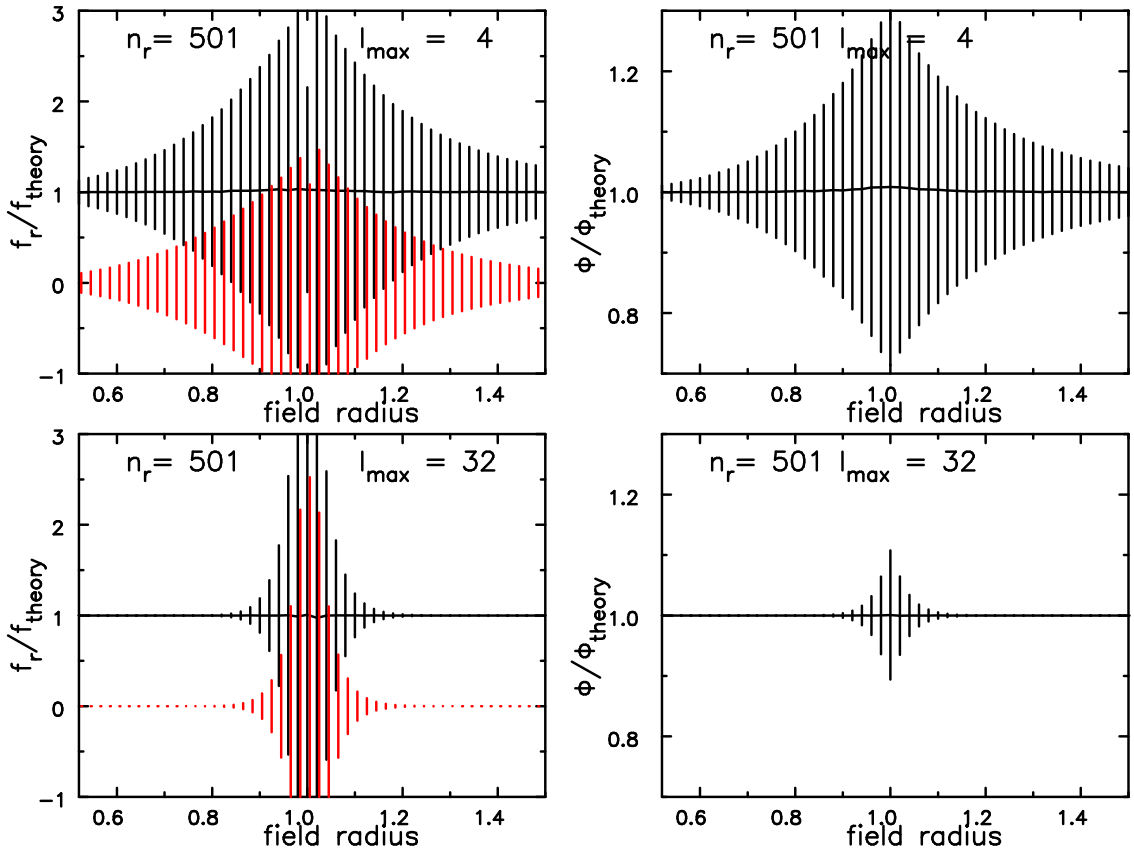


FIG. 5.— The solid black lines in left panels show the radial variation of the average attaction towards a unit point mass at radius $r' = 1$ computed by this method, divided by the theoretical attraction. The right panels show the same ratio for the potential. The top row shows the effect of truncating the expansion at $l_{\max} = 4$, while the expansion was taken to $l_{\max} = 32$ in the bottom row. The error bars show the $\pm\sigma$ variation in these ratios at a set of radii straddling that of the source mass. Each error bar is computed from the variations of the attraction or potential of 1000 randomly chosen field points on a sphere of radius $r$, each for 100 randomly chosen sources placed on the sphere at $r' = 1$. The red error bars show the $\pm\sigma$ variation in $f_\perp$, defined in eq. (92).

When the forces acting on each particle are evaluated using the above expressions, accelerations change continuously as particles cross mesh points. Particles experience forces from others in the same shell (even themselves!), but the relative contribution is small when $N$ is large, because each particle has little mass, and the shells are thin.

These considerations suggest that the number of radial mesh points could be increased as the number of particles rises. (Note that the running time is almost independent of the number of tabulation points.) The radii of the grid points is completely arbitrary. With the adopted treatment of the center, however, forces acting on a particle are tapered to zero for $0 < r < r_1$, so $r_1$ should not be too large.

Figure 5 illustrates how the force and potential errors in the field of a unit point mass at $r' = 1$ vary with the radius $r$ of the field point. For every pair of source and field points, I computed

$$f_{\rm rad} = \boldsymbol{a} \cdot \boldsymbol{\xi} \qquad \text{and} \qquad f_\perp = \left(\boldsymbol{a} \cdot \boldsymbol{a} - f_{\rm rad}^2\right)^{1/2}, \tag{92}$$

where $\boldsymbol{a}$ is the acceleration vector at the field point and $\boldsymbol{\xi}$ is the vector from the field to the source point. Thus $f_\perp$ is the magnitude of the part of the attraction that is not along that line, which ideally should be very small. The quantities plotted in black in the left panels Figure 5 are the mean and $\pm\sigma$ variations of $-f_{\rm rad}\xi^2/G\mu$, which should be unity, while the red error bars show the $\pm\sigma$ variations of $f_\perp\xi^2/G\mu$. The error bars were computed from 1 000 evaluations at randomly selected field points on the surface of a sphere at radius $r$, each for 100 positions of the source on the sphere at radius $r'$. The top row shows the errors when $l_{\rm max} = 4$, while the bottom row illustrates $l_{\rm max} = 32$.

As noted above (§8.10.2), the method computes the attraction of a mass distributed over a sphere at the same radius of the particle. The mass distribution is more localized to the position of the source particle as $l_{\rm max}$ is increased leading, in the lower panels, to more rapid convergence with increasing distance from the source to the correct point mass field, as expected. The large errors that are predicted at smaller radial separations are not seen because of smoothing by interpolation between grid shells.

However, the reader should not conclude that large $l_{\rm max}$ is needed. The method is ideally suited to computing the self-consistent evolution of nearly spherical systems of particles, for which it is rarely necessary to include $l > 4$ terms. For a collisionless system of this shape, we desire the attraction of the smoothed density that is created by a low-order expansion. The eye-popping force errors shown in Figure 5 are not concerning because the attraction of each particle contributes little and the errors largely cancel when many particles have similar radii. The left-hand panel of Figure 2 (on p6) provides the evidence to support this assertion.

## 8.11. **Field methods**

This approach was pioneered by Clutton-Brock (1972b), and further developed by Hernquist & Ostriker (1992), Saha (1993), Weinberg (1999), and others. These papers were for near-spherical systems, but the method can also applied when motion is confined to a plane (Clutton-Brock 1972a), and the code includes the technique employed by Earn & Sellwood (1995). Hernquist & Ostriker (1992) coined the acronym SCF (self-consistent field) for this method, but since all $N$-body methods are self-consistent, I prefer the acronym SFP (smooth-field-particle) method.

### 8.11.1. *Some theory*

SFP methods employ a *basis* set of density-potential pairs $\{\rho_j, \Phi_j\}$ for which each member satisfies

$$\Phi_j(\boldsymbol{x}) = -G \int \frac{\rho_j(\boldsymbol{x}')}{|\boldsymbol{x}' - \boldsymbol{x}|}\, d^3\boldsymbol{x}', \qquad j = 0, \ldots, \infty. \tag{93}$$

Many basis sets arise in potential theory (*e.g.* Jackson 1962) from separation of variables in a variety of different coordinates, and are useful if the source distribution can be conveniently expressed in those coordinates.

In order to be able to determine the field of an arbitrary model, we require the basis set to be complete. A basis set is said to be *biorthogonal* if $\langle \rho_j, \Phi_k \rangle \neq 0$ if and only if $j = k$, and *biorthonormal* if

$$\langle \rho_j, \Phi_k \rangle \equiv -\frac{R_0}{GM^2} \int \rho^* \Phi'\, d^3\boldsymbol{x} = \delta_{jk}\,, \qquad \text{for all } j \text{ and } k \tag{94}$$

where the asterisk denotes complex conjugation, and $R_0$ and $M$ are respectively length and mass scales. Any basis can be made biorthonormal with the Gram-Schmidt algorithm (*e.g.* Arfken 1985).

With a complete set of basis density functions, an arbitrary density distribution, such as that given by eq. (2) can be expanded as

$$\rho(\boldsymbol{x}, t) = \sum_{j=0}^{\infty} c_j(t)\rho_j(\boldsymbol{x})\,. \tag{95}$$

The essence of the SFP method is to choose the basis set such that the first few functions yield an adequate approximation to the density; *i.e.*

$$\rho(\boldsymbol{x},t) \simeq \sum_{j=0}^{j_{\max}} c_j(t)\rho_j(\boldsymbol{x}) \ , \tag{96}$$

with small $j_{\max}$. If the basis is biorthonormal, it follows that

$$c_j(t) = \langle \rho(\boldsymbol{x},t), \Phi_j(\boldsymbol{x}) \rangle = -\frac{R_0}{GM^2} \sum_{i=1}^{N} \mu_i \Phi_j\left(\boldsymbol{x}_i(t)\right) \ . \tag{97}$$

The contribution of each particle to these coefficients is added in subroutine `sfpsum`.

The exact Newtonian field generated by the truncated density expansion is easily computed and the acceleration of particle $i$ at time $t$ is

$$\boldsymbol{a}_i(t) = -\sum_{j=0}^{j_{\max}} c_j(t) \nabla \Phi_j(\boldsymbol{x}) \, . \tag{98}$$

The gradients of the basis potentials can be calculated analytically with no further approximation. The acceleration components for each particle are computed in subroutine `sfpacc`.

Note that, since all but the lowest order monopole functions of any reasonable basis set are oscillatory, the summation (97) will contain many cancelling contributions resulting in significant loss of precision, which worsens as the number of particles is increased. It is therefore essential to compute the functions and the sum (97) using double precision arithmetic when employing more than a few hundred particles [although the accelerations (98) can be calculated in single precision].

### 8.11.2. *Applications*

The computer time required for an SFP simulation depends linearly on the product $Nj_{\max}$. The linear $N$-dependence is greatly superior to all PP methods (direct or tree) and is rivalled only by PM methods. The computational expense of calculating the coefficients (97) clearly depends strongly on the complexity of the basis functions. But all basis sets become equally efficient if a table of function values is stored rather than evaluating the exact functions every time they are required. Furthermore, the SFP method is almost perfect for parallel machines, as was noted by Hernquist & Ostriker (1992).

The principal weakness of this elegant and highly efficient method is that the mass distribution generally evolves away from its initial state, and is no longer well approximated by a few functions. One could compensate for this by increasing $j_{\max}$, but the calculation is slowed by the rising number of terms and the precision of the summation (97) declines as the higher-order functions become more oscillatory. The method is therefore useful only when the model under study is close to equilibrium. The two principal applications are to

1. studies of relaxation in stable models and
2. studies of the linear instabilities of equilibrium models, although they cannot be followed far into the non-linear regime.

See Petersen *et al.* (2016) for a dissenting view.

### 8.11.3. *Application to 2D disks*

We must adopt a basis set that separates in polar coordinates in order to apply this method to near-axisymmetric flat disk galaxies; *i.e.*

$$\Phi_{mn}(r,\theta) = e^{im\theta}\phi_{mn}(r) \ , \qquad \Sigma_{mn}(r,\theta) = e^{im\theta}\sigma_{mn}(r) \ . \tag{99}$$

It may seem that we should choose the radial functions to have infinite extent because disk galaxies have no natural outer boundary, even if the model is initially confined to a finite region. Standard bases having this property and that could be used for flat disks include Bessel functions (BT08), logarithmic spirals (Kalnajs 1971), and Hankel-Laguerre functions (Clutton-Brock 1972a).

However, for linear stability studies of even an infinite axisymmetric equilibrium system (*e.g.* Earn & Sellwood 1995), a basis set of finite radial extent can be used since the instabilities are expected to be confined to the inner parts. Moreover, in linear perturbation theory the different azimuthal components of the density are decoupled, and simulations can be performed with a single active sectoral harmonic ($e^{im\theta}$, $m \neq 0$) with the unperturbed axisymmetric force treated as a fixed background field. For this problem we require merely that the first few members of the basis set chosen are able to represent the low-order growing modes without undue bias.

The code provides three basis sets: associated Legendre functions (Hunter 1963; Kalnajs 1972), and a two members of the class of Abel-Jacobi functions given by Kalnajs (1976a).

#### 8.11.4. *Application to thickened disks*

Earn (1996) describes several ways to generalize this approach to disks of finite thickness. The option offered in this package is to use Bessel functions, since the potential of the mass distribution on each plane parallel to the disk mid-plane decays exponentially from the plane containing the mass at the rate $e^{-|kz|}$ for each function $J_m(k)$ of this basis. Unfortunately, the range of $k$ needed to obtain an adequate representation of the density of a disk over a reasonanble radial range is quite broad, and moderate sampling in $k$ requires many functions. So while this option is available, I have made little use of it.

#### 8.11.5. *Application to spheres*

The angular dependence of all basis sets derived from separation of variables in spherical coordinates is that of the surface harmonics described above (§8.10). The radial dependence of the adopted basis should depend on the mass profile of the model. The code provided here is adapted from that devised by Hernquist & Ostriker (1992) and the relevant subroutines (`scfadd` and `scfacc`) are included in the code with their permission. It offers two function sets, one with a cored mass profile based on the Plummer sphere (Clutton-Brock 1972b) and the other based on the cusped profile of the Hernquist (1990) sphere.

### 8.12. **Direct summation**

Inserting expression (2) for the density as a set of point masses into the Coulomb integral eq. (8) yields the acceleration of particle $j$ due to all other particles:

$$\boldsymbol{a}_j = -\sum_{i\neq j} \mu_i \frac{dP}{d\xi} \frac{\boldsymbol{x}_i - \boldsymbol{x}_j}{|\boldsymbol{x}_i - \boldsymbol{x}_j|}, \tag{100}$$

where $P$ is the softened potential at distance $\xi$ from the source.

Forces between all particle pairs are computed in subroutine `drctN2`, which halves the work required since the negative of each term in the summation (100) contributes to the acceleration of particle $i$. As this is an $\mathcal{O}(N^2)$ calculation, it rapidly becomes impractical for large $N$, and the extra arrays used for this part of the calculation therefore do not need to be large. At present the code does not allow block time steps to be used with this method, neither has it been parallelized. Its only useful application has been to introduce a moderate number of heavy masses to perturb the evolution of a collisionless model (Jardel & Sellwood 2009).

### 8.13. **Tree method**

Many different tree codes exist. Most authors of these codes have invested substantial effort to improve their efficiency, and some have been made public. Here I describe briefly a tree code that I have written in Fortran, which is based on the algorithm first described by Barnes & Hut (1986). Since the method is fundamentally far less efficient than PM or SFP methods, I have made no special effort to optimize it; in particular, it does not allow block time steps to be used, neither has it been parallelized. Thus it is not intended to be competitive, and it is useful only for comparing the evolution with that from other methods, if the user wishes and the cpu time can be afforded.

The Barnes-Hut algorithm divides the simulation into an octal hierarchy of ever smaller cubic boxes until each box contains no more than a single particle. The tree structure establishes chains such that the position of the center of mass of all the particles in sub-volumes of the model can be identified. Rather than summing over every particle individually, as in the direct-$N$ method (§8.12), a tree code achieves a significant speed improvement by approximating the attraction of a distant group of particles by the single attraction of their center of mass. Nearby groups, for which the monopole attraction is too approximate, are opened up, by ascending the tree, until each sub-volume contains either a single particle, or is sufficiently far that the monopole approximation for the grouping is again adequate.

The angular criterion applied to determime whether a particular group needs to be opened is whether $\theta > \theta_{\max}$; here $\theta = L/d$ is the angle in radians subtended by the group, and is the ratio of the linear size $L$ of the box containing the group to the distance $d$ of its center-of-mass from the field point. Typically $0.3 \leq \theta_{\max} \leq 0.7$ radians.

Some tree codes evaluate higher multipoles of the field from the distant group, in order to allow a larger $\theta_{\max}$ and avoid opening more nearby groupings. If, as here, the center of mass is used, rather than the geometric center of the group, then the dipole terms are zero. I do not use any higher order multipoles.

The tree is rebuilt at every step in subroutine `bhtree`. The accelerations, which are derived from a softened force law (see §9), are computed for each particle in subroutine `bhtacc`.

## 8.14. **Shifting the grid or expansion center**

It can be desirable when using a polar or spherical grid and field methods, though not a Cartesian grid or direct methods, that the grid or expansion center be the point in the model with highest density of particles. The code therefore includes an option to shift the center. It is an option that needs to be explicitly turned on, *i.e.* the default is not to recenter.

Note that recentering should never be used when the simulation includes a rigid mass component, since then the grid center should be fixed to the center of the rigid component. (In such cases, it can also be important to suppress lop-sided terms, $m = 1$ or $l = 1$, in the force determination, in order to avoid unbalanced forces.)

Note also that only the coordinates of the grid center are revised while the particle coordinates remain unchanged, so that linear and angular momenta about the original center are conserved (§13.1.2).

In the case of multiple grids, the center defined by the particles associated with each grid is computed separately, and the grids can move separately if desired (§12.2). But when the grids are supposed to have a common center, in the case of a single multiple component galaxy say (§12.3), the two centers are forced to coincide.

### 8.14.1. *Finding the center*

The code offers three options:

1. The most well tested method employs the technique suggested by McGlynn (1984), who determined the point $\boldsymbol{c}$ that minimizes
$$\Omega = \sum_N \left[|\boldsymbol{x}_n - \boldsymbol{c}|^2\right]^k . \tag{101}$$
Adopting $k = 1/2$ finds the centroid, or the median position, of all the particles. (Choosing $k = 1$ weights the particles by distance, which is equivalent to finding the CoM if all particles have equal mass.) If we define $a_n = |\boldsymbol{x}_n - \boldsymbol{c}|$, we need to find the zeros of the three derivatives
$$\boldsymbol{f} = \frac{\partial \Omega}{\partial \boldsymbol{c}} = -\sum_N 2k a_n^{2k-2}(\boldsymbol{x}_n - \boldsymbol{c}), \tag{102}$$
which we do by Newton-Raphson iteration (see Press *et al.* 1992, §9.6, p271 with the signs changed):
$$\boldsymbol{c}_{\rm new} = \boldsymbol{c}_{\rm old} - \boldsymbol{d} \qquad \text{where the components of } \boldsymbol{d} \text{ are the solutions of:} \qquad \sum_j \alpha_{ij} d_j = f_i. \tag{103}$$
The matrix coefficients are
$$\alpha_{ij} = \frac{\partial f_i}{\partial c_j} = -\sum_N \frac{\partial}{\partial c_j}\left[2k a_n^{2k-2}(x_{i,n} - c_i)\right] = -\sum_N 4k(1-k) a_n^{2k-4}\left[(x_{i,n} - c_i)(x_{j,n} - c_j) - a_n^2 \delta_{ij}\right]. \tag{104}$$
The iteration is repeated until the change is acceptably small.
2. Computing the center of mass of some number, currently 100, of the most gravitationally bound particles in each component.
3. Following the motion of an extra-heavy particle that is initially placed at rest at the center. In the case of an orbiting satellite, heavy particles are needed to track both centers. This method suffers from Brownian motion of the heavy particle due to the attractions of the other particles, and is not recommended.

### 8.14.2. *Predicting the motion of the center*

Having estimated the position of the center from the current positions of the particles, mass assignment requires a prediction for its future position, one step later. This is done by fitting a 2nd order polynomial to a few, typically 5, recent positions, giving decreasing weight to older values. The polynomial is extrapolated forward over time to predict the position of the center when the particle masses are assigned to the grid.

It is inefficient to redetermine the center at every step and, when time step zones (§11.2) are implemented, it can be done only when particle positions are all synchronous at the end of the step when the most slowly moving particles are advanced. Therefore, the predicted positions of the grid center must be computed for each time zone separately, and the predictions are continued until the next moment at which the center is determined. This is usually at the interval of the largest step size, but could be a multiple of that.

Since mass was assigned using the predicted center, the code adopts the predicted position as the new center. However, the center determined by one of the procedures described above is used as the most recent value in the polynomial fit to predict the next shift.

The potential integral (4) yields the Newtonian gravitational potential of a density distribution $\rho(\boldsymbol{x})$ if the kernel is the potential of unit point mass: to wit, $P_{\rm N}(\xi) = -G/\xi$, where $\xi = |\boldsymbol{x}' - \boldsymbol{x}|$ is the scalar distance of the source from the field point. This singular form for the kernel is numerically inconvenient for point mass particles, because it would require very short time steps for accurate integration during the close passage of a particle pair, and the large angle deflections that would result from such an event are negligibly rare in galaxies, where the particles (stars) are more numerous, and therefore less massive.

Most collisionless simulations therefore either directly or indirectly, adopt a spherically symmetric kernel that asymptotes to the Newtonian form for $\xi \gg \epsilon$, but which behaves as $P(\xi) \to$ const. for $\xi \ll \epsilon$, where $\epsilon$ is denoted the **softening length**. Note that softening clearly introduces a **bias** that limits spatial resolution. It is generally argued (*e.g.* Monaghan 1992) that the bias is minor provided the kernel scale is small in comparison with the scale on which the density varies. While a valid argument, it should be borne in mind that the relevant scale for a disk is the vertical density scale, not the radial.

Since collisional relaxation in quasi-spherical systems is dominated by the cumulative effect of distant encounters (BT08), the rate is little affected by gravity softening. In fact, the formula for the relaxation rate is changed only by a small adjustment to the Coulomb logarithm caused by the substitution of $\epsilon$ for $b_{\rm min}$, the minimum impact parameter for large deflections, in the notation of BT08. This is not true in 2D, where close encounters dominate (see §9.1) and the story in finitely thick disks is more complicated (see *e.g.* Sellwood 2013a).

An acceptable softening kernel should yield the full Newtonian attraction at distances $\xi \gg \epsilon$, and forces that taper smoothly to zero for $\xi \ll \epsilon$. These asymptotic properties of a softening law, together with continuity of first and second derivatives, are the only requirements of significance. The precise form of the force at short-range should not matter because forces in a 3D collisionless system are dominated by the distant mass distribution. Thus, if the behavior of the $N$-body model is to mimic that of a galaxy, its evolution should be insensitive to the adopted force law at short-range. Put another way, if the behavior is affected by the short-range form of the softening kernel, then the simulation is not collisionless.

In the *GALAXY* package where softening is either explicit or implicit, $\epsilon$ has a fixed value for all particles. If particles were to have unequal softening parameters, the forces between particles must take account of the values of $\epsilon$ of both particles to ensure their mutual attractions are equal and opposite. The use of a fixed softening parameter avoids this complication.

Explicit softening is used for cylindrical polar grids and direct methods while softening is implicit when using the 3D Cartesian grid. Implicit softening arises because mass assignment and force interpolation cause particles separated by less than a few mesh spaces to attract each other more weakly than Newtonian (see §10.1.1) as if they were overlapping particles of finite size; indeed, grid methods have long been dubbed "finite size particle methods" in plasma physics (Hockney & Eastwood 1981). Forces are smoothed differently with the spherical grid, while field methods yield the full Newtonian attraction of a smooth mass distribution without softening.

The *GALAXY* package offers two explicit softening kernels for methods, such as polar grids and trees, that require one. Either can be selected for any application, but the recommended choice depends on whether full 3D motion is allowed or whether particles are confined to a plane. This is because they differ slightly in their *long-range* behavior, as follows.

### 9.1. **2D simulations**

Simulating galaxy disks with the motion of particles confined to a plane has the obvious advantage of reduced computational cost over fully 3D simulations. Rybicki (1972) correctly pointed out that close encounters dominate relaxation in a 2D system of particles interacting with inverse-square forces, which therefore could never be collisionless. Unlike in 3D, however, the dominance of short-range encounters implies that the relaxation rate is greatly slowed by gravity softening, enabling these codes to satisfactorily mimic collisionless systems (*e.g.* Sellwood 1983; Inagaki *et al.* 1984).

The most appropriate gravity softening kernel for 2D simulations is:

$$P_{\rm P} = -G(\xi^2 + \epsilon^2)^{-1/2}. \tag{105}$$

This rule is simple and inexpensive to implement. It is usually described as Plummer softening because the implicit 3D density profile (eq. 6) is that of Plummer's model (BT08). To retain the concept of a 2D system, we can equivalently think of each particle being replaced by a 2D Kuzmin-Toomre disk (eq. 173), which has identically the same mid-plane potential as a Plummer sphere of the same mass and radial scale.

The principal advantage of the Plummer softening rule for application to disks is that it provides an approximate allowance for disk thickness, as follows. Convolution of the kernel (105) with the mass distribution of a razor-thin disk clearly yields the exact Newtonian field in the mid-plane, were the disk mass divided into two equal fractions offset above and below the mid-plane by a vertical distance $\epsilon$. In real, finitely-thick, galaxy disks, the field everywhere is the sum of the Newtonian fields of the various mass elements spread in layers parallel to the mid-plane. The Newtonian forces experienced by the

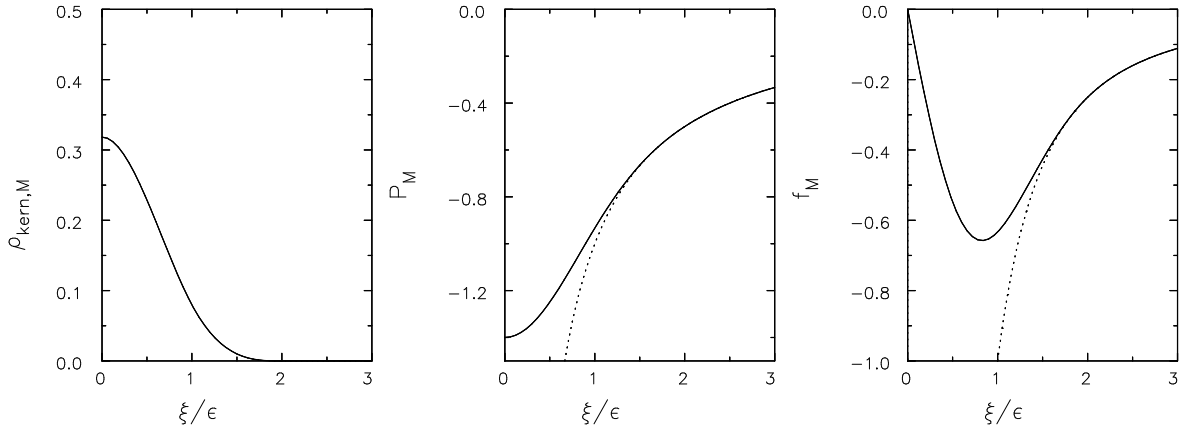


FIG. 6.— Solid curves show the density, potential, and central attraction that results from the cubic-spline softening kernel proposed by Monaghan. The dotted curves show the coresponding Newtonian functions, which are identical when $\xi/\epsilon \geq 2$.

stars are therefore weaker than if the mass distribution were razor-thin. Thus the value chosen for $\epsilon$ in a 2D simulation should be related to the finite thickness of the disk (Romeo 1998).

Note that gravity softening weakens non-axisymmetric instabilities (Sellwood 1983), as follows. The Newtonian potential of an arbitrary razor-thin mass distribution can be determined by expansion in Bessel functions (BT08 §2.6.2), for which the potential of each radial wavenumber, $k$, of the expansion decays away from the plane as $\exp(-|kz|)$. Since softened gravity is equivalent to sampling the field of a 2D mass sheet in a plane offset vertically by a distance $\epsilon$, the disturbance potential of each term is therefore weaker by the factor $\exp(-|k|\epsilon)$. Hence gravity softening weakens the potential of the disturbance, causing instabilities to be less vigorous. However, this weakening is physically realistic because softening provides an approximate allowance for the real finite thickness of galaxy disks as explained above.

### 9.2. **3D simulations**

Many authors have noted that Plummer softening converges only slowly to the Newtonian attraction, and significantly weakens forces at all distances. While this property is desirable for 2D disks, because it makes some allowance for thickness, it is undesirable in 3D and kernels that soften the Newtonian force law to a finite distance only are to be preferred.

I have adopted the piecewise cubic kernel proposed by Monaghan (Monaghan & Lattanzio 1985; Monaghan 1992) that has the effective density form

$$\rho_{\rm kern,M}(\xi) = \frac{1}{4\pi\epsilon^3}\begin{cases} 4-6x^2+3x^3 & 0\leq x\leq 1 \\ (2-x)^3 & 1\leq x\leq 2 \\ 0 & x>2, \end{cases} \tag{106}$$

with $x = \xi/\epsilon$. The potential is

$$P_{\rm M}(\xi) = \frac{G}{\epsilon}\begin{cases} -1.4+x^2(2/3-0.3x^2+0.1x^3) & 0\leq x<1 \\ -1.6+1/(15x)+x^2(4/3-x+0.3x^2-x^3/30) & 1\leq x<2 \\ -\epsilon/\xi & x\geq 2, \end{cases} \tag{107}$$

and the mass profile

$$m_{\rm kern,M}(\xi) = \begin{cases} 4x^3(1/3-0.3x^2+0.125x^3) & 0<x<1 \\ x^3(8/3-3x+1.2x^2-x^3/6)-1/15 & 1<x<2 \\ 1 & x>2. \end{cases} \tag{108}$$

The central attraction $f_{\rm M}(\xi) = -Gm_{\rm kern,M}(\xi)/\xi^2$. These somewhat clumsy expressions, plotted in Fig. 6, are well thought out to fulfill all requirements of continuity of derivatives.

The particles in a simulation move continously while, in a grid method, the gravitational field is determined at points in a regular raster. Furthermore the field is calculated from masses on the same discrete set of points. Thus two interpolation rules are required: the mass of a particle must be distributed over surrounding grid points, and the acceleration to be applied to the particle must be interpolated from the values at surrounding grid points. It is relatively easy to show that we must distribute the mass with the same weighting scheme as we use to interpolate the forces in order to guarantee that forces between particles are equal and opposite and to avoid a grid-induced self-force acting on each particle.

The default scheme that I employ is linear interpolation between the nearest grid points, which is also known as the cloud-in-cell (CIC) scheme. This involves bi-linear interpolation between 4 grid points in 2D, as illustrated in Fig. 7, while in 3D we use tri-linear interpolation between 8. I have programmed quadratic interpolation (aka TSC), which smooths over 27 points in 3D and does improve force quality, though at the cost of slightly reduced resolution and more computational effort. However, as shot noise from the particles is generally the most important limitation of simulations, I have found it preferable to utilize computational resources to maximize $N$ rather than to improve interpolation.

## 10.1. **Cartesian grids**

The mass-fractions or weights, to be assigned to each of the surrounding grid points (in 3D) are computed as follows. Let the particle be at $(x, y, z)$, while the grid-point positions are $(ih, jh, kh)$, where $h$ is the grid spacing. Assuming, as is arranged internally in the code for efficiency, that particle position coordinates are in units of $h$, it is straightforward to identify the grid cell that contains the particle. We then find the sub-cell positions

$$\delta x = x - i, \qquad \delta y = y - j, \qquad \delta z = z - k, \tag{109}$$

which are all between zero and one. The weights for the different grid points are therefore

$$\begin{aligned}
w(i,j,k) &= (1-\delta x)\ (1-\delta y)\ (1-\delta z)\\
w(i+1,j,k) &= \delta x\ (1-\delta y)\ (1-\delta z)\\
w(i,j+1,k) &= (1-\delta x)\ \delta y\ (1-\delta z)\\
w(i+1,j+1,k) &= \delta x\ \delta y\ (1-\delta z)\\
w(i,j,k+1) &= (1-\delta x)\ (1-\delta y)\ \delta z\\
w(i+1,j,k+1) &= \delta x\ (1-\delta y)\ \delta z\\
w(i,j+1,k+1) &= (1-\delta x)\ \delta y\ \delta z\\
w(i+1,j+1,k+1) &= \delta x\ \delta y\ \delta z\ .
\end{aligned} \tag{110}$$

The location of the $(i, j, k)$ corner and the eight $w$ values are computed for each particle twice at each step. The first time to determine how the particle mass is distributed over the nearby grid points, and the second to interpolate from the grid values to obtain the acceleration to be applied to the particle. While a slight improvement in efficiency would result if this additional information for each particle were stored during the solution for the field, the memory requirements for large $N$ argue against doing so.

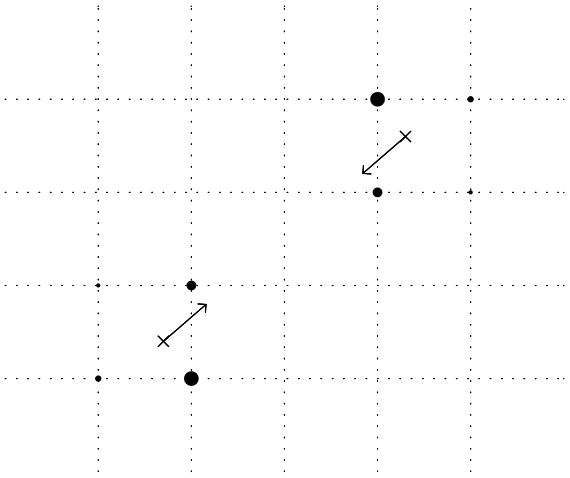

FIG. 7.— An illustration of interpolation on a 2D Cartesian grid. Two particles are marked by crosses and the relative weights assigned to the grid corners are indicated by the sizes of the black circles. The mutual attractive forces, indicated by the arrows, are equal and opposite in magnitude, but are deliberately shown to be not quite co-linear, as may happen.

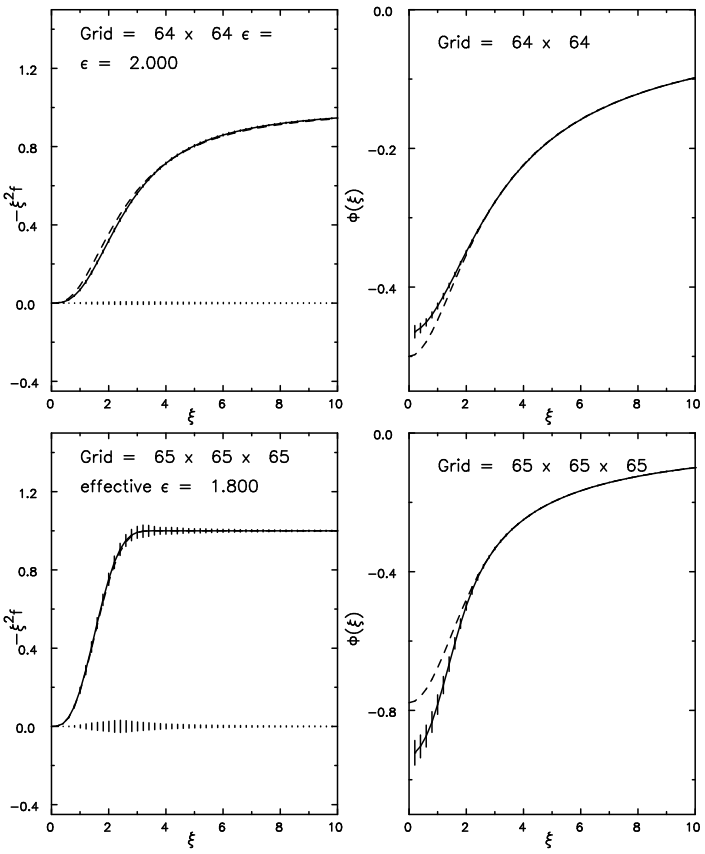


FIG. 8.— The central attraction (left), and potential (right) experienced at many field points computed with many positions of a unit source mass. The top panels show the effect of adopting a Plummer softening kernel on the 2D Cartesian grid and lower panels show the forces on the 3D Cartesian grid. The dashed curves show the analytic Plummer softening function for the 2D grid (above) while in the lower panels they show the Monaghan kernel for illustration.

### 10.1.1. *Force quality*

An ideal collisionless simulation should not be concerned with forces between individual particles, since each particle is supposed to experience only the mean attraction of the large-scale mass distribution. Thus one might suppose that the only test worth making is of the global attraction of a set of particles representing a galaxy. However, grid induced errors in the acceleration applied to a particle introduce spurious deflections to its motion, known as grid noise, which is an artifact of grid methods that needs to be minimized.

Fig. 7 exaggerates that the grid-mediated attraction between two particles may not be perfectly co-linear, which causes a residual mild couple of random orientation between every particle pair. Furthermore, both this couple and the magnitude of the mutual attraction between the particle pair may also depend upon their precise positions within each grid cell. Both these minor differences are relatively more important when the distance between the particles is less than a few grid spaces.

Thus the domain of influence of these errors can always be reduced by refining the grid. The errors can also be reduced by spreading the mass of each particle over more grid spaces – see §10. However, since particle shot noise is the principal source of error in simulations, such refinements are generally not worthwhile.

Fig. 8 illustrates the relative magnitude of these grid-dependent parts of the forces that results on Cartesian grids. To make these plots, I placed a unit point mass at 100 arbitrary positions $\boldsymbol{x}$ within one grid cell and for each computed the forces and potentials experienced at 100 field points $\boldsymbol{x}'$ spaced uniformly around a circle in 2D or on the surface of a sphere in 3D at each of the radii marked by error bars. The force components are

$$f_r = \boldsymbol{a}\cdot(\boldsymbol{x}-\boldsymbol{x}')/\xi \qquad \text{and} \qquad f_t = [|\boldsymbol{a}| - f_r]\,, \tag{111}$$

where $\xi = |\boldsymbol{x}-\boldsymbol{x}'|$. The error bars in the left hand panels show the means from the $10^4$ estimates at each distance and $\pm\sigma$ variations of $\xi^2 f_r$ and of $f_t$, which is close to zero at all distances. The right hand panels show the mean $\Phi(\xi)$ at the same field points.

The top panels show results from employing a Plummer softening kernel on a 2D Cartesian grid with $\epsilon = 2$ grid spaces. It can be seen that the force variations around the analytic value, together with any couples that result from slightly

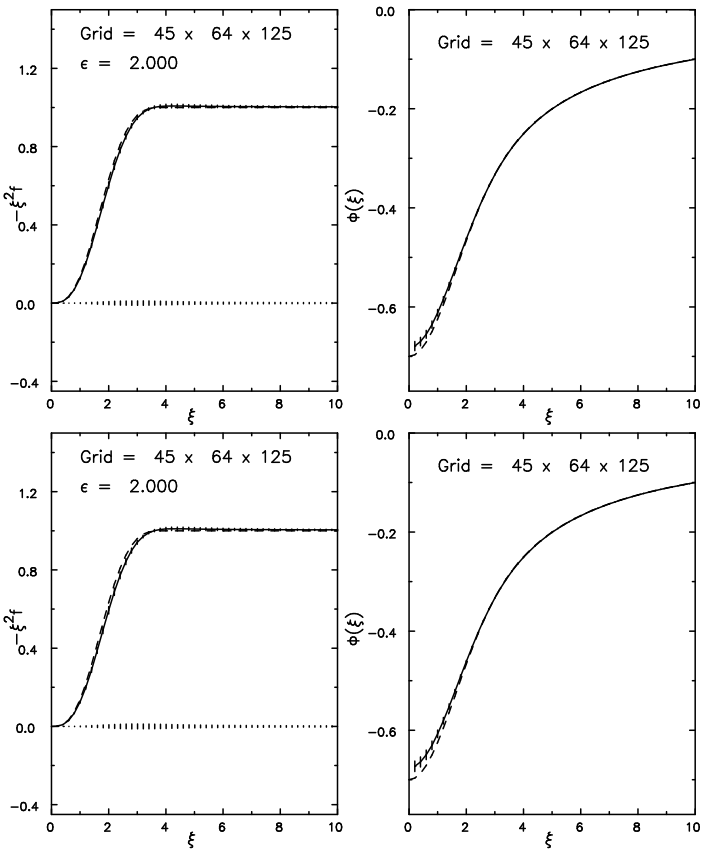


FIG. 9.— As in Fig. 8 but for the 3D polar grid, and the Monaghan kernel. The top row is for tri-linear interpolation, or 8-point mass assignment, and the bottom row for tri-quadratic, or 27-point mass assignment.

non-central forces, are both tiny. Notice that the potential function is smoothed slightly more by the grid at short range.

The lower panels show the same for the 3D grid. Note that for this grid type only, forces are derived from differencing the potential (eq. 7), since only the potential is computed by solving Poisson's equation instead of employing force convolution in all other methods. It can be seen immediately that the forces are of lower quality, with significantly greater variation, especially over the range 2 – 3 mesh spaces. Comparison of these variations with the much smaller ones in the top panels confirms that the extra convolutions needed to obtain the forces on other grids do yield a significant benefit (as anticipated in §8.1). Softening arises only from the grid in this case, and cannot be predicted analytically, but I have drawn the function that results from the Monaghan softening kernel (§9.3) with $\epsilon = 1.8$ grid spaces, that seems to be an excellent fit to the actual grid-determined attraction.

In the 3D case the potential computed from the grid is a little deeper than that of the same Monaghan kernel, whereas the deviation between the measured potential and the Plummer kernel on the 2D grid was in the opposite sense (Fig. 8). Differences between the measured potential and that of the fitted analytic function imply that the code cannot conserve energy exactly. Energy conservation could never hold because *both* the forces and the potential vary linearly across a grid cell. There is no simple fix for this, since an attempt to use quadratic interpolation for the potential, to conform to the line integral of the forces acting, results in discontinuities in the potential function at cell boundaries. Provided the mean distance between particles does not change, energy variations average to zero, but measurable changes, which are independent of time step and particle number, can result in models that collapse to denser structures.[13]

### 10.2. **Polar grids**

Interpolation on polar grids is similar, except that grid spacings are non-uniform. The code identifies the appropriate grid cell from the Cartesian position coordinates and, since the radial grid points are spaced logarithmically, it interpolates linearly in the logarithm of the radius. Exceptionally, quadratic interpolation is also an option when using the 3D polar grid, but the default is linear even for this grid.

Fig. 9 shows the force quality on the 3D polar grid, where again the three components of acceleration are convolved separately. The top row is for the default mass assignment scheme while the bottom row is for quadratic interpolation,

[13] Energy changes can also result from the use of too large a time step, which can be remedied by using a shorter step.

for which the error bars, which indicate the rms variation, are somewhat smaller.

The good correspondence between the measured force and the analytic kernel in this Figure shows that grid can deliver the forces of a softened point mass. However, this is the case only when the source particle is placed in a well-resolved part of the grid and all possible sectoral harmonics are included in the field determination. On a polar grid, the attraction of a point mass will generally be altered by two features:

1. Grid cell sizes increase linearly (almost – see eq. 25) with cylindrical distance from the grid center, and generally $> \epsilon$ in the outer parts, although the largest cell dimensions should never exceed a few $\times\epsilon$. (The fixed vertical spacing should always be less than $\epsilon$.)

2. It is usually desirable to discard the contributions of the higher sectoral harmonics to the grid-determined forces of a mass distribution that has no physically meaningful density variations on those spatial scales, as described in §8.5.

Both these features cause the forces between individual particles to have much greater variation than that shown in Fig. 9. However, it should be borne in mind that collisionless simulations are not concerned with forces between individual particles, since each particle is supposed to experience only the mean attraction of the large-scale mass distribution. Thus the variations that would result in these cases are not a cause for concern. However, for calculations that yield important results, the user should rerun with a finer grid in order to verfy that grid noise is not affecting results.

### 10.3. **Digression on arithmetic precision**

It was asserted early in this manual (§1.2) that single precision arithmetic is adequate for simulations. There are two reasons for this.

- Shot noise from the finite number of particles is usually the greatest source of error, which is clearly unaffected by converting to higher precision arithmetic.

- The grid-dependent part of the inter-particle forces are also little affected by numerical precision. Since the interpolation scheme is designed to ensure linear momemtum conservation to machine precision, improving precision does lead to tighter linear momentum conservation. But the other effects of the grid, the mild couple between particle pairs, and the small grid-dependent variance in the radial force result from approximations that are inherent to the grid. Thus, using higher precision arithmetic makes no practical difference to force quality – *e.g.* Figs. 8 & 9 are no different when computed using double precision.

As already noted, careful direct tests have verified that numerical results are unaffected by use of double precision arithmetic, confirming these arguments.

Since particle noise is the principal limitation of collisionless $N$-body methods there is nothing to be gained from using a computationally expensive high-order integration algorithm; the code therefore uses leap frog in Cartesian coordinates. This symplectic method is time-reversible, remarkably stable, and achieves second order accuracy (*e.g.* BT08, p200) from two apparently first order shifts when the positions and velocities are half a step apart in time.

## 11.1. **Cartesian leap frog**

Assuming that the particle positions $\boldsymbol{x}_n$ and velocities $\boldsymbol{v}_n$ are known at step $n$, the conventional drift+kick+drift time step (*e.g.* BT08, p200) proceeds as follows:

$$\begin{aligned} \text{Drift:} \qquad & \boldsymbol{x}_n + \boldsymbol{v}_n \delta t/2 = \boldsymbol{x}_{n+1/2} \\ \text{Kick:} \qquad & \boldsymbol{v}_n + \boldsymbol{a}_{n+1/2}\delta t \ = \boldsymbol{v}_{n+1} \\ \text{Drift:} \qquad & \boldsymbol{x}_{n+1/2} + \boldsymbol{v}_{n+1}\delta t/2 \ = \boldsymbol{x}_{n+1} \end{aligned} \tag{112}$$

Thus the positions are drifted in two half steps and the velocites kicked only once. The accelerations $\boldsymbol{a}_{n+1/2}$ must be determined from the particles at their positions $\boldsymbol{x}_{n+1/2}$. Since the second drift in this step uses the same velocity as the first drift of the next step, it is computationally more efficient to use a single full-step kick+drift, which requires the positions to be stored half a step *ahead* of the velocities while the gravitational field is calculated. I refer to this implementation as APLF, for advanced-position leap frog.

An equivalent scheme is kick+drift+kick, where each kick is for half a step, while the drift is a full step. Again, since the second kick uses the same acceleration as the first kick of the next step, one can also optimize this scheme to a single full step kick+drift, provided the velocities are saved half a step *behind* the positions. Thus

$$\begin{aligned} \text{Kick:} \qquad & \boldsymbol{v}_{n-1/2} + \boldsymbol{a}_n \delta t = \boldsymbol{v}_{n+1/2} \\ \text{Drift:} \qquad & \boldsymbol{x}_n + \boldsymbol{v}_{n+1/2}\delta t \ = \boldsymbol{x}_{n+1} \end{aligned} \tag{113}$$

These two simple shifts yield the positions $\boldsymbol{x}_{n+1}$ at the next step, from which the accelerations $\boldsymbol{a}_{n+1}$ may be calculated in readiness to integrate forward again. I refer to this implementation as RVLF, for retarded-velocity leap frog.

Note that both APLF and RVLF schemes are identical in their implementation (kick+drift), except that to start the integration, the positions need to be advanced half a step in the APLF scheme, while the velocities need to be backed-up half a step in the RVLF scheme. The RVLF implementation is the default in the code, since that is required when using block time steps, but the user can insist, if he/she wishes, on using the APLF scheme when a fixed time step is employed for all particles.

Two further points need to be discussed:

1. **Starting** When selecting particles for the initial model, all the phase space coordinates are known at time zero. However, the above equations require $\boldsymbol{v}_{-1/2}$ not $\boldsymbol{v}_0$. In order to prepare to use the RVLF scheme, we must back-up the velocities at step 0 using the following first-order relation and the accelerations at step 0
$$\boldsymbol{v}_0 - \boldsymbol{a}_0 \delta t/2 \simeq \boldsymbol{v}_{-1/2}. \tag{114}$$
2. **Extracting information** When saving velocity information from the simulation, we must also ensure that it is recorded at the same moment as the positions. Again to first order, the time centered velocity $\boldsymbol{v}_n$ is given by:
$$\boldsymbol{v}_{n-1/2} + \boldsymbol{a}_n \delta t/2 \equiv (\boldsymbol{v}_{n-1/2} + \boldsymbol{v}_{n+1/2})/2 \simeq \boldsymbol{v}_n. \tag{115}$$

While higher order schemes could be implemented, I have found these first order formulae to be adequate.

## 11.2. **Block time steps**

As noted in §2.5, it is generally more efficient to use different time steps for subsets of the particles; those in the dense center require shorter steps than those in the periphery of the model, where orbital time scales can be much longer. To achieve this, I allow particles in dense regions, where accelerations are greater, to be advanced with shorter time steps. It is most convenient if the time steps are shortened by successive factors of two, as first explained in Sellwood (1985), which is often described as using block time steps. Adopting block time steps affords a significant reduction in running time for the simulation since only a fraction of the particles need be advanced using the shortest time step, while the majority are moved less frequently. This option is implemented for all field methods and all grid methods except C3D, but is unavailable for direct methods.

The number $n_z$ of levels, or *zones*, in the time step hierarchy is chosen by the user, but is limited to $n_z \leq 2$ for the 3D Cartesian grid only. The time step in the $i$th zone is therefore $\delta t_i = 2^{i-n_z}\delta t$, with $\delta t$ being the basic step entered by the

user as the value of `ts` (see §2.4). Particles in zone 1 therefore move with the shortest step, while those in zone $n_z$ move with the basic step. Note that the variable in the code `istep` counts the number of steps taken in zone 1.

In versions of the code before v16, the time step used for each particle was determined by its location in a number of pre-defined concentric spatial zones. Thus the time step was determined by the particle's distance from the grid center, which works adequately in near equilibrium models in which the overall mass distribution changes little. This option is still available, but a second option has been added at v16 that estimates the time step needed from the magnitude of the acceleration the particle is experiencing. The time step to be used in this case is the largest $\delta t_i$ that is less than the quantity $f_{\rm ts}\sqrt{\ell/|\boldsymbol{a}|}$, where $\boldsymbol{a}$ is the acceleration to be applied to the particle, and $\ell$ is the unit of length (see §2.4). The dimensionless factor $f_{\rm ts}$ is chosen by the user; the value $f_{\rm ts} = 0.05$ seems to work well, but larger values allow longer time steps and *vice versa*. I find it convenient to continue to use the moniker **zone** to categorize the particles having a given time step, even though no spatial region is implied when the time step depends on the acceleration.

Note added on December 13, 2022: The central attraction in mass models having a uniform core rises from zero at the center to the core radius, before decreasing beyond. In this case, the acceleration-dependent time step rule causes time steps to *increase* toward the center, which is undesirable for two reasons:

- Orbit periods are approximately constant within the core, yet those nearer the center are assigned a longer time step, and
- particles on eccentric orbits experience a range of accelerations as they pursue their orbits, and may change their time steps some number of times every (short) radial period.

For these reasons, employing spatially-defined time step zones is to be preferred for near equilibrium models having quasi-harmonic central potentials, because it requires a fixed time step throughout the inner zone.

### 11.2.1. *Advice on the number of acceleration-dependent zones to use*

The purpose of time step zones is to allow the motion of more slowly moving particles to be advanced less frequently, thereby inproving efficiency. However, each zone that is added increases the cpu effort to compute the field, and it is therefore important to find a balance between the number of zones employed and the efficiency of orbit integration.

In order to assist the user to strike the right balance for the acceleration-dependent rule, the code outputs information about the choices of adjustable time steps made during a simulation. the zone number required for a particle at a given step is the integer smaller than the value of $a = f_{\rm ts}\sqrt{\ell/|\boldsymbol{a}|}$, although it cannot be less than 1. In order to check that the number of zones is suitable, the code records the least and greatest values of $a$, and reports it in the runXXXX.lis file at intervals of the largest step. This information should be interpreted as follows:

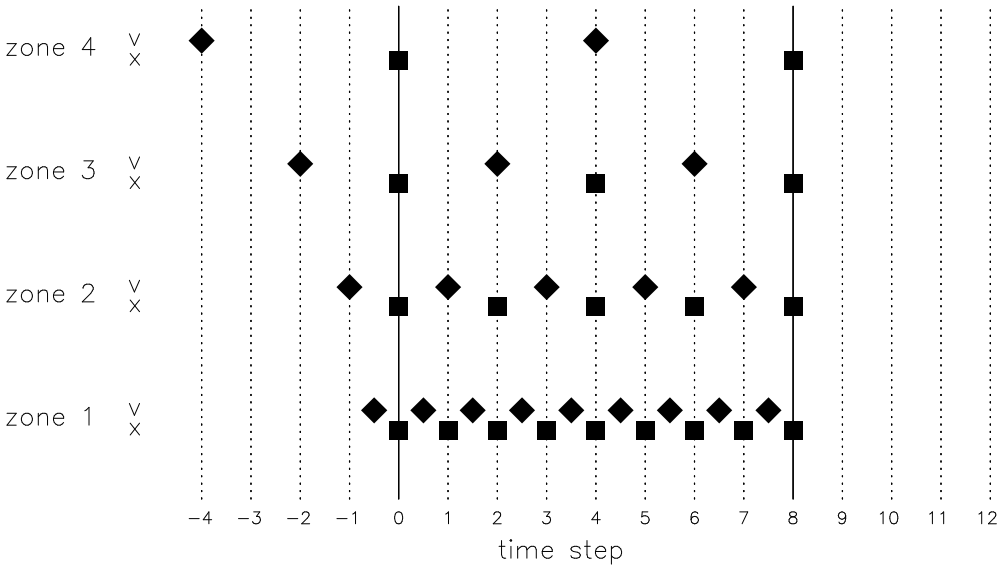


FIG. 10.— The use of block time steps. Particles in the different zones have time steps that increase by successive factors of two. The positions (squares) of all the particles are synchronized at $t = 0$ when the forces can be determined everywhere. Forces to advance particles in zone 1 are needed at the times marked by dotted lines, so the mass distributions from particles in the outer zones are interpolated to those times before the gravitational field is determined. Once the field is available at the appropriate moment, the velocities (diamonds) can be advanced, followed by the positions.

The smallest value, $a_{\min}$, should be of order unity; if it much less, then the time integration of particles that experience the strongest accelerations is inaccurate, and the user should consider restarting the simulation with either a shorter basic step or with more zones. If, on the other hand, the smallest value is greater than 2, then the number of zones can be reduced, which will improve running speed.

The largest value, $a_{\max}$, indicates the maximum number of zones that could be used. It is wasteful of memory and execution time for this number to be smaller than the number of zones, and the solution is to increase the basic time step and reduce the number of zones. However, nothing undesirable happens if this number is larger than the number of zones – particles in the highest zone are being advanced using a step that is shorter than the minimum that is really needed. The user could consider employing a larger basic step, and more zones so that the most strongly accelerated particles are still integrated properly. But since the particle density generally drops rapidly in the periphery of the model, the increased efficiency of using larger steps for the diminishing numbers of particles in the outer parts of the simulation can quickly be outweighed by the cost of computing the gravitatinal field in more zones.

### 11.2.2. *Combining the field from all the zones*

We continue to employ leap frog for particles that are advanced by a step size that is a fraction of $\delta t$. The equations are the same as eq. (113), except that $\delta t$ is replaced by $\delta t_i$, and the time centered velocity components are saved at time $\delta t_i/2$ behind the positions. The whole scheme is illustrated schematically for four zones in Fig. 10. The positions of the particles in each zone (squares) are synchronized at step 0, so that the gravitational field of the entire system can be computed. The particle velocities (diamonds) are all known at half a time step earlier, so that all the particles can be advanced by one step, moving them to the right, *i.e.* forward in time, to the next locations marked respectively by the squares and diamonds. As soon as this step is taken, however, the particle positions are no longer synchronized, and particles with shorter time steps need to be advanced until they have caught up with those on longer steps.

In order to take the next step for the particles having shorter steps, we require forces from those in the outer zones at times that are intermediate between their known positions, marked by squares. Since we have already advanced particles in the outer zones, we can assign their masses to a copy of the grid at their new positions, while we keep a second copy of their mass distribution at their previous step. Thus, at intermediate times, we can **interpolate** linearly (in time) between the mass arrays from the outer zones to approximate the mass distribution at the required time. Combining the mass arrays for the different zones yields a total mass distribution from which forces can be determined, appropriate for the time we need to advance particles having shorter steps.

Prior to v16, the calculation of the field on polar grids when multiple zones were employed could be restricted to only those regions of the grid where it would be needed, allowing some minor improvement in efficiency. However, the region of the grid where accelerations would be required can no longer be predicted when time steps are acceleration dependent, and code was revised at v16.04 to compute the field over the entire grid at every (sub)step.

It is important to note that whenever a particle is stepped forward, its own contribution to the force arises from its current position, without interpolation. This scheme is stable therefore and no particle experiences a self-force. It is also time-reversible because the mass distributions are interpolated.

Block time steps require the RVLF implementation because, as particles in outer zones need to be advanced, the forces from themselves and all particles in inner zones are available at the same instant. Were the APLF scheme to be used, mass distributions are never synchronized and the forces on particles in each zone would require a mass distribution that would need to be interpolated from all the other zones separately, thereby necessitating both considerable extra work and additional approximation.

Thus we must save mass arrays from the particles moving with each time-step both from their previous positions, when they were last moved, and from their new positions. For all but the shortest steps, where interpolation is never required, the code determines the weighted sum, proportionately to the difference in time, of the "before" and "after" mass arrays from each of the outer zones as a good approximation to the complete mass distribution at the current moment. As each zone of particles is stepped forward, the mass array at their new positions is computed and stored, replacing that from their positions one step earlier.

This scheme is not implemented for the 3D Cartesian grid for two reasons: (1) multiple copies of a large grid would take up a great deal of memory and (2) this type of grid, with its fixed spatial resolution, is unsuitable for mass models having dense mass concentrations that would require shorter steps.

### 11.2.3. *Changing time steps*

The time step of a particle must reduced immediately it enters a zone with a shorter step, or when the acceleration requires it, which is possible because its motion will be synchronous with all the other particles moving with that shorter step. However, when the time step can safely be increased, its motion may or may not lag behind the other particles having the longer step. If it lags, it must continue to be integrated at the shorter step until it has caught up with the particles in the new zone; its time step is then adjusted as the particles in its new zone are being stepped forward.

A particle's time step is changed when the above conditions are met. Its time-centered velocity must be adjusted for the new step, which again is carried out to first order:

$$\left(\boldsymbol{v}_{n-1/2}\right)_{\rm old} + \boldsymbol{a}_n \left(\delta t_{\rm old} - \delta t_{\rm new}\right)/2 \simeq \left(\boldsymbol{v}_{n-1/2}\right)_{\rm new} . \tag{116}$$

This formula requires an acceleration, $\boldsymbol{a}_n$, that must be at the appropriate position and time, *i.e.*, the velocity must be adjusted when the particle is at the position $\boldsymbol{x}_n$ before the most recent step was taken. Thus, we revert to its position $\boldsymbol{x}_n$ and velocity $(\boldsymbol{v}_{n-1/2})_{\rm old}$, and implement its new time step, $\delta t_{\rm new}$, by adjusting its velocity using eq. (116). Then, using the new time step $\delta t_{\rm new}$, the revised velocity $(\boldsymbol{v}_{n-1/2})_{\rm new}$, and the original position $\boldsymbol{x}_n$, we advance its motion a second time in the usual way (eqs. 113).

I have tested this scheme by integrating test particle orbits for several hundred radial periods as they oscillated across several zone boundaries. Even for a particle that crossed and recrossed zone boundaries, the repeated first order revision to its velocities did not lead to any secular changes in its energy, *etc.*

### 11.3. **Time reversibility**

The current release of the code is not exactly time reversible. In order to make the above scheme excactly time-reversible, an increase of time step should be made only when *both* the new position $\boldsymbol{x}_{n+1}$ and the old position $\boldsymbol{x}_n$ are in the new zone. With this rather minor modification, I have confirmed that successive positions of a particle across step changes were identical (to machine precision) after its motion was reversed. In fact, this must be the case because the change of step occurs at the same position regardless of the direction of motion, which is sufficient to ensure exact reversibility.[14] In fact, every particle is in exactly the same position, ensuring that self-consistent forces are unchanged and therefore the entire model is reversible. I thank Walter Dehnen for prompting me to address this issue.

Not only is the current time-step procedure not precisely reversible, but perfect reversibility is rendered impossible when the grid center is shifted (see §8.14) or when guard radii are employed (§11.4). The latter option is rarely used, however.

Perfect time reversibility is hard to achieve (Dehnen 2017). Though pleasing, it is not critical and evolution can be retraced even when the scheme not perfectly reversible. For example, Sellwood & Debattista (2009, their Fig. 22) present reasonably successful tests of reversibility in a self-consistent simulation when the motions of all the particles were reversed simultaneously.

### 11.4. **Guard radii**

It may also be advantageous, *e.g.* in the vicinity of a dense central mass, to integrate the motion using extra-short steps during which the field is assumed to be frozen, as first described in Shen & Sellwood (2004). We described this as employing **guard radii**.

Computing the full self-consistent field of the particles in even a small zone can be time consuming and, if only a small fraction of the total acceleration to be applied to each particle in that region arises from the particles, then it may be an adequate approximation to advance the motion on a shorter time step, neglecting any possible change to the self-consistent part of the acceleration during the sub-steps.

The code allows a nested sequence of guard radii to be defined, inside of each the time step used to advance the motion is subdivided as desired to achieve a more accurate integration in the strong external field. Such particles are described as being in "intensive care" and are advanced one at a time in as many substeps as are needed until it has been advanced by the time $\delta t$, when a new force determination is required for the whole inner zone. As the force field is assumed fixed, changes to the time step can be made the moment that each particle crosses any one of the guard radii.

The acceleration applied at each substep is computed at the instantaneous position of the particle, and has two parts. The attraction of the particles is approximated by the most recent determination of their gravitational field, which therefore neglects any small changes caused by their motion, together with the fixed attraction of the central mass. This scheme, which is no longer time-reversible, is adequate only if the attraction of the fixed mass is much greater than that of the particles throughout the region where it is applied.

As each particle moves ahead of the point at which it last contributed to the field, it experiences a self-force towards its previous position. It is possible to compute the self-force and subtract it off from the grid-determined attraction, although I have found it necessary to do this only in the case of the spherical grid, where strong forces between particles are well-resolved in the grid center.

### 11.5. **Subroutine STEP**

[14] When checking this argument, it is important to bear in mind that a step increase may be delayed until particles having the larger step are to be moved.

Before advancing the motion, the accelerations to be applied to each particle need to be prepared, generally by interpolation from the grid (see §10). In addition, this subroutine manages any changes in time step zones by the particles and assigns the mass of the particles at their new positions to the appropriate zone on the grid. The user can specify that the analysis software be executed at particular time steps, which is also implemented in this routine.

### 11.5.1. *Linked lists*

While employing a variety of different time steps is conceptually very simple, it creates requirements that drive the structure of the code.

Any given step will need to advance the motion of the particles in the innermost zone, while particles in the outer zones are advanced progressively less frequently. The routine first determines which zones are to be advanced, and then works through the pre-established chains of particles in each of the selected zones.

The coordinates and flags of all the particles are stored in a 1D array `ptcls`, as described in §2.7. As the simulation progresses, the address of each particle in this array never changes but, of course, the values of the coordinates of each particle do change.

The use of block time steps implies that only a fraction of the particles are advanced at a general time step. In order to pick out the particles in any one time step zone, I employ a linked list, or chain, system, as follows:

In addition to saving the coordinates of each particle, I save a pointer that is the index in the `ptcls` array of the next particle in the list. The index of the first particle in each chain is stored in a separate table, and the end of a chain is flagged by a negative value. These chains are rebuilt at every step, since particles may change time step zones at any time. As the new coordinates of a particle are saved, the pointer to the start of its chain is copied from the start table, and the start table updated with the array index of the new particle. Thus the computational overhead associated with maintaining these lists is minimal.

Note that were most particles to stay in the same zone, such as happens in rotationally supported disks, the linked list system reverses the order in which particles are processed at each step. Were it not for stochasticity (§11.7), the order in which they are advanced would not matter at all.

### 11.5.2. *Local buffer*

The entire set of particle coordinates, and any flags must, of course, be stored between time steps. However, while stepping particles forward, it has proven convenient to save additional information temporarily for each particle. The code therefore advances particles in groups of up to 1000, keeping a wide variety of information about each in a local buffer.

Working down the chain, the particles in a given list are processed in batches, or groups, of up to 1000, in the following manner:

1. A call to `gather` collects the next group of particles in the current chain from the main array `ptcls`. This routine places a copy of their coordinates at the start of the step in the local array `oldc` and also decodes any flags.
2. The accelerations to be applied to each of these particles are determined in a call to `getacc` and stored in array `acc`. They are generally interpolated from the grid and may include externally applied accelerations.
3. If this is an analysis step, a call to `anlgrp` will cause the contributions of this group of particles to be added to the cumulative analysis of the model. The needed initial coordinates and accelerations have previously been assembled.
4. The motion of each particle is advanced in a call to `stpgrp`, which places the updated coordinates in the array `newc`. This routine also flags any particles that leave the grid at this step.
5. A call to `scattr` copies the updated coordinates of each particle in the group, temporarily stored in array `newc`, back to its original location in the main particle array, and updates the linked list for whichever zone they are in.
6. Finally, the mass of each particle at its updated position is assigned to the grid array that accumulates the masses of particles in each zone by a call to `massgn`

Steps 1 – 6 are followed for successive groups of particles in the list until all in that zone have been processed. The routine then initiates processing of the particles in the next zone to be advanced at the current step, working in batches through steps 1 – 6, until all particles in the selected zones have been processed.

## 11.6. **Off-grid particles**

Particles leaving the grid can no longer contribute to the grid-determined forces. The loss of particles from the grid will not materially affect the evolution of those that remain provided the escapers are few in number and the bulk of the active mass is far from a boundary. Escaping particles may be, and generally are, still gravitationally bound to the model.

If the evolution remains of interest after a significant fraction (say 5%) of the particles have left the grid, the user should restart the simulation with a smaller value of `lscale` in order that the initial particle distribution occupies a smaller fraction of the grid volume. Equivalently, the user can employ a larger grid, which is inexpensive to accomplish in the case of cylindrical polar or spherical grids.

The code provides four options for dealing with particles that leave the grid, aside from ignoring them (*i.e.* all logicals set to `.false.`).

1. Logical variable `offanal` set to `.true.` Continue their motion in a straight line at constant speed. The kinetic energy and angular momentum of these particles is conserved and included in the total measurements. This option is recommended at a minimum.
2. Logical variable `offthf` set to `.true.` Continue their motion under the combined theoretical attraction of all mass components, neglecting any changes to their mass profiles that may have taken place as a result of the evolution.
3. Logical variable `offmnp` set to `.true.` Continue their motion under the combined attraction of any rigid mass components together with that of all the particles remaining on the grid, estimated with the assumption that their center of mass is the grid center.
4. Logical variable `offfor` set to `.true.` Continue their motion with the attraction between each particle off the grid and all others, whether on or off the grid, computed by pairwise summation. This option, while avoiding approximation, becomes very time-consuming if more than a handful of particles are off the grid and is not recommended; the user should simply restart with a larger grid, as noted above.

Under options 2 – 4, any particles re-entering the grid at later times are re-activated.

## 11.7. **Stochasticity**

Since the evolution of all gravitating $N$-body systems is stochastic, differences at the round-off level can be quickly magnified into macroscopic differences, behavior that can be especially troublesome for disks (Sellwood & Debattista 2009). However, the code is constructed in such a way that results from it can be reproduced exactly, if the same set of particles is rerun with the same numerical parameters on computers with the same operating system and arithmetic units. This feature enables the calculation to be stopped and restarted so that the evolution will continue as if no interruption had occurred.

The thinking reader may wonder how this can be true, since changing the order in which particles are included into the mass distribution for the next call to the Poisson solver, or to construct the tree, or whatever, does affect the resulting accelerations at the round-off error level. However, I have taken care to ensure that not only are the particle coordinates saved, but all necessary information is also saved so that a restarted run will execute exactly the same arithmetic operations in the same order, as in a comparison run that was not interrupted. This is a useful feature that enables a simulator to replicate a previous calculation exactly.

Simulations need not be restricted to a single grid, or to isolated systems. The software offers a number of possibilities to study the effects of perturbers on galaxies, both as external companions or as central masses, or bars that are not treated self-consistently. The companion could also be represented by particles that are a hosted on a separate grid, with the centers of the two grids moving with the density maximum of each component. A further option is to host a large number of particles on the grid to represent a collisionless system and to add some heavy particles as perturbers. This section describes all these possibilities in more detail.

## 12.1. **Rigid perturbers**

A rigid perturber can be introduced into the model by adding lines to the `.dat` file (described in §5.3). The user can either supply the code to determine the acceleration components to be added, or use that provided for a few special cases, as outlined in this subsection.

The attraction of the perturber is added to the acceleration acting on each particle before it is moved. The vector sum of all the attractions from the particles is also computed in order to advance the motion of the perturber.

If the perturber moves, as in some of the following examples, then the grid recentering option should be activated when polar grids are used in order to prevent the model from drifting away from the optimum grid location. One needs to think carefully whether it makes dynamical sense to skip the $l = 1$ terms or to have a second rigid component in the system. *Note that forces from a second rigid rigid mass component are not included in the acceleration acting on the perturber.*

### 12.1.1. *Generic case*

A new feature, added to the code in v14.1, allows the user to supply the attraction of any arbitrary external mass via their own customized code that can be added to the acceleration experienced by all the particles. The reaction forces from those added to the particles are summed and are used to accelerate the center of mass (CoM) of the external mass, so that the system conserves momentum and energy.

The user should copy subroutine `ptrbac.f` from the `lib15` directory, and edit the last few lines of code to provide the desired accelerations and potential. This is a follows:

```
      subroutine ptrbac( is, ac, old )
```

...

```
c do not change any lines above here
c
c the following lines should calculate the x-, y-, and z-components of the acceleration from the rigid mass
c that must be added to the acceleration of the particle. The particle is at position ( xd( i ), i = 1, 3 )
c and lies at distance d relative to the center of the rigid mass.
c The potential, phi, at this position is needed only so that the code can check energy conservation
c
c you may wish to change these expressions, which are for a point mass
c Cartesian acceleration components
      do i = 1, 3
         ac( i ) = -mptbr * xd( i ) / d**3
      end do
c potential
      ac( 4 ) = -mptbr / d
```

The perturber mass, and CoM position and accelerations should be in the system of units described in §2.3 (*i.e.* $G = M = \ell = 1$). The expressions for the three Cartesian components of the acceleration can be as complicated as desired. *e.g.* the system represented by the particles could be in orbit around some larger mass such as a galaxy, in which case the forces need not be directed along the line `xd` from the center of mass to the particle.

This edited version of `ptrbac.f` should be compiled with the source code for `galaxy` and linked with the other libraries to create the modified executable.

To ensure that these forces are added to those acting on each particle, the `.dat` file has some extra lines as follows:

```
A 1   run no          1000
A 2   grid type        s3d
      #
      # s3d grid
      #
C 1   grid size        301     110.        # number of radial grid shells (N_r) and the radius of the grid outer boundary
C 2   lmax               8                 # value of l_max
C 3   sect               1                 # order of symmetry imposed
C 4   skip                                 # list of any additional surface harmonics to skip
      #
      ...                                  # other lines of input
D 1   cntd              16       1         # recenter grid at this step interval
      #
D 2   perturber       GNRC   0.125         # pertuber type and mass
D 3   position          5.     0.0   0.0   # initial position of the perturber center
D 4   velocity          0.     0.5   0.    # initial velocity of the perturber
      #
      ...                                  # other lines of input
      #
E 1   end time        200.0
```

The perturber mass, and CoM position and velocities are in the system of units described in §2.3 (*i.e.* $G = M = \ell = 1$).

Line D1 above, which can be anywhere in part D of the file, causes the grid origin to be shifted to the poistion of the particle centroid every 16 steps, which is desirable if the vector sum of forces on the particles from the perturber exerts a net translational force.

### 12.1.2. *Spherical mass*

Code is supplied for the simple case of a rigid mass perturber of any spherical form listed in §16.1.2.

In this case, the first extra line in the `.dat` file specifies the 4-character type, mass *and scale size* of the perturber, and the next two lines give the six phase space coordinates, the initial position and velocity, of the perturber relative to the frame defined by the main particles. Naturally, entering zeros for all these values will place the perturber at rest in the center of the grid. An example of the extra lines is as follows:

```
A 1   run no          1000
A 2   grid type        s3d
      #
      # s3d grid
      #
C 1   grid size        301     110.        # number of radial grid shells (N_r) and the radius of the grid outer boundary
C 2   lmax               8                 # value of l_max
C 3   sect               1                 # order of symmetry imposed
C 4   skip                                 # list of any additional surface harmonics to skip
      #
      ...                                  # other lines of input
      #
D 1   perturber       Plum   0.125   0.1   # pertuber type, mass, and scale size
D 2   position          5.     0.0   0.0   # initial position of the perturber center
D 3   velocity          0.     0.5   0.    # initial velocity of the perturber
D 4   cntd              16       1         # recenter grid at this step interval
      #
      ...                                  # other lines of input
      #
E 1   end time        200.0
```

The perturber mass, size, and velocities are in the system of units described in §2.3 (*i.e.* $G = M = \ell = 1$).

### 12.1.3. *Bar or spiral perturbations*

It is sometimes useful, especially when checking against theoretical predictions, to compute the dynamical friction between a halo and a rigid bar (*e.g.* Sellwood 2006). Such a bar can also be introduced through extra lines in the `.dat` file. The perturber type is a homogeneous, tri-axial bar, with the size of the long axis, the axis ratio, and the masses all specified. It is assumed to spin about its shortest axis at the specified initial rate.

The gravitational attraction of the bar is added to the grid-determined force acting on each particle before it is accelerated, and the torque acting on the bar computed from its reaction to the forces it exerts on the particles. The bar rotation is decelerated assuming its moment of inertia is that of the ellipsoidal bar.

In the bar case, the `.dat` file has one extra line as follows:

```
A  1    run no          1000
A  2    grid type        s3d
        #
        # s3d grid
        #
C  1    grid size        301     110.                    # N_r and r_grid
C  2    lmax               8                             # value of l_max
C  3    sect               1                             # order of symmetry imposed
C  4    skip               1                             # list of any additional surface harmonics to skip
        #
        ...                                              # other lines of input
        #
D  1    perturber        bar   0.125   1.0   0.5, 0.2, 0.1  # pertuber type, mass, sma, pattern speed and b : a & c : a
        #
        ...                                              # other lines of input
        #
E  1    end time       200.0
```

The bar is a homogeneous 3-axial ellipsoid with mass, size, and pattern speed given in the system of units described in §2.3 (*i.e.* $G = M = \ell = 1$).

When conducting experiments with rigid bars that rotate about a fixed center, it is important to prevent the system from becoming lop-sided, which can give rise to a large and unphysical torque (Sellwood 2003; McMillan & Dehnen 2005). This can be accomplished simply by skipping the $l = 1$ terms in the determination of the field from the particles as in the above example (see §5.3).

### 12.2. **Two independently moving grids**

Interacting pairs of galaxies are better modeled as two self-consistent, live stellar systems. Scaling them both down in size so that the two systems begin and remain on a single grid leads to poor resolution for at least one, usually both, members of the pair whatever type of grid is employed. A better solution is to employ two grids, each centered on the densest point of two separate systems. In order that the particles in each galaxy feel the attraction of those in the other, at least one grid must encompass the combined mass distribution. The centers of the two grids must move with the location of the centroid of the particles assigned to that grid.

When employing two spherical grids, with logarithmic spacing of the grid shells, the centers of each system are well resolved and the forces from the particles in the other galaxy are included. Note that the low spatial resolution of the forces from the other galaxy is of no concern when the two galaxies are widely separated, and resolution improves as they move closer together and eventually merge.

Unfortunately, the results from trying this were unsatisfactory. Total angular momentum was not well-conserved, and changes varied with numerical parameters, such as the value of $l_{\rm max}$ and the number of grid shells, with no sign of convergence to acceptable behavior. The problem stemmed from force errors arising from the distorted mass distribution of the satellite; the attraction of tidal streams of particles have large errors when computed by the s3d method – see §8.10.2.

### 12.3. **Hybrid coincident grids**

The 3D cylindrical polar grid is ideally suited to modeling the rotationally supported disk of a spiral galaxy, but the fixed vertical spacing of the grid planes makes it less well suited to include a substantial classical bulge, and an extensive DM halo could not reasonably be included as a live component. Spherical grids, on the other hand, are ideal for ellipsoidal galaxy models, but are not well suited to galaxy disks.

I have therefore implemented a hybrid procedure for modeling composite disk-halo galaxies. The field of the disk particles is computed using the 3D polar grid, while the bulge and halo particles are assigned to a concentric, and much more extensive, spherical grid. Advancing the motion of the disk particles is straightforward, since the field of the bulge and halo particles is available throughout the volume where both grids overlap and the total acceleration on the $i$-th particle is the sum of the accelerations computed on the two separate grids

$$\boldsymbol{a}_d(\boldsymbol{x}_i) = \boldsymbol{a}_1(\boldsymbol{x}_i) + \boldsymbol{a}_2(\boldsymbol{x}_i), \tag{117}$$

where $\boldsymbol{a}_1$ is computed from the disk particles using grid 1, and $\boldsymbol{a}_2$ is computed from the bulge/halo particles using grid 2. If the grid centers are allowed to move, they are forced to remain concentric.

In order to capture forces from the disk particles acting on the halo, I recompute the forces from the disk particles on a second spherical grid and the bulge and/or halo particles are advanced in that combined field

$$\boldsymbol{a}_h(\boldsymbol{x}_i) = \boldsymbol{a}_2(\boldsymbol{x}_i) + \boldsymbol{a}_{2'}(\boldsymbol{x}_i), \tag{118}$$

where $\boldsymbol{a}_{2'}$ is computed from the disk particles using grid $2'$. (This second spherical grid is simply a replica of the first, and should not be specified as a third grid in the `.dat` file.) Since the two acceleration terms are computed using eq. (83) at the same field position on two identical grids, it is convenient to sum the separate $\mathcal{Q}_{lm}$ coefficients from each grid and evaluate the combined acceleration from a single evaluation of eq. (83).

Any disk particles that spill outside the 3D polar grid are relabeled as belonging to the spherical grid, although they continue to carry a disk-particle flag. These, generally rather few, particles are reassigned back to the polar grid when their orbits re-enter that volume.

This option is selected by the grid keyword "hyb" with an integer parameter to indicate the number of grids. This is followed by the keyword and parameters of the two grids separately. In this case, the extra lines in the `.dat` file are as follows:

```
A  1    run no          1000
A  2    grid type        hyb      2
A  3    1st type         p3d
A  4    2nd type         s3d
        #
        ...                                  # information describing each component of the composite galaxy
        #
        # p3d grid
        #
C  1    grid size          90    128   125   # mesh dimensions (N_R, N_a, N_z)
C  2    z spacing         0.1                # in grid units
C  3    HASH                8                # components active
C  4    softl             0.5      2        # softening length in grid units and softening rule
C  5    sect                1                # order of rotational symmetry imposed
C  6    skip                                 # list of any additional sectoral harmonics to skip
        #
        # s3d grid
        #
C  7    grid size         301    110.        # number of radial grid shells (N_r) and the radius of the grid outer boundary
C  8    lmax                8                # value of l_max
        #          the "sect" and "skip" lines should be omitted here – the values for the 1st grid are used for both
        #
        ...                                  # other lines of input
        #
D13     pgrd                1                # the grid to be used for this particle population
        #
        ...                                  # other lines of input
        #
E  1    end time        200.0
```

One instance of line D13 is required for each live particle population, and the value 1 selects the first grid, 2 the second grid, *etc.*, with the numbering being the order in which they were named in lines A3, A4, *etc.*

While the only applications (Sellwood 2003; Shen & Sellwood 2006; Sellwood & Debattista 2006) have used the 3D polar and spherical grids, it should possible to employ any pair of grids – although a user entering this uncharted territory will need extra vigilance to check that the results remain reasonable.

For interacting disk galaxies, it might be possible to use four grids, two for each component of each of the two galaxies. This has never been tried and extra code would need to be written to orient the polar grids when the disks of the two galaxies are not coplanar.

### 12.4. **Heavy particles**

Another option is to add a set of heavy, softened particles and have them attract, and be attracted by, the light particles on the grid through direct summation of forces. The grid-determined acceleration of a light particle at position $\boldsymbol{x}$ is augmented by the attraction of $N_h$ heavy, softened point masses at positions $\boldsymbol{x}'$:

$$\ddot{\boldsymbol{x}} = -G \sum_{N_h} \mu_h \boldsymbol{S}(\boldsymbol{x}' - \boldsymbol{x}), \tag{119}$$

where $M_h$ is the mass of each heavy particle and $\boldsymbol{S}(\xi)$ is given by eq. (9). The acceleration of each heavy particle is

$$\ddot{\boldsymbol{x}}' = -G \sum_{N_l} \mu_l \boldsymbol{S}(\boldsymbol{x} - \boldsymbol{x}'), \tag{120}$$

where the summation is over all $N_l$ light particles whose masses are $\mu_l$, plus the forces from all other heavy particles as given by eq. (119). This strategy is practicable as long as $N_h$ is not too large (Jardel & Sellwood 2009).

The `.dat` file should have the extra lines:

```
A  1   run no          1000
A  2   grid type        two      2
A  3   1st type         c3d
A  4   2nd type         dr3d
       #
       ...                                  # information describing the spatial distribution and total mass of each
       ...                                  # component, including the component represented by the heavy particles
       #
       # c3d grid
       #
C  1   grid size        257      257    257  # mesh dimensions (N_R, N_a, N_z)
C  2   shape            1.0      1.0    1.0  # cubic cells
       #
       # dr3d
       #
C  3   softl            1.0        2        # softening length and softening rule
       #
       ...                                  # other lines of input
       #
D13    pgrd             2                   # the "grid" to be used for the heavy particle population
       #
       ...                                  # other lines of input
       #
E  1   end time       200.0
```

The analysis software routines fall into two classes: Routines to extract information from the particles at a given step in the evolution of the run and save them into a "results" file (`.res`), and programs that process the information that has previously been saved in the results file. Some details about how this file is structured and created are given in §13.4.

## 13.1. **On-the-fly analysis options**

The use of leap frog (§11.1) implies that particle cordinates are time-centered and the positions and velocities are stored at times that are half a time step apart. But analysis of the physical properties of the system should be completed at a fixed time, which requires that it be completed as the particles are advanced when values of the velocities can be interpolated from their before and after velocities.

Block time steps (see 11.2) also dictate that a full analysis of the simulation can be completed only at intervals that are integral multiples of the basic time step.

When the analysis flag is raised, `step` begins by initializing the arrays used for analysis in a first call to `measure`. It then ensures that every group of particles contributes to the analysis by a call to `anlgrp`, and finishes up by outputting the results in a second call to `measure`.

The selection of any of the following forms of analysis is optional.

### 13.1.1. *Analyses that must be completed before advancing any particles*

1. **Density 1**: keywords `dnst` and/or `danl`. The density assigned to the computational grid, and possibly its decomposition in sectoral harmonics. This involves little more than writing out the appropriate numbers from the grid and therefore includes all particles assigned to that grid. This analysis is performed before the gravitational field is determined, since that calculation generally overwrites the density array.

2. **Density 2**: keyword `rhor`. The radial variation of the density of each separate mass component estimated from the particles. This is an altogether more powerful routine, that determines the radius (cylindrical or spherical) of each particle and sorts them. Then using a stride through the sorted radii, it saves the radius of every $n$-th particle and either the volume density for spherical components, or vertically integrated surface density for disk components, between the radius of this particle and that of the previously selected particle, and writes out this information.

3. **Frequencies**: keyword `frqs`. The values of $\Omega_c$, $\kappa$, and $\nu$ in the midplane at radial intervals $\delta R = \ell/n$, with $n$ being the input parameter in the `.dat` file. These frequencies are computed from

$$\Omega_c^2 = -\frac{|a_R|}{R}, \qquad \kappa^2 = 4\Omega_c^2 + R\frac{d\Omega_c^2}{dR}, \quad \text{and} \quad \nu^2 = -\frac{d|a_z|}{dz}. \tag{121}$$

   In these expressions, $|a_R|$ and $|a_z|$ are azimuthal averages of respectively the radial and vertical forces experienced by rings of test particles (*i.e.* including any external fields) in the disk mid-plane, and therefore reflect the instantaneous combined attraction of all the mass components.

4. **Moment-of-inertia tensor**: keyword `moit`. This routine sorts the particles of each spheroidal component by their binding energies and divides them into groups of equal numbers of particles, and computes and saves the MoI tensor

$$I_{jkl} = \sum x_j x_k / N_l, \tag{122}$$

   where, $x_j$ and $x_k$ are the $j$ and $k$ coordinates of the $i$-th particle, and $N_l$ in the number of particles in each energy group. Shouldn't this be just $\sum \mu_i x_j x_k$?

### 13.1.2. *Analyses that require accumulation during processing*

5. **Global integrals**: keyword `intg`. This accumulates the total energy and virial of Clausius of the entire model, plus a tally of all the particles that are outside the largest grid. It also saves the kinetic energy, potential energy, center of mass, linear momentum, acceleration, and angular momentum separately for each separate population of particles. These quantities are defined as follows:

   The total energy, $E_T = T + W$, of a collisionless simulation should be a conserved quantity. Here

$$T = \tfrac{1}{2}\textstyle\sum_N \mu_i |\boldsymbol{v}_i|^2 \qquad \text{and} \qquad W = \tfrac{1}{2}\textstyle\sum_N \mu_i \Phi(\boldsymbol{x}_i), \tag{123}$$

with $\Phi(\boldsymbol{x}_i) = \sum_N \mu_j P(\xi_{ij})$ (eq. 4) and the factor $\frac{1}{2}$ in the $W$ term arises because the double summation includes every particle pair twice. In principle, the $W$ term should exclude the particle self-energies when $i = j$, which are finite, but vary with sub-grid position. However, inclusion of the self-energy terms has a negligible effect for large $N$, but they need to be subtracted for the binary test (§14.1.3), for example. With a rigid external field, we must include an extra term for each particle, plus (optionally because it is an unchanging additive constant) the self-energy, $W_{\rm self}$, of the rigid component

$$W = W_{\rm self} + \sum_N \mu_i \Phi_{\rm ext}(\boldsymbol{x}_i) + \tfrac{1}{2}\textstyle\sum_N \mu_i \Phi(\boldsymbol{x}_i). \tag{124}$$

On a single grid, the value of $\Phi(\boldsymbol{x}_i)$ may be computed by any of the methods described in §8. However, it should be borne in mind that accelerations computed on a grid are not precisely the derivatives of the potential function $\Phi(\boldsymbol{x}_i)$, as described in §10. This implies that $E_T$ defined above cannot be perfectly conserved, although fluctuations generally average to little change, typically $< 0.1\%$, as long as the mean distances between particles do not change. But the formation of a bar or an overall contraction or expansion can lead to larger changes. (See also footnote[13].)

With two grids, each having mobile particles, the potential of the disk and halo particles on the two separate grids is, as for eq. (118)

$$\Phi_d(\boldsymbol{x}_i) = \Phi_1(\boldsymbol{x}_i) + \Phi_2(\boldsymbol{x}_i) \qquad \text{and} \qquad \Phi_h(\boldsymbol{x}_i) = \Phi_2(\boldsymbol{x}_i) + \Phi_{2'}(\boldsymbol{x}_i). \tag{125}$$

These expressions imply that the interaction potentials between the particles in the two separate components are computed on different grids, so that they are not necessarily equal, but again this technical difference causes negligible changes to $E_T$ in quasi-equilibrium models.

The scalar virial theorem implies that $2T + W = 0$ for an equilibrium system of particles interacting with inverse-square law forces, which affords a useful check for a supposedly equilibrium initial model. However, $W \neq -T/2$ for an equilibrium $N$-body simulation in which inter-particle forces are softened at short range, either explicitly through a softening kernel (§9) or implicitly through finite size particles on a grid (§8.1.2). However, an equilibrium system that interacts with any arbitrary force law satisfies $2T + W_C = 0$, where the virial of Clausius (Goldstein 1950, eq. 3-26) is

$$W_C = \sum_N \mu_i \boldsymbol{a}_i \cdot \boldsymbol{x}_i. \tag{126}$$

Note that the acceleration here includes not only that arising from the other particles, but also any external field. It is easy to show that $W_C \equiv W$ when forces vary precisely as an inverse-square law.

Other global integrals are accumulated separately for each active population of particles:

$$\begin{aligned} \text{linear momentum components:} &\quad \textstyle\sum \mu_i \boldsymbol{v}_i \\ \text{angular momentum components:} &\quad \textstyle\sum \mu_i \boldsymbol{r}_i \times \boldsymbol{v}_i \\ \text{net acceleration components:} &\quad \textstyle\sum \mu_i \boldsymbol{a}_i \\ \text{center of mass:} &\quad \textstyle\sum \mu_i \boldsymbol{x}_i / \sum \mu_i \end{aligned} \tag{127}$$

6. **Snapshot information**: keyword `plot`. The positions and velocities of a representative subset of the particles in each population that can be used to create dot plots, *etc.*

After these two pieces of analysis, the time-centered positions of the particles are shifted to a frame whose origin coincides with the center of the appropriate grid. It does not at the present time, subtract off the bulk motion of the mass component, which can be confusing.

7. **Velocity field analysis**: keywords `vfld` and/or `vflh`. Computes the mass-weighted means and dispersions of each component of the particle velocities in a set of spatial bins $\{k, l\}$ via:

$$m(k,l,t) = \sum_j \mu_j, \qquad a_i(k,l,t) = \sum_j \mu_j v_{i,j}, \qquad \text{and} \qquad b_i(k,l,t) = \sum_j \mu_j v_{i,j}^2, \tag{128}$$

where $\mu_j$ and $v_{i,j}$ are the mass and $i$th velocity component, in cylindrical polar coordinates, of the $j$th particle at time $t$. From these summations, we have

$$\langle v_i \rangle(k,l,t) = \frac{a_i(k,l,t)}{m(k,l,t)} \qquad \text{and} \qquad \sigma_i(k,l,t) = \left[ \frac{b_i(k,l,t)}{m(k,l,t)} - \langle v_i \rangle^2 \right]^{1/2}. \tag{129}$$

For particles in disk populations, the bins are evenly spaced in $\Delta R$ and $\Delta\phi = 2\pi/n_a$, with no bounds on $z$, while for spheroidal populations, the bins are evenly spaced in $\Delta R$ and $\Delta z$, with no subdivision in $\phi$.

8. **Logarithmic spiral analysis**: keyword `lgsp`. Forms the transform

$$A(m,\alpha,t) = \frac{\sum_j \mu_j \exp[im(\phi_j + \cot\alpha \ln R_j)]}{\sum_j \mu_j}, \tag{130}$$

where $(R_j, \phi_j)$ are the cylindrical polar coordinates of the $j$-th particle at time $t$, which has mass $\mu_j$, and $\alpha$ is the pitch-angle of an $m$-arm logarithmic spiral, with positive values for trailing spirals. This analysis is performed and saved separately for each disk population.

9. **Bending distortions of the disk**: keyword `zbnd`. Forms the transform

$$Z(R_k,m,t) = \frac{\sum_j \mu_j z_j e^{im\phi_j}}{\sum_j \mu_j}, \qquad \text{for} \qquad R_k < R_j < R_{k+1}, \tag{131}$$

where $(R_j, \phi_j, z_j)$ are the cylindrical polar coordinates of the $j$-th particle at time $t$, and $\{R_k\}$ define a set of radial bins. This analysis is performed and saved separately for each disk population.

10. **Spherical Bessel function**: keyword `sphb`. Forms the transform

$$B(n,l,m,t) = \frac{\sum_j \mu_j j_n(r_j/r_{\rm B}) P_{l,m}(\theta_j,\phi_j)}{\sum_j \mu_j}, \tag{132}$$

where $j_n$ is a spherical Bessel function of order $n$, $P_{l,m}$ are the usual surface harmonics, $(r_j, \theta_j, \phi_j)$ are the spherical polar coordinates of the $j$-th particle at time $t$, and $r_{\rm B}$ is a limiting outer radius so that the argument of the Bessel function never exceeds unity. This analysis is performed and saved separately for each spheroidal population.

11. **z-density profile of disk**: keyword `zprf`. Determines the mean density from

$$\rho(R_k,z_l,t) = \frac{\sum_j \mu_j}{\pi(R_k^2 - R_{k-1}^2)\Delta z}, \qquad \text{for} \qquad R_k < R_j < R_{k+1}, \quad z_{l-1} < z_j < z_l, \tag{133}$$

where $(R_j, z_j)$ are the cylindrical polar coordinates of the $j$-th particle at time $t$, and $\{R_k\}$, $\{z_l\}$ define a set of bins that are evenly spaced by $\Delta R$ and $\Delta z$. This analysis is performed and saved separately for each disk population.

12. **Rings of test particles**: keyword `rngs`. Saves the positions, velocities and energies of a population of test particles initially spaced uniformly around rings and started on circular orbits.

Once these analysis routines have been completed for all the particles, the plottting buffer is emptied by a final call to `disply` and results from the other analyses are processed and saved to the `.res` file by `phyprp`.

## 13.2. **Adding new analysis options**

A user can write additional code to output results from different forms of analysis, if desired.

The new form of analysis could be undertaken prior to each step, provided it does not depend on the particle velocities or require the potential. If it needs either or both these quantities, then it will need to be embedded within the analysis that is performed "on the fly" as the particles are processed within a time step. The simplest thing to do will be to add a call to the user's new analysis subroutine from routine `anlgrp` which is called for every group of particles during an analysis step. The accumulated results will have to be processed (if needed) in subroutine `measure` and then output afterwards. `measure` is also called at the start of an analysis step, and can be used to initialize an array in which the results will be accumulated.

Of course the output could be directed to a new file that differs from any other currently in use, but it could also be added into the current output stream, the `.res` file, as follows.

The user will need first to choose a 4 letter keyword to flag the data, that differs from others in use. This should be appended to the list in subroutine `datnam` after increasing by one the parameter `mtype` that is in the include file `params.f`.[15] Then the output to the `.res` file at each time the analysis is performed should be in the form of two records of binary data:

- A record header that contains just 5 values:
  `irun, bhol, istep, i1, i2`
  where `irun` and `istep` are variables in the common block `/admin/`, `bhol` is 4 charater keyword chosen by the user that is written as a Hollerith string (see examples elsewhere in the code). The last two variables are usually integers that can be used to deduce the length of the next data record.

[15] Whenever a change is made to any of the include files in the `lib15/inc/` subroutine directory, it is essential to rebuild the libraries and to rerun make, as described in §3.

- A single record that can be of any (reasonable) length containing the values from the analysis. This is typically an array of values, with `i1` and `i2` from the previous record specifying the array dimensions, for example.

To integrate the analysis more fully and to enable it to be controlled from the `.dat` file, subroutine `setrun` will need to be edited. Again it will require a new keyword for input (that does not have to be the same as the keyword for output) together with 1 or 2 values that will control the analysis.

The post-run program `analys` will then need to be updated to recognize the data type that has been added to the `.res` file. This will necessitate editing `decode` that identifies the keyword read from the record header and uses the two integers (now `mr` and `ma`) to set the calling argument `nw` to be the number of (real*4) words of data expected in the next record. Finally, the user will have to supply a subroutine that uses the data that can now be read to create whatever kind of plot is desired.

## 13.3. **Displaying the results**

The user may display results using the interactive program `analys` that was described above (§7). Not only may this program be used after the simulation has completed, but it can also be used at any time while `galaxy` is running; indeed it is recommended that the user runs `analys` at an early stage, in order to verify that the initial set up was successful.

The program `analys` has many options which the user may select via input from the terminal, and can display results either in an *XTERM* window or in a *PostScript* image. The latter outputs are useful for publications, although some editing of the analysis code is generally required to create precisely the information the user wishes to publish.

`Analys` may also be used to compare the results from two or more simulations, although it is unable to do so for every option, especially where overlaying information even in different colors would result too confusing a plot.

### 13.3.1. *Mode fitting*

Program `modefit` was described in Sellwood & Athanassoula (1986). A **mode** is a standing wave oscillation that has a fixed shape and frequency, which is a steady rotation in the case of non-axisymmetric modes. Neutrally stable modes have a fixed amplitude, but the amplitude of growing modes rises exponentially in the linear regime. The program attempts to fit waves having these properties to output of a given type from the simulation. The most useful data to fit are some kind of global transform of the particle positions, such as logarithmic spirals, amplitudes and phases of the sectoral harmonics on grid rings, spherical Bessel functions, *etc.*

In this interactive program, the user must first select the fraction of the available data to be fitted. A growing mode generally emerges from noise and manifests exponential growth for a period before saturating. The program can display the part of the data selected, and a good fit to the unstable mode is obtained if there is an extended period after rapid growth has begun and before non-linear effects cause the growth rate to slow. As this is a least squares fit, it can be advantageous to reduce at least part of the rising amplitude by rescaling the data with an exponentially decreasing function of time to allow the low-amplitude initial data to contribute to the fit with more equal weight.

The program may not find the global minimum if the initial guess of the mode frequency supplied by the user is not close to the actual frequency. A good initial guess for the real part of the frequency can be found from the *spct* option of program `analys`, while having a good initial guess for the growth rate is less critical. The program reports the fraction of the selected data that were *not* fitted by the wave(s) at the position of the minimum; for a good fit, this number may be a few percent, whereas a value above 90% indicates that the fitter has failed to identify a coherent disturbance in the data, which could occur either if there is none, or that the initially guessed frequency led to a local minimum far from the global minimum.

Once a fit is found, there are options to display the data, fitted mode, and residuals, as well as to draw the fitted mode. It is usually worth saving the result from each successful fit into a direct access file called `mbase` located in the directory one higher than the current directory; results from many different simulations can be saved into the same file.

Sometimes the simulation will manifest more than a single mode, which is usually obvious when displaying the residuals. The user can then choose to fit as many additional modes (up to five in total) as the data appear to warrant, although cases where more than two *credible* modes can be fitted are extremely rare.

Uncertainties in the fitted frequencies are best estimated from examination of multiple fits in which user input parameters, such as the time range, scale factor, number of modes, type of data fitted, *etc.*, are varied. All fits saved to the direct-access file `mbase` contain all this information and the fitted frequencies from a single run can be displayed using the program `estfreq`.

An example script for running `modefit` non-interactively is supplied in subdirectory `EXAMPLES/p2d` of the download package.

## 13.4. **Creation of the `.res` file**

Program `galaxy` generally appends new data to an existing `.res` file; only if the file does not exist will it create a new `.res` file and start writing information to it.

Program `galaxy` always closes the `.res` file after each analysis step, to ensure that the output buffer with the results of the completed analysis step is emptied to disk. The file is reopened by `galaxy` in append mode in order to be ready to receive subsequent analysis records. The main reason this is done is to minimize the possible data loss during a system crash, or whatever, but it also enables the latest results from the evolution to be examined by running `analys`.

### 13.4.1. *Structure of the `.res` file*

The binary data in the `.res` file consist of a few header records followed by an unlimited number of data records. The header records contain information about the original mass model and numerical parameters of the run. Preceding each record of actual data, there is a short "record header" that has the following structure

1. an integer giving either the mass component to which the next data relate or, if they are not specific to a component, the run number,
2. an integer giving the time step at which the data were saved,
3. a 4-byte Hollerith string, which is a key word to indicate the type of the next record of data,
4. and 5: two integers, which together with the key word, describe the arrangement of values in the following data record(s).

### 13.4.2. *Tidying up a `.res` file*

System crashes or user errors can sometimes result in a `.res` file that contains duplicate data, which may mess up plotting, *etc.* Also, the user may have requested analysis steps too frequently, or saved data of types that are of no relevance to the science, but make the file excessively large. Both these situations can be remedied without rerunning the simulation by use of the programs `merge` and `weed`.

If the user is concerned that the `.res` file may not have been closed properly after a crash, then it is vital to use program `merge` to clean up the file as described next (§13.4.3) *before* resuming the evolution. If this is not done, there will be no (easy) way to recover the good data that was appended to the file after a corrupted record.

Note that neither of these programs should be run on a file that is being added to by a current run of `galaxy`. If the user wishes to continue the evolution after manipulating the `.res` file using either of these programs, `galaxy` should first be stopped as described in §6.2.

### 13.4.3. *Running* `merge`

Program `merge` is designed to remove repeated data from a `.res` file, and it will also copy useable data from a file that has been corrupted by, say, a system crash before a write had completed. It also has a less useful option of creating an ASCII summary of the global integrals at each analysis step on a file with extension `.sum`.

Program `merge` should be run interactively. It first asks the user to input the run number, and then give a y/n answer to whether to produce a summary (the above mentioned `.sum`) file. It then copies the records of the `.res` file one at a time to a new file in the same directory extension that has `.cpy`. It skips a repeat occurrence of any record encountered in the `.res` file that matches the type and time step of one previously copied, and reports to the screen that it has done so. It also stops gracefully, properly closing the output file `.cpy`, if a record in the `.res` file is unreadable. **Warning:** data that may have been appended to the file after the corrupted record will not be copied to `.cpy`.

Thus the size of the output file `.cpy` can be less than that of the input `.res` file; if the files are the same size then no issues occurred as `merge` copied the file. If the user understands the reason for any differences, and all the expected data have been copied, she/he can replace the old `.res` file with the new `.cpy` and, if desired, `galaxy` can be restarted to continue the evolution, which will append additional records to the now cleaned `.res` file.

### 13.4.4. *Running* `weed`

Program `weed` functions in a similar manner, but has a different purpose. Again it should be run interactively.

After identifying the run number, the header of the `.res` file is copied to `.cpy`, and then the user is invited to enter the new desired number of time steps between analysis steps; a value of zero, or of the original analysis interval will ensure

that all the desired data are to be copied. Since the `pnts` data for pictures is large, the user has the options to save these data on a different time step stride from the other data types.

Then, at the first ocurrence of each data type as the `.res` file is being read, the user is asked whether that type of data is to be copied. Answering "no" will result in a `.cpy` that contains no data of that type. The program can be rerun as often as desired, with new versions of `.cpy` overwriting the old, until the last run results in an output file that contains only and all the information the user wishes. Once the user is happy with the new `.cpy` file, it can replace the original `.res` file, but once that is done, there is no going back to the original, and the simulation will have to be rerun if valuable data were discarded.

Tests of $N$-body codes take a variety of forms that are designed to test different aspects of the code. Single particle, or binary particle tests are very strong tests for programming blunders, but give no indication of the performance of the code on the kinds of problems it is designed to solve, namely the evolution of stellar systems with smooth density profiles. A completely different kind of test is required for an end-to-end test.

## 14.1. **Tests of individual pieces of the code**

### 14.1.1. *The solution for the field*

The solution for the gravitational field by FFTs on a grid is a sufficiently complex piece of code that it is desirable to have a stand-alone test of that part of the code. Program `ptest` introduces a number (between 1 and 10) of unit masses at locations specified by the user onto an otherwise empty grid. It then proceeds to solve for the gravitational field on the grid by the selected FFT method and then compares the values of the potential, and acceleration components at every grid point with those obtained by direct summation using the adopted kernel. This is a very strong test for progamming blunders, since the results should agree to machine precision.

The test can still be made when the expansion in sectoral harmonics is truncated in polar grid codes, since `ptest` computes the smoothed interaction kernel, which is saved in file `.grd` (the only place where this file type is used). It assumes that all sectoral harmonics up to the value given by the parameter HASH will be used, and erroneous results will be obtained if any of these are "skipped". The package comes with sample input `.dat` files for each grid type in sub-directories `EXAMPLES/*/ftest`.

### 14.1.2. *Force quality*

The program `isotropy` checks the inter-particle forces derived from the grid in a different manner, and results were presented in §10. The package comes with sample input `.dat` files for each grid type in sub-directory `EXAMPLES/isotropy`.

The program `isotropy` computes the attraction experienced by rings (in 2D) or shells (in 3D) of test particles surrounding a single source particle placed at an abritrary point on the grid. Working over a set of radii for the rings or shells, the program computes and saves the mean and variance of the attractions experienced by the test particles, both toward the test particle and perpendicular to that direction (which should be zero, but forces between particles on grids may not be perfectly along the line centers). The source point is then moved to a new position, and the exercise repeated as often as desired. On Cartesian grids, the source point need be moved only within one grid cell. The program then plots the combined mean and variance of the two acceleration components as functions of distance from the source particle.

This test yields information on how well the code hides the existence of the grid. The existence of grid-dependent forces is clearly undesirable, and adds numerical jitter (aka grid noise) to the motion of particles. It also tests the mass assignment and force interpolation parts of the code.

The application of this test to polar grids requires some discussion. First, it is essentially a test of the attraction of a point mass, which is not the field that results when some sectoral harmonics are eliminated; the alarming variances that would be obtained in this case have no significance. When all sectoral harmonics allowed by the grid are included, the source particle should be confined to the inner grid where the cell sizes are smaller than the softening length $\epsilon$. Exploring the outer grid, where this is no longer true, will result in increased variances. Since polar grids are not intended to yield accurate estimates of the attractions of point masses, but the smooth field of a continuous mass distribution, the user should not be alarmed by the results of such a test. The variance arises not from harmful grid noise, but from the beneficial smoothing.

### 14.1.3. *A binary test*

Another very strong test is to compute the motion of just two particles that are gravitationally bound to each other, since the motion can be cross-checked against a direct calculation. This is achieved in program `binary`; sample input `.dat` files for each grid type can be found in sub-directory `EXAMPLES/binary`.

This test not only verifies the force determination method, but also provides a strong test of the interpolation of masses and accelerations from the grid points, and the integration algorithm. Shortcomings, such as non-conservation of momentum or energy, would stand out clearly, even when the time step is short. Since it tests for the attraction of point masses, the test is useful only on polar and Cartesian grids, and requires the force determination on a polar grid to employ all possible sectoral harmonics (*i.e.* $m_{\max} = N_a/2 + 1$). Since the self-energy of a particle varies with its sub-cell position, energy conservation with just two particles is acceptable only if the self-energy of each particle is subtracted; thus the program includes a non-standard version of `grdacc` to subtract this large, non-constant, contribution to the total energy.

When s3d only (run341) or a hybid grid (run401) are used, the motion is integrated acceptably well except when the particles cross in radius. The force errors associated with these "shell crossings" change the orbit somewhat. Since the relative error in the total force decreases linearly with the mass of each particle, it is negligible in a simulation with many particles.

## 14.2. **End-to-end tests**

### 14.2.1. *Equilibria*

The selection of sets of particles to represent simple mass models with known DFs is described in §15. These particle sets should be in equilibrium (if the DF was derived from the self-consistent potential), and therefore afford good tests – but see also §15.10.

### 14.2.2. *Relaxation tests*

We can seek evidence for collisional relaxation through signs of energy exchange between particles of differing masses (*e.g.* Sellwood 2013a, 2015).

### 14.2.3. *Common sense precautions*

Aside from running the above tests, the user of the code is strongly advised to check that important results are (1) physically reasonable, (2) continue to be obtained when grid resolution, time steps, particle numbers, *etc.* are varied by reasonable factors. It may even be desirable to compare with a case in which the field was determined by a different method.

While other checks, such as conservation of energy and momentum (§13.1.2) are worth making, such global properties are a rather blunt diagnostic of possible inaccuracies.

### 14.2.4. *A linear mode*

By far the most powerful test of the code is to show that it can reproduce behavior predicted analytically, or by another method (*e.g.* Inagaki *et al.* 1984). Of course, the principal use of the code is to compute evolution of systems that cannot be calculated analytically because the number of systems whose behavior can be predicted is very limited. However, it is possible, to calculate normal modes of equilibrium models by linear perturbation theory, and Sellwood (1983), Sellwood & Athanassoula (1986), Earn & Sellwood (1995), Sellwood & Evans (2001) showed that the 2D polar grid can reproduce the predicted dominant normal modes in several cases.

The simplest such case is a cold, Kuzmin-Toomre disk, in which axisymmetric stability is ensured by using a large softening length with the Plummer rule. The predicted eigenfrequency can be reproduced in program `stest` to a precision of about 1% by when employing a quiet start (§4.1.2) with a modest number of particles.

It is possible to try this test on other grids, and the author has achieved satisfactory results with a 2D Cartesian grid as well as the 3D polar grid. Even when the plane containing the particles was tipped 10° to the $z = 0$ plane of the grid, the disk remained coherent, without twisting, while the linear mode grew at the predicted frequency until distortions became non-linear.

### 14.2.5. *Mode of a warm disk*

Earn & Sellwood (1995) showed that the 2D SFP method could reproduce the dominant mode predicted by Kalnajs for an isochrone disk model with random motion (the DF given by Kalnajs 1976b, for index 12). As this is a non-axisymmetic Jeans mode of a razor-thin disk with random motion, it cannot be checked in models that allow 3D motion because the disk will also be subject to vigorous buckling instabilities (see Sellwood 2013b, for a review).

As Kalnajs did not soften the gravitational forces in his linear stability analysis, using his prediction to check a grid code requires two separate corrections.

- The central attraction from the mass distribution is weakened by gravity softening on a polar grid (§9.1), and particles generated from his DF will not be in equlibrium without the supplementary central attraction described in §15.10.
- The disturbance field of the mode as it grows is less strong than expected in linear theory, which reduces the growth rate, in particular (§9.1 again). Thus one needs estimates of the growth rate from a series of the simulations with different values of $\epsilon$, to test whether extrapolation as $\epsilon \to 0$ yields the predicted value.

Note that the measured frequency from simulations with a fixed softening length can also vary with particle selection, as shown in §15.9. Thus an estimate of the mode frequency for each value $\epsilon$ should be made from multiple simulations, making this an expensive check of a grid method.

## 15. CREATING A SUITABLE INITIAL SET OF PARTICLES

### 15.1. **Distribution functions**

The best possible way to create an equilibrium set of particles for a particular component is to select them from a distribution function (hereafter DF) that is a function of the isolating integrals. If integration of the DF over all accessible velocities $\boldsymbol{v}$ in the total potential yields the desired density at every point $\boldsymbol{x}$ in position space, then the model will be in equilibrium. Note that the potential does not have to be the solution of Poisson's equation from the resulting density – it could include contributions from other components. The procedure for selecting particles from a known DF is described in §15.9.

The two classical integrals, $E$ and $L_z$, are adequate for razor-thin disks, but these two integrals for a thickened disk or spheroid imply

$$\frac{\langle v_R^2\rangle}{\langle v_z^2\rangle} = \frac{\int f(E,L_z)v_R^2 dv^3}{\int f(E,L_z)v_z^2 dv^3} \equiv 1, \tag{134}$$

which follows because both $v_R$ and $v_z$ enter equally in $E$. Since $\langle v_R^2\rangle \neq \langle v_z^2\rangle$ in real galaxy disks, we conclude that the DF must depend upon three integrals. Binney (2010) gives a prescription for an approximate 3-integral DF for a simple axisymmetric disk. However, no exact 3-integral DFs, even for axisymmetric mass distributions, such as disks or spheroids, are known partly because the third integral cannot be expressed as a simple function of position and velocity and partly because phase space contains some chaotic regions. I explain below how this limitation can be overcome.

### 15.2. **Finding a distribution function**

BT08 give a few analytic DFs for simple models, and a small number of others can be found in the literature. But in general, one cannot hope to find that someone has previously worked out the DF of an arbitrary model. Other approaches could be adopted, such as linear programming (Schwarzschild 1979; Vasiliev & Athanassoula 2015) or "made-to-measure" (Syer & Tremaine 1996; De Lorenzi *et al.* 2007; Dehnen 2009; Malvido & Sellwood 2015), but these options are not part of the present software package.

Composite systems, that may have a disk, bulge and halo say, present a particular challenge. This package offers options to create good quality equilibria for such situations, including such cases as an initial equilibrium disk, bulge and halo with all three mass components represented by particles. While finding the DF for one or more of the spheroidal components, it is best to keep the disk as a rigid, unresponsive mass distribution, and to realize it with particles and assign their velocities once the bulge and/or halo populations are set up.

Some workers adopt a spherical model for the halo and simply add a second, temporarily rigid, mass component to an equilibrium halo by "slowly" ramping up its mass while allowing the halo orbits to adjust. This is crude and computationally expensive, since an extensive halo has some very long period orbits for which only very slow potential changes could be adiabatic or, if disk growth is hurried, the temporary disequilibrium of the halo is slow to settle. Hernquist (1993) describes how to set approximate equilibrium velocities for the halo particles using the Jeans equations for a given density profile and potential. Other methods have been proposed (Rodionov & Sotnikova 2006; Rodionov *et al.* 2009; Yurin & Springel 2014). However, there are a number of superior numerical approaches.

1. For a spherical halo of a given density profile one can determine an isotropic DF, $f(E)$, from **Eddington's inversion** formula (BT08). Holley-Bockelmann *et al.* (2005) find that even if the total mass distribution includes a disk, the spherically averaged potential yields a halo that is in pretty good equilibrium.
2. Prendergast & Tomer (1970) describe an **iterative approach** to creating an equilibrium model, from a chosen functional form for $f(E,L)$.
   (a) Start from an initial guess for the density
   (b) Solve for its potential, and add any external field such as the potential of a disk and/or bulge
   (c) Integrate the adopted DF over all accessible velocities to determine a revised density at each point
   (d) Complete steps (b) & (c), until the density converges to a steady profile. Convergence is usually acceptable after fewer than 10 iterations.

This method was used by Raha *et al.* (1991), Kuijken & Dubinski (1995) and by Debattista & Sellwood (2000) and is implemented in program `dfiter`, which is described in §15.4. Its advantages are (i) that it works for flattened potentials and (ii) that one can solve for the potential using the exact same numerical method as employed in the $N$-body simulation, so that all numerical issues, such as gravity softening, are automatically allowed for. The disadvantage is that the resulting density profile is not easily predicted, although one can experiment by adjusting the parameters in a trial and error fashion until a reasonable approximation to the desired model is found. The powerful *AGAMA* package (Vasiliev 2019) in which the DF is defined in terms of actions, also uses this philosophy.

3. Young (1980), Wilson (2004), and Sellwood & McGaugh (2005) described an algorithm for **halo compression** that uses the fact that both radial action and angular momentum are conserved during compression. One can start from any spherical halo with an isotropic (ergodic) DF or even an anisotropic DF $f(E, L)$. The method, which is implemented in program `compress` (§15.5), yields the DF of the compressed density, which is generally anisotropic, from which particles can be selected. In fact, this procedure works for multiple live components, and can be used to set up equilibrium bulge and halo in the presence of a disk.

All three methods yield equilibrium DFs for bulge and/or halo components in the presence of other mass components. We describe them in more detail below.

Having determined the equilibrium DF by one of these methods, the particles can be selected as described in §15.9 and used in `mkpcs` as described in §4. Note that it is possible to rescale the halo and disk masses and spatial scales at this point, and `mkpcs` can accomodate this possibility by rescaling the masses, positions, and velocities of the halo particles.

## 15.3. **Eddington inversion**

BT08 write Eddington's formula as (their formula 4-46b, p289)

$$f(\mathcal{E}) = \frac{1}{\sqrt{8}\pi^2}\left[\int_0^{\mathcal{E}} \frac{d^2\rho}{d\Psi^2}\frac{d\Psi}{\sqrt{\mathcal{E}-\Psi}} + \frac{1}{\sqrt{\mathcal{E}}}\left(\frac{d\rho}{d\Psi}\right)_{\Psi=0}\right]. \tag{135}$$

Here

$$\Psi(r) \equiv \Phi_0 - \Phi(r) \qquad \text{and} \qquad \mathcal{E} \equiv \Psi - \tfrac{1}{2}|\boldsymbol{v}^2|, \tag{136}$$

where $\Phi(r)$ is the usual gravitational potential, $\Phi_0$ is chosen such that $f = 0$ when $\mathcal{E} \le 0$, and $\boldsymbol{v}$ is the velocity. The second term in the square brackets above is the boundary term, which is usually small when the halo density profile decreases more rapidly than $\sim r^{-3}$ (see Sellwood, Shen & Li 2019, for how to cope when this is untrue). The main integral has a square-root type singularity at the upper limit, which can be factored out of the integrand for a special purpose integrator, while the remaining integrand can also be a very steep function.

It must be stressed that eq. (135) does not guarantee that $f > 0$ over the entire allowed range of $\mathcal{E}$, and any mass model for which the derived $f < 0$ over any part of the range of $\mathcal{E}$ cannot have an isotropic DF.

The second derivative in the integrand can be manipulated as follows

$$\frac{d^2\rho}{d\Psi^2} = \frac{d}{d\Psi}\left(\frac{d\rho}{dr}\frac{dr}{d\Psi}\right) = \frac{d}{d\Psi}\left(\frac{1}{f_r(r)}\frac{d\rho}{dr}\right), \tag{137}$$

since the central acceleration is $d\Psi/dr = -d\Phi/dr = f_r$ and $r$ is a well-behaved function of $\Psi$. Differentiating a second time, we obtain

$$\frac{d^2\rho}{d\Psi^2} = \frac{1}{f_r}\frac{d}{dr}\left(\frac{1}{f_r}\frac{d\rho}{dr}\right) = \frac{1}{f_r^2}\left[\frac{d^2\rho}{dr^2} - \frac{1}{f_r}\frac{df_r}{dr}\frac{d\rho}{dr}\right]. \tag{138}$$

Further manipulation is possible for the case of single mass component, as described below.

For numerical evaluation, an adaptive integrator that also allows for the square root type singularity at the end of the range must be used. Since very many function evaluations of $f$ are required in order to select particles, it is expedient to tabulate $f$ over the full range of $\mathcal{E}$, so that the function can be estimated much more efficiently by interpolation in the table. While, $f(\mathcal{E})$ is generally smooth, it typically ranges over many orders of magnitude; it is therefore better to interpolate in a table of $\log f$ values, which also provides a convenient opportunity to check that $f \ge 0$ over the entire range.

### 15.3.1. *Single component models*

For the simple case of an isolated spherical density distribution, we have $f_r = -GM(r)/r^2$, with the radial derivative

$$\frac{df_r}{dr} = G\frac{2M - rdM/dr}{r^3} = G\frac{2M(r) - 4\pi r^3\rho(r)}{r^3}. \tag{139}$$

Thus eq. (138) becomes

$$\frac{d^2\rho}{d\Psi^2} = \frac{r^4}{G^2M^2}\left[\frac{d^2\rho}{dr^2} + \frac{d\rho}{dr}\frac{r^2}{GM}G\frac{2M(r) - 4\pi r^3\rho(r)}{r^3}\right]$$
$$= \frac{r^4}{G^2M^2}\left\{\frac{d^2\rho}{dr^2} + \frac{d\rho}{dr}\left[\frac{2}{r} - \frac{4\pi r^2\rho(r)}{M(r)}\right]\right\}. \quad (140)$$

Note that Hénon (1960), Merritt (1985), and Hernquist (1990) were able to integrate these expressions analytically for isochrone, Jaffe, and Hernquist halo models, respectively, and their expressions are programmed as known DFs. The current *GALAXY* package provides numerical solutions to Eddington inversion of NFW and Einasto halo models only.

### 15.3.2. *Composite mass models*

When seeking to determine the DF for a component of density $\rho(r)$ in a composite model, one must use the combined potential, $\Phi$, and central attraction, $f_r$, of all mass components in eq. (135). Although the formula assumes spherical symmetry, it can be used to obtain an acceptable DF for a spherical halo in the presence of a disk. In this case, eq. (138) requires the monopole term only for the attraction of the flattened component, *i.e.*, $f_{r,\rm disk} = -GM_{\rm disk}(r)/r^2$, where $M_{\rm disk}(r)$ is the mass of the thickened disk enclosed in a *sphere* of radius $r$. One must also compute the disk contribution to the total potential from this function for use in eq. (135). Holley-Bockelmann *et al.* (2005) reported, in agreement with my own experience, that no departure from equilibrium was discernible when they used a halo DF created in this way with the embedded flattened disk.

Additional coding will be required to use this method for some desired model, principally in function `dfedfn`, which requires expressions for the first and second derivatives of the density of the chosen mass component. The potential of the model, and its central attraction, are already programmed if every component is one of the models listed in §16. If not, the user will also have to add expressions for its density, mass profile, central attraction, *etc.* as described in §16.2.

## 15.4. **Iterative solution for a halo**

To implement the method proposed by Prendergast & Tomer (1970), we first choose a functional form for the halo DF

$$f = Cf(I_1, I_2, ...), \quad (141)$$

where $C$ is a normalization constant and $f$ can be more or less any reasonable function of the isolating integrals, $I_n$. *e.g.*, in an axisymmetric potential, they can be the classical integrals $E$ and $L_z$. The form of $f$ adopted determines the density profile and shape of the resulting halo; functions of $E$ alone tend to produce almost spherical halos (the disk makes them only slightly oblate), while adding a dependence on $L_z$ generally produces more strongly spheroidal halos. Vasiliev (2019) uses actions in place of the classical integrals in his *AGAMA* package.

We start by adopting an initial guess for the total gravitational potential $\Phi_1(R, z)$ of the disk and halo, which need not be even approximately correct. We set $\Phi_{\rm max}$ to be potential of the surface at which the halo density drops to zero, which is most conveniently determined by the limiting radius in the mid-plane $\Phi_{\rm max} = \Phi(R_{\rm max}, 0)$. We then calculate a first approximation to the halo density by integrating the DF over all allowed velocities: $\rho_1(R, z) = \int f\, d^3\boldsymbol{v}$. A single integration is sufficient for an isotropic DF

$$\rho_1(R, z) = 4\pi \int_0^{u_{\rm max}} u^2 f(E)\, du, \quad (142)$$

where $u_{\rm max} = [2(\Phi_{\rm max} - \Phi(R, z)]^{1/2}$, and $E = \Phi(R, z) + u^2/2$. A 2D integral is required for an anisotropic DF

$$\rho_1(R, z) = 4\pi \int_0^{u_{\rm max}} u \int_0^{v_{\rm max}} f(E, L_z)\, dvdu, \quad (143)$$

where $v_{\rm max} = (u^2_{\rm max} - u^2)^{1/2}$, $E = \Phi(R, z) + (u^2 + v^2)/2$, and $L_z = Rv$.

We assign mass to the grid to represent the smooth function $\rho_1(R, z)$, add the mass distribution of a smooth disk and solve for a new gravitational field $\Phi_2(R, z)$. We then determine $\rho_2(R, z)$ using this improved potential in (142) or (143), and iterate until the potential distribution converges. The value of the normalizing constant $C$ can be adjusted at each iteration step to drive the solution to the desired halo mass. The iteration converges rapidly; 10 iterations are usually ample.

Note that although the halo density profile, and therefore the net rotation curve, cannot be specified in advance, the rapid convergence permits many models to be explored (for different mass ratio, truncation radius, choice of $f$, *etc.*) from which one having the desired properties may be selected.

### 15.4.1. *Using* `dfiter`

This procedure is implemented in the program `dfiter`. The program first requests a run number, then reads the ASCII `.dat` file described in §5, which specifies the mass model, DF type to be used for the halo (see below), and the grid parameters. There are two interactive questions: a beginning user will always want to select a cold start and allow for disk flattening. It is possible to use it to find an equilibrium DF for spherical halo in the presence of a rigid, spherically symmetric mass component. In either case, the program can also be used to refine a previously obtained potential (a "warm" start) by insisting on further iterations, but this option is rarely useful.

The program then iterates to convergence, usually in a couple of minutes, creating a file `runX.pft` (for potential fit), which contains the coefficients of a cubic-spline interpolant of the 2D function $\Phi(R, z)$. This file is required for all subsequent steps in the setup process; the selection of particles from the DF, the creation and running of the simulation, and also the post-run analysis.

As usual, the `.dat` file specifies the number of mass components, at least one of which should be the disk. The spheroidal component that will be composed of particles is not a disk; select type 'DFIT' with an additional integer parameter to specify the functional form for the DF. All available options in the released version are for isotropic DFs, but the generalization to $f(E, L_z)$, *e.g.* to create rotating halos (Debattista & Sellwood 2000), is straightforward:

1. Polytropic form: $f(E) = C(\Phi_{\max} - E)^{\alpha}$, where $\alpha$ is a parameter
2. Lowered isothermal, or King model: $f(E) = C \exp[(\Phi_{\max} - E)/\sigma^2 - 1]$, where $\sigma$ is a parameter

Then specify the desired mass and scale length as usual, and select DFtype 'DFIT' with the desired parameter. The model can optionally include other mass components, which must be rigid.

The mid-plane rotation curve of the composite model can be viewed using the stand-alone program `plotpft`. Note that the fourth and last plot displayed by this program subtracts off the theoretical attraction in the mid-plane of the disk, assuming it to be razor thin and without gravity softening. The halo contribution inferred from the remaining central attraction is therefore under-estimated, and may even appear to imply a negative density at some radii! The user should not be alarmed by this apparent implication; the halo density is positive everywhere, and this ugly feature simply resulted from an over-estimate of the disk attaction.

## 15.5. **Compressed halos**

As described above, adiabatic compression of a halo is probably the best method to create spherical components that will be in equilibrium with an embedded disk. The procedure has three steps:

1. Determine the potential and revised halo densities that will be in equilibrium after adiabatic compression with an added disk, and optionally another, rigid component.
2. Select particles from the DF(s) of the compressed halo(s).
3. Create an $N$-body realization of the composite system.

### 15.5.1. *Using* `compress`

This program determines the potential and revised density of a bulge and/or halo that will be in equilibrium after adiabatic compression with added rigid mass, usually a disk. The following *unix* script creates a file `compress.dat`, assuming the executable `compress` is in your path.

The program also creates a file `E0tab.dat` that is quite time-consuming to build. If the user wishes to try different disk masses or radial scales, while keeping the same halo, the `E0tab.dat` file should be preserved in order to avoid having to recompute it in each run of `compress`. On the other hand, the file `compress.dat` will be overwritten as it is specific to the complete composite model. If the user is experimenting with different disks, and believes a previous version from an earlier run of `compress` could still be useful, that version of `compress.dat` should be saved elsewhere, or renamed, before rerunning `compress`.

The following script, which could take 10-20 minutes to execute, is for the most general case of a model with a bulge, halo and disk. The line numbers are for reference and are not part of the script.

```
    #!/bin/sh
    compress << eof
 1  0            # input from this script (or teminal if interactive)
    #
    # Specify uncompressed and compressing components
    #
 2  3            # number of mass components
    #
    # Uncompressed bulge
    #
 3  n            # this is not a disk?
 4  hern         # bulge type – selected from §16.1.2
 5  0.333        # bulge mass
 6  0.2          # bulge radial scale
 7  10           # bulge truncation radius r_max in units of bulge radial scale
 8  hern         # bulge DF type – selected from §16.1.3
 9  0            # bulge DF parameter (0 for ergodic)
    #
    # Uncompressed halo
    #
10  n            # this is not a disk?
11  isot         # halo type – selected from §16.1.2
12  0.333        # asymptotic velocity of this halo
13  0.2          # halo radial scale
14  10           # halo truncation radius r_max in units of halo radial scal
15  eddi         # bulge DF type – selected from §16.1.3
    #
    # Compressing disk
    #
16  y            # yes a disk
17  kt           # disk type – selected from §16.1.1
18  0.6667       # disk mass
19  1.           # disk scale length
20  5.0          # disk truncation radius r_max in units of disk radial scal
21  none         # disk DF type – selected from §16.1.1
22  1.5          # initial Q (ignored here)
23  4.5          # inner edge of outer taper
    #
28  y            # OK to overwrite existing file compress.dat
    eof
```

The files `E0tab1.dat`, `E0tab2.dat`, and `compress.dat` created by this script will be needed at all subsequent stages of setting up, running, and analyzing the simulation.

**Notes on the input values**

Line 1: a toggle to say whether the input data are to be read from the terminal or a file. Entering a 1 here (not recommended) generates a new prompt to enter the run number X, and `compress` will then attempt open the file runX.dat, and read the data from there.

Line 2: specify the number of mass components (practical maximum is 3)

Lines 3 – 23: Standard inputs, similar to those described in §5. Infinite models, such as the Hernquist bulge and cored isothermal halo, need to be truncated at some radius $r_{\rm max}$ (entered in lines 7 and 14).

Lines 24 – 27: usual input lines to select graphics output, which will show how the halo density and rotation curve are changed by the addition of the disk.

Line 28: If a file `compress.dat` already exists, the program queries whether it would be OK to overwite it. Answering "n" will preserve the existing file and discard the result from the current run.

The program `compress` is versatile enough to allow for a two compressed components (Berrier & Sellwood 2016) together with one or two rigid components (specified but giving the DF type = 'NONE'), or even two spherical components with no disk, and/or a some extra perturber such as a central black hole.

### 15.5.2. *Selecting particles*

Compression changes $f(E, L)$ in a manner that is calculated in `compress` and the crucial information has been saved in the files it created. To run `smooth`, and other programs that use them, the halo type(s) should be set to "ADIA" and the DF type should be "COMP". The remaining inputs to `smooth` to select particles should be as given in the above script.

If, as in the above example, there are two compressed halo components, `smooth` will request the user to input which, (*i.e.* 1 or 2) is to be created with this run of `smooth`. The resulting files should be named `.df1` and `.df2`, and not the usual `.dfn`.

### 15.5.3. *Running* `mkpcs`

The `runXXX.dat` file should again specify the mass components in the order in which they were introduced in the above script for `compress`. However, the bulge/halo types should be changed to "ADIA" and the DF type should be "COMP".

Before running `mkpcs`, you should ensure that the file(s) `E0tab.dat` are accessible within your working directory, as well as a file `runXXX.cmp` that is a logical link to the previously created file `compress.dat`. Both names of that file are used by various programs not ideal.

## 15.6. **In-plane motion in disks**

While generating "cold" disks in centrifugal balance with no random motion is quite trivial, setting velocities for a warm disk is much harder. One must start by choosing the level of random motion $\sigma_R(R)$, usually through specification of the local stability parameter (Toomre 1964)

$$Q(R) = \frac{\sigma_R}{\sigma_{R,\rm crit}} \qquad \text{where} \qquad \sigma_{R,\rm crit} = \frac{3.36 G\Sigma}{\kappa}. \tag{144}$$

Here $\Sigma(R)$ is the local mass surface density and $\kappa(R)$ is the local epicyclic frequency.

1. Some disks have **known DF**s that are in equilibrium in a known mid-plane potential. In these ideal cases, we can select particles from the DF using the selection software described in § 15.9. However, the DF is generally derived for the Newtonian attraction of a razor-thin disk, and the central attraction in an $N$-body code usually differs from that because of softened gravity, finite disk thickness, and disk truncations. Thus the particles selected from the DF will not be in equilibrium unless the numerically determined central attraction in the mid-plane is corrected for these shortcomings (§15.10).

2. Shu (1969) describes a method to **derive** a DF $[f(E, L)]$ for a specified radial dispersion and surface mass density. This option, which is recommended when random speeds are a significant fraction of the circular speed (always true in the center), is described in detail below (§15.7).

3. When random motions are a small fraction of the circular speed, $v_c$, one can set the azimuthal dispersion using the epicyclic relation

$$\sigma_\phi = \frac{\kappa}{2\Omega}\sigma_R, \tag{145}$$

where $\Omega = v_c/R$. We then use the **Jeans equation** (BT08, eq. 4.227) to estimate the mean orbital speed $\overline{v_\phi}$

$$\overline{v_\phi}^{\,2} = v_c^2 - \sigma_\phi^2 + \sigma_R^2\left[1 + \frac{d\ln(\Sigma\sigma_R^2)}{d\ln R}\right]. \tag{146}$$

These formulae are useful for cool, low-mass disks, but the epicycle approximation is woefully inadequate near the centers of realistically massive and warm disks, where the RHS of the Jeans equation can also be negative!
**Caution**: if the disk has been abruptly truncated at some outer radius $R_{\rm max}$, then these formulae yield a finite dispersion for all $R \le R_{\rm max}$, and generating random velocities with the computed $\sigma_R$ will cause an initial disequilibrium: the outer edge will pulsate radially at the epicyclic frequency; this is because the epicycle phases at the outer edge are not uniformly populated. To avoid this, I recommend tapering the surface density smoothly to zero over a finite radial range, which can be requested in a line in the input data file (see §5.3.1). The surface density taper causes the required random motions to decline to zero yielding better radial balance near the edge.

## 15.7. **Shu's DF for disks**

### 15.7.1. *Shu's derivation*

These notes are an adaptation, using the notation of BT08, of the derivation of the DF for an arbitrary razor-thin disk in §§IV-V of the paper by Shu (1969).

The circular angular frequency $\Omega(R)$ in the mid-plane of the gravitational potential $\Phi(R)$, is given by $R\Omega^2(R) = d\Phi/dR$. The guiding center, or home, radius of a star $R_g$ is defined by its specific angular momentum

$$L_z = R_g^2\,\Omega(R_g). \tag{147}$$

The specific energy of a star on a circular orbit is $E_c(L_z) = L_z^2/(2R_g^2) + \Phi(R_g)$, and the epicyclic energy, $\mathcal{E} = E - E_c$, is its excess energy $E$ above that for a circular orbit of the same $L_z$.

Let the desired radial velocity dispersion be $\sigma_R(R_g)$, and write the DF as

$$f(E, L_z) = \begin{cases} \mathcal{F}(L_z) e^{-\mathcal{E}/\sigma_R^2(R_g)} & 0 < \mathcal{E} \le -E_c(L_z) \text{ and } L_z \ge 0, \\ 0 & \text{otherwise.} \end{cases} \tag{148}$$

Although this assumes no retrograde stars, the angular momentum of some particles can be reversed later without affecting the equilibrium. We also assume a finite disk with an outer edge $R_{\rm max}$ and maximum angular momentum $L_{\rm max}$, which is that of a circular orbit at $R_{\rm max}$.

The function $\mathcal{F}(L_z)$ must be chosen to satisfy the condition at all $R > 0$

$$R\Sigma(R) = \int\int f(E, L_z) dv_R dL_z, \tag{149}$$

where $\Sigma(R)$ is the disk surface density. Following Shu, we subtract the kinetic energy of radial motion, $v_R^2/2$, from the rest of the non-circular energy, $\mathcal{E}$, which leaves

$$\mathcal{E}_p(L_z, R) \equiv \mathcal{E} - \frac{v_R^2}{2} = \Phi_{\rm eff}(L_z, R) - E_c(L_z) \qquad \text{where} \qquad \Phi_{\rm eff}(L_z, R) = \Phi(R) + \frac{L_z^2}{2R^2}. \tag{150}$$

Thus the integral for the surface density becomes

$$R\Sigma(R) = \int_0^{L_{z,\rm max}(R)} \mathcal{F}(L_z) e^{-\mathcal{E}_p/\sigma_R^2(R_g)}\, 2\int_0^{v_{R,\rm max}} e^{-v_R^2/2\sigma_R^2(R_g)} dv_R dL_z. \tag{151}$$

The inner integral is doubled because the DF is symmetric for inward and outward moving stars and the upper integration bound is $v_{R,\rm max} = \{2[E_{\rm max}(L_z) - E_c(L_z)]\}^{1/2}$, with $E_{\rm max}(L_z) = \Phi(R_{\rm max}) + 0.5(L_z/R_{\rm max})^2$. The upper bound on the outer integral, $L_{z,\rm max}(R)$, is the largest angular momentum of orbits that pass through $R$. This value is for a star whose pericenter is $R$ and apocenter is the disk edge; since it has the same energy and $v_R = 0$ at both turning points $E(L_z) = \Phi(R_{\rm max}) + L_z^2/(2R_{\rm max}^2) = \Phi(R) + L_z^2/(2R^2)$ and we find

$$L_{z,\rm max}(R) = \left[2\frac{\Phi(R) - \Phi(R_{\rm max})}{R_{\rm max}^{-2} - R^{-2}}\right]^{1/2}. \tag{152}$$

Substituting $v_R = \sqrt{2}\,\sigma_R t$, and using the standard form $\int_0^x e^{-t^2} dt \equiv \frac{1}{2}\pi^{1/2}{\rm erf}(x)$, eq. (151) becomes

$$R\Sigma(R) = (2\pi)^{1/2} \int_0^{L_{z,\rm max}(R)} \mathcal{F}(L_z) \exp\left[-\frac{\mathcal{E}_p(L_z, R)}{\sigma_R^2(R_g)}\right] \sigma_R(R_g)\,{\rm erf}\left[\frac{v_{R,\rm max}}{\sqrt{2}\,\sigma_R(R_g)}\right] dL_z. \tag{153}$$

More compactly

$$R\Sigma(R) = \int_0^{L_{z,\rm max}(R)} \mathcal{F}(L_z)\mathcal{K}(L_z, R)\, dL_z, \tag{154}$$

where the kernel is

$$\mathcal{K}(L_z, R) = (2\pi)^{1/2} \exp\left[-\frac{\mathcal{E}_p(L_z, R)}{\sigma_R^2(R_g)}\right] \sigma_R(R_g)\,{\rm erf}\left[\frac{v_{R,\rm max}}{\sqrt{2}\,\sigma_R(R_g)}\right] \tag{155}$$

Shu changes integration variables from $(v_R, L_z)$ to $(v_R, R_g)$, but I do not see any advantage in doing that.

The central surface density at $R = R_g = 0$ requires separate treatment. In the center, $v_{R,\rm max}^2 = 2\,[\Phi(r_{\rm max}) - \Phi(0)]$, and $\mathcal{E} = (v_R^2 + v_\phi^2)/2$. We therefore have

$$\Sigma(0) = 2\int_0^{v_{R,\rm max}} dv_R \int_0^{v_{\phi,\rm max}(v_R)} dv_\phi\, \mathcal{F}(0)\, e^{-(v_R^2+v_\phi^2)/2\sigma_R^2} = \mathcal{F}(0)\,\pi\sigma_R^2(0)\,\left[1 - \exp\left(-\frac{v_{R,\rm max}^2}{2\sigma_R^2(0)}\right)\right]. \tag{156}$$

### 15.7.2. *Method of solution*

We wish to find a function $\mathcal{F}(L_z)$ for $0 \le L_z \le L_{z,\max}(R)$ that satisfies eq. (154) for all $0 \le R \le R_{\max}$. As there are very many possible functions $\mathcal{F}(L_z)$ that meet this requirement, we seek the one for which $\mathcal{F}(L_z) \ge 0$ over the allowed range and is a smooth as possible wrt variations in $L_z$, as is physically expected for a well-relaxed disk.

We write $\mathcal{F}(L_z)$ as a Chebyshev series (Press *et al.* 1992, their formula 5.6.8)

$$\mathcal{F}(L_z) \simeq \left[\sum_{k=1}^{N_k} c_k T_{k-1}(L')\right] - \frac{1}{2}c_1 \qquad \text{with} \qquad L' \equiv \frac{2L_z - L_{z,\max}(R_{\max})}{L_{z,\max}(R_{\max})}. \tag{157}$$

Thus $-1 \le L' \le 1$, as required for Chebyshev polynomials, and the set of $\{c_k\}$ are to be determined. Since $T_0(L') \equiv 1$ for all $L'$, an equivalent expression is $\mathcal{F}(L_z) \simeq \sum_{k=1}^{N_k} c_k T_{k-1}(L')(2-\delta_{k1})/2$, with $\delta_{k1}$ being the Kroneker delta. Inserting this expansion into eq. (154) and interchanging the summation and integration, we obtain

$$\Sigma(R) = \begin{cases} R^{-1}\sum_{k=1}^{N_k} c_k(2-\delta_{k1})/2 \int_0^{L_{z,\max}(R)} T_{k-1}(L')\ \mathcal{K}(L_z,R)\, dL_z & R > 0 \\ S_0 \sum_{k=1}^{N_k} c_k T_{k-1}(-1)(2-\delta_{k1})/2 & R = 0. \end{cases} \tag{158}$$

From eq. (156), we see that the constant $S_0 = \pi\sigma_R^2(0)\ \left\{1 - \exp\left[(\Phi(0) - \Phi(R_{\max}))/\sigma_R^2(0)\right]\right\}$. Since the $c_k$s are constants, we can separate the expression for each $k$ to find for $R > 0$

$$\left[\Sigma(R_i)\right]_k = \frac{c_k(2-\delta_{k1})}{2R_i} \int_0^{L_{z,\max}(R_i)} T_{k-1}(L')\ \mathcal{K}(L_z,R_i)\, dL_z \equiv \frac{c_k(2-\delta_{k1})}{2R_i}\mathcal{I}(k,R_i). \tag{159}$$

Now the integral on the RHS can be evaluated for any given $R_i > 0$, and the problem now becomes to find the set of $\{c_k\}$ that minimize

$$\mathcal{C} = \left[\Sigma(0) - S_0\sum_{k=1}^{N_k}(-1)\frac{c_k(2-\delta_{k1})}{2}T_{k-1}\right]^2 + \sum_{i=2}^{N_R}\left[\Sigma(R_i) - \frac{1}{R_i}\sum_{k=1}^{N_k}\frac{c_k(2-\delta_{k1})}{2}\mathcal{I}(k,R_i)\right]^2 w(R_i), \tag{160}$$

at a set of $N_R$ radii $\{R_i\}$, with $R_1 = 0$. The summation over $R_i$ implies that the system of equations is over-determined and we must seek a solution for the set of $\{c_k\}$ numerically. We have included a weight function $w(R_i)$ because the terms in square brackets that are differenced decrease in value with increasing radius, and without aggressive weighting, will have little constraint on the global minimum. Setting $w(R_i) = \Sigma(R_i)^{-2}$ seems to work well.

It turns out that the $R = 0$ term in eq. (160) generally spoils the fit (perhaps because of an error in eq. (156)?) and I have found it expedient to omit it and instead adopt a tiny value for $R_2$ to ensure a resonable fit to $\Sigma(R)$ at all radii.

Numerical minimizations are faster when derivatives can be supplied, which is a further advantage of using Chebyshev funtions because we can readily differentiate $\mathcal{C}$ wrt each $c_k$ in turn. To wit

$$\frac{\partial\mathcal{C}}{\partial c_j} = -2\sum_{i=2}^{N_R}\left[\Sigma(R_i) - \frac{1}{R_i}\sum_{k=1}^{N_k}\frac{(2-\delta_{k1})c_k}{2}\mathcal{I}(k,R_i)\right] w(R_i)\frac{2-\delta_{j1}}{2}\frac{\mathcal{I}(j,R_i)}{R_i}. \tag{161}$$

However, the minimization must be subject to the additional constraints, mentined above, that $\mathcal{F}(L_z) \ge 0$ for all $L_z$, and $\mathcal{F}(L_z)$ should not oscillate strongly over the allowed range of $L_z$, behavior that is likely when the Chebyshev expansion includes functions up to even modest $N_k$. In order to include these constraints, the minimization of $\mathcal{C}$ in eq. (160) must be supplemented by penalty functions.

The positivity constraint requires that for each suggested set of $\{c_k\}$, we compute

$$\mathcal{S} = \left[\mathcal{F}(L_z)\right]_{\min} \qquad \text{for many values in the range} \qquad 0 < L_z < L_{z,\max}(R_{\max}) \tag{162}$$

and, only if $\mathcal{S} < 0$, add to $\mathcal{C} + \upsilon_n\mathcal{S}^2$, where a large value of the constant $\upsilon_n = 10^4$, say (a hard-wired value), strongly disfavors all negative values. It turns out that $\mathcal{S} > 0$ for important cases, when this constraint has no effect.

In order to inhibit oscillatory behavior of $\mathcal{F}(L_z)$, we minimize

$$\mathcal{C} + \upsilon_o\mathcal{T} \qquad \text{where} \qquad \mathcal{T} = \sum_{L_z}\left[\frac{d^2\mathcal{F}}{dL_z^2}\right]^2, \tag{163}$$

and $\upsilon_o$ is a user-supplied constant. Press *et al.* (1992, their §5.7) describe how derivatives can readily be obtained from the coefficients of an expansion in Chebyschev polynomials. Experience suggests $\upsilon_o \sim 10^{-5}$ can be appropriate; too large a penalty causes a linear variation of $\mathcal{F}(L_z)$ and a very poor fit to the desired surface density, while too small a value allows mild oscillations in the function $\mathcal{F}(L_z)$.

Sadly, neither of these penalty functions is readily differentiable wrt the $\{c_k\}$, and so requires a minimzation routine that does not supply derivatives. However, we have found it efficient to make a first estimate for $\{c_k\}$, neglecting the penalty terms for which the derivatives of eq. (161) can be used for rapid convergence, and then to use this set as the starting guess for $\upsilon_o > 0$ to find the constrained fit.

Since particle selection requires millions of DF values from eq. (148), we construct a 1D table of values for $\mathcal{F}$ and interpolate to find its value for the desired $L_z$, before factoring the Gaussian in epicyclic energy. Complete information that enables this fit to be reconstructed is saved in a file `.dft`, which is needed subsequently for model set up and analysis.

### 15.7.3. *Limitations of the method*

While this strategy achieves a DF for a disk that approximates the desired $\Sigma(R)$ and $Q(R)$-profiles, a perfect match seems elusive. The largest discrepancies generally arise near the outer edge of the disk, where the required surface density should be tapered smoothly to zero. Without an outer taper, a sharp outer boundary to the disk that would not blur as the particles move on their epicycles would require a DF having an increasing fraction of low-eccentricity orbits, tending to purely circular ones to maintain a finite surface density at the edge. That would force $Q \to 0$ as $R \to R_{\rm max}$, which is undesirable. However, an outer taper causes there to be few orbits that reach reach the outer edge, and the large $L_z$ end of $\mathcal{F}$ is therefore not strongly constrained by the minimization of $\mathcal{C}$, which results in increasgly large departures from the desired $\Sigma$ and $Q$ profiles over the region of the outer taper. Physically, this should not matter when the interest is in the dynamics of the massive part of the disk.

### 15.7.4. *Advice to the user*

This program cannot be used as a "black box", since it will find one of the many different solutions that apparently fit the desired disk surface density and $Q$-profiles, but which may have other undesirable properties. The user is *strongly advised* not to accept a solution for $\mathcal{F}(L_z)$ without careful examination; even quite mild undulations away from the disk center and outer edge can prejudice the dynamical stability of the entire disk, and an improved solution should be sought.

The program includes an interactive graphical interface to enable the user to visualize intermediate results. After the completion of each minimization, the resulting disk surface density is compared with the target density, and function $\mathcal{F}(L_z)$ is displayed. The user is then prompted whether to accept that solution or to find a new fit using a different value for the smoothing penalty.

The two most important parameters to choose are $N_k$, the number of Chebyshev functions that are used, and $\upsilon_o$, the smoothing penalty. If $N_k$ is too small, on the one hand, the fit has insufficient freedom to match the desired radial profiles of $\Sigma$ and $Q$. While on the other hand too large an $N_k$ makes it hard to suppress small-scale variations in the function $\mathcal{F}(L_z)$. Even very mild oscillations in the density of particles as a function of $L_z$ can provoke instabilities related to groove modes, which need to be suppressed by a larger smoothing penalty, $\upsilon_o$, but at the cost of biasing the DF. I have found that the sweet spot for a simple exponential disk having a constant $Q$ is for $N_k = 40$ and $\upsilon_o \simeq 5 \times 10^{-5}$.

The user may also consider changing the radial weighting function $w(R_i)$. I have found $w(R_i) = \Sigma^{-2}$ is ideal for nearly full mass disks, but is perhaps too aggressive for submaximum disks where $w(R_i) = \Sigma^{-1}$ seems to work better. Note that changes to either $N_k$ or to the radial weighting requires that the source code be edited and recompiled.

### 15.7.5. *Thickened disks*

Expressions for $\Omega(R_g)$, *etc.* are usually known for a razor-thin disk without gravity softening. The potential of a thickened disk could, in principle, be determined using Hankel transforms (BT08), but it is easier to use the gravitational field determined by one of the methods described in §8, which takes account of both finite thickness and gravity softening. Since particle positions and velocities are not available until *after* the DF is determined, we assign to the grid the mass of a smooth disk that has been tapered to zero at $R_{\rm max}$ and spread vertically, as well as any additional mass components, from which we determine an azimuthally averaged field in the disk mid-plane.

### 15.7.6. *Procedure*

The program `smooth`, described in §15.9.3, performs all stages of this task. It determines the gravitational field, solves for the function $\mathcal{F}(L_z)$, selects particles in the manner described in §15.9 below, and outputs the radius and in-plane velocities of each particle. It also creates two auxiliary files, with extensions `.dft` and `.ftb`, which should be preserved if the user is content with the set up, but should be deleted if the user wishes to try other parameters.

The vertical height of each particle is determined separately and a vertical velocity assigned as described next.

## 15.8. **Vertical motion in disks**

As noted above, 2-integral DFs for thin disks must necessarily have a velocity ellipsoid that is unrealistically spheroidal and almost no 3-integral DFs are known; therefore the only viable approach is to treat the vertical balance separately from the in-plane motions. We require a practical scheme to set vertical velocities in equilibrium for a non-uniform disk, with possibly varying disk thickness, and in the presence of other mass components. Since vertical oscillations have short periods, an imbalanced disk does settle quickly, although usually to a different thickness and vertical density profile.

Spitzer (see BT08) derived the isothermal DF $f(E_z)$ for a uniform slab with a $\mathrm{sech}^2(z/z_0)$ density profile. This DF has a number of disadvantges:

1. while the density asymptotes to an exponential at large $z$, real galaxy disks have vertical profiles that are generally more sharply peaked than is this function,
2. applying Spitzer's formulae locally in disks with radial density gradients results in an only approximate equilibrium,
3. the formulae neglect the possible existence of other mass components, such as a bulge or halo, and
4. above all, particle softening reduces the sharpness of the restoring force to the mid-plane, which causes the predicted vertical dispersion to be too large, leading to a rapid initial thickening of the disk.

Thus I recommend against the use of Spitzer's formulae.

The code therefore uses a better method, which is to integrate the vertical 1D Jeans equation (BT08, eq. 4222b) for a slab, *i.e.* neglecting radial variation. In this case

$$\sigma_z^2(R,z) = \frac{1}{\rho(R,z)} \int_z^\infty \rho(R,z') \frac{\partial \Phi}{\partial z'}\, dz', \tag{164}$$

where the vertical gradient of the total potential $\Phi$ should be determined from the particles, as well as any additional mass components. This formula generally yields a better equilibrium, and is more versatile. It allows any reasonable vertical density profile $\rho(R,z)$, and works in the presence of other mass components that contribute to the total potential. This approach assumes that the radial and vertical motions are decoupled, an assumption that becomes progressively less valid for disks with large in-plane motions; thus the quality of the resulting equilbrium degrades for dynamically hot disks.

Sanders & Binney (2016) reviewed this and other methods to construct a thickened disk, and found that methods based on a Stäckel approximation are superior and yield a better disk equilibrium. Their method becomes more necessary for disks that are hotter or thicker than are customarily employed.

## 15.9. **Selecting particles from a DF**

The usual approach (*e.g.* Hernquist 1993; Kuijken & Dubinski 1995) to selecting particles from a DF is to generate candidate particles and to compare the value of the DF for each candidate with an additional random number to decide whether to accept or reject it. This approach yields a density in integral space that differs, at the shot noise level, from that specified; in effect the model will have the dynamical properties of a DF slightly different from that intended.

To illustrate the consequences of random selection, the blue points in Fig. 11 show the scatter in the estimated values of the eigenfrequency of the mode mentioned in §14.2.5. The blue points report the real and imaginary parts of the estimated eigenfrequency in separate simulations in which coordinates of the 40K independent particles were selected by random sampling, while the red points show the values from a separate set of simulations in which the integrals for the particles were selected in the smooth manner as described below. The SFP method (§8.11.3) used only $m = 2$ functions and we suppressed the initial $m = 2$ amplitude by employing 3 copies of each particle arranged symmetrically around a half-ring. The blue symbols range ±21% (or ±8% if the outlier is omitted) from the mean pattern speed and ±24% from the mean value of the growth rate, whereas the ranges of the red symbols are respectively ±4.4% and ±12%. The frequency predicted from linear theory by Kalnajs (1978) is marked by the black dot.

The reason for the large scatter of the blue points is random fluctuations in the representation of the isochrone/12 DF (see the LH panel of Fig. 12). When I chose particles smoothly by the method described below (RH panel of Fig. 12), which still has a small random element, the simulations yielded the much tighter distribution of points colored red. The smaller scatter results from a much reduced **sampling error** in the selection from the DF, which is achieved by selecting values of the classical integrals $(E, L_z)$ in such a way that their density in this plane is as close as possible to the specified functional form of the DF. The concept is standard practice in Monte-Carlo integration (*e.g.* Press *et al.* 1992).

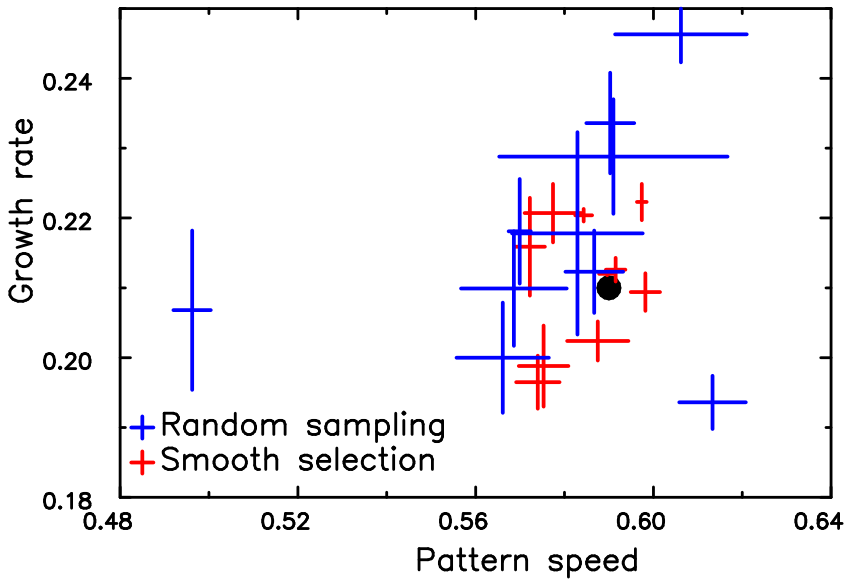


FIG. 11.— The scatter from many simulations of the measured pattern speed and growth rate of the dominant mode of the isochrone/12 disk. Each colored point is the best-fit value from a simulation employing 120K particles, with error bars indicating the full range from different acceptable fits to the data. The blue points record the measured values when random sampling is used, the red points when the sampling error is reduced by the smooth selection procedure summarized here. The large black dot marks the frequency predicted by Kalnajs (1978) from linear theory.

### 15.9.1. *Smooth particle selection*

Here I outline the particle selection procedure described in full by Sellwood (2024). The density of particles in the sub-space of the classical integrals is not simply given by the DF; it needs to be multiplied by a "density of states" function that is the phase-space volume per unit interval of $E$ and $L_z$ (BT08, their §4.3.1(b)). Sellwood (2024) derives the following expressions for the mass fraction in an infinitesimal element of integral space:

1. For a razor-thin disk: $d^2M/dEdL_z = 2\pi\tau(E,L_z)f(E,L_z)$, where $\tau$ is the period of one complete radial oscillation of a particle with the given $(E,L_z)$.
2. For a sphere: $d^2M/dEdL = 8\pi^2 L\tau(E,L)f(E,L)$, with $L$ being the total angular momentum.
3. For a spheroid: $d^2M/dEdL_z = 4\pi^2 S(E,L_z)f(E,L_z)$, where $S(E,L_z)$ is the cross-sectional area of the bounding torus in the meridional plane.

The value of $\tau$ or $S$ for each $(E,L)$ generally needs to be computed from a 1D integral. Although each quadrature is fast, the number of evaluations needed makes the overall calculation time-consuming, and it is more efficient to interpolate in a precalculated 2D table of values.

The total active mass represented by the DF is therefore

$$M = \int_{E_{\rm min}}^{E_{\rm max}} \int_{L_{\rm min}(E)}^{L_{\rm max}(E)} \frac{d^2M}{dEdL_z} dLdE, \tag{165}$$

where the bounds are determined by the potential well and any limits imposed by the user. For spheres and spheroids, I generally employ $E_{\rm max} = \Phi(r_{\rm max})$ with $r_{\rm max}$ being the desired radial extent of the population, and $E_{\rm min} = \Phi(0)$. However, for disks, an abrupt energy truncation leaves no particles on nearly circular orbits in the outer disk and, since the DF generally takes on large values near circular orbits, a sharp energy cut-off removes an excessive fraction of the mass. In this case, therefore I allow the upper energy bound to increase with $L_z$ as

$$E_{\rm max}(L_z) = \Phi(R_{\rm max}) + 0.5\,(L_z/L_{z,\rm max})^2\,, \tag{166}$$

with $L_{z,\rm max}$ being the angular momentum of a circular orbit at $R_{\rm max}$. This condition allows particles on circular orbits up to the truncation edge at $R_{\rm max}$, but excludes any eccentric orbit having an apocenter $> R_{\rm max}$.

To select particles, we proceed by slicing $(E,L)$ space into $k$ small areas in such a way that the integral of $d^2M/dE,dL$ over each elemental area encloses a fraction $k^{-1}$ of $M_{\rm act}$. We then select a point within each of these areas to determine the $(E,L)$ values for an orbit. The only random part is the selection of the point within each small element, in order

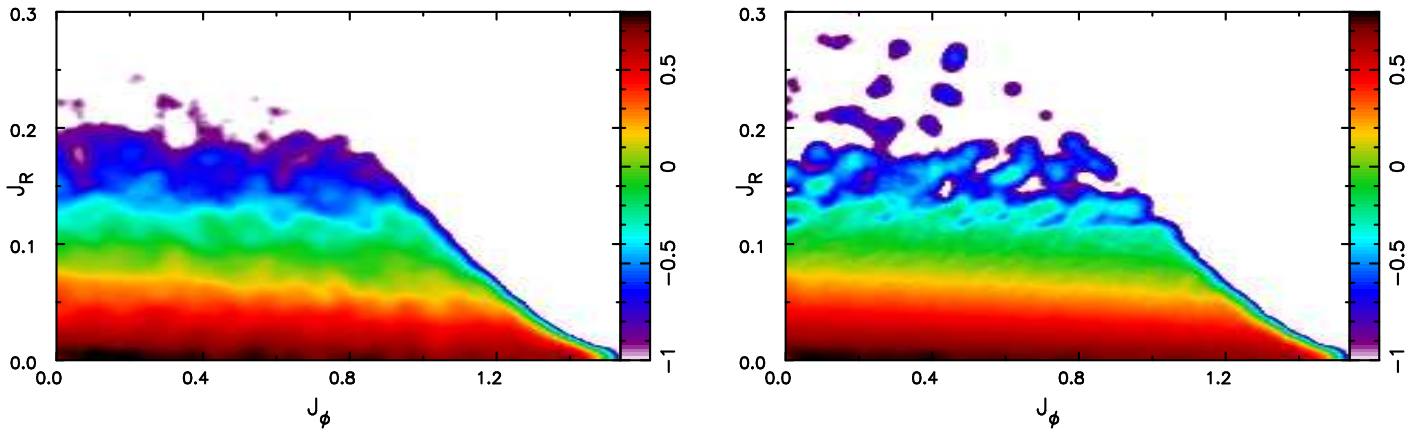


FIG. 12.— The logarithmic density of selected particles in the space of the two actions $J_\phi \equiv L_z$ and the radial action $J_R$ when random sampling is used (left panel) and when $E$ and $L_z$ are selected smoothly (right panel), as descibed here. 40K particles were selected by each method from the isochrone/12 DF described in §14.2.5, with the radial extent limited such that no particle has sufficient energy to cross the truncation radius. Notice the non-smoothness in the densely populated lower part of the left panel.

to avoid a regular raster of selected points. This approach yields a density of selected values in $(E, L)$ space that is an almost noise-free approximation to $d^2M/dE, dL$, as illustrated in the RH panel of Fig. 12.

Each $(E, L)$ pair selected in this way specifies the entire orbit of the selected particle in the known potential. We must next choose phases to determine the initial position and velocity components for each particle; if $n$ particles are selected from each orbit, the total number selected is $N = kn$ and each has a mass $\mu = M_{\rm act}/N$. In contrast to the selection of integrals, experience suggests that both the quality of the equilibrium and the behavior of the simulation is much less sensitive to the manner in which orbital phases are selected. In general, random selection from the appropriate distribution is adequate, although it is easy to improve upon random when desired for a particular application. For example: it can be desirable to space several otherwise identical particles equally around a ring when searching for small-amplitude non-axisymmetric instabilities, a **quiet start** (see §4.1.2, Sellwood 1983, 2024).

In a razor-thin disk, or in a sphere, the orbit lies in a plane in which particles oscillate between peri- and apocenter. The probability of selecting a particular radius varies inversely with the (non-uniform) radial speed. It is easiest to select a fraction of the radial period and integrate the orbit (usually numerically) for this time to determine the radius $r$. The radial and azimuthal speeds are completely determined (except for the sign ambiguity of the radial speed) by the selected values of $E$, $L$ and $r$. The azimuthal phase and, for a sphere, the orientation of the plane can be selected in a straightforward manner.

In spheroidal models, the two classical integrals, $E$ and $L_z$, confine the particle to a torus in real space. When the DF does not depend upon a third integral, the probability density distribution for any given orbit is uniform in the meridional plane within the boundaries of the torus and selection of $(R, z)$ pairs is straightforward. The values of $E$ and $L$ and $\Phi(R, z)$ determine the magnitude of the velocity in the meridional plane, and its direction in that plane is uniformly distributed in angle.

This procedure is implemented in program `smooth`, and produces a file (`.dfn`) containing a partial set of coordinates for each particle, such as $(r, v_r, v_\phi)$. The other missing coordinates that are not included can readily be generated from simple distributions, and are created in program `mkpcs` (see §4).

### 15.9.2. *Unequal mass particles*

It is sometimes desirable to use particles having a range of masses, and the code described above can readily be modified to achieve this, provided the desired mass of each particle is also a function of the integrals. Any reasonable weight function of either or both integrals $w(E, L)$ can be used, although it must be positive over the entire accessible range of the integrals. The adjusted total mass becomes

$$M' = \int_{E_{\rm min}}^{E_{\rm max}} \int_{L_{\rm min}(E)}^{L_{\rm max}(E)} \frac{1}{w(E,L)} \frac{d^2M}{dEdL}\, dLdE. \qquad (167)$$

Dividing the DF by $w$ necessarily results in the selection of a larger fraction of particles where $w$ has a low value. For example, one may wish to have more low-mass tightly bound particles. Choosing $w(E)$ to be small for low (large negative) $E$ will cause more tightly bound particles to be selected. Alternatively, setting $w(L)$ to dip at a particular $L$ will pack more particles at that angular momentum, *etc.*

To take advantage of this capability, the user will have to edit the function `wghtDF` to return the desired functional form, and then recompile the program `smooth`.

The masses of each selected particle must vary inversely as the same function, *i.e.* $\mu'_j = M'w(E_j, L_j)/N$. Thus the function $w$ does not require a normalization factor. The value of $M'$ is written into the file header and the scaled mass, $w$, is saved for each particle.

### 15.9.3. *Running* `smooth`

The following *unix* script will create a `.dfn` file for most DF types, assuming the executable `smooth` is in your path. A slightly different script, descibed below, is needed for the Shu DF for a disk. The line numbers are for reference and are not part of the script.

```
    #!/bin/sh
    smooth << eof
 1  0              # input from this script (or terminal if rn interactively)
 2  1              # number of mass components
    #
    # component 1
    #
 3  y              # is this a disk?
 4  kt             # disk type – selected from §16.1.1
 5  1              # mass of this component
 6  1              # radial scale of this component
 7  6              # cut off radius r_max
 8  kaln           # DF type – selected from §16.1.3
 9  6              # DF constant
10  .2             # L_crit for retrograde particle rule
11  6              # inner edge of outer taper, which could be less than r_max
    #
    # particle selection
    #
12  y              # equal mass particles?
13  100 20 10      # the product of these three integers is the number of particles to be created
14  0              # enter random seed, or 0 for default
15  y              # OK to accept change of mass fraction
    eof
```

The output file will be named `kaln20k.dfn`, which must be renamed to runX.dfn before use in `mkpcs`.

Like most DFs, Kalnajs (1976b) defined this function for a disk without any other mass component. However, the DF will be in equilibrium, and this file can be used, if the disk has a fraction of the total mass but the total potential is unchanged. In such cases, it is always easier to run `smooth` as if the disk were a single full-mass component, as the software may not recognize that the DF is consistent with a multi-component model.

**Notes on the input values**

Line 1: a toggle to say whether the input data are to be read from the terminal or a file. Entering a 1 here (not recommended) generates a new prompt to enter the run number X, and `smooth` will then attempt open the file runX.dat, described in §5, and read data about the mass components from there, before returning to read lines 12 onwards.

Line 2: the number of mass components: generally 1 only

Lines 3 – 9: Standard inputs, similar to those described in §5. If the number of mass components is greater than one, lines 3–9 will need to be repeated. Infinite models, such as an exponential disk, need to be cut off at some radius $r_{\max}$ (entered in line 7), and `smooth` will ensure that no particles have enough energy to pass $r_{\max}$ in the smooth potential of the infinite model. The mass density therefore tapers smoothly to zero at this radius.

Line 10: Some disk DFs have no retrograde stars, and this parameter determies the width of the angular momemtum region from which retrograde particles are created using a smooth taper.

Line 11: If this parameter is smaller than $r_{\max}$, then the mass density is additionally tapered from the given value.

Line 12: A toggle to indicate whether equal mass particles are required. If 'n', then the executable `smooth` must be recompiled with the function described in §15.9.2, otherwise the supplied version terminates execution with an error message.

Line 13: Three integers, $N_E$, $N_L$, $N_{LE}$ giving the numbers of slices in $E$ and in $L$ together with the number of randomly chosen radial phases of particles at each selected value of $E$ and $L$. The total number of particles to be created is

$N_E \times N_L \times N_{LE}$. Experience suggests that better results are obtained if these values are ordered such that $N_E > N_L > N_{LE}$, but they should not differ by factors $> 10$.

Line 14: The random seed to be used. Any integer value is acceptable and the the value 0 selects the default seed.

Line 15: The double integral of the DF over all accessible $(E, L)$ yields the total active mass of this component, which is compared with expected mass if tapers are ignored. If this difference is greater than 1%, `smooth` asks the user whether the difference is acceptable or indicates an error. If the user is using a DF supplied with the code, the answer should be 'y' to allow the code to proceed. Those who have added a new DF, may wish to consider whether they understand the magnitude of the discrepancy before continuing.

**Using the Shu DF**

No script is needed for the Shu DF since the program `smooth` should be run interactively. The code and parameters for the field determination should be specified in a standard auxilliary input file `runXXX.dat` file (see §5). This file also specifies the mass model, and scaling, as well as the grid to use and its parameters. Having read and used these values, program `smooth` will set up the DF and, if that is successful, it will then request a few more input values from stdin to determine the particle selection. In fact, I recommend that the user exits the interactive program `smooth` after an acceptable solution has been found, which is saved in an ancilliary file. It is then possible to use a script to restart `smooth` which reads the ancilliary file and behaves as for all other DFs. Two examples of how this can be implemented are in the code directory: EXAMPLES/p3d/README or EXAMPLES/hybrid/README

### 15.9.4. *Examining the* `.dfn` *file*

Program `dflook` can be used to check whether the particles have in fact been selected as intended. It is run interactively and produces plots of selected quantities, such as the radial mass profile $m(r)$, surface density $\Sigma$, volume density $\rho$, the first $\bar{v}$ and second $\sigma$ moments of the velocity components, *etc.* Program `dflook` can even compute the expected values of these quantities by integration of the DF over the allowed ranges, provided the DF has a reasonable functional form.[16]

## 15.10. **Equilibrium**

An $N$-body realization of a supposed equilibrium model may not be in precise balance for three good reasons:

1. DFs are generally, though not always, obtained assuming the Newtonian potential, whereas many $N$-body codes use softened gravity, which results in slightly weakened forces.
2. Many models of disks and halos are infinite in extent, and need to be truncated to fit into a finite grid volume. This is best achieved by imposing an energy cut-off that discards particles whose equiibrium orbits would take them beyond the desired outer radius, causing the density of the remaining particles to taper smoothly to zero at the outer edge. The central attraction of the truncated component differs from that assumed for the derived DF, which upsets the radial balance – an effect that is surprisingly large for disks (*e.g.* Toomre 1963; Casertano 1983). In spherical models, however, the discarded mass clearly cannot alter the central attraction well interior to the outer radius (see Sellwood, Shen & Li 2019, for caveats).
3. Forces from a thickened disk of particles in 3D will be weaker than those from a razor-thin disk. The DFs of almost all disks are derived assuming the stronger attraction of the razor-thin case.

For these reasons, it is desirable to determine a supplementary central attraction to correct for these shortcomings. This option is requested by including the line with keyword SUPP in the input `.dat` file, as described in §5.3.2, item #8

This (small) supplemental attraction acts as low-mass rigid halo. When it is activated, therefore, it is desirable to keep the grid center fixed and to suppress dipole ($m = 1$) terms in the force determination, as noted in §8.14.

The correction is determined, if requested, in program `mkpcs`, which is described in §4.1.3. It solves for the field either of the selected particles before they are moved or of a smooth mass distribution that is determined from integrating the truncated DF over the allowed velocity ranges at each radius to obtain the surface density profile, and thickening the disk with the adopted vertical profile. The azimuthally, or spherically, averaged attraction computed from the $N$-body code is then differenced from the Newtonian attraction expected for the infinite and/or razor-thin model and the tabulated differences are stored in a file.

The user should run program `corrplt` to verify that the values in the table of differences are small compared with the actual attraction. Large differences would be symptomatic of some kind of blunder, such as in the spatial or mass scaling of the model.

[16] Exotic DFs, such the $\Omega$-models (Kalnajs 1972) or the uniform sphere (Binney & Tremaine 2008, problem 4.11) are properly handled by the code, but a user adding others should consider adding special code if the DF is not a smooth function of the integrals.

At every step during the simulation, `galaxy` interpolates from this table to estimate the supplementary central attraction to be added to the other forces acting on each particle.

### 15.10.1. *Creating your own table of supplementary forces*

A user who is able to initiate a simulation using `mkpcs` will not need to create their own table as long as the theoretical attraction of the particles is available within the code - *i.e.* the particles are all drawn from one or more of the mass models listed in §§16.1.1 and/or 16.1.2.

Those who create their own `.pcs` file (see §4.2) for use in `pcs2dmp` and who also wish to have the code add correcting forces will need to make their own table. The user will have to calculate the forces from the particles, and difference them from the theoretical attraction. If the system is spherical, the spherically averaged attraction from the particles should be computed, but in the case of a flattened system the azimuthally averaged attracion in the midplane should be computed. (The difference will be applied as a spherically symmetric term, but the polar variation is rarely needed far from the mid-plane and the difference forces should be small ayway.)

The table should be in a binary file (`runXXX.stb`) that will be read by subroutine `suptab`. It must contain a header giving $n_T$ (type integer*4), which cannot exceed 500, and the self-energy $W_c$ (type real*4), which is given by (eq. 172) below. Each of the subsequent $n_T + 1$ Fortran records should have 3 real*4 values: the radius, the correction for the central attraction at that radius, and the gravitational potential (so that the code can keep track of total energy) at that radius. All dimensional quantities should be in the units described in §2.3. The radial spacing of the table cannot be arbitrary, but *must* be such that the $j$-th radius is

$$r_j = \ell \left(\exp^{\alpha_2 j} - 1\right) \qquad \text{with} \qquad 0 \le j \le n_T, \tag{168}$$

where $\ell$ is the adopted length unit (§2.3). Thus

$$\alpha_2 = \frac{\log(r_{n_T}/\ell + 1)}{n_T}. \tag{169}$$

The correction force at each radius is

$$a_c(r_j) = a_{\rm analytic}(r_j) - \langle \boldsymbol{a}_{\rm particles} \cdot \hat{\boldsymbol{r}} \rangle|_{r=r_j} \qquad \text{with} \qquad \boldsymbol{a}_{\rm particles}(\boldsymbol{x}) = \sum_{i=1}^{N} \mu_i \boldsymbol{S}(\boldsymbol{x}_i - \boldsymbol{x}), \tag{170}$$

where $\hat{\boldsymbol{r}}$ is a unit vector in the radial direction, $\boldsymbol{S}(\xi)$ is given by eq. 9, and the angle brackets denote either a spherical average, or an azimuthal average in the mid-plane.

The correction force term can be thought of as arising from a virtual rigid mass component, $M_v(r) = -r^2 a_c(r)/G$. The work done by this force needs to taken into account when checking energy conservation (§13.1.2), which is most easily done by computing the potential of this virtual mass distribution. Thus

$$\Phi_c(r_j) = \Phi(r_{n_T}) + \int_{r_j}^{r_{n_T}} a_c(r')\, dr' \qquad \text{with} \qquad \Phi(r_{n_T}) = -\frac{G M_v(r_{n_T})}{r_{n_T}}, \tag{171}$$

which assumes its density is zero outside the last given point.

An approximate expression for the self-energy of this virtual component is

$$W_c \simeq -\frac{1}{2} \sum_{j=1}^{n_T} \left[ r_j^2 a_c(r_j) - r_{j-1}^2 a_c(r_{j-1}) \right] \frac{\Phi_c(r_j) + \Phi_c(r_{j-1})}{2}. \tag{172}$$

Sections 16.1.1 and 16.1.2 below list available disk and halo types whose properties are pre-progammed. The user can select up to five of these to be super-posed to construct a multi-component galaxy model. Most distribution function types listed in §16.1.3 will yield equilibrium models for only single component galaxies.

## 16.1. **Details of each component for the ASCII input file**

§5.1 explained how the mass model is specified in the ASCII `.dat` file. The first line B1, specifies the number of separate mass components, disk, bulge, halo, *etc.*, regardless of whether they will be rigid or composed of particles.

In the following expressions, the unit of mass is $M$ and of length is $\ell$ and these quantities are defined in §2.3.

Each component requires a group of input lines that must always begin with a line with the keyword disc, which has a single toggle parameter: T for a disk, F otherwise.

In detail, the subsequent lines for each component are as follows.

### 16.1.1. *Disks*

An outer taper line is needed for any disk of infinite extent, which always has to be truncated. Inserting F for the taper results in an abrupt truncation at the outer radius specified a couple of lines above, but it recommended that the density be tapered to the edge, and the parameter of the taper is the radius at which the surface density begins to be tapered is a cubic polynomial to zero at the above specified edge.

Selecting DF type = none works for all disk types. If this option is selected for an active disk of particles, then program `mkpcs` will select radii at random from the desired surface density, and assign velocities using the Jeans equations (§15.6) in the total potential. Using a DF would result in a superior initial set-up, and this option is available where noted and further information is provided in §16.1.3.

1. NONE: never useful for simulations
2. KT : Kuzmin's disk or model 1 of Toomre (1963)
$$\Sigma(R) = \frac{Mf}{2\pi} a (R^2 + a^2)^{-3/2}. \tag{173}$$

```
type     kt          # type keyword
mass     0.5         # the desired mass fraction f in the above expression
scale    1.0   5.0   # length scale a in units of ℓ and truncation radius in units of a
dftype   none  1.5   # DF type and parameter or initial Q. Other DFs are available.
taper    t     4.0   # toggle for outer taper and its start radius in units of a
```

3. SOFT: Softened Kuzmin-Toomre disk (Athanassoula & Sellwood 1986). The surface density is the same as for the KT disk (above) but the mid-plane potential is
$$\Phi(R) = -\frac{GMf}{[R^2 + (a+\epsilon)^2]^{1/2}} \tag{174}$$

```
type     soft  0.3   # type keyword with parameter ϵ/a
mass     1.0         # the desired mass fraction f in the above expression
scale    1.0   5.0   # length scale a in units of ℓ and truncation radius in units of a
dftype   none  1.5   # DF type and parameter or initial Q. Other DFs are available.
taper    t     4.0   # toggle for outer taper and its start radius in units of a
```

4. MFK : Uniformly rotating Maclaurin-Freeman-Kalnajs disk
$$\Sigma(R) = \frac{3Mf}{2\pi a^2} \sqrt{1 - (R/a)^2}. \tag{175}$$

It has a finite radial extent, so no truncation radius or taper is needed.

```
type     mfk         # type keyword
mass     1.0         # the desired mass fraction f in the above expression
scale    1.0         # length scale a in units of ℓ
dftype   none  1.5   # DF type and parameter or initial Q. Other DFs are preferable for the analytic harmonic
                     # potential, i.e. if supplementary forces (§15.10) are included to correct for softening.
```

5. MTZ : Infinite V=const (Mestel-Toomre-Zang) disk

$$\Sigma(R) = \frac{f v_c^2}{2\pi G R} = \frac{fM}{2\pi a R}, \tag{176}$$

where $a$ is a reference radius and $fM$ is the mass interior to $a$.

```
type    mtz          # type keyword
mass    1.0          # the desired mass fraction f in the above expression
scale   1.0   20.0   # length scale a in units of ℓ and truncation radius in units of a
dftype  none   1.5   # DF type and parameter or initial Q. The Zang DF is also available, but requires
                     # different tapering instructions – see case 5 in §16.1.3
taper   t     18.0   # toggle for outer taper and its start radius in units of a
```

6. EXP : Exponential disk (Freeman 1970)

$$\Sigma(R) = \frac{Mf}{2\pi R_d^2} e^{-R/R_d} \tag{177}$$

```
type    exp          # type keyword
mass    0.5          # the desired mass fraction f in the above expression
scale   1.0   5.0    # length scale R_d in units of ℓ and truncation radius in units of R_d
dftype  none  1.5    # DF type and parameter or initial Q. The Shu DF is also available.
taper   t     4.0    # toggle for outer taper and its start radius in units of a
```

7. SC : "Sc galaxy" model Sellwood & Carlberg (2014)

$$\Sigma(R) = 1.563 \frac{Mf}{a^2} \left(1 + \frac{20R}{a}\right)^{-3/4} \left(1 + \frac{R}{5a}\right)^{-2}, \tag{178}$$

```
type    sc           # type keyword
mass    0.7          # the desired mass fraction f in the above expression
scale   1.0   7.0    # length scale a in units of ℓ and truncation radius in units of a
dftype  none  1.5    # DF type and parameter or initial Q. The Shu DF is also available.
taper   t     6.0    # toggle for outer taper and its start radius in units of a
```

8. ISOC: Isochrone disk (Kalnajs 1976b)

$$\Sigma(R) = \frac{Mf}{2\pi a^2} \left[\ln \frac{R + R_*}{a} - \frac{R}{R_*}\right], \qquad \text{where} \qquad R_* = \sqrt{R^2 + a^2} \tag{179}$$

```
type    isoc         # type keyword
mass    1.0          # the desired mass fraction f in the above expression
scale   1.0   3.0    # length scale a in units of ℓ and truncation radius in units of a
dftype  none  1.5    # DF type and parameter or initial Q. Other DFs are available.
taper   t     2.5    # toggle for outer taper and its start radius in units of a
```

9. HUNT: The family of disks given by Hunter (1963).

$$\Sigma(R) = \frac{Mk(2k+1)}{2\pi a^2} \left[1 - (R/a)^2\right]^{k-1/2}, \tag{180}$$

where $k > 0$ is an integer, and $k = 1$ is for the MFK disk. This family of disks all have finite radial extent, so no truncation radius or taper is needed.

```
type    hunt    2    # type keyword, and the parameter is k
mass    1.0          # the desired mass fraction f in the above expression
scale   1.0          # length scale a in units of ℓ
dftype  none  1.5    # DF type and parameter or initial Q. The Shu DF is also available
```

10. SAND: This model has the same surface density as an exponential disk, but invokes the finite anti-gravity forces of a postulated Yukawa-like gravitational component as discussed by Sanders (1986).

11. POWE: Power-law disks (Evans 1994)

$$V_c = V_0 \left(\frac{R}{a}\right)^c \qquad \text{and} \qquad \Sigma(R) = f\Sigma_0 \left(\frac{R}{a}\right)^{2c-1}, \tag{181}$$

where $V_0$ is a velocity scale and $\Sigma_0$ is the full surface density both specified at the reference radius $a$, and $c$ is the power law index. If $c = 0$ is desired, then the MTZ disk above should be selected.

```
type    powe   -1.2   # type keyword, with the value of c as parameter
mass    1.0           # the desired mass fraction f in the above expression
scale   1.0    20.0   # length scale a in units of ℓ and truncation radius in units of a
dftype  none    1.5   # DF type and parameter or initial Q. The EVAN DF is also available, but requires
                      # different tapering instructions
taper   t      18.0   # toggle for outer taper and its start radius in units of a
```

12. COSI: Cosine annular disk. The parameter is the aspect ratio $0 < c < 1$, which is half-width $c$ of the hollow "cosine" annulus, of central radius $a$. Thus the surface density is

$$\Sigma(R) = \frac{cMf}{\pi^2 a^2}\left[1 - \left(\frac{Rc}{a}\right)^2\right]^{1/2} \tag{182}$$

It has a finite radial extent, so no truncation radius or taper is needed.

```
type    cosi   0.2   # type keyword, and the parameter the fractional length c/a
mass    1.0          # the desired mass fraction f in the above expression
scale   1.0          # length scale a in units of ℓ
dftype  none   1.5   # DF type and parameter or initial Q. The Shu DF is also available
```

13. GAUS: "Gaussian" disk or model $\infty$ of Toomre (1963)

$$\Sigma(R) = \frac{Mf}{2\pi\sigma^2} e^{-R^2/(2\sigma^2)}, \tag{183}$$

where $\sigma$ is a length and $R > 0$ only.

```
type    gaus          # type keyword
mass    1.0           # the desired mass fraction f in the above expression
scale   1.0    5.0    # length scale σ in units of ℓ and truncation radius in units of σ
dftype  none   1.5    # DF type and parameter or initial Q. The Shu DF is also available.
taper   t      4.0    # toggle for outer taper and its start radius in units of σ
```

14. FMES: Finite Mestel (1963) disk has the surface density

$$\Sigma(R) = \frac{Mf}{4aR}\left[1 - 2\frac{\arcsin(R/a)}{\pi}\right]. \tag{184}$$

It has a finite radial extent, so no truncation radius or taper is needed.

```
type    fmes          # type keyword
mass    1.0           # the desired mass fraction f in the above expression
scale   1.0           # length scale a in units of ℓ
dftype  none   1.5    # DF type and parameter or initial Q. The Shu DF is also available
```

15. ARBT: Disk specified in tabular form

16. RYBI: Rybicki disk (Evans & Collett 1993), or Toomre (1963) model 0

$$\Sigma(R) = \frac{Mf}{2\pi a\sqrt{R^2 + a^2}}. \tag{185}$$

```
type    rybi          # type keyword
mass    1.0           # the desired mass fraction f in the above expression
scale   1.0    5.0    # length scale a in units of ℓ and truncation radius in units of σ
dftype  none   1.5    # DF type and parameter or initial Q. Other DFs are available.
taper   t      4.0    # toggle for outer taper and its start radius in units of σ
```

17. SPLY: The family of polynomial disks studied by Sawamura (1988), parameter index $c$

$$\Sigma(R) = \frac{Mf}{a^2}\left[1 - (R/a)^2\right]^{1/2}\left[9 + c\frac{16(1-x^2) - 6}{6\pi(1 + 2c/5)}\right]. \tag{186}$$

They have a finite radial extent, so no truncation radius or taper is needed.

```
type    sawa   0.2   # type keyword, and the parameter is the index c
mass    1.0          # the desired mass fraction f in the above expression
scale   1.0          # length scale a in units of ℓ
dftype  none   1.5   # DF type and parameter or initial Q. The Shu DF is also available
```

18. PPLY: The polytropic discs introduced by Polyachenko & Polyachenko (1994) are not fully implemented

19. UNKN: Unknown type - properties not predictable. This is a useful option to indicate that the code does not know anything of the properties of the disk – *e.g.* when the input particles come from outside the world of GALAXY

```
type    unkn        # type keyword
mass    1.0         # an arbitrary value that is unused
scale   1.0         # an arbitrary value that is also unused
dftype  none  1.5   # DF type and an unused parameter. No DFs could be available
```

20. COMP: Composite exponential + Gaussian where did this come from?

$$\Sigma(R) = \frac{Mf}{\pi a^2}\left(\frac{e^{-x}}{2} + 0.9e^{-x^2}\right) \qquad \text{where} \qquad x = 3R/a \tag{187}$$

```
type    comp        # type keyword
mass    1.0         # the desired mass fraction f in the above expression
scale   1.0   5.0   # length scale a in units of ℓ and truncation radius in units of a
dftype  none  1.5   # DF type and parameter or initial Q. The Shu DF is available.
taper   t     4.0   # toggle for outer taper and its start radius in units of a
```

21. MTAB: M(r) table

22. DOTH: Model created by Rohlfs & Kreitschmann (1980), as the difference of 2 exp disks and used by Donner & Thomasson (1994)

$$\Sigma(R) = \frac{Mf}{1.5\pi a^2}\left(e^{-x} - e^{-2x}\right) \qquad \text{where} \qquad x = R/a \tag{188}$$

```
type    doth        # type keyword
mass    1.0         # the desired mass fraction f in the above expression
scale   1.0   5.0   # length scale a in units of ℓ and truncation radius in units of a
dftype  none  1.5   # DF type and parameter or initial Q. The Shu DF is available.
taper   t     4.0   # toggle for outer taper and its start radius in units of a
```

### 16.1.2. *Halos or bulges*

Some of these halo models are specified by a density profile, but others specify the rotation curve of the combined model.

1. NONE: never useful for simulations

2. ASDI: invokes a frozen spherical halo that has the same potential in the disk plane that would come from the already selected disk of the given mass.

```
type    asdi        # type keyword
mass    0.5         # the desired mass fraction, usually 1 − f selected for the disk
scale   1.0   5.0   # length scale a in units of ℓ that can only be that of the disk, the truncation radius is ignored
dftype  none        # No other DF is possible.
```

3. UNIS: uniform sphere

$$\rho(r) = \frac{3Mf}{4\pi a^3} \tag{189}$$

when $r \le a$. It has a finite radial extent, so no truncation radius is needed.

```
type    unis        # type keyword
mass    1.0         # the desired mass fraction f in the above expression
scale   1.0         # length scale a in units of ℓ
dftype  none        # No DF if rigid. The USPS DF is best for the analytic harmonic potential
                    # i.e. if supplementary forces (§15.10) are included to correct for softening.
```

4. PLUM: Plummer model (BT08), which is also a polytrope of index 5.

$$\rho(r) = \frac{3Mf}{4\pi a^3}\left[1 + (r/a)^2\right]^{-5/2} \tag{190}$$

```
type    plum        # type keyword
mass    1.0         # the desired mass fraction f in the above expression
scale   1.0   5     # length scale a in units of ℓ and truncation radius in units of a
dftype  none        # No DF if rigid. The DEJO family of DFs are available for the analytic potential
```

5. FE : Fall & Efstathiou (1980) specified a model through the total rotation curve of the disk plus halo

$$V(r) = V_0 \frac{r}{(r^2 + a^2)^{1/2}}, \tag{191}$$

where $V_0$ is the asymptotic circular speed when $r \gg a$, and $a$ is the core radius. The implied halo density can be determined only when the disk contribution is known.

```
type     fe      1.0   # type keyword and the value of V_0 in units of (GM/ℓ)^1/2
mass     1.0           # it seems this line is required even though the input value is ignored!
scale    1.0     5     # length scale a in units of ℓ and truncation radius in units of a
dftype   none          # No other possibility.
```

6. ISOT: simple cored 'isothermal'

$$\rho(r) = \frac{V_0^2}{4\pi G a^2} \frac{3 + x^2}{(1 + x^2)^2} = \frac{M_*}{4\pi a^3} \frac{3 + x^2}{(1 + x^2)^2}. \tag{192}$$

where $x = r/a$. Since $M_* \equiv aV_0^2/G$ or $V_0^2 = GM_*/a$, no mass need be specified.

```
type     isot    0.7   # type keyword and the value of V_0 in units of (GM/ℓ)^1/2
scale    0.5     20    # length scale a in units of ℓ and truncation radius in units of a
dftype   none          # No DF if rigid. Other DFs are possible
```

7. BSS : Bahcall *et al.* (1982) model for the Milky Way. This model is specified through a tabulated rotation curve

8. 3198: This model is specified through a simple parameterized fit to the observed rotation curve of NGC 3198.

9. JAFF: Jaffe (1983) model

$$\rho(r) = \frac{Mfa}{4\pi r^2 (a + r)^2}, \tag{193}$$

```
type     jaff          # type keyword
mass     1.0           # the desired mass fraction f in the above expression
scale    1.0     5     # length scale a in units of ℓ and truncation radius in units of a
dftype   none          # No DF if rigid. The MERR family of DFs is preferable for the analytic potential
```

10. KEPL: Kepler potential – the potential of a point mass $Mf$. No meanigful length scale or truncation radius can be defined, but the code still demands this line be present!

```
type     kepl          # type keyword
mass     1.0           # the desired mass fraction f in the above expression
scale    1.0     5     # meaningless input
dftype   none          # No DF possible
```

11. POWE: power law

12. RQUA: Since the $r^{1/4}$-law is for the projected surface density, the volume density is given by Abel inversion (see Young 1976, who implemented functions from the original formulae given by Poveda).

13. SCHM: modified Schmidt model for the Milky Way. This model is specified through a tabulated density profile

14. DFIT: iterative DF model having properties that are not describable analytically – see §15.4 for a full description

15. TEDS: Thomasson *et al.* (1990) – no longer available

$$M(r) = \frac{Mf}{0.59} \left[ \frac{0.5 r^3}{(a^2 + r^2)^{3/2}} + \frac{0.09 r^3}{(c^2 a^2 + r^2)^{3/2}} \right] \tag{194}$$

16. KING: Lowered isothermal functions (King 1966). These models are derived from an adopted form for DF and are fully described in BT08 (pp 307 - 311). The input parameter is $\Psi_0/\sigma^2$.

17. POLY: Spherical polytrope. These models are determined by the isotropic DFs and the models they yield are fully described in BT08 (pp 300 - 302). The input parameter is $n$, where $\rho = c_n \Psi^n$.

18. POWC: power law with a harmonic core - no longer available

19. KKSP: Kuzmin & Kutuzov (1963) oblate isochrone, which is also described in full detail by Dejonghe & de Zeeuw (1988). The parameter is the axis ratio $c/a$, and can be greater than or less than unity.

20. HERN: Hernquist (1990) model

$$\rho(r) = \frac{Mfa}{2\pi r(r+a)^3} \tag{195}$$

```
type    hern      # type keyword
mass    1.0       # the desired mass fraction f in the above expression
scale   1.0   5   # length scale a in units of ℓ and truncation radius in units of a
dftype  none      # No DF if rigid. The HERN DF is preferable for the analytic potential
```

21. UNKN: Unknown type - properties not predictable. This is a useful option to indicate that the code does not know anything of the properties of the halo – *e.g.* when the input particles come from outside the world of GALAXY

```
type    unkn      # type keyword
mass    1.0       # an arbitrary value that is unused
scale   1.0       # an arbitrary value that is also unused
dftype  none      # DF type. No DFs could be available
```

22. AGME: Aguilar & Merritt (1990) model

$$m(r) = fM \begin{cases} (r/a)^{3+c} & \text{if } r \le a \\ 1 & \text{if } r > a \end{cases} \tag{196}$$

where $a$ is the outer radius of the model and $c$ a parameter.

23. ADIA: adiabatically compressed halo – see §15.5 for a full description

24. ISOC: Hénon (1960) spherical isochrone (BT08)

$$\rho(r) = \frac{Mfa(a+2r_*)}{4\pi r_*^3(a+r_*)^2}, \qquad \text{where} \qquad r_* = \sqrt{a^2+r^2}. \tag{197}$$

```
type    isoc      # type keyword
mass    1.0       # the desired mass fraction f in the above expression
scale   1.0   5   # length scale a in units of ℓ and truncation radius in units of a
dftype  none      # No DF if rigid. The HNON DF is available for the analytic potenetial
```

25. NFW : Navarro *et al.* (1997) defined the broken power law density profile

$$\rho(r) = \frac{f\rho_s r_s^3}{r(r+r_s)^2} = f\rho_s \left[x(1+x)^2\right]^{-1}, \qquad \text{where} \qquad x = r/r_s \tag{198}$$

and $r_s$ is the break radius. We define a mass unit $M_* = 4\pi\rho_s r_s^3$.

```
type    nfw       # type keyword
mass    1.0       # the desired mass fraction f in the above expression
scale   1.0   20  # length scale r_s in units of ℓ and truncation radius in units of r_s
dftype  none      # No DF if rigid. The EDDI DF is preferable for the analytic potential
```

26. SISP: singular isothermal sphere (BT08)

$$\rho(r) = \frac{Mf}{4\pi r^2} \qquad \text{where} \qquad M = \frac{aV_0^2}{G} \tag{199}$$

with $a$ being a reference radius and $V_0$ is the circular speed.

```
type    sisp      # type keyword
mass    1.0       # the desired mass fraction f in the above expression
scale   1.0   10  # length scale a in units of ℓ and cutoff radius c in units of a
dftype  none      # No DF if rigid. The SISP DF is OK if there are no other mass components
```

27. VKA1: fit to Valenzuela & Klypin (2003) model $A_1$

28. MYNG: Miyamoto-Nagai flattened Plummer sphere or thickened Kuzmin disk (BT08)

29. MTAB: tabulated M(r)

30. EXPH: uses the mid-plane potential of the 2 component exponential disk invoked by (Donner & Thomasson 1994)

31. ISOG: cored isothermal with Gaussian cutoff (Hernquist 1993)

$$\rho(r) = \frac{Mf\alpha}{2\pi^{3/2}a}\frac{e^{-r^2/c^2}}{r^2+a^2}, \tag{200}$$

where $a$ and $c$ are respectively the core and cutoff radii, and $\alpha$ is given in eq. (2.3) of (Hernquist 1993).

```
type     isog          # type keyword
mass     1.0           # the desired mass fraction f in the above expression
scale    1.0     10    # length scale a in units of ℓ and cutoff radius c in units of a
dftype   none          # No DF if rigid. The JEAN DF is recommended by Hernquist
```

32. EINA: Einasto profile (Einasto & Haud 1989)

$$\rho(r) = \frac{fM}{16\pi r_s^3}\exp\left\{-\frac{2}{\kappa}\left[\left(\frac{r}{r_s}\right)^{\kappa}-1\right]\right\}, \tag{201}$$

where $\kappa$ is a parameter.

```
type     eina    0.7   # type keyword and the value of κ
mass     1.0           # the desired mass fraction f in the above expression
scale    1.0      5    # length scale r_s in units of ℓ and truncation radius in units of r_s
dftype   none          # No DF if rigid. The EDDI DF is preferable for the analytic potential
```

33. BURK: Burkert (1995) profile

$$\rho(r) = \frac{f\rho_0 r_0^3}{(r+r_0)(r^2+r_0^2)}. \tag{202}$$

We define $M = 4\pi r_0^3\rho_0$.

```
type     burk          # type keyword
mass     1.0           # the desired mass fraction f in the above expression
scale    1.0     5     # length scale r_s in units of ℓ and truncation radius in units of r_s
dftype   none          # No DF if rigid. The EDDI DF becomes negative for this density profile
```

34. CUBI: a finite radius bulge with a cubic density profile

$$\rho(r) = \frac{15Mf}{4\pi a^3}\left[\left(\frac{2r}{a}\right)^3-\left(\frac{3r}{a}\right)^2+1\right] \tag{203}$$

when $r \le a$. It has a finite radial extent, so no truncation radius is needed.

```
type     cubi          # type keyword
mass     1.0           # the desired mass fraction f in the above expression
scale    1.0           # length scale a in units of ℓ
dftype   none          # No DF if rigid
```

### 16.1.3. *Distribution functions*

It is preferable to select particles from a DF, when one is available or can be constructed by one of the methods described in §15. The Shu DF for disks, and bulge/halo DFs created by the iterative or compression methods described there generally yield good equilibrium models. But most of the other DFs that are listed below were derived for the exact Newtonian potential of some single component mass model, and will be in equilibrium only if the total attraction exactly matches that expected. In particular, it is likely that the attraction of an $N$-body disk will differ from the analytic function of a razor-thin disk model due to gravity softening, and perhaps also truncation and disk thickess. It is therefore vital, when adopting that DF, that this difference be corrected for by including supplementary forces, as described in §15.10.

1. NONE: useful when the component is rigid. When selected for a disk, it requires a parameter giving the desired $Q$ (ignored if the disk is rigid), which causes `mkpcs` to create an approximation to the desired warm disk by using the Jeans equations in the epicycle approximation.

2. KALN: A family of DFs for KT, isochrone, and perhaps other disks described by Kalnajs (1976b). The required parameter is the index $m$.

3. LIA : Generalized Kalnajs DFs for KT disks only (Athanassoula & Sellwood 1986), which needs two parameters: $m$ and $b$

4. MIYA: Miyamoto (1974) DF for KT disk only. The required parameter is the index $m$

5. ZANG: Zang DF for Mestel disk (Toomre 1977). The parameter specifies the desired value of Toomre's $Q$. Tapers for this DF are specified in a non-standard way:

```
dftype   zang   1.5   # type keyword, and desired initial Q
iind     4            # inner taper index
outt     6      15    # outer taper index and mean radius in units of a
```

6. OMEG: Kalnajs (1972) DF for his "Omega" models in the MFK disk. The required parameter is $\Omega/\Omega_0$, the mean angular rotation rate in units of the rate for circular orbits.

7. MERR: DF for a spherical Jaffe model given by (Merritt 1985). The parameter can be positive, negative or zero. When positive, the value is interpreted as the anisotropy radius, a zero value will yield purely radial velocities, while any negative value will yield an isotropic DF.

8. DEJO: General DFs for a Plummer sphere (Dejonghe 1987). The parameter 0 indicates an isotropic model, a positive value gives a radially biased DF, with 2 being the maximum allowed value. A negative parameter gives an azimuthally biased DF, with $-\infty$ giving pure curcular orbits.

9. LUCY: Generalised Newton-Binney method

10. NEIL: Any abitrary file of particles!

11. CPB0: The DF for composite Omega model $B_0$ for the MFK disk, shown by Kalnajs (1972) to have no instabilities.

12. KING: Lowered isothermal functions (King 1966). These isotropic DFs and the models they yield are fully described in BT08 (pp 307 - 311). The input parameter is $\Psi_0/\sigma^2$.

13. POLY: Spherical polytrope. These isotropic DFs and the models they yield are fully described in BT08 (pp 300 - 302). The input parameter is $n$, where $\rho = c_n \Psi^n$.

14. EVAN: Evans (1994) DFs for power law discs

15. DJDZ: Dejonghe & de Zeeuw (1988) DFs for Kuzmin-Kutuzov spheroids

16. EVCO: DFs for the Rybicki disk or Toomre (1963) model 0 (Evans & Collett 1993), parameter $n$

17. SAWA: Sawamura (1988) DFs for his polynomial discs

18. DFIT: Halo DF from `dfiter` – see §15.4 for a full description

19. PPLY: Polyachenko & Polyachenko (1994) lowered polytropic DFs with two parameters

20. HERN: Hernquist (1990) isotropic DF requires a parameter. A zero value is required for an isotropic model. Non-zero values are accepted, but anisotropic models have different densities

21. ADIA: adiabatically compressed halo – see §15.5 for a full description

22. HNON: Hénon (1960) spherical isochrone (eq. 4.54 of BT08)

23. EDDI: Eddington's inversion to obtain an isotropic DF for a spherical density distribution (BT08)

24. SISP: Singular isothermal sphere (BT08)

25. COMP: adiabatically compressed halo same as 21?

26. SHUE: Shu (1969) DF for a generic disk model in an arbitrary potential and for any disk density profile – see §15.7 for a full description

27. USPS: Polyachenko & Shukhman DF for a uniform sphere (BT08 problem 4.11)

28. JEAN: Hernquist (1993) velocities from the radial Jeans equation

### 16.2. **Adding extra options**

It is not difficult to add an extra mass component or DF that differs from all those listed above, if the user wishes. First, the user must increment the parameter giving the number of different types in the appropriate routine (`dtype.f`, `htype.f`, or `dftype.f`) and supply the additional acronym.

Then the program `genplt` should be recompiled, and the option tested. This program exercises every subroutine and function in the code that will need to be updated, and each will cause a graceful exit with a suitable message when it is called with the new option. The user should update each in turn and recompile as the program tries to execute.

Not all functions will fail, since the default $M(r)$ will attempt to integrate $\Sigma(R)$ or $\rho(r)$, *etc.* Where the mass distribution $M(r)$, potential $\Phi(r)$, radial attraction $F_r(r)$, circular speed $V_c(r)$, *etc.* can be calulated analytically, these functions should be updated to evaluate the analytic expressions.

There is a goof test, which checks that the circular speed is compatible with the central attraction, the gradient of the potential is the radial acceleration, the volume or surface density is the appropriately scaled derivative of the radial mass profile, *etc.*, providing cross checks of the added pieces of code. Finally, program `genplt` graphs many of the key properties of the model.

ACKNOWLEDGMENTS

The author wishes to thank all the many collaborators and graduate students with whom he has worked over the years, many of whom have made helpful suggestions to improve or add to the software. Especially noteworthy contributions have been made by my thesis advisor Richard James, who kindly made his Poisson solver code available for inclusion, and my students Neil Raha, David Earn, Victor Debattista, and especially Juntai Shen. The SCF option in the code originates from Hernquist & Ostriker (1992) and I am grateful for their permission to include some of their software. Joel Berrier and Juntai Shen have assisted by testing pre-release versions of the package. This work was also supported in part by NSF grant AST/1108977.

REFERENCES


Abramowitz, M. & Stegun, I. A. 1965, *Handbook of Mathematical Functions*, (Dover, New York)
Arfken, G. 1985, *Mathematical Methods for Physicists,* 3rd ed. (Academic Press, Orlando)
Aguilar, L. A. & Merritt, D. 1990, ApJ, **354**, 33
Athanassoula, E. & Sellwood, J. A. 1986, MNRAS, **221**, 213
Bahcall, J. N., Schmidt, M. & Soniera, R. M. 1982, ApJL, **258**, L23
Barnes, J. E. 2012, MNRAS, **425**, 1104
Barnes, J. & Hut, P. 1986, Nature, **324**, 446
Berrier, J. & Sellwood, J. A, 2016, ApJ, **831**, 65
Binney, J. 2010, MNRAS, **401**, 2318
Binney, J. & Tremaine, S. 2008, *Galactic Dynamics* (Princeton Univ. Press, Princeton) BT08
Burkardt, J. 2017, Website: `https://people.sc.fsu.edu/~jburkardt/`
Burkert, A. 1995, ApJL, **447**, L25
Casertano, S. 1983, MNRAS, **203**, 735
Clutton-Brock, M. 1972, Ap. Sp. Sci., **16**, 101
Clutton-Brock, M. 1972, Ap. Sp. Sci., **17**, 292
Debattista, V. P. & Sellwood, J. A. 2000, ApJ, **543**, 704
Dehnen W., 2009, MNRAS, **395**, 1079
Dehnen W., 2017, MNRAS, **472**, 1226
Dejonghe, H. 1987, MNRAS, **224**, 13
Dejonghe, H. & de Zeeuw, P. T. 1988, ApJ, **333** 90
De Lorenzi F., Debattista V. P., Gerhard O. & Sambhus N., 2007, MNRAS 376, 71 (DL07)
Donner, K. J. & Thomasson, M. 1994, A&A, **290**, 475
Earn, D. J. D. 1996, ApJ, **465**, 91
Earn, D. J. D. & Sellwood, J. A. 1995, ApJ, **451**, 533
Einasto J. & Haud U., 1989, A&A 223, 89
Evans, N. W. 1994, MNRAS, **267**, 333
Evans, N. W. & Collett, J. L. 1993, MNRAS, **264**, 353
Fall, S. M. & Efstathiou, G. 1980, MNRAS, **193**, 189
Freeman, K. C. 1970, ApJ, **160**, 811
Goldstein, H. 1950, *Classical Mechanics*, 1st ed. (Addison-Wesley; Reading Mass.)
Hénon, M. 1960, AnAp, **23**, 474 (see also BT08, eq. 4.54)
Hernquist, L. 1990, ApJ, **356**, 359
Hernquist, L. 1993, ApJS, **86**, 389
Hernquist, L. & Ostriker, J. P. 1992, ApJ, **386**, 375
Hockney, R. W. & Eastwood, J. W. 1981, *Computer Simulation Using Particles* (McGraw Hill, New York)
Hohl, F. 1971, ApJ, **168**, 343
Hohl, F. 1972, J. Comp. Phys.**9**, 10
Holley-Bockelmann, K., Weinberg, M. & Katz, N. 2005, MNRAS, **363**, 991
Hunter, C. 1963, MNRAS, **129**, 299
Inagaki, S., Nishida, M. T. & Sellwood, J. A. 1984, MNRAS, **210**, 589
Jackson, J. D. 1962, *Classical Electrodynamics,* (John Wiley, New York)
Jaffe, W. 1983, MNRAS, **202**, 995
James, R. A. 1977, J. Comp. Phys., **25**, 71
Jardel, J. & Sellwood, J. A. 2009, ApJ, **691**, 1300
Kalnajs, A. J. 1971, ApJ, **166**, 275
Kalnajs, A. J. 1972, ApJ, **175**, 63
Kalnajs, A. J. 1976a, ApJ, **205**, 745
Kalnajs, A. J. 1976b, ApJ, **205**, 751
Kalnajs, A. J. 1978, in IAU Symposium **77** *Structure and Properties of Nearby Galaxies* eds. E. M. Berkhuisjen & R. Wielebinski (Dordrecht:Reidel) p. 113
King, I. R. 1966, AJ, **71**, 64
Klypin, A., Gottlöber, S., Kravtsov, A. V. & Khokhlov, A. M. 1999, ApJ, **516**, 530
Kuijken, K. & Dubinski, J. 1995, MNRAS, **277**, 1341
Kuzmin, G. G. & Kutuzov, S. A. 1962. *Bull. Abastumani Ap. Obs.*, **14**, 52
LAPACK User's guide. Website: `http://www.netlib.org/lapack/lug/`
Malvido, J. C. & Sellwood, J. A. 2015, MNRAS, **449**, 2553
May, A. & James, R. A. 1984, MNRAS, **206**, 691
McGlynn, T. A. 1984, ApJ, **281**, 13
McMillan, P. J. & Dehnen, W. 2005, MNRAS, **363**, 1205
Merritt, D. 1985, AJ, **90**, 1027
Mestel, L. 1963, MNRAS, **126**, 553
Miller, R. H. 1976, J. Comp. Phys., **21**, 400
Miyamoto, M. 1974, A&A, **30**, 441
Monaghan, J. J. 1992, ARAA, **30**, 543
Monaghan, J. & Lattanzio, J. 1985, A&A, **149**, 135
Navarro, J. F., Frenk, C. S. & White, S. D. M. 1997, ApJ, **490**, 493
NIST Core Math Library *CMLIB*. Website: `http://gams.nist.gov/cgi-bin/serve.cgi/Package/CMLIB/`
Petersen, M. S., Weinberg, M. D. & Katz, N. 2016, MNRAS, **463**, 1952
Pfenniger, D. & Friedli, D. 1991, A&A, **252**, 75
Piessens, R., de Doncker-Kapenga, E., Ü berhuber, C, & Kahaner, D. 1983 "QUADPACK, A Subroutine Package for Automatic Integration" Springer-Verlag
Prendergast, K. H. & Tomer, E. 1970 AJ, **75**, 674
Press, W. H., Flannery, B. P., Teukolsky, S. A. & Vetterling, T. A. 1992, *Numerical Recipes* (Cambridge, Cambridge Univ. Press)
Polyachenko, V. L. & Polyachenko, E. V. 1994, *Pis'ma Astron. Zh.*, **20**, 491; English translation: *Astronomy Letters*, **20**, 416
Raha, N., *et al.* 1991, Nature, **352**, 411
Rodionov, S. A. & Sotnikova, N. Ya., 2006, Astron. Rep., **50**, 983
Rodionov, S. A., Athanassoula, E. & Sotnikova, N. Ya., 2009, MNRAS, **392**, 904
Rohlfs, K. & Kreitschmann, J. 1980, A&A**87**, 175
Romeo, A. B. 1998, A&A, **335**, 922
Rybicki, G. B. 1972, in IAU Colloq. **10**, *Gravitational N-body Problem*, ed. M. Lecar (Dordrecht: Reidel), 22
Saha, P. 1993, MNRAS, **262**, 1062
Sanders, R. H. 1986, MNRAS, **223**, 53
Sanders, J. L. & Binney, J. 2016, MNRAS, **457**, 2107
Sawamura, M. 1988, PASJ, **40**, 279

Schwarzschild, M. 1979, ApJ, **232**, 236
Swarztrauber, P. N. 1982, in *Parallel Computations*, ed. G. Rodrigue (Academic Press), p51
Sellwood, J. A. 1981, A&A, **99**, 362
Sellwood, J. A. 1983, J. Comp. Phys., **50**, 337
Sellwood, J. A. 1985, MNRAS, **217**, 12
Sellwood, J. A. 1996, ApJ, **473**, 733
Sellwood, J. A. 2003, ApJ, **587**, 638
Sellwood, J. A. 2006, ApJ, **637**, 567
Sellwood, J. A. 2013a, ApJL, **769**, L24
Sellwood, J. A. 2013b, in *Planets Stars and Stellar Systems*, v.**5**, eds. T. Oswalt & G. Gilmore (Heidelberg: Springer) p. 923 (arXiv:1006.4855)
Sellwood, J. A. 2014, arXiv:1406.6606
Sellwood, J. A. 2015, MNRAS, **453**, 2919
Sellwood, J. A. 2020, MNRAS, **492**, 3103
Sellwood, J. A. 2024, MNRAS, **529**, 3035
Sellwood, J. A. & Athanassoula, E. 1986, MNRAS, **221**, 195
Sellwood, J. A. & Carlberg, R. G. 2014, ApJ, **785**, 137
Sellwood, J. A. & Debattista, V. P. 2006, ApJ, **639**, 868
Sellwood, J. A. & Debattista, V. P. 2009, MNRAS, **398**, 1279
Sellwood, J. A. & Evans, N. W. 2001, ApJ, **546**, 176
Sellwood, J. A. & McGaugh, S. S. 2005, ApJ, **634**, 70
Sellwood, J. A. & Merritt, D. 1994, ApJ, **425**, 530
Sellwood, J. A., Shen, J. & Li, Z. 2019, MNRAS, **486**, 4710
Sellwood, J. A. & Valluri, M. 1997, MNRAS, **287**, 124
Shen, J. & Sellwood, J. A. 2004, ApJ, **604**, 614
Shen, J. & Sellwood, J. A. 2006, MNRAS, **370**, 2
Shu, F. H. 1969, ApJ, **158**, 505
Solway, M., Sellwood, J. A. & Schönrich, R. 2012, MNRAS, **422**, 1363
Springel, V. 2005, MNRAS, **364**, 1105
Stadel, J. G. 2001, PhD. thesis, University of Washington.
Syer D. & Tremaine S., 1996, MNRAS 282, 223 (ST96)
Thomasson, M., Elmegreen, B. G., Donner, K. J. & Sundelius, B. 1990, ApJL, **356**, L9
Toomre, A. 1963, ApJ, **138**, 385
Toomre, A. 1964, ApJ, **139**, 1217
Toomre, A. 1977, ARAA, **15**, 437
Valenzuela, O. & Klypin, A. 2003, MNRAS, **345**, 406
van Albada, T. S. 1982, MNRAS, **201**, 939
van Albada, T. S. & van Gorkom, J. H. 1977, A&A, **54**, 121
Vasiliev, E. 2019, MNRAS, **482**, 152
Vasiliev, E. & Athanassoula, E. 2015, MNRAS, **450**, 2842
Villumsen, J. V. 1982, MNRAS, **199**, 493
Weinberg, M. D. 1999, AJ, 117, 629
White, S. D. M. 1983, ApJ, **274**, 53
Wilson, G. M. 2004, PhD. thesis, Australian National University
Young, P. 1976, AJ, **81**, 807
Young, P. 1980, ApJ, **242**, 1232
Yurin, D. & Springel, V. 2014, arXiv:1402.1623

Table 2
Schedule of releases

| Version | Release date | Description |
|---|---|---|
| 16.10 | Aug 16, 2023 | Shu DFs for disks now created with an improved algorithm |
| 16.10 | Mar 3, 2023 | improvements to the code to create Shu DFs for disks |
| 16.04 | Sep 14, 2022 | fixed 2 bugs in v16.0 |
| 16.0 | Jun 1, 2022 | added acceleration-dependent criterion for time step |
| 15.4 | May 25, 2020 | fixed bug in s3d forces in guard region |
| 15.34 | Jul 20, 2019 | |
| 15.33 | May 18, 2019 | |
| 15.32 | Feb 4, 2019 | |
| 15.31 | Jan 2, 2019 | |
| 15.22 | Aug 6, 2018 | |
| 15.21 | Feb 10, 2018 | |
| 15.10 | Apr 12, 2017 | |
| 15.01 | Mar 23, 2017 | Use of commercial software eliminated |
| 14.52 | Dec 20, 2016 | |
| 14.511 | Nov 10, 2016 | |
| 14.5 | May 13, 2016 | Most array sizes now set at run time |
| 14.10 | Sep 19, 2014 | |
| 14.01 | Jun 25, 2014 | Original release |

# Part IV
# Appendix

## 17. versions

The various versions and their release dates are listed, in reverse order, in Table 2. A brief description of the changes and the reasoning behind them follows.

### 17.1. **Version 16**

A number of changes have been made in version 16, most notably that that block time steps are decided adaptively, depending of the acceleration to be applied to each particle – see §11.2. Previous implementations specified a number of predefined time step "zones" that implicitly assumed the density profile of the model would not evolve much. A dimensionless constant must be specified to scale the criterion for when steps must be subdivided.

The structure of the editable ASCII file that describes the model and the simulation options has been changed slightly, so that the duration of the simulation and analysis intervals are no longer specified by the number of time steps, but by the physical evolution time. Also the user now inputs the longest time step to be used, which is subdivided as required, while the maximum allowed number of sub-divisions in the simulation is also pre-specified.

#### 17.1.1. *Changes at v16*

Principal changes at version 16.0

- added acceleration-dependent criterion for time steps
- changed the .dat file to input the longest time step that will used, which will be reduced by successive factors of two in either spatial zones or as required by accelerations
- input analysis time interval and duration of the simulation to be in units of time, not steps
- created a blkdat segment to initialize variables in common
- made more use of allocatable arrays

- made ptcls a 2D array
- a number of minor bug fixes (see revisions.v16.0 in the DOCS directory)
- updated most of the README files, which unfortunately contained a lot of misinformation in version 15.

#### 17.1.2. *Changes at v16.04*

The principal reason for this new release was to fix 2 bugs that had been introduced in v16.0.

- Eliminated partial solutions for the field on polar grids, since accel dependent zones are not spatially separate. This was a bug, introduced at v16, that would have caused large changes to the total energy
- Another recently introduced bug that is now fixed caused failure to restart simulations that used the hybrid code on multiple processors

  Other changes:

- Monitor and report out at intervals the min and max of the time step parameter actually experienced by the particles. The interpretation of this information is described in §11.2.1
- Extensively revised treatment of massless disk particles

#### 17.1.3. *Changes at v16.10*

This release features improvements to the derivation of a Shu DF for disks. Program `smooth` should be run interactively only if it is to create the DF for a disk by Shu's method using `dfShuc`, non-interactive runs are OK for other DFs. It creates the `.dft` file which is now used separately by `dfShut`. (Previously they were combined into an unweildy routine `dfShu`.) The improvements to `dfShuc` are described above in §15.7.2. New functions `QofR` and `taperR` are supplied to set default profiles that can be edited by the user, if desired.

A number of minor improvements to fix little bugs and to improve parts of the analysis software are also included.

#### 17.1.4. *Changes at v16.11*

The principal improvement is to the software to set up a DF by Shu's method. This part of the code has been completely rewritten - see the notes in §15.7.2 on p82. The new method is both faster and better than the old, but it still requires careful management as described in "Advice to the user" (§15.7.2.4).

This release also makes it possible to specify the run number on the command line for most of the executables in the package. If the run number is omitted, the code still prompts for it. analys can be invoked with multiple run numbers if it is desired to compare the evolution of the first mentioned run with others.

Other minor improvements are to fix little bugs and to improve parts of the analysis software as detailed in DOCS/revisions.v16.11

### 17.2. **Version 15**

The main improvement is the elimination of all commercial software. In particular, the code makes no use of the *NAG* library. Calls to routines in this commercial software package have been replaced by calls to open access software. In many cases, the substituted routine is **identical** to the *NAG* version, which was simply a front that passed the call to an open access routine from *QUADPACK* or *LAPACK* for example. In other cases, equivalents have been found from various sources, such as *FFTPACK*, *CMLIB*, or the valuable software portal maintained by Burkardt (2017).

All these substitutions have been tested in the various scripts supplied with the code, but there are bound to be options that have not yet been tested and the author would appreciate reports of software failures that need to be corrected.

#### 17.2.1. *Changes at v15.4*

Fixed a significant bug, found by Vance Wheeler, that caused large force errors inside the guard radius from the s3d when not all particles have the same mass. The self-force on a particle that was subtracted from s3d grid force was not scaled by the relative particle mass. This bug was fixed in this version.

### 17.2.2. *Changes at v15.34*

- The few mostly inconsequential bug fixes are listed in the file `DOCS/revisions.v15.34`
- Most subdirectories of `EXAMPLES` now have updated README files and standardized scripts

### 17.2.3. *Changes at v15.33*

- Fixed an array subscripting bug caused by different roundoff in gfortran from other compilers. This will have very occasionally, and at one random step only, caused the wrong accelerations to have been applied to a particle on 2D or 3D polar grids. I am grateful to Agris Kalnajs for reporting this bug.
- A few minor improvements to scripts.

### 17.2.4. *Changes at v15.32*

- Subroutine qsetup now ensures $\kappa$ is always real and no smaller than $\Omega$. Before it was possible that some initial velocities of particles near the edge of a heavy disk with no halo were NaN. I am grateful to Mahmood Roshan for reporting this bug.
- The Makefile now includes the compiler option in gfortran to terminate excecution on most floating point exceptions.

### 17.2.5. *Changes at v15.31*

- The arrays islist and s3rad, which were in common blocks admin and grids respectively are now allocatable, and therefore held in module aarrays. The local input buffer in psetup is now allocatable also
- The total energy (PE + KE) of active particles now includes contributions from all active mass components - previously it was the first two only - a book-keeping error only
- Other improvements and minor bug fixes in GALAXY:
  1. dumpp & restrt omitted array s3dmss for zone 1, which caused a temporary glitch energy in the total initial energy for s3d runs with multiple time step zones - a book-keeping error only
  2. The potential from an external perturber is now computed correctly another book-keeping error
  3. The forces from a generic perturber are now included properly
  4. The accelerations from a rigid halo now allow Einasto halos
  5. The forces acting on particles outside the s3d grid now decrease correctly as they recede from the boundary
  6. Centering of the grid is now skipped for Cartesian and direct methods
  7. Accelerations from s3d grid are computed only for particles that need them
  8. Fixed the failure of multiple if conditions when one could depend on an invalid subscript (compiler dependent)
  9. Allow c3d to be one of the hybrid grids
  10. Set sectors and harmonics to skip only once when two grids are used
- Changes in setup code:
  11. Using a polytrope or King model now works
  12. m(R) for a disk is computed from Sigma when the profile is tapered
  13. eliminated bug in smmass for hybrid grids
- Some minor improvements to plotting software

#### 17.2.6. *Changes at v15.22*

- The main improvement in this minor update is to the Shu DF, which now yields much better equilibria for heavy disks. In addition, it reports either success or aborts with a helpful message on failure.
- The user now has the option to skip creating mpi versions of the main programs. The single processor versions are still found in /progs and the Makefile in that directory no longer attempts to create mpi versions. The mpi versions have been moved to a separate directory /progs_mpi, which has its own Makefile
- The program smooth now asks whether to create equal mass particles *before* entering the number of particles to generate
- Subroutine msetup now reports more helpful diagnostics when there are errors in the input data
- Tabs in the .dat file should no longer cause read errors
- Two routines in to the _mpi version of galaxy were changed to fix little bugs, and a couple of analysis options have been improved

#### 17.2.7. *Changes at v15.21*

- This version incorporates better generation of random numbers, which now really are `real*8`.[17] Also, the user can now set the random seed, and the state of the random generator is saved and restored when a simulation is stopped and restarted – capabilities that were lost at v15.01.
- The program smooth will now allow files of equal mass particles to be created – another problem unearthed by Juntai Shen; thanks for his diligence.
- A real bug of an incorrect array subscript in subroutine `s3dacc0` has been fixed. Fortunately, it affected only simulations that employed guard radii on s3d grids, and its effect was minor.
- All other changes are minor and are mostly improvements to the plotting part of the code.

#### 17.2.8. *Changes at v15.1*

This update enables the use of the Shu (1969) DF for a disk in a compressed halo potential, which was not possible in v15.01.

#### 17.2.9. *Other changes at v15.01*

Several minor improvements are

- Addition of two alternative methods to recenter the grid (see §8.14). The original method (1) followed the particle centroid. The new methods are: (2) following the center of mass of a fixed number of the most tightly bound particles on each mass component, or (3) following the motion of an extra heavy particle initially at rest at the center. The performance of both these new methods seems inferior to that of method (1) but they are there for users who may wish to try them.
- Specification and handling of rigid, spherical external perturbers (see §12.1).
- Many improvements to the analysis package.
- Handling of Greek characters, super- and sub-scripts in the annotation of plots should no longer depend on the compiler used.

### 17.3. **Updates in v14.52**

The main improvement here is to the s3d code, as described in §8.10, in which the strategy for interpolation between grid rings was revised. It resulted in minor improvements to the accuracy of the resulting forces as well as executing a little faster. In addition, the treatment of special characters in the plotting software has been made consistent with the gfortran compiler.

[17] The f90 `random_ number` function was creating `real*4` values that were occasionally rounded to precisely zero. This was the root cause of a segmentation fault that Juntai Shen obtained when he employed very large particle numbers.

## 17.4. **Minor patch in v14.511**

This change fixes a few minor bugs in the `dfiter` program that failed when used for hybrid codes. Also: a bug in program `smooth` when multiple components have DFs, `loadup` did not properly initialize two moving centers, and the MPI version of `massum` had a bug for the hybrid option. New downloads of the package include these changes, but those already using v14.50 can download the software patch from `http://www.physics.rutgers.edu/~sellwood/updates.html`

## 17.5. **Changes included in v14.50**

Version 14.5 of the *GALAXY* package contains a number of significant improvements as summarized here. In addition, the names of the directories have been changed. As a consequence, there are some slight changes to the installation procedure.

### 17.5.1. *Easier setup*

A major change at this version is to separate more completely the part of the code related to setting up the initial particle distribution, from that used to compute the evolution. Most runs of `galaxy` need only a short ASCII file (`.dat`) to select parameters, and a binary (`.pcs`) file containing the particles. The file of particles can be created in one of three ways:

1. For those wishing to take advantage of the author's set up codes, the particle file can be created using the program `mkpcs`, as described in §4.1.
2. The user can create his/her own `.pcs` file, as described in §4.2.
3. Alternatively, for those with files of particles suitable for use in *GADGET-2*, I have provided an executable to convert them to the format required for *GALAXY*. This executable is described in §4.3.

Program `start` has been retired, and programs `begin` and `finish` have been renamed `pcs2dmp` and `dmp2pcs` for clarity of purpose. The reason that programs `pcs2dmp` and `dmp2pcs` are separate from `galaxy` is given in §6.4.

### 17.5.2. *Minor improvements to the* `.dat` *file*

The first argument to the analysis options `dvel` and `hvel` is now the number of radial bins *per scale length* of that component. (Previously it was the total number of radial bins, and the code made a sometimes inconvenient conversion.) Also the `frqs` anlysis option requires a parameter which is the number of values to be computed per scale length. (Previously, the radial spacing was the same as for the `dvel` bins, which was too limiting.)

### 17.5.3. *New capabilities in model creation*

Program `compress` has been improved and extended to enable equilibrium bulge plus halo models, in the presence of a disk or other component, to be set up. This is described in §15.5.

Restrictions on the choice of mass and length scale of any component (older versions of §15.5) have been eliminated. New mass components include the Einasto halo, and isotropic DFs for both that and the cored isothermal sphere created by Eddington inversion.

The code now offers a much improved procedure to set up an equilibrium disk with a significant velocity dispersion. This is achieved by determining a DF for the disk by the method outlined by Shu (1969) and implemented as described in §15.7. This new capability has not been tested very extensively, and may require tweaks in future.

### 17.5.4. *3D Cartesian grid*

I have included a new (f95) version of the Poisson solver on a 3D Cartesian grid that was kindly supplied by Richard James. This new version of his software eliminates the only remaining arrays that were dimensioned at compilation time in previous versions. Now all parts of the code use dynamically allocated arrays, which means that the executables provided place no restrictions on either the size of any grid or the number of particles that can be employed.

#### 17.5.5. *Less evident improvements*

Those who use the code as a black box might notice that v14.5 both requires a little less memory when multiple time step zones are employed and runs slightly faster, than v14.1. But anyone looking at the source code will see that it has been greatly improved in several respects: logical unit numbers are now assigned dynamically to ensure none could ever be muddled for two different files, some ugly features such as "`go to`"s and long calling argument lists have been eliminated. More significantly, two of the larger arrays have been broken into separate pieces making their use a little more understandable. One consequence of this improvement is that the compiler bug when -O2 optimization is selected for the *ifort* compiler should have been eliminated (thanks to Matthieu Portail for chasing that down).

#### 17.5.6. *Analysis*

User inputs have been improved somewhat. A list of available components is displayed when the user is asked to select one. Selecting first, last, and interval is now mostly done in time units, instead of number of time steps. An additional analysis option is to create color images of the non-axisymmetric density on polar grids – option `cntd`. The user also has the option to mark scales on some (but not yet all) plots in physical units.

#### 17.5.7. *Plotting*

Calls to *PGPLOT* routines can now be made from the analysis code without the need to interface with routines written by the author, although the older calls also work.

#### 17.5.8. *Bug fixes*

The approximate expression used in earlier versions to estimate the gravitational potential on the spherical grid has been replaced by the exact expression (*i.e.* not really a bug fix, but an improvement). Some minor bugs have been fixed that were unlikely to have mattered to most users.

### 17.6. **Changes included in v14.10**

#### 17.6.1. *Reduced use of NAG*

Some progress has been made to eliminate *NAG* calls when they are not really needed. If a mass component, either a disk or a halo, is described as unknown, keyword 'UNKN' (see §5.1), then a number of *NAG* calls in the set-up procedure and in the analysis package will be skipped, enabling the code to be used more easily by those who lack the *NAG* libary.

#### 17.6.2. *Generic perturber*

A generic perturber has been added as an option, which allows the user to program whatever external mass distribution is desired, as described in §12.1. All the particles in the code will experience the attraction of this perturber, and the CoM of the perturber will be moved in reaction to the cumulative sum of all the forces experienced by the particles. It is up to the user whether the model also contains frozen mass components that are nailed down, which may not make dynamical sense.

#### 17.6.3. *Supplementary correction forces*

§15.10 describes the reasons that a table of radial attractive forces may be needed to supplement the self-consistent attraction of the particles in order to maintain radial balance. This table is usually created by program `mkpcs` and is written out to an addition file `..stb`. This file is read by `pcs2dmp` and the table is passed on to program `galaxy` through the `.dmp` file. Under version 14.01, it could not be used by program `pcs2dmp`, making it very difficult for the user to create a substitute. The improvements here help a little:

- The table is saved in a `.stb` file by program `mkpcs` and read back by program `pcs2dmp`. It is therefore a little easier for the user to create his own file of tabulated values that can be read with a `.pcs` file. See §15.10 for instructions.
- The radii at which values were tabulated depended on the adopted grid. This has now been made independent of the grid.

### 17.6.4. *Bug fixes*

The version of subroutine `loadup` released in v14.01 contained a serious bug that prevented multiple populations of particles from being loaded properly into the simulation code. The bug is fixed in this version.

Other minor bugs that have been fixed, *e.g.* suppressing output except on the master node, *etc.*, are unlikely to have been noticed.

TABLE 3
UNIT CONVERSIONS

| Quantity | Internal | External | Unit |
|---|---|---|---|
| position | $\boldsymbol{x}^*$ | $\boldsymbol{x} = \boldsymbol{x}^*/\texttt{lscale}$ | $\ell$ |
| velocity | $\boldsymbol{v}^*$ | $\boldsymbol{v} = \boldsymbol{v}^*/(\texttt{lscale} \times \texttt{ts})$ | $v_{\rm dyn}$ |
| acceleration | $\boldsymbol{a}^*$ | $\boldsymbol{a} = \boldsymbol{a}^*/(\texttt{lscale} \times \texttt{ts}^2)$ | |
| time | $t^*$ | $t = t^* \times \texttt{ts}$ | $\tau_{\rm dyn}$ |
| mass | $m^*$ | $m = m^* \times \texttt{lscale}^3 \times \texttt{ts}^2$ | $M$ |
| potential | $\Phi^*$ | $\Phi = \Phi^*/(\texttt{lscale} \times \texttt{ts})^2$ | |

Conversion factors from internal units within the code (here denoted by an asterisk) to external units described in §2.3.