Developer's Guide for Nbody ICs

David Riethmiller

July 29, 2013

Contents

1 Introduction

The Nbody ICs code uses the Osipkov-Merrit (Osipkov, 1979; Merritt, 1985a,b) model for arranging a distribution of self-gravitating non-collisional particles, such as stars or dark matter. The code is based on the work of Kazantzidis et al. 2004, although with significant modifications. **It is assumed that the user has already read these papers.**

I recommend adding the directory containing the nbody code files to the IDL_PATH environment variable, since the typical use of this code is to operate in a working directory other than the code directory.

This code will also draw from the `MG5` library, which contains additional utilities not described here. I recommend that the user execute a unix `diff` command against my MG5 library, as I've modified some of the routines.

1.1 A Note on Units

The units used in this code are chosen such that the gravitational constant is approximately unity; that is, length is given in kpc, mass is given in units of 10^{10} solar masses, and time (although not relevant for initial conditions) would be in units of 4.7 Myrs. We also assume the scale radius used in the density computation is equal to 1.

2 Compilation

The code is compiled at the IDL user interface by executing the command:

```
.r NBODYwrapper
```

This compiles the wrapper program and all of its calls.

3 Starting the Code

The code is executed by calling NBODYwrapper, whose type definition is given here:

```
pro NBODYwrapper,$
   x,y,z,vx,vy,vz,mass,rho,$    ; these are variables, don't input numbers!
   nparticles,raniso,rmin,rmax,rdecay,rvir,totmass,$ ; must choose numbers.
   nfw=nfw,$                    ; alpha, beta, gamma set to NFW model
   hernquist=hernquist,$        ; alpha, beta, gamma set to Hernquist model
   jaffe=jaffe,$                ; alpha, beta, gamma set to Jaffe model
   moore=moore,$                ; alpha, beta, gamma set to Moore model
   plummer=plummer,$            ; alpha, beta, gamma set to Plummer model
   readin=readin,$              ; read in alpha, beta, gamma interactively
   readme=readme,$              ; provide [alhpa,beta,gamma] as array at run time
   rejcheck=rejcheck,$          ; watch graphical rejection check
   dfcheck=dfcheck,$            ; watch graphical rej check via distribution function
   kazfile=kazfile,$            ; overplot Kaz models
   skipanalysis=skipanalysis,$  ; skip the analysis plot at the end.
```

```
readdist=readdist,$             ; read in the distribution function from a file
calcrvir=calcrvir,$             ; calculate a value for the virial radius
conc=conc,$                     ; set the rvir using specified concentration
addpot=addpot,$                 ; provide an addiational gravitational potential
sendtext=sendtext               ; send a text message when complete
```

Explanation of parameters:

`x,y,z,vx,vy,vz,mass,rho`

Placeholder variables for cartesian position, cartesian velocity, particle mass, and particle density arrays. This line should be input verbatim.

`nparticles`

The number of Nbody particles to attempt.

`raniso`

The anisotropy radius. For completely isotropic, we input a large number.

`rmin, rmax`

The minimum and maximum radius to use. Since we'll be taking logarithms of these values, rmin must be greater than zero.

`rdecay, rvir`

The decay constant and virial radius. See Kazantzidis et. al, Equations 1, 2, and 3:

$$\rho(r) = \frac{\rho_0}{(r/r_s)^\gamma[1+(r/r_s)^\alpha]^{(\beta-\gamma)/\alpha}}, (r \leq r_{\rm vir}) \tag{1}$$

$$\rho(r) = \frac{\rho_0}{c^\gamma(1+c^\alpha)^{(\beta-\gamma)/\alpha}} \left(\frac{r}{r_{\rm vir}}\right)^\epsilon \exp\left(-\frac{r-r_{\rm vir}}{r_{\rm decay}}\right), (r > r_{\rm vir}) \tag{2}$$

$$\epsilon = \frac{-\gamma-\beta c^\alpha}{1+c^\alpha} + \frac{r_{\rm vir}}{r_{\rm decay}} \tag{3}$$

`totmass`

The desired total Nbody mass of the system.

`/nfw, /hernquist, /jaffe, /moore, /plummer, /readin, readme=[a,b,c]`

Exactly ONE of these options must be chosen in order to set the values of α, β, and γ in Kazantzidis Equations 1-3, thus establishing the shape of the density profile.

`/rejcheck, /dfcheck`

Watch a graphical representation of the rejection algorithm. This slows down the code execution significantly, so it is recommended only as a diagnostic tool.

`kazfile=[filename]`

During the analysis at the end, overplot data from the Kazantzidis models. These files are found in the directory "kazfiles" within the code directory. This option assumes that the user has chosen input parameters consistent with those in the models described within the Kazantzidis paper.

`/skipanalysis`

Skip the diagnostic analysis routine at the end of the code. The actual code results are still output as before; we just don't bother to plot the density, velocitiy dispersions, or anisotropy measure.

`readdist=[filename]`

Instead of computing the distribution function, we read in in from a file. This is particularly useful if we already know that the DF should be.

`/calcrvir`

Calculate a value for the virial radius using the critical density of the universe, overwriting the value provided at run time.

`conc=[value]`

We overwrite the virial radius using a given concentration. We assume the scale radius is 1, and compute via $conc = r_{vir}/r_{scale}$.

`/addpot`

This option tells the code that we intend to create an Nbody distribution within the static gravitational potential of yet another distribution. For the moment, it is assumed that this additional potential is an NFW model.

`/sendtext`

Set this option to send a text message to the phone number hardcoded at the bottom of NBODYwapper.pro when the code has finished running. **Please change it to YOUR phone number if you choose to use this option!** The format of the phone number is as follows:

XXXXXXXXXX@SUFFIX

where XXXXXXXXXX is the number, and the suffix depends on the carrier.

Carrier	Suffix
Verizon	vtext.com
AT&T	txt.att.net
Sprint PCS	messaging.sprintpcs.com
T-Mobile	tmomail.net

Table 1: Phone Number Suffix

Then the calling sequence, for an isotropic NFW distribution of 10,000 Nbody particles, between 10^{-5} and 100 kpc, with a decay transition of 15 kpc on the density's exponential tail, a virial radius at 30 kpc, a total mass of 50×10^{10} solar masses, would look like this:

```
NBODYwrapper, x, y, z, vx, vy, vz, mass, rho, 10000, 1.d10,
    1.d-5, 1.d2, 15., 30., 50., /nfw
```

4 Utilities

I've written some very useful utilities, which are implemented throughout this code. Before describing the code's physical pipeline, it is helpful to explain how some of these work.

4.1 Invertfunction

Because this code frequently "regrids" arrays, we need a program that will re-interpolate the values of those arrays at their newly defined points. In other words, given arrays `xold` and `yold,` and a new y-coordinate `ynew,` we want to return the corresponding new x-coordinate `xnew.`

In the file `invertfunction.pro`, there are two versions of basically the same routine, each of which executes a linear interpolation. The simpler version is called as:

```
invertfunction_old, xold, yold, ynew, xnew, suppress=suppress
```

where `xold` and `yold` are the input arrays of $y(x)$ data, `ynew` is the y-coordinate of the new point, and `xnew` is the x-coordinate of the new point, which we want to determine. (`/suppress` is an option that suppresses all diagnostic output to the terminal screen.)

Assuming that in the discreet order of the `yold` array, for the index i it is true that

$$y_{old}[i] < y_{old}[i+1] \tag{4}$$

(in other words, `yold` is monotonically increasing), the routine locates the index for which it is true that

$$y_{old}[i] \leq y_{new} \leq y_{old}[i+1] \tag{5}$$

Then we define a fraction μ such that

$$\mu = \frac{x_{old}[i+1] - x_{old}[i]}{y_{old}[i+1] - y_{old}[i]} \tag{6}$$

which makes the value of `xnew`

$$x_{new} = x_{old}[i] + \mu(y_{new} - y_{old}[i]) \tag{7}$$

This linear interpolation assumes first that `yold` is monotonic, second that the value of `ynew` falls within the bounds of the `yold` array, and third that we only want a single set of coordinates for `(xold, yold)`.

The other routine within `invertfunction.pro` checks explicitly for these assumptions. That is, a call to

```
invertfunction, xold, yold, ynew, xnew, plot=plot, suppress=suppress
```

will check to see if `yold` is monotonic (see section §4.2) and exit if not. It also looks to make sure the value of `ynew` is within the table limits; if not, we perform a linear *extrapolation* rather than *interpolation*. Finally, the input of `ynew` may itself be an array; we execute Equations 4-7 for every value of `ynew` until the array of `xnew` is populated. Enabling the option `/plot` will simply produce a graphical representation of the function inversion.

4.2 Monotonicity Check

This utility lives in the file `is_mono.pro` and checks to see if an array is monotonically increasing, monotonically decreasing, or non-monotonic. The function call looks like

```
is_mono, array, bad, monoinc, monodec, quiet=quiet
```

where `array` is the input array, and the other inputs are optional. `bad` is an integer that will equal 2 if the array is non-monotonic, and `monoinc` and `mondec` are each character arrays that carry either the value "yes" or "no". The option `/quiet` has the same function as the `/suppress` option above.

4.3 Normalization Density ρ_0

In order to specify the density profile in Equation 1, we need to compute a normalization density ρ_0 by integrating ρdV and normalizing to the target mass. We call the routine `calc_rhonorm` from the file `calc_rhonorm.pro`:

```
calc_rhonorm, alpha, beta, gamma, totmass, rmax, rvir, rdecay,
    rhonorm, test=test, dump=dump
```

All of the parameters are as defined above. The keyword `/test` will execute a comparison of a linear integration against a logarithmic integration, and `/dump` will output an ASCII file with arrays of radius, density, and unnormalized density.

4.4 timer

We can track the wall clock time elapsed while the code is running.

```
timer, date1, date2, elapsed, message, start=start, stop=stop,
    print=print, keepfile=keepfile
```

The timer is started or stopped with the obvious `/start` or `/stop` flags, and the `date1, date2, elapsed` values are placeholders to extract data from the routine. The `/print` flag will tell the routine to print the starting time, finishing time, elapsed time, and the string data stored in `message`. The `/keepfile` flag prints this same data to the file `timer.dat`.

4.5 adjust_function

This code frequently requires functions to be monotonic, and since it's very difficult to compute the functions perfectly, we need a way to tweak the almost-but-not-quite-right arrays into something usable. For example, a common problem is that the very last point in an array suffers from small-number division, and makes the function non-monotonic. Or, perhaps there exists a small kink in the array, where two components were joined less-than-smoothly. `adust_function` is an interactive routine designed to bandage imperfections in an otherwise usable array. (I should also note that this routine is very new and lacks some robustness in the user-interface; some patience is required.)

```
adjust_function, x, y
```

The function $y(x)$ is read in via the two array arguments provided to the routine, and the user is presented with a menu of options:

```
MENU:
1: smooth function
2: fit low end
3: fit high end
4: fit middle
5: replot with different options
6: isolate monotonic failure
7: continue
```

4.5.1 smooth function

This option applies the IDL `smooth` routine (with box size 20) iteratively five times to the entire `y` array, overplots the result in green, asks the user to accept or reject the smoothing, replots the finished array, and returns to the menu prompt.

4.5.2 fit low end

This option will extrapolate a linear fit to the low (left plot side) end of the array. Use the prompts to x-zoom in on the low (left) edge of the function, including only the points to be included in the fit (i.e. zoom in x such that "bad" points are not included), then choose the "fit" option. A green line will show the fit which will replace errant points, and the user is prompted to accept or reject the fit, then returned to the menu.

4.5.3 fit high end

This option does the same linear fit procedure as the low end fit, except at the other end of the array. The user is returned to the menu prompt.

4.5.4 fit middle

This option is similar to fitting the ends, except there are a few additional choices. The user must x-zoom into the region to be fitted. Once this region has been selected, the user can choose option 2 to *fit* the region to a line, option 3 to *replace* the region with a line fixed at the region's endpoints, option 4 to apply a smoothing to the region only, or option 5 to *replace* the region with a polynomial function between the region's two endpoints. The user is returned to the menu prompt.

4.5.5 replot with different options

By default, the function is plotted as `plot,x,y,psym=4,/ylog`. This option prompts the user for a different set of plotting instructions. Note that for the boolean flags such as `/ylog`, 0 represents "off" and 1 represents "on." The user is returned to the menu prompt.

4.5.6 isolate monotonic failure

This option will overplot red data points on top of the white data until it reaches a y-value that is non-monotonic when compared with the previously plotted red data. This is useful for isolating the region where one of the fitting options needs to be applied. The user is returned to the menu prompt.

4.5.7 continue

The user selects this option when the current state of the function is satisfactory, and the routine exits.

4.6 Derivatives

As is apparent from Kazantzidis Equation 7, the algorithm that will assemble this Nbody distribution requires the second derivative of an anisotropy-modified density with respect to the negative of the gravitational potential, $d^2\rho_Q/d\psi^2$. While Kazantzidis claims that this reduces to "simple quadrature, with no numerical differentiation required," our purposes are somewhat more complicated.

The gist of the math here, while avoiding all of the specific details, is to obtain the potential $\psi = -\Phi$ by integrating the Poisson equation with the density $\nabla^2\Phi = 4\pi G\rho$. Then we take derivatives $d\psi/dr$ and $d\rho/dr$, such that the first derivative is given by

$$\frac{d\rho}{d\psi} = \frac{\frac{d\rho}{dr}}{\frac{d\psi}{dr}} \tag{8}$$

and the second derivative is given by

$$\frac{d^2\rho}{d\psi^2} = \frac{d}{dr}\left(\frac{d\rho}{d\psi}\right) \div \frac{d\psi}{dr} \tag{9}$$

for each component of the density profile (before and after the virial radius), and then join them together.

There are several routines for calculating these derivatives...and none of them are perfect.

4.6.1 Single Generic Potential

If we're only dealing with one potential, the analytic math for equations 8 and 9 has been generalized for any value of α, β, γ in the routine `gen_derivs`, found in `gen_derivs.pro`. This routine allows for anisotropy in the velocity dispersion, and is one of several options called from inside `distributionfunction.pro`. The user may uncomment and explore the other options at will; we will explain only the `gen_derivs` routine here.

```
gen_derivs, r, rhonorm, raniso, rhoQ, psi, mofr, alpha, beta,
    gamma, rvir, rdecay, d1, d2, plot=plot, fix=fix
```

Here, `rhoQ` is the density profile modified for anisotropy $\rho_Q = \rho(r)(1 + r^2/r^2_{\rm aniso})$, and `mofr` is an array containing the total enclosed mass as a function of radius. `/plot` does the obvious thing and plots the derivatives, and `/fix` shifts part of the first derivative curve in order to force monotonicity, although it may leave a kink in the curve.

Documentation for the math that goes into the derivative calculation for $r \leq r_{\rm rvir}$ can be found in the file `Gen_Derivs_Dens.pdf`, while the corresponding math for $r > r_{\rm rvir}$ (that is, the exponential density tail) can be found in `Gen_Derivs_Expo.pdf`.

When the `gen_derivs` routine finishes (and the `/plot` flag is given, which should really just be the case always), the user is presented with a set of options to select one set of derivatives, or to apply the `adjust_function` routine to one of them. The completely analytic derivatives are plotted in white, numeric derivatives are plotted in red, and the blue curve shows the numeric second derivative of the analytic first derivative. None of them are perfect as-is (and can, in fact, suffer from asymptotic behavior in the numerical calculations). After applying `adjust_function`, second derivatives depending on newly-changed first derivatives are recalculated, and the set of derivatives replotted. When the user is satisfied with the shape of the derivatives, he may choose the option 0, 1, or 2 to continue.

4.6.2 Multiple Potentials

The routine for calculating derivatives when there are multiple potentials to be considered is located in the file `added_derivs.pro` and called directly from the `NBODYwrapper` routine, if the `/addpot` option is provided:

```
if keyword_set(addpot) then begin
   print,'Enter/Re-Enter masses.'
   print,'Total NFW mass?'
   read,M_NFW
   print,'Total Hernquist mass?'
   read,M_hern
   added_derivs,r,rmax,rvir,rdecay,M_hern,M_NFW,deriv1,deriv2,potential
```

```
    goto,distfunc  ;; we skip calcuation of the potential, since we already have it
  endif
```

In this case, we assume that the only two density profiles of interest are the NFW profile and the Hernquist profile. We also assume that the desired distribution will be completely isotropic.

This routine splices together the two sections of the combined density profile (before and after the virial radius, see Equations 1 and 2) and guides the user through an interactive process to determine the derivatives. If numerical derivatives are sufficient, we continue; otherwise more complicated analytical math is used to compute the derivatives, which are likely still imperfect. In this latter case, further tweaks such as smoothing or fitting become available in the interactive menu via the `adjust_function` routine.

This routine approximately follows the math outlined in the document `Multiple_Potentials.pdf`, although with some minor deviations.

5 Pipeline

The pipeline starts with the call to `NBODYwrapper`, as in Section §3. Immediately the parameters α, β, γ are determined, the virial radius $r_{\rm vir}$ is set, and the normalization density is determined (see Section §4.3). Then the real work begins.

5.1 radialgrid

```
radialgrid,rmin,rmax,npoints,r,logr
```

We establish a grid of points in radius r, equally spaced in $\log r$, between `rmin` and `rmax`, having `npoints` points.

5.2 densityprofile

```
densityprofile,alpha,beta,gamma,rhonorm,rvir,r,density,c,rdecay
```

We calculate a radial density profile $\rho(r)$ on the grid of r values set up in the previous step. The model density profile has the form of Equation 1 interior to the radius $r_{\rm vir}$, and the form of Equation 2 exterior to $r_{\rm vir}$. Once this profile is established, the utility `adjust_function` is called to allow the user to bandage imperfections. (In most cases, this is unnecessary.)

5.3 massprofile

```
massprofile,r,logr,rho,m_of_r,diverge,alpha,beta,gamma,rhonorm
```

We integrate the density profile from the previous section to obtain the total mass enclosed within radius r, `m_of_r` (also used interchangeably with `mofr`.) This routine assumes a uniform grid in $\log r$ and a power-law density into the center. The mass at the innermost radial point (remember that setting `rmin` to zero creates a singularity) is obtained by rewriting Equation 1 under the approximation of small r, and $\gamma < 3$:

$$\rho(r) \approx \rho_0 (r/r_s)^{-\gamma} \tag{10}$$

and integrating between zero and `rmin` to get the mass:

$$M(r_{\rm min}) = \int_0^{r_{\rm min}} \rho(r) dr = 4\pi\rho_0 r_s^\gamma \frac{r_{\rm min}^{3-\gamma}}{3-\gamma} \tag{11}$$

The mass profile is then integrated iteratively outward. This stage dumps a diagnostic data file called `massprofile.dat`, containing columns `r` and `mofr`.

5.4 potential

If we are not using multiple potentials (see Section §4.6.2), then the gravitational potential $\Phi(r)$ needs to be determined. If one of the known model keywords is set (i.e., `/hernquist` or `/nfw`), then the form of the potential is known analytically. Otherwise, we call

```
potential,r,logr,m_of_r,potential
```

which will integrate the the force law $f = GM(r)/r^2$ from the outermost radial point inwards, assuming the density is zero beyond `rmax`. This routine also assumes logarithmic spacing in r. From this point on, `phi = -potential`.

5.5 distributionfunction

If we are using a single Hernquist potential, the distribution function is known analytically. Otherwise, we call

```
distributionfunction, rho, phi, newlogrho, newlogphi, df, r,
    raniso, alpha, beta, gamma, rvir, mofr, c, rdecay, rhonorm,
    totmass, deriv1, deriv2, addpot=addpot
```

The first thing we do is to ***regrid*** `phi`, `r`, `mofr`, and `rhoQ`

$$\rho_Q = \rho(r)(1 + r^2/r_{\rm aniso}^2) \tag{12}$$

to be equally spaced in log `phi`, instead of log `r`. Then we evaluate the derivatives of `rhoQ` with respect to `phi` using the `gen_derivs` routine from Section §4.6.1. We can then compute the Osipkov-Merrit distribution function given in Kazantzidis Equation 7:

$$f(Q) = \frac{1}{\sqrt{8}\pi^2} \left[\int_0^Q \frac{d^2\rho_Q}{d\psi^2} \frac{d\psi}{\sqrt{Q-\psi}} + \frac{1}{\sqrt{Q}} \left(\frac{d\rho_Q}{d\psi} \right)_{\psi=0} \right] \tag{13}$$

As Kazantzidis points out, for any well-behaved function, the second term should vanish, leaving

$$f(Q) = \frac{1}{\sqrt{8}\pi^2} \left[\int_0^Q \frac{d^2\rho_Q}{d\psi^2} \frac{d\psi}{\sqrt{Q-\psi}} \right] \tag{14}$$

This is the actual distribution function that we compute. First we calculate the integral itself, via trapezoidal integration. `deriv2` is the entire array of $\frac{d^2\rho_Q}{d\psi^2}$, and `d2` isolates only the section of that derivative between 0 and the current value of ψ. `diff` is the quantity under the square root, and the integral is done in $d\log\psi$.

```
for i=1L,n-1 do begin
   if (i mod 100 eq 0) then print,i,'of '+strcompress(n,/remove_all)

   d2 = deriv2[0:i-1]
   diff = Q[i] - newphi[0:i-1]
   for j=1L,i-1 do begin      ; trapezoidal rule integration
      df[i]=df[i]+(d2[j]/sqrt(diff[j])*Q[j] $
                   +d2[j-1]/sqrt(diff[j-1])*Q[j-1])/2.d0*dlogpsi
   endfor
   df[i]=df[i]+(deriv2[i]+deriv2[i-1])*sqrt(Q[i]-Q[i-1])
   if not finite(df[i]) then stop
endfor
```

There's just one catch: because Kaz Equation 7 (our Equation 13) assumes a continuous distribution that starts at `psi` = 0, and our distribution of `psi` values is discrete and non-zero, we need to apply a small correction to the resulting distribution function. It is likely that this correction is negligible, but we compute it anyway.

```
DF_correction, q, df
```

The correction is approximated by extrapolating a line backwards using the second and third points in the distribution function in order to correct the first point. This shift is then applied to the entire function.

Finally, we multiply the entire array by the prefactor, $1/\sqrt{8}\pi^2$, and feed into the `adjust_function` routine for the user to bandage any imperfections. **It is imperative that the distribution function be monotonic.**

This routine dumps a diagnostic file called `distfunc.dat`, which contains columns `df` (the value of the distribution function) and `q` (the value of psi).

5.6 particleload

This routine will create an Nbody realization of the spherical system specified by the radial grid `r` and `logr`, the mass profile `mofr`, the [negative of] potential `psi`, and the distribution function `df` tabulated on the grid `q` of values

$$Q = E - \frac{L^2}{2r^2_{\rm raniso}} \tag{15}$$

where E is the binding energy and L is the angular momentum.

First we space particles uniformly in enclosed mass to determine the radii, assuming equal mass per particle. We set up the masses linearly between zero and the maximum extent of `mofr`. Next we choose the angular positions randomly; at this point, we've completely determined the positions of the particles, all that is left is to set the kinetic energies.

We note that the maximum radial velocity a particle can have is its escape velocity from the system,

$$v_{r,\max} = \sqrt{2\psi_0} \tag{16}$$

where ψ_0 is the negative of the particle's gravitational potential (which is itself negative, making ψ_0 positive and the square root real). If the system has a strongly anisotropic velocity dispersion, there exists also a tangental escape velocity for the θ and ϕ directions:

$$v_{\theta,\phi,\max} = \sqrt{\frac{2\psi_0}{1 + r^2/r^2_{\rm aniso}}} \tag{17}$$

Note that in the limit of large anisotropy radius, Equations 16 and 17 are identical.

To determine the particle energies (and thus their velocities), we use the following steps for each particle:

1. Choose the components of the test velocity randomly, within the bounds of the escape velocities:

$$0 \leq v_r \leq v_{r,\max} \tag{18}$$

$$0 \leq v_\theta \leq v_{\theta,\max} \tag{19}$$

$$0 \leq v_\phi \leq v_{\phi,\max} \tag{20}$$

2. For ease of understanding we convert spherical to cartesian velocities:

$$[v_r, v_\theta, v_\phi] \longmapsto [v_x, v_y, v_z] \tag{21}$$

3. Compute the binding energy of the particle, given its potential and kinetic energies:

$$E = \psi_0 - \frac{v_x^2 + v_y^2 + v_z^2}{2} \tag{22}$$

4. Compute the angular momentum of each particle about [0,0,0]:

$$\vec{L} = \vec{r} \times \vec{v} \tag{23}$$

$$L^2 = \vec{L} \bullet \vec{L} \tag{24}$$

5. Then we want to choose and compare the values of $Q_{\rm test}$

$$Q_{test} = E - \frac{L^2}{2r^2_{\rm raniso}} \tag{25}$$

   against the value of ψ_0:

6. We use the distribution function $f(Q)$ (see Figure 1) to choose a random number w such that:

$$0 \leq w \leq f(\psi_0) \tag{26}$$

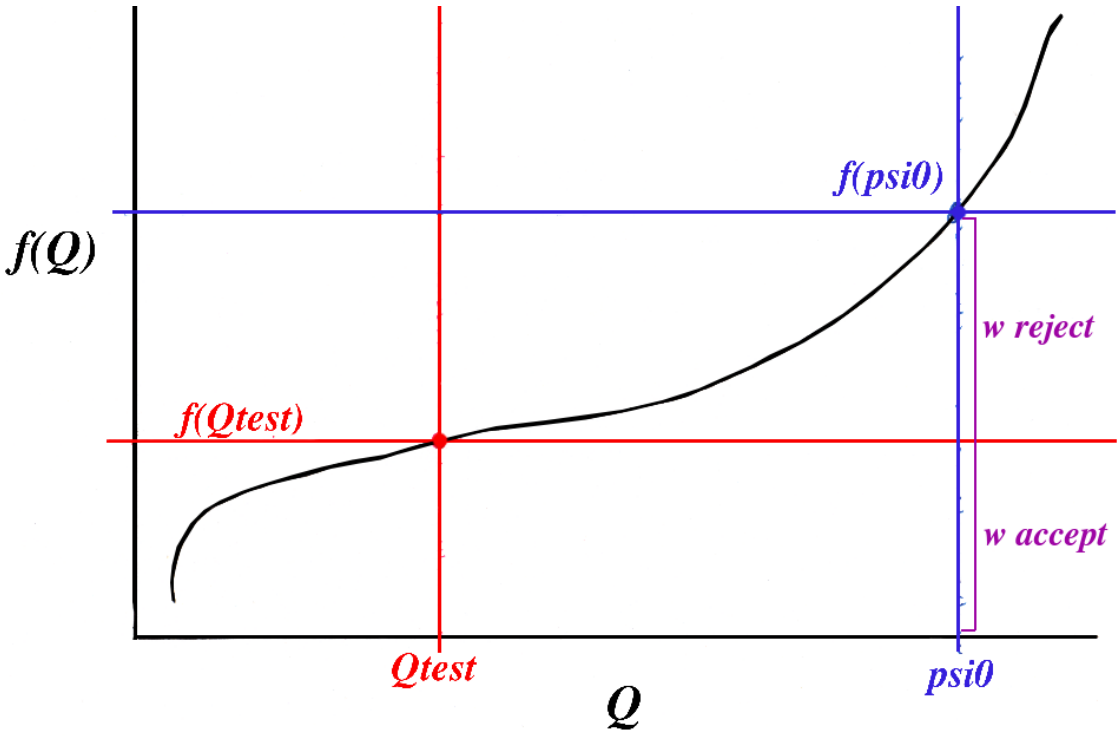

Figure 1: The distribution function $f(Q)$ used in a rejection algorithm

7. If it is true that

$$w \leq f(Q_{test}) \tag{27}$$

then we accept the chosen test velocities. Otherwise, we return to step 1 and try again.

These steps are executed for every particle. Once complete, the entire velocity distribution is renormalized to guarantee zero total linear momentum of the system.

There are some additional bells and whistles built into the `particleload` routine, such as displaying a real-time version of Figure 1 as particles are accepted/rejected, or putting a limit on the number of times the rejection loop may execute before giving up and moving on to the next particle, but the basic algorithm works as described here.

5.7 analyze_nbody

A file called `nbody.dat` is written, containing the newly computed particle data in columns `x,y,z,vx,vy,vz,mass,fake_pot`. This file is similar to the end result of the code, except that the last column, `fake_pot` is a placeholder for the actual gravitational potential at each point, which we'll compute later in Section §5.9.

Then we call the routine `analyze_nbody`:

```
analyze_nbody, alpha, beta, gamma, x, y, z, vx, vy, vz, mass,
    raniso, rhonorm, rvir, psplot=psplot, readin=readin, kazfile=kazfile,
    logbin=logbin, hernquist=hernquist, gadfile=gadfile
```

where `/psplot` will save the resulting plot to a postscript file, `/readin` will read in the particle data from the `nbody.dat` file, `kazfile=[filename]` will overplot data from one of the Kazantzidis models, `/logbin` will plot the density on a log scale rather than linear scale, `/hernquist` will overplot data from the known Hernquist model, and the `/gadfile` option reads in a GADGET2 format file, which you can probably just ignore.

This routine produces four plots in a single IDL window for analysis. In the upper left pane is plotted the density as a function of radius. We calculate the velocity dispersion, and plot the radial component σ_r in the upper right pane, and the tangental component σ_t in the lower left pane. Finally, we compute the actual and theoretical anisotropy measure

$$B_{\rm actual} = 1 - \frac{1}{2}\left(\frac{\sigma_t^2}{\sigma_r^2}\right) \tag{28}$$

$$B_{\rm theoretical} = \frac{r^2}{r^2 + r_{aniso}^2} \tag{29}$$

and plot them together in the lower right pane.

This routine may be called independently on a completed `nbody.dat` file; simply provide the `/readin` flag.

5.8 remove_nan

We read in the `nbody.dat` file. Compute $r = \sqrt{x^2 + y^2 + z^2}$, and rewrite the `nbody.dat` file including only those lines of data for which r is finite and non-zero. If everything previous has gone correctly, this step should be unnecessary.

5.9 write_potential

We read in the `nbody.dat` file and compute $r = \sqrt{x^2 + y^2 + z^2}$ as in the previous step. We use the `invertfunction` utility to interpolate the value of the gravitational potential at every point, and rewrite `nbody.dat` using this correct potential.

5.10 virialcheck

This routine is a check of how "stable" is the system we've created, or how close to virial equilibrium we've achieved. We calculate the total kinetic and potential energies:

$$PE = \frac{1}{2}\sum_i m_i \Phi_i \tag{30}$$

$$KE = \frac{1}{2}\sum_i m_i (v_x^2 + v_y^2 + v_z^2)_i \tag{31}$$

where m_i, Φ_i, and v_i are the mass, gravitational potential, and velocity of each particle, respectively. (In the case of added potentials, PE contains the *total* potential energy.) Then the test of stability is:

$$\frac{2KE}{|PE|} \approx 1 \tag{32}$$

We define the stability percentage as:

$$P = \left(1 - \left|1 - \frac{2KE}{|PE|}\right|\right) \times 100\% \tag{33}$$

5.11 recommend_eps

Finally, we compute the number of particles found within a given radius r, and the average particle spacing within that same r, for a range of values. This gives the user some recommendations for the optimal smoothing length, if the results of this code are to be used in an Nbody-SPH code.

6 End Result

The final result of this code is an ASCII file called `nbody.dat`, which contains the following:

1. A line of: `n_elements(x), tpos, iter, G`
   That is, the number of particles stored, the starting time (zero), the iteration number (zero), and the value of the gravitational constant used.

2. A line of: `alpha, beta, gamma, raniso, rhonorm, rvir`
   The parameters α, β, γ that determine the shape of the density profile, the anisotropy radius, the density normalization, and the virial radius.

3. Columns of particle data: `x, y, z, vx, vy, vz, mass, pot`
   The cartesian locations of the particles, the cartesian velocity components, the particle mass, and the gravitational potential at the particle location.

7 List of Files

Essential program files that comprise this code:

```
DF_correction.pro
NBODYwrapper.pro
add_potential.pro
added_derivs.pro
adjust_function.pro
advanced_derivs.pro
analyze_nbody.pro
builtin_derivs.pro
calc_rhonorm.pro
compare_derivs.pro
density.pro
densityprofile.pro
derivs.pro
distributionfunction.pro
extract_pots.pro
fit_functions.pro
fitme.pro
gen_derivs.pro
hern_dist.pro
hypergeom.pro
invertfunction.pro
is_mono.pro
kaz_derivs.pro
mod_curvefit.pro
model_derivs.pro
num_derivs.pro
particleload.pro
plotdf.pro
plummer_derivs.pro
polyfit_deriv.pro
potential.pro
radialgrid.pro
recommend_eps.pro
removenan.pro
return_model.pro
simple_derivs.pro
timer.pro
virialcheck.pro
virialrad.pro
write_potential.pro
```

* Those files whose programs are not explained in this user guide are likely routines that are employed in some of the alternate or antiquated derivative options, accessible in `distributionfunction.pro`.

In directory `kazfiles/`:

`kazD.dat`

`kazE.dat`

Non-Essential utilities:

`read_gadget.pro` - Read a Gadget2 file.

`rescale_system.pro` - Rescale the system. See `Notebook_Page_124.pdf`

`rescale_units.pro` - Rescale the units. See `Notebook_Page_125.pdf`

Documentation:

`Gen_Derivs_Dens.pdf`

`Gen_Derivs_Expo.pdf`

`Multiple_Potentials.pdf`

`Nbody_User_Guide.pdf`

`Notebook_Page_124.pdf`

`Notebook_Page_125.pdf`

8 Appendix A: Things That Can Go Wrong

1. Just to point out the obvious, the user must choose exactly ONE of the calling options `/nfw, /hernquist, /jaffe, /moore, /plummer, /readin, readme=[a,b,c]` at runtime, in order to define the shape of the profile. Error trapping is graceful in the event of failure here; the user is informed of the mistake and instructed to try again.

2. The most likely routine to crash is `adjust_function`. It is the newest routine written, and lacks a more robust user interface. If it does crash, one of the more probable causes is a bad plot range; follow the chain of logic choices inside of `adjust_function.pro` to the line that plots your choice, and examine the plot range given. Remember that any fitting or replacement that happens will use only the points currently plotted. For example, in order to perform a low-end fit on point zero using the first ten points, zoom the x-coordinate such that points one through nine are plotted, and point zero is excluded. Then the range will be replotted, with a green line showing how point zero will be adjusted.

   I will also note that the "poly replace" option is one that I rarely use, in favor more often of the "low end fit" or "middle fit," "linear replace," and some version of smoothing options. If prompted for a value for errors, I choose a reasonably small

number like `1.d-3`.

3. Aside from `adjust_function` issues, I'll note that things tend to get messy the more anisotropic the system attempted. For completely isotropic systems (large `raniso`), this code is fairly robust, with minimal amount of function bandaging required. However, the smaller `raniso` is set, the more creative the user may have to get in order to achieve a well-behaved monotonic distribution function...

4. ...which brings up another issue: the distribution function must be monotonic. Because the DF is used in the `invertfunction` interpolation routine during `particleload`, we need the DF to be monotonic. Check the output to the terminal screen; `adjust_function` calls `is_mono` after every adjustment. If monotonicity is not achieved when the user is adjusting the DF, and the code continues anyway, all bets are off.

5. If the second derivative of ρ_Q with respect to ψ (see section 4.6.1) fails to be monotonic, the distribution function will also likely be non-monotonic. Use `adjust_function` to fix both of these arrays.

9 Bibliography

Kazantzidis, S., Magorrian, J., & Moore, B. 2004, Astrophys. J., 601, 37

Merritt, D. 1985a, Mon. Not. R. Astron. Soc., 214, 25P

—. 1985b, Astron. J., 90, 1027

Osipkov, L. P. 1979, Pisma v Astronomicheskii Zhurnal, 5, 77