Using the Self-consistent Field Method to Compute the Gravity inside an N-body Dark Matter Halo

Wayne Ngan

March 15, 2016

Abstract

This document describes the code to quickly compute gravitational potential and accelerations ($\Phi(\mathbf{x})$ and $-\nabla\Phi(\mathbf{x})$, respectively) inside the N-body data of a dark matter halo. Given the positions and masses of the N-body particles, the self-consistent field (SCF) method solves the Poisson equation by basis decomposition. The basis coefficients are first pre-computed, and then they can be used to evaluate the linear sum of analytical basis functions. The code package includes the coefficients of the main Via Lactea II halo and its subhalos, a driver code to demonstrate how to load these coefficients and setup an orbit, as well as the routines to generate the coefficients if you have your own N-body data.

The SCF method was first developed by Hernquist & Ostriker (1992), and the code being documented here was used in Ngan et al. (2015) and Ngan et al. (2016).

Contents

1 Introduction

1.1 The SCF method

Given a snapshot of an N-body simulation of a dark matter halo, commonly obtained from cosmological simulations, how can we evaluate the gravity at arbitrary positions inside this halo? The naive way is to directly sum up the forces of every particle, but this is impractical for modern simulations with $N \gtrsim 10^9$. The self-consistent (SCF) method (Hernquist & Ostriker, 1992, hereafter HO92) is a much better way.

The problem is solving the Poisson's equation: $\nabla^2\Phi(\mathbf{x}) = 4\pi G\rho(\mathbf{x})$. The N-body data is $\rho(\mathbf{x})$, and we need to find $\Phi(\mathbf{x})$ and $-\nabla\Phi(\mathbf{x})$. The SCF method solves for Φ by basis decomposition such that

$$\Phi(\mathbf{x}) = \sum_{n=0}^{n_{max}} \sum_{l=0}^{l_{max}} \sum_{m=-l}^{l} A_{nlm}\Phi_{nlm}(\mathbf{x}) \tag{1}$$

where A_{nlm} are the basis coefficients, and $\Phi_{nlm}(\mathbf{x})$ are the basis functions. Because the basis functions are analytical, once we obtain A_{nlm}, the derivative $\nabla\Phi = \sum A_{nlm}\nabla\Phi_{nlm}$ can also be computed very easily. The central parts of the SCF method are choosing Φ_{nlm}, and how to compute A_{nlm}.

This code and document follows Sections 2 and 3 in HO92 very closely in terms of notation. Readers who are deeply interested in the code's implementation are encouraged to read HO92. In their derivation for Φ_{nlm} and A_{nlm}, the zeroth-order basis function is the Hernquist profile

$$\Phi_{000} = \frac{1}{1+r} \tag{2}$$

The higher order basis terms are essentially deviations from the Hernquist profile. The basis functions are complete so that Equation 1 can be used for any arbitrary mass distributions as $n_{max}, l_{max} \to \infty$. However, because of the choice of the zeroth-order basis, the method is best used for dark matter halos which are reasonably similar to the Hernquist profile to begin with (e.g. something spheroidal with a strong density peak at the centre, and not uniform in density like a tophat profile).

The basis coefficients A_{nlm} can be computed with Equation 2.33 from HO92 which is

$$A_{nlm} = \frac{1}{I_{nl}} \int \rho(\mathbf{r}) \left[\Phi_{nlm}(\mathbf{r})\right]^* d\mathbf{r} \tag{3}$$

where the * denotes complex conjugate, and I_{nl} is a constant that depends on n and l. In practice, this integral is a linear sum over all halo particles, so it can be easily parallelized for a large number of particles. This code package includes a set of these coefficients that can be readily used to compute orbits, as well as the routines to actually compute these coefficients using your own N-body data.

1.2 Code package

This package includes the following items:

- The SCF coefficients for the first 30 orders computed from the zero redshift snapshot of VL-2.
  (*data/host_smooth_scf30_30.scf*)
- The subhalo catalog extracted from the zero redshift snapshot of VL-2.
  (*data/subhalo_catalog.dat*).
- The SCF coefficients for the first 10 radial orders (n) for each subhalo listed in the subhalo catalog.
  (total of 11523 subhalos are in the *data/subhalo_scf/* directory)
- The code which evaluates Φ and $-\nabla\Phi$ at a given position using the SCF coefficients.
  (*scf_potential.c*, which requires *Phi_nl.c, gegenbauer.c, ind.c*)
- Driver code that demonstrates how to load the SCF coefficients and use them to compute the orbit and energy of a single test particle.
  (*driver.c*)
- Routines to generate the SCF coefficients using your own N-body data of a halo. These routines use MPI for the parallel functions, due to the size of modern halo simulations.
  (*scf.c* and *scf_spherical.c*, which require *scf_integral.c, Phi_nl.c, gegenbauer.c, ind.c, read_snapshot.c*)

Part I

Using the included VL-2 at $z = 0$ coefficients

2 Compiling and running the code

The code requires GSL. In the default *Makefile*, make sure to define GSL_INC and GSL_LIB variables for the include and library paths, respectively, if they are non-standard. After that, simply run `make` using that *Makefile*, and this will produce the *driver* executable. *driver* is a small demo of how to initialize and run the SCF code using one test particle on "orbit 1" in Ngan et al. (2015).

driver takes two required and one optional argument

```
driver nmax lmax Nsub
```

`nmax` and `lmax` are required. They are the order that you want to use in your decomposition. In Ngan et al. (2015), order $\equiv n_{max} = l_{max} = 10$. The included SCF coefficient file (see below) contains coefficients of up to $n_{max} = l_{max} = 30$, so this is highest order you can use.

`Nsub` is the optional number of subhalos to include in the simulation (by default, `Nsub=0`). The code reads in the first `Nsub` lines in the subhalo catalog which contains the initial positions and velocities of the subhalos so they can also orbit around the main halo. The code also reads in each subhalo's SCF coefficients. Subhalos don't interact with each other, but they interact with your test particle of interest. In other words, the net force on each subhalo is just $F_{net} = F_{main}$, but the net force on your test particle is $F_{net} = F_{main} + \sum F_{subhalo}$, where F_{main} is the force due to the main halo.

To select a particular subset of subhalos, simply modify the catalog so that it only contains those subhalos. You can also change the catalog file name constant SUBHALO_CATALOG to something else near the top of *driver.c*. See Section 4 below for the structure of the catalog.

The drive code outputs the t, x, y, z columns of the particle in "orbit 1" in Ngan et al. (2015).

Example

```
driver 10 10
```

This would use $n_{max} = 10$, $l_{max} = 10$ for the decomposition, even though it is reading the coefficient file that contains n_{max} and l_{max} up to 30.

```
driver 15 15 100
```

This would use $n_{max} = 15$, $l_{max} = 15$ for the decomposition for the main halo. And in addition, there will be 100 subhalos (defined by the first 100 lines in the subhalo catalog) orbiting around that main halo.

3 SCF Coefficient files

3.1 Coefficients for the main halo

The coefficient file *data/host_smooth_scf30_30.scf* contains the coefficients up to order 30 computed from the zero redshift snapshot of VL-2. In both Ngan et al. (2015) and Ngan et al. (2016), "order" $\equiv n_{max} = l_{max}$ in Equation 1. Note that A_{nlm} and Φ_{nlm} are both complex, but are implemented in real form (Equation 3.13 in HO92), where sine and cosine terms are separate. The sum in Equation 1 above would have $N_{terms} = (n_{max} + 1) \times (l_{max} + 1) \times (l_{max} + 2)$ terms. Note, however, the coefficient file contains more than N_{terms} numbers.

The file structure is as follows. The beginning of the file contains two integers: n_{max} and l_{max} for the coefficients that are being stored in the file. They should both be 30 in *data/host_smooth_scf30_30.scf*. These two numbers are also redundantly written in the file name.

After the two integers, the file contains the array for the cosine terms. This array consists of $(n_{max} + 1) \times (l_{max} + 1)^2$ double precision real numbers. *driver.c* shows how this array is read and referenced by pointer `mat_cos`, which is passed directly to `scf_potential()` as an argument.

After the array for the cosine terms, the file contains the array for the sine terms which is another $(n_{max} + 1) \times (l_{max} + 1)^2$ numbers. The structure of this array is identical to the one for the cosine terms, with the same number of elements and passed to `scf_potential()` as `mat_sin`.

With the two integers in the beginning and two coefficient arrays, the coefficient file is $2 \times 4 + 2 \times (n_{max} + 1) \times (l_{max} + 1)^2 \times 8 = 47664$ bytes in size.

If you are simply using the included coefficient file, then you don't need to worry about how these coefficient were calculated or implemented, since they are simply read from the file and are passed to `scf_potential()` right away. The implementation is explained in Section 8.2 below for those who wish to compute their own coefficients or understand the code more deeply.

Also note that both `mat_cos` and `mat_sin` are three-dimensional arrays, stored in an order such that n varies the most rapidly, followed by l, and then m. The `ind(n,l,m)` function included in *ind.c* conveniently converts three indices into the single index used to address the desired element in the array.

3.2 Coefficient files for subhalos

Each of the 11523 subhalos in the subhalo catalog has its own coefficient file in the *data/subhalo_scf/* directory. The structure for each subhalo coefficient file is similar but not the same as the one for the main halo. Each file also begins with two integers `nmax=10` and `lmax=0` (i.e. they are spherical), but the coefficients are stored in single precision. The reason is that for the purposes of Ngan et al. (2015) and Ngan et al. (2016), it was not necessary to model the subhalos accurately.

Because $l_{max} = 0$ (hence $m = 0$), the implementation in Equation 1 can be simplified a lot. Instead of using `scf_potential()` for the main halo, we used `scf_potential_spherical()` specifically for subhalos to skip the loops over l and m. And because $m = 0$, the coefficients for the sine terms in Equation 3.13 in HO92 are always 0, so they are never stored. **With the two integers in the**

beginning and only one coefficient array, each subhalo coefficient file is $2 \times 4 + (n_{max} + 1) \times 4 = 52$ bytes in size.

4 Subhalo catalog

The subhalo catalog (*data/subhalo_catalog.dat*) contains the physical parameters of the 11523 first-level subhalos (i.e. Only the immediate children of the VL-2 main halo, and not sub-subhalos) in the zero redshift snapshot of VL-2. The information stored in this text file are reported by the halo finder. The text file has 10 columns:

- subhalo ID
- Viriral mass (solar masses)
- position x (kpc)
- position y (kpc)
- position z (kpc)
- velocity v_x (km/s)
- velocity v_y (km/s)
- velocity v_z (km/s)
- Virial radius (kpc)
- scale radius (kpc)

where both the positions and velocities are halocentric. The subhalos are sorted by mass in the file, so it is easy to select a desirable mass range to include in the simulations.

Each subhalo is identified by its ID in the catalog. Note that the IDs are not sequential and have gaps between numbers. This is because these subhalos are only the first-level subhalos, and the ones deeper in the hierarchical structures are not included. The ID can be used to reference the corresponding coefficient file inside *data/subhalo_scf/*.

5 Implementation notes

The function `scf_potential()` is where Equation 1 above and its derivative (Φ and $-\nabla\Phi$) are evaluated. The function returns the Φ value and takes the following arguments

- `*a` is the placeholder array of 3 elements for the $-\nabla\Phi$ result
- `*mat_cos` and `*mat_sin` are the 3D arrays that contain the SCF coefficients, as described in Section 3 above.
- `orbit_nmax` and `orbit_lmax` are the desired order for your decomposition

- `x, y, z` are the given halocentric position (in cartesian coordinates scaled to the main halo's scale radius)

As shown in Equation 2.25 in HO92, the basis functions contain $C_n^\alpha(\xi)$ which are Gegenbauer polynomials (GP). In the code, GPs can be generated by two methods.

Method A is to use a hybrid between hard code and GP's recurrence relation. In *gegenbauer.c*, the polynomials are simply hard coded for $n \leq 7$. GPs at higher n can be also hard coded in principle, but they can become prohibitively complicated. So for now, GPs at $n > 7$ rely on its recurrence relation, which can be found in mathematical references[1].

Method B is to use GSL's `gsl_sf_gegenpoly_array()` function.

For the purposes of Ngan et al. (2015) and Ngan et al. (2016) ($n_{max} = 10$ for the main halo and $n_{max} = 4$ for subhalos), it seemed like method B is faster for the main halo, and method A is faster for subhalos. The difference is small, possibly because $n = 10$ is not high enough to matter that much. When it does matter, you can switch between Methods A and B using the flag `USE_GSL_GEGENBAUER_HOST` (or `USE_GSL_GEGENBAUER_SUBHALO` for subhalos) near the top of *scf.h*.

[1] `http://mathworld.wolfram.com/GegenbauerPolynomial.html`

Part II

Computing your own coefficients

6 Compiling and running the code

A separate *Makefile_scf* is provided for this part of the code to compute SCF coefficients if you have your own snapshot of an N-body halo. The idea is to first compute the coefficients and store them. The included VL-2 coefficient file (*host_smooth_scf30_30.scf*) and the subhalo coefficient files (inside *data/subhalo_scf*) are such files. Once the coefficients are pre-computed, they can be used by routines described in Part I to compute potentials.

Since modern N-body simulations of halos consist of $N \gtrsim 10^9$ particles, the code requires MPI so that the snapshot can be read and processed in parallel.

Two executables can be produced with this part of the code:

- `scf` provides the full three-dimensional treatment to the halo. The included VL-2 coefficients were computed this way and were used for the main halo in Ngan et al. (2015) and Ngan et al. (2016). This is the default executable if you compile with the *Makefile_scf* makefile with the command

  ```
  make -f Makefile_scf
  ```

  `scf` takes two arguments:

  ```
  scf nmax lmax
  ```

  where `nmax` and `lmax` are the maximum orders for the coefficient which you want to pre-compute. For example, the included *host_smooth_scf30_30.scf* file was produced using the command `scf 30 30`

- `scf_spherical` provides only the spherical treatment to the halo. The included subhalo coefficients were computed this way. The spherical code is specifically written to eliminate the loops over l and m, so the code is simpler and faster. To compile the spherical code, use the command

  ```
  make -f Makefile_scf scf_spherical
  ```

  `scf_spherical` takes no arguments because it is hard coded to do `nmax=10` and `lmax=0` (by definition of spherical symmetry). The value of n_{max} and the input file name are changeable in the code.

7 Reading the snapshot for your own halo

The function `read_snapshot()` in *read_snapshot.c* reads the snapshot file for the particle positions and masses. Right now the function is written to read VL-2's output files which are in standard

TIPSY format, and relies on constants defined in *scf_decompose.h* for the file name, scale radius, center position, simulation box size, and mass unit.

You would likely need to rewrite `read_snapshot()` in order to read in your own snapshot. The function must assign the following global variables, declared in *scf_decompose.h*:

- The total number of particles of the entire snapshot: `TotalNumPart`
- The number of particles the current node is processing: `NumPart`.
- Allocate memory to `xx, yy, zz, mm` with length `NumPart`, and populate them with the **halocentric positions (in units of the scale radius)** and masses (in $M_\odot$), respectively, of the particles in the snapshot.

8 Implementation notes

8.1 Scale radius

The positions `xx,yy,zz` in the code are stored in units of scale radius. In our VL-2 coefficients, we used a scale radius of 60.42 kpc as reported by the halo finder code. At order 10, our decomposition gives a force error of only $\sim 1\%$ (Ngan et al., 2015).

As discussed in Lowing et al. (2011), the scale radius should be chosen such that the zeroth order basis function (Hernquist profile) would produce a best fit to the actual halo's profile. However, because the Hernquist profile is not a perfect fit to high resolution dark matter halos, finding a rigorous best fit is not very meaningful. Lowing et al. (2011) also noted that the force accuracy is insensitive to the choice of scale radius, which is expected from a complete set of basis functions. Completeness means that the basis functions can approximate any distribution to arbitrary accuracy at high enough order. Therefore, even if the scale radius of choice does not provide a perfect fit between the zeroth order basis function and the actual profile, it can be compensated by increasing the decomposition order.

In principle, where computational resources matter, it may be best to vary the scale radius while fixing the decomposition order in order to find the scale radius that gives the best force accuracy. However, this was not done in Ngan et al. (2015, 2016), as we simply adopted the value report by the halo finder.

8.2 Coefficients stored in the files

The actual implementation of the coefficients follows Section 3 of HO92 (in particular, Equations 3.13 and 3.21–3.23) very closely. Following their notation, computing the potential and acceleration requires $C_{lm}(r)$ and $D_{lm}(r)$ (defined in Equation 3.17), and $E_{lm}(r)$ and $F_{lm}(r)$ (defined in Equation 3.24).

Notice that the coefficients in front of $\cos(m\phi)$ (i.e. $C_{lm}(r)$ and $E_{lm}(r)$) consist of this sum

$$N_{lm}\tilde{A}_{nl}\sum_k \left[m_k\tilde{\Phi}_{nl}(r_k)P_{lm}(\cos\theta_k)\cos(m\phi_k)\right] \tag{4}$$

The index k in the sum denotes the kth particle in the N-body snapshot. r_k, θ_k, ϕ_k are the spherical coordinates of the kth particle, and m_k is the mass of the kth particle (not to be confused with the angular quantum number m). This three-dimensional array is what is being stored in the coefficient file and read by *driver.c* as `mat_cos`. When the array is passed to `scf_potential()`, the code would construct C_{lm} and E_{lm}, and then the potential and acceleration.

After the cosine term elements, the coefficients for the sine terms are very similar. The coefficients $D_{lm}(r)$ and $F_{lm}(r)$ consist of this sum

$$N_{lm}\tilde{A}_{nl}\sum_k \left[m_k\tilde{\Phi}_{nl}(r_k)P_{lm}(\cos\theta_k)\sin(m\phi_k)\right] \tag{5}$$

Similar to the cosine coefficients, these sine coefficients are stored in `mat_sin`, which is processed in a similar way as the cosine coefficients in `scf_potential()`.

Notice that both Equations 4 and 5 above involve summing over all particles in the N-body snapshot. This is why this pre-computation can be parallelized easily.

References

Hernquist, L., & Ostriker, J. P. 1992, ApJ, 386, 375

Lowing, B., Jenkins, A., Eke, V., & Frenk, C. 2011, MNRAS, 416, 2697

Ngan, W., Bozek, B., Carlberg, R. G., et al. 2015, ApJ, 803, 75

Ngan, W., Carlberg, R. G., Bozek, B., et al. 2016, ArXiv e-prints, arXiv:1601.04681