User's Guide for EnBiD

Sanjib Sharma [*]

March 7, 2007

Contents

1 Introduction

EnBiD (**En**tropy Based **Bi**nary **D**ecomposition) is a code for calculation of densities from a discrete set of data points distributed in a multi-dimensional space. The code is completely metric free. The most important feature of EnBiD is its ability to determine the appropriate metric at each point in space. It is based upon the idea of binary space partioning tree (kd-tree). The algorithm EnBiD was inspired by an earlier algorithm dubbed FiEstAS developed by Ascasibar & Binney 2005 ,MNRAS,356,872. The main feature of EnBiD is its unique node splitting criterion, which is based on *Shannon Entropy* and helps to extract maximum information from the data. After the spatial decomposition has been done the code offers a variety of methods for obtaining smoothed estimates of

[*]E-mail:sanjib.sharma@gmail.com

density. The code determines the appropriate metric at each point in space from the shape of the leaf nodes which are generated by the spatial decomposition. The main method of density estimation is the kernel based density estimation. Spherical kernels as well as kernels of the product form can be used. In addition a variety of different kernel functions can also be used. FiEstAS style smoothing as originally proposed by Ascasibar & Binney 2005 can also be done. The algorithm and its test results are described in the paper astroph/0507550. The code is distributed under the GNU general public license.

This guide provides the technical details which will help others to use the code. Some routines in this code are derived from the code GADGET which is a code for cosmological N-Body/SPH simulations, developed by Springel,Yoshida & White 2001, New Astronomy,6,51, The algorithm for nearest neighbor search is based on the algorithm of code SMOOTH
http://www-hpcc.astro.washington.edu/tools/smooth.html

2 Compilation

The code is written in C++. Unpacking the code using
tar -zxvf Enbid-2.0.tar.gz
generates a directory `Enbid-2.0`. The `src` directory contains `.cpp` and `.h` files along with a `Makefile`. Typing **make** in the `src` directory should compile the code on most Unix and linux based systems. Below are listed some options from the `Makefile` which need to be specified during compilation time.

```
OPT1    =  -DWARN     # to put print statements for debugging
OPT2    =  -DDIM3     # For doing calculations in 3 dimensions
OPT2    =  -DDIM6     # For doing calculations in 6 dimensions
OPT2    =  -DDIMO     # For other spaces
OPT3   =   -DMEDIAN   #enables optimizations for faster kernel smoothing
OPT4   =   -DPERIODIC #enables periodic boundary conditions
```

Option `WARN` should normally be kept enabled. In case of a problem it makes the code print a warning and diverts it into an interactive mode asking the user for an input. To run it in an un-interrupted fashion this option should be disabled. The number of dimensions need to be specified during the compilation. 3 and 6 dimensional spaces can be invoked by selecting the appropriate `OPT2`. For other spaces one has to use option `DIMO` and then alter the variable `#define ND` in file `allvars.h`. When `OPT3` is enabled special optimizations are enabled which work with the median splitting criterion, this speeds up the neighbor search by about 10% . Periodic boundary conditions are enabled by option `OPT4`. Any change in the options in Makefile should generally be followed by a **make clean** procedure before compiling it again.

Compiling the code generates the executable file `enbid` in the directory `Enbid-2.0` which can be run from command line along with a parameterfile.
./enbid parameterfile

3 Parameters

The main method for calculating densities is the tree based scheme followed by Kernel smoothing. Other than this FiEstAS based smoothing option is also available. The parameter file helps the user to choose the available options and also provides various parameters to fine tune the efficiency and accuracy of the code. A short description of these parameters is provided below.

```
InitCondFile            Examples/hernquist1_small/snapshot_ici
```

This is the name of the input file which provides the data for density calculation.

```
SnapshotFileBase        _ph
```

The output file name is derived from the input file name by adding the above base name and a suffix `.est`. For the above choice of parameters the file generated is **snapshot_ici_ph.est**.

```
ICFormat                    1
```

For a value of 1 the code reads files in GADGET format. For a value of 0 it can read data written in ASCII format. Files of other formats can be made to read in by writing ones own reading routine. A template function is provided for this. This can be invoked by setting the above parameter to 2.

```
SpatialScale              1
```

This parameter is used to define global scaling relations between various co-ordinates. This choice is useful when one wants to use normal SPH (isotropic kernels and non-adaptive metric) otherwise the parameter should be set to 1. For `SpatialScale` $= 0$ the scaling is done based on the global co-variance of the data. If $\sigma_i = \sqrt{< x_i^2 >}$ is the standard deviation of the ith co-ordinate then the code performs the following transformation $x_i \rightarrow x_i/\sigma_i$. For phase space density calculations (i.e. number of dimensions $d = 6$), if `SpatialScale` > 0, the co-ordinates of real space (dimensions $1, 2$ and 3) are scaled globally with respect to velocity space (dimensions $4, 5$ and 6) by the following transformation, for $i = 1$ to 3 $x_i \rightarrow x_i/\text{SpatialScale}$.

```
PartBoundary              7
```

During tree generation if the number of particles in a node is greater than or equal to the above parameter a boundary correction is applied to its surfaces. Optimum choice of this paramater is $d + 1$, d being the dimensionality of space under consideration. If the densities are found to be underestimated then lower it or vice versa. If the value is too small this can give rise to Poisson errors, so irrespective of the number of dimensions a minimum value of 7 should be used. When the system is known to have well defined rectangular boundaries e.g. systems with periodic boundary conditions, the boundary correction can be switched off by setting this parameter to zero. Read option `PeriodicBoundaryOn` for more details. If this parameter has value 1 boundary correction is applied to the leaf nodes only (as done originally in FiEstAS).

```
NodeSplittingCriterion     1
```

This parameter is used to alter the node splitting criteria used to generate the tree. For a value of 1 *Shannon Entropy* is evaluated in each dimension and the dimension with lowest entropy is chosen to be split. Setting this parameter to 0 disables entropy based node splitting criterion and splits each dimension alternately.

```
CubicCells                 0
```

When this parameter has a value of 1 the
`NodeSplittingCriteria` is used to select the subspace (real or velocity) to be split rather than the dimension. Then from this subspace the axis having maximum variance ($< x_i^2 > - < x_i >^2$) is selected to be split. For systems whose subspaces are known to be Euclidean this gives better results. It results in cubic cells in each sub-space. This option only works for 3 and 6 dimensional spaces.

```
MedianSplittingOn          0
```

Ideally the position of splitting should be close to the median of the data points so that there are equal number of data points on both sides of the split. For building the tree it is computationally more efficient to instead choose the splitting position close to the mean of the data points. When the code is compiled with the `-DMEDIAN` flag setting this parameter to 1 results in faster kernel density estimates.

```
TypeOfSmoothing             3
```

This parameter helps to select various different smoothing options. For a value of zero no smoothing is done this is useful to get fast estimates of density but at the price of a large dispersion. For a value of 1 FiEstAS style smoothing is done, for 2 normal spherical Kernel based scheme is used (having isotropic metric same scale in all dimensions) and for 3 spherical Kernel based scheme with an appropriate adaptive metric (estimated by the size and the shape of the leaf node containing the data point) is used. For a value of 4 and 5 the kernel used is of product form with the metric being isotropic for the former and adaptive for the later. For FiEstAS style smoothing particles of a given type should not have multiple masses.

```
DesNumNgb                   40
```

This gives the number of smoothing neighbors that are used for density calculation. For FiEstAS style smoothing optimum range of this parameter is 2 to 10. For SPH smoothing the optimum choice depends upon the number of dimensions. For a Hernquist sphere in 6 dimensions `DesNumNgb=40` was found to give smoothing equivalent to FiEstAS smoothing with `DesNumNgb=2`.

```
VolCorr                  1
```

During smoothing if the smoothing box extends outside the boundary of the system density might be underestimated. Setting this parameter to 1 enables this correction for both Kernel and FiEstAS smoothing.

```
TypeOfKernel             3
```

For Kernel based smoothing three different types of kernels can be used. For a value of 0 the B-Spline kernel is used, for 1 top hat kernel is used, for 2 Bi-weight kernel is used and for 3 Epanechnikov kernel is used. For 4 cloud in cell (CIC) type of linear kernel is used and for 5 triangular shaped cloud (TSC) type of kernel is used. In our tests Epanechnikov kernel was found to give best results.

```
KernelBiasCorrection        1
```

Normal Kernel based schemes have significant bias in their estimated densities for irregularly distributed data. This bias arises when the densities are calculated at the location of the data points and can be eliminated by suitably displacing the central data point within the smoothing volume. This is enabled by setting the parameter to 1. For regularly spaced data (e.g. glass or lattice like systems) this option should be disabled by setting the value to 0.

```
AnisotropicKernel         0
```

This enables the use of Anisotropic kernels which can have both shear and rotation. The kernel instead of being spherical is now an rotated ellipsoid in general. The local co-variance matrix of the data is used to determine the orientation and the smoothing lengths of the kernel. For this option the `TypeOfSmoothing` should be either 2 or 3 (i.e spherical kernel smoothing).

```
Anisotropy                0
```

Depending upon the anisotropy of the problem, density estimation with anisotropic kernels can be computationally very intensive. The parameter `Anisotropy` can be used to set the minimum allowable minor to major axis ratio of the kernel smoothing lengths. Maximum accuracy is achieved by setting the value to zero while a value of 1 corresponds to an isotropic kernel.

```
DesNumNgbA                100
```

This is the number of neighbors used for estimation of the co-varaince matrix, which is in turn used for calculating the anisotropic kernel. Its value should be between $100-200$. Setting this value to less than 100, results in poor estimation of the co-variance matrix.

```
TypeListOn              0
```

It is possible to specify multiple species (types) of particles in the same initial condition file. Each particle type is treated independently for the purpose of density estimation and separate tree is constructed for each of them. For Gadget format initial condition files the number of particle types is limited to 6. To overcome this limitation and to offer more flexibility the parameter `TypeListOn` can be enabled by setting it to 1. This overrides the type of particles specified in the initial condition file. If this parameter is set to 1 a typelist file in ascii text, with the name `InitCondFile` + suffix `_typelist`, should be provided. If a total of N particles are divided into l independent groups of different types having particles n_1, n_2 ... n_l respectively such that $\sum n_i = N$, then they can be read in by specifying this list in an ascii format in the typelist file as follows

$n_1 n_2 ... n_l$

```
PeriodicBoundaryOn              0
```

Periodic boundary conditions can be enabled for Kernel based smoothing by setting this parameter to 1 and compiling the code with option `-DPERIODIC`. Periodicity of each dimension can be set independently. A file `periodic_lenghts.txt` should be provided in the current working directory, which contains the periodic lengths l_i, for each of the dimensions. Boundary correction to the tree nodes as specified by parameter `PartBound` are not applied in the dimensions that are periodic. If l_i is set to zero the code automatically determines the appropriate periodic length from the range of the particles in that dimension ($l \sim (x_{max} - x_{min})$). If it is set to less than zero then that dimension is not considered periodic. In the code this is achieved by setting $l \sim 10(x_{max} - x_{min})$) and enabling `PartBound` for that dimension. Periodic support is as such not available for FiEstAS smoothing, enabling periodic boundaries in this case only effects the tesselation process (i.e boundary correction to the tree nodes).

4 Input

The data can be read from either an ASCII file, GADGET format file or a user defined file format these can be invoked by setting the parameter `ICFormat` to 0 , 1 and 2 respectively. For GADGET format files the code can read both little and big endian format binary files. The endianness of the output file is the same as that of input file.

4.1 ASCII input file format

In ASCII format the co-ordinates of the data points are arranged in rows as follows.
$x_1 y_1 z_1$
$x_2 y_2 z_2$

........so on

4.2 User defined file format

The user can read in files of his own format by writing his own reading routine. A template function `void read_ic2(char *fname)` in file `read_ic.cpp` is provided for this. The user should read his data and assign the co-ordinates to a predefined structure. Make the following changes in the function `read_ic2`

```
  /* SPECIFY TOTAL NUMBER OF PARTICLES*/
  NumPart= specify total number of particles;

/* leave this unchanged */
//-------------------------------------
  All.MaxPart = NumPart;
  for(i=0;i<6;i++)
    {
      header1.npart[i]=0;
      header1.npart[i]=0;
      header1.mass[i]=0;
    }
  header1.npart[1]=NumPart;
  header1.npartTotal[1]=NumPart;
  allocate_memory();
//-------------------------------------

  /* READ THE DATA HERE AND ASSIGN IT TO P[i].Pos */
  for(i=1; i<=NumPart; i++)
   for(k=0;k<ND;k++)
     P[i].Pos[k]=assign positions;
  for(i=1; i<=NumPart; i++)
    P[i].Mass=assign mass or set it to 1;
```

4.3 GADGET input file format

In a GADGET format file the data is written in binary mode and consists of a header followed by the main data containing position ,velocities and id's of particles. The data fields are separated by block-size fields so that it can be read in with unformatted fortran format. A description of these fields is given below. Read statements in fortran would be as follows.

```
read (1) npart,massarr,a,redshift,flag1,flag2,nall,unused
read (1) pos
read (1) vel
read (1) id
read (1) masses
```

The code can handle multiple species of particles and calculates the density of each specie separately, but for a given specie all the particles must have same mass or else the results might be erroneous. For 6 dimensional phase space density estimation, each particle has 6 co-ordinates, the position of the particle constitutes the first three co-ordinates and the three velocity components are assigned co-ordinate numbers $4, 5$ and 6.

Table 1: Header

Variable	Type	Bytes	Description
npart(6)	int	4×6	Array specifying number of particles of various types
massarr(6)	double	8×6	mass of different types of particles
a	double	4	Time or expansion factor
redshift	double	4	Redshift
flag1	int	4	unused here
flag2	int	4	unused here
nall(6)	int	4×6	Number of particles of each type, same as npart(6) here
flag3	int	4	unused here
numfiles	int	4	number of files holding the data
boxsize	double	4	size of periodic box
omega-0	double	4	Cosmological parameter specifying matter density
omega-l	double	4	Cosmological parameter specifying vacuum energy density
hubble-param	double	4	Hubble parameter
unused			used to fill the header to a total size of 256 bytes

Table 2: Particle Positions

Variable	Type	Bytes	Description
Pos(3,N)	float	$4 \times 3 \times N$	position of particles

Table 3: Particle Velocities

Variable	Type	Bytes	Description
Vel(3,N)	float	$4 \times 3 \times N$	velocities of particles

Table 4: Particle ID's

Variable	Type	Bytes	Description
id(N)	int	$4 \times N$	id of particles used to uniquely identify them in case the order is not same among different files.

Table 5: Particle Masses

Variable	Type	Bytes	Description
mass(N)	int	$4 \times N$	mass of those particles types which have zero entries in massarr.

5 Output

The output file contains a list of densities at the location of the data points (particles). The order is same as that specified in the input file. For `ICFormat=1` , if the mass of the particles is specified the output density is mass density. For `ICFormat=0` mass of each particle is assumed to be unity so the density evaluated by the code is the number density instead of mass density. The output in this case is written in ASCII format. One can also specify ones own output format a template function `void savepositions_ioformat2(int )` is provided for this in file `io.cpp`.

5.1 EnBiD output file format

For `ICFormat=1` the output is written in a binary file with a GADGET format header and a list of densities. Some extra flags are added in the header which might be useful for reading the output densities. The data fields are separated by block-size fields. A description of these fields is given below. Read statements in fortran would be as follows.

```
read (1) npart,massarr,a,redshift,flag1,flag2,nall,unused1
         ,flag-dim,flag-density,unused
read (1) density
```

Table 1: Header

Variable	Type	Bytes	Description
npart(6)	int	4×6	Array specifying number of particles of various types
massarr(6)	double	8×6	mass of different types of particles
a	double	4	Time or expansion factor
redshift	double	4	Redshift
flag1	int	4	unused here
flag2	int	4	unused here
nall(6)	int	4×6	Number of particles of each type, same as npart(6) here
flag3	int	4	unused here
numfiles	int	4	number of files holding the data
boxsize	double	4	size of periodic box
omega-0	double	4	Cosmological parameter specifying matter density
omega-l	double	4	Cosmological parameter specifying vacuum energy density
hubble-param	double	4	Hubble parameter
flag-id	int	4	unused here
flag-dim	int	4	number of dimensions used for density calculations
flag-density	int	4	indicates that it is a `.est` file containing densities and not a GADGET format file
unused			used to fill the header to a total size of 256 bytes

Table 2: Particle Density

Variable	Type	Bytes	Description
Density(N)	float	$4 \times N$	Calculated Density of particles

6 Examples

Two example data files are provided in the directory `Examples`. The file `snapshot_ici` in directory `uniform_6d_box_4` is a GADGET format data of 10^4 particles distributed uniformly in a cubic box of length 100 units in both real and velocity space. The file `snapshot_ici.ascii` is the corresponding ASCII format file. The directory `hernquist1_small` contains the data for a Hernquist sphere modelled with 10^4 particles.

7 General usage tips

The main method of density estimation is the kernel based method along with adaptive metric which is shown as option 3 below, but the code also offers various other options for calculating densities. More accurate estimates require more CPU time. Below we list four main options in increasing order of accuracy and CPU time. Parameterfiles for these can be found in the directory parameterfile.

1. **Unsmoothed estimate:**
   For fast crude estimates of any general data use

   ```
   NodeSplittingCriterion=1
   CubicCells= either 1 or 0 % 1 for symmetric systems 0 for general
   TypeOfSmoothing=0
   ```

2. **FiEstAS smoothing:**
   For smooth densities with FiEstAS style smoothing

   ```
   TypeOfSmoothing=1
   DesNumNgb=2-10
   All.VolCorr=1
   ```

3. **Kernel smoothing with adaptive metric:**
   For kernel smoothed estimates set

   ```
   TypeOfSmoothing=3
   TypeOfKernel=3
   KernelBiasCorrection=1
   DesNumNgb=10-50
   All.VolCorr=1
   ```

4. **Anisotropic kernel smoothing:**
   If one has more time one can also use Anisotropic kernels with

   ```
   TypeOfSmoothing= 3 or 2 % more faster with 2 but less accurate
   AnisotropicKernel=1
   Anisotropy=0
   DesNumNgbA=100-200
   ```

The table below gives the CPU time in seconds required on an AMD 1666.7 MHz processor for calculating the density of a data of 1 million particles modelled as a Hernquist sphere.

Table 3: Cpu Time

Type of Estimate	6 Dimensions
Unsmoothed	10-24 s
Fiestas smoothing	356 s
Kernel smoothing with adaptive metric	863 s

- For a hernquist sphere options 3 and 4 were found to give nearly same results. This is because the anisotropy has already been corrected by means of an adaptive metric.
- For real and phase space density calculation of systems with special symmetries e.g. spherical systems use `CubicCells=1`. Hernquist sphere and galactic halos fall into this category.
- For kernel smoothing on regularly spaced data e.g. a lattice like system disable the parameter `KernelBiasCorrection` by setting it to 0.
- For FiEstAS smoothing all the particles of a given type must have same mass. For other types of density estimation multiple mass support is available but the density estimates are most accurate if the particles have same mass or if at least their distribution in space is smooth.
- The code has mostly been tested in 3 and 6 dimensions and on systems having spherical symmetry in real and velocity sub-spaces. For other situations it will be useful to first test the accuracy and efficiency of the code by experimenting with the parameters and options provided here.
- Although the code can support arbitrary number of dimensions but for kernel based smoothing, for dimensions greater than 20, one needs to specify appropriate normalization constants of the kernel in routine `set_sph_kernel` (file `begrun.cpp`). For other schemes there is no restriction.
- For FiEstAS smoothing of systems with periodic boundary conditions or well defined rectangular boundaries use `VolCorr=1` to avoid underestimation of densities in boundary regions. Periodic support is as such not available for FiEstAS smoothing, enabling periodic boundaries in this case only effects the tesselation process (i.e boundary correction to the tree nodes).
- The mean splitting criterion gives better results as compared to median criterion, for systems having substructures. But if one does not want

to use adaptive metric and instead wants to use isotropic kernels (i.e `TypeOfSmoothing=2` or 4) the code can be compiled with `-DMEDIAN` and used with `MedianSplittingOn=1`, this results in faster density estimates.