A tale of two tools, Galaxia and EBF

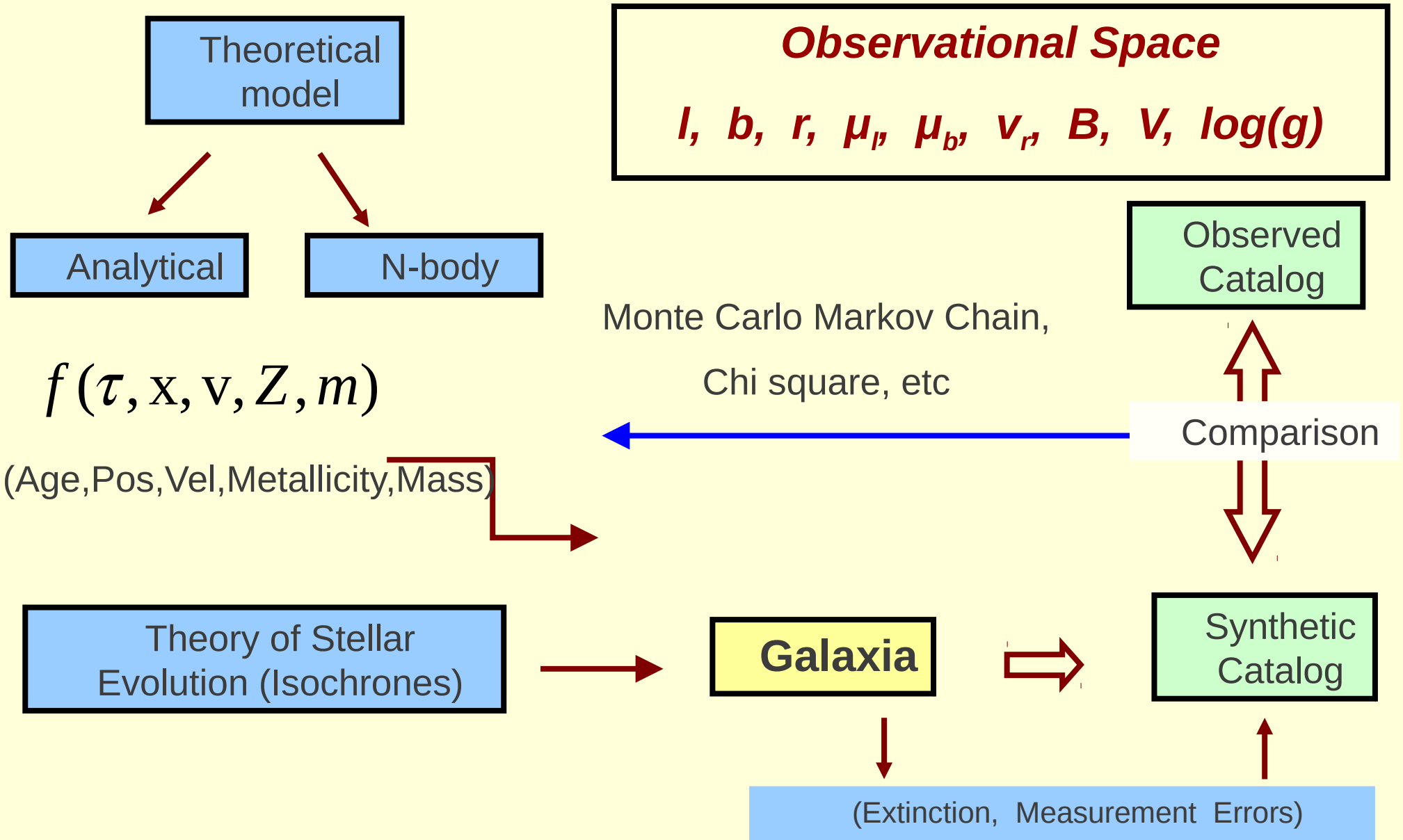
Will be publicly available as open source project at (Feb)
http://galaxia.sourceforge.net

Sanjib Sharma (Univ of Sydney)
Joss Bland-Hawthorn
Kathryn V Johnston
James Binney

sanjib.sharma@gmail.com

Motivation

- A framework to compare theoretical models of our Galaxy with observations.



Other uses of Synthetic Catalogs

- Test capabilities of different instruments to answer key scientific questions.
- Check for systematic errors, biases in analyses.
- Device strategies to reduce measurement errors

Drawbacks of current schemes

- Besancon Model- state of the art (Robin et al 2003)
- Also Trilegal, (Girardi et al, Padova group)
- Designed for simulating a particular line of sight
  - at max 25 line of sights
- Discrete (l,b,r) step sizes to be supplied by user
- Not suitable for wide area surveys, or large catalog of stars
  - takes too much time
- No possibility to simulate substructures or incorporate N-body models
  - Sagittarius dwarf galaxy, simulation of tidally disrupted galaxies

Theoretical Model-Analytical Models

$$f(\tau, \mathrm{x}, \mathrm{v}, Z, m)$$

Age Metallicity Relation
AMR

$$\frac{e^{-(\log Z - \log \bar{Z}(\tau))/\sigma_{\log Z}(\tau)}}{\sigma_{\log Z}(\tau)\sqrt{2\pi}}$$

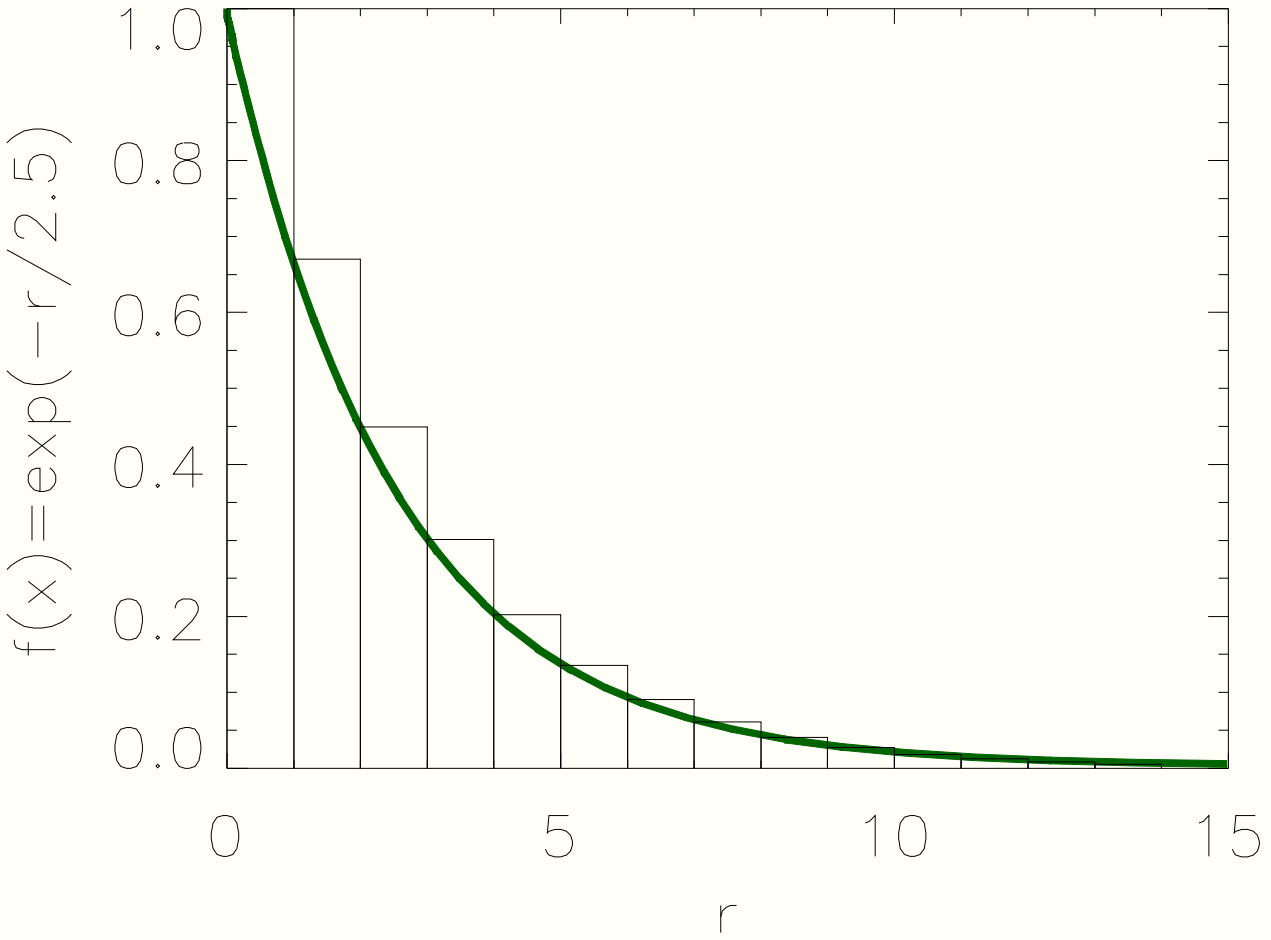
Star Formation Rate
SFR

Phase space
distribution

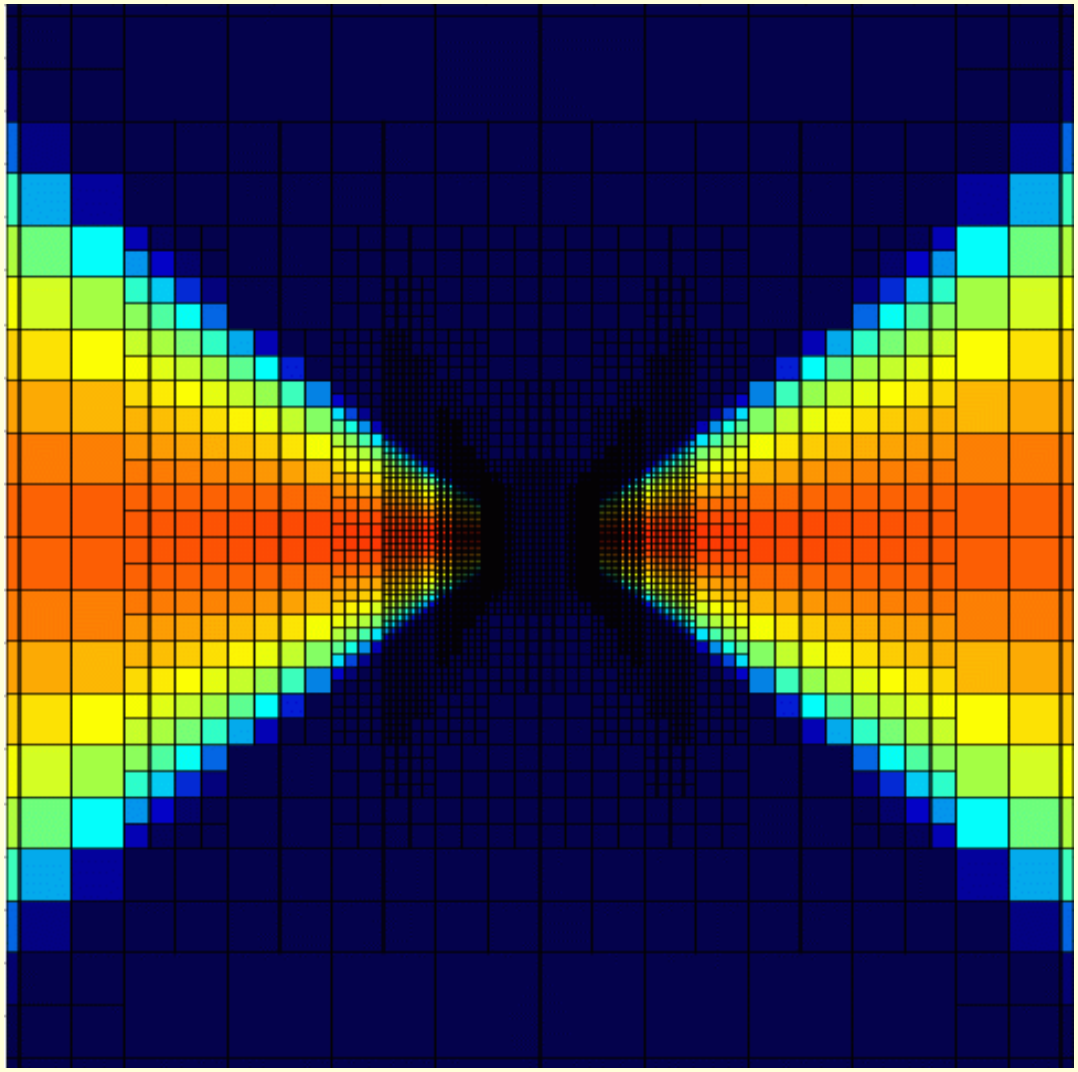
$$= \frac{\Psi(\tau)}{\overline{m}_{\xi}} \xi(m) f_{xv}(\tau, \mathrm{x}, \mathrm{v}) f_{Z}(\tau, \mathrm{x}, Z)$$

Initial Mass Function
IMF

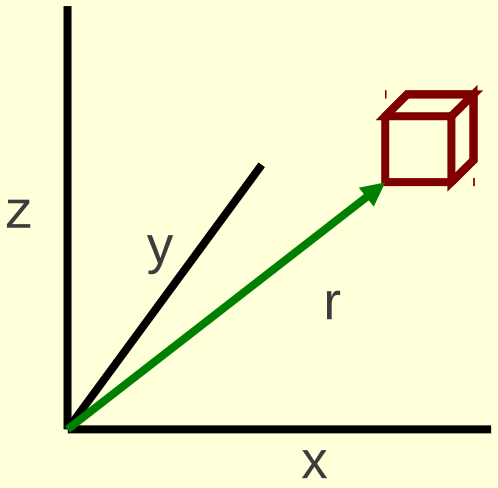
Sampling Analytical Model
(Von Neumann rejection sampling)

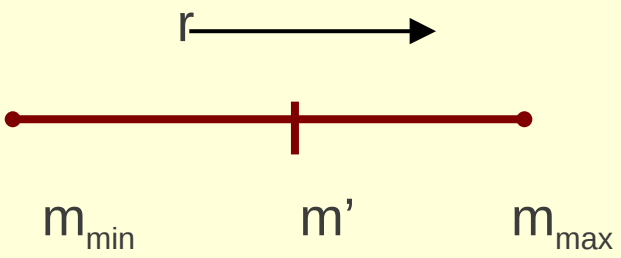


Adaptive Mesh (Barnes Hut Tree)



Optimization



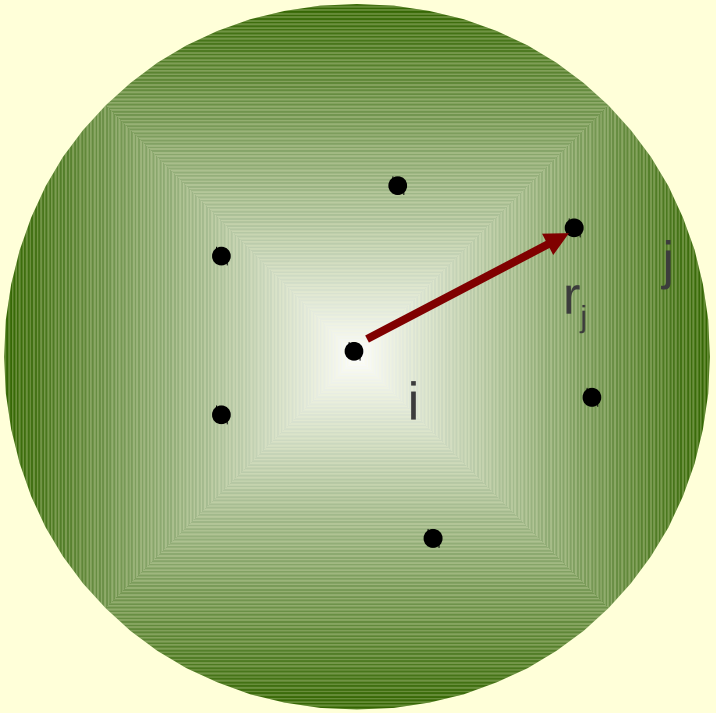


- To generate a patch do not need to generate the full galaxy
  - If a survey is not all sky, first check if a node intersects with survey geometry.
- Faint stars which dominate in number are visible only for nearby nodes.
- For far away nodes there is a minimum mass above which stars are visible
  - Sort nodes according to distance. Calculate appropriate m'
  - Generate only those stars that are visible.

Sampling an N-body model

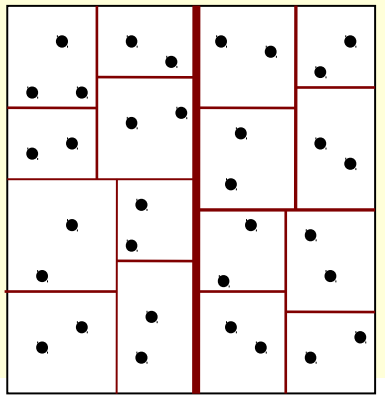
- Number of N-body particles are finite
  - $f_{N\text{-body}}(x,v) < f_{stars}(x,v)$
  - Need to oversample
  - Need to distribute the stars in space
  - How to do this such that the stars sample the phase space distribution of the N-body particles
- Inverse of density estimation
  - Spread the stars over a volume that encloses k nearest neighbor.
- In phase space volume is hyper-ellipsoidal. How to choose correct smoothing length in different dims (pos,vel), i.e., appropriate metric in a multi-dimensional space?

$$f_i = \sum_j \frac{1}{h_j^d} K\left(u_j\right) m_j$$

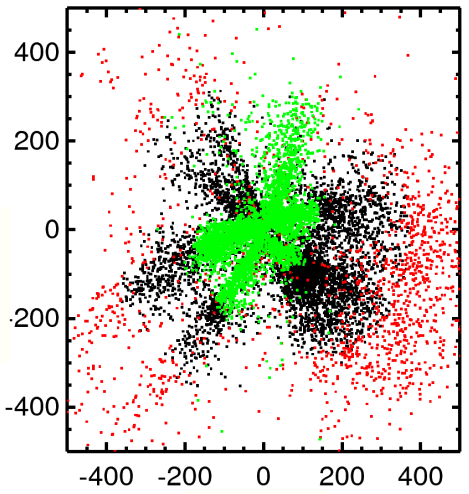


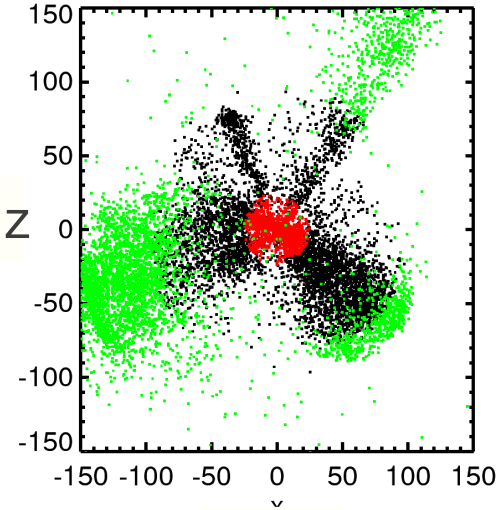
Need for a locally adaptive metric

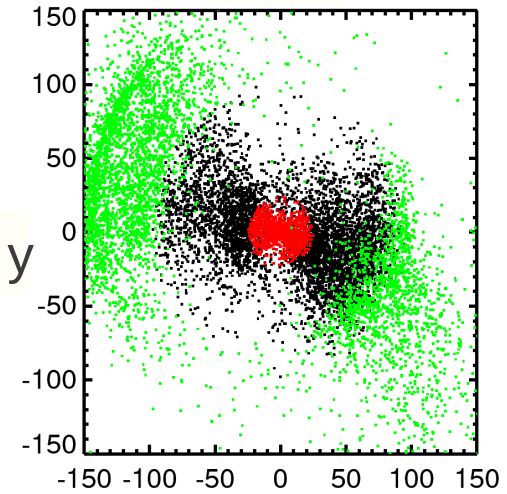
- **EnBiD**-**En**tropy Based **Bi**nary **D**ecomposition in space (Sharma & Steinmetz 2006, an improvement of Ascasibar & Binney 2005)
  - A code for multidimensional density estimation
  - Automatic calculation of the appropriate metric or smoothing lengths
  - Metric is locally adaptive and unique for each point in space.

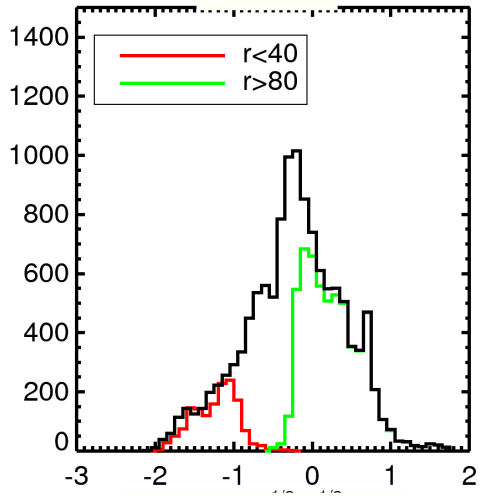


Sharma et al 2011









Publicly available at
http://sourceforge.net/projects/enbid

Computational Performance

- Run time nearly linear with mass of the galaxy being simulated
  - Due to the use of adaptive mesh or node
- Speed- 0.16 million stars per second (2.44 GHz proc)
  - For shallower surveys a factor of 3 less
- V<20, 10,000 sq degrees towards NGP, 35 $\times 10^6$ stars, 220 secs
- V<20 GAIA like survey 4 billion stars can be generated in 6 hours on a single CPU

>>EBF<<

An efficient and easy to use binary file format

Motivation

- Why do we need a format?
  - Otherwise only the program that wrote the data can read it. Or custom reading routine for each data
  - Difficult to share data with others.
- Why binary and not ascii?
  - 700 (6) MB/s, 1800 (18) MB/s
- Need to write multiple items in same file and have random access support?
  - Organize data in one place
  - If not random access then the exact sequence in which the data was written need to be known.
  - New features cannot be easily introduced.
  - 100x100 grid (in age and metallicity) of isochrone tables

Problems with binary data.

- Binary data without specified data type is just 0 and 1. Hence data type information needs to be specified.
- Not portable due to Endianness (little vs big)
  - In a multi byte word the most significant byte is to the left or right. Intel vs IBM processors.
-

b1	b2	b3	b4

b4	b3	b2	b1

Why not use HDF5 or FITS?

- Fits does not support multiple items tagged by names.
- Sequential access too slow for large number of items
- HDF5 a complicated format. (460 functions)
- API not user friendly. Steep learning curve.
- Main API only C. In other languages one has to rely on foreign language interface to call C the routines.
- Not fully type safe. Errors not detected at compile time
  - (…,...,...,...,HDF5_NATIVE_INT,x)
- Writing lot of small items requires too much memory. Per item 4KB for FITS and 2KB for HDF

EBF design goals

- Binary format for speed
- Multiple items with random access
  - Like HDF5, each data item is specified by a unique taganme, which follows unix style pathnames
  - e.g., /x1 , /mydata/x1 and so on
- Ease of use
- Design APIs such that it is harder to make mistakes, and when you do it will give compilation error.
- Support for multiple programming languages.
  - No use of foreign language interface
  - Pure code in all languages.
- Automatic type and endian conversion
- Support for attributes and data units.

The Format

Header

char	Signature[8]
char	version[4]
int32	endian_test =1684234849 (abcd)
int32	header_size
int32	name_size
int32	data_type
int32	data_size
int32	rank
int32	unit_size
int32	nfields
int64	dim[rank]
char	name[name_size]
char	unit[unit_size]
int8	field_name_size
char	field_name
int32	field_size
char	field[field_size]
char	extra[64]

44 bytes

Defining structures

Nested (recursive structures allowed)

Field name:
“sdef”

Field:

```
struct
{
    float32 density;
    float64 mass;
    int32    metals 3 2;
    struct   {
    float32 pos 3;
    float32 vel 3;
    }point 1;
}
```

- Only idl and python
- Byte alignment issues make it less portable for static languages like C/C++
- Preferably split and write each field as separate arrays.

Supported Data types

Data Type	Integer Code
undefined	0
char	1
int32 (int)	2
int64 (long)	3
float32 (float)	4
float64 (double)	5
int16 (short)	6
structure	8
Int8 (unsigned char)	9
uint8 (signed char)	10
uint16 (unsigned short)	11
uint32 (unsigned int)	12
uint64 (unsigned long)	13

The API

- `double x[100];`
- `ebfwrite("check.ebf","/x1","w",&x[0],"100 km/s",100);`
- `ebfwriteAs<int>("check.ebf","/x2","a",&x[0],"100,km/s",10,10);`
- `vector<float> y;`
- `ebfread("check.ebf","/x","w",y);`
- `EbfDataInfo dinfo=Ebf_GetDataInfo("check.ebf","/x1");`
- `y.resize(dinfo.elements);`
- `ebfread("check.ebf","/x",&y[0],dinfo.elements);`
- `Float* y=ebfallocFloat32("check.ebf","/x");`

  - Automatic Endian conversion
  - Automatic Type conversion

API contd

- `Efile efile;`
- `efile.open("check.ebf","/x","w",Ebf_type("int32"),"km/s");`
- `efile.write(&x[0]);`
- `efile.write(&x[1],10);`
- `Efile.close();`
-
- `efile.open("check.ebf","/x");`
- `efile.read(&y[0]);`
- `efile.read(&y[1],10);`
- `Efile.close();`

Iterating without loading the full data (C++ only)

- `ebfarray<float> x("check.ebf","/x");`
- `x[i];`
- `x(i,j);  // multidimensional index`
- `x(i,j,k);  // multidimensional index`
- Only 1000 items loaded at a time, full data never loaded.
- Useful for traversing large data sets with a small amount of memory.

Dynamic languages IDL,Python,Matlab

- `Ebf.write("check.ebf","/x","w",x)`
- `x=Ebf.read("check.ebf","/x")`
- `data=Ebf.read("check.ebf","/mydata1/")`
  - Only objects in current path
  - `data["x1"], data["x2"]`
- `data=Ebf.read("check.ebf","/mydata1/","rec")`
  - All objects recursively in current path
  - `data["x1"]`
  - `data["x1_attributes"]["mass"]`
- `Ebf.write("check.ebf","/mydata1/","a",data)`
  - Fully reversible read write

The ebf toolkit *ebftk*

$ebftk –help

NAME:

ebftk - a toolkit for Extendend Binary Format (EBF) files (version 0.2)

USAGE:

```
ebftk    -diff  file1 file2
ebftk    -list filename
ebftk    -stat filename "TagName1 TagName2 .."
ebftk    -copy src_file dest_file
ebftk    -copy src_file dest_file TagName
ebftk    -cat filename "TagName1 TagName2 .."
ebftk    -csv filename "TagName1 TagName2 .."
```

Performance

(1000 data items of size 4 bytes, array of 10^7 float)

Language	Item write	Item read	Data write	Data read
	KOP/s	KOP/s	MB/s	MB/s
C/C++ EBF	9	23	775	1800
C/C++ HDF5	1.5	1.5	775	1800
C/C++ FITS	0.2	0.5	344	502
C/C++ ASCII			5.6	18
Fortran90/2003	6.0	8.3	950	1120
Java	2.3	7.4	270	727
Python EBF	1.72	1.07	466	620
Python HDF5	0.95	1.0	659	1030
Python FITS	0.74	0.0012	427	1047
IDL EBF	2.7	2.6	113	772
IDL HDF5	5.0	7.4	110	94
IDL FITS	2.7	0.007	80	360
Matlab EBF	0.26	0.26	680	1175
Matlab HDF5	0.26	0.86	1000	1030
Matlab FITS		0.0004		78

Attributes and data units

- Unlike HDF or FITS, no special interface for attributes, just write like other data items.
  - /data, /data_attributes/attr1, /data_attributes/attr2
- Units are not attributes they are part of definition of data.
  - Attributes can also have units
  - e.g /density1 , /density1_attributes/time

Conclusions

- Size of items cannot be expaned. Could be supported in future.
- No support for hyperslab selection
  - HDF5 can do both of above, as it uses B-trees
- Easier to use and at the same time performance at par with HDF.
- Galaxia a tool well suited for comparing theoretical models of Milky Way with observations.
- For release check at
  - http://galaxia.sourceforge.net
  - final release of EBF probably at git-hub
  - sanjib.sharma@gmail.com