05_Data_Inspection_with_yt

September 26, 2014

1 Starting Out and Loading Data

We're going to get started by loading up yt. This next command brings all of the libraries into memory and sets up our environment.

```
In []: import yt
```

Now that we've loaded yt, we can load up some data. Let's load the `IsolatedGalaxy` dataset.

```
In []: ds = yt.load("IsolatedGalaxy/galaxy0030/galaxy0030")
```

1.1 Fields and Facts

When you call the `load` function, yt tries to do very little – this is designed to be a fast operation, just setting up some information about the simulation. Now, the first time you access the "index" it will read and load the mesh and then determine where data is placed in the physical domain and on disk. Once it knows that, yt can tell you some statistics about the simulation:

```
In []: ds.print_stats()
```

yt can also tell you the fields it found on disk:

```
In []: ds.field_list
```

And, all of the fields it thinks it knows how to generate:

```
In []: ds.derived_field_list
```

yt can also transparently generate fields. However, we encourage you to examine exactly what yt is doing when it generates those fields. To see, you can ask for the source of a given field.

```
In []: print ds.field_info["gas", "vorticity_x"].get_source()
```

yt stores information about the domain of the simulation:

```
In []: print ds.domain_width
```

yt can also convert this into various units:

```
In []: print ds.domain_width.in_units("kpc")
       print ds.domain_width.in_units("au")
       print ds.domain_width.in_units("mile")
```

2 Mesh Structure

If you're using a simulation type that has grids (for instance, here we're using an Enzo simulation) you can examine the structure of the mesh. For the most part, you probably won't have to use this unless you're debugging a simulation or examining in detail what is going on.

```
In []: print ds.index.grid_left_edge
```

But, you may have to access information about individual grid objects! Each grid object mediates accessing data from the disk and has a number of attributes that tell you about it. The index (`ds.index` here) has an attribute `grids` which is all of the grid objects.

```
In []: print ds.index.grids[1]
```

```
In []: g = ds.index.grids[1]
       print g
```

Grids have dimensions, extents, level, and even a list of Child grids.

```
In []: g.ActiveDimensions
```

```
In []: g.LeftEdge, g.RightEdge
```

```
In []: g.Level
```

```
In []: g.Children
```

2.1 Advanced Grid Inspection

If we want to examine grids only at a given level, we can! Not only that, but we can load data and take a look at various fields.

This section can be skipped!

```
In []: gs = ds.index.select_grids(ds.index.max_level)
```

```
In []: g2 = gs[0]
       print g2
       print g2.Parent
       print g2.get_global_startindex()
```

```
In []: print g2["density"][:,:,0]
```

```
In []: print (g2.Parent.child_mask == 0).sum() * 8
       print g2.ActiveDimensions.prod()
```

```
In []: for f in ds.field_list:
           fv = g[f]
           if fv.size == 0: continue
           print f, fv.min(), fv.max()
```

3 Examining Data in Regions

yt provides data object selectors. In subsequent notebooks we'll examine these in more detail, but we can select a sphere of data and perform a number of operations on it. yt makes it easy to operate on fluid fields in an object in *bulk*, but you can also examine individual field values.

This creates a sphere selector positioned at the most dense point in the simulation that has a radius of 10 kpc.

```
In []: sp = ds.sphere("max", (10, 'kpc'))
```

```
In []: print sp
```

We can calculate a bunch of bulk quantities. Here's that list, but there's a list in the docs, too!

```
In []: print sp.quantities.keys()
```

Let's look at the total mass. This is how you call a given quantity. yt calls these "Derived Quantities". We'll talk about a few in a later notebook.

```
In []: print sp.quantities.total_mass()
```