Visualization Tutorials for the 2018 European Einstein Toolkit Workshop

Jonah M. Miller

Los Alamos National Lab

September 2018

Inspecting Your Data in Python, Part 1: Uniform Curvilinear Grids

In this notebook, we open a GRMHD simulation run on a uniform curvilinear grid in Python and inspect the structure of the data. We show how this can give you insight into what the data is doing and how you can make some simple plots.

Matplotlib and h5py

HDF5

HDF5 is an output library. It supports outputting in serial or in parallel. It spits out compact (even compressed!) binary data and it is self-describing. A single file can have many datasets, and even a whole internal directory structure. It has a convenient Python API. It's a common output format used by the Einstein Toolkit and I highly recomend you get familiar with it.

Matplotlib

Matplotlib is just an excellent plotting library in Python.

In [ ]:
```
%matplotlib inline
import h5py
import numpy as np
from matplotlib import pyplot as plt
mpl.rcParams.update({'font.size':18})
```

Loading your data

Here I have prepared two files for you. One contains the coordinate data describing the grid the simulation was run on. The other contains a number of interesting physical quantities. I could leave this data on disk (if it was too large to load into memory or if I wanted to exploit parallelism). But instead, I'll just open the files, read the data in, and then close the files.

Each dataset in the hdf5 file has a name, which I'm storing as the key in a dictionary.

In [ ]:
```
grid = {}
with h5py.File('harmdisk2d/grid.h5','r') as f:
    for k,v in f.items():
        grid[k] = v.value
data = {}
with h5py.File('harmdisk2d/data.h5','r') as f:
    for k,v in f.items():
        data[k] = v.value
```

Let's look at what we have

In [ ]:
```
list(grid.keys())
```

In [ ]:
```
list(data.keys())
```

You can investigate a variable now as a numpy array. For example:

In [ ]:
```
data['mbh'],data['N1tot'],data['N2tot'],data['N3tot']
```

In [ ]:
```
data['PRESS'][56:68,56:68,0]
```

Exercise

Explore the data and see what information you can extract about what this simulation was and what the parameters of it were.

Simple Plots

Matplotlib can create simple 2D heatmaps with Cartesian topology fairly easily. This simulation was run with curvilinear coordinates, however. So if we want to use this feature, we either need to do some work, or we need to plot in r, θ coordinates:

In [ ]:
```
r = grid['Xbl'][:,:,0,1]
theta = grid['Xbl'][:,:,0,2]
P = data['PRESS'][:,:,0]*data['U_unit']/1e28
plt.pcolor(r,theta,P)
cbar = plt.colorbar()
cbar.set_label(r'$10^{28}\times$ Pressure (cgs)')
plt.xlim(0,50)
plt.xlabel(r'$r/r_g$')
plt.ylabel(r'$\theta$')
plt.savefig('../figures/harmdisk2d.pdf',bbox_inches='tight')
```

Exercise

Can you use the data available in these files to generate a 2D plot in Cartesian coordinates?

Inspecting Your Data, Part 2: Carpet AMR Data

In this notebook, we show how to investigate Einstein Toolkit data in Python. The Einstein Toolkit uses an AMR library, called Carpet. Carpet is a complicated piece of machinery, so let's see if we can understand some of its moving parts as we go along.

First, let's import the same python tools as we used before:

In [ ]:
```
%matplotlib inline
import numpy as np
import h5py
import matplotlib as mpl
from matplotlib import pyplot as plt
```

In [ ]:
```
mpl.rcParams.update({'font.size':18})
```

Here's the files that Carpet output:

In [ ]:
```
!ls qc0-mclachlan-cell-t0
```

Carpet outputs one hdf5 file per "variable group" (or per variable, depending on your settings) per MPI thread. Syou can see that this simulation was run on four MPI threads (though it could be many more openmp threads since Carpet is hybrid-parallelized.) Let's look inside one:

In [ ]:
```
f = h5py.File('qc0-mclachlan-cell-t0/admbase-lapse.file_0.h5','r')
list(f.keys())
```

This is the typical format for a Carpet IO file. Let's unpack it:

`ADMBASE::alp` means that the variable is named `alp` and is defined in the `ADMBASE` thorn. This is the lapse in the ADM formulation of the Einstein equations.

`it=0` means this output is from the zeroth iteration of the simulation. It's initial data.

`tl=0` refers to the subcycling in time that Carpet can do. (Finer grids can be evolved at shorter timesteps than coarser grids.) Unless you explicitly ask, Carpet will always output data with `tl=0`.

`rl=0` specifies the coarseness of the grid. The grid spacing Δx is divided by a factor of two for each increased refinement level. So `rl=6` has a grid spacing 2^6 times smaller than `rl=0`.

`c=0` specifies the *component* of the grid that Carpet has output. When Carpet distributes the grid structure onto a distributed memory supercomputer, it has to give different pieces of the grid to different MPI threads. Each of these pieces of a grid is called a component. Carpet chooses to output components as it likes to balance the load as best it can. In this example, carpet assigned one component at each refinement level to each MPI thread. This is a typical configuration but *it is not guaranteed.*

Let's look in more detail what's inside that `Parameters and Global Attributes` group. A group is a kind of subdirectory in an HDF5 file. It's a way of making the files self-describing. Carpet puts useful metadata in this group.

In [ ]:
```
list(f['Parameters and Global Attributes'].keys())
```

You can probably guess what some of these contain. The `Datasets` dataset just contains a list of variable names the file contains. Ours only contains the lapse.

In [ ]:
```
f['Parameters and Global Attributes/Datasets'].value
```

The other datasets in this group are more interesting. `All Parameters` contains what is essentially a copy of the parameter file used to run the simulation:

In [ ]:
```
print(str(f['Parameters and Global Attributes/All Parameters'].value,encoding='
utf'))
```

Finally, the `Grid Structure v5` dataset contains a description of how Carpet structured the AMR hierarchy and broke up the AMR grids into components:

In [ ]:
```
print(str(f['Parameters and Global Attributes/Grid Structure v5'].value,encodin
g='utf'))
```

This isn't meant to be human readable---it's a direct dump of the internal represnetation Carpet keeps.

In [ ]:
```
f.close() # make sure to close an hdf5 file when you're done with it
```

Let's read in the lapse and the grid data. We can read in all the relevant files using the Python glob tool.

In [ ]:
```
from glob import glob

grid_filenames = sorted(glob('qc0-mclachlan-cell-t0/grid-coordinates.file_*.h5'
))
lapse_filenames = sorted(glob('qc0-mclachlan-cell-t0/admbase-lapse.file_*.h5'))

grid_filenames
```

And now we can read in all the relevant components:

In [ ]:
```
grid = {}
for filename in grid_filenames:
    with h5py.File(filename,'r') as f:
        for k,v in f.items():
            try: # this little trick reads only data sets from the root group
                grid[k] = v.value
            except:
                pass
lapse = {}
for filename in lapse_filenames:
    with h5py.File(filename,'r') as f:
        for k,v in f.items():
            try: # this little trick reads only data sets from the root group
                lapse[k] = v.value
            except:
                pass
```

In [ ]:
```
list(grid.keys())
```

In [ ]:
```
list(lapse.keys())
```

A commom trick I like to use is to define functions to access the various field names. For example:

In [ ]:
```
def get_coord(d, rl, c, it = 0, tl = 0):
    return grid['GRID::{} it={} tl={} rl={} c={}'.format(d,it,tl,rl,c)]
def get_lapse(rl,c,it=0,tl=0):
    return lapse['ADMBASE::alp it={} tl={} rl={} c={}'.format(it,tl,rl,c)]
```

Now we can load in and plot a component. (Note that the index ordering is fortran order. The indices go zyx.) For example:

In [ ]:
```
rl = 6
c = 3
alpha = get_lapse(rl,c)
X,Y,Z = [get_coord(d,rl,c) for d in ['x','y','z']]
iz = int(Z.shape[2]/2)
mesh = plt.pcolormesh(X[iz],Y[iz],alpha[iz],rasterized=True)
cbar = plt.colorbar()
cbar.set_label(r'$\alpha$')
plt.xlabel(r'$x$')
plt.ylabel(r'$y$')
plt.savefig('../figures/lapse2d.pdf',bbox_inches='tight',dpi=600)
```

Exercise

This simulation data contains two black holes in-spiraling in puncture coordinates. Can you find their coordinate locations?

HINT: The lapse α is minimal near the punctures.

Installing yt

Here we show you how to install yt. yt is a visualization tool that can be used out-of-the-box for visualizing and analyzing simulation output. It's a python package and installing it is as simple as using pip:

In [ ]:
```
!pip install yt --user
```

Once it's installed, we have to ensure it's added to our path

In [ ]:
```
import sys
sys.path.append('.local/lib/python3.5/site-packages')
```

Now we can import it. We also tell jupyter to display yt plots inline with the little piece of magic

```
%matplotlib inline
```

In [ ]:
```
%matplotlib inline
import yt
import numpy as np
```

Let's check that it all works by importing some random numbers into an AMR hierarchy.

In [ ]:
```
grid_data = [None,None]
grid_data[0] = dict(left_edge=[0.0, 0.0, 0.0],
                    right_edge=[1.0, 1.0, 1.0],
                    level=0,
                    dimensions=[32, 32, 32])
grid_data[1] = dict(left_edge=[0.25, 0.25, 0.25],
                    right_edge=[0.75, 0.75, 0.75],
                    level=1,
                    dimensions=[32, 32, 32])

for g in grid_data:
    g["density"] = np.random.random(g["dimensions"]) * 2 ** g["level"]

ds = yt.load_amr_grids(grid_data, [32, 32, 32])
```

Beautiful! Let's plot it.

In [ ]:
```
plot = yt.ProjectionPlot(ds,'z','density')
plot.annotate_grids()
plot.save('../figures/yt-rand.pdf')
```

Loading Arbitrary Curvilinear Data into yt

In this tutorial, we will use yt to perform some simple visualization tasks with curvilinear data. Make sure you have yt installed for this tutorial. If you don't, you can install it by following the instructions in the `installing-yt` notebook.

First, let's load the relevant libraries...

In [ ]:
```
import sys
sys.path.append('.local/lib/python3.5/site-packages')
```

In [ ]:
```
%matplotlib inline
import h5py
import numpy as np
import yt
```

We will use the same accretion disk GRMHD simulation as we did in a previous tutorial. Let's load it in just as we did before.

In [ ]:
```
grid = {}
with h5py.File('harmdisk2d/grid.h5','r') as f:
    for k,v in f.items():
        grid[k] = v.value
data = {}
with h5py.File('harmdisk2d/data.h5','r') as f:
    for k,v in f.items():
        data[k] = v.value
```

If your data is evenly spaced, you can load it via the `load_uniform_grid` method:

In [ ]:
```
help(yt.load_uniform_grid)
```

Unfortunately, our data is logarithmic in radius, so instead we will use the `load_hexahedral_mesh` method, which loads sime-structured grid data. The expectation for this method is that the data is of Cartesian product topology.

In [ ]:
```
help(yt.load_hexahedral_mesh)
```

We'll use the helper function `hexahedral_connectivity` to tell `yt` how the grid points are connected to each other:

In [ ]:
```
help(yt.hexahedral_connectivity)
```

First, we extract the 1d list of cell centers

In [ ]:
```
r = grid['Xbl'][:,0,0,1]*data['L_unit'][0]
theta = grid['Xbl'][0,:,0,2]
phi = grid['Xbl'][0,0,:,3]
```

The hexahedral mesh code actually wants cell faces, though. So we need to extract these. We can do so by finding `dx`.

In [ ]:
```
# r
dr = r[1]-r[0],r[-1]-r[-2]
rf = np.empty(len(r)+1)
rf[:-1] = r - dr[0]/2.
rf[-1] = rf[-2] + dr[1]
# theta
dth = theta[1]-theta[0],theta[-1]-theta[-2]
thf = np.empty(len(theta)+1)
thf[:-1] = theta - dth[0]/2.
thf[-1] = thf[-2] + dth[1]
# phi is special. It's only one zone and we know what the faces are
phif = np.array([0,2*np.pi])
```

Now we can generate the `coords` and `conn` data structures which tell us how grid vertices are connected to each other

In [ ]:
```
coords,conn = yt.hexahedral_connectivity(rf,thf,phif)

rf[-2]
```

And now we can load our data into yt

In [ ]:
```
arr_data = {'pressure':np.abs(data['PRESS'])*data['U_unit'] + 1e12, # to preven
t underflow
            'entropy':data['ENT']}
bbox = np.array([[rf.min(),rf.max()],
                 [thf.min(),thf.max()],
                 [phif.min(),phif.max()]])
periodicity=[False,False,True]

ds = yt.load_hexahedral_mesh(arr_data,conn,coords,
                             #length_unit=data['L_unit'][0],
                             bbox=bbox,
                             periodicity=periodicity,
                             geometry='spherical')
```

By jumping through these few small hoops, we've enabled the ability to do some cool analysis tricks. For example, we can plot our data in spherical coordinates, which you may recall was an earlier exercise.

In [ ]:
```
slc = yt.SlicePlot(ds,'phi','pressure')
slc.save('../figures/harmdisk-xz-slice.pdf')
slc.show()
```

Exercise:

Explore what data you can plot with yt using curvilinear coordinates. What works? What doesn't?