REU Tutorial 2019

Abid Khan, Sean Connelly, Cunwei Fan, and Patchara Wongsutthikoson
Edited: Eric Connelly, Imran Sultan, Minh Nguyen

April 25, 2019

1   Intoduction

This tutorial will familiarize you with the basic software and methods used in this REU. It is not comprehensive, but upon completion, you should be able to contribute effectively to the group. Previous incoming students have typically completed this tutorial in a few weeks.

2   Accounts and Keys

One of the senior members will join you to get this section done. You can take your time, but getting these done quickly will make your life easier.

- Get keys for Loomis and the REU office.

- Talk to Rebecca Wiltfong (rwiltfon@illinois.edu, 290L LLP) about permissions for the REU webpage.

- Get your Blue Waters Token and have the account activated. You should have gotten an email about it. Talk to Milton Ruiz (setisam@gmail.com) if you come across troubles.

3   Linux

This group uses Linux exclusively. Over the years, you will learn to use it and eventually prefer it over any other operating system. One of the most important and powerful tools you will need to master on Linux is the **terminal**. How to use terminal will become clear as you work with it. A good way to start is by familiarizing yourselves with the following basic commands (i.e. by googling, etc.): `cd, ls, mkdir, touch, cp, rm` (be *very careful* with this one). A quick way to learn what a command does is to type `man commandname` into the terminal. After that, you can look at some more useful commands below.

3.1   Gereral commands

`grep` This command outputs items that match your input pattern strings. It is very useful in finding files and filter the output of another command when used with pipe (|). eg: `grep 1200 foo.txt` will output the lines that has "1200" in file foo.txt.

`pipe (|)` This command makes compound commands available, allowing you to perform more complicated operations in one command line. It will feed the output of previous command as the input for the next one. e.g: `ls | grep foo`, this use the output of ls command as the input for grep command. This command line will display the items in current directory that has pattern string "foo" in it.

`redirect (>)` This command will redirect the output display to another proper place (we usually use this to output into a text file.) e.g `echo hello world > new.txt` will modify the file "new.txt" (or make a new file "new.txt", if it doesn't exist) to have "hello world" as its content. Note that > erases the file before writing, while >> appends to it.

`sort` What do you think it does?

`uniq` Delete *consecutive* duplicate lines of a file. See `awk` for how to delete *non-consecutive* duplicate lines.

`scp` At times we will be needing to send files or data to each other for collaboration. How do we do that? EMAIL?! Where's the fun in that? To send a file from one machine to another type
`scp filename colten1@machine.physics.illinois.edu:/home/colten1/Desktop`,
where instead of "machine" put in the name of the computer you want to send it to. Also you can retrieve a file from another machine using the same command:
`scp colten1@machine.physics.illinois.edu:/full/path/to/file /path/to/folder`
If you want to transfer a directory instead, put an `-r` after scp

scripting Bash supports a few simple common functions like for loop and if condition. These function allow you to write more complicated scripts that can perform a series of operations. I'm not going to write down the usage of those function, as most of them you should easily find documentations on google whenever you need them. The only one thing I want mention is that in bash scripts, when you want to refer to a variable you defined earlier, put a dollar sign ($) in the front. eg. `var=1; echo $var`

awk (optional) Technically a programming language, but useful for editing text files. Ubiquitous in Unix-like systems. This command allows you to perform data extraction based on patterns. A few simple ways of using it will be `awk 'print $2'` (print 2nd column of a file), and `awk '!seen[$0]++'` (remove non-consecutive duplicate lines)
Refer to `http://www.pement.org/awk/awk1line.txt`

sed (optional) Short for 'stream editor'. This is a very powerful string editing tool which allows you to perform text deleting and modifying operations.
The official documentation is at `http://sed.sourceforge.net/sed1line.txt`, but if you want a guide, I'd recommend looking here: `http://www.grymoire.com/Unix/Sed.html`

Now you should be able to manage your files and manipulate them in a basic way.

3.2 Vim

On Blue Waters, because we are remote accessing it, the only text editor we will be able to use is Vim. So learn it, live it, love it becasue you will be working with it all the time. To open a file with vim, do `vim <filename>` (note that this will create a new file if `<filename>` doesn't exist). There are three different modes to Vim:

Command mode Used to navigate the file and execute commands that affect the file as a whole. This is the mode you start in when you open a file. To get to this mode, press Esc (maybe many times).

Insert mode Used to type text in your file. To get to this mode, press a.

Replace mode Similar to Insert mode, but makes you overwrite existing text as you type. To get to this mode, press R.

To help expedite the learning process here are a few helpful commands in **Command mode**:

`:w`   To save (or write) the buffer to file.

`:q`   To quit out of a file (without saving). Adding '!' at the end (i.e `:q!`) will force quit without saving.

`:x` or `:wq`   To save and then quit the file.

`/<txt>`   To search the file for `<txt>`. This is case sensitive. To get to the next (previous) search result, simply press `n` (`N`).

`:n` and `:N`   If you have multiple files open, :n goes to the next file, and :N goes to the previous file. If you're on the last file, :n will not work, so you will have to exit the files.

`gg`   To go to the beginning of the file.

`G`   To go to the end of the file.

`0`   To go the the beginning of the line.

`$`   To go the the end of the line.

`dd`   To delete the line the cursor is on. This will also copy the line for pasting.

`yy`	To copy everything on the line after the cursor.
`p`	To paste the text you just copied or deleted on the line below your cursor.
`u`	To undo the last command.
`Ctrl+r`	To redo the command.
`r`	To replace the character your cursor is on. This takes an input, i.e pressing `re` will replace the character with 'e'
`x`	To delete the character your cursor is on.
`<num><command>`	To execute `<command>`, `<num>` times, i.e `5dd` will delete 5 lines.

Now you should be able to do most of the basic stuffs completely within terminal.

3.3   Exercises

Don't spend too much time trying to work these out yourselves; if you get stuck, feel free to use Google to whenever you need to. These are mainly designed for pedantic purposes.

1. Count the number of files in a folder.

2. Remove the extension part of the file name. eg. "foo.txt" $\rightarrow$ "foo".

3. Use the `tail` and `head` command to remove the first and last line of a text file. After that, try using sed.

4. Use `grep` command to delete all the lines in a text file that contains a specific pattern (say "NaN"). *Hint:* `-v` (While `sed` can be used, it is much slower for larger files).

5. Use `sort` command to sort a text file based on the value in the second column in each line. Assume your text file has multiple rows and columns. *Hint:* `-k, -n`

4   Basic Imaging

The group presently uses VisIt (`https://wci.llnl.gov/codes/visit/`) to render movies and still images. This software is installed on all REU machines, as well as Blue Waters. You are encouraged to read the manuals at:
`https://wci.llnl.gov/codes/visit/manuals.html`
To begin using VisIt, simply type 'visit' into a terminal of your choice.

4.1   Exercises:

1. Useful to us is the `.3d` data format. The `.3d` data format just contains four columns of the form `x y z scalar`. Create a `.3d` datafile of the scalar field $d = x^2 y^3 / z^5$ and visualize it in VisIt using a custom colormaps that you make. Be creative and try to explore VisIt's features.

2. Next, create a sphere, using Python, C++ or whatever language you want, with radius 2 at a distance 8 from the origin, and visualize it. *Hint: the Delaunay operator creates a surface from a set of points.*[1] Display the sphere in VisIt.

3. You will be given an HDF5 file representing a vector field. If you don't have it, ask one of the seniors in the group. Visualize the vector field contained in this HDF5 file with a streamline plot. Create seed particles for streamlines and visualize these as well.

---

[1] The Delaunay operator is not enabled in VisIt by default. You may need to turn it on by going to *Options > Plugin manager*, and make sure to save settings. You'll have to restart VisIt.

5 Advanced Imaging

The goal of this section is to get you to use the skills and knowledge you have acquired so far. You will do this by visualizing the magnetic field of a current loop in two different ways. Upon successful completion of this exercise you should have the necessary background to start learning the daily work of the group.

5.1 Excercise Instuctions:

1. First you should generate the x, y,and z components of the magnetic field of a loop of current. The loop has a current of 4A and an area of $4\pi m^2$. Generate this data for an area of 12m×12m×12m. The resolution choice is up to you. However, you will be integrating over this data so choose wisely. Useful equations can be found:

   - https://en.wikipedia.org/wiki/Magnetic_moment
   - https://en.wikipedia.org/wiki/Biot-Savart_law
   - Griffiths EM textbook.

2. Now that you have a data file, visualize the field in VisIt.

3. Following that, generate a data file containing a collection of positions corresponding to points on the ring of current. You shall then use these points as seeding points for streamlines.

4. Your next task is to generate your own field lines and to visualize them in VisIt. You will use the seed points from above as starting points for each field line. For the integration, please implement the fourth order Runge-Kutta method. You can find it on Wikipedia.

5. Compare your field lines with those generated by VisIt. Then, just play around with the seed points until you start to get a feel for them.

6. Save a high quality image of the field and stream lines.

hese are mainly designed for pedantic purposes

6 Gravitational Waves

Just as how E-M waves come in two polarizations, so do gravitational waves, namely h_+ and $h_\times$. You will be generating movies of these two polarizations for various systems. The data that is obtained from the post-docs to generate gravitational wave movies is called `Psi4_rad.mon.3`. This file contains columns of data. The first column is the time column, and the columns after that are coefficients ψ_4^{lm} in the equation

$$\psi_4(t,r,\theta,\phi) = \sum_{l=2}^{\infty} \sum_{m=-l}^{l} \psi_4^{lm}(t,r) \cdot {}_{-2}Y_{lm}(\theta,\phi), \tag{1}$$

where the ψ_4^{lm} columns are evaluated at an extraction radius R. In equation 1, ψ_4 is decomposed into $s = -2$ spin-weighted spherical harmonics. From equation 1, we can obtain h_+ and $h_\times$ by

$$\psi_4 = \ddot{h}_+ - i\ddot{h}_\times \tag{2}$$

Our code, "hplus_hcross.cpp", takes ψ_4^{lm}, the ADM mass M in code units, and the extraction radius R in code units as inputs. It outputs h_+ and $h_\times$, from which we take and generate our movies. The code does this by fast Fourier transforming the equation

$$\sum_{l=2}^{\infty} \sum_{m=-l}^{l} \psi_4^{lm}(t,r) \cdot {}_{-2}Y_{lm}(\theta,\phi) = \ddot{h}_+ - i\ddot{h}_\times, \tag{3}$$

which is just a combining of equations 1 and 2. It then does a backward fast Fourier transform to obtain coefficients C_{lm} in the equation

$$\frac{1}{2}(h_+ - ih_\times) = \sum_{l,m} C_{lm} \cdot {}_{-2}Y_{lm},$$ (4)

from which we can obtain h_+ and $h_\times$. The ${}_{-2}Y_{2,2}$ and ${}_{-2}Y_{2,-2}$ modes dominate this sum, so we can approximate h_+ and $h_\times$ as

$$h_+ \approx 2\Re\left(C_{2,2} \cdot {}_2Y_{2,2} + C_{2,-2} \cdot {}_{-2}Y_{2,-2}\right)$$ (5)

$$h_\times \approx -2\Im\left(C_{2,2} \cdot {}_2Y_{2,2} + C_{2,-2} \cdot {}_{-2}Y_{2,-2}\right)$$ (6)

6.1 Calculating h_+ and $h_\times$

Zach's code is called to sum over the ${}_{-2}Y_{lm}$ modes and evaluate h_+ and $h_\times$ at a chosen θ and ϕ. It is assumed that in the far zone the amplitudes of h_+ and $h_\times$ drop off like $\frac{1}{R}$. Therefore, Zach's code removes the radial dependence and gives $R \cdot h_+$ and $R \cdot h_\times$. The output from Zach's code when it is passed a chosen (θ_i, ϕ_i) is $R \cdot h'_+(\theta_i, \phi_i)$ and $R \cdot h'_\times(\theta_i, \phi_i)$ evaluated at a series of values of $\frac{t}{M}$, each seperated from the next by some timestep Δt.

At a given point (r_j, θ_i, ϕ_i) (defined in a coordinate system with origin at the center of mass of the initial configuration) and time t_k, the amplitudes of h_+ and $h_\times$ will be

$$h_+ = \frac{h'_+(\theta_i, \phi_i)(t')}{r_j}$$

$$h_\times = \frac{h'_\times(\theta_i, \phi_i)(t')}{r_j}$$

where h'_+ and $h'_\times$ are the functions obtained from Zach's code and t' is the retarded time $t' = t - \frac{r_j}{c}$ (c being the speed of light, (taken to be $c = 1$). Since t only comes in discrete intervals, Δt, we choose radii $r_j = n\Delta t$ and times $t_k = m\Delta t$ for integer n,m at which to evaluate the amplitude of the wave. This ensures that the retarded time $t' = t_k - r_j = (m - n)\Delta t$ is an integer multiple of Δt and can be evaluated with the data given by Zach's code. If t' is less than zero then the amplitude of the wave is taken to be zero, because this implies that $r_j > ct$ (i.e. the wave has not yet reached this radius).

6.2 Data-Processing:

1. In order to make a gravity wave movie, you will be given a number of raw Psi4 datasets in the form of Psi4.rad.mon.#, together with black hole event horizen data and density data. You may also have a diagnostic datafile that gives you the ADM mass, extraction radius etc. Make sure you have all these parts before proceeding.

2. Next, choose which Psi4 file to work with. Generally speaking, we prefer to use a larger extraction radius since waveforms generated by numerical codes at smaller radius aroud black hole are less trustworthy. However, the problem is sometimes waveforms with larger extraction radius have abnormal behaviors, thus you need to check the Psi4 data yourself. One way to check the data is using gnuplot. For example, you can type in a terminal:
   gnuplot
   plot './Psi4.rad.mon.#' u 1:2 w l.

3. Due to checkpointing, the raw data has a lot of duplicated timesteps. Duplicate timesteps, and those containing a 'NaN' datum must be removed before proceding. You can try and create a script yourself however we have a convenient script called "clean_and_sort.bash" that will do it for you.

4. With the sorted data, you can proceed with our main processing code: hplus_hcross.cpp. The code will read-in the raw Psi4 data in certain radius, calculate the waveforms, and output to time-series VTK files organized into numbered folders.

5. There are a couple of things you need to check in the code. In the function Write3D, you will need to specify the ADM mass and the extraction radius for example. Then you will need to specify the spatial dimensions (usually we use 150*150) and spatial resolution (use an odd number to avoid singularity at z=0^2). Make sure the spatial resolution is neither too low nor too high because then the final data will be unmanagably huge (just gives you an idea, 151*151 may works fine).

6. To use the code run the "run_hphc.sh" script which should be in the root directory of abidbot. Make sure to set the params first!

6.3 Making gravity wave images

Time to actually make the gravity wave images! To start you will need a few scripts which should be abidbot: "runGW.sh", "/bin/gw_code/GW_up.py", and "/bin/gw_code/makeGW_movie.pbs"

1. In "runGW.sh" you will need to change a few things.

   - Choose which type of gravity wave you want to visualize (either h_+ or h_x)
   - Choose how many VTK folders you want to run. Each folder has 25 of each kind in it and we typically do 5 frames per rank.
   - Choose the job name (usually related to the type of gw)
   - Choose the pbs script you want to use (almost always "/bin/gw_code/makeGW_movie.pbs")
   - Choose the visit script you want to use (again almost always "/bin/gw_code/GW_up.py") For the gravity ways we typically just use the volume attributes that are hard coded into the file. (Don't ask me why we haven't made an xml for it yet)

2. Then just run the "runGW.sh" script in the root folder as ./runGW.sh and you're off to making a movie!

6.4 Superimposing density profiles

You may be asked to create a film where the density profile is superimposed over the gravitational radiation waveform. Unfortunately, VisIt can only render one volume plot via raycasting. The following method will circumvent this limitation.

1. Film a density movie with slightly different camera settings as your GW movie. We have found that if you use the same settings, the stellar objects will appear much bigger than they actually are comapred to the gravity waves. Here is what we have found has worked:

   - Keep the up vector, normal vector, focal point, and the zoom the same
   - Divide the parallel scale and the near and far clipping of the density view by 10

   If this doesn't look perfect, play around with the zoom until it does.

2. Next you will use our "prepare_gw_rho.py" script to match up the density times with the gravity wave times. We need to do this because the output frequency of the datasets is usually different. Typcially, the gravity wave data is outputed more frequently so we need to duplicate frames from the density movie so everything looks smooth.

3. Once that script is run, you will have two new directories holding the aligned data. At this point, to combine the images, you will want to open gimp, go to the "filters" tab, mouse over "Python-Fu" and click console. This will open up a small python2 console inside gimp.

4. Open the script "gw_rho_merger.py". Make sure all the paths to the folders are correct. Then simply copy the code (starting from the line with "root") into the console in gimp and hit enter twice. It will take a while to run as gimp has to open, edit, and close every single image.

---

^{2}This may no longer be true. I don't know.

Things to note:

- The resulting images will use the time from the gravity wave data so make sure that is correct before you start making images

- You may need to play with the opactites in the gimp script to get a better image

7 Python Scripting

VisIt can be scripted via python, which allows easy standardization of movie parameters. The scripting is relatively straightforward. Ask a senior member to provide a preexisting script, then adapt this script to automatically create a movie of the gravity wave you've been examining. Try adjusting the camera settings to zoom and rotate around the waveform.

Python Scripting: You are going to make a movie that will be filmed entirely using a python script. That means the data will be loaded, the plots will be set up, and your dataset will be iterated through time. Also the camera should move around all pretty and such. I'll have you create this script piece by piece so that its all not a giant mess.

You are going to need a time varying vector dataset. Its three sets of scalar fields that you are going to have to combine into a vector field in VisIt using expressions. Go make this.

Your first task is to open up your dataset using just the cli. Maybe there is a function that will do this for you. Maybe its called OpenDatabase(databaseName). Maybe databaseName will be set to something like `/home/projects/TutorialData/myfilename_*.vtk database`. Make sure that your database has opened without errors. Now would be a good time to make a script. In your favorite text editor (*cough* vim *cough*), open up a new document and call it whatever you want. Make sure to add .py to the end. At the top of your new script put "Source(/path/to/script.py)" This will help because you can't copy and paste into the cli normally because its stupid. Instead, you have to highlight what you want to copy and paste it by clicking with the middle mouse button (This magical thing you just did is a very simple copy and paste. highlighting text will copy it into a buffer, and clicking the middle mouse pastes it to where the cursor is. Learn to use this tactic, as it's going to be very handy). This helps when you need to rerun things and don't want to type a full path all the time.

Now that that's done, delete all the plots. You'll need this because otherwise if you rerun your script, you'll end up with multiple plots. Next we want to make a plot. For now we want a streamline plot.

Also make sure to add all the settings. Next you can do things like setup the settings for window saving, change the backround color to your favorite color, remove the bounding box, set your initial view angle, etc. There are functions that will do this. Usually one of them includes the words Get and Attributes. You can probably find it in the Python interface manual.

Now, you can begin coding the movie making loop. You'll want a for loop that iterates through each frame once (hint: range(GetNSteps())). For each iteraction, you'll set the timestep, make the changes that you want (i.e. add new streamline seeds, change the view angle, etc.), and call SaveWindow(). For now, just update the timestep and call SaveWindow(). With the above done, you should have a basic movie script. You can run your script in the cli using `Source(/path/to/script.py)`. To make things more interesting, you are now going to add code that will update the streamline plot so that the streamline seeds will change each timestep. This is pretty important for actual movies since we often use particles that move through time as seeds. The seed points are given to us (if they are particles) or we can generate them ourselves (we have to process them into .txt files though). This time, just generate some seeds. Edit your code that you used to visualize a rotating sphere to just give a txt file with some rotating points. Or, you could make moving points in some other way. It doesn't matter too much for the purpose of this exercise. To see a good amount of streamlines, I'd make it so there are about maybe 10 points total on the sphere (or whatever), but thats just a guestimate. You can do more or less depending on how you feel it will look. You can also always go back and change it. With this data and updated code, make a new movie where the streamline seeds are updated at each timestep. Now add code to make the camera pan across your data set. Probably use something like cubicevalspline to help you with this.

With that done, you've done most of the fancy things you need to do in an ordnary movie.

8   Blue Waters

The size of our simulation data is on the order of terabytes. Thus it requires a mighty machine to both store that data, and parse through it quickly. This is where Blue Waters comes in. Each simulation is a couple hundred frames long, and each frame takes about 20 minutes to create. Blue Waters allows us to generate these frames in a matter of minutes, rather than days. This is done using parallel programming. Read the getting started guide on the blue waters page https://bluewaters.ncsa.illinois.edu/getting-started.

You are now going to edit your movie script so that it can be run on Blue Waters. Hopefully by now, you guys have accounts on Blue Waters. If not you can just borrow anyone else's account for the purpose of this tutorial.

The first step is moving your data to Blue Waters. There are two ways to do this. One is to use a terminal command such as scp or rsync. The other is a program called globus online. Globus online should be used for transfering large amounts of data because that's how the BW guys want you to do it. You can use the terminal for smaller files such as scripts, seeds, and .txt files. Even though your data set is not very large, I'm going to have you transfer the data using globus online. Ask me how to do that when you get to this point. Then, when you are done with that scp or rsync the rest of your files over. Nodes on blue waters can be thought of as individual computers. There are two types of nodes found on any computing system of this type. The first is a login node. These are the nodes which you automatically ssh onto. The second are the compute nodes.

You can't log on to these nodes but you can use a command called qsub to submit jobs. The jobs are submitted to a queue and run when its your turn. The default compute node has something like 32GB of RAM split up between 16 processor cores. These nodes are all connected with a very fast ethernet connection so they can send data to each other really fast. They are also all connected to the same data on disk as well. Change your .py script so that the files you access correspond to where you data is on bw. Then go here https://bluewaters.ncsa.illinois.edu/visit and read the section on visit's command line interface and batch mode. You can use the sample .pbs file they give you. With that file in hand, call "qsub myfile.pbs" after editing the file.

You can check on your submission with "qstat -u username" (or ask one of the older guys for their ∼/.bashrc file). If that works, now try and "parallelize" your code. More specifically, you want to make it so that you can split up the frame making work amongst multiple computers. For example, say your dataset has 200 timesteps. You have 4 computers. That means you'll probably want to give each computer 50 timesteps to make pictures from. Make adjustments in your code so that when given the total number of computers and the computer number (we'll call it rank), you'll successfully split up the work and output frames for you movie. In our example, we have 4 computers total. One computer will have rank 0, another will have rank 1, etc. However, don't hardcode these numbers directly. Keep them as variables which you can pass to your script. To access them in your code, use `sys.argv`. If these concepts don't make sense, feel free to ask me for clarification.

8.1   Selected Exercises:

1. Log into Blue Waters using `ssh -X <username>@bw.ncsa.illinois.edu`

2. Open Visit using `module load visit; visit`. Check the settings to ensure that the Delaunay operator is loaded. Close VisIt.

3. Make a batch script following the recipe found online, and submit it to the blue waters job queue.

4. Bask in the glory of your life because you're using a heavenly supercomputer.

9   Particle Picker

We need particles to create a seeding points for magnetic fieldlines and to create particle movies. You will get a raw data file for particle position at all time, usually named as `bhns-particles.mon` or `particles.mon`. Our task is to choose some useful particles, and convert the file into a format that is readable in VisIt. This can be done by using the particle picker module.

You will need to transfer the abid bot to your Blue Waters. This is the ultimate code that you will be working with most of the time. For now we will skip most of the features and focus on particle picker module. The module is located in `/path/to/your/abid/bot/bin/particle_code/`. But there are a few steps you need to do before getting there.

1. Ask for the `particles.mon` file from previous REU members. Then copy it to `/path/to/your/abid/bot/h5data/`. Note that you need to rename it to `particles.mon` if it is named differently.

2. Change parameters inside the file `/path/to/your/abid/bot/params`. We are going to work with particle seeds, so set `fields=true`, `particleSeeds=true`, and `updateParticleMon=true`. Other parameters will be different from case to case. Ask others if you are stuck here for too long.

3. Now setup the parameters. First, enter `params` in command line. This will save parameters that you specified in params on your current shell. Next, normally you will run the `setup.sh` script to set everything. But since we are only doing particle picker, you can run abid bot partially by going into bin folder and run `setup_seeds.sh`. This step will take considerably long time, usually about 1+ min per GB of your `particles.mon` file. After this is done, go to `/path/to/your/abid/bot/bin/particle_code/` and check `dat` folder. You should have a bunch of files with the name like `00012.34567000000.dat` and the name of your first file should be `<firstTime>.dat`. If you get this, then cheers!, you've finished the most time consuming part of particle picker. Set `updateParticleMon=false` in params to skip this part in future runs, as it is only necessary to do so once.

Now go to `/path/to/your/abid/bot/bin/particle_code/`. There are three files involving particle picker: `particlePicker.py`, `particlePicker-tracer.py`, and `particlePickerModule.py`. Do not edit the `particlePickerModule.py` unless you know what you are doing. By all means, I encourage you to take a look inside and try to understand the module. The file you will be editing is either `particlePicker.py` or `particlePicker-tracer.py`. They are exactly the same except that the tracer one save files in `.3d` format instead of `.txt`. These script will pick particles based on your criteria. There are currently two methods to choose particles: `findInVolume` and `nearestNeighbor`. Details for these methods are in `particlePickerModule.py`.

Your task for this part is to make a simple particle tracer movie (only particle, without density and fieldlines). If you don't know what it is, please see
`http://research.physics.illinois.edu/cta/movies/NSNS_high_res/particles_ext.html`. You will have to edit `particlePicker-tracer.py`, set `particleTracer=true` in params, run `params`, and run `setup_seeds.sh`. If it is working correctly, you will get `trace*.3d` files inside trace1 folder. Copy these files to your local machine and open them in VisIt (You may need to rename them so that VisIt can see them as a database). Then plot them using Pseudocolor plot. I will let you play with VisIt and figure how to create a movie from here.

10   The Final Task

It's now time for you to get your hands on some real data. Ask one of the senior members to provide you with an old case to work with. This step can only be done on Blue Waters, as the data is too big for our local machines. While the data is transferring on your BW, transfer the file `abid_bot.tar.gz` file there if you have not done so. Go through the package (hint: README is a good place to start) and try to understand what's going on in different parts. Use this code to make a movie. This movie will be your certificate that you've successfully completed the tutorial.

11   Bonus

While the following sections are not part of the main tutorial, they are still good knowledge to have, so we have put theme together here as a bonus section. All are welcome to read through.

11.1 HDF5 data structure

This section is written with reference to the python module h5py, which can be very useful when you need to manipulate HDF5 (h5) files. A general picture of the content of an h5 file is something like this:
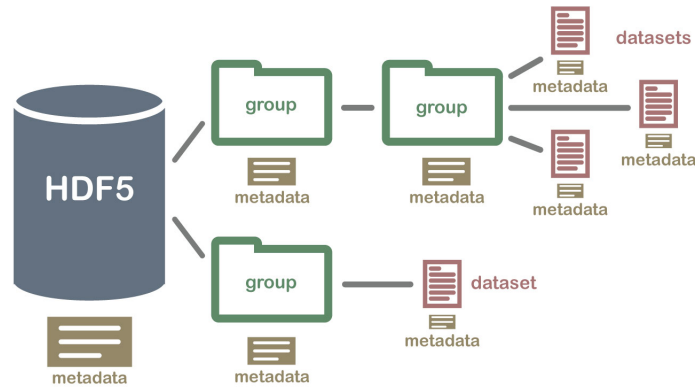

Figure 1: HDF5 data structure summary

11.1.1 Group

A group is somewhat analogous to a folder. A group can contain groups and other objects, or nothing. Two key differences between HDF5 groups and the conventional folder are:

12 Final Thoughts

Congratulations! While there are a couple of things that you will need to do, such as working with kdenlive and editing the website, those things are best taught by joining in the next time the group as a whole needs to use those tools. You should now be ready to jump right in to the group and start making science!