H5 Tutorials

Steven Walton
walton{dot}stevenj{at}gmail{dot}com
https://github.com/stevenwalton/tutorials
Last Updated:

March 16, 2016

What is H5 and why do I want it?

H5, or HDF5, is a file format created for loading and saving large datasets quickly and efficiently. Two common formats people may be familiar with already are ASCII and Binary. ASCII is easy to read, but is large and slow. Binary is fast, but can be difficult to migrate. HDF5 provides a standardized file format that anyone can use. It also provides easy organization by including a POSIX like structure. If you do not have the h5 headers installed you can either do it through your package manager of go to The hdfgroup's website to download it. Also, if you have Anaconda installed for use through Python, you will already have the h5 commands and libraries.

```
h5c++ -show
g++ -D_LARGEFILE64_SOURCE -D_LARGEFILE_SOURCE
    -L/home/steven/.anaconda/lib -lhdf5_hl_cpp -lhdf5_cpp -lhdf5_hl
    -lhdf5 -lrt -lz -ldl -lm -Wl,-rpath -Wl,/home/steven/.anaconda/lib
```

A common way to view the data that is inside an h5file would be by running the following command

```
h5dump -H file.h5
```

This will return the headers for the h5 file. This will give you the directory structure, showing the groups and array names, as well as the type of data that is stored. Another way to check the data is to open it up in python. I like to do this because it is easy to manipulate the data and to verify.

```
import h5py as h5
import numpy as np
f = h5.File("data.h5",'r')
f.keys()
# We will get back something like ['group1','data1']
# to check the contents of group1 we do
f['group1'].keys()
# we can check the data by
data = np.asarray(f['data'])
```

The hdfgroup has some tutorials to make and read some h5 files, but I find these difficult to learn from and may not be obvious for those not well versed in C++, but just need to use the data formats.

Reading Data

Let's say that you just want to read the data and turn it into a vector. I wrote a small function to do just that for you. It does not require you to actually know what size the data is and just requires you to know the name and the data type. Here we will assume IEEE standard float.

```
vector<float> readh5var(string path, string varName)
{
    H5std_string FILE_NAME(path);
    H5File file(FILE_NAME, H5F_ACC_RDONLY); // Opens file as read only
    DataSet dataset = file.openDataSet(varName);
    DataSpace dataspace = dataset.getSpace();
    // gets number of points
    const int arrSize = dataspace.getSimpleExtentNpoints();

    float *data = new float[arrSize]; // allocate array size at run time
    dataset.read(data, PredType::IEEE_F32BE); // read float data
    vector<float> v(data, data + arrSize); // convert array to vector
    // Protect our memory
    delete[] data;
    dataspace.close();
    dataset.close()
    file.close();
    return v;
}
```

Recognize here that we are using the standard namespace as well as the H5 namespace. The inputs to this function are the path to the h5 file and the full path of the float data that you wish to import. Be sure to include your h5 header and vector.

importAnyData

The code is too long to include here in full, so refer to the GitHub copy of the code.

This code will allow you to import any (STD_I32LE/IEEE_F32BE) data with a single getData call. This is accomplished with the use of a Proxy class. We cannot normally overload the return type of a function, but if we use this proxy class we can cheat our way around that.

To do this we first need to create our regular functions. In this case we have a function that will import and return STD_I32LE data to a integer vector and another that does the same for IEEE_F32BE data (back to vector⟨float⟩)

To modify this code you need to create another normal like function for the new data type. Note that if you use a different type of float, even if you want to return vector⟨float⟩ you will need to change some code because we explicitly state that we are looking for IEEE_F32BE data in our import function.

The code itself contains a lot of comments, but does build upon previous work.

writeAnyData

Again we are only using the long int (my native int) and the IEEE standard. In this case we do not need to overload the return type and use a proxy class. This is much simpler and we can just overload the input. In this case we can get our writeData function to do different things if it is passed an int vector or a float vector. We really could get it to do the same thing for any datatype. For example if you also wanted to be able to pass it arrays, not vectors, you could do the same thing there. Function overloading is an extremely useful practice.