storeAndManageDataEffortlesslyWithHDF5

March 13, 2016

1 What is HDF5?

HDF5 stands for (**H**)eirarchical (**D**)ata (**F**)ormat, v**5**

An HDF5 file is a container for two main kinds of objects: * datasets: array-like collections of data * groups: folder-like containers that hold datasets and other groups

AND attributes: metadata on datasets or groups

The most fundamental thing to remember when using h5py is:

Datasets work like NumPy arrays and groups work like dictionaries

You should also remember:

Every object in an HDF5 file has a name, and they're arranged in a POSIX-style hierarchy with /-separators

1.1 Getting started with HDF5 in Python

1.1.1 Imports & data setup

```
In [1]: import numpy as np
        import h5py
        import os

        # Can also use HDF5 in PyTables, but won't be covered in this notebook
```

If installed Anaconda, simply

```
conda install h5py
```

```
In [2]: # Create data
        sm_array = np.random.random(size = (3,3))
        sml_array = np.random.random(size = (2,2))
```

1.1.2 Creating & handling HDF5 files

```
In [3]: # Create HDF5 file
        f = h5py.File("data.hdf5", 'w')
```

HDF5 files work like standard Python file objects, support standard modes, and should be closed when no longer in use.

- 'r': Read only, file must exist
- 'r+': Read/write, file must exist
- 'w': Create file, truncate if exists
- 'w-' or 'x': Create file, fail if exists
- 'a': Read/write if exists, create otherwise (default)

1.1.3 Working with datasets, groups, & attributes

Datasets

- Like NumPy arrays
  - Collections of data elements
  - Immutable datatype and (hyper)rectangular shape
  - Descriptive attributes: shape, size, dtype
  - Data slicing
- Unlike NumPy arrays
  - Compression
  - Chunked-I/O

```
In [4]: # IPython reminder trick #1
        # Attributes & properties on an object (tab complete)
        #f.
```

```
In [5]: # IPython reminder trick #2
        # See the docstring and all the details on an object by using the ?
        #f.create_dataset?
```

```
In [6]: # Create a dataset
        dataset = f.create_dataset("data", data=sm_array)
        # Keywords shape and dtype may be specified along with data
        # If so, they will override data.shape and data.dtype
        print "Dataset dataspace is", dataset.shape
        print "Dataset datatype is", dataset.dtype
        print "Dataset name is", dataset.name
        print "Dataset is a member of the group", dataset.parent
```

```
Dataset dataspace is (3, 3)
Dataset datatype is float64
Dataset name is /data
Dataset is a member of the group <HDF5 group "/" (1 members)>
```

```
In [7]: # Alternatively
        f['sm_data'] = sml_array
        dset = f['sm_data']
        print "Dataset dataspace is", dset.shape
        print "Dataset datatype is", dset.dtype
        print "Dataset name is", dset.name
        print "Dataset is a member of the group", dset.parent
```

```
Dataset dataspace is (2, 2)
Dataset datatype is float64
Dataset name is /sm_data
Dataset is a member of the group <HDF5 group "/" (2 members)>
```

```
In [8]: f.close()
        # Command line utilities h5dump and h5stat useful to see info about file
        !h5dump data.hdf5
```

```
HDF5 "data.hdf5" {
GROUP "/" {
   DATASET "data" {
```

```
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SIMPLE { ( 3, 3 ) / ( 3, 3 ) }
      DATA {
      (0,0): 0.30961, 0.631119, 0.314471,
      (1,0): 0.521217, 0.257428, 0.820622,
      (2,0): 0.795544, 0.588312, 0.760122
      }
   }
   DATASET "sm_data" {
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SIMPLE { ( 2, 2 ) / ( 2, 2 ) }
      DATA {
      (0,0): 0.869497, 0.752933,
      (1,0): 0.0906557, 0.737728
      }
   }
}
}
```

```
In [9]: # Working with subsets of data using NumPy syntax for data slicing
        f = h5py.File("data.hdf5", 'w')
        f['intArray'] = np.ones((3,4))
        dset2 = f['intArray']
        dset2[...]

Out[9]: array([[ 1.,  1.,  1.,  1.],
               [ 1.,  1.,  1.,  1.],
               [ 1.,  1.,  1.,  1.]])

In [10]: f['intArray'][:,2:] = 2
         dset2[...]

Out[10]: array([[ 1.,  1.,  2.,  2.],
                [ 1.,  1.,  2.,  2.],
                [ 1.,  1.,  2.,  2.]])

In [11]: f.close()
```

Groups

- Like Python dictionaries
  - keys - names of group members
  - values - group members (either dataset or group objects)
  - support iteration, indexing syntax, and standard exceptions
- Objects in an HDF5 file can be stored in multiple groups

```
In [12]: # File object is the *root group* and serves as entry point into the file.
         f = h5py.File('data.hdf5', 'w')
         print f.name

/

In [13]: # Create a group
         grp = f.create_group("group1")
         print grp.name
```

```
/group1
```

```
In [14]: # Create a group within a group
         subgrp = grp.create_group("subgrp")
         print subgrp.name
```

```
/group1/subgrp
```

```
In [15]: # Create groups implicitly
         grp2 = f.create_group("/group2/subgrp2/anothergroup")
         print grp2.name
         grp3 = f['/group2/subgrp2']
         print grp3.name
```

```
/group2/subgrp2/anothergroup
/group2/subgrp2
```

```
In [16]: f.keys()
```

```
Out[16]: [u'group1', u'group2']
```

```
In [17]: f['group1'].keys()
```

```
Out[17]: [u'subgrp']
```

```
In [18]: # Group objects have create_* methods like files
         # Here, creating a dataset in a group
         dataset2 = grp.create_dataset("small", data=sm_array)
         dataset2.name
```

```
Out[18]: u'/group1/small'
```

```
In [19]: # More ways to create a dataset in a group
         f['/group2/subgrp2/anothergroup/dset1'] = [4,4] # create dataset
         g = f['group2/subgrp2/anothergroup'] # create group
         g['dset2'] = [5,5] # create dataset in group
         dset3 = f.create_dataset('group2/dset3', data=[6,6]) # create dataset in group
```

```
In [20]: f.close()
         !h5dump data.hdf5
```

```
HDF5 "data.hdf5" {
GROUP "/" {
   GROUP "group1" {
      DATASET "small" {
         DATATYPE  H5T_IEEE_F64LE
         DATASPACE  SIMPLE { ( 3, 3 ) / ( 3, 3 ) }
         DATA {
         (0,0): 0.30961, 0.631119, 0.314471,
         (1,0): 0.521217, 0.257428, 0.820622,
         (2,0): 0.795544, 0.588312, 0.760122
         }
      }
      GROUP "subgrp" {
      }
   }
   GROUP "group2" {
```

```
      DATASET "dset3" {
         DATATYPE  H5T_STD_I64LE
         DATASPACE  SIMPLE { ( 2 ) / ( 2 ) }
         DATA {
         (0): 6, 6
         }
      }
      GROUP "subgrp2" {
         GROUP "anothergroup" {
            DATASET "dset1" {
               DATATYPE  H5T_STD_I64LE
               DATASPACE  SIMPLE { ( 2 ) / ( 2 ) }
               DATA {
               (0): 4, 4
               }
            }
            DATASET "dset2" {
               DATATYPE  H5T_STD_I64LE
               DATASPACE  SIMPLE { ( 2 ) / ( 2 ) }
               DATA {
               (0): 5, 5
               }
            }
         }
      }
   }
}
}
```

```
In [21]: f = h5py.File("data.hdf5", 'r+')
         'data' in f # containership testing
```

```
Out[21]: False
```

```
In [22]: '/group1/small' in f   # full path names works too
```

```
Out[22]: True
```

Attributes All groups and datasets support attached named bits of data called attributes (i.e., metadata), dictionary-style objects. Attributes are a critical part of what makes HDF5 a self-describing format.

Attributes have the following properties: * created from any scalar or NumPy array * small (generally < 64k) * no partial I/O (i.e. slicing); the entire attribute must be read

```
In [23]: # Recall, dataset2 is the small array
         dataset2 = f['group1/small']
         dataset2.attrs['sampling rate'] = 10e5
         dataset2.attrs['task-type'] = 'rest'
```

```
In [24]: dataset2.attrs.keys()
```

```
Out[24]: [u'sampling rate', u'task-type']
```

```
In [25]: dataset2.attrs.items()
```

```
Out[25]: [(u'sampling rate', 1000000.0), (u'task-type', 'rest')]
```

```
In [26]: # Can also label the dimensions of a dataset
         f['group1/small'].dims[0].label = 'x'
         f['group1/small'].dims[1].label = 'y'
```

```
In [27]: f.close()
         !h5dump data.hdf5
```

```
HDF5 "data.hdf5" {
GROUP "/" {
   GROUP "group1" {
      DATASET "small" {
         DATATYPE  H5T_IEEE_F64LE
         DATASPACE  SIMPLE { ( 3, 3 ) / ( 3, 3 ) }
         DATA {
         (0,0): 0.30961, 0.631119, 0.314471,
         (1,0): 0.521217, 0.257428, 0.820622,
         (2,0): 0.795544, 0.588312, 0.760122
         }
         ATTRIBUTE "DIMENSION_LABELS" {
            DATATYPE  H5T_STRING {
               STRSIZE H5T_VARIABLE;
               STRPAD H5T_STR_NULLTERM;
               CSET H5T_CSET_ASCII;
               CTYPE H5T_C_S1;
            }
            DATASPACE  SIMPLE { ( 2 ) / ( 2 ) }
            DATA {
            (0): "x", "y"
            }
         }
         ATTRIBUTE "sampling rate" {
            DATATYPE  H5T_IEEE_F64LE
            DATASPACE  SCALAR
            DATA {
            (0): 1e+06
            }
         }
         ATTRIBUTE "task-type" {
            DATATYPE  H5T_STRING {
               STRSIZE H5T_VARIABLE;
               STRPAD H5T_STR_NULLTERM;
               CSET H5T_CSET_ASCII;
               CTYPE H5T_C_S1;
            }
            DATASPACE  SCALAR
            DATA {
            (0): "rest"
            }
         }
      }
      GROUP "subgrp" {
      }
   }
   GROUP "group2" {
      DATASET "dset3" {
```

```
         DATATYPE  H5T_STD_I64LE
         DATASPACE  SIMPLE { ( 2 ) / ( 2 ) }
         DATA {
         (0): 6, 6
         }
      }
      GROUP "subgrp2" {
         GROUP "anothergroup" {
            DATASET "dset1" {
               DATATYPE  H5T_STD_I64LE
               DATASPACE  SIMPLE { ( 2 ) / ( 2 ) }
               DATA {
               (0): 4, 4
               }
            }
            DATASET "dset2" {
               DATATYPE  H5T_STD_I64LE
               DATASPACE  SIMPLE { ( 2 ) / ( 2 ) }
               DATA {
               (0): 5, 5
               }
            }
         }
      }
   }
}
}
```

1.1.4 What else can I do with HDF5?

Chunking

- Datasets created with the default settings will be contiguous (laid out on disk in traditional C order)
- Datasets created with the chunked storage will be divided up into regularly-sized pieces which are stored haphazardly on disk, and indexed using a B-tree
    - Recommended to keep larger chunks for larger datasets
    - Note: when any element in a chunk is accessed, the entire chunk is read from disk

In [28]:
```python
f = h5py.File("data2.hdf5", 'w')
# Set the keyword chunks to a tuple indicating the chunk shape:
dset = f.create_dataset("chunked", (1000, 1000), chunks=(100, 100))
# In this case, data will be read and written in blocks with shape (100,100)
```

In [29]:
```python
# Let h5py decide your chunk shape
dset = f.create_dataset("autochunk", (1000, 1000), chunks=True)
```

Filters Data is compressed on the way to disk and decompressed when read. Once the dataset is created with a particular compression filter applied, data may be read and written as normal with no special steps required.

In [30]:
```python
# Enable a specific type of compression
dset = f.create_dataset("zipped", (100, 100), compression="gzip")
```

Parallel HDF5

- It uses the MPI (Message Passing Interface) standard for interprocess communication accomplished through the mpi4py Python package.
- To use parallel HDF5, must do a separate build, although a parallel version of HDF5 might be available through your package manager.

More HDF5 Python Examples: https://www.hdfgroup.org/HDF5/examples/py.html