

Wikiprint Book

Title: PyBc/PyTables

Subject:

Version: 19

Date: 09/09/10 08:33:31

Table of contents

Python/C Extension: PyTables	3
What are HDF5 files?	3
Viewing HDF5 files	3
Using PyTables	3
Writing our own datasets	4

Error: Macro TOC(PyBc, PyBc/Session01, PyBc/Session02, PyBc/Session03, PyBc/Session04, PyBc/Session05, PyBc/Session06, PyBc/Session07, PyBc/Session08, PyBc/Session09, PyBc/f2py, PyBc/swig, PyBc/Cpython, PyBc/Cython, PyBc/PyTables, PyBc/PyTaps, PyBc/PythonBots, PyBc/Django, PyBc/GIS, PyBc/AdvancedPython, PyBc/WxPython, PyBc/standardlib, depth=1) failed

unsupported operand type(s) for +=: 'NoneType' and 'unicode'

Python/C Extension: PyTables

Written by: Milad Fatenejad

Typically, scientific applications need to write data to the hard disk at some point. Traditionally, you have two choices: to write text files or binary files. Each has their benefits and draw backs:

- Text files are easy to read (you can just open them and look at them) and basically platform independent (there are some issues with line endings, but you can typically deal with these). Unfortunately, they also take up a lot of space because you need to write one or more bytes per character.
- Binary files store data in less space than text files, but they are platform dependent and sometimes language dependent. They are also difficult to view.
- Writing/Reading your own text/binary files directly is also difficult when you start generating large amounts of data.

HDF5 files exist to solve a lot of these problems. The HDF5 library was originally written in C and pytables allows you to access HDF5 files in python. Just so you know, pytables is my favorite python package.

What are HDF5 files?

HDF stands for "Hierarchical Data Format". HDF5 files are binary files where the data is stored in a hierarchical structure. Essentially, HDF5 files are organized like a directory structure. You create "groups" which represent directories. You can then place data within those groups. The data is stored in "datasets" within the groups. You can think of a dataset as an array storing some information - such as an array of numbers. The top group, which contains everything in the file, is called "root".

Viewing HDF5 files

You can view the contents of HDF5 files using a tool called vitables which, incidentally, is written in python. The file is represented in a tree like structure. You can view the contents of the groups/datasets in the file and modify them.

Using PyTables

I've [attached](#) an example HDF5 file to this wiki page that I will use to demonstrate some of the features of pytables. Download the file to your Desktop, fire up ipython, and enter "cd" then "cd Desktop" to navigate to your Desktop. All of the pytables code is stored in the tables module. So the first step is to import tables:

EXAMPLE: SELECT ALL

```
import tables
```

Now, lets open our test file, "free.h5":

EXAMPLE: SELECT ALL

```
h5file = tables.openFile("free.h5", "r")
```

The "r" tells pytables we are opening the file for reading. You can print the contents of the file by entering:

EXAMPLE: SELECT ALL

```
print h5file
```

You should then see something like:

```
free.h5 (File) ''
Last modif.: 'Wed Jan 13 22:29:30 2010'
```

```

Object Tree:
/ (RootGroup) ''
/dims (Array(3,)) ''
/checkpoints (Group) ''
/materials (Group) ''
/streams (Group) ''
/streams/energy (Table(1000,)) 'Energy Information'
/streams/time (Table(1001,)) 'Time Information'
/materials/Helium4 (Group) ''
/checkpoints/checkpoint_1000 (Group) ''
/checkpoints/checkpoint_1000/position (Array(601, 2, 2)) ''
/checkpoints/checkpoint_1000/temperature (Array(600, 1, 1)) ''
/checkpoints/checkpoint_1000/velocity (Array(601, 2, 2)) ''
/checkpoints/checkpoint_500 (Group) ''
/checkpoints/checkpoint_500/position (Array(601, 2, 2)) ''
/checkpoints/checkpoint_500/temperature (Array(600, 1, 1)) ''
/checkpoints/checkpoint_500/velocity (Array(601, 2, 2)) ''

```

This is a listing of every group and dataset in the file. For example, we can see that there is a group called "checkpoints" within the root group. We can also see that the root group contains a dataset, called "dims" which is an array of length three. Lets find out what this dataset contains. To do this, I have read the contents of the dataset into a numpy array. I can then print the numpy array.

EXAMPLE: SELECT ALL

```

import numpy as np
arr = np.array(h5file.root.dims)
print arr

```

You should then see that the dataset contains the array (600, 1, 1). Notice how easy it is to access a dataset.

Hands-on Example

- Load the data inside the dataset "temperature" stored in the group "checkpoint_1000" into a numpy array.
- Print the array
- Print the dimensions (shape) of the array
- Reshape the array into a 1D array of length 600 using the reshape function.

At this point, you should have a 1D array ready to use. I've called mine "arr". This array contains the temperature from a simulation I ran at time-step 1000. Lets plot the temperature:

EXAMPLE: SELECT ALL

```

from matplotlib import pyplot as plt
plt.plot(arr)
plt.show()

```

I hope this simple example has demonstrated reading datasets from HDF5 files is simple with pytables. One thing to note: pytables doesn't actually read any data from the file until we load it into a numpy array. Because of this pytables, is able to handle arbitrarily large files which, unfortunately, is often what we are stuck with in scientific computing.

Writing our own datasets

Lets start by creating two arrays to write to the file. I've plotted the arrays so that we know what we're dealing with. Remember, if you've forgotten what a function does, use the "help" function to find out.

EXAMPLE: SELECT ALL

```
import numpy as np
from math import pi
xvals = np.linspace(0, 2*pi, 1000)
yvals = np.sin(xvals) + 0.2*np.sin(10*xvals)

from matplotlib import pyplot as plt
plt.plot(xvals, yvals)
plt.show()
```

Now lets create a new file called "test.h5".

EXAMPLE: SELECT ALL

```
import tables
h5file = tables.openFile("test.h5", "w")
```

The "w" tells pytables we are opening a file for writing. Since the file doesn't exist, pytables will make it for us. Now, lets make a group called "data" within the root group:

EXAMPLE: SELECT ALL

```
h5file.createGroup("/", "data")
```

Finally, we'll add our two arrays to the "data" group using the "createArray" method:

EXAMPLE: SELECT ALL

```
h5file.createArray("/data", "xvals", xvals)
h5file.createArray("/data", "yvals", yvals)
h5file.flush()
```

After I created the arrays, I used the "flush" method to instruct HDF5 to write all of the data to the files. Sometimes, your computer may decide not to actually write the data to the hard drive because it is not convenient - the "flush" method will force the write operation.

I've now decided that I don't like the "data" group. Lets destroy it. Afterwards, I close the HDF5 file.

EXAMPLE: SELECT ALL

```
h5file.root.data._f_remove(recursive=True)
h5file.close()
```

The "_f_remove" method can be used to remove anything in the HDF5 file. However, because "data" contained other HDF5 datasets, we had to set the recursive keyword argument to True so that pytables knows that we really really want it gone.

Thats all I have time for. I hope that this little example demonstrates the ease of use of pytables and power of HDF5. There is much, much more to the HDF5 and pytables that I didn't have time to go into but I encourage you to learn about it.

I'll leave you with some hands on exercises:

Hands-on Example

- Create an HDF5 file called "test2"
- Create a numpy array which contains the 100 numbers evenly spaced between 1 and 2. Call the array "xvals"
- Create another numpy array which contains the log of the first array. Call this array "yvals"
- Write the two arrays directly to the root group with the names "xvals" and "yvals".
- Figure out (using dir and help) how to rename the datasets, then rename "xvals" to "logxvals" and rename "yvals" to "logyvals"

- Delete the two datasets you just created.