For internal purposes only! (Copyright/Citing issues)
Recommended book: 'Python and HDF5 - Unlocking scientific Data'

1 HDF5

HDF5 is short for hierarchical data format version 5 and is nowadays the *de facto* standard for scientific data. Its usage is heavily encouraged when working on super computers for various reasons, which we will briefly explain.

HDF5 is a great way to store large numerical arrays of homogeneous type, for data models that can be organized hierarchically and benefit from tagging of datasets with arbitrary metadata. The three main elements of the HDF5 data model are *datasets* (array-like objects), *groups* (hierarchical containers that store datasets and other groups) and *attributes* (user-defined bits of metadata that can be attached to datasets and groups.

The files are *self-describing* (one can search through the group-structure and explore the data) and the file specification is open-source with a large ecosystem built around it. It includes an official C- and Fortran-API, and a Python library (`h5py`) built on top of the C-API. It goes so far that even the latest netCDF4 standard is built on top of the HDF5.

The Python library mentioned before (*h5py*) allows for easy access to the HDF5 file (in comparison to the official APIs which are quite cumbersome to work with). In combination with other scientific libraries like *numpy*, *matplotib*, *scipy*, etc. one gets a good and easy-to-use environment for scientific work in Python.

2 HDF5 - h5ls

If one wants to quickly glance over the structure of an HDF5 file or even wants to look at small arrays in the command line, one can use `h5ls` (comes with the hdf5 installation). Usage (h5ls -h for all possible flags):

- `h5ls -lr file.hdf5`: (r)ecursivly (l)ist all groups
- `h5ls -vlr file.hdf5`: (r)ecursivly (l)ist all groups with extended information about the datasets
- `h5ls -d file.hdf5/group/dataset` : display the content of the given dataset. The values here are displayed according to their contiguous memory position.

To name a few other tools: *HDFView* (graphical), *ViTables* (graphical), *h5dump* (CLI).

3 HDF5 - h5py

H5py represents a Python wrapper around the official HDF5 C-API. I recommend reading the book given at the beginning for all the information about the library. To give a short introduction we will show how to extract datasets, how to manipulate them within numpy and how to plot the data in matplotlib. (Please refer to the actual numpy and matplotlib introductions for more details.)

Plotting 1D data

```
from __future__ import print_function, division
import numpy as np
import h5py
import matplotlib.pyplot as plt

# open the file in the read-format
f = h5py.File('adga-12345.hdf5','r')

# extract the data into a numpy array
dset = f['selfenergy/nonloc/dga'][()]

# this dataset has the form of ndim,ndim,npx,npy,npz,2*iwf
print(dset.shape)        # prints the shape of the array

# plot the first band at the gamma-point
plt.plot(dset[0,0,0,0,0,:].imag)

# show the figure now
plt.show()
```

Plotting 2D data

```
from __future__ import print_function, division
import numpy as np
import h5py
import matplotlib.pyplot as plt

f = h5py.File('adga-12345.hdf5','r')

# extract the fermionic vertex box size
iwf = f['input/iwfmax_small'][()]

dset = f['selfenergy/nonloc/dga'][()]

# plot the first band at the first fermionic frequency
# in the kz = 0 plane
f = plt.figure()

f.add_subplot(211) # 2x1 subplots
plt.pcolormesh(dset[0,0,:,:,0,iwf].imag)

f.add_subplot(212)
plt.pcolormesh(dset[0,0,:,:,0,iwf].real)

plt.show()
```

Building the DΓA Green's function

```
from __future__ import print_function, division
import numpy as np
import h5py
import matplotlib.pyplot as plt
import scipy.linalg

# open the file in the read-format
f = h5py.File('adga-12345.hdf5','r')

# extract the necessary components
siwk_dga = f['selfenergy/nonloc/dga'][()]
hk = f['input/hk'][()]
dc = f['input/dc'][:,0]
mu = f['input/mu'][()]
iwf = f['input/iwfmax_small'][()]
ndim = hk.shape[0]
nqx = f['input/nqpxyz'][0]
nqy = f['input/nqpxyz'][1]
nqz = f['input/nqpxyz'][2]

# create the matsubara axis
fmats = np.linspace(-(iwf*2-1)*np.pi/beta,(iwf*2-1)*np.pi/beta,2*iwf)

# building DGA Greens function from scratch
# according to [iw + mu - dc - H(k) - Sigma(k,iw)]**(-1)

gdgainv = np.zeros((ndim,ndim,nqx,nqy,nqz,2*iwf),dtype=np.complex128)
gdgainv += -siwk_dga - hk[...,None]
gdgainv[np.arange(ndim),np.arange(ndim),...] \
        += 1j*fmats+mu-dc[np.arange(ndim),None,None,None,None]

gdga = np.empty_like(gdgainv, dtype=np.complex128)

for ikx in xrange(nqx):
    for iky in xrange(nqy):
        for ikz in xrange(nqz):
            for iw in xrange(2*iwf):
                    gdga[:,:,ikx,iky,ikz,iw] = \
                    scipy.linalg.inv(gdgainv[:,:,ikx,iky,ikz,iw])

# this time with the actual matsubara axis
plt.plot(fmats, gdga[0,0,0,0,0,:].imag)
plt.show()
```

How to build a minimal DMFT file

Here we assume that the user has all necessary data for starting a DΓA run, just not in the correct file format. The following script contains the necessary commands to create a minimal input file in the correct format.

```
import h5py

def read_1p_data(iw,siw,giw,dc,mu,beta)
  # read in your data according to your format.
  # save them in iw,siw,giw,dc,mu,beta as necessary for ADGA
  pass


read_1p_data(iw,siw,giw,dc,mu,beta)
f=h5py.File('1p-data.hdf5','w')
f['.axes/iw']=iw
f.create_group('.config')
f['.config'].attrs['general.beta']=beta
f['dmft-001/mu/value']=mu
f['dmft-001/ineq-001/dc/value']=dc
f['dmft-001/ineq-001/siw/value']=siw
f['dmft-001/ineq-001/giw/value']=giw
f.close()
```