Basic Introduction to HDF5

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.
The AbinitioDΓA project makes extensive use of HDF5. This format makes it possible to store large numerical data, while still preserving original array structures. 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. For a detailed description and tutorials, please refer to `https://support.hdfgroup.org/documentation/` and `http://docs.h5py.org/en/latest/`.
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*, *scipy* (`https://docs.scipy.org/doc/`), *matplotib* (`https://matplotlib.org/contents.html`), etc. one gets a good and easy-to-use environment for scientific work in Python (`https://docs.python.org/2/index.html`).

2 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).

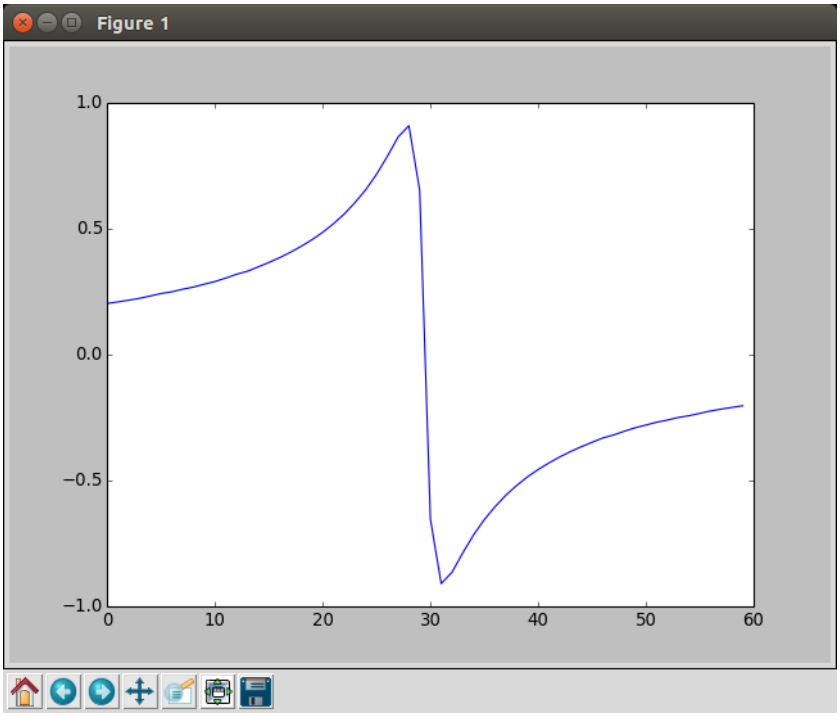
3 h5py

H5py represents a Python wrapper around the official HDF5 C-API. It is recommended to read all the documentations given above. 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.

Plotting 1D data: `1D_plot_minimal.py`

```
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()
```

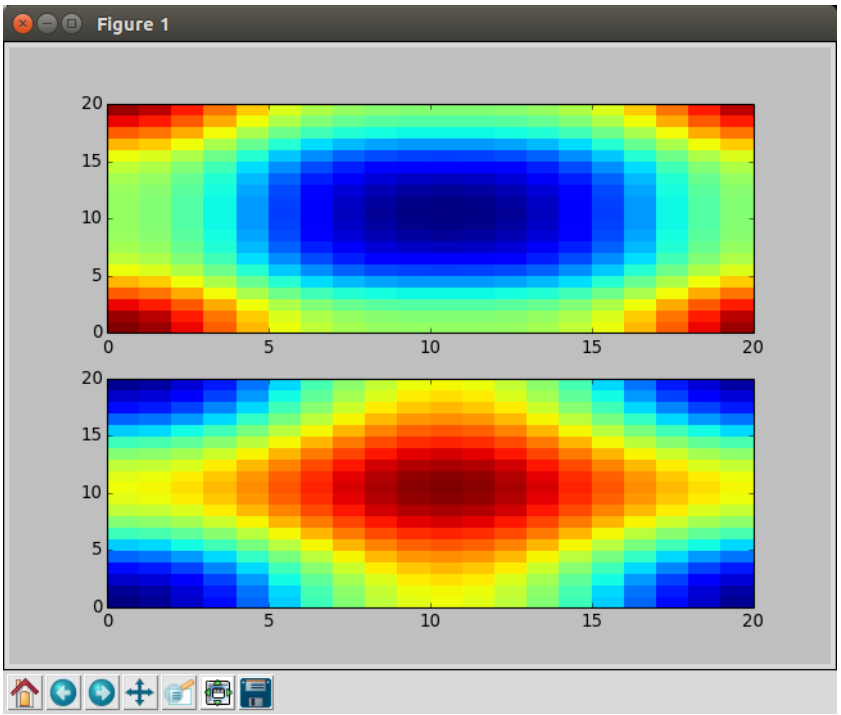
If everything works correctly, you should get the following plot:



Plotting 2D data: `2D_plot_minimal.py`

```
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()
```

If everything works correctly, you should get the following plot:



If you want to make a proper plot of the full Brillouin zone, please have a look at the script `documentation/plot_scripts/2D_plot_selfenergy_plane.py`.

How to build a minimal DMFT file: `construct_input_structure.py`

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.

```
#! /usr/bin/env python

from __future__ import print_function, division, absolute_import
import numpy as np
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

def read_2p_data(group, g2):
    # read in one spin-band group of your two-particle data
    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()


g = h5py.File('2p-data.hdf5','w')

# groups holds all spin-band combinations of
# the two particle Greens function
# as strings, e.g. '00001'
for group in groups:
    read_2p_data(group, g2)
    g['worm-001/ineq-001/g4iw-worm/'+group+'/value'] = g2
g.close()


n4iwf=g2.shape[0]//2
n4iwb=g2.shape[-1]//2

g['.axes/iwf-g4'] = np.pi/beta * (2 * np.arange(-n4iwf,n4iwf) +1)
g['.axes/iwb-g4'] = np.pi/beta * 2 * np.arange(-n4iwb,n4iwb+1)

g.close()
```