3_reading_hdf5_complex_voltage_data

March 24, 2021

1 Tutorial: HDF5 complex voltage data

This tutorial will cover how to read beamformed complex voltage data in HDF5 format.

Again, let's load the python libraries that we will use.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import h5py
import os
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

As we covered in the lecture, complex voltage data is located over 4 files, with the real and imaginary values of polarizations X and Y mapped to files as follows: S0 = x_r, S1 = x_i, S2 = y_r and S3 = y_i.

Let's start by opening the S0 and S1 files of OBSID L645495 and reading the attributes of the `STOKES_0` group in the S0 file. Note that to read the `raw` files, the *h5py* library requires the path to be that of the files, so use `os.chdir` to change the directory to that of the data.

[2]:
```
os.chdir("../../data/L645495")
h5_s0 = h5py.File("L645495_SAP000_B000_S0_P000_bf.h5", "r")
h5_s1 = h5py.File("L645495_SAP000_B000_S1_P000_bf.h5", "r")
```

[3]:
```
group = h5_s0["/SUB_ARRAY_POINTING_000/BEAM_000/STOKES_0"]
keys = sorted(["%s"%item for item in sorted(list(group.attrs))])
for key in keys:
    print(key + " = " + str(group.attrs[key]))
```

```
DATATYPE = float
GROUPTYPE = bfData
NOF_CHANNELS = [1 1 1 1 1 1 1 1 1 1]
NOF_SAMPLES = 117178368
NOF_SUBBANDS = 10
STOKES_COMPONENT = Xr
```

1.1 Exercise 1

Have a look at the output above. You'll see that the data is stored as floating point values, with 10 subbands per file. As expected, S0 contains the real value of X.

1. Now read the S1 file and inspect the Stokes group. Can you confirm that this contains the imaginary part of X?
2. Read the attribute storing the number of subbands.
3. Read the attribute storing the sampling time. Remember from the earlier tutorial that this is stored in `INCREMENT` attribute of the `/SUB_ARRAY_POINTING_XXX/BEAM_YYY/COORDINATES/COORDINATE_0` group.
4. Read the attribute storing the subband frequencies.

[4]:
```
group = h5_s1["/SUB_ARRAY_POINTING_000/BEAM_000/STOKES_1"]
keys = sorted(["%s"%item for item in sorted(list(group.attrs))])
for key in keys:
    print(key + " = " + str(group.attrs[key]))
```

```
DATATYPE = float
GROUPTYPE = bfData
NOF_CHANNELS = [1 1 1 1 1 1 1 1 1 1]
NOF_SAMPLES = 117178368
NOF_SUBBANDS = 10
STOKES_COMPONENT = Xi
```

[5]:
```
tsamp = h5_s0["/SUB_ARRAY_POINTING_000/BEAM_000/COORDINATES/COORDINATE_0"].
 ↪attrs["INCREMENT"]
nsub = h5_s0["/SUB_ARRAY_POINTING_000/BEAM_000/STOKES_0"].attrs["NOF_SUBBANDS"]
freq = h5_s0["/SUB_ARRAY_POINTING_000/BEAM_000/COORDINATES/COORDINATE_1"].
 ↪attrs["AXIS_VALUES_WORLD"]
print(tsamp, nsub, freq)
```

```
5.12e-06 10 [1.37109375e+08 1.37304688e+08 1.37500000e+08 1.37695312e+08
 1.37890625e+08 1.38085938e+08 1.38281250e+08 1.38476562e+08
 1.38671875e+08 1.38867188e+08]
```

1.2 Exercise 2

We now have the information to read the data in the `STOKES_0` and `STOKES_1` groups. Since the input files have 4.4GB of data each, it is better to read in parts of the timeseries, as the entire timeseries would fill up most of the memory of the computer. 1. We want to extract the timeseries between 300 and 310 seconds from the start of the observation. Using the sampling time, compute the array indices belonging to these times. (Remember that indices need to be integers). 2. Use these indices to extract the real and imaginary components of X. What is the shape of the output (use the `.shape` function of `numpy` arrays)? 3. Create an array with the time of each sample in the selected range. 4. Use the slicing operations (e.g. `a[0:10, 20:22]`) to plot the first 100 samples of x_r and x_i as a function of time. Try to plot each subband separately (e.g. using `plt.subplot(5, 2, isub)`.) 5. What do the timeseries look like? What did you expect?

[6]:
```
tmin = 300
tmax = 310
imin = int(tmin/tsamp)
imax = int(tmax/tsamp)
print(imin,imax)
```

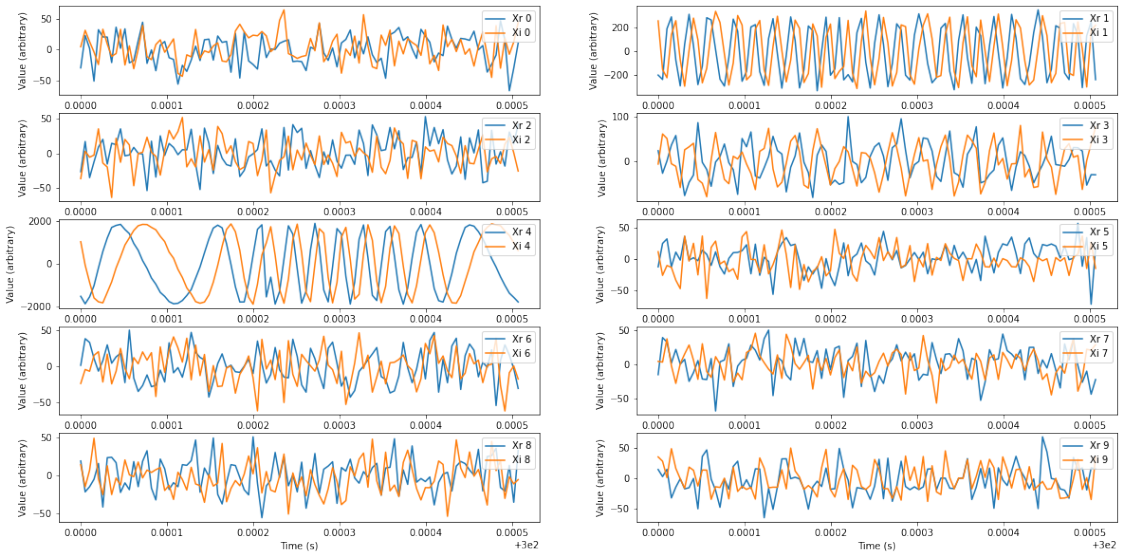
```
58593750 60546875
```

[7]:
```
xr = h5_s0["/SUB_ARRAY_POINTING_000/BEAM_000/STOKES_0"][imin:imax]
xi = h5_s1["/SUB_ARRAY_POINTING_000/BEAM_000/STOKES_1"][imin:imax]
print(xr.shape, xi.shape)
```

```
(1953125, 10) (1953125, 10)
```

[8]:
```
t = np.arange(imin, imax) * tsamp
```

[9]:
```
fig, axes = plt.subplots(5, 2, figsize=(20, 10))
for isub, ax in enumerate(axes.ravel()):
    ax.plot(t[:100], xr[:100, isub], label="Xr %d" % isub)
    ax.plot(t[:100], xi[:100, isub], label="Xi %d" % isub)
    ax.set_xlabel("Time (s)")
    ax.set_ylabel("Value (arbitrary)")
    ax.legend(loc="upper right")
plt.show()
```



1.3 Exercise 3

The next step is to Fourier transform the complex voltage data to generate dynamic spectra. This requires several steps, and we will go through them one by one. 1. We want the dynamic spectrum to have 1024 channels. Compute how many spectra (let's call them integrations) can be obtained from the input timeseries. Store this in a `nint` variable. (Make sure it is an integer and rounded down with e.g. `np.floor`). 2. Use the slice option to select `nchan * nint` values from the real and imaginary timeseries of X, and create a complex timeseries of the form `cin = Xreal + 1j * Ximag`. What is the shape of the complex timeseries? 3. Next, we will Fourier transform the complex timeseries. Assuming it is called `cin`, use `cout = np.fft.fftshift(np.fft.fft(cin.reshape(nint,`

`nchan, -1), axis=1), axes=1)` to perform the transform. This single command performs multiple steps: `cin.reshape(nint, nchan, -1)` reshapes the 2D array to a 3D array with shape $n_{\text{int}} \times n_{\text{chan}} \times n_{\text{sub}}$, next `np.fft.fft(..., axis=1)` performs the 1D Fourier transform along axis 1, which is n_{chan}. The resulting output has the two halves of the spectrum swapped, and `np.fft.fftshift..., axes=1)`, swaps the two halves back into the correct frequency order along the n_{chan} axis. What is the shape and type of the output values of `cout`? 4. `numpy` has a handy function to compute the frequencies of channels of a Fourier transform. Run `np.fft.fftfreq?` to see the usage of this function. Use this function, together with `np.fft.fftshift` to compute an array of the frequency values around the center frequency of each subband. 5. Plot the real and imaginary components of the first spectrum (`cout[0, :, isub]`) of each subband against the frequency values. Use `np.real` and `np.imag` to get the complex components.

[10]:
```
nchan = 1024
nint = int(np.floor(xr.shape[0] / nchan))
print(nint, nchan, nint * nchan, xr.shape[0])
```

```
1907 1024 1952768 1953125
```

[11]:
```
cin = xr[:nint*nchan] + 1j * xi[:nint * nchan]
cin.shape
```

[11]:
```
(1952768, 10)
```

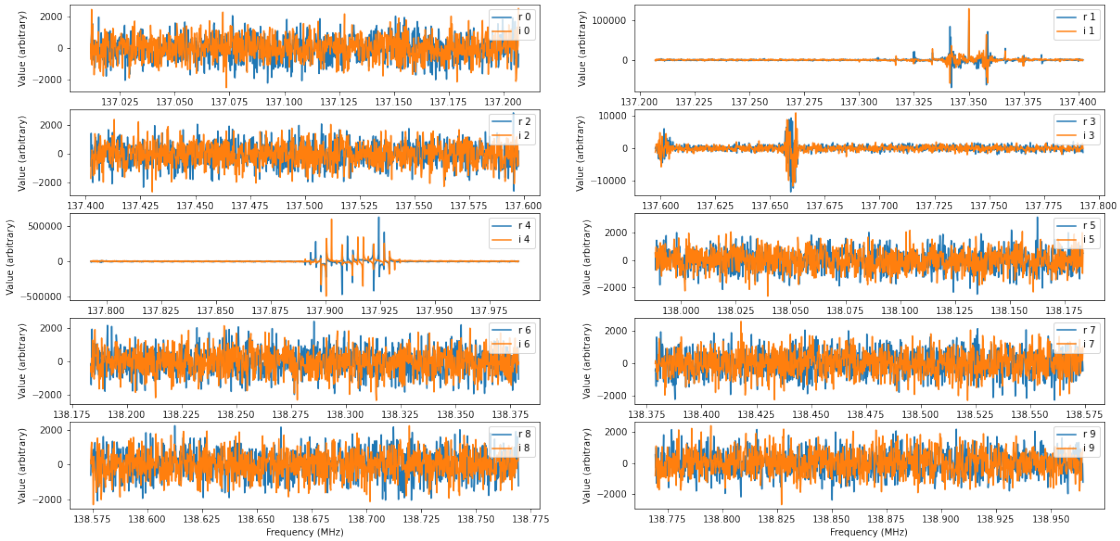
[12]:
```
cout = np.fft.fftshift(np.fft.fft(cin.reshape(nint, nchan, -1), axis=1), axes=1)
```

[13]:
```
print(cout.shape, type(cout[0, 0, 0]))
```

```
(1907, 1024, 10) <class 'numpy.complex128'>
```

[14]:
```
f = np.fft.fftshift(np.fft.fftfreq(nchan, d=tsamp))
```

[15]:
```
fig, axes = plt.subplots(5, 2, figsize=(20,10))
for isub, ax in enumerate(axes.ravel()):
    ax.plot((freq[isub] + f) * 1e-6, np.real(cout[0, :, isub]), label="r %d" %␣
 ↪isub)
    ax.plot((freq[isub] + f) * 1e-6, np.imag(cout[0, :, isub]), label="i %d" %␣
 ↪isub)
    ax.set_xlabel("Frequency (MHz)")
    ax.set_ylabel("Value (arbitrary)")
    ax.legend(loc="upper right")
```

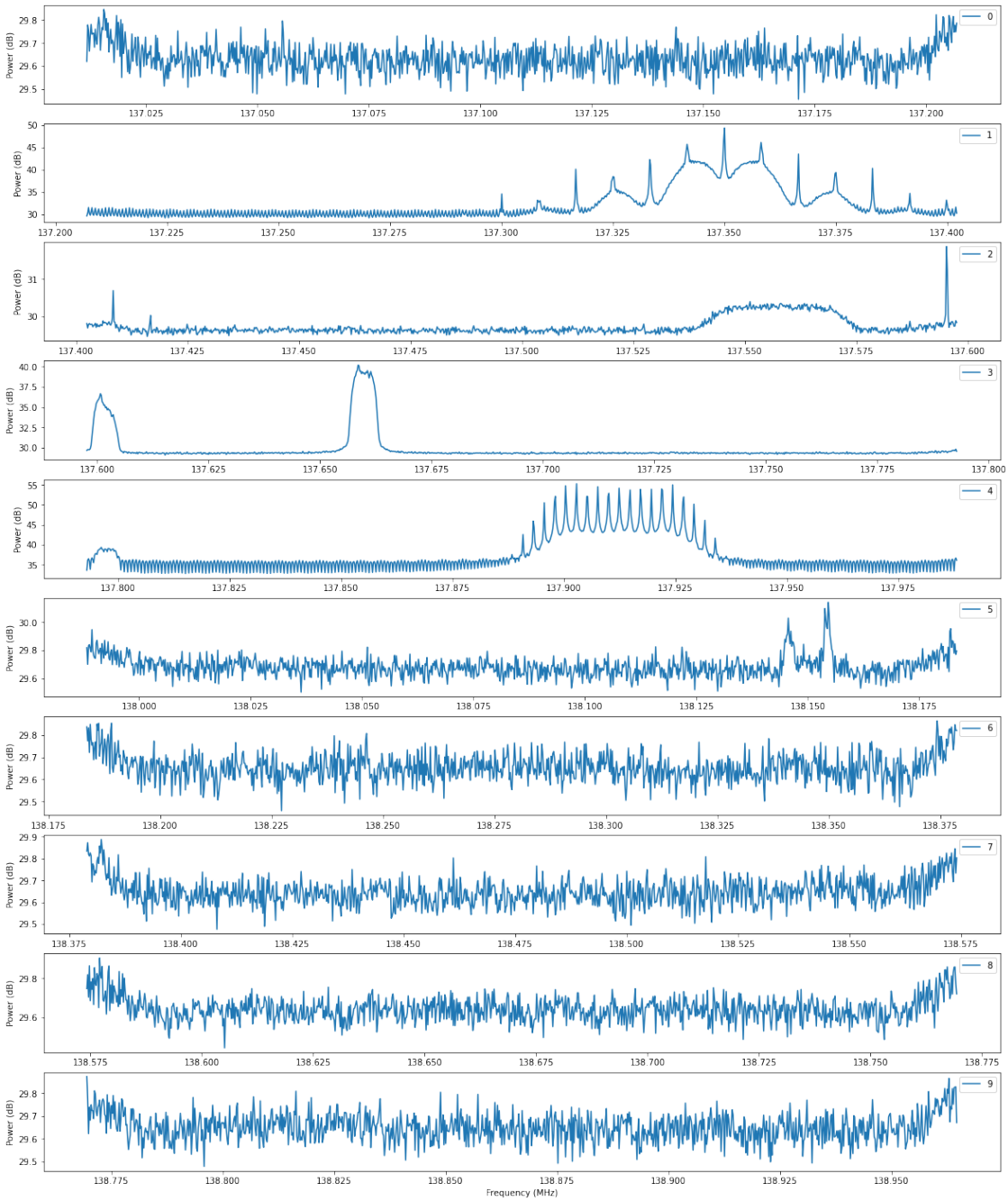


1.4 Exercise 4

You will recall from Michiel's lecture that the bandpass correction of COBALT is not perfect. Compute and plot the mean amplitude values (use `np.abs`) of each subband, averaged over time. Plot the amplitude values in decibels (dB: $x_{\mathrm{dB}} = 10 \log_{10}(x)$). What can you see from the averaged bandpasses?

[16]:
```
fig, axes = plt.subplots(10, 1, figsize=(20, 25))
for isub, ax in enumerate(axes.ravel()):
    ax.plot((freq[isub] + f) * 1e-6, 10 * np.log10(np.mean(np.abs(cout[:, :,␣
 ↪isub]), axis=0)), label="%d" % isub)
    ax.set_ylabel("Power (dB)")
    ax.legend()
ax.set_xlabel("Frequency (MHz)")
```

[16]:
```
Text(0.5, 0, 'Frequency (MHz)')
```



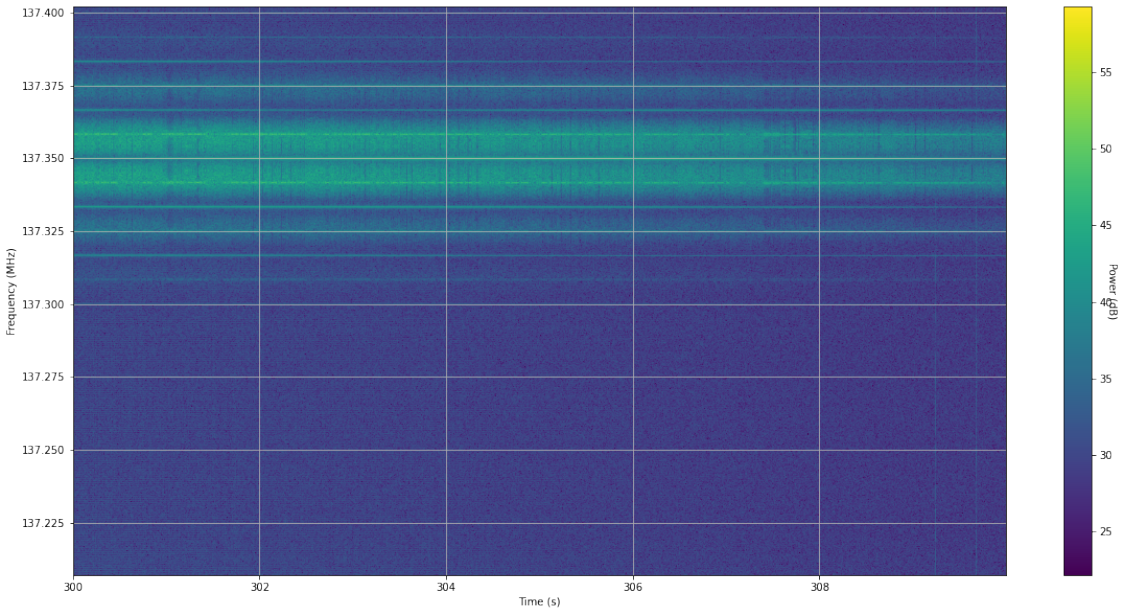
1.5 Exercise 5

Let's plot dynamic spectra. 1. Compute an array with times for each spectrum. 2. Pick a subband and compute amplitudes for this subband. Convert them to decibels. 3. Plot the dynamic spectrum (using `plt.imshow`) and make sure the axes are labelled correctly.

[17]:
```
t = tmin + np.arange(nint) * (tsamp * nchan)
```

[18]:
```
isub = 1
amp = np.absolute(cout[:, :, isub])
amp_db = 10.0 * np.log10(amp)
vmin = np.mean(amp_db) - 2.0 * np.std(amp_db)
vmax = np.mean(amp_db) + 6.0 * np.std(amp_db)
```

[19]:
```
fig, ax = plt.subplots(figsize=(20, 10))
img = ax.imshow(amp_db.T, origin="lower",
                aspect="auto", extent=[t[0], t[-1], (freq[isub]+f[0]) * 1e-6,␣
 ↪(freq[isub] + f[-1]) * 1e-6],
                vmin=vmin, vmax=vmax)
ax.grid()
ax.set_xlabel("Time (s)")
ax.set_ylabel("Frequency (MHz)")
cbar = fig.colorbar(img, ax=ax)
cbar.set_label("Power (dB)", rotation=270)
```



[ ]: