Station Data Products

An overall description of the output products of an international LOFAR station, and pointing out the small details that are left out of the cookbook.

Introduction

Most of the information in this chapter can be found within the LOFAR Station Cookbook, though there are a few other handy additions regarding the output formats that can be of interest. We will also discuss the current methogology for handling the data products and converting them to other formats, if needed.

XST Data

Array correlation/crosslet statistics statistics (XSTs) describe the autocorrelation between different antenna for a single subband at a given point in time (visibilities, covariances). A single XST file contains a single subband's data, which differs from a ACC file which loops over a range of subbands. They can be used to generate all-sky maps, a representation of the sky birghtness distribution of the station at a given frequency.

Generating XSTs

XSTs are generated using the `rspctl` command while the station is in `swlevel 2` , a known mode is set by `rspctl --mode=N` or `rspctl --band=N_M` and the station is expected to be in bitmode 16 as no other observations can be performed simultaneously. It will generate an output intergrated over the last N_{sec} to a file in a given *local_folder* . The overall syntax to generate an output for a given *subband* is

```
user1@lcu$ rspctl --xcsubband=subband

user1@lcu$ rspctl --xcstatistics --duration=n_observation_seconds --integration=n_sec --directory=local_folder
```

It is recommended to run this in combination with a script that will move the file to a new location after $N_{observation\_sec}$ as the file contains no metadata of the mode or subband used for the observation. Moving to a specified *modeN/sbM/* subband is highly recommend as a result. A sample observing script, assuming the station is in `swlevel 2` can be found here.

A note on HBA all-sky observations

Due to the rigid tile structure of the HBAs aligning the side lobes, performing all-sky observations with the entire HBA array is a hopeless endeavour.

However, at the GLOW stations, James Anderson worked around this by activating a single antenna in each HBA tile in a pesudo-random fashion to minimise sidelobe collision. Further work on the methodology by Menno Norden and Michiel Brentjens optimized this method and resulted in the script found here, slightly modified for use on the Irish station, which will automate the process for you. It should be run before any attempts to perform all-sky observations with the HBA tiles.

XST Data Format

XSTs are antenna-majour files that are written to disk every integration period. They do not come with any metadata outside of the starting time, which is present in the file name.

Each sample is 4 x $N_{antenna}$ x $N_{antenna}$ long (by default, an international station has 96 antenna). Each antenna generates a real and imaginary sample for it's correlation with every other antenna (Complex128 type, 2 Float64s, for each polarization). As a result, at the (n,n) indictes we generate the autocorrelation of the antenna with no lag. The overall format can be thought of as a 3D cube, with (n,n) squares stacked for *m* time samples.

(N0,N0)	(N0,N1)	...	(N0,Nn)
(N1,N0)	(N1,N1)		
...		...	
(Nn,N0)			(Nn,Nn)

All-Sky Plotting

Some sample mode 3, subband 210 data from IE613 can be found here. it is highly recommended to use an existing implementation for generating all-sky images, I provide the example David worked on here, but there is also Griffin Foster's SWHT, a module inside Tobia Carozzi's iLiSA and some scripts floating around the LOFAR community.

The method for generating all-sky maps is heavily involed and as a result we will only outline the steps involved here.

- Acquire XST data, antenna locations and (optionally) calibrations files for your station
- Load your data and metadata into an enirоment, samples for XST and calibrtion data,

```
xstData = np.fromfile(dataRef, dtype = np.complex128)
reshapeSize = xstData.size / (192 ** 2)
xstData = xstData.reshape(192, 192, reshapeSize, order = 'F')


calData = np.fromfile(calRef, dtype = np.complex128)
rcuCount = calVar.size / 512
calData = calData.reshape(rcuCount, 512, order = 'F')
calData = calData[:, subband_of_interest]


calXData = calData[::2]
calYData = calData[1::2]
```

```
calData = np.dstack([calXData, calYData])
```

- Apply your calibrations if needed to each of the X/Y polarizations

```
calSubbandX = calData[:, subband, 0]
calSubbandY = calData[:, subband, 1]


calMatrixXArr = np.outer(calSubbandX, np.conj(calSubbandX).T)[..., np.newaxis]
inputCorrelations[::2, ::2, ...] = np.multiply(np.conj(calMatrixXArr), inputCorrelations[::2, ::2, ...])


calMatrixYArr = np.outer(calSubbandY, np.conj(calSubbandY).T)[..., np.newaxis]
inputCorrelations[1::2, 1::2, ...] = np.multiply(np.conj(calMatrixYArr), inputCorrelations[1::2, 1::2, ...])
```

- Generate a range of pointings from sin(-pi/2) -> sin(+pi/2) across a grid, lower values will limit your field of view ((l,m) grid size)
- Calculate your wavelength and corresponding wavenumber for your subband (= (2 * np.pi) /wavelength)
- Form a weight matrix (and it's conjugate) for your pointing using the x-coordinates and y-coordinates of your antennae

```
wX = np.exp(-1j * k * posX * lVec) # (l, 96)
wY = np.exp(-1j * k * posY * mVec) # (m, 96)
weight = np.multiply(wX[:, np.newaxis, :], wY[:, :, np.newaxis]).transpose((1,2,0))[..., np.newaxis] # (l,m,96,1)
conjWeight = np.conj(weight).transpose((0,1,3,2)) # (l,m,1,96)
```

- For each sample of correlations, find the dot product of the correations with the complex conjugate of your weight matrix, and then the weight matrix

```
for frame in np.arange(frameCount):
    correlationMatrixChan = correlationMatrix[..., frame] # (96,96)
    tempProd = np.dot(conjWeight, correlationMatrixChan) # w.H * corr # (l,m, 1, 96)
    prodProd = np.multiply(tempProd.transpose((0,1,3,2)), weight)
    outputFrame[..., frame] = np.sum(prodProd, axis = (2,3))
```

Your output image will need to be rotated by the same mount as your station is rotated. You can find that information within the new iHBADeltas.conf files.

BST Data

Beamlet statictics (BSTs) are effectively dyanmic spectra produced by the station at a regular, relatively low cadence, interval.

Generating BSTs

BSTs are generated using the `rspctl` command in conjunction with a running `beamctl` command (as a result, the station must be in at least `swlevel 3`). It will write an output summary of the last N_{sec} (int) of antenna correlations to disk in a given *folder_location*. The overall synctax to call a `rspctl` command is

```
user1@lcu$ rspctl --statistics=beamlet --duration=n_observation_sec --integration=n_sec --
directory=folder_location/
```

The `rspctl` command will generate data for $N_{observation\_sec}$ or until the process is killed. As a result, the process must be kept active in a screen or by trailing the execution with an ampersand to send it to the background.

Enabling BST generation with the `rspctl` command will disable the CEP packet data stream, which can be re-enabled afterwards by calling `$ rspctl --datastream=1` . You can verify this worked by calling `$ rspctl --datastream` to get the current status of the datastream.

BST Data Format

BSTs are frequency-majour files that are written to disk every integration period. They do not come with any metadata outside of the stating time, output ring (only 0 available to international stations) and antenna polarization which are visible within the file name.

Each beamlet controlled by a `beamctl` command will generate a single output sample per intergration. So the output array dimensions will depend on your observation, and may be up to 244 in 16-bit mode, 488 in 8-bit mode and 976 in 4-bit mode.

The output data is a float, 4 times the size of your input.

Bitmode	Output (float)
4	float32 (verify)

8	float64
16	float128

BST Plotting

A fast way to plot BSTs in Python can be achieved with the *numpy* and *matplotlib* libraries. If you want to test this out for yourself, there are a large number of BSTs available on the data.lofar.ie archive which contain 488 subband observations from our Solar monitoring campaign.

```
import numpy as np
import matplotlib.pyplot as plt


bstLocation = "/path/to/bst.dat"
#bstDtype = np.float32 # 4-bit obs
#bstDtype = np.float64 # 8-bit obs
#bstDtype = np.float128 # 16-bit obs
rawData = np.fromfile(bstLocation, dtype = bstDtype)

#numBeamlets = 976 # 4-bit obs
#numBeamlets = 488 # 8-bit obs
#numBeamlets = 244 # 16-bit obs
rawData = rawData.reshape(-1, numBeamlets)



plt.figure(figsize=(24,12))
plt.imshow(np.log10(rawData.T), aspect = "auto")
plt.xlabel("Time Samples"); plt.ylabel("Subband"); plt.title("BST Data ({})".format(bstLocation))
plt.show()
```

ACC Data

Array correlation/crosslet statistics statistics (ACCs) describe the autocorrelation between different antenna for a number of subbands across a period of time (visibilities, covariances). A single ACC file typically contains one or more scans across the entire subband range of an observing mode, which differs from a XST file which only observed a single subband. They can be used to generate all-sky maps, a representation of the sky brightness distribution of the station at a given frequency, though typically ACCs are used for station debugging and calibration

Generating ACCs (Validation Required)

ACCs are generated using the `rspctl` command while the station is in `swlevel 2` , a known mode is set by `rspctl --mode=N` or `rspctl --band=N_M` and the station is expected to be in bitmode 16 as no other observations can be performed simultaneously. It will generate an output integrated over the last N_{sec} to a file in a given *local_folder* . The overall syntax to generate an output is

```
user1@lcu$ rspctl --xcstatistics --duration=n_observation_seconds --integration=n_sec --directory=local_folder
```

This differs from XSTs as no subband is specified. It is recommended to keep the observing mode in the *local_folder* name due to the lack of metadata associated with the observation.

You are also recommended to check in with your observer or station configuration to see the subbands the scan will be performed over. While by default, this covers every subband in a given mode before looking, your configuration may have changed and only use a limited subset of frequency channels.

A note on HBA all-sky observations

Due to the rigid tile structure of the HBAs aligning the side lobes, performing all-sky observations with the entire HBA array is a hopeless endeavour.

However, at the GLOW stations, James Anderson worked around this by activating a single antenna in each HBA tile in a pesudo-random fashion to minimise sidelobe collision. Further work on the methodology by Menno Norden and Michiel Brentjens optimized this method and resulted in the script found here, slightly modified for use on the Irish station, which will automate the process for you. It should be run before any attempts to perform all-sky observations with the HBA tiles.

ACC Data Format (Validation Required)

ACCs are antenna-major files that are written to disk every integration period for each subband. They do not come with any metadata outside of the starting time, which is present in the file name.

Each sample is 4 x $N_{subbands}$ x $N_{antenna}$ x $N_{antenna}$ long (by default, an international station has 96 antenna). Each antenna generates a real and imaginary sample for it's correlation with every other antenna (Complex128 type, 2 Float64s, for each polarization). As a result, at the (n,n) indictes we generate the autocorrelation of the antenna with no lag. The overall format can be thought of as a 3D cube, with (n,n) squares stacked for *m* time samples.

(N0,N0)	(N0,N1)	...	(N0,Nn)
(N1,N0)	(N1,N1)		
...		...	
(Nn,N0)			(Nn,Nn)

Each sample above is the index of 2 Complex128 values, for each polarizations of an antenna pair, giving an overall dimension of (192, 192) Complex128 elements for a standard international station.

See the "XST Data" page for an explanation on using these data products for all-sky maps.

CEP Packet Stream

The CEP packet stream is the lowest level, constantly produced data product from the station. It is a series of UDP packets that provide information on the digitized voltages for a given antenna while a beamformed observaiton is performed. There are normally 4 ports of data produced, though this may be lower if you are not fully utilizing the beamlets available to your station.

The CEP packet stream is highly variable, based on both your observing setup and your RSP configuration (`/var/lofar/etc/RSPDriver.conf`), we will describe this format as generic as ossible and provide default values for an international station.

Generating the CEP Packet Stream

The CEP packet stream should be generated whenever a beamformed observation is started via `beamctl`, though your station ocnfiguration or the use of `rspctl` to create BST data may interrupt or stop the data stream. You can verify the stream is enabled through the `rspctl --datastream` command, and re-enable it by calling `rspctl --datastream=1`.

While enabled, the CEP packet stream will send data to the MAC addresses configured in your `RSPDRiver.conf` every N time samples (by default, 16, 81.92µs) on all ports; though some ports may not contain data if the beamlets are not allocated. The subbands are distributed as described below (values are always inclusive).

Port Offset	4-bit Data	8-bit Data	16- bit Data
0	0:243	0:121	0:60
1	244:487	122:243	61:121
2	488:731	244:365	122:182
3	732:975	366:487	183:243

As the packets are UDP packets, the data may arrive at your machine out of order or not arrive at all. Packet loss to on-site machines is normally relatively low and in-order, so performing operations on the raw recorded data should be fine for more cases.

CEP Packet Data Format

The full CEP packet is described on page 32 of the Station Cookbook, but for now we will focus in on the CEP header, without any of the UDP information, and the raw data

There are 16 bytes of relelvant information at the start of every CEP packet. While it doesn't provide as much metatdata as we wouldl ike, it gives a good starting point for validation the structure of a packet and some of the base observing parameters.

Parameter	Byte(:bit)	Usage
RSP Version	0	Validation (~=3)
Source Info	1-2	Observing Configuration
RSP ID	1:0-4 (5-bit)	Output RSP ID
Unused	1:5 (1-bit)	Validate 0
RSP Error	1:6 (1-bit)	RSP Error, validate 0
Clock Bit	1:7 (1-bit)	1 if 200MHz clock, 0 if 160MHz
Bit mode	2:0-1 (2-bit)	0: 16-bit, 1: 8-bit: 2: 4-bit, 3: ERR
Unused	2:2-7 (6-bit)	Vaidate 0
Config	3	
Station ID	4-5	Station Code (see below)
Number of beamets	6	Beamlets in the current packet
Number of time samples	7	Time samples in the current packet (normally 16)
Timestamp	8-11	Unix timestamp of observation
Sequence	12-15	Leading time sample sequence

The actual packet size can vary based on the observing methodology. You can predict the size of each packet, and the raw data deminsions as a result, from the number of beamlets and number of time samples (bytes 6 and 7).

The reported statino code does not correspond to the public station names, I.e. IE613, SE607, etc. The RSPs report the internal station code, multiplied by 32, with an offset depending on the output port. The full list of station codes can be found here. As an example, if we were to analyse the short value produced from IE613, we would expect to see (214 * 32) + (0,1,2,3) depending on the CEP

port analysed.

The data is then in time-majour order, where each time sample contains the (Xreal, Ximag, Yreal, Yimag) samples. Size of each sample depends on your bitmode, varying from half a byte (4-bit) to 2 bytes (16-bit). This repeats for $N_{time\ sampels}$ (default is 16), before moving on to the next beamlet.

Recording / Handling Methodology

This is discussed more in-dept within the REALTA user's guide where we describe the methodology used at IE613. The overall view is that the data should be

- Recorded with some UDP packet capturing software -- wireshark, Olaf Wucknitz's volage recorder, etc.
- Data is read back either blindly or checking for missing packets, out of sequence data, headers analyzed, etc
- Voltages can be used to form Stokes vectors to the output data product required

TBB Data (Empty)

SST Data (Empty)