Wide-Band Calibration

January 10, 2019

by Aaron Parsons

The purpose of this memo is to explore the coarse absolute calibration of wideband (nominally 50--250-MHz) data from initial H2C observations with 3 functional antennas. We do this by using visibility data for the 3 (shortest) baselines and comparing them in the 100--200-MHz range to LST-binned, absolutely calibrated H1C-IDR2.1 data for the equivalent baseline spacing.

Because the Vivaldi-type feed used in H2C observations has a different beam response pattern than the sleeved-dipole feed used in H1C, it is not expected that these observations should match perfectly for the same baselines at the same LST. However, the beams are similar enough that a rough calibration is possible. We solve for the electronic delay and overall amplitude of H2C relative to H1C and extend these parameters to the entire 50--250-MHz band.

We conclude that this approach could be used once more antennas are available in H2C observations to roughly solve for degenerate parameters after redundant calibration. The approach is likely to only be coarsely correct, but it may provide a reasonable starting point for developing an absolute-calibration model for wideband data that can then be improved through imaging techniques.

Finally, we see clear evidence in each of the baselines studied of spurious power at delays of $\sim \pm 2800$ ns. These are likely reflections traversing the new fiber-optic cables employed in H2C. Their amplitudes appear to be between -20 and -40 dB in temperature-squared units. The timescales of these reflections, assuming transmission at $0.7c$, suggests a cable length of approximately 300 m (traversed twice).

```
In [1]: %matplotlib inline
        import pylab as plt, numpy as np
        import uvtools, pyuvdata, glob, aipy, hera_qm
        from scipy.interpolate import RectBivariateSpline
```

1 Acquire H2C Wide-Band Data

```
In [25]: files = glob.glob('*.HH.uvh5')
         data, times = {}, {}
         print 'Reading', files[0], 'to', files[-1]
         for f in files:
             uvf = pyuvdata.UVData()
             uvf.read_uvh5(f)
             keys = uvf.get_antpairpols()
             for k in keys:
                 data[k] = data.get(k,[]) + [uvf.get_data(k)]
```

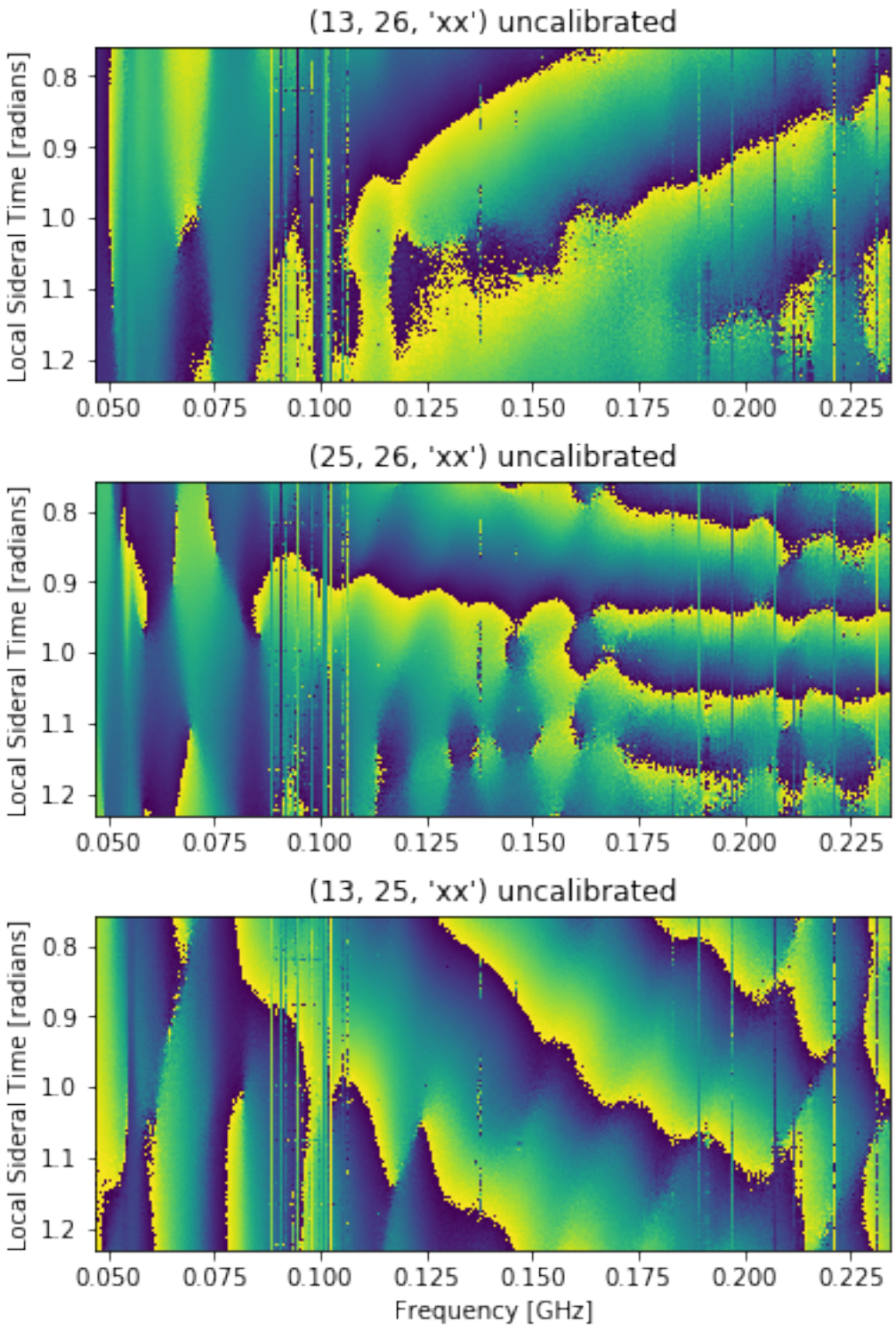
```
            lsts = uvf.lst_array[np.append(*uvf._key2inds(k)[:2])]
            times[k] = times.get(k,[]) + [lsts]
    data = {k: np.concatenate(dk, axis=0) for k,dk in data.items()}
    times = {k: np.concatenate(tk, axis=0) for k,tk in times.items()}
    freqs = uvf.freq_array
```

```
Reading zen.2458471.31840.HH.uvh5 to zen.2458471.39148.HH.uvh5
```

```
In [5]: for k, d in data.items():
            if not np.all(d == 0) and k[-1] == 'xx' and k[0] != k[1]:
                print k, d.shape
```

```
(13, 14, 'xx') (756, 1536)
(13, 26, 'xx') (756, 1536)
(25, 14, 'xx') (756, 1536)
(25, 26, 'xx') (756, 1536)
(26, 14, 'xx') (756, 1536)
(13, 25, 'xx') (756, 1536)
```

```
In [6]: POL = 'xx'
        ks = [(13,26,POL), (25,26,POL), (13,25,POL)]
        plt.figure(figsize=(6,8))
        for cnt,k in enumerate(ks):
            plt.subplot(3, 1, cnt+1)
            plt.title(str(k) + ' uncalibrated')
            uvtools.plot.waterfall(data[k], mode='phs',
                extent=(freqs[0,0]/1e9,freqs[0,-1]/1e9,times[k][-1],times[k][0]))
            plt.ylabel('Local Sideral Time [radians]')
        plt.subplots_adjust(top=0.95, bottom=0.07, hspace=.3)
        plt.xlabel('Frequency [GHz]')
        plt.show()
```


(13, 26, 'xx') uncalibrated
Local Sideral Time [radians]
0.8
0.9
1.0
1.1
1.2
0.050
0.075
0.100
0.125
0.150
0.175
0.200
0.225
(25, 26, 'xx') uncalibrated
Local Sideral Time [radians]
0.8
0.9
1.0
1.1
1.2
0.050
0.075
0.100
0.125
0.150
0.175
0.200
0.225
(13, 25, 'xx') uncalibrated
Local Sideral Time [radians]
0.8
0.9
1.0
1.1
1.2
0.050
0.075
0.100
0.125
0.150
0.175
0.200
0.225
Frequency [GHz]

2 Acquire H1C IDR2.1 Data

```
In [27]: files = glob.glob('../../hera_idr2.1/zen.grp1.of1.xx.LST.0.[789]*.uvOCRSL')
         files += glob.glob('../../hera_idr2.1/zen.grp1.of1.xx.LST.1.[01]*.uvOCRSL')
         ks_h1c = [(12,25,'xx'), (12,13,'xx'), (13,25,'xx')] # matches order of ks above
```

```
In [28]: data_h1c = {}
         times_h1c = {}
         for f in files:
             print 'Reading', f
             uvf = pyuvdata.UVData()
             uvf.read_miriad(f)
             for k in ks_h1c:
                 if k[-1] != 'xx': continue
                 data_h1c[k] = data_h1c.get(k,[]) + [uvf.get_data(k)]
                 lsts = uvf.lst_array[np.append(*uvf._key2inds(k)[:2])]
                 times_h1c[k] = times_h1c.get(k,[]) + [lsts]
         data_h1c = {k: np.concatenate(dk, axis=0) for k,dk in data_h1c.items()}
         times_h1c = {k: np.concatenate(tk, axis=0) for k,tk in times_h1c.items()}
         freqs_h1c = uvf.freq_array
```

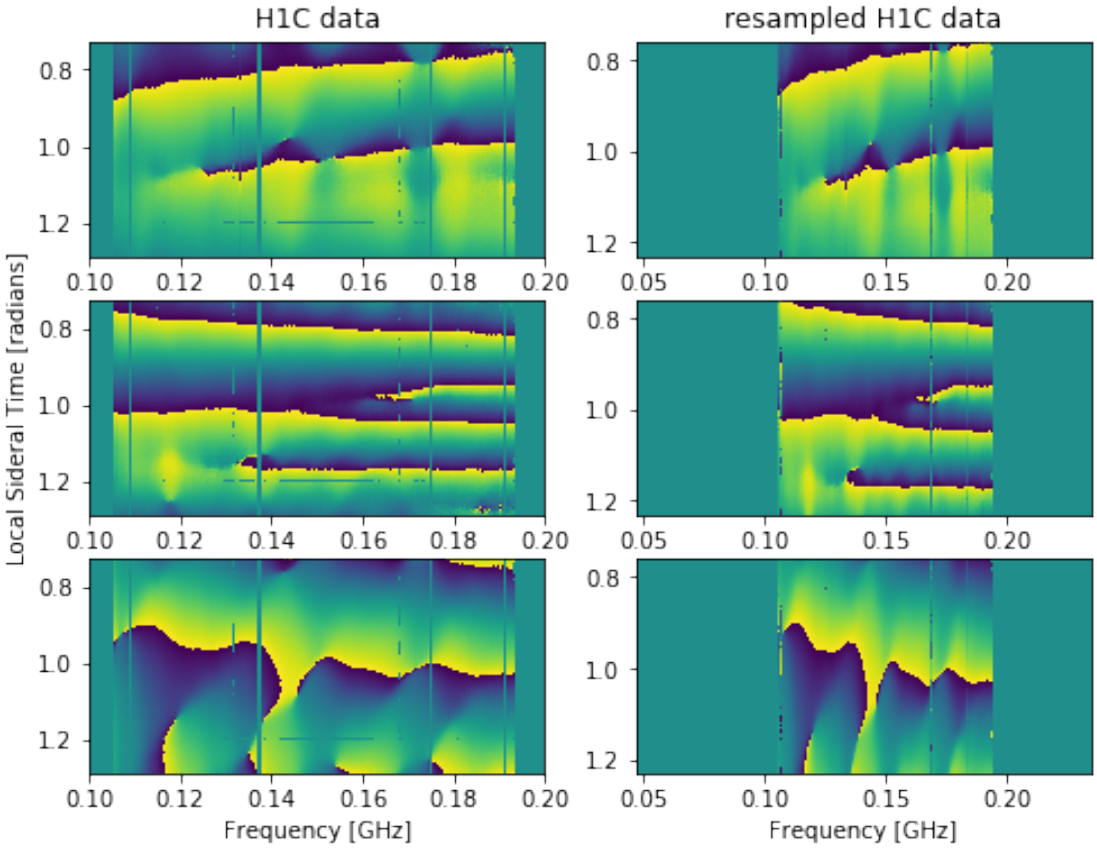
```
Reading ../../hera_idr2.1/zen.grp1.of1.xx.LST.0.72453.uvOCRSL
Reading ../../hera_idr2.1/zen.grp1.of1.xx.LST.0.81849.uvOCRSL
Reading ../../hera_idr2.1/zen.grp1.of1.xx.LST.0.91245.uvOCRSL
Reading ../../hera_idr2.1/zen.grp1.of1.xx.LST.1.00641.uvOCRSL
Reading ../../hera_idr2.1/zen.grp1.of1.xx.LST.1.10036.uvOCRSL
Reading ../../hera_idr2.1/zen.grp1.of1.xx.LST.1.19432.uvOCRSL
```

2.1 Resample H1C IDR2.1 Data onto H2C Observations

```
In [10]: data_h1c_interp = {}
         for k,kh1c in zip(ks,ks_h1c):
             t,f,d = times_h1c[kh1c], freqs_h1c[0], data_h1c[kh1c]
             intr = RectBivariateSpline(t, f, d.real)
             inti = RectBivariateSpline(t, f, d.imag)
             data_h1c_interp[k] = intr(times[k],freqs[0]) + 1j*inti(times[k],freqs[0])
```

```
In [14]: plt.figure(figsize=(8,6))
         for cnt,(k,kh1c) in enumerate(zip(ks, ks_h1c)):
             plt.subplot(3, 2, 2*cnt+1)
             uvtools.plot.waterfall(data_h1c[kh1c], mode='phs',
                 extent=(freqs_h1c[0,0]/1e9,freqs_h1c[0,-1]/1e9,
                         times_h1c[kh1c][-1],times_h1c[kh1c][0]))
             plt.subplot(3, 2, 2*cnt+2)
             uvtools.plot.waterfall(data_h1c_interp[k], mode='phs',
                 extent=(freqs[0,0]/1e9,freqs[0,-1]/1e9,
                         times[k][-1],times[k][0]))
         plt.subplot(3, 2, 1); plt.title('H1C data')
```

```
plt.subplot(3, 2, 2); plt.title('resampled H1C data')
plt.subplot(3, 2, 3); plt.ylabel('Local Sideral Time [radians]')
plt.subplot(3, 2, 5); plt.xlabel('Frequency [GHz]')
plt.subplot(3, 2, 6); plt.xlabel('Frequency [GHz]')
plt.show()
```



3 Calibration of H2C Data to H1C Model

In this section, we do a per-baseline (for 3 baselines) calibration using a model with a single phase parameter (electronic delay) and a single amplitude parameter (overall gain):

$$V_{\text{cal}} = V_{\text{uncal}} A e^{2\pi i \nu \tau} \tag{1}$$

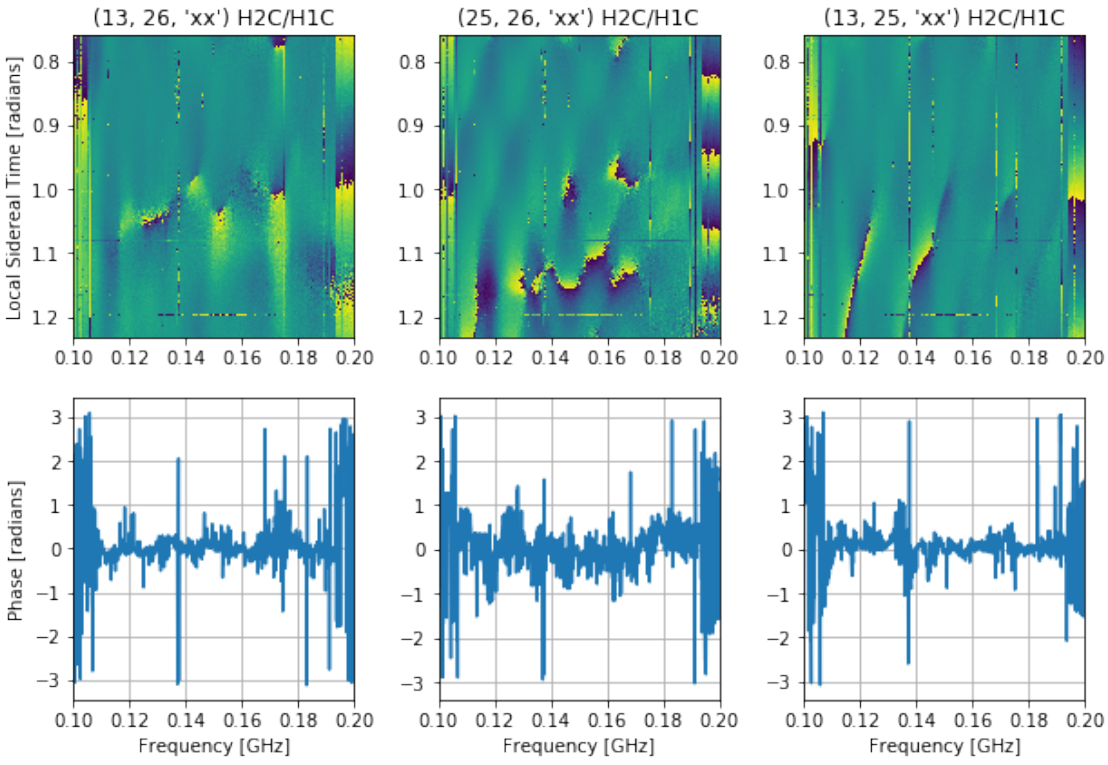
The solutions below were manually obtained "by eye".

In [15]:
```
data_cal = {}
data_cal[13,26,POL] = data[13,26,POL] * 0.6e-3 * np.exp(2j * np.pi* 6.59e-9 * freqs)
data_cal[25,26,POL] = data[25,26,POL] * 0.6e-3 * np.exp(2j * np.pi* -5.15e-9 * freqs)
data_cal[13,25,POL] = data[13,25,POL] * 0.7e-3 * np.exp(2j * np.pi* 11.75e-9 * freqs)
```

3.1 Verify Phase calibration

Below, we show that the calibration parameters used above do a reasonable job of flattening the phase difference between H2C and H1C data. The waterfall plots below show that there remain phase differences between H1C and H2C data. These are likely the result of beam differences between the Vivaldi-type and sleeved-dipole-type antenna feeds. Nonetheless, averaged over time (bottom plot), the overall phase difference is close to zero.

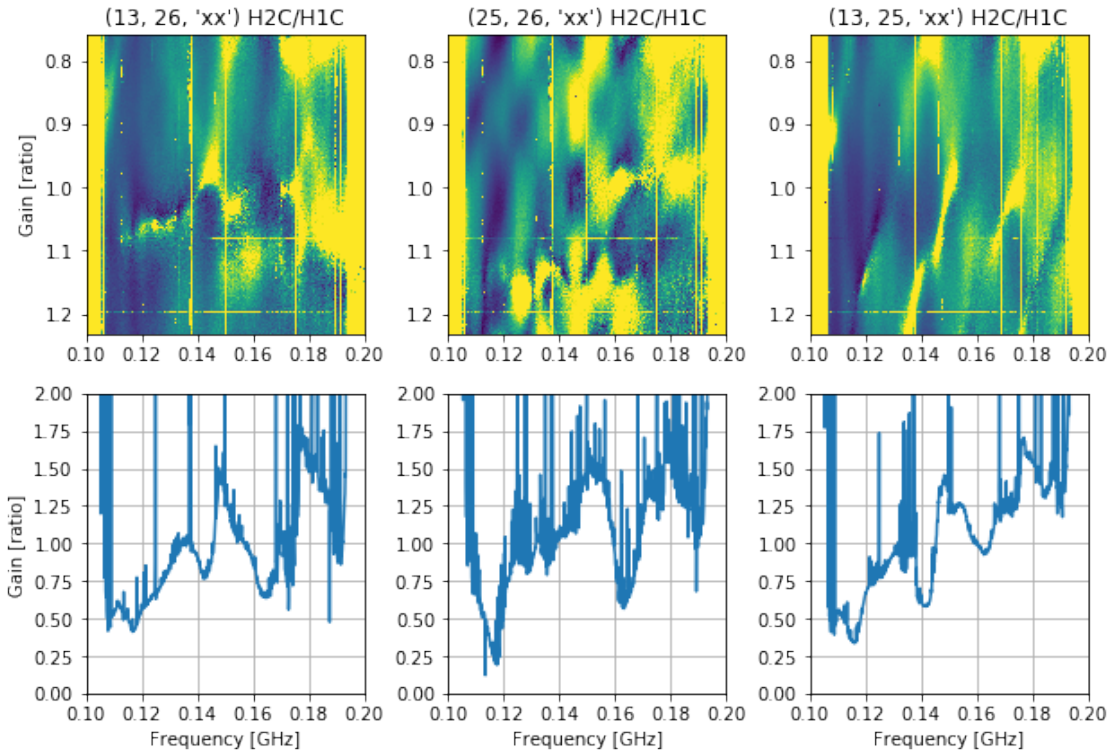
```
In [17]: plt.figure(figsize=(10,6))
         for cnt, k in enumerate(ks):
             ratio = data_cal[k] * data_h1c_interp[k].conj() / np.abs(data_h1c_interp[k])**2
             plt.subplot(2, 3, cnt + 1)
             uvtools.plot.waterfall(ratio, mode='phs',
                 extent=(freqs[0,0]/1e9,freqs[0,-1]/1e9,
                         times[k][-1],times[k][0]))
             plt.xlim(freqs_h1c[0,0]/1e9, freqs_h1c[0,-1]/1e9)
             plt.title(str(k) + ' H2C/H1C')
             plt.subplot(2, 3, cnt + 4)
             plt.plot(freqs[0]/1e9, np.angle(np.median(ratio, axis=0)))
             plt.xlim(freqs_h1c[0,0]/1e9, freqs_h1c[0,-1]/1e9)
             plt.xlabel('Frequency [GHz]')
             plt.grid()
         plt.subplots_adjust(top=0.95, bottom=0.1, hspace=.2, wspace=.3)
         plt.subplot(2, 3, 1); plt.ylabel('Local Sidereal Time [radians]')
         plt.subplot(2, 3, 4); plt.ylabel('Phase [radians]')
         plt.show()
```



3.2 Verify Amplitude Calibration

Here we do the same for the amplitude parameter, showing that the gain ratio of H2C data to H1C data between 100 and 200 MHz is unity, $\pm$ 50%. Obviously, we'd like to do better, but higher-order parameter fits to the bandpass between 100 and 200 MHz are unlikely to extrapolate well out of band.

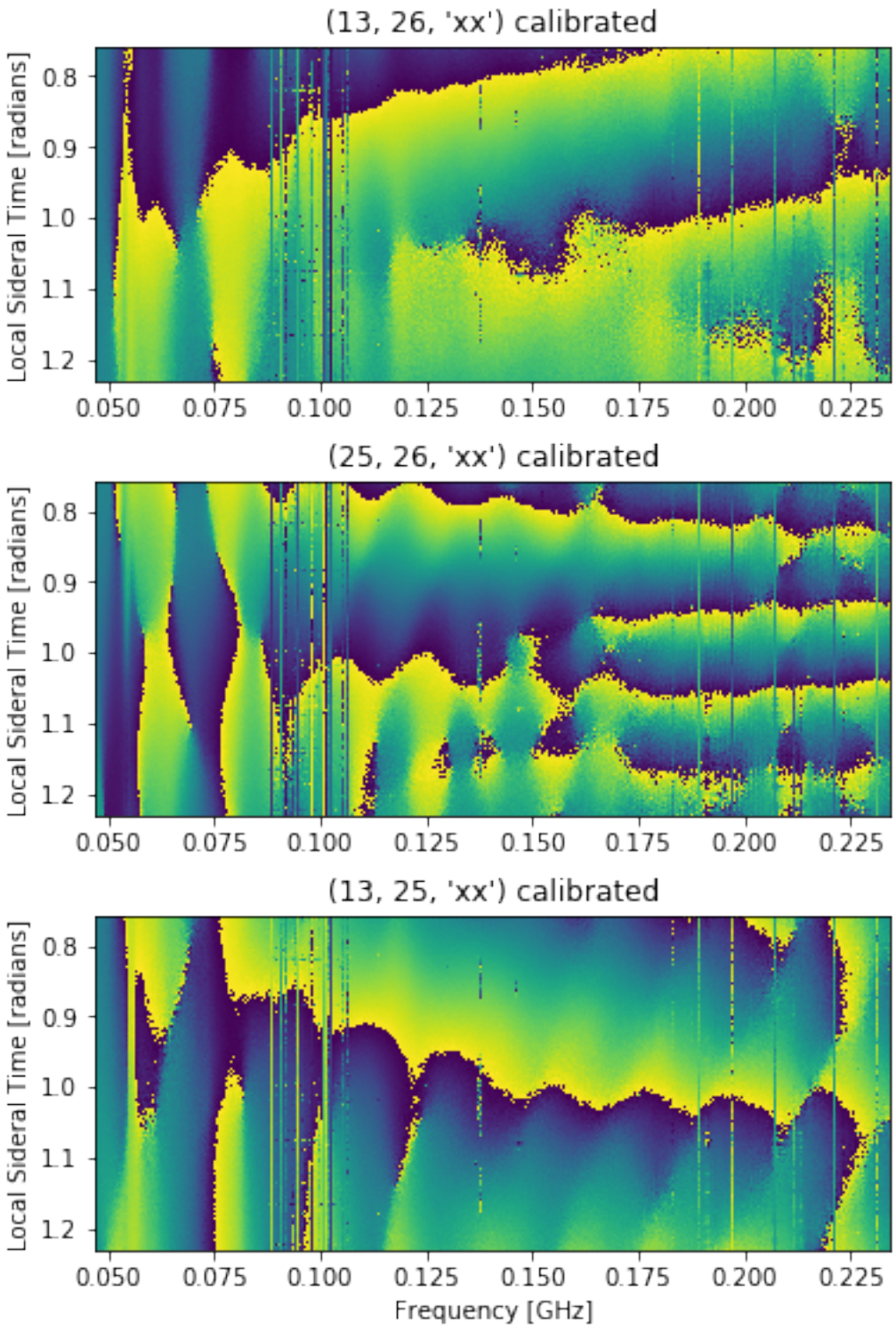
```
In [18]: plt.figure(figsize=(10,6))
         for cnt, k in enumerate(ks):
             ratio = data_cal[k] * data_h1c_interp[k].conj() / np.abs(data_h1c_interp[k])**2
             plt.subplot(2, 3, cnt + 1)
             uvtools.plot.waterfall(ratio, mode='lin', mx=2, drng=2,
                 extent=(freqs[0,0]/1e9,freqs[0,-1]/1e9,
                         times[k][-1],times[k][0]))
             plt.xlim(freqs_h1c[0,0]/1e9, freqs_h1c[0,-1]/1e9)
             plt.title(str(k) + ' H2C/H1C')
             plt.subplot(2, 3, cnt + 4)
             plt.plot(freqs[0]/1e9, np.abs(np.median(ratio, axis=0)))
             plt.xlim(freqs_h1c[0,0]/1e9, freqs_h1c[0,-1]/1e9)
             plt.ylim(0,2)
             plt.xlabel('Frequency [GHz]')
             plt.grid()
         plt.subplots_adjust(top=0.95, bottom=0.1, hspace=.2, wspace=.3)
         plt.subplot(2, 3, 1); plt.ylabel('Gain [ratio]')
         plt.subplot(2, 3, 4); plt.ylabel('Gain [ratio]')
         plt.show()
```



3.3 Wide-Band Calibrated Data?

With the caveat that the calibration we have obtained is quite coarse, we present the calibrated H2C data below.

```
In [19]: plt.figure(figsize=(6,8))
         for cnt,k in enumerate(ks):
             plt.subplot(3, 1, cnt+1)
             plt.title(str(k) + ' calibrated')
             uvtools.plot.waterfall(data_cal[k], mode='phs',
                 extent=(freqs[0,0]/1e9,freqs[0,-1]/1e9,times[k][-1],times[k][0]))
             plt.ylabel('Local Sideral Time [radians]')
         plt.subplots_adjust(top=0.95, bottom=0.07, hspace=.3)
         plt.xlabel('Frequency [GHz]')
         plt.show()
```


(13, 26, 'xx') calibrated
Local Sideral Time [radians]
0.8
0.9
1.0
1.1
1.2
0.050
0.075
0.100
0.125
0.150
0.175
0.200
0.225
(25, 26, 'xx') calibrated
Local Sideral Time [radians]
0.8
0.9
1.0
1.1
1.2
0.050
0.075
0.100
0.125
0.150
0.175
0.200
0.225
(13, 25, 'xx') calibrated
Local Sideral Time [radians]
0.8
0.9
1.0
1.1
1.2
0.050
0.075
0.100
0.125
0.150
0.175
0.200
0.225
Frequency [GHz]

3.4 RFI Flagging

In order to perform a delay transform, we need to first flag off radio-frequency interference (RFI). First, we manually flag the FM radio band from 87 to 108 MHz, then use the watershed flagging algorithm discussed in Kerrigan et al. (2019).

```
In [20]: wgts = {}
         wgt_fm = np.where(np.logical_and(freqs > 0.087e9, freqs < 0.108e9), 0, 1)
         for k in ks:
             flags = hera_qm.xrfi.xrfi(wgt_fm * data_cal[k])
             wgts[k] = ~flags * wgt_fm
```

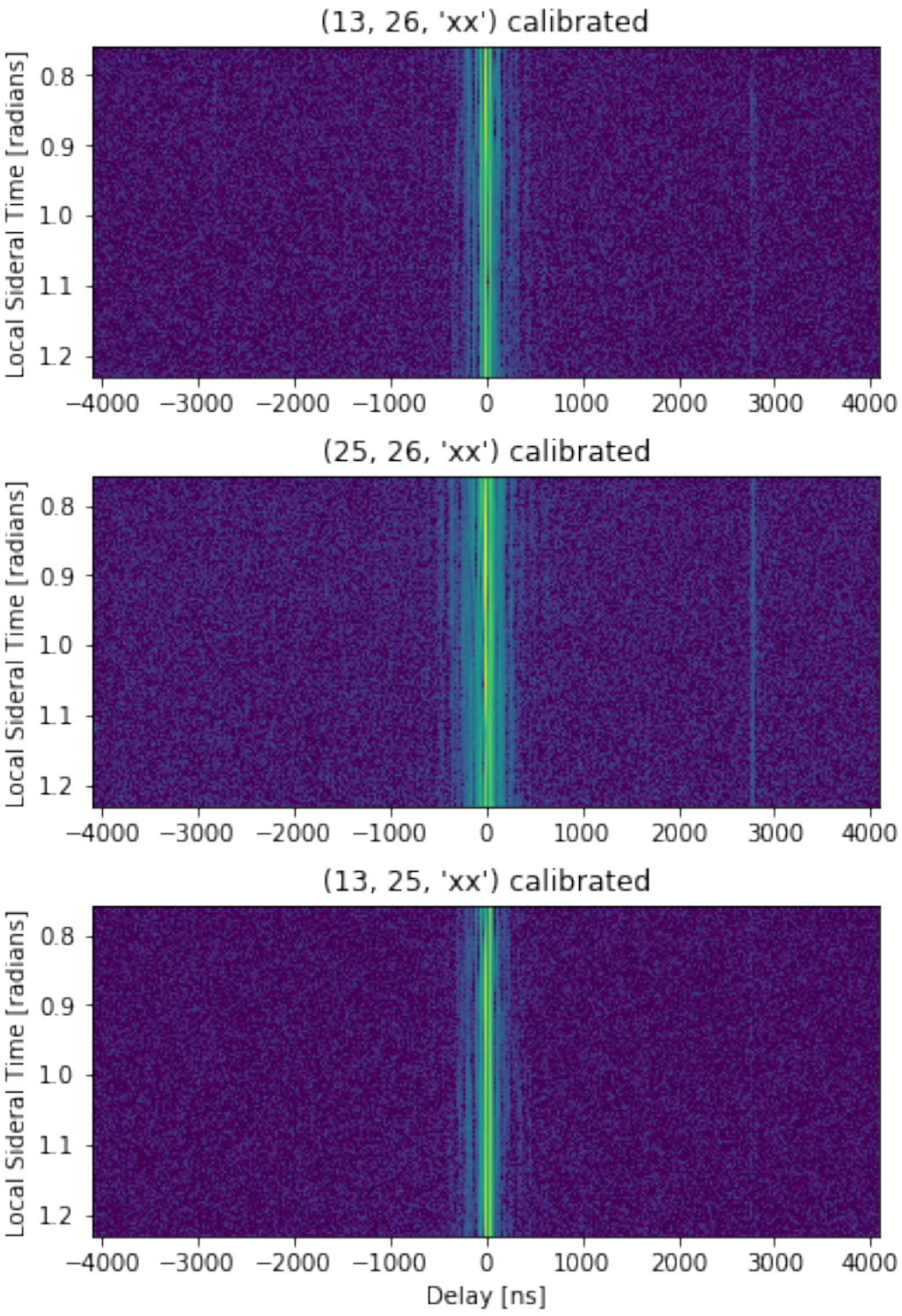
3.5 Delay Transform

Finally, we perform a Fourier transform over the frequency axis for H2C data, using the standard CLEAN deconvolution to compensate for flagged data (Parsons & Backer 2009)

```
In [21]: _data_cl, _data_rs = {}, {}
         for k in ks:
             print k
             df = wgts[k] * data_cal[k]
             _d = np.fft.ifft(df, axis=1)
             _w = np.fft.ifft(wgts[k], axis=1)
             _d_cl, _d_rs = np.zeros_like(_d), np.zeros_like(_d)
             for i in range(_d.shape[0]):
                 _d_cl[i], info = aipy.deconv.clean(_d[i], _w[i], tol=1e-5, stop_if_div=False)
                 _d_rs[i] = info['res']
             _data_cl[k] = _d_cl
             _data_rs[k] = _d_rs
```

```
(13, 26, 'xx')
(25, 26, 'xx')
(13, 25, 'xx')
```

```
In [22]: plt.figure(figsize=(6,8))
         taus = np.fft.fftfreq(freqs.size, freqs[0,1]-freqs[0,0])
         taus = np.fft.fftshift(taus)
         for cnt,k in enumerate(ks):
             _d = _data_cl[k] + _data_rs[k]
             plt.subplot(3, 1, cnt+1)
             plt.title(str(k) + ' calibrated')
             uvtools.plot.waterfall(np.fft.fftshift(_d, axes=1), mode='log', drng=3,
                 extent=(taus[0]*1e9,taus[-1]*1e9,times[k][-1],times[k][0]))
             plt.ylabel('Local Sideral Time [radians]')
         plt.subplots_adjust(top=0.95, bottom=0.07, hspace=.3)
         plt.xlabel('Delay [ns]')
         plt.show()
```



In order to make a reflection feature at ± 2800 ns more clear, we average $|\tilde{V}(\tau, t)|^2$ over time to obtain a power spectrum that is proportional to $P(k)$ in units of K^2.

```
In [24]: plt.figure()
```

```
for k in ks:
    _d = _data_cl[k] + _data_rs[k]
    plt.semilogy(taus * 1e9, np.fft.fftshift(np.mean(np.abs(_d)**2, axis=0)), label=k)
plt.ylabel('Power [K$^2$, arbitrary scale]')
plt.xlabel('Delay [ns]')
plt.grid()
plt.show()
```