Updated Power Spectrum Normalization

Adrian Liu, 1st February 2018

This memo is an update of HERA memo #27 ("Power Spectrum Normalizations for HERA", by A. Parsons).

As outlined in the previous memo, the data that comes in to the power spectrum pipeline has already been converted from the Jy units to temperature units, assuming a primary beam area of

$$\Omega_p(\nu) \equiv \int d\Omega A(\hat{r}, \nu)$$

Additionally, the delay transform is slightly different from that which is usually defined in theory papers. In papers, the delay transform gives the visibilities an extra unit of frequency, because it is an integral over frequency. In data analysis, the delay transform is accompanied by a division of

$$B_p \equiv \int d\nu$$

so that the delay transform does not introduce any extra units.

As stated in the previous memo, this means that the delay-transformed visibilities as defined in code, $\tilde{V}'_{\rm mK}$, are related to the delay-transformed visibilities as defined in Parsons et al. (2012; arXiv: 1304.4991), $\tilde{V}$ by

$$\tilde{V}'_{\rm mK} \equiv \frac{\lambda^2}{2k_B \Omega_p B_p} \tilde{V}.$$

The equivalent of Equation (B8) of Parsons et al. (2012) is then

$$\langle |\tilde{V}'_{\rm mK}|^2 \rangle = \hat{P} \int \frac{d\Omega d\nu}{X^2 Y} \frac{A^2(\hat{r},\nu) W^2(\nu)}{\Omega_p^2(\nu) B_p^2},$$

where one difference between this and the published equation is the inclusion of the spectral tapering function $W(\nu)$. From this point forward, we deviate from the previous memo. There, the frequency dependence on the cosmological conversion scalars (X and Y) and the primary beam were neglected; here, we take them into account. As shown in Liu et al. (2016; arXiv: 1609.04401), this is simply a matter of not factoring any of the frequency-dependent quantities out of the integral (i.e., there are no further complications due to the curved sky).

The "scalar" normalization is then given by

$$\text{Scalar} = \left[\int \frac{d\Omega d\nu}{X^2 Y} \frac{A^2(\hat{r},\nu) W^2(\nu)}{\Omega_p^2(\nu) B_p^2} \right]^{-1}.$$

This is what one multiplies $|\tilde{V}'_{\rm mK}|^2$ by, in order to get power spectra in correct "cosmological units".

In terms of practical implementation, we have to perform an integral of $A^2(\hat{r}, \nu)$ over frequency. Since our beam models are available only as discrete samplings in frequency, we must interpolate. One way to do this is to interpolate the beams pixel-by-pixel (or if one prefers, spherical harmonic mode by spherical harmonic mode). However, it is probably better to interpolate a single-variable function. For instance, if we write

$$\frac{1}{\text{Scalar}} = \int \frac{d\nu}{X^2 Y} \frac{W^2(\nu)}{\Omega_p^2(\nu) B_p^2} \int d\Omega A^2(\hat{r}, \nu),$$

we see that one can perform the spatial integral first, and then interpolate in frequency for the next integral. Note that in the past, the beam models have often been stored in terms of their spherical harmonic coefficients. If that's the case, the relevant spatial integral can be done without going back to image space by taking advantage of Parseval's theorem:

$\int d\Omega A^2(\hat{r},\nu) = \sum_{\ell m} |a_{\ell m}|^2 = \sum_{\ell} (2\ell+1) C_{\ell}$.

This is currently how things are implemented in code, although it may not be the most efficient depending on how we end up storing the primary beam models.

In [1]:
```
import numpy as np, pyuvdata as uv, healpy as hp, hera_pspec as ps, aipy
from sklearn.gaussian_process import GaussianProcessRegressor
from scipy import integrate
%pylab inline
```

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

In [2]:
```
beam_file_loc = '../data/NF_HERA_Beams.beamfits'
HERA_beam = uv.UVBeam()
HERA_beam.read_beamfits(beam_file_loc)
```

In [3]:
```
conversions = ps.conversions.Cosmo_Conversions()
```

In [4]:
```
npols, nfreqs, npix = HERA_beam.data_array.shape[-3:]
beam_freqs = HERA_beam.freq_array
```

In [5]:
```
HERA_power_beam = np.zeros((nfreqs,npix))
HERA_power_beam_Cl = []
HERA_power_beam_int = np.zeros(nfreqs)
for i in range(nfreqs):
    # Do I need a factor of 2 below?
    HERA_power_beam[i] = (HERA_beam.data_array[0,0,0,i] + HERA_beam.data
_array[0,0,1,i])
    HERA_power_beam_int[i] = np.sum(HERA_power_beam[i])
    HERA_power_beam_Cl.append(hp.anafast(HERA_power_beam[i]))
HERA_power_beam_int *= 4.*np.pi / npix
two_ell_plus_one = 2 * np.arange(192) + 1
sq_beam_integral = np.einsum('i,ji', two_ell_plus_one, HERA_power_beam_C
l)
```

In [6]:
```
pspec_nfreqs = 40000
pspec_freqs_MHz = np.linspace(145.,155.,pspec_nfreqs)
pspec_freqs_Hz = 10**6 * pspec_freqs_MHz
redshifts = conversions.f2z(pspec_freqs_Hz).flatten()
X2Y = np.array(map(conversions.X2Y,redshifts)) # (Mpc**3) / (Str Hz)
```

In [7]:
```
dBpp_over_BpSq = np.ones_like(pspec_freqs_Hz) #aipy.dsp.gen_window(pspec
_nfreqs,'blackman-harris')**2
dBpp_over_BpSq /= (pspec_freqs_Hz[-1] - pspec_freqs_Hz[0])**2
```

In [8]:
```
gp = GaussianProcessRegressor()
gp.fit(beam_freqs.T/(1.*10**6), sq_beam_integral/HERA_power_beam_int**2)
```

Out[8]:
```
GaussianProcessRegressor(alpha=1e-10, copy_X_train=True, kernel=None,
             n_restarts_optimizer=0, normalize_y=False,
             optimizer='fmin_l_bfgs_b', random_state=None)
```

In [9]:
```
dOpp_over_Op2 = gp.predict(np.atleast_2d(pspec_freqs_MHz).T)
```

In [10]:
```
d_inv_scalar = dBpp_over_BpSq * dOpp_over_Op2 /  X2Y
```

In [11]:
```
scalar = 1 / integrate.trapz(d_inv_scalar, pspec_freqs_Hz)
print scalar#* (pspec_freqs_Hz[-1] - pspec_freqs_Hz[0])
```

```
416563509.959
```