HERA Memo 89
Uploaded 22 Jan 2021

SimpleDS Shadow Pipeline Test 0.1

Matthew Kolopanis

October 28, 2020

- Initial Memo: May 29, 2020
- Revision 1: June 5, 2020

SimpleDS is a python packaged developed to perform FFTs along frequency axis of radio interferometric data and perform all possible baseline cross multiplications, while accounting for units and cosmological conversion factors. In order to provide a credible alternative power spectrum estimate to the hera_pspec power spectrum pipeline, additional tests have been designed to evaluate the outputs of this python package against known and unknown inputs.

For a complete list of a tests involved in the shadow pipeline evaluation and more information on the parameters of each test, refer to the document attached at the end of this memo. I will provide a brief summary here. The shadow pipeline test suite involves five total tests characterized by the following simulation parameters:

0.1 A known input power spectrum with no noise.

1.1 Foregrounds, noise, no EoR and no systematic

1.2 Foregrounds, noise, an unknown EoR and no systematic

1.3 Foregrounds, noise, no EoR and an unknown systematic

1.4 Foregrounds, noise, an unknown EoR and an unknown systematic

This memo details the results from the HERA SimpleDS shadow pipeline test 0.1 "Flat $P(k)$". The data product analyzed consists of simulated HERA data containing one realization of three unique baseline groups. The goal is to estimate the input power spectrum the simulator used which is expected to be a flat in power spectrum space as a function of cosmological wavenumber, k. While not a necessary parameter of this test, we have elected to perform this test with the input power spectrum level unknown.

After initial analysis and results are reported, the input power spectrum value is compared with the results of this memo. Additional analysis is then appended to the original memo and any changes made to previous figures and tables are noted in line.

For completeness, the entire analysis notebook is also included.

```
[31]: %matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm, SymLogNorm
```

```
import os
from itertools import cycle
from IPython.display import display, Latex

from pyuvdata import UVData, UVBeam, utils as uvutils
import pyuvdata
import glob

from scipy import integrate
from scipy.signal import windows
from scipy.interpolate import interp1d, UnivariateSpline as USpline

from itertools import product

import simpleDS as sds
from simpleDS import  DelaySpectrum, utils, cosmo

from astropy import units, constants as const
from astropy.coordinates import Angle
from astropy.visualization import quantity_support
from astropy.cosmology import LambdaCDM, WMAP9, Planck15
```

[2]:
```
quantity_support();
```

[70]:
```
def sci_notation(num, err=None, decimal_digits=1, precision=None, exponent=None):
    """
    Returns a string representation of the scientific
    notation of the given number formatted for use with
    LaTeX or Mathtext, with specified number of significant
    decimal digits and precision (number of decimal digits
    to show). The exponent to be used can also be specified
    explicitly.
    """
    if isinstance(num, units.Quantity):
        unit = num.unit
        num = num.value

    else:
        unit = units.dimensionless_unscaled
    if exponent is None:
        exponent = np.int(np.floor(np.log10(np.abs(num))))
    coeff = np.round(num / np.float(10**exponent), decimal_digits)
    if precision is None:
        precision = decimal_digits
    if err is not None:
        err = err.to_value(unit)
        uncert = np.round(err / np.float(10**exponent), decimal_digits)
        return r"(${0:.{3}f} \pm {1:.{3}f}) \cdot10^{{{2:d}}}$ {4}".
↪format(coeff, uncert, exponent, precision, unit.to_string("latex"))
    else:
        return r"${0:.{2}f}\cdot10^{{{1:d}}}$ {3}".format(coeff, exponent,␣
↪precision, unit.to_string("latex"))
```

[4]:
```
# print version info
for m in [np, sds, pyuvdata]:
    print("{} verison {}".format(m.__name__, m.__version__))
```

```
numpy verison 1.18.4
simpleDS verison 2.0.1.dev14+gc154036
pyuvdata verison 2.0.3.dev149+gf7497fa2
```

[5]:
```
DATA_PATH="/lustre/aoc/projects/hera/alanman/eor_sky_sim/"
BEAM_PATH = "/lustre/aoc/projects/hera/Validation/HERA_beams"
data_files = [os.path.join(DATA_PATH, "eorsky_3.00hours_Nside128_sigma0.
↪03_fwhm12.13_uv.uvh5")]
beam_file = os.path.join(BEAM_PATH, "NF_HERA_dipole_IQ_power_healpix128.fits")
outfile = "/lustre/aoc/projects/hera/mkolopan/shadow_pipeline/flat_pk_sds.hdf5"
spws = [[0, 383]]
```

[6]:
```
uvb = UVBeam()
uvb.read_beamfits(beam_file)
```

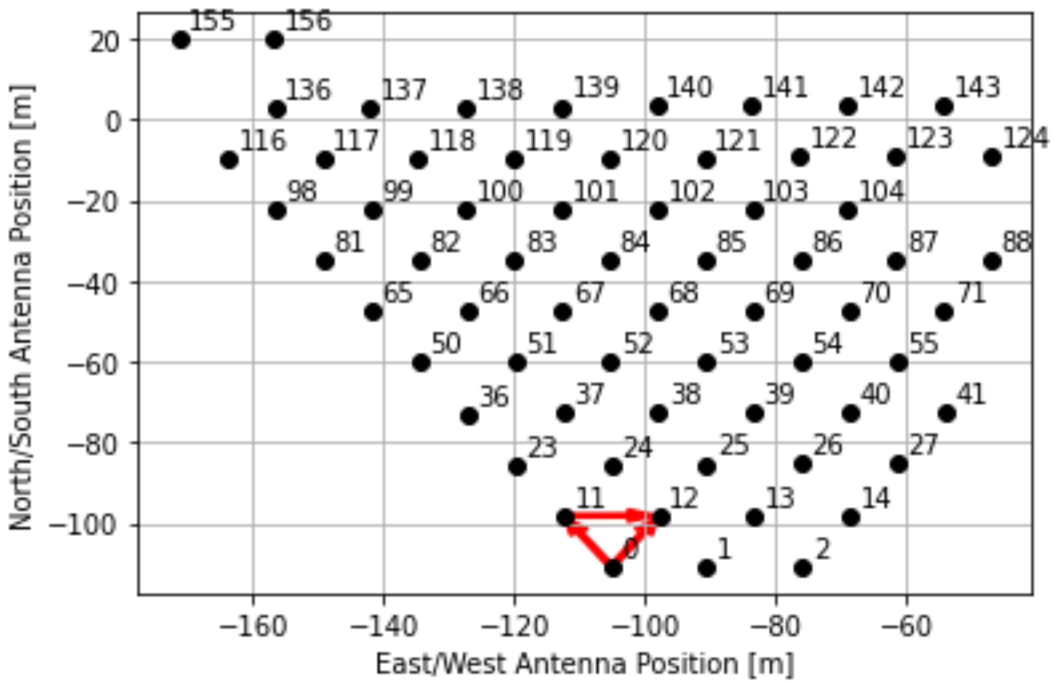
[7]:
```
uv = UVData()
uv.read(data_files[:], read_data=False, file_type='uvh5')
uv.read(
    data_files[:],
    file_type='uvh5',
    polarizations=np.intersect1d(
        uvb.polarization_array, uv.polarization_array
    )
)
```

```
Telescope eorsky is not in known_telescopes.
```

[8]:
```
uv.extra_keywords
```

[8]:
```
{'bm_fwhm': 12.129942517040208,
 'bsq_int': 0.025354522553525 03,
 'nside': 128,
 'skysig': 0.031,
 'slurm_id': '1321259'}
```

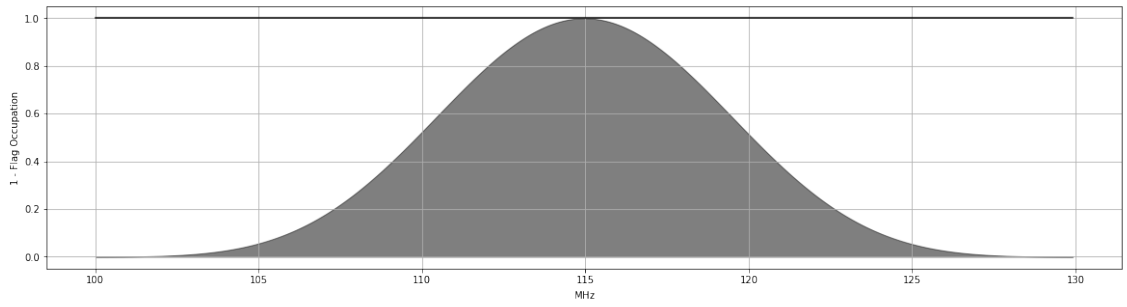
```
[12]: antpos, ants = uv.get_ENU_antpos(pick_data_ants=False)
      x, y, z = antpos.T
      fig, ax = plt.subplots(1, facecolor='white')
      plt.plot(x,y, 'ko');

      # ax1.set_xlim([50, 175])
      # ax1.set_ylim([200,300])
      for _ant, (_x,_y) in zip(ants, zip(x,y)):
          xlims = ax.get_xlim()
          ylims = ax.get_ylim()
          if np.logical_and(
              _x >= np.min(xlims), _x < np.max(xlims)
          ) and  np.logical_and(
              _y >= np.min(ylims), _y < np.max(ylims)
          ):
              ax.text(_x +1.5, _y + 2.5, _ant)

      for bl, bl_ind in zip(*np.unique(uv.baseline_array, return_index=True)):
          a1, a2 = uv.baseline_to_antnums(bl)
          ind1 = np.argwhere(ants == a1)
          ax.arrow(
              x[ind1].squeeze(),
              y[ind1].squeeze(),
              uv.uvw_array[bl_ind, 0],
              uv.uvw_array[bl_ind, 1],
              length_includes_head=True,
              color='red',
              width=1.0625,
          )

      ax.grid()
      ax.set_xlabel("East/West Antenna Position [m]");
      ax.set_ylabel("North/South Antenna Position [m]");
```



The simulation consists of the three baselines (0, 11), (0, 12), and (11, 12) to represent the three shortest redundant groups of the HERA hex show in the figure and covers the spectral band of 100-130 MHz.

The default spectral taper for a SimpleDS object is a Blackman-Harris shown in the figure below along with the inverse flag occupation (total fraction of unflagged samples). For a simulation this is expected to be identically 1.

[13]:
```
fig, ax = plt.subplots(1, figsize=(20,5))
freqs = uv.freq_array[0] * units.Hz
flags = uv.nsample_array.astype(float).squeeze()
ax.plot(
    freqs.to("MHz"),
    flags.sum(0)/flags.sum(0).max(),
    'k-'
);

for chan in spws:
    mid = np.mean(chan)
    _w = windows.blackmanharris(chan[1] - chan[0] + 1)
    ax.fill_between(freqs[chan[0] : chan[1] + 1].to('MHz'), 0, _w,␣
 ↪color='black',alpha=.5 )
ax.set_ylabel("1 - Flag Occupation")
ax.grid()
```



Since a SimpleDS object can only perform power spectrum estimation on a single baseline group, three objects are used to estimate the power spectrum of all redundant groups in this simulations.

[14]:
```
bl_cross = [(11, 12), (0, 11), (0, 12)]
times = np.unique(uv.time_array)

uv1 = uv.select(bls=bl_cross[0], times=times[::2], inplace=False)
uv2 = uv.select(bls=bl_cross[0], times=times[1::2], inplace=False)
ds = DelaySpectrum(uv=[uv1, uv2])
ds.add_uvbeam(uvb)
ds.cosmology = Planck15
# ds.set_taper(windows.boxcar)

uv1 = uv.select(bls=bl_cross[1], times=times[::2], inplace=False)
uv2 = uv.select(bls=bl_cross[1], times=times[1::2], inplace=False)
ds2 = DelaySpectrum(uv=[uv1, uv2])
ds2.add_uvbeam(uvb)
ds2.cosmology = Planck15

uv1 = uv.select(bls=bl_cross[2], times=times[::2], inplace=False)
uv2 = uv.select(bls=bl_cross[2], times=times[1::2], inplace=False)
ds3 = DelaySpectrum(uv=[uv1, uv2])
ds3.add_uvbeam(uvb)
ds3.cosmology = Planck15
```

```
lst_array parameter value is array, values are not close

Input LST arrays differ on average by 0.18383526862692248 min. Keeping LST array
stored from the first data set read.

lst_array parameter value is array, values are not close
lst_array parameter value is array, values are not close
```

[15]:
```
ds_list = [ds, ds2, ds3]
for _d in ds_list:
    ds.set_taper(windows.blackmanharris)
```

[18]:
```
fft_freq = lambda x: np.fft.fftshift(np.fft.fft(x * ds.taper(x.shape[1]).
 ↪reshape(1,-1), axis=1), axes=1)
fft_fringe = lambda x: np.fft.fftshift(np.fft.fft(x * ds.taper(x.shape[0]).
 ↪reshape(-1,1), axis=0), axes=0)
fringe_rates = (np.fft.fftshift(np.fft.fftfreq(ds.Ntimes, d=np.diff(ds.lst_array.
 ↪to('hourangle'))[0].value) )/units.hour).to('mHz')
```

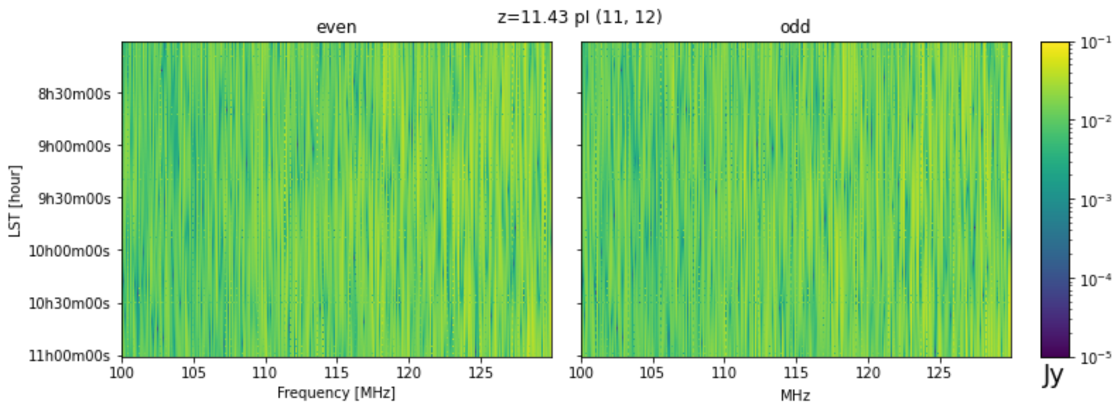
The Data

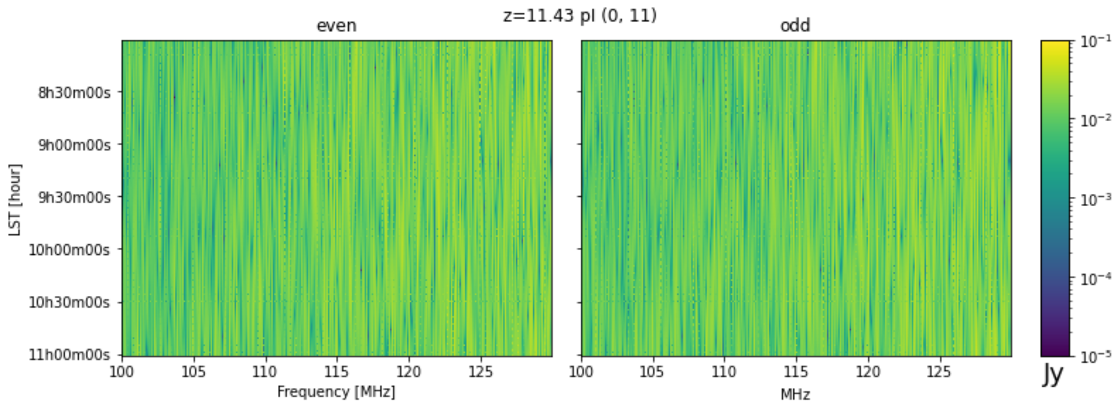
A quick look at the data in the simulations illustrates a noise-like variation in the amplitude over frequency and time. This is consistent with the expected underlying flat $P(k)$ input.

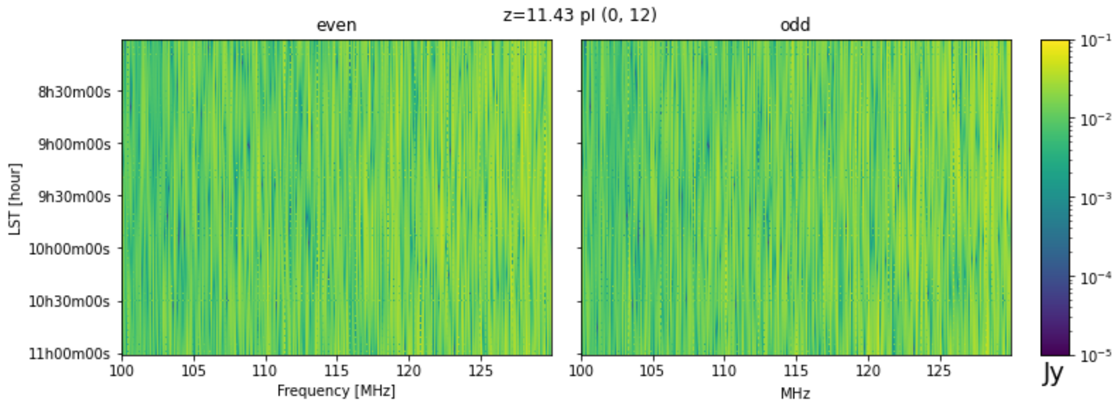
[19]:
```
ncols = ds.Nuv + 1
nrows = ds.Nbls - np.all(ds.flag_array, axis=(0,1,2,4,5)).sum()
cmap = plt.cm.viridis
cmap.set_bad('black')
vmin = -5
vmax = 5
norm = LogNorm(vmin=1e-5, vmax=1e-1)
for _d in ds_list:
    fig, ax = plt.subplots(
            figsize=(12, 3 * nrows + 1),
            nrows=nrows,
            ncols=ncols,
            sharex=False,
            sharey=False,
            facecolor='white',
            gridspec_kw={"width_ratios":[15] * ds.Nuv + [1]},
            squeeze=False,
            )
    fig.subplots_adjust(hspace=.35, wspace=0.1, top=.9)
    fig.suptitle(
        f"z={_d.redshift.item(0):.2f} "
        f"{uvutils.polnum2str(_d.polarization_array.item(0))} "
        f"{uvutils.baseline_to_antnums(_d.baseline_array[0], _d.
 ↪Nants_telescope)}"
    )
    for day_cnt, day in enumerate(["even", "odd"]):
        for bl_cnt, bl in enumerate(_d.baseline_array):

            if np.all(ds.flag_array[:, :, :, bl_cnt]):
                continue
            sharedy = ax[bl_cnt, 0].get_shared_y_axes()

            sharedx = ax[bl_cnt, 0].get_shared_x_axes()
            ax[bl_cnt, 0].set_ylabel("LST [hour]")

            ax[bl_cnt, 0].set_xlabel(r"Frequency [MHz]")
            for pol_cnt, pol in enumerate(uvutils.polnum2str(ds.
↪polarization_array)):
                sharedx.join(ax[bl_cnt, 0], ax[bl_cnt, day_cnt])
                sharedy.join(ax[bl_cnt, 0], ax[bl_cnt, day_cnt])

                masked_data = np.ma.masked_array(
                    _d.data_array[0, day_cnt, pol_cnt, bl_cnt].value,
                    mask=_d.flag_array[0, day_cnt, pol_cnt, bl_cnt]
                )

                im = ax[bl_cnt, day_cnt].pcolormesh(
                    _d.freq_array[0].to('MHz'),
                    _d.lst_array.to("hourangle"),
                    np.abs(masked_data),
                    cmap=cmap,
                    norm=norm,
                )

                ax[bl_cnt, day_cnt].set_title(f"{day}")
                _ylim = ax[bl_cnt, day_cnt].get_ylim()
                ax[bl_cnt, day_cnt].set_ylim([np.max(_ylim), np.min(_ylim)])


                tick_labels =  ax[bl_cnt, day_cnt].get_yticks()
                tick_labels = [Angle(t*units.hourangle).to_string() for t in␣
↪tick_labels]
                ax[bl_cnt, day_cnt].set_yticklabels(tick_labels);

                if  0 < day_cnt < ncols - 1:
                    plt.setp(ax[bl_cnt, day_cnt].get_yticklabels(),␣
↪visible=False)
                    ax[bl_cnt, day_cnt].set_ylabel("")


            cbar = fig.colorbar(im, cax=ax[bl_cnt, -1], label="Jy")#
            cbar.ax.set_xlabel(
                cbar.ax.get_ylabel(),
                fontsize=18,
            )
            cbar.ax.set_ylabel(None)
```

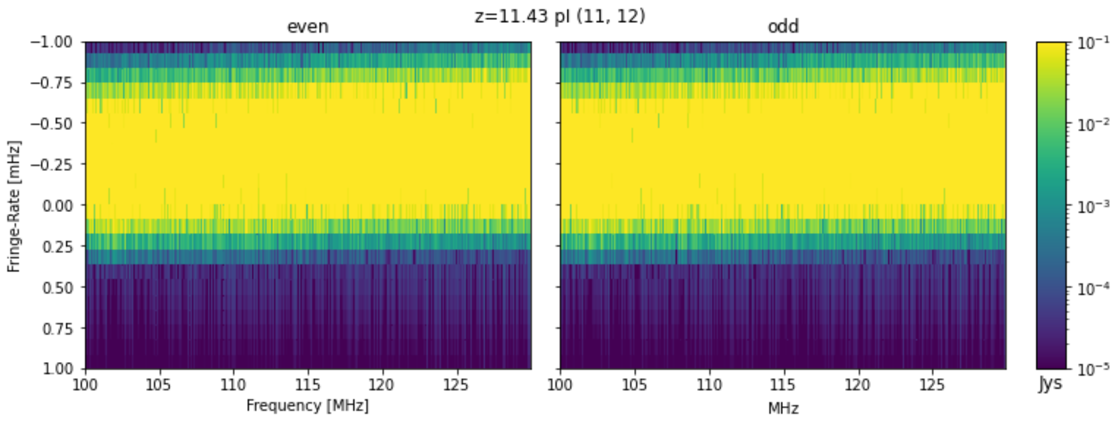
Transforming to Fringe-Rate space illustrates the slight evolution of the data over frequency with respect to Fringe-Rate for each baseline orientation. Again this small evolution is consistent with expectations over the 30MHz band. Positive and negative fringe-rates may be a result of conjugation conventions of the data.

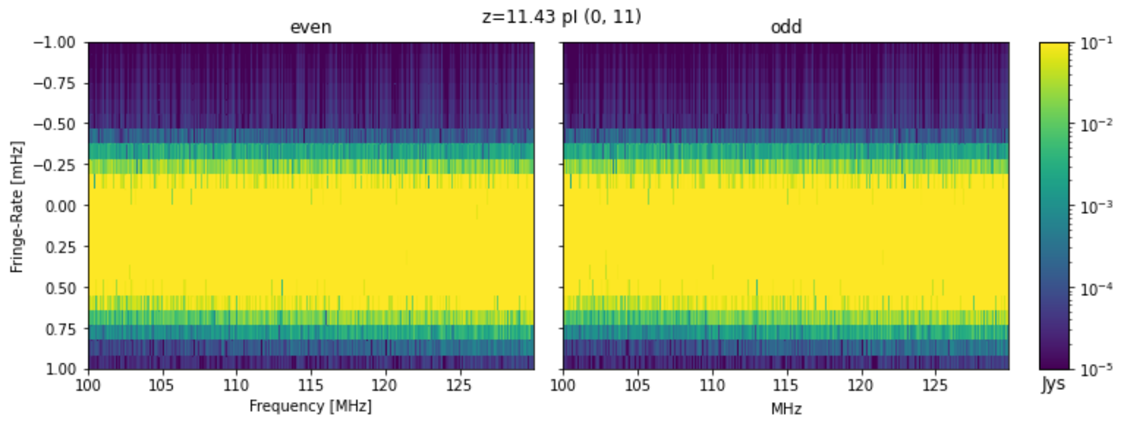
[20]:
```
ncols = ds.Nuv + 1
nrows = ds.Nbls - np.all(ds.flag_array, axis=(0,1,2,4,5)).sum()
cmap = plt.cm.viridis
cmap.set_bad('black')
vmin = -5
vmax = 5
norm = LogNorm(vmin=1e-5, vmax=1e-1)
for _d in ds_list:
    fig, ax = plt.subplots(
            figsize=(12, 3 * nrows + 1),
            nrows=nrows,
            ncols=ncols,
            sharex=False,
            sharey=False,
            facecolor='white',
            gridspec_kw={"width_ratios":[15] * ds.Nuv + [1]},
            squeeze=False,
            )
    fig.subplots_adjust(hspace=.35, wspace=0.1, top=.9)
    fig.suptitle(
        f"z={_d.redshift.item(0):.2f} "
        f"{uvutils.polnum2str(_d.polarization_array.item(0))} "
        f"{uvutils.baseline_to_antnums(_d.baseline_array[0], _d.
 ↪Nants_telescope)}"
    )
    for day_cnt, day in enumerate(["even", "odd"]):
        for bl_cnt, bl in enumerate(_d.baseline_array):

            if np.all(ds.flag_array[:, :, :, bl_cnt]):
                continue
            sharedy = ax[bl_cnt, 0].get_shared_y_axes()

            sharedx = ax[bl_cnt, 0].get_shared_x_axes()

            ax[bl_cnt, 0].set_ylabel("Fringe-Rate [mHz]")

            ax[bl_cnt, 0].set_xlabel(r"Frequency [MHz]")
            for pol_cnt, pol in enumerate(uvutils.polnum2str(ds.
 ↪polarization_array)):
                sharedx.join(ax[bl_cnt, 0], ax[bl_cnt, day_cnt])
                sharedy.join(ax[bl_cnt, 0], ax[bl_cnt, day_cnt])

                masked_data = np.ma.masked_array(
                    _d.data_array[0, day_cnt, pol_cnt, bl_cnt].value,
                    mask=_d.flag_array[0, day_cnt, pol_cnt, bl_cnt]
                )

                im = ax[bl_cnt, day_cnt].pcolormesh(
                    _d.freq_array[0].to('MHz'),
                    fringe_rates,
                    np.abs(fft_fringe(masked_data)),
                    cmap=cmap,
                    norm=norm,
                )

                ax[bl_cnt, day_cnt].set_title(f"{day}")
                _ylim = ax[bl_cnt, day_cnt].get_ylim()
                ax[bl_cnt, day_cnt].set_ylim([np.max(_ylim), np.min(_ylim)])

                if  0 < day_cnt < ncols - 1:
                    plt.setp(ax[bl_cnt, day_cnt].get_yticklabels(),␣
↪visible=False)
                    ax[bl_cnt, day_cnt].set_ylabel("")
            ax[bl_cnt, 0].set_ylim([1, -1])
            cbar = fig.colorbar(im, cax=ax[bl_cnt, -1], label="Jys")#
            cbar.ax.set_xlabel(
                cbar.ax.get_ylabel(),
                fontsize=12,
            )
            cbar.ax.set_ylabel(None)
```

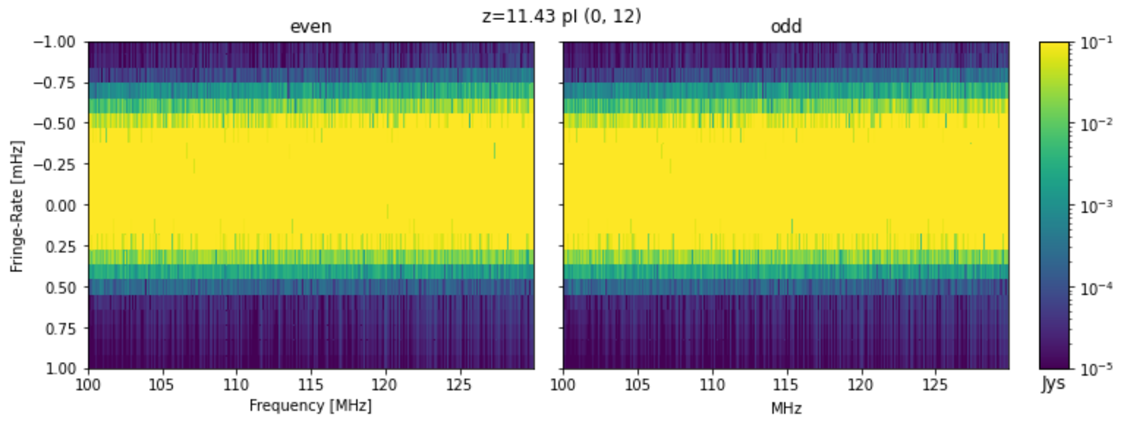



z=11.43 pI (0, 11)
even
odd
LST [hour]
8h30m00s
9h00m00s
9h30m00s
10h00m00s
10h30m00s
11h00m00s
Frequency [MHz]
MHz
Jy


z=11.43 pI (0, 12)
even
odd
LST [hour]
8h30m00s
9h00m00s
9h30m00s
10h00m00s
10h30m00s
11h00m00s
Frequency [MHz]
MHz
Jy


z=11.43 pI (11, 12)
even
odd
Fringe-Rate [mHz]
Frequency [MHz]
MHz
Jys





[21]:
```
for _d in ds_list:
    _d.select_spectral_windows(spws)
    _d.check()
```

[22]:
```
for _d in ds_list:
    _d.calculate_delay_spectrum(littleh_units=True)
```

[23]:
```
ncols = np.int(np.ceil(np.sqrt(ds.Npols)))
nrows = np.int(np.ceil(ds.Npols/float(ncols)))
```

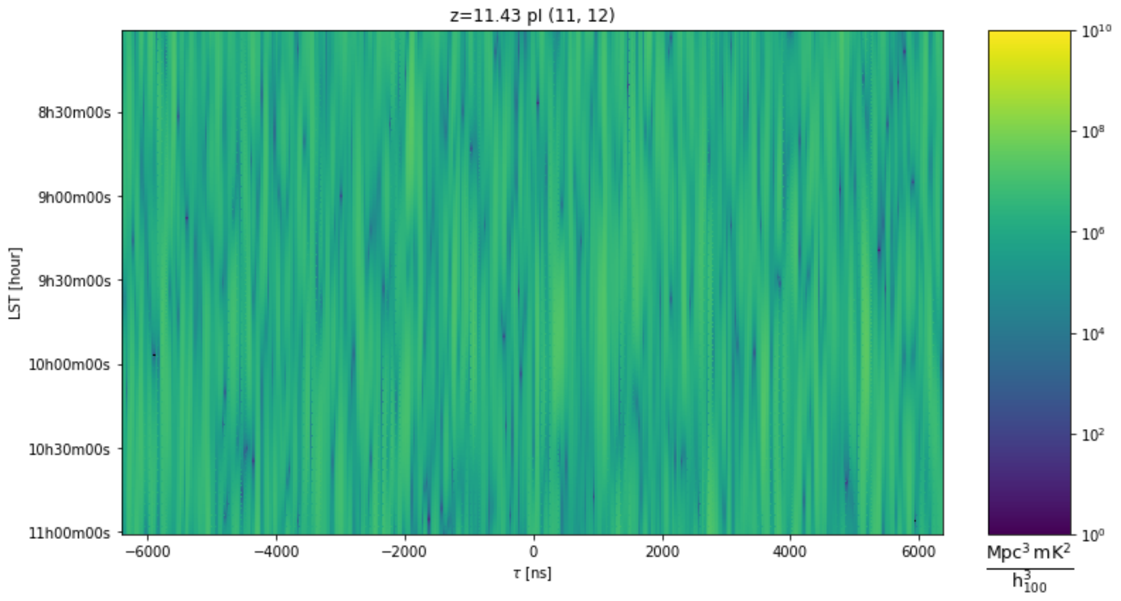
The Power Spectrum

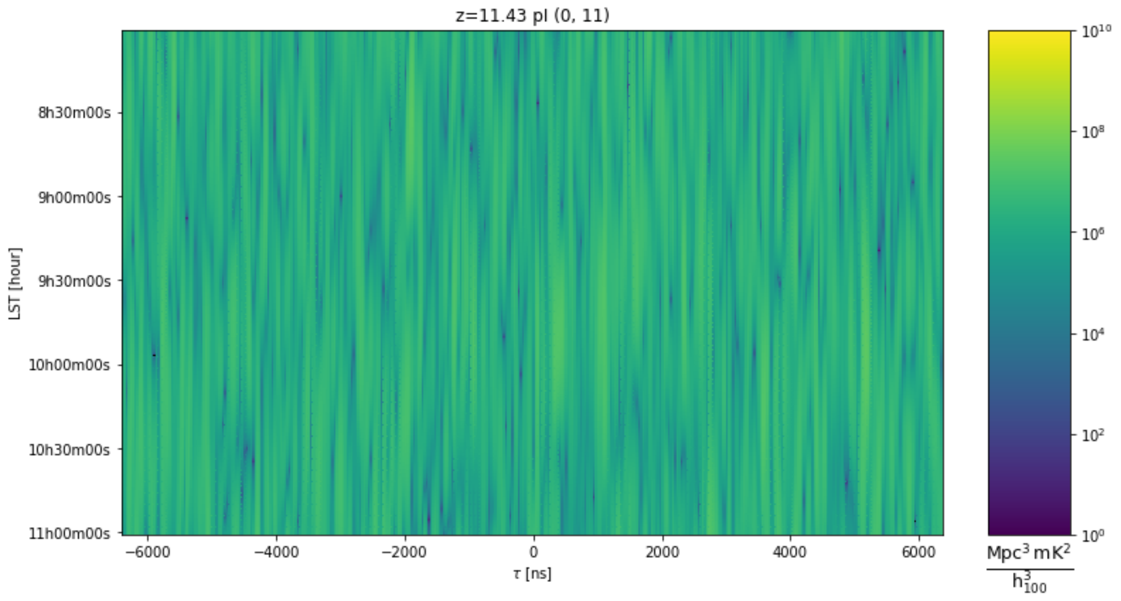
In power spectrum space the data again looks noise-like in amplitude, this is consistent with the Fourier transform of a flat spectrum input.

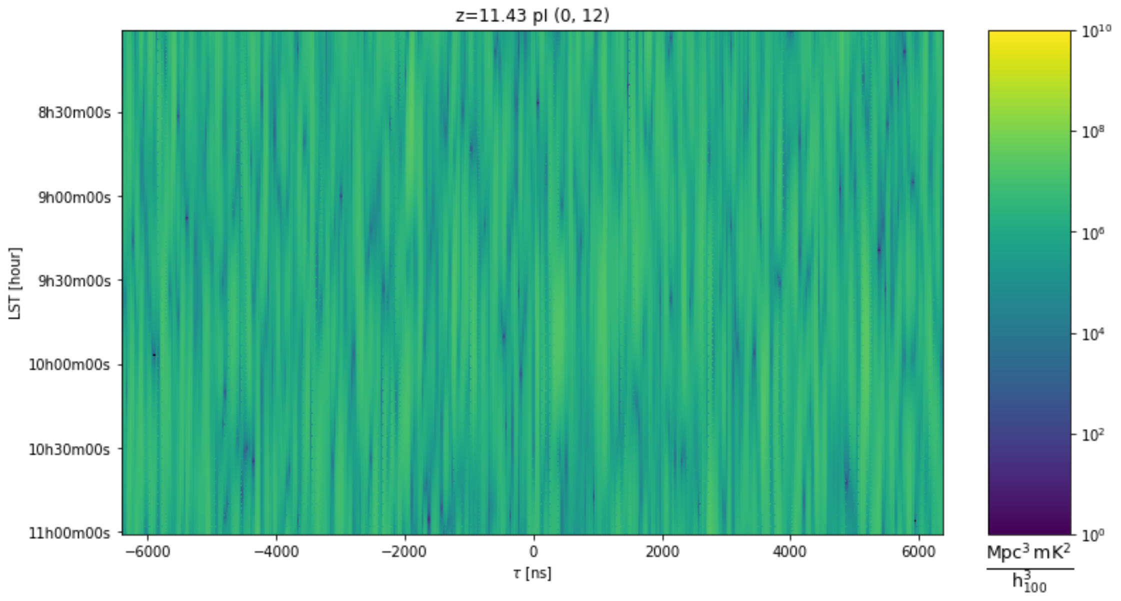
[24]:
```
cmap = plt.cm.viridis
cmap.set_bad('black')
```

```
vmin = -5
vmax = 5
norm = LogNorm(vmin=1e0, vmax=1e10)
for _d in ds_list:
    for z_cnt, z in enumerate(_d.redshift):
        ncols = _d.Npols + 1
        nrows = _d.Nbls - np.all(_d.flag_array, axis=(0, 1, 2, 4, 5)).sum()
        fig, ax = plt.subplots(
            figsize=(12, 5 * nrows + 1),
            nrows=nrows,
            ncols=ncols,
            sharex=False,
            sharey=False,
            facecolor='white',
            squeeze=False,
            gridspec_kw={"width_ratios":[10] * _d.Npols + [1]},
            )
        fig.subplots_adjust(hspace=.35, wspace=0.1, top=.95)
    #         fig.suptitle(f"{np.around(np.linalg.norm(_d.uvw.value),1)} {day:
 ↪s}")
        fig_count = 0
        for bl_cnt, bl in enumerate(_d.baseline_array):

            if np.all(ds.flag_array[:, :, :, bl_cnt]):
                continue
            sharedy = ax[fig_count,0].get_shared_y_axes()

            sharedx = ax[fig_count,0].get_shared_x_axes()

            ax[fig_count, 0].set_ylabel("LST [hour]")

            ax[fig_count, 0].set_xlabel(r"$\tau$ [ns]")
            for pol_cnt, pol in enumerate(uvutils.polnum2str(_d.
 ↪polarization_array)):
                sharedy.join(ax[fig_count,0], ax[fig_count, pol_cnt])
                sharedx.join(ax[fig_count,0], ax[fig_count, pol_cnt])
                masked_data = np.ma.masked_array(
                    ds.power_array[z_cnt, pol_cnt, 0, 0].value,
                    mask=ds.flag_array[z_cnt, pol_cnt, 0, 0],
                )
                im = ax[fig_count, pol_cnt].pcolormesh(
                    _d.delay_array.to('ns'),
                    _d.lst_array.to('hourangle'),
                    np.real(masked_data),
                    cmap=cmap,
                    norm=norm,
                )
                ax[fig_count, pol_cnt].set_title(
                    f"z={z:.2f} "
                    f"{uvutils.polnum2str(_d.polarization_array[pol_cnt])} "
                    f"{uvutils.baseline_to_antnums(_d.baseline_array[0], _d.
↪Nants_telescope)}"
                )
                _ylim = ax[fig_count, pol_cnt].get_ylim()
                ax[fig_count, pol_cnt].set_ylim([np.max(_ylim), np.min(_ylim)])

                tick_labels =  ax[fig_count, pol_cnt].get_yticks()
                tick_labels = [Angle(t*units.hourangle).to_string() for t in␣
↪tick_labels]
                ax[fig_count, pol_cnt].set_yticklabels(tick_labels);
                if  0 < pol_cnt < ncols - 1:
                    plt.setp(ax[fig_count, pol_cnt].get_yticklabels(),␣
↪visible=False)
                    ax[fig_count, pol_cnt].set_ylabel("")

            cbar = fig.colorbar(im, cax=ax[fig_count, -1], label=ds.power_array.
↪unit.to_string("latex"))#
            cbar.ax.set_xlabel(
                cbar.ax.get_ylabel(),
                fontsize=18,
            )
            cbar.ax.set_ylabel(None)
            fig_count += 1
```

[25]:
```
ncols = np.int(np.ceil(np.sqrt(ds.Npols)))
nrows = np.int(np.ceil(ds.Npols/float(ncols)))
```


z=11.43 pI (11, 12)
LST [hour]
8h30m00s
9h00m00s
9h30m00s
10h00m00s
10h30m00s
11h00m00s
−6000
−4000
−2000
0
2000
4000
6000
τ [ns]
10^{10}
10^8
10^6
10^4
10^2
10^0
$\frac{\mathrm{Mpc}^3\,\mathrm{mK}^2}{\mathrm{h}_{100}^3}$


z=11.43 pI (0, 11)
LST [hour]
8h30m00s
9h00m00s
9h30m00s
10h00m00s
10h30m00s
11h00m00s
−6000
−4000
−2000
0
2000
4000
6000
τ [ns]
10^{10}
10^8
10^6
10^4
10^2
10^0
$\frac{\mathrm{Mpc}^3\,\mathrm{mK}^2}{\mathrm{h}_{100}^3}$



Estimating Input $P(k)$

To properly estimate the variance of the power spectrum as a function in time, it is necessary to estimate the number of unique LST bins based on the FWHM of the input beam. To perform this estimation I use the following relations

$$\theta = \frac{\lambda}{D}$$

The beam crossing time in LSTs is then calculated as

$$LST_\Omega = \theta \frac{12 \text{ hours}}{\pi \, cos(l)}$$

where l is the latitude of the telescope location. Finally the number of unique LST bins is

$$N_{LSTS} = \frac{T_{obs}}{LST_\Omega}$$

where T_{obs} is the total length of the input simulation. **Revision 1: The form of this equation has changed slightly to more clearly indicate that the numerator should reflect the entire observation time of the input data.**

The uncertainty of the power spectrum is then estimated as the standard deviation over the LST axes, divided by $\sqrt{N_{LST}}$, or the standard deviation of the mean accounting for the total number of unique elements along this axis.

When fitting for $\hat{\mu}$ (input power spectrum level), the effective degrees of freedom must account for the number of unique delay bins after performing a spectral taper. Since the Blackman-Harris taper has a noise equivalent bandwidth (NEBW) of 2, the number of unique bins is then $\frac{N_{delays}}{2}$ where N_{delays} is the total number of delay bins. The total number of degrees of freedom of the fit is then $dof = \frac{N_{delays}}{2} - 1$.

All three baseline orientations fit an input power spectrum are not consistent with a single input $P(k)$ but average to $\hat{\mu} = 1.55 \times 10^6 \frac{\mathrm{Mpc}^3\,\mathrm{mK}^2}{\mathrm{h}_{100}^3}$. The exact fits, uncertainties and $\frac{\chi^2}{dof}$ are outlined in the table below.

	Unweighted Average			Inverse Variance Average		
Ant Pair	$\hat{\mu}_u$	$\hat{\sigma}_u$	$\frac{\chi^2}{dof}$	$\hat{\mu}_{iv}$	$\hat{\sigma}_{iv}$	$\frac{\chi^2}{dof}$
(11, 12)	2.70	0.13	2.706	1.56	0.06	1.077
| (0, 11)	2.69	0.11	2.616	1.64	0.06	1.193
| (0 ,12)	2.68	0.12	3.592	1.46	0.06	1.462

Table 1: The values of the $\hat{\mu}$ and $\hat{\sigma}$ are presented in the units $10^6 \frac{\mathrm{Mpc}^3\,\mathrm{mK}^2}{\mathrm{h}_{100}^3}$. **Revision 1: This table has been altered to also include the unweighted mean and uncertainty to compare against values computed during revision 1.**

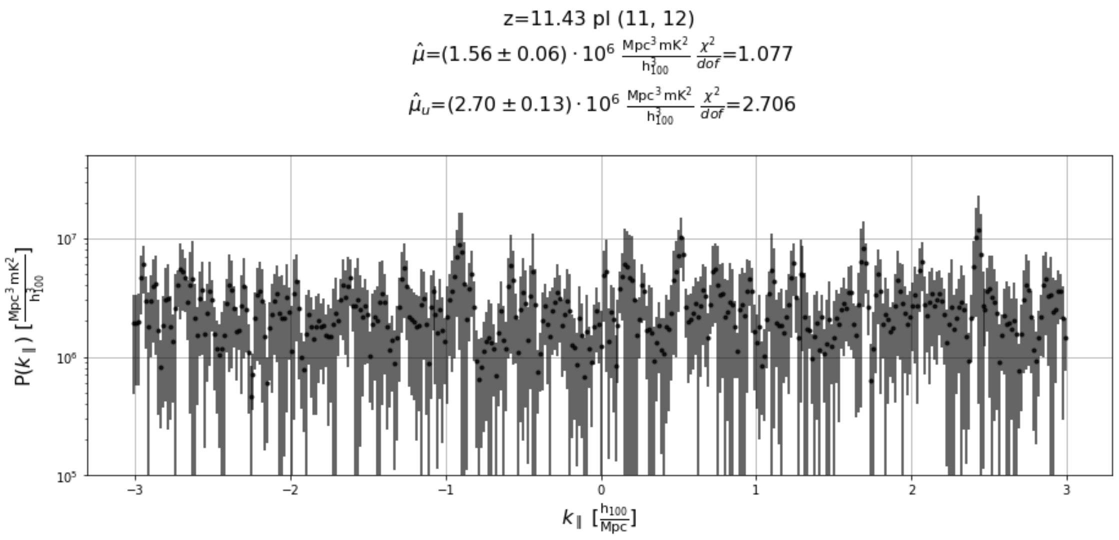
[55]:

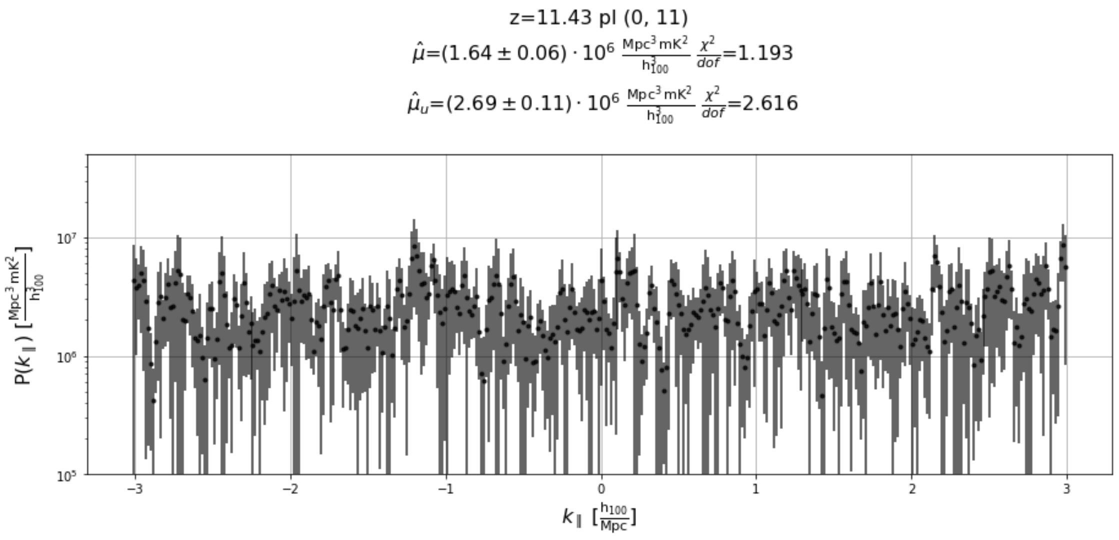
```
```

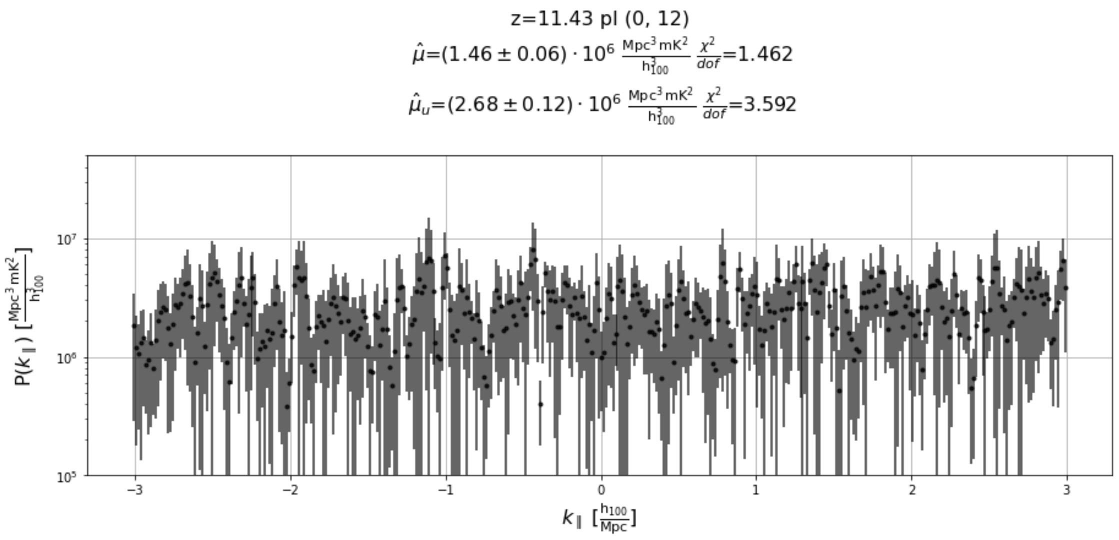
[77]:

```
for _d in ds_list:
    # Trying to estimate the number of independent time samples to properly␣
↪account for the number of samples when taking the variance.
    beam_crossing = ((const.c / _d.freq_array.mean()) / (14.4*units.m) * 12 *␣
↪units.h / np.pi / np.cos(uv.telescope_location_lat_lon_alt[0])).to(units.min)

    for z_cnt, redshift in enumerate(ds.redshift):
        fig, ax = plt.subplots(
            ncols=ncols,
            nrows=nrows,
            figsize=(15,5),
            facecolor='white',
            sharex=True,
            sharey=True,
            squeeze=False
        )
        ax = ax.ravel()
        for pol_ind in range(_d.polarization_array.size):
            vals = _d.power_array[z_cnt,pol_ind].mean((0,1,2)).real
            errs = 2 * _d.power_array[z_cnt,pol_ind].mean((0,1)).real.std(0) /␣
↪np.sqrt((_d.Ntimes * _d.integration_time.item(0) / beam_crossing).si)

            _mean =  np.average(vals, weights=1. / (errs) ** 2)
            _err = np.sqrt(1./np.sum(1. / (errs)**2))
            _chi2 = np.sum((_mean - vals) ** 2 / ((errs) ** 2))


            dof = (ds.Ndelays * utils.noise_equivalent_bandwidth(ds.taper(ds.
↪Nfreqs)) - 1)
```

```
            ax[pol_ind].errorbar(
                _d.k_parallel[0],
                vals,
                errs,
                fmt='k.', alpha=.8,
                label="Data" if pol_ind==0 else ''
            );

            ax[pol_ind].grid()
            ax[pol_ind].set_ylabel(r'P($k_{{\parallel}}$) [{0}]'.format((_d.
↪power_array.unit).to_string('latex')), fontsize=16)
            ax[pol_ind].set_xlabel(r'$k_{{\parallel}}$ [{0}]'.format(_d.
↪k_parallel.unit.to_string('latex')), fontsize=16)
            ax[pol_ind].set_title(
                f"z={_d.redshift[z_cnt]:.2f} "
                f"{uvutils.polnum2str(_d.polarization_array[pol_ind])} "
                f"{uvutils.baseline_to_antnums(_d.baseline_array[0], _d.
↪Nants_telescope)}\n"
                r" $\hat{\mu}$="
                f"{sci_notation(_mean, _err, decimal_digits=2)} "
                r"$\frac{\chi^{2}}{dof}$="
                f"{_chi2/dof:.3f}\n"
                , fontsize=16)

        sharex = ax[0].get_shared_x_axes()
        for pol_cnt in range(_d.polarization_array.size):
            sharex.join(ax[0], ax[pol_cnt])

        sharey = ax[0].get_shared_y_axes()
        for pol_cnt in range(ds.polarization_array.size):
            sharey.join(ax[0], ax[pol_cnt])
        ax[0].set_yscale('log')
        ax[0].set_ylim(1e5, 5e7)
        fig.subplots_adjust(wspace=.05, hspace=.175, top=.85)
        fig.suptitle(
            f" "

        );
#          ax[0].axhline(P0mk, color='red', linestyle='dashed')
#          fig.legend(frameon=False, ncol=3, numpoints=1,bbox_to_anchor=[.5, .
↪06]);
```


z=11.43 pl (11, 12)
$\hat{\mu}$=$(1.56 \pm 0.06) \cdot 10^6$ $\frac{\mathrm{Mpc}^3\,\mathrm{mK}^2}{\mathrm{h}_{100}^3}$ $\frac{\chi^2}{dof}$=1.077
$\hat{\mu}_u$=$(2.70 \pm 0.13) \cdot 10^6$ $\frac{\mathrm{Mpc}^3\,\mathrm{mK}^2}{\mathrm{h}_{100}^3}$ $\frac{\chi^2}{dof}$=2.706
$P(k_\parallel)$ $[\frac{\mathrm{Mpc}^3\,\mathrm{mK}^2}{\mathrm{h}_{100}^3}]$
$k_\parallel$ $[\frac{\mathrm{h}_{100}}{\mathrm{Mpc}}]$
10^5
10^6
10^7
−3
−2
−1
0
1
2
3


z=11.43 pl (0, 11)
$\hat{\mu}$=$(1.64 \pm 0.06) \cdot 10^6$ $\frac{\mathrm{Mpc}^3\,\mathrm{mK}^2}{\mathrm{h}_{100}^3}$ $\frac{\chi^2}{dof}$=1.193
$\hat{\mu}_u$=$(2.69 \pm 0.11) \cdot 10^6$ $\frac{\mathrm{Mpc}^3\,\mathrm{mK}^2}{\mathrm{h}_{100}^3}$ $\frac{\chi^2}{dof}$=2.616
$P(k_\parallel)$ $[\frac{\mathrm{Mpc}^3\,\mathrm{mK}^2}{\mathrm{h}_{100}^3}]$
$k_\parallel$ $[\frac{\mathrm{h}_{100}}{\mathrm{Mpc}}]$
10^5
10^6
10^7
−3
−2
−1
0
1
2
3



Revision 1: These figures have been altered to also include the mean and uncertainty obtained by performing an unweighted average.

Hidden Value Conclusions

This analysis produces estimates for the input flat power spectrum most consistent with $\hat{\mu} = 1.55 \times 10^6 \frac{\mathrm{Mpc^3\,mK^2}}{\mathrm{h^3_{100}}}$ for all three baseline groups.

- **Beginning of Revision 1: June 5, 2020**

Comparison with Input

After the initial results are reach without knowledge of the input $P(k)$, the parameters in the input UVData objects are read to compute the expected value of $P(k)$.

[59]:
```
# get the parameters needed to compute the expected amplitude of the power␣
 ↪spectrum
nside = uv.extra_keywords[u'nside']
dOmega = 4 * np.pi/(12 * nside ** 2)  * units.sr                  # sr
skysig = uv.extra_keywords[u'skysig'] * 1e3  * units.mK
df = np.diff(np.unique(uv.freq_array)).mean() * units.Hz # Hz
z = cosmo.calc_z(uv.freq_array.mean() * units.Hz)
X2Y = cosmo.X2Y(z) # (h^-1 Mpc)^3 sr^-1 Hz^-1

# calculate expected power spectrum amplitude
P0 = skysig ** 2 * df.si * dOmega * X2Y
#
P0mk=P0.to(ds.power_array.unit, units.with_H0(_d.cosmology.H0))
```

[72]:
```
display(Latex(" The initial test results were inconsistent with the expected␣
↪$P(k)$= " +sci_notation(P0.to("mK^2Mpc^3/littleh^3", units.with_H0(cosmo.
↪Planck15.H0)), decimal_digits=2)))
```

The initial test results are inconsistent with the expected $P(k)$= $2.89 \cdot 10^6\ \frac{\mathrm{Mpc}^3\,\mathrm{mK}^2}{\mathrm{h}_{100}^3}$

Analysis Differences

After reviewing validation test 0.1 notebook the following areas were identified to be potential causes of discrepancies

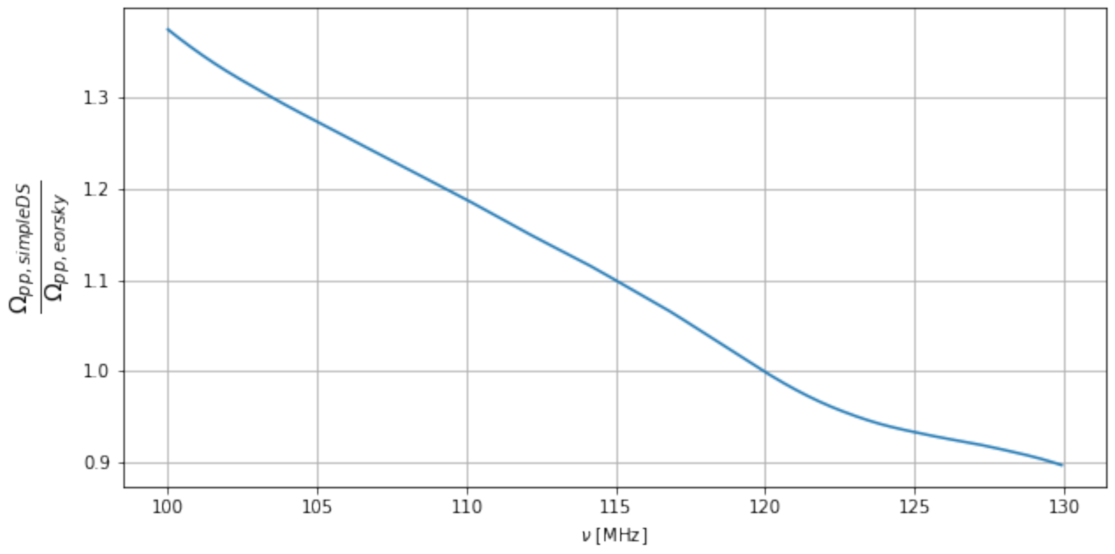
- Inverse Variance vs Unweighted averaging
- Power spectrum normalization via the beam square integral

This memo now reports values for fits using both weighting schemes to determine whether or not the choice of weighting scheme affects the fit value,

Power spectrum normalization and unit conversion factors can contribute to significant discrepancies between expected inputs and fit values.

To evaluate whether difference can cause discrepancies in the final fit, a ratio of the beam square area computed by simpleDS from the HERA beam input file ($\Omega_{pp,simpleDS}$), and the reported beam square integral in the provided data file ($\Omega_{pp,eorsky}$) is plotted below. Since all three baseline types are being transformed over the same contiguous band, only one beam squared integral is plotted in the figure.

[16]:
```
fig, ax = plt.subplots(1, figsize=(10,5))
ax.set_ylabel(r"$\frac{\Omega_{pp,simpleDS}}{\Omega_{pp,eorsky}}$", fontsize=20)
ax.set_xlabel(r"$\nu$ [MHz]")
ax.plot(ds.freq_array[0].to("MHz"), ds.beam_sq_area[0,0].value/uv.
 ↪extra_keywords["bsq_int"])
ax.grid()
```



This difference in the beam squared integral Ω_{pp} can result in normalization difference in final power spectrum.

The ratio of the $\int \frac{d\nu}{\Omega_{pp}(\nu)}$ for each beam square integral will provide an overall normalization factor, B_{Ω}, which may explain small difference in the fit values of the input power spectrum.

$$B_{\Omega} = \frac{\int \frac{d\nu}{\Omega_{pp,simpleDS}}}{\int \frac{d\nu}{\Omega_{pp,eorsky}}}$$

```
[17]: beam_sq_int_sds = integrate.trapz(1./ds.beam_sq_area[0,0], x=ds.freq_array[0])
      beam_sq_int_input = integrate.trapz(np.ones_like(ds.beam_sq_area[0,0].value)/(uv.
       ↪extra_keywords["bsq_int"] * units.sr), x=ds.freq_array[0])
      b_omega = beam_sq_int_sds/beam_sq_int_input
      print(b_omega, f"Percent difference: {100*(1-b_omega):.3f}")
```

```
0.9216614501215799 Percent difference: 7.834
```

Indicating up to $\sim$ 8% difference may be expected between the input $P(k)$ and the fit power spectrum values.

This normalization factor B_{Ω} is not a value normally applied to data. It is a necessary factor to convert between the integrated spectrally varying beam square area of the beam provided for the shadow pipeline analysis and the spectrally constant beam reported by the simulation file.

Refined Results

The fitting is repeated but this time accounting for the B_{Ω} normalization discrepancy, this is equivalent to multiplying each $\hat{\mu}$ and the uncertainties $\hat{\sigma}$ by the scale factor $\frac{1}{B_{\Omega}}$.

Two fits are also now performed, an average ($\hat{\mu}_u$) to be consistent with previous test 0.1 notebook averaging scheme and the inverse variance average ($\hat{\mu}_{iv}$) to compare how results change with the analysis previously computed in this memo.

All three baseline orientations fit an input power spectrum consistent with an unweighted fit consistent with $\hat{\mu}_u = 2.92 \times 10^6 \frac{\mathrm{Mpc}^3\,\mathrm{mK}^2}{\mathrm{h}_{100}^3}$ and when fit with inverse variance weighting are not all consistent with a single input $P(k)$ but will average to $\hat{\mu}_{iv} = 1.69 \times 10^6 \frac{\mathrm{Mpc}^3\,\mathrm{mK}^2}{\mathrm{h}_{100}^3}$. The exact fits, uncertainties and $\frac{\chi^2}{dof}$ are outlined in the table below.

	Unweighted Average			Inverse Variance Average		
Ant Pair	$\hat{\mu}_u$	$\hat{\sigma}_u$	$\frac{\chi^2}{dof}$	$\hat{\mu}_{iv}$	$\hat{\sigma}_{iv}$	$\frac{\chi^2}{dof}$
(11, 12)	2.92	0.14	2.706	1.70	0.07	1.077
(0, 11)	2.92	0.12	2.616	1.78	0.07	1.193
(0 ,12)	2.91	0.13	3.592	1.58	0.07	1.462

Table 2: The values of the $\hat{\mu}$ and $\hat{\sigma}$ are presented in the units $10^6 \frac{\mathrm{Mpc}^3\,\mathrm{mK}^2}{\mathrm{h}_{100}^3}$.

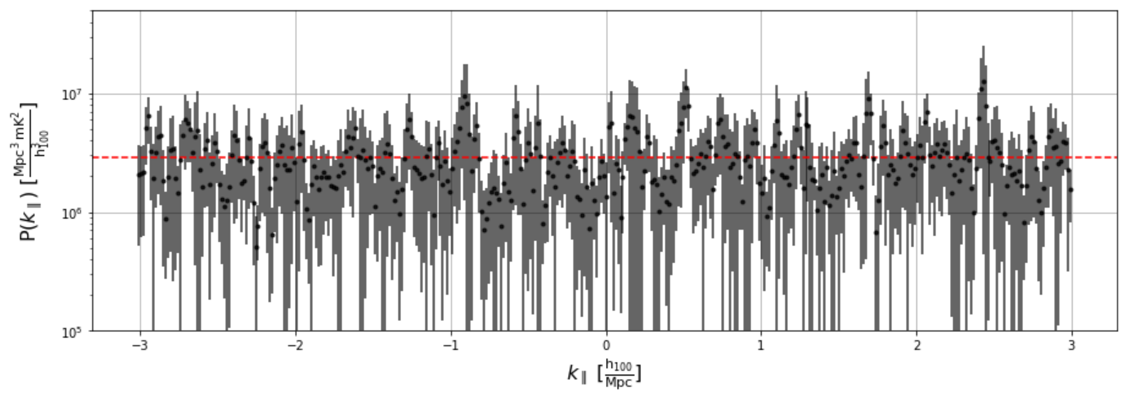
[76]:
```
for _d in ds_list:
    # Trying to estimate the number of independent time samples to properly␣
 ↪account for the number of samples when taking the variance.
    beam_crossing = ((const.c / _d.freq_array.mean()) / (14.4*units.m) * 12 *␣
 ↪units.h / np.pi / np.cos(uv.telescope_location_lat_lon_alt[0])).to(units.min)

    for z_cnt, redshift in enumerate(ds.redshift):
        fig, ax = plt.subplots(
            ncols=ncols,
            nrows=nrows,
            figsize=(15,5),
            facecolor='white',
            sharex=True,
            sharey=True,
            squeeze=False
        )
        ax = ax.ravel()
        for pol_ind in range(_d.polarization_array.size):
            vals = _d.power_array[z_cnt,pol_ind].mean((0,1,2)).real/ b_omega
            errs = 2 * _d.power_array[z_cnt,pol_ind].mean((0,1)).real.std(0) /␣
 ↪np.sqrt((_d.Ntimes * _d.integration_time.item(0) / beam_crossing).si) / b_omega

            u_mean = np.average(vals)
            u_err = np.sqrt(np.sum(errs ** 2)/np.abs(errs.size) **2 )
            u_chi2 = np.sum((u_mean - vals) ** 2 / ((errs) ** 2))

            iv_mean =  np.average(vals, weights=1. / (errs) ** 2)
            iv_err = np.sqrt(1./np.sum(1. / (errs)**2))
            iv_chi2 = np.sum((iv_mean - vals) ** 2 / ((errs) ** 2))


            dof = (ds.Ndelays * utils.noise_equivalent_bandwidth(ds.taper(ds.
 ↪Nfreqs)) - 1)


            ax[pol_ind].errorbar(
                _d.k_parallel[0],
                vals,
                errs,
                fmt='k.', alpha=.8,
                label="Data" if pol_ind==0 else ''
            );

            ax[pol_ind].grid()
            ax[pol_ind].set_ylabel(r'P($k_{{\parallel}}$) [{0}]'.format((_d.
 ↪power_array.unit).to_string('latex')), fontsize=16)
```

```
            ax[pol_ind].set_xlabel(r'$k_{{\parallel}}$ [{0}]'.format(_d.
↪k_parallel.unit.to_string('latex')), fontsize=16)
            ax[pol_ind].set_title(
                f"z={_d.redshift[z_cnt]:.2f} "
                f"{uvutils.polnum2str(_d.polarization_array[pol_ind])} "
                f"{uvutils.baseline_to_antnums(_d.baseline_array[0], _d.
↪Nants_telescope)}\n"
                r" $\hat{\mu}_{u}$="
                f"{sci_notation(u_mean, u_err, decimal_digits=2)} "
                r"$\frac{\chi^{2}}{dof}$="
                f"{u_chi2/dof:.3f}\n"
                r" $\hat{\mu}_{iv}$="
                f"{sci_notation(iv_mean, iv_err, decimal_digits=2)} "
                r"$\frac{\chi^{2}}{dof}$="
                f"{iv_chi2/dof:.3f}\n"
                , fontsize=16)

        sharex = ax[0].get_shared_x_axes()
        for pol_cnt in range(_d.polarization_array.size):
            sharex.join(ax[0], ax[pol_cnt])

        sharey = ax[0].get_shared_y_axes()
        for pol_cnt in range(ds.polarization_array.size):
            sharey.join(ax[0], ax[pol_cnt])
        ax[0].set_yscale('log')
        ax[0].set_ylim(1e5, 5e7)
        fig.subplots_adjust(wspace=.05, hspace=.175, top=.85)
        fig.suptitle(
            f" "

        );
        ax[0].axhline(P0mk, color='red', linestyle='dashed')
#          fig.legend(frameon=False, ncol=3, numpoints=1,bbox_to_anchor=[.5, .
↪06]);
```

z=11.43 pl (11, 12)

$\hat{\mu}_u = (2.92 \pm 0.14) \cdot 10^6 \, \frac{\mathrm{Mpc}^3\,\mathrm{mK}^2}{\mathrm{h}_{100}^3} \; \frac{\chi^2}{dof} = 2.706$

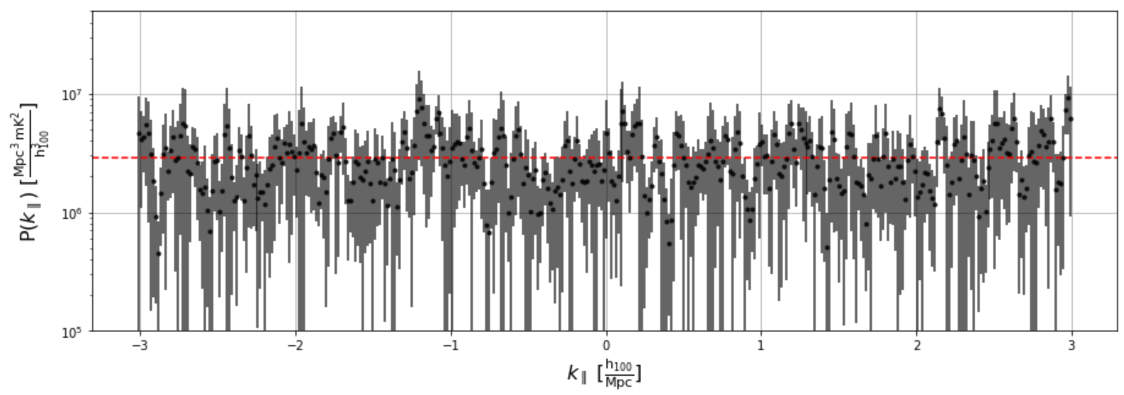
$\hat{\mu}_{iv} = (1.70 \pm 0.07) \cdot 10^6 \, \frac{\mathrm{Mpc}^3\,\mathrm{mK}^2}{\mathrm{h}_{100}^3} \; \frac{\chi^2}{dof} = 1.077$

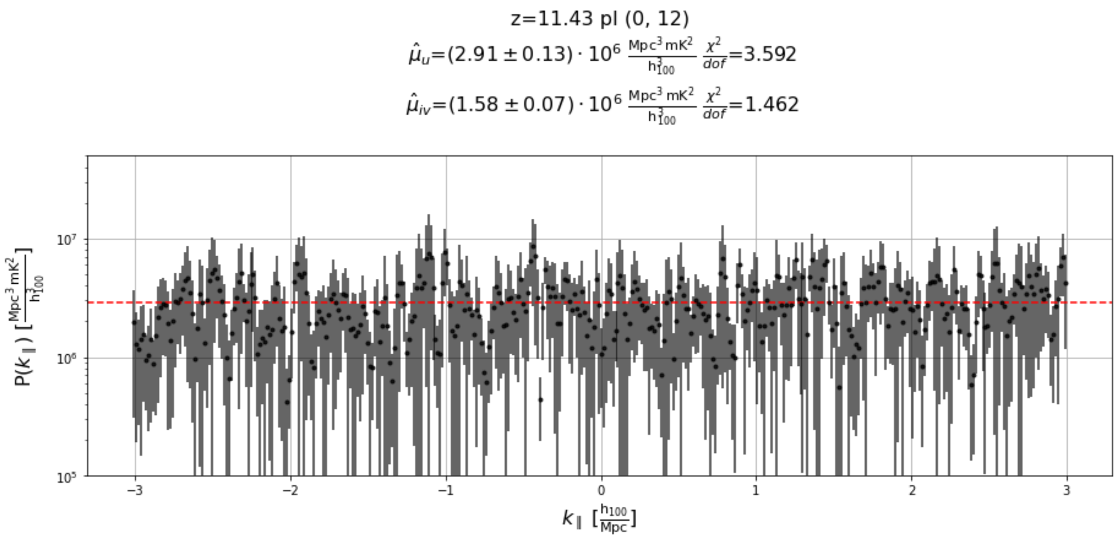


z=11.43 pl (0, 11)

$\hat{\mu}_u = (2.92 \pm 0.12) \cdot 10^6 \, \frac{\mathrm{Mpc}^3\,\mathrm{mK}^2}{\mathrm{h}_{100}^3} \; \frac{\chi^2}{dof} = 2.616$

$\hat{\mu}_{iv} = (1.78 \pm 0.07) \cdot 10^6 \, \frac{\mathrm{Mpc}^3\,\mathrm{mK}^2}{\mathrm{h}_{100}^3} \; \frac{\chi^2}{dof} = 1.193$





Conclusion

Comparing the fits before and after applying the normalization constant indicates the averaging scheme is the largest contributing factor to the discrepancy between expectation and fitted value.

When accounting for differences in averaging schemes and power spectrum normalization, the fit $\hat{\mu}$ is consistent with the expected value of $2.89 \cdot 10^6 \frac{\mathrm{Mpc^3\,mK^2}}{\mathrm{h_{100}^3}}$ however these fits are accompanied by a $\frac{\chi^2}{dof} \sim 3$ may indicate this is not the optimal fit for the computed power spectra.

Outstanding Questions

When this analysis is repeated with a boxcar spectral taper, numerical results are consistent with fits obtained above. However, the computed $\frac{\chi^2}{dof}$ indicates that the unweighted average is a better fit to the data than the inverse variance weighted average. This poses the question of whether the number of degrees of freedom for the fit are being computed correctly.