new phasing

May 21, 2021

```
[1]: from pyuvdata.uvdata import UVData
     import os
     import numpy as np
     import pyuvdata.utils as uvutils
     from pyuvdata.data import DATA_PATH
     import matplotlib.pyplot as plt
     from astropy.coordinates import Angle, SkyCoord
     from astropy.utils import iers
     from astropy.time import Time
```

The file `1133866760.uvfits` is mentioned in the phasing memo as having been the "truth" data that was used to verify the old data, so let's use it here to verify that the new phasing method looks reasonably correct.

```
[2]: filename = os.path.join(DATA_PATH, '1133866760.uvfits')
     uv_in = UVData();
     uv_in.read(filename);
```

```
[3]: uv_in = UVData()
     uv_in.read(os.path.join(DATA_PATH, '1133866760.uvfits'))
```

Since the above file is a UVFITS file, that means that apparent coordinates are not stored within the data itself – rather, they need to be calculated upon read, via the method `_set_app_coords_helper`. Important to note here that, as read, the new code assumes that the coordinate system is in the FK5 system, and not in GCRS as previously documented

```
[4]: print(uv_in.phase_center_frame, uv_in.phase_center_epoch)
```

```
fk5 2000.0
```

```
[5]: print("(RA,Dec) = ",end='')
     SkyCoord(
         uv_in.phase_center_ra,
         uv_in.phase_center_dec,
         unit='rad',
         frame=uv_in.phase_center_frame).to_string('hmsdms')
```

```
(RA,Dec) =
```

[5]: '01h39m00s -17d57m00s'

UVWs are read in from the UVFITS file, although we can manually recalculate them using either the `set_uvws_from_antenna_positions` method for `UVData` objects, or by using the `calc_uvw` function found in `pyuvdata.utils`.

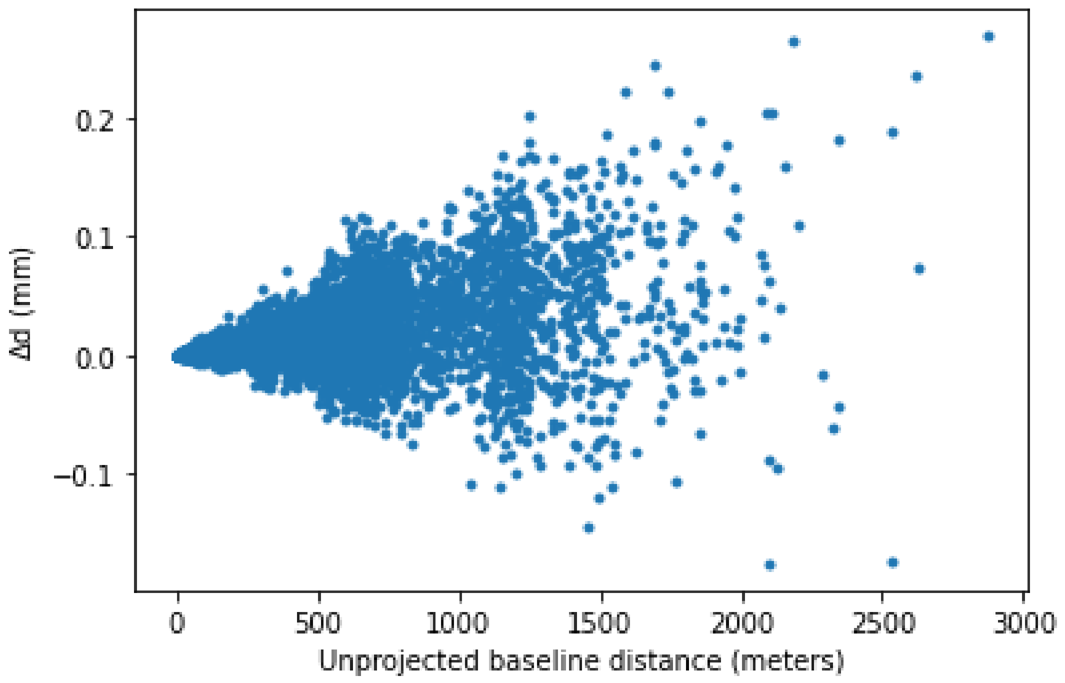
```
[6]: old_uvw_array = uv_in.uvw_array.copy()

     new_uvw_array = uvutils.calc_uvw(
         app_ra=uv_in.phase_center_app_ra,
         app_dec=uv_in.phase_center_app_dec,
         frame_pa=uv_in.phase_center_frame_pa,
         lst_array=uv_in.lst_array,
         antenna_positions=uv_in.antenna_positions,
         antenna_numbers=uv_in.antenna_numbers,
         ant_1_array=uv_in.ant_1_array,
         ant_2_array=uv_in.ant_2_array,
         telescope_lat=uv_in.telescope_location_lat_lon_alt[0],
         telescope_lon=uv_in.telescope_location_lat_lon_alt[1],
     )
```

Let's compare old (cotter-derived) vs new! First, lets compare relative lengths of the baseline vectors.

```
[7]: new_uvd = np.sum(new_uvw_array**2.0,axis=1)**0.5
     old_uvd = np.sum(old_uvw_array**2.0,axis=1)**0.5
     plt.plot(new_uvd,(new_uvd - old_uvd)*1000.0,'.')
     plt.xlabel('Unprojected baseline distance (meters)')
     plt.ylabel('$\Delta$d (mm)')
```

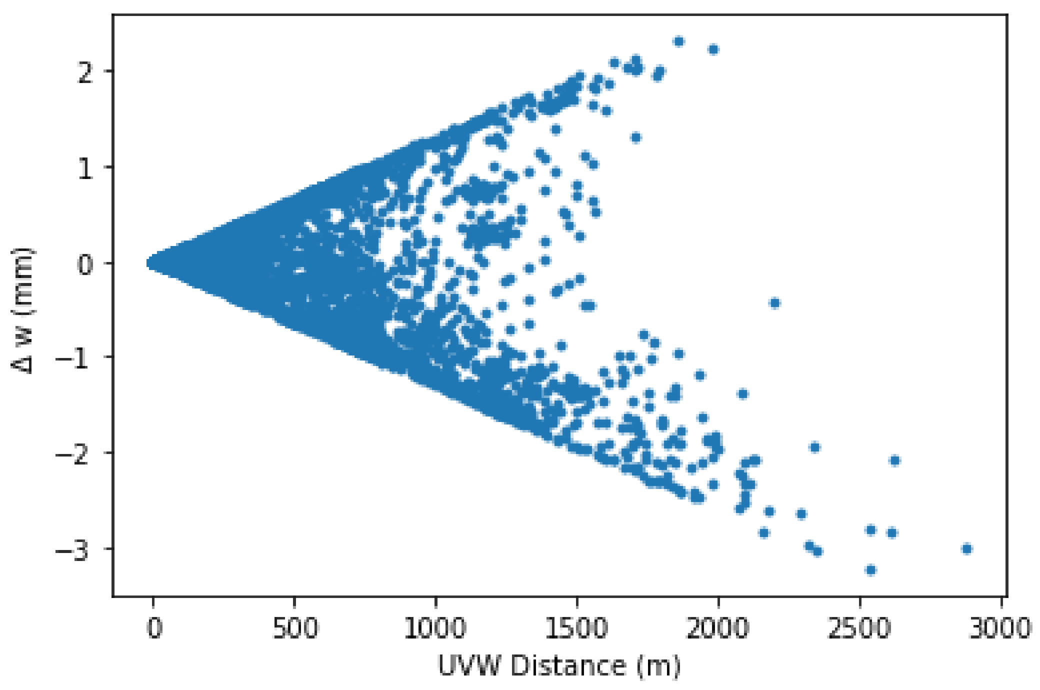
```
[7]: Text(0, 0.5, '$\\Delta$d (mm)')
```



Alright, total baseline lengths look good, and generally accurate to about 1 part in 10^7. Next, let's check the w-values, since that directly couples to the phasing.

[8]:
```
plt.plot(old_uvd,(new_uvw_array[:,2] - old_uvw_array[:,2])*1000,'.')
plt.xlabel('UVW Distance (m)')
plt.ylabel('$\Delta$ w (mm)')
```

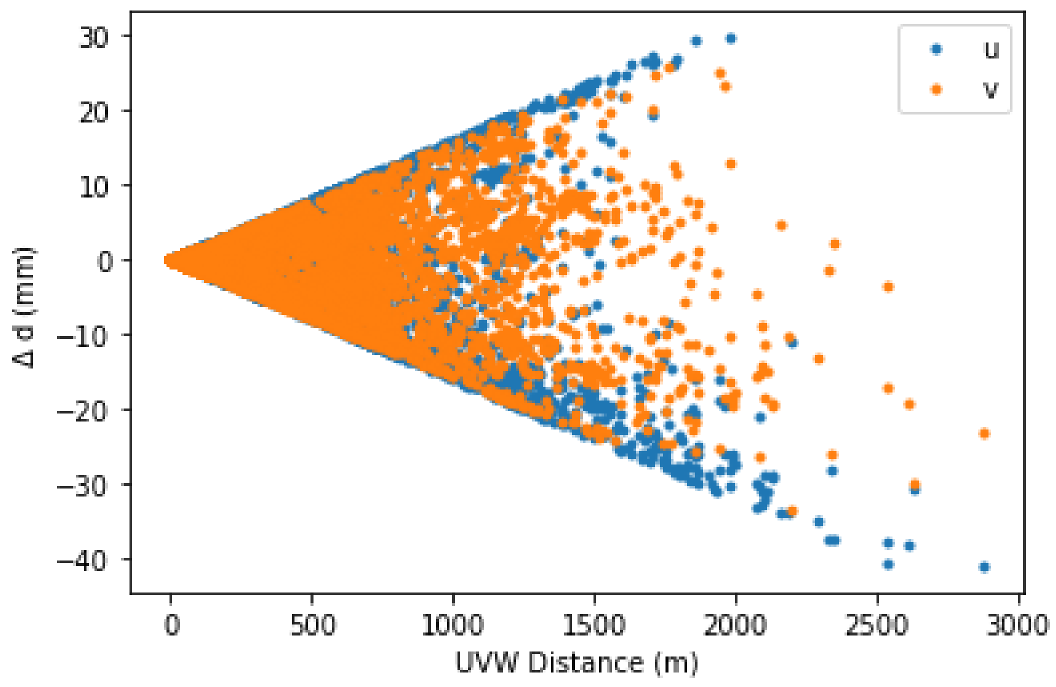
[8]:
```
Text(0, 0.5, '$\\Delta$ w (mm)')
```



The differences here for w are larger than expected – about one part in 10^6, or a positional error of the order of tenths of an arcsecond. Let's check u and v.

```
[9]: plt.plot(old_uvd,(new_uvw_array[:,0] - old_uvw_array[:,0])*1000,'.', label='u')
     plt.plot(old_uvd,(new_uvw_array[:,1] - old_uvw_array[:,1])*1000,'.', label='v')
     plt.xlabel('UVW Distance (m)')
     plt.ylabel('$\Delta$ d (mm)')
     plt.legend()
```

```
[9]: <matplotlib.legend.Legend at 0x7fc06d2a0100>
```



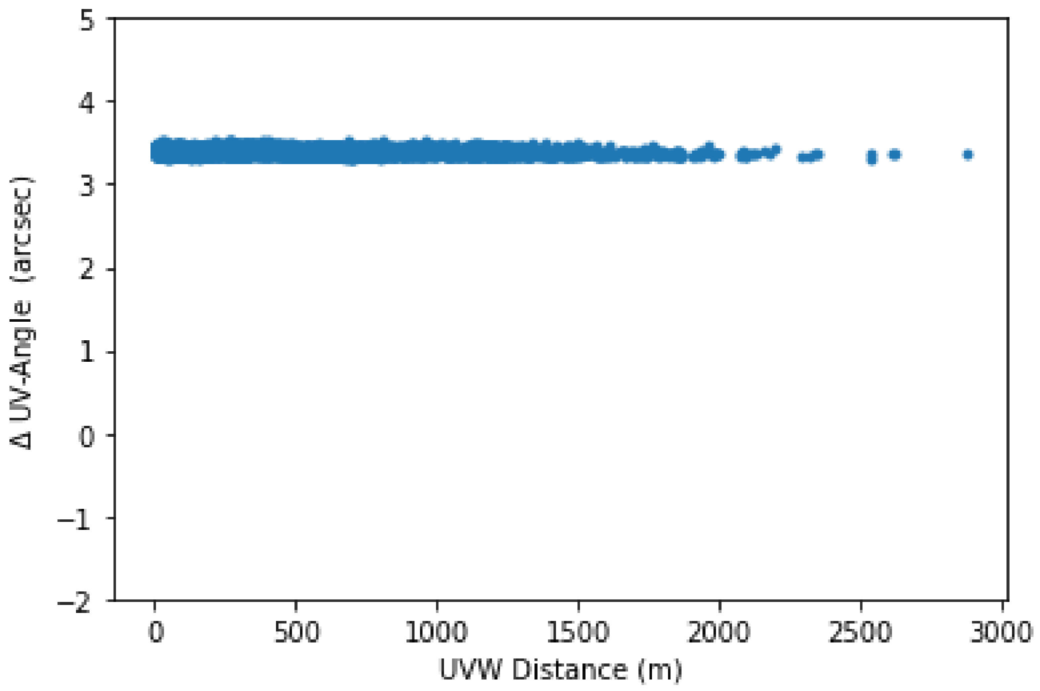
Differences here are larger still – on a hunch, I decide to check the difference in UV angle between the old and new UVW coordinates

[10]:
```
old_uva = np.arctan2(old_uvw_array[:,1], old_uvw_array[:,0]) * (180/np.pi) *␣
 ↪3600
new_uva = np.arctan2(new_uvw_array[:,1], new_uvw_array[:,0]) * (180/np.pi) *␣
 ↪3600
diff_uva = old_uva - new_uva
print(np.mean(diff_uva[np.abs(diff_uva) < 10]))

plt.plot(old_uvd,(old_uva-new_uva), '.')
plt.xlabel('UVW Distance (m)')
plt.ylabel('$\Delta$ UV-Angle  (arcsec)')
plt.ylim([-2, 5])
```

```
3.397284958266529
```

[10]:
```
(-2.0, 5.0)
```



Ah ha! It looks like the orientation angle of what we call the frame position angle is different, but by a relatively fixed amount. The shift here is relatively small – a total of 3 arcseconds in rotation, although it is large enough that several potential explanations are ruled out pretty quickly. The one term that might make the most sense would be if the annual abberation term were neglected, since that would look like a rotation of order ~10 arcsec. We can do a spot check of this by seeing what the difference in postion angle is between a GCRS and FK5 at the same RA and Dec.

[11]:
```
uv_in.phase_center_frame = 'gcrs'
uv_in._set_app_coords_helper()
gcrs_pa = uv_in.phase_center_frame_pa.copy()
uv_in.phase_center_frame = 'fk5'
uv_in._set_app_coords_helper()
print(np.mean(gcrs_pa - uv_in.phase_center_frame_pa) * (180 / np.pi) * 3600)
```

```
3.842006735900848
```

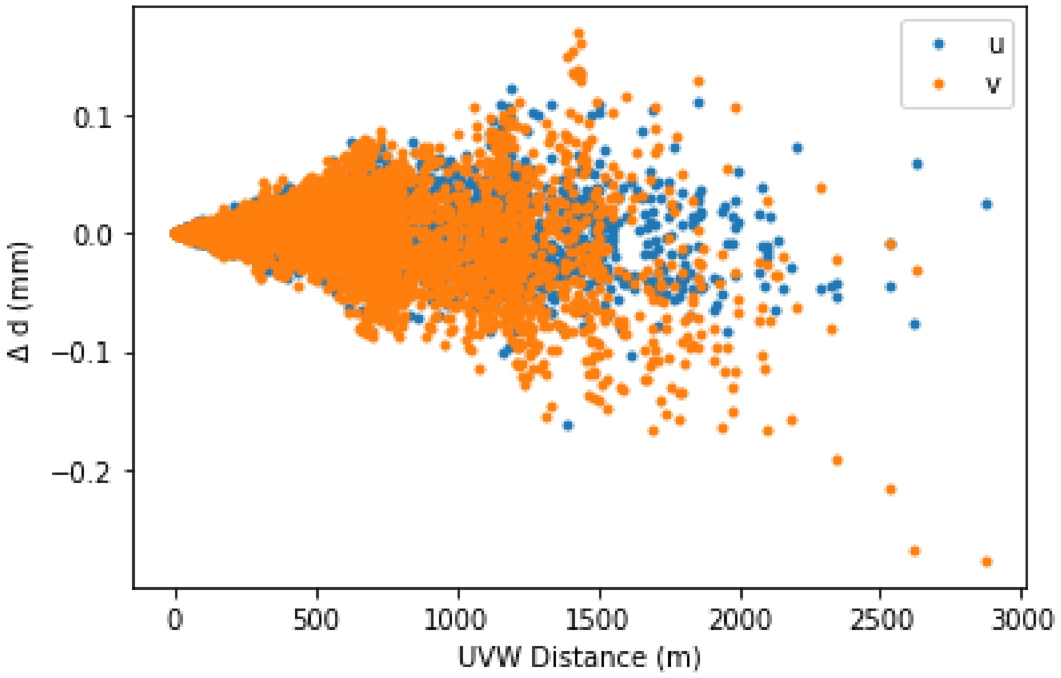
Bingo – that's a pretty excellent level of agreement, given that we haven't attempted to account for diurnal abberation or polar motion components.

With a bit of trial and error I decide to see what happens if I introduce some offsets to the apparent RA, Dec, and position angle of the coordiante frame to the apparent meridian. With some small adjustments, I find that I'm able to get much better agreement.

[12]:
```
new_uvw_array = uvutils.calc_uvw(
    app_ra=uv_in.phase_center_app_ra + Angle(-0.043, unit='arcsec').rad,
    app_dec=uv_in.phase_center_app_dec +  Angle(-0.2698, unit='arcsec').rad,
    frame_pa=uv_in.phase_center_frame_pa + Angle(3.37, unit='arcsec').rad,
    lst_array=uv_in.lst_array,
    antenna_positions=uv_in.antenna_positions,
    antenna_numbers=uv_in.antenna_numbers,
    ant_1_array=uv_in.ant_1_array,
    ant_2_array=uv_in.ant_2_array,
    telescope_lat=uv_in.telescope_location_lat_lon_alt[0],
    telescope_lon=uv_in.telescope_location_lat_lon_alt[1],
)
new_uvd = np.sum(new_uvw_array**2.0,axis=1)**0.5
old_uvd = np.sum(old_uvw_array**2.0,axis=1)**0.5
```

[13]:
```
plt.plot(new_uvd,(new_uvw_array[:,0] - old_uvw_array[:,0])*1000,'.',label='u')
plt.plot(new_uvd,(new_uvw_array[:,1] - old_uvw_array[:,1])*1000,'.',label='v')
plt.xlabel('UVW Distance (m)')
plt.ylabel('$\Delta$ d (mm)')
plt.legend()
```

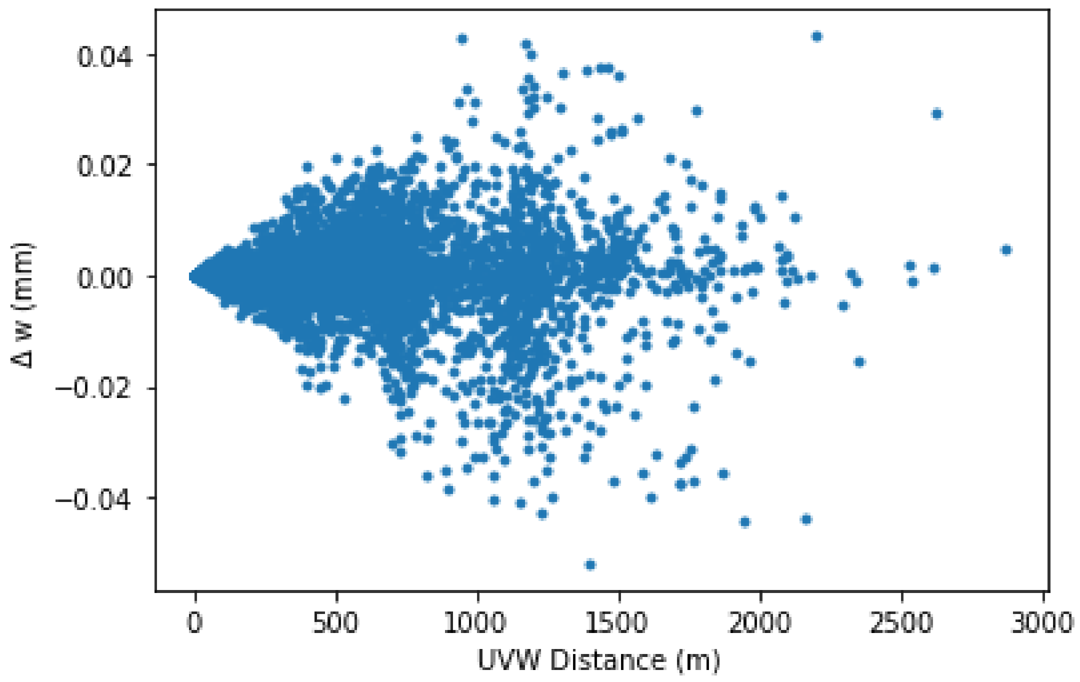
[13]:
```
<matplotlib.legend.Legend at 0x7fc06d52dac0>
```



Much better! Now let's check *w*...

[14]:
```
plt.plot(new_uvd,(new_uvw_array[:,2] - old_uvw_array[:,2])*1000,'.')
plt.xlabel('UVW Distance (m)')
plt.ylabel('$\Delta$ w (mm)')
```

```
[14]: Text(0, 0.5, '$\\Delta$ w (mm)')
```



We've got agreement down to the 10s of µm level, which is great!

The source of the *apparent* offset is not clear, and it is entirely possible that were forcing the right answer here my pseudo-random guess. However, it's probably not a terribly inaccurate statement to say that the new method would appear to agree with the UVWs derived by cotter down to the level of tenths of an arcsec (one part in 10^6), modulo the apparent discrepancy in the frame position angle. The exact source of the this discrepancy isn't clear – both the diurnal aberration and polar motion components are of order this offset, although I was not immediately able to get the error to cancel out by playing with these effects. It's also possible that a small error in the array reference center (to the tune of 40 meters) could also explain the discrepancy, particularly if the antenna positions were translated from ENU to ECEF. A time/LST error would seemingly be ruled out, given that the most significant component of the apparent offset is in declination, whereas a time error *should* show up as an RA offset.