pset_5

November 20, 2022

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib

font = {'family' : 'serif',
        'weight' : 'normal',
        'size'   : 14}

matplotlib.rc('font', **font)
```

1 Problem 1

For this entire problem set, we construct our source vector using our usual spherical coordinates notation:

$$\vec{s} = [\ \sin\theta\cos\phi,\ \sin\theta\sin\phi,\ \cos\theta\ ] = [\ \sin\theta, 0,\ \cos\theta\ ]. \tag{1}$$

Assuming the source is only in the NS direction, we set $\phi = 0$. Further, since we are still working in the global coordinate system (z points out of the North pole, and x and y form a plane along the equator).

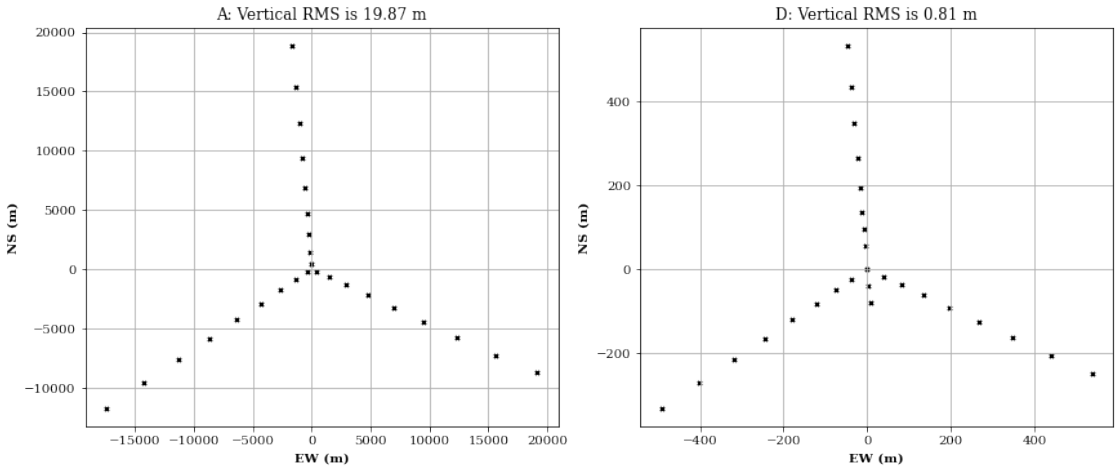
Note that in the 2D case, we ignore the third component. **To compute θ in our usual spherical coordinate system, θ begins at zero at the North pole and increases to 180 degrees at the South pole. In that case, for a given latitude in the Northern hemisphere, $\theta = 90 -$ latitude.**

1.1 Part A

[2]:
```
def get_antenna_pos(file, return_antpos=False):

    data = np.loadtxt(file)
    antpos = data[:, :3] * 1e-9 * 3e8

    latitude = (90 - 34.1) * np.pi / 180 # Want latitude in radians

    zenith = np.array([np.sin(latitude), 0, np.cos(latitude)])

    EW = np.array([0,1,0])
    NS = np.cross(zenith, EW)
```

```
    xyz = antpos@np.vstack([EW, NS, zenith]).T

    if return_antpos:
        return xyz, antpos

    return xyz
```

[3]:
```
files = ['vla_a_array.txt', 'vla_d_array.txt']


titles = ['A', 'D']
fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(12,5),␣
↪constrained_layout=True)

for i, f in enumerate(files):

    xyz = get_antenna_pos(f)

    # Extract EW and NS baselines
    EW = xyz[:,0]
    NS = xyz[:,1]


    vertical_rms = np.sqrt(np.mean(xyz[:,2]**2))

    axs[i].scatter(EW, NS, s = 10, marker='x', color='black')
    axs[i].set_title(titles[i] + ': Vertical RMS is {0} m'.
↪format(round(vertical_rms,2)))
    axs[i].set_xlabel('EW (m)', fontweight='bold')
    axs[i].set_ylabel('NS (m)', fontweight='bold')
    axs[i].grid()
```

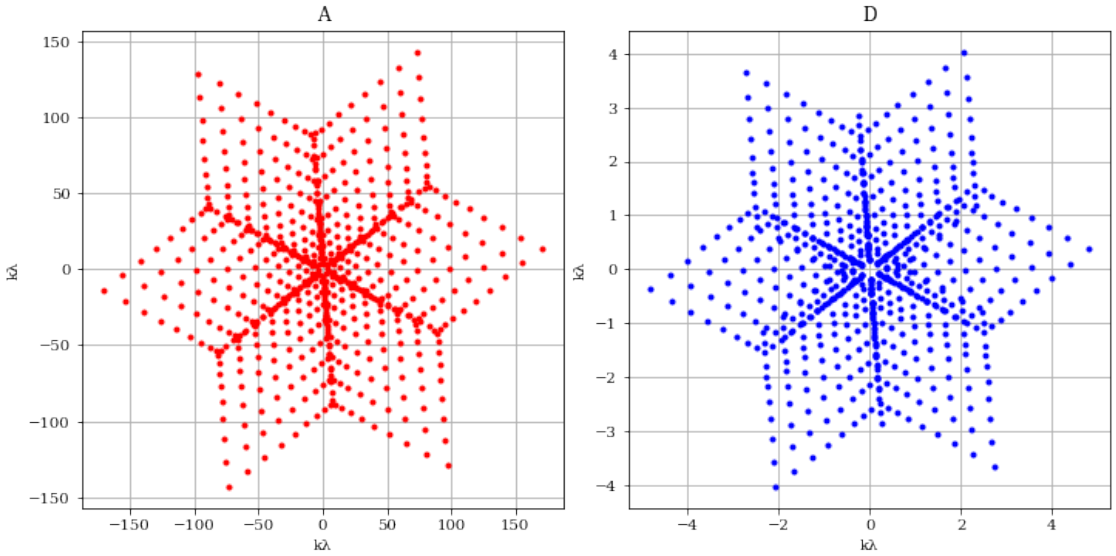


1.2 Part B

[4]:
```
def get_uv(file, dec_deg_arr, freq = 1.4e9, norm = 1000, return_all = False):

    xyz, antpos = get_antenna_pos(file, return_antpos = True)

    nant = xyz[:,0].shape[0]
    nvis = nant * (nant - 1) // 2
    uv = np.zeros([nvis,3])
    icur=0
    for i in range(nant):
        for j in range(i+1,nant):
            uv[icur,:]=antpos[i,:]-antpos[j,:]
            icur=icur+1
    uv=np.vstack([uv,-uv]) / 3e8 * freq / norm # will be in kilowavelengths by␣
 ↪default


    # Declination to observe at
    dec_arr = (90 - dec_deg_arr) * np.pi / 180

    for d in dec_arr:
        zenith = np.array([np.sin(d), 0, np.cos(d)])
        EW = np.array([0, 1, 0])
        NS = np.cross(zenith, EW)
        proj_mat = np.vstack([EW, NS])
        uv_snap=uv@proj_mat.T

    if return_all:
        return uv_snap, uv

    return uv_snap
```

[5]:
```
titles = ['A', 'D']
color = ['red', 'blue']
fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(10,5),␣
 ↪constrained_layout=True)

for i, f in enumerate(files):

    uv_snap = get_uv(f, np.array([31.5]))
    axs[i].plot(uv_snap[:,0],uv_snap[:,1], '.', color=color[i])
    axs[i].set_title(titles[i])
```

```
    axs[i].set_xlabel('k ')
    axs[i].set_ylabel('k ')
    axs[i].grid()
```

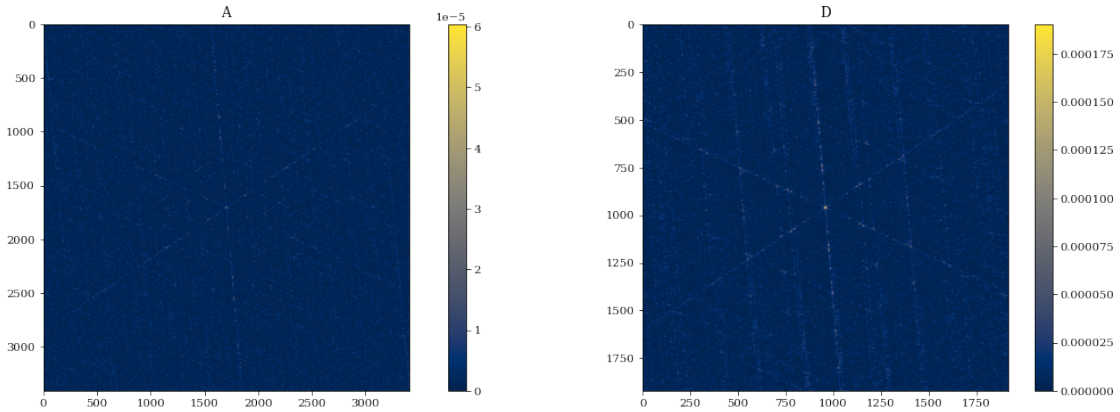


1.3 Part C

[6]:
```
def get_beam(file, du, dec_deg_arr, freq = 1.4e9):

    uv_snap, uv = get_uv(file, dec_deg_arr, freq, norm=10, return_all=True)

    pad = 4
    sz = int(np.max(np.abs(uv))/du)
    uv_mat = np.zeros([pad*2*sz, pad*2*sz])
    uv_int=np.asarray(uv_snap/du,dtype='int')

    for i in range(uv_snap.shape[0]):
        uv_mat[uv_int[i,0],uv_int[i,1]]=uv_mat[uv_int[i,0],uv_int[i,1]]+1

    return np.abs(np.fft.ifft2(uv_mat))
```

[7]:
```
titles = ['A', 'D']
color = ['red', 'blue']
du = [40, 2]
fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(15,5),␣
 ↪constrained_layout=True)
```

```
for i, f in enumerate(files):

    beam = get_beam(f, du[i], np.array([31.5]))
    im = axs[i].imshow(np.fft.fftshift(beam), cmap='cividis',␣
 ↪interpolation='nearest')
    fig.colorbar(
        im,
        ax=axs[i],
        orientation="vertical",
        label="",
    )
    axs[i].set_title(titles[i])
```



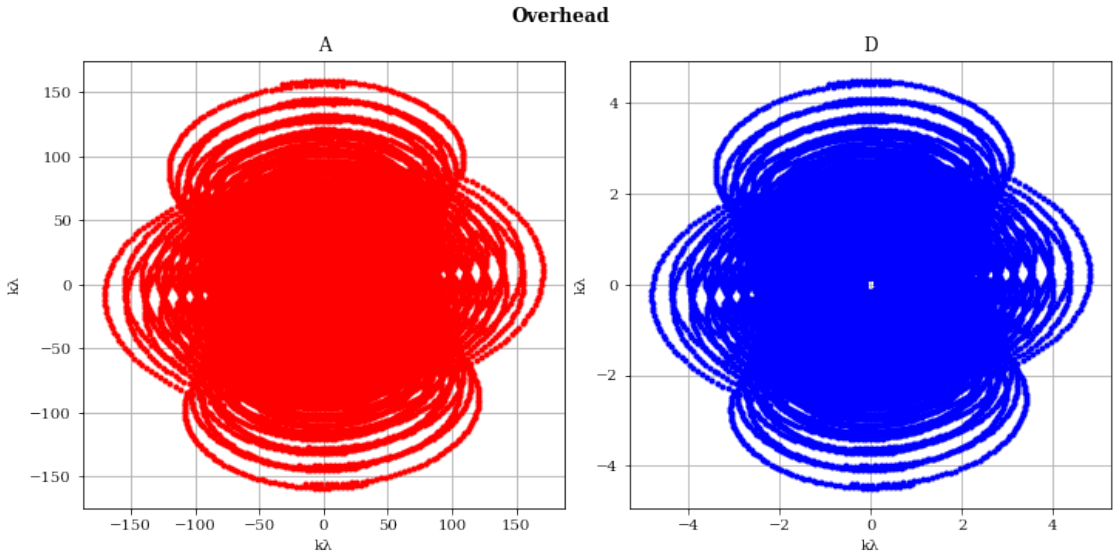
1.4 Part D

[8]:
```
def get_uv_with_time(files, source_loc, freq = 1.4e9, norm = 1000,␣
 ↪suptitle=None):

    titles = ['A', 'D']
    color = ['red', 'blue']
    fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(10,5),␣
 ↪constrained_layout=True)
    du = [40, 2]

    for k, file in enumerate(files):
        xyz, antpos = get_antenna_pos(file, return_antpos = True)

        nant = xyz[:,0].shape[0]
        nvis = nant * (nant - 1) // 2
        uv = np.zeros([nvis,3])
        icur=0
        for i in range(nant):
```

```
            for j in range(i+1,nant):
                uv[icur,:]=antpos[i,:]-antpos[j,:]
                icur=icur+1
        uv=np.vstack([uv,-uv]) / 3e8 * freq / norm # will be in kilowavelengths␣
↪by default

        # Specify time range, convert to angles
        t_range=np.linspace(-4,4,64)
        theta_range=t_range*2*np.pi/24

        for theta in theta_range:
            rot_mat=np.zeros([3,3])
            rot_mat[0,0]=np.cos(theta)
            rot_mat[1,1]=np.cos(theta)
            rot_mat[2,2]=1.0
            rot_mat[0,1]=np.sin(theta)
            rot_mat[1,0]=-np.sin(theta)
            uv_rot=uv@rot_mat

            EW = np.array([0, 1, 0])
            NS = np.cross(source_loc, EW)
            proj_mat = np.vstack([EW, NS])
            uv_snap=uv_rot@proj_mat.T

            axs[k].plot(uv_snap[:,0],uv_snap[:,1],'.', color=color[k])


        axs[k].set_title(titles[k])
        axs[k].set_xlabel('k ')
        axs[k].set_ylabel('k ')
        axs[k].grid()
        fig.suptitle(suptitle, fontweight='bold')

    return fig
```

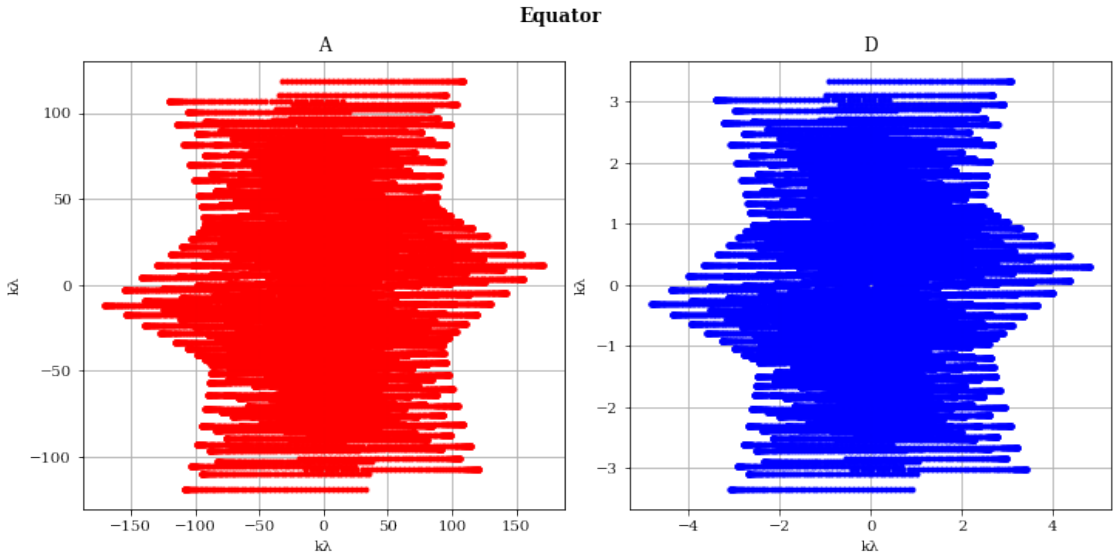
1.4.1 Source #1: Overhead

[9]:
```
source_loc = np.array([np.sin((90-34.1)*np.pi/180), 0, np.cos((90-34.1)*np.pi/
↪180)]) # located at Zenith
fig = get_uv_with_time(files, source_loc, freq = 1.4e9, norm = 1000,␣
↪suptitle='Overhead')
```



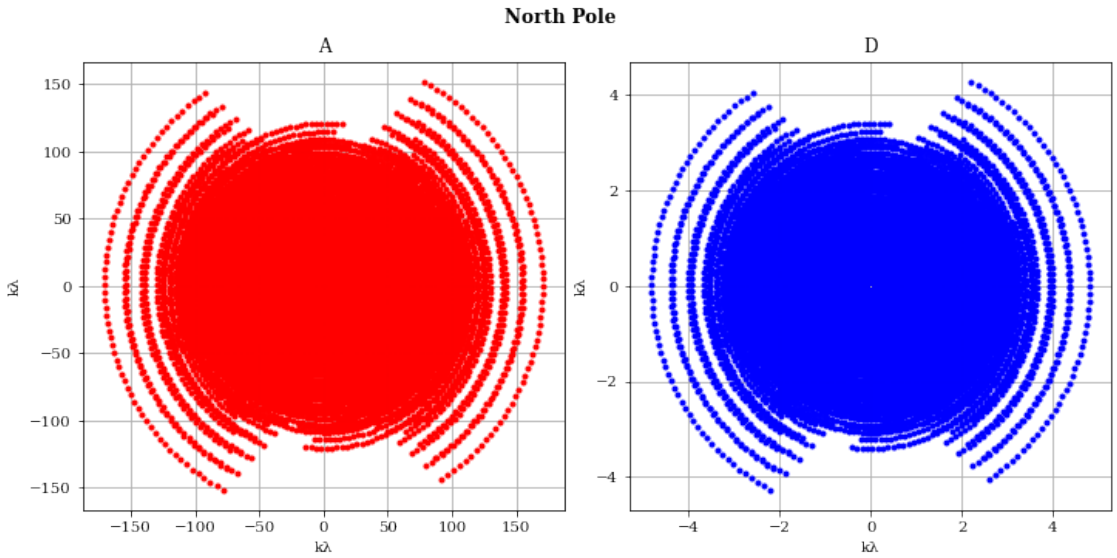
1.4.2 Source #2: At Equator

[10]:
```
source_loc = np.array([np.sin(np.pi/2), 0, np.cos(np.pi/2)]) # located at␣
↪equator
fig = get_uv_with_time(files, source_loc, freq = 1.4e9, norm = 1000,␣
↪suptitle='Equator')
```



1.4.3 Source #3: At North Pole

[11]:
```
source_loc = np.array([np.sin(0), 0, np.cos(0)]) # located at North Pole
fig = get_uv_with_time(files, source_loc, freq = 1.4e9, norm = 1000,␣
↪suptitle='North Pole')
```



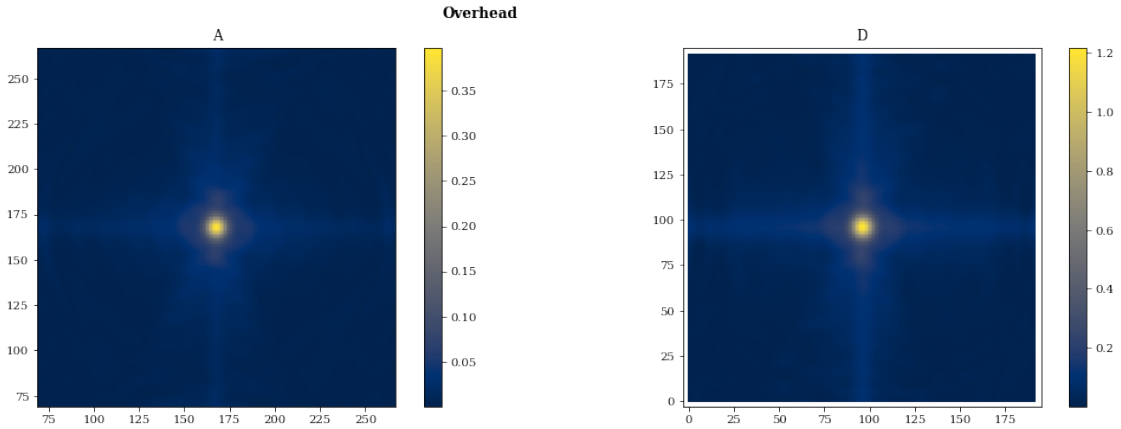
1.5 Part E

[12]:
```
def get_beam_with_time(files, source_loc, freq = 1.4e9, norm = 100,␣
↪suptitle=None):

    titles = ['A', 'D']
    color = ['red', 'blue']
    fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(15,5),␣
↪constrained_layout=True)
    du = [40, 2]

    for k, file in enumerate(files):
        xyz, antpos = get_antenna_pos(file, return_antpos = True)

        nant = xyz[:,0].shape[0]
        nvis = nant * (nant - 1) // 2
        uv = np.zeros([nvis,3])
        icur=0
        for i in range(nant):
            for j in range(i+1,nant):
                uv[icur,:]=antpos[i,:]-antpos[j,:]
                icur=icur+1
```

```
        uv=np.vstack([uv,-uv]) / 3e8 * freq / norm

        # Specify time range, convert to angles
        t_range=np.linspace(-4,4,64)
        theta_range=t_range*2*np.pi/24

        pad=4
        sz=int(np.max(np.abs(uv))/du[k])
        uv_mat=np.zeros([pad*2*sz,2*pad*sz])

        for theta in theta_range:
            rot_mat=np.zeros([3,3])
            rot_mat[0,0]=np.cos(theta)
            rot_mat[1,1]=np.cos(theta)
            rot_mat[2,2]=1.0
            rot_mat[0,1]=np.sin(theta)
            rot_mat[1,0]=-np.sin(theta)
            uv_rot=uv@rot_mat

            EW = np.array([0, 1, 0])
            NS = np.cross(source_loc, EW)
            proj_mat = np.vstack([EW, NS])
            uv_snap=uv_rot@proj_mat.T
            uv_int = np.asarray(uv_snap/du[k], dtype='int')

            for i in range(uv_snap.shape[0]):
                ␣
↪uv_mat[uv_int[i,0],uv_int[i,1]]=uv_mat[uv_int[i,0],uv_int[i,1]]+1


        beam = np.abs(np.fft.ifft2(uv_mat))
        x0=beam.shape[0]//2
        dx=99
        im = axs[k].imshow(np.fft.fftshift(beam), cmap='cividis',␣
↪interpolation='nearest')
        fig.colorbar(
            im,
            ax=axs[k],
            orientation="vertical",
            label="",
        )
        axs[k].set_xlim([x0-dx,x0+dx])
        axs[k].set_ylim([x0-dx,x0+dx])
        axs[k].set_title(titles[k])
        fig.suptitle(suptitle, fontweight='bold')

    return fig
```

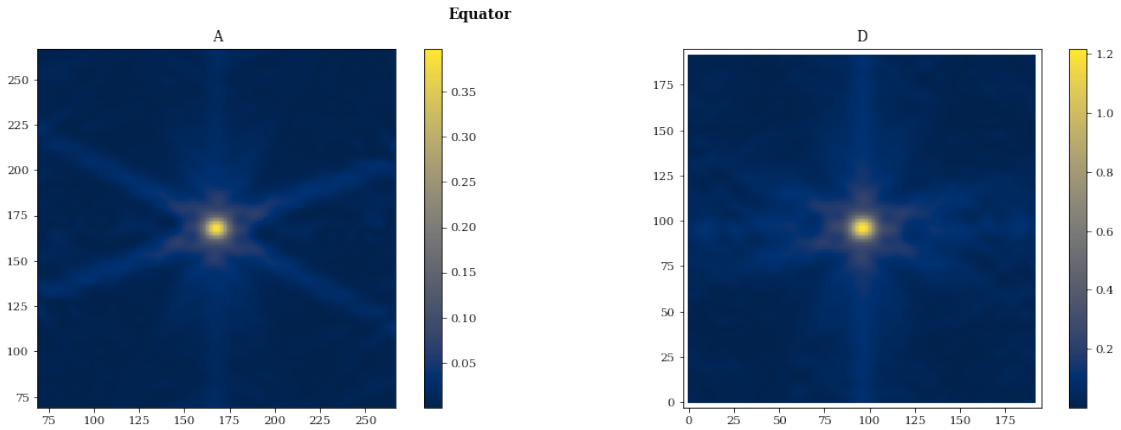
1.5.1 Source #1: Overhead

[13]:
```
source_loc = np.array([np.sin((90-34.1)*np.pi/180), 0, np.cos((90-34.1)*np.pi/
↪180)]) # located at Zenith
fig = get_beam_with_time(files, source_loc, freq = 1.4e9, suptitle='Overhead')
```



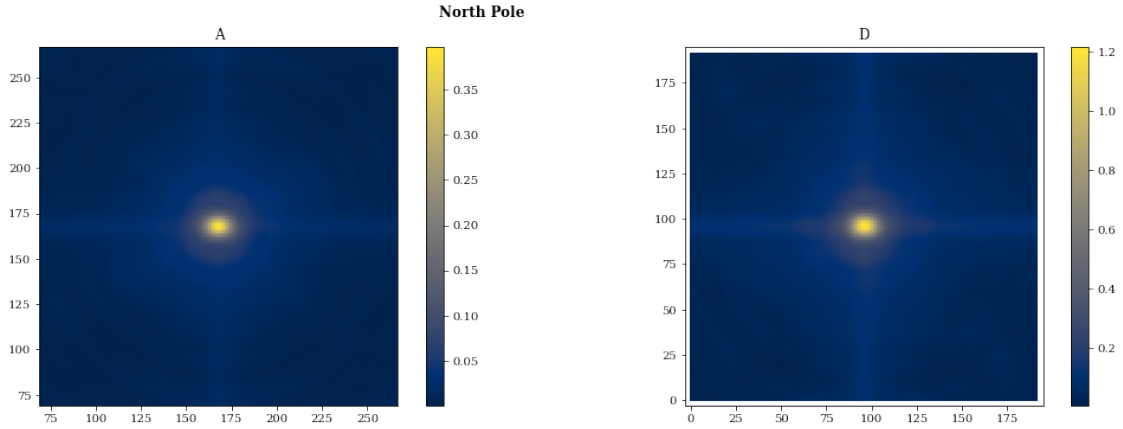
1.5.2 Source #2: At Equator

[14]:
```
source_loc = np.array([np.sin(np.pi/2), 0, np.cos(np.pi/2)]) # located at␣
↪equator
fig = get_beam_with_time(files, source_loc, freq = 1.4e9, suptitle='Equator')
```



1.5.3 Source #3: At North Pole

[15]:
```
source_loc = np.array([np.sin(0), 0, np.cos(0)]) # located at North Pole
fig = get_beam_with_time(files, source_loc, freq = 1.4e9, suptitle='North Pole')
```



2 Problem 2

2.1 Part A

The field of view of the VLA is roughly given by the width of the beam described by:

$$\theta = 1.22\frac{\lambda}{D} \tag{2}$$

Since a VLA dish has a diameter of 25 m, we can easily calculate the FOV at 1.4 and 8 GHz.

[16]:
```
D = 25 # m

freq = [1.4e9, 8e9]

for f in freq:
    beam = 1.22*3e8/(f*D) * 180 / np.pi
    print('Frequency: {0} GHz, FOV: {1} deg'.format(f/1e9, beam))
```

```
Frequency: 1.4 GHz, FOV: 0.5991501514796609 deg
Frequency: 8.0 GHz, FOV: 0.10485127650894066 deg
```

2.2 Part B and C (combined)

[17]:
```
def get_diff_of_diff(file, theta0, theta1):

    xyz, antpos = get_antenna_pos(file, return_antpos = True)

    # Build our UV matrix
```

```
    nant = xyz[:,0].shape[0]
    nvis = nant * (nant - 1) // 2
    uv = np.zeros([nvis,3])
    icur=0
    for i in range(nant):
        for j in range(i+1,nant):
            uv[icur,:]=antpos[i,:]-antpos[j,:]
            icur=icur+1

    uv_2d = uv[:,:2] # we exclude the last column for now

    # 2D

    # Source directly over head
    source_vec_2d_overhead = np.array([np.sin(theta0), 0])
    d_overhead = np.dot(uv_2d,source_vec_2d_overhead)

    # Source 30 arcmin south
    source_vec_2d_south = np.array([np.sin(theta1), 0])
    d_south = np.dot(uv_2d,source_vec_2d_south)


    path_diff = d_overhead - d_south
    rms0 = np.sqrt(np.mean(path_diff**2))

    # 3D

    # Source directly over head
    source_vec_3d_overhead = np.array([np.sin(theta0), 0, np.cos(theta0)])
    d_overhead = np.dot(xyz,source_vec_3d_overhead)

    # Source 30 arcmi south
    source_vec_3d_south = np.array([np.sin(theta1), 0, np.cos(theta1)])
    d_south = np.dot(xyz,source_vec_3d_south)

    path_diff = d_overhead - d_south
    rms1 = np.sqrt(np.mean(path_diff**2))

    print('2D RMS: {0}'.format(rms0))
    print('3D RMS: {0}'.format(rms1))
    print('Difference of difference: {0} m'.format(rms1 - rms0))

    return rms1 - rms0
```

[18]:
```
files = ['vla_a_array.txt',  'vla_d_array.txt']

for f in files:
```

```
    print(f)
    theta0 = (90 - 34.1) * np.pi / 180
    theta1 = (90 - 34.1 + 30/60) * np.pi / 180
    diff_rms = get_diff_of_diff(f, theta0, theta1)
    print()
```

```
vla_a_array.txt
2D RMS: 28.901329009409974
3D RMS: 38.70008417315197
Difference of difference: 9.798755163741998 m

vla_d_array.txt
2D RMS: 0.821408133042811
3D RMS: 1.0950779659231713
Difference of difference: 0.2736698328803603 m
```

2.3 Part D

The difference in phase difference is

$$\Delta\phi = 2\pi\Delta x/\lambda, \tag{3}$$

where Δx is the difference in path difference (e.g. path difference from beam centre - path difference at FWHM).

[19]:
```
def get_diff_path_diff(file, theta0, theta1):

    xyz, antpos = get_antenna_pos(file, return_antpos = True)

    source_vec_3d = np.array([np.sin(theta0), 0, np.cos(theta0)])
    path_diff0 = np.dot(xyz,source_vec_3d)

    source_vec_3d = np.array([np.sin(theta1), 0, np.cos(theta1)])
    path_diff1 = np.dot(xyz,source_vec_3d)

    return np.abs(path_diff1 - path_diff0)
```

2.3.1 #1: Pointing Centre is Directly Overhead

2.3.2 f = 1.4 GHz

[20]:
```
freq = 1.4e9
lamda = 3e8/freq
print('Wavelength:', lamda , ' m')
FWHM = 3e8/(freq*D) * 180 / np.pi * 60
print('Beam FWHM is located {0} arcsec from beam centre'.format(FWHM))
```

```
Wavelength: 0.21428571428571427  m
Beam FWHM is located 29.466400892442337 arcsec from beam centre
```

For the A array, we'll have to worry about the w term since the difference of differences is much larger than one wavelength. For the D array, the difference of difference is roughly the same size, so we should be fine to use either one.

[21]:
```
for f in files:
    theta0 = (90 - 34.1) * np.pi / 180  # Beam centre
    theta1 = (90 - 34.1 + FWHM/60) * np.pi / 180  # 1 FWHM from beam centre
    path_diff = get_diff_path_diff(f,theta0, theta1)
    phase_diff = 2*np.pi* path_diff / lamda
    print(f)

    print('RMS of difference of phase difference:', np.sqrt(np.
 ↪mean(phase_diff**2)), 'rad')
    print()
```

```
vla_a_array.txt
RMS of difference of phase difference: 1114.6917960133483 rad

vla_d_array.txt
RMS of difference of phase difference: 31.54188636528481 rad
```

2.3.3 f = 8 GHz

[22]:
```
freq = 8e9
lamda = 3e8/freq
print('Wavelength:', lamda , ' m')
FWHM = 3e8/(freq*D) * 180 / np.pi * 60
print('Beam FWHM is located {0} arcsec from beam centre'.format(FWHM))
```

```
Wavelength: 0.0375  m
Beam FWHM is located 5.156620156177409 arcsec from beam centre
```

We apply a similar approach, except we must not ignore w term since in both cases the diff of diff is greater than one wavelength.

[23]:
```
for f in files:
    theta0 = (90 - 34.1) * np.pi / 180  # Beam centre
    theta1 = (90 - 34.1 + FWHM/60) * np.pi / 180  # 1 FWHM from beam centre
    path_diff = get_diff_path_diff(f,theta0, theta1)
    phase_diff = 2*np.pi* path_diff / lamda
    print(f)

    print('RMS of difference of phase difference:', np.sqrt(np.
 ↪mean(phase_diff**2)), 'rad')
    print()
```

```
vla_a_array.txt
RMS of difference of phase difference: 1120.5762475661375 rad
```

```
vla_d_array.txt
RMS of difference of phase difference: 31.70749265149746 rad
```

2.3.4 #2: Pointing Centre is at the Equator

We repeat the same process, but in this case our θ will change to roughly 90 $\pm$ beam FWHM.

2.3.5 f = 1.4 GHz

[24]:
```
freq = 1.4e9
lamda = 3e8/freq
print('Wavelength:', lamda , ' m')
FWHM = 3e8/(freq*D) * 180 / np.pi * 60
print('Beam FWHM is located {0} arcsec from beam centre\n'.format(FWHM))

for f in files:
    theta0 = (90) * np.pi / 180  # Beam centre
    theta1 = (90 + FWHM/60) * np.pi / 180  # 1 FWHM from beam centre
    path_diff = get_diff_path_diff(f,theta0, theta1)
    phase_diff = 2*np.pi* path_diff / lamda
    print(f)

    print('RMS of difference of phase difference:', np.sqrt(np.
 ↪mean(phase_diff**2)), 'rad')
    print()
```

```
Wavelength: 0.21428571428571427  m
Beam FWHM is located 29.466400892442337 arcsec from beam centre

vla_a_array.txt
RMS of difference of phase difference: 11.564946888222893 rad

vla_d_array.txt
RMS of difference of phase difference: 0.24416099232612348 rad
```

2.3.6 f = 8 GHz

[25]:
```
freq = 8e9
lamda = 3e8/freq
print('Wavelength:', lamda , ' m')
FWHM = 3e8/(freq*D) * 180 / np.pi * 60
print('Beam FWHM is located {0} arcsec from beam centre\n'.format(FWHM))

for f in files:
    theta0 = (90) * np.pi / 180  # Beam centre
```

```
    theta1 = (90 + FWHM/60) * np.pi / 180  # 1 FWHM from beam centre
    path_diff = get_diff_path_diff(f,theta0, theta1)
    phase_diff = 2*np.pi* path_diff / lamda
    print(f)

    print('RMS of difference of phase difference:', np.sqrt(np.
↪mean(phase_diff**2)), 'rad')
    print()
```

```
Wavelength: 0.0375  m
Beam FWHM is located 5.156620156177409 arcsec from beam centre

vla_a_array.txt
RMS of difference of phase difference: 5.773950473499296 rad

vla_d_array.txt
RMS of difference of phase difference: 0.1907601495716688 rad
```

3 Problem 3

See attached PDF for analytic explanation.

[26]:
```
def get_frequency_shift(file, theta):

    xyz, antpos = get_antenna_pos(file, return_antpos = True)

    source_vec_3d = np.array([np.sin(theta), 0, np.cos(theta)])
    path_diff = np.dot(xyz,source_vec_3d)

    freq_shift = 1 / ((2*np.pi*path_diff/3e8))

    return freq_shift
```

3.0.1 1. Source directly overhead

[27]:
```
for f in files:
    theta = (90 - 34.1) * np.pi / 180
    freq_shift = get_frequency_shift(f, theta)


    print(f)
    print('Required Frequency Shift: ', np.sqrt(np.mean(freq_shift**2)), ' Hz')
    print()
```

```
vla_a_array.txt
Required Frequency Shift:  313217.72429159225  Hz
```

```
vla_d_array.txt
Required Frequency Shift:  19455857.396180708  Hz
```

3.0.2 2. Source 1 FWHM to the south (~30 arcmin)

[28]:
```
for f in files:
    theta = (90 - 34.1 + 0.5) * np.pi / 180
    freq_shift = get_frequency_shift(f, theta)


    print(f)
    print('Required Frequency Shift: ', np.sqrt(np.mean(freq_shift**2)), ' Hz')
    print()
```

```
vla_a_array.txt
Required Frequency Shift:  311379.15168074565  Hz

vla_d_array.txt
Required Frequency Shift:  19682011.46327809  Hz
```

4 Problem 4

See attached PDF for analytic write-up.

4.1 Part b

[29]:
```
def S_confusion(nu, D):
    c=3e8
    return (4.06e-7 * nu ** 2 * D ** 2 / (np.pi * c ** 2)) ** (-2/3)
```

[30]:
```
# GBT
nu = 1.4e9
D = 100 # m
sigma_c_gbt = S_confusion(nu, D)
print('GBT Confusion Limit: S >= {0} mJy'.format(S_confusion(nu, D)))
```

```
GBT Confusion Limit: S >= 10.807871805896001 mJy
```

[31]:
```
# FAST
nu = 1.4e9
D = 300 # m
sigma_c_fast = S_confusion(nu, D)
print('FAST Confusion Limit: S >= {0} mJy'.format(S_confusion(nu, D)))
```

```
FAST Confusion Limit: S >= 2.497919922784783 mJy
```

[32]:
```
# A Array
nu = 1.4e9
D = 30e3 # m
sigma_c_vlaa = S_confusion(nu, D)
print('VLA A Array Confusion Limit: S >= {0} mJy'.format(S_confusion(nu, D)))
```

```
VLA A Array Confusion Limit: S >= 0.005381605334569302 mJy
```

[33]:
```
# D Array
nu = 1.4e9
D = 1e3 # m
sigma_c_vlad = S_confusion(nu, D)
print('VLA D Array Confusion Limit: S >= {0} mJy'.format(S_confusion(nu, D)))
```

```
VLA D Array Confusion Limit: S >= 0.5016569708936527 mJy
```

4.2 Part c

[34]:
```
def get_integration_time(A_e, sigma_c, bw = 500e6):
    return 4.72e15 / (A_e**2 * bw * sigma_c**2)
```

[35]:
```
# GBT
D = 100 # m
A_e = 0.7 * np.pi * (D/2)**2
tau_gbt = get_integration_time(A_e, sigma_c_gbt)
print('Integration Time:  = {0} s'.format(tau_gbt))
```

```
Integration Time:  = 0.002673719121192744 s
```

[36]:
```
# FAST
D = 500 # m
A_e = 0.7 * np.pi * (D/2)**2
tau_fast = get_integration_time(A_e, sigma_c_fast)
print('Integration Time:  = {0} s'.format(tau_fast))
```

```
Integration Time:  = 8.008646243583715e-05 s
```

[37]:
```
# VLA A
A_e = 27 * np.pi * 12.5 ** 2
tau_vlaa = get_integration_time(A_e, sigma_c_vlaa)
print('Integration Time:  = {0} s'.format(tau_vlaa))
```

```
Integration Time:  = 1855.5855343344713 s
```

[38]:
```
# VLA D
A_e = 27 * np.pi * 12.5 ** 2
tau_vlad = get_integration_time(A_e, sigma_c_vlad)
print('Integration Time:  = {0} s'.format(tau_vlad))
```

```
Integration Time:  = 0.21354576629117897 s
```

4.3 Part D

[39]:
```
def scale_s(S_0, nu_1 = 8, nu_0 = 1.4):

    return S_0 * (nu_1 / nu_0) ** -0.8
```

[40]:
```
# GBT
D = 100 # m
A_e = 0.7 * np.pi * (D/2)**2
# Scale sigma_c
sigma_c_gbt_1 = scale_s(sigma_c_gbt)
print('Confusion Limit: S >= {0} mJy'.format(sigma_c_gbt_1))
tau_gbt = get_integration_time(A_e, sigma_c_gbt_1)

print('Integration Time:  = {0} s'.format(tau_gbt))
```

```
Confusion Limit: S >= 2.680221111745638 mJy
Integration Time:  = 0.04347653296966532 s
```

[41]:
```
# FAST
D = 100 # m
A_e = 0.7 * np.pi * (D/2)**2
# Scale sigma_c
sigma_c_fast_1 = scale_s(sigma_c_fast)
print('Confusion Limit: S >= {0} mJy'.format(sigma_c_fast_1))
tau_fast = get_integration_time(A_e, sigma_c_fast_1)

print('Integration Time:  = {0} s'.format(tau_fast))
```

```
Confusion Limit: S >= 0.6194538418604769 mJy
Integration Time:  = 0.813913496213252 s
```

[42]:
```
# VLA A
A_e = 27 * np.pi * 12.5 ** 2
# Scale sigma_c
sigma_c_vlaa_1 = scale_s(sigma_c_vlaa)
print('Confusion Limit: S >= {0} mJy'.format(sigma_c_vlaa_1))
tau_vlaa = get_integration_time(A_e, sigma_c_vlaa_1)

print('Integration Time:  = {0} s'.format(tau_vlaa))
```

```
Confusion Limit: S >= 0.0013345728457777367 mJy
Integration Time:  = 30173.111686293305 s
```

[43]:
```
# VLA D
A_e = 27 * np.pi * 12.5 ** 2
```

```
# Scale sigma_c
sigma_c_vlad_1 = scale_s(sigma_c_vlad)
print('Confusion Limit: S >= {0} mJy'.format(sigma_c_vlad_1))
tau_vlad = get_integration_time(A_e, sigma_c_vlad_1)

print('Integration Time:  = {0} s'.format(tau_vlad))
```

```
Confusion Limit: S >= 0.12440484383891784 mJy
Integration Time:  = 3.4724027199047 s
```

[ ]: