Wavefront Reconstruction v1

July 25, 2018

```
In [1]: # Setup

        from hcipy import *
        import numpy as np
        from matplotlib import pyplot as plt

        N = 128
        D = 9.96
        sps = 40
        aperture = circular_aperture(D)
        pupil_grid = make_pupil_grid(N, D)
        wf = Wavefront(aperture(pupil_grid))

        aberrated = wf.copy()
        amplitude = 0.3
        spatial_frequency = 5
        aberrated.electric_field *= np.exp(1j * amplitude * np.sin(2*np.pi * pupil_grid.x / D *
```

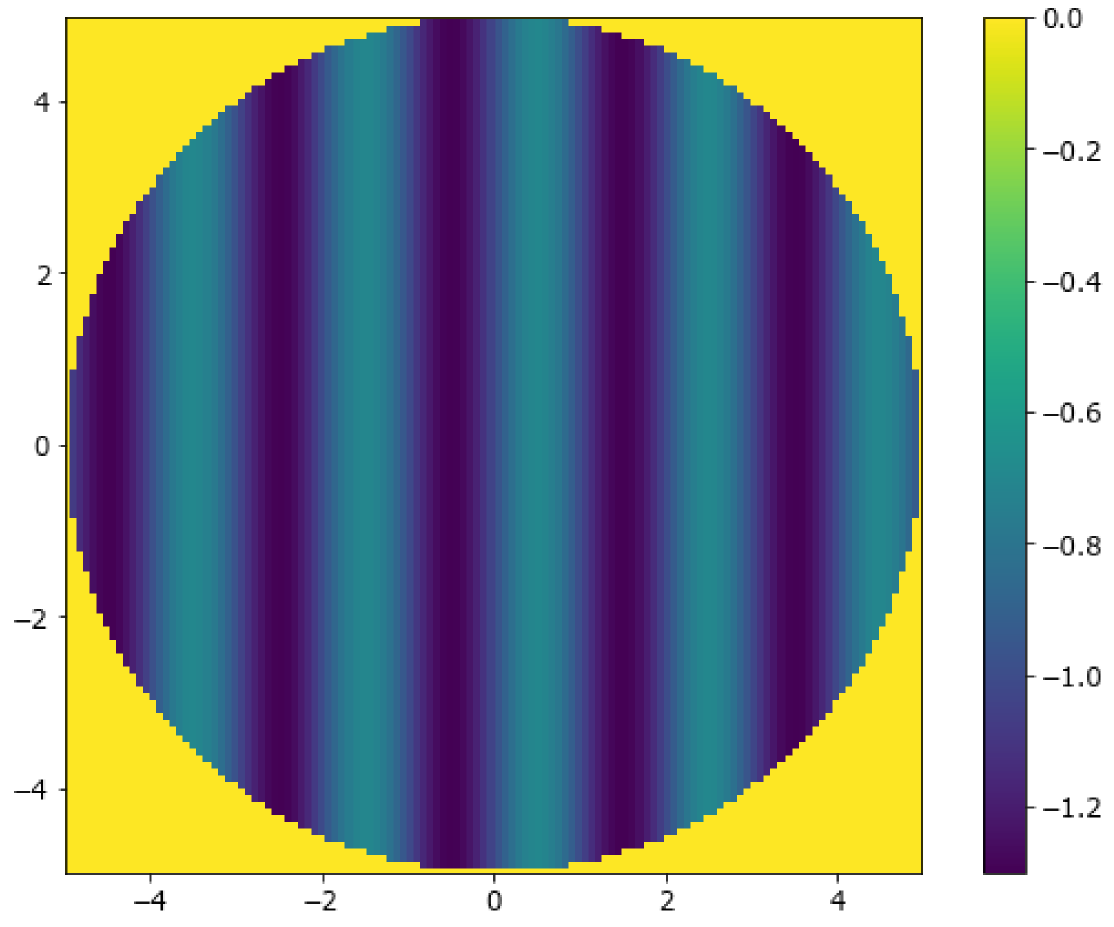
```
In [2]: # Making an aberration basis, should be size 12644. (Probably a better way to do this wi

        aberration_mode_basis = []
        for i in range(N):
            for j in range(N):
                wf = Wavefront(aperture(pupil_grid))
                wf.electric_field.shape = (N, N)
                l = wf.electric_field.tolist()
                if np.real(l[i][j]) > 0:
                    l[i][j] = 0
                    wf.electric_field = Field(np.asarray(l).ravel(), wf.grid)
                    aberration_mode_basis.append(wf)
        print(len(aberration_mode_basis))
```

```
12644
```

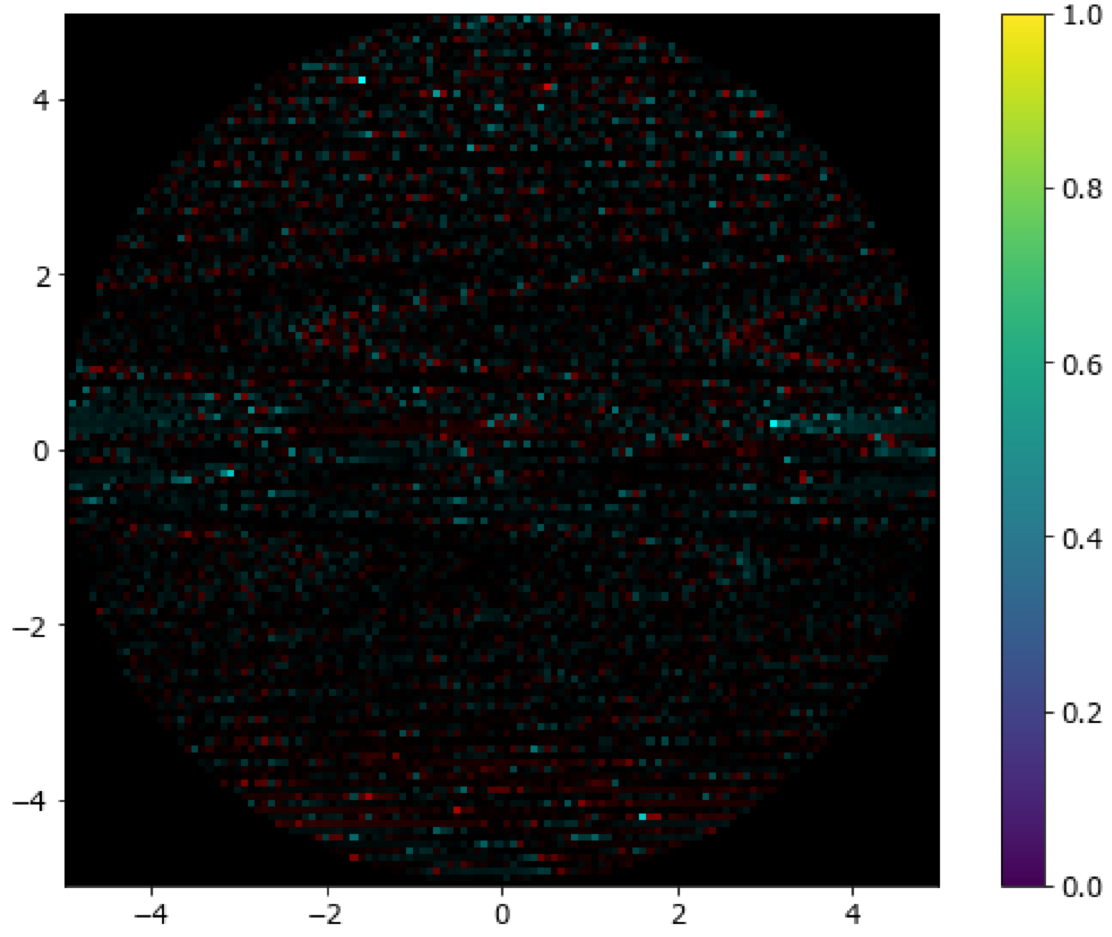
```
In [3]: # Propagating to pyramid. (Longest step, about 8-9 minutes)
        keck_pyramid = PyramidWavefrontSensorOptics(pupil_grid, pupil_separation=65/39.3, num_pu
        pyramid_output_basis = [keck_pyramid.forward(x) for x in aberration_mode_basis]
```

```
In [5]: def get_sub_images(intensity):
            buffer = 0
            D_grid = 3.6e-3
            pyramid_grid = make_pupil_grid(N, D_grid)
            images = Field(np.asarray(intensity).ravel(), pyramid_grid)
            images.shape = (107, 107)
            image = images

            sub_images = [image[66:106, 66:106],
                          image[66:106, 0:40],
                          image[0:40, 0:40],
                          image[0:40, 66:106]]
            subimage_grid = make_pupil_grid(sps, D_grid * sps / N)
            for count, img in enumerate(sub_images):
                img = img.ravel()
                img.grid = subimage_grid
                sub_images[count] = img
            return sub_images

        sub_images_basis = [get_sub_images(x.intensity) for x in pyramid_output_basis]

In [6]: def estimate(EstimatorObject, images_list):
            # Restored the numbering convention to what it was originally because this version o
            I_a = images_list[0]
            I_b = images_list[1]
            I_c = images_list[2]
            I_d = images_list[3]
            norm = I_a + I_b + I_c + I_d
            I_x = (I_a + I_b - I_c - I_d) / norm
            I_y = (I_a - I_b - I_c + I_d) / norm
            return [I_x.ravel(), I_y.ravel()]

        keck_pyramid_estimator = PyramidWavefrontSensorEstimator(aperture, make_pupil_grid(sps*2
        estimated_basis = [estimate(keck_pyramid_estimator, x) for x in sub_images_basis]

In [8]: # All the above steps, to generate flat wavefront slopes
        flat_mode = Wavefront(aperture(pupil_grid))
        flat_pyramid_output = keck_pyramid.forward(flat_mode)
        flat_x, flat_y = estimate(keck_pyramid_estimator, get_sub_images(flat_pyramid_output.int

In [10]: estimated_basis = np.asarray([[x - flat_x, y - flat_y] for x, y in estimated_basis])
         # In regular Python, write to file here, and read in a separate file. Here, continuing

In [11]: M_inv = field_inverse_tikhonov(Field(estimated_basis, make_pupil_grid(sps, D)), 1e-15)
         M_inv = M_inv.copy()
         M_inv.shape = (12644, 3200)

In [13]: # Quickly do all the above without explaining anything
         get_pyramid_output = lambda wf: np.asarray(estimate(keck_pyramid_estimator, get_sub_ima
In [16]: # make
         aberrated_images = get_pyramid_output(aberrated)
         flat_images = get_pyramid_output(Wavefront(aperture(pupil_grid)))
         aberrated_res = aberrated_images - flat_images
         reconstructed = M_inv.dot(aberrated_res).tolist()
         project_onto = Wavefront(aperture(pupil_grid)).electric_field
         project_onto.shape = (N, N)
         project_onto = project_onto.tolist()

         count, i, j = 0, 0, 0
         while count < 12644:
             if np.real(project_onto[i][j]) > 0:
                 project_onto[i][j] = reconstructed[count]
                 count += 1
             j += 1
             if j == N - 1:
                 j = 0
                 i += 1
```

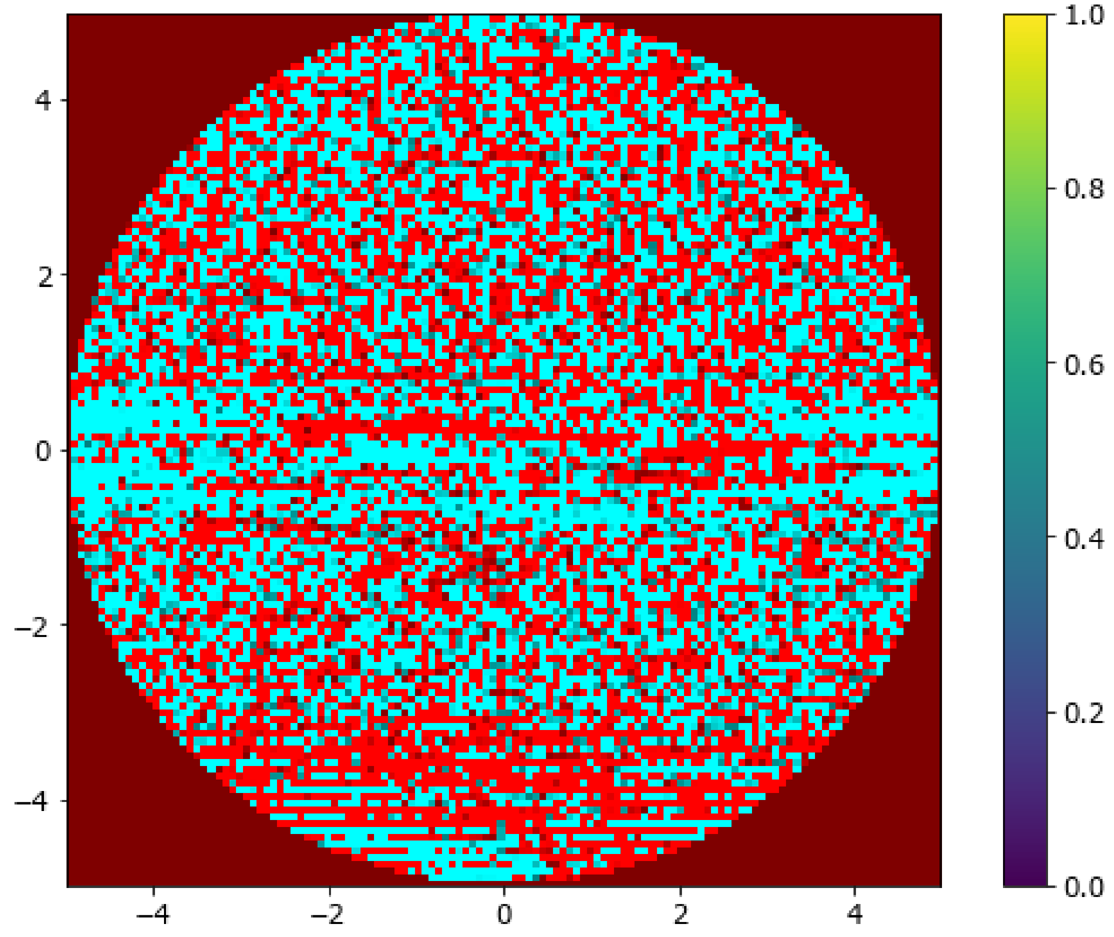
```
In [30]: imshow_field(aberrated.phase - Wavefront(aperture(pupil_grid)).intensity, pupil_grid)
         plt.colorbar()
         plt.show()
```



```
In [32]: imshow_field(np.asarray(project_onto).ravel() * aperture(pupil_grid), pupil_grid)
         plt.colorbar()
         plt.show()
```



```
In [33]: imshow_field(np.asarray(project_onto).ravel() * aperture(pupil_grid), pupil_grid, vmin=
         plt.colorbar()
         plt.show()
```


4
2
0
−2
−4
−4
−2
0
2
4
1.0
0.8
0.6
0.4
0.2
0.0