```
import math
import zernike
import matplotlib.pyplot as plt
from matplotlib import cm
import numpy as np

#Cartiesan to polar
def xytort(x, y):
    rho = math.sqrt((x**2 + y**2))
    theta = math.atan2(y, x)
    return rho, theta

#Polar to cartiesan
def rttoxy(rho, theta):
    x = int(math.round(rho*math.cos(theta)))
    y = int(math.round(rho*math.sin(theta)))
    return x,y

#Take the proved wavefront object and returns the pre and post focal images
def computeImages(wavefront, useDiags = False, givederivs = False, radius = 10
0):
    dx, dy = np.gradient(wavefront.image)[1], np.gradient(wavefront.image)[0]
    d2x, d2y = np.gradient(dx)[1], np.gradient(dy)[0]

    laplacian = d2x + d2y

    dn = np.cos(wavefront.thetacoords)*dx + np.sin(wavefront.thetacoords)*dy
    dnangles = []
    for i in np.arange(0, 2*np.pi, 0.01):
        x = int(np.cos(i)*radius)
        y = int(np.sin(i)*radius)
        dnangles.append((i, dn[y][x]))

    mask1 = np.zeros(wavefront.image.shape)
    mask2 = np.zeros(wavefront.image.shape)
    plt.imshow(dn)

    points = [(np.cos(theta), np.sin(theta)) for theta in np.arange(0, 2*np.pi
, 0.01)]
    points = [(int(x*radius/2 + 100), int(y*radius/2 + 100)) for x,y in points
]

    newpoints = []

    for x, y in points:
        theta = np.arctan2(y-100, x-100)
        mag = dn[y][x]
        dx = np.cos(theta)*mag*1000
        dy = np.sin(theta)*mag*1000
        newpoints.append((x+dx, y+dy))

    plt.plot([p[0] for p in points], [p[1] for p in points])
    plt.plot([p[0] for p in points], [p[1]-19 for p in points], color = 'green
```

```
')
    plt.plot([p[0] for p in newpoints], [p[1] for p in newpoints], color = 're
d')
    plt.show()

    for y, tr in enumerate(wavefront.thetacoords):
        for x, theta in enumerate(tr):
            dnatp = min(dnangles, key=lambda tup: np.abs(tup[0] - theta))[1]
            if wavefront.rhocoords[y][x] < dnatp*10 + radius/wavefront.x_res:
                mask1[y][x] = 1
            else:
                mask1[y][x] = 0

            if (wavefront.rhocoords[y][x] < -dnatp*10 + radius/wavefront.x_res
).astype(np.int):
                mask2[y][x] = 1
            else:
                mask2[y][x] = 0

            #dn = np.array([[min(dnangles, key = lambda tup: np.abs(tup[0] - t
heta))[1] for theta in tr] for tr in wavefront.thetacoords])
    #print(dn)

    offsetbrightness = 0.5*np.ones(laplacian.shape)
    mask1 = (wavefront.rhocoords < dn*10 + radius/wavefront.x_res).astype(np.i
nt)
    mask2 = (wavefront.rhocoords < -dn*10 + radius/wavefront.x_res).astype(np.
int)

    img1 = offsetbrightness + laplacian
    img2 = offsetbrightness - laplacian

    img1 *= mask1
    img2 *= mask2

    return img1, img2

    #Second derivative test functionality
    """

    secder = [[0 for i in range(x_res)] for j in range(y_res)]
    firstder = [[0 for i in range(x_res)] for j in range(y_res)]
    sdx = [[0 for i in range(x_res)] for j in range(y_res)]
    sdy = [[0 for i in range(x_res)] for j in range(y_res)]
    #note I am not implementing the first derivative stuff yet!
    for y in range(1, y_res - 1):
        for x in range(1, x_res - 1):
            if xytort(X[y][x], Y[y][x])[0] > 0.97:
                secder[y][x] = 0
                continue
            xdelt = wavefront[y][x+1] + wavefront[y][x-1] - 2*wavefront[y][x]
            ydelt = wavefront[y+1][x] + wavefront[y-1][x] - 2*wavefront[y][x]
            lap = xdelt + ydelt
```

```
            secder[y][x] = lap
            firstder[y][x] = wavefront[y][x+1] - wavefront[y][x] + wavefront[y
+1][x] - wavefront[y][x]
            sdx[y][x] = xdelt
            sdy[y][x] = ydelt
            if useDiags:
                cross1 = wavefront[y+1][x+1] + wavefront[y-1][x-1] - 2*wavefro
nt[y][x]
                cross2 = wavefront[y+1][x-1] + wavefront[y-1][x+1] - 2*wavefro
nt[y][x]
                secder[y][x] += cross1 + cross2
    if givederivs:
        return sdx, sdy


    img1 = [[d for d in dr] for dr in secder]
    img2 = [[-d for d in dr] for dr in secder]

    return img1, img2
    """

if __name__ == "__main__":
    coeff = [0,0,-1,0,0,0]
    wavefront = zernike.wavefront(coeff, 200, 200, 100)
    im1, im2 = computeImages(wavefront, radius = 100)

    plt.subplot(1, 3, 1)
    plt.imshow(wavefront.image, cmap=cm.coolwarm)
    plt.subplot(1, 3, 2)
    print(im1[0][0], im1[100][100])
    plt.imshow(1-im1, cmap=cm.Greys, vmin = 0, vmax = 1)
    plt.subplot(1, 3, 3)
    plt.imshow(1-im2, cmap=cm.Greys, vmin = 0, vmax = 1)
    plt.show()
```