```
import numpy as np

from  analyze_pore_samples.polar_cartesian_convert import linear_polar,
polar_linear, map_pixel

import os

import time as time
from matplotlib import pyplot as plt
def calc_polar(boundary, xs, ys, zs, voxelsize = 1, imstack= None):
    # convert 2-D image to polar coordinates
    profile_2d= []
    for sample in range(boundary.shape[2]):
        center = boundary[:,:, sample].shape[1]//2
        polar, rs, ts,o, r, out_h, out_w  = linear_polar(boundary[:,:,
sample], verbose = 1)
        profile_2d.append(np.argmax(polar, axis = 0))

    profile_2d = np.array(profile_2d)
    polar= linear_polar(boundary[:,:, sample])
    radii = []
    angles = []
    for idx, x in enumerate(xs):
        z = int(zs[idx]/voxelsize)
        r_index, theta_index, theta = map_pixel(int(x/voxelsize),int(ys[idx]
/voxelsize),boundary[:,:,z], o = o, r =r, out_h = out_h, out_w = out_w, debug=
False )
        radius = profile_2d[z, theta_index] - r_index
        radii.append(radius)
        angles.append(theta)

    return radii, angles



def topolar(img, order=1):
    """
    Transform img to its polar coordinate representation.

    order: int, default 1
        Specify the spline interpolation order.
        High orders may be slow for large images.
    """
    # max_radius is the length of the diagonal
    # from a corner to the mid-point of img.
    max_radius = 0.5*np.linalg.norm( img.shape )

    def transform(coords):
        # Put coord[1] in the interval, [-pi, pi]
        theta = 2*np.pi*coords[1] / (img.shape[1] - 1.)

        # Then map it to the interval [0, max_radius].
        #radius = float(img.shape[0]-coords[0]) / img.shape[0] * max_radius
        radius = max_radius * coords[0] / img.shape[0]
```

```
51
52          i = 0.5*img.shape[0] - radius*np.sin(theta)
53          j = radius*np.cos(theta) + 0.5*img.shape[1]
54          return i,j
55
56      polar = geometric_transform(img, transform, order=order)
57
58      rads = max_radius * np.linspace(0,1,img.shape[0])
59      angs = np.linspace(0, 2*np.pi, img.shape[1])
60
61      return polar, (rads, angs)
62
63
64
65  def unit_vector(vector):
66      """ Returns the unit vector of the vector.  """
67      return vector / np.linalg.norm(vector)
68
69  def angle_between(v1, v2):
70      """ Returns the angle in radians between vectors 'v1' and 'v2'::
71
72              >>> angle_between((1, 0, 0), (0, 1, 0))
73              1.5707963267948966
74              >>> angle_between((1, 0, 0), (1, 0, 0))
75              0.0
76              >>> angle_between((1, 0, 0), (-1, 0, 0))
77              3.141592653589793
78      """
79      v1_u = unit_vector(v1)
80      v2_u = unit_vector(v2)
81      return np.arccos(np.clip(np.dot(v1_u, v2_u), -1.0, 1.0))
82
83  def nan_helper(y):
84      """Helper to handle indices and logical indices of NaNs.
85
86      Input:
87          - y, 1d numpy array with possible NaNs
88      Output:
89          - nans, logical indices of NaNs
90          - index, a function, with signature indices= index(logical_indices),
91            to convert logical indices of NaNs to 'equivalent' indices
92      Example:
93          >>> # linear interpolation of NaNs
94          >>> nans, x= nan_helper(y)
95          >>> y[nans]= np.interp(x(nans), x(~nans), y[~nans])
96      """
97
98      return np.isnan(y), lambda z: z.nonzero()[0]
99
100
```