```
import numpy  as np
from matplotlib import pyplot as plt
import pickle
import pandas
import numpy as np
import matplotlib.pyplot as plt
from skimage import measure
import imageio
import skimage
from scipy import spatial
import time
import h5py
import os
import numpy as np
import sys
from mpl_toolkits.mplot3d import Axes3D
from skimage import measure, transform, segmentation, morphology
from pylab import gca
from scipy.ndimage.interpolation import geometric_transform
import numpy as np
import torch
import os
from  reconstruction.polar_cartesian_convert import linear_polar,
polar_linear, map_pixel, calc_theta, unmap_pixel
from skimage.transform import resize
from mpl_toolkits.axes_grid1 import make_axes_locatable
from scipy.signal import savgol_filter
import numpy as np
import scipy.ndimage as ndimage


import pandas as pd

import numpy as np
import matplotlib.pyplot as plt
import os

import numpy as np
def unit_vector(vector):
    """ Returns the unit vector of the vector.  """
    return vector / np.linalg.norm(vector)


def read_file(fname):
    f = h5py.File(fname, 'r')
    return f['data']
def angle_between(v1, v2):
    """ Returns the angle in radians between vectors 'v1' and 'v2'::

            >>> angle_between((1, 0, 0), (0, 1, 0))
            1.5707963267948966
            >>> angle_between((1, 0, 0), (1, 0, 0))
            0.0
            >>> angle_between((1, 0, 0), (-1, 0, 0))
```

```
54              3.141592653589793
55      """
56      v1_u = unit_vector(v1)
57      v2_u = unit_vector(v2)
58      return np.arccos(np.clip(np.dot(v1_u, v2_u), -1.0, 1.0))
59
60
61  def set_axes_equal(ax):
62      '''Make axes of 3D plot have equal scale so that spheres appear as
    spheres,
63      cubes as cubes, etc..  This is one possible solution to Matplotlib's
64      ax.set_aspect('equal') and ax.axis('equal') not working for 3D.
65
66      Input
67        ax: a matplotlib axis, e.g., as output from plt.gca().
68      '''
69
70      x_limits = ax.get_xlim3d()
71      y_limits = ax.get_ylim3d()
72      z_limits = ax.get_zlim3d()
73
74      x_range = abs(x_limits[1] - x_limits[0])
75      x_middle = np.mean(x_limits)
76      y_range = abs(y_limits[1] - y_limits[0])
77      y_middle = np.mean(y_limits)
78      z_range = abs(z_limits[1] - z_limits[0])
79      z_middle = np.mean(z_limits)
80
81      plot_radius = 0.5*max([x_range, y_range, z_range])
82
83      ax.set_xlim3d([x_middle - plot_radius, x_middle + plot_radius])
84      ax.set_ylim3d([y_middle - plot_radius, y_middle + plot_radius])
85      ax.set_zlim3d([z_middle - plot_radius, z_middle + plot_radius])
86
87
88
89
90  '''
91  Make frame thicker, make tick pointing inside, make tick thicker
92  default frame width is 2, default tick width is 1.5
93  '''
94  def frame_tick(frame_width = 2, tick_width = 1.5):
95      ax = gca()
96      for axis in ['top','bottom','left','right']:
97          ax.spines[axis].set_linewidth(frame_width)
98      plt.tick_params(direction = 'in',
99                      width = tick_width)
100
101 '''
102 legend:
103 default location : upper left
104 default fontsize: 8
105 Frame is always off
106 '''
107 def legend(location = 'upper left', fontsize = 8):
```

```
108     plt.legend(loc = location, fontsize = fontsize, frameon = False)
109
110 '''
111 savefig:
112 bbox_inches is always tight
113 '''
114 def savefig(filename):
115     plt.savefig(filename, bbox_inches = 'tight')
116
117
118 def topolar(img, order=1):
119     """
120     Transform img to its polar coordinate representation.
121
122     order: int, default 1
123         Specify the spline interpolation order.
124         High orders may be slow for large images.
125     """
126     # max_radius is the length of the diagonal
127     # from a corner to the mid-point of img.
128     max_radius = 0.5*np.linalg.norm( img.shape )
129
130     def transform(coords):
131         theta = 2*np.pi*coords[1] / (img.shape[1] - 1.)
132
133         radius = max_radius * coords[0] / img.shape[0]
134
135         i = 0.5*img.shape[0] - radius*np.sin(theta)
136         j = radius*np.cos(theta) + 0.5*img.shape[1]
137         return i,j
138
139     polar = geometric_transform(img, transform, order=order)
140
141     rads = max_radius * np.linspace(0,1,img.shape[0])
142     angs = np.linspace(0, 2*np.pi, img.shape[1])
143
144     return polar, (rads, angs)
145 def improve_pairplot(g, replacements):
146     g.fig.set_dpi(300)
147     for idx,i in enumerate(g.axes[0]):
148         for idx_j,j in enumerate(g.axes):
149             g.axes[idx_j][idx].spines['left'].set_linewidth(2)
150             g.axes[idx_j][idx].spines['bottom'].set_linewidth(2)
151             g.axes[idx_j][idx].tick_params(direction = 'in', width = 1.5)
152             xlabel = g.axes[idx_j][idx].get_xlabel()
153             ylabel = g.axes[idx_j][idx].get_ylabel()
154             if xlabel in replacements.keys():
155                 g.axes[idx_j][idx].set_xlabel(replacements[xlabel], fontsize
    = 18)
156             if ylabel in replacements.keys():
157                 g.axes[idx_j][idx].set_ylabel(replacements[ylabel], fontsize
    = 18)
158     return g
159 def analyze_pore(im):
160     voxelsize = 1
```

```
161
162     realvols = []
163     realorientations = []
164     realanisotropies = []
165     realmin_axis_l = []
166     realmaj_axis_l = []
167     im = np.array(im, dtype = 'int')
168     pores = measure.regionprops(im)
169     if len(pores) == 0:
170         return np.array([0,0,0])
171     pore_idx = np.argmax([pore.area for pore in pores])
172     if pores[pore_idx].area > 1:
173         realvols.append(pores[pore_idx].area)
174     realmaj_axis_l.append((pores[pore_idx].major_axis_length)*voxelsize)
175     realmin_axis_l.append((pores[pore_idx].minor_axis_length)*voxelsize)
176
177
178     pore = pores[pore_idx]
179
180     inertia_eigval = pore.inertia_tensor_eigvals
181     inertia = pore.inertia_tensor
182     maxeig = np.argmax(inertia_eigval)
183     eigvec = np.linalg.eig(pore.inertia_tensor)[1]
184     eigvals = np.linalg.eig(pore.inertia_tensor)[0]
185     anis = 0
186     realanisotropies.append(anis)
187     maxvector = eigvec[:, maxeig]
188     orientation = angle_between(maxvector, np.array([0,0,1]))
189     realorientations.append(orientation)
190     return np.array([anis, realvols[0]**(1/3), orientation/np.pi])
191
192 def load_image(fname, dim = 4):
193     im = np.loadtxt(fname).reshape(566,571,dim)
194     img = np.copy(im)
195     boundary = np.zeros((im.shape))
196     boundary[img == [1, 1, 1]] = 1
197     boundary = boundary[:,:, 0]
198     img[im == [1, 1, 1]] = 0
199     return boundary, img
200
201 def load_shell(fname,num_frames = 200, dim = 4, start = 0):
202     total = num_frames
203     im_stack = np.zeros((566,571, total, dim))
204     shells = np.zeros((566,571, total))
205     for i in range(start, start+total):
206         im =
    np.loadtxt(fname+'tiff_threephase_id_'+str(i)).reshape(566,571,dim)
207         print(np.unique(im[:,:,2]))
208         print(fname+'tiff_threephase_id_'+str(i))
209
210         img = np.copy(im)
211         boundary = np.zeros((im.shape))
212         boundary[img == [1, 1, 1]] = 1
213
214         img[im == [1, 1, 1]] = 0
```

```
215         boundary = boundary[:,:, 0]
216         shells[:,:,i-start] = boundary
217         im_stack[:,:,i-start] = img
218 
219         print(i)
220 
221     return shells ,im_stack
222 
223 
224 
225 def compute_statistics(pore_dataset, voxelsize, imstack = None):
226     anisotropies = []
227     orientations = []
228     vols = []
229     sphericity = []
230     x_locs = []
231     y_locs = []
232     z_locs = []
233     maj_axis_l = []
234     min_axis_l = []
235     phis = []
236     for i in range(len(pore_dataset)):
237         pore = pore_dataset[i]
238 
239         if pore_dataset[i]['area'] == 1:
240             continue
241         vols.append(pore_dataset[i]['area']*voxelsize*voxelsize*voxelsize)
242         y_locs.append(pore_dataset[i]['centroid'][1]*voxelsize)
243         x_locs.append(pore_dataset[i]['centroid'][0]*voxelsize)
244         z_locs.append(pore_dataset[i]['centroid'][2]*voxelsize)
245         maj_axis_l.append((pore_dataset[i].major_axis_length)*voxelsize)
246         thresh = 0.3
247 
248         inertia_eigval = pore.inertia_tensor_eigvals
249         inertia = pore.inertia_tensor
250         maxeig = np.argmax(inertia_eigval)
251         eigvec = np.linalg.eig(pore.inertia_tensor)[1]
252         eigvals = np.linalg.eig(pore.inertia_tensor)[0]
253         anis = 1 - np.min(eigvals)/np.max(eigvals)
254         anisotropies.append(anis)
255         maxvector = eigvec[:, maxeig]
256         orientation = angle_between(maxvector, np.array([0,0,1]))
257         orientations.append(orientation)
258 
259         phi = angle_between(maxvector, np.array([0,1,0]))
260         phis.append(phi)
261     stats = {
262         'anisotropies': anisotropies,
263         'orientations' : orientations,
264         'vols' : vols,
265         'x_locs' : x_locs,
266         'y_locs' : y_locs,
267         'maj_axis_l' : maj_axis_l,
268         'z_locs' : z_locs,
269         'phis': phis
```

```
270     }
271     if not len(np.unique([len(stat) for stat in stats.values()])) == 1:
272         print([len(stat) for stat in stats.values()])
273         breakpoint()
274 
275     return stats#, examples
276 def extract_pores(imstack):
277     boundary_imstack = np.copy(imstack)
278     boundary_imstack[imstack != 255] = 0
279     boundary_imstack[imstack == 255] = 1
280 
281     imstack[imstack  == 255] = 0
282     imstack[imstack == 159] = 1
283     imstack = np.array(imstack, dtype = 'uint8')
284     im = measure.label(imstack[:,:,:])
285     props = skimage.measure.regionprops(im[:,:,:])
286     return props, im, boundary_imstack
287 
288 
289 def draw_centroid(label_img, frame_id = 0):
290     plt.imshow(label_img, cmap = 'binary')
291 
292     props =  measure.regionprops(label_img)
293     prop_labels = [prop.label for prop in props]
294     for idx in np.unique(label_img):
295         if idx != 0:
296 
297             prop_idx = np.where(prop_labels == idx)[0][0]
298             centroid = props[prop_idx].centroid
299             plt.text(centroid[1], centroid[0], idx)
300     plt.title('Frame ' + str(frame_id))
301     plt.savefig('centroidgen'+ str(frame_id) + '.png')
302     plt.clf()
303 
304 def plot_image(images,shell, title, global_idx = 0):
305     os.makedirs(title, exist_ok = True)
306     print("SAVING TO", title)
307     for p in range(images.shape[2]):
308         plt.imsave(title + str(global_idx+p) + '.png',np.array((images[:,:,
    p]*(255//2) + shell[:,:,p]*255), dtype = int), cmap = 'gist_gray')
309         plt.clf()
310 
311 
312 def load_existing(start = 0, num_frames = 200):
313     total = num_frames
314     seconds = time.time()
315     shell, im_stack = load_shell('/media/cmu/DATA/francis/pore_gan
    /8approxstyleganthreechannelboundaryfulltest_threephase_2D/', dim = 4, start
    = start, num_frames = num_frames)
316     pore_part = np.zeros((566,571, total)) #shell.shape
317     sum_tot = np.sum(im_stack[:,:,:], axis = 3)
318     sum_tot[sum_tot > 0] = 1
319     return pore_part, shell, im_stack
320 
321 def read_from_file(filename, iteration = None, restart = None):
```

```
322     tensors = torch.load(filename)
323     im_opt = np.squeeze(tensors['tensor_opt'])
324     plt.imshow(im_opt, cmap = 'binary')
325     plt.colorbar()
326     plt.title('MST generated surface roughness profiles, iteration: ' +
    str(iteration) + ' instance: ' + str(restart))
327     plt.savefig('modelCfigs' + str(restart) + '_' + str(iteration)
    +'profiletest' + '.png')
328 
329     plt.clf()
330 
331 def load_boundary(num_frames = 500, start = 0, pore_part_shape = (566, 571),
    return_profile = False, im_opt = None, resultsdir = None, folder_index = 0):
332     i = folder_index
333 
334     if resultsdir is None:
335         resultsdir = 'make_surface/results
    /sample_number_0original_folder_{}'.format(i)
336     # i = 0
337     xmean = np.loadtxt(resultsdir+ '/xmean{}'.format(i))
338     ymean = np.loadtxt(resultsdir+ '/ymean{}'.format(i))
339     minmax = pd.read_csv(resultsdir+ '/minmax_values{}.csv'.format(i,i))
340     print(minmax['max'])
341     print(minmax['min'])
342     maxim = np.array(minmax['max'])
343     minim = np.array(minmax['min'])
344     ratio = minim/maxim
345 
346 
347     if im_opt is None:
348         fname = resultsdir + '/modelC_krec' + str(0) + '_start' + str(1) +
    '.pt'
349         tensors = torch.load(fname)
350         im_opt = np.array(np.squeeze(tensors['tensor_opt']) + xmean+ymean)
351     print(i)
352 
353     polar_index_r = linear_polar(np.zeros(pore_part_shape)).shape[0]
354     polar_index_theta = linear_polar(np.zeros(pore_part_shape)).shape[1]
355     new_im = resize(im_opt, (2000, polar_index_theta), order = 3)
356     line = new_im[0]
357 
358     line_int = np.array(new_im[0], dtype = 'int')
359     polar_image = np.zeros((polar_index_r, polar_index_theta))
360     idxs = (line_int, np.arange(polar_index_theta, dtype = int))
361     polar_image[idxs] = 1
362     shells = np.zeros((pore_part_shape[0], pore_part_shape[1], num_frames),
    dtype = 'uint8')
363 
364 
365 
366 
367     xs = np.linspace(0, pore_part_shape[0], pore_part_shape[0])
368     ys = np.linspace(0, pore_part_shape[1], pore_part_shape[1])
369 
370     # full coordinate arrays
```

```
371
372     xx, yy = np.meshgrid(xs, ys)
373     zz = np.sqrt((xx - pore_part_shape[0]//2)**2 + (yy -
    pore_part_shape[1]//2)**2)
374     toprow = zz[0]
375     bottomrow = zz[-1]
376     left = zz[:,0]
377
378
379     right = zz[:,-1]
380     continuous = np.hstack((toprow, bottomrow, left, right))
381     # Calculate the radius at which it would
382     edge_radii = np.array(continuous)
383     max_valid_radius = np.min(edge_radii)
384     min_valid_radius = ratio*max_valid_radius
385
386     for k in range(start, num_frames+start):
387         line = new_im[k]*(max_valid_radius - min_valid_radius) +
    min_valid_radius
388         line[-1] = line[0]
389         line = savgol_filter(line, 27 , 3)
390         line_int = np.array(line, dtype = 'int')
391         polar_image = np.zeros((polar_index_r, polar_index_theta))
392         idxs = (line_int, np.arange(polar_index_theta, dtype = int))
393         polar_image[idxs] = 1
394         shell_img = polar_linear(polar_image, output = (pore_part_shape[0],
    pore_part_shape[1]))
395         shells[:,:, k-start] = np.array(shell_img > 0,dtype='uint8')
396
397     if return_profile:
398         return shells, new_im*(max_valid_radius - min_valid_radius) +
    min_valid_radius
399     return shells
400
401 def analyze_results(original, pore_reconstruct, fname = None,
    lists_all_original = None, lists_all_new = None, voxelsize = 3.49):
402     print('FINISHED RECONSTRUCTION')
403     title = fname
404     frame_tick()
405     os.makedirs(title, exist_ok = True)
406     if lists_all_original == None:
407         props, _, _ = extract_pores(pore_reconstruct)
408         props_orig, _, _ = extract_pores(original)
409         stats = compute_statistics(props, voxelsize=voxelsize)
410         # stats_orig = compute_statistics(props_orig, voxelsize=3.49)
411
412
413         total_anisotropies = []
414         total_x = []
415         total_y = []
416         total_orientations = []
417         total_z = []
418         total_maj = []
419         total_vols = []
420
```

```
421         total_phis = []
422         gt_total_phis =[]
423         total_x.extend(stats['x_locs'])
424         total_y.extend(stats['y_locs'])
425         total_maj.extend(stats['maj_axis_l'])
426         total_vols.extend(stats['vols'])
427         total_anisotropies.extend(stats['anisotropies'])
428         total_orientations.extend(stats['orientations'])
429         total_phis.extend(stats['phis'])
430         total_z.extend(stats['z_locs'])
431         stats = compute_statistics(props_orig, voxelsize=voxelsize)
432 
433 
434         gt_total_anisotropies = []
435         gt_total_x = []
436         gt_total_y = []
437         gt_total_orientations = []
438         gt_total_z = []
439         gt_total_maj = []
440 
441         gt_total_vols = []
442 
443         gt_total_x.extend(stats['x_locs'])
444         gt_total_y.extend(stats['y_locs'])
445         gt_total_maj.extend(stats['maj_axis_l'])
446         gt_total_vols.extend(stats['vols'])
447         gt_total_anisotropies.extend(stats['anisotropies'])
448         gt_total_orientations.extend(stats['orientations'])
449         gt_total_phis.extend(stats['phis'])
450         gt_total_z.extend(stats['z_locs'])
451 
452 
453         lists_all_original = {'x_locs': gt_total_x, 'y_locs': gt_total_y,
    'maj_axis_l': gt_total_maj, 'vols': gt_total_vols,
    'anisotropies':gt_total_anisotropies, 'orientations': gt_total_orientations,
    'z_locs': gt_total_z, 'phis' : gt_total_phis }
454         lists_all_new  = {'x_locs': total_x, 'y_locs': total_y, 'maj_axis_l':
    total_maj, 'vols': total_vols, 'anisotropies':total_anisotropies,
    'orientations': total_orientations, 'z_locs': total_z, 'phis': total_phis }
455 
456 
457     density = True
458     fig = plt.figure(figsize=[4,3], dpi = 300)
459     histogram = plt.hist((np.array(lists_all_new['orientations'])/np.pi)*180,
    density = density, bins=30, edgecolor = 'k', label = 'reconstructed', alpha
    = 0.7)
460     histogram2 =
    plt.hist((np.array(lists_all_original['orientations'])/np.pi)*180, density =
    density, bins=30, edgecolor = 'k', label = 'original', alpha = 0.7)
461     plt.title("Orientation")
462 
463     np.savetxt(title+ 'num_pores',
    np.array([len(lists_all_original['orientations']),
    len(lists_all_new['orientations'])]))
464     plt.xlabel(r"Angle [Degrees]")
```

```
465     plt.ylabel("Probability")
466     frame_tick()
467     legend()
468     plt.tight_layout()
469     plt.savefig(title + "orientation" + ".png")
470     plt.show()
471     plt.clf()
472
473
474
475
476     fig = plt.figure(figsize=[4,3], dpi = 300)
477     histogram = plt.hist((np.array(lists_all_new['phis'])/np.pi)*180, density
    = density, bins=30, edgecolor = 'k', label = 'reconstructed', alpha  = 0.7)
478     histogram2 = plt.hist((np.array(lists_all_original['phis'])/np.pi)*180,
    density = density, bins=30, edgecolor = 'k', label = 'original', alpha = 0.7)
479     plt.title("Phi")
480     np.savetxt(title+ 'num_pores', np.array([len(lists_all_original['phis']),
    len(lists_all_new['phis'])]))
481     plt.xlabel(r"Angle [Degrees]")
482     plt.ylabel("Probability")
483     frame_tick()
484     legend()
485     plt.tight_layout()
486     plt.savefig(title + "phi" + ".png")
487     plt.show()
488     plt.clf()
489
490
491     fig = plt.figure(figsize=[4,3], dpi = 300)
492     histogram = plt.hist((np.array(lists_all_new['anisotropies'])), density =
    density, bins=30, edgecolor = 'k', label = 'reconstructed', alpha  = 0.7)
493     histogram2 = plt.hist(np.array(lists_all_original['anisotropies']),
    density = density, bins=30, edgecolor = 'k', label = 'original', alpha = 0.7)
494     plt.title("Anisotropy")
495     plt.xlabel(r"Anisotropy")
496     plt.ylabel("Probability")
497     frame_tick()
498     legend()
499     plt.tight_layout()
500     plt.savefig(title+"anisotropy" + ".png")
501     plt.show()
502     plt.clf()
503     fig = plt.figure(figsize=[4,3], dpi = 300)
504     histogram = plt.hist((np.array(lists_all_new['y_locs'])), density =
    density, bins=30, edgecolor = 'k', label = 'reconstructed', alpha  = 0.7)
505     histogram2 = plt.hist((np.array(lists_all_original['y_locs'])), density =
    density, bins=30, edgecolor = 'k', label = 'original', alpha = 0.7)
506     plt.title("Y location")
507     plt.xlabel(r"Y location [micrometers]")
508     plt.ylabel("Probability")
509     frame_tick()
510     legend()
511     plt.tight_layout()
512     plt.savefig(title+"yloc" + ".png")
```

```
513     plt.show()
514 
515     plt.clf()
516     fig = plt.figure(figsize=[4,3], dpi = 300)
517 
518     histogram = plt.hist((np.array(lists_all_new['x_locs'])), density =
    density, bins=30, edgecolor = 'k', label = 'reconstructed', alpha  = 0.7)
519     histogram2 = plt.hist((np.array(lists_all_original['x_locs'])), density =
    density, bins=30, edgecolor = 'k', label = 'original', alpha = 0.7)
520     plt.title("X location")
521     plt.xlabel(r"X location [micrometers]")
522     plt.ylabel("Probability")
523     frame_tick()
524     legend()
525     plt.tight_layout()
526     plt.savefig(title+"xloc" + ".png")
527     plt.show()
528 
529     plt.clf()
530     fig = plt.figure(figsize=[4,3], dpi = 300)
531 
532     histogram = plt.hist(lists_all_new['vols'], density = density, alpha =
    0.7,bins=np.logspace(np.log10(10e1),np.log10(10e5)), edgecolor = 'k', label =
    'reconstructed')
533     histogram2 = plt.hist(lists_all_original['vols'], density = density,
    alpha = 0.7,bins=np.logspace(np.log10(10e1),np.log10(10e5)), edgecolor = 'k',
    label = 'original')
534 
535     plt.title("Volume")
536 
537     plt.xscale('log')
538     plt.xlabel(r"Volume [$\mu m^3$]")
539     plt.ylabel("Probability")
540     frame_tick()
541     legend()
542     plt.tight_layout()
543     plt.savefig(title + "vols" + ".png")
544     plt.show()
545     plt.clf()
546 
547     print("Pores in the original sample: " +
    str(len(lists_all_original['vols'])))
548     print("Pores in the new sample: " + str(len(lists_all_new['vols'])))
549 def replace_sampling(pore_part,  generated_boundary, n_bins = 30, window_size
    = 100, properties_folder = './analyze_pore_samples/results/pore_properties
    /probability_matrices/', use_generated = True, use_gt = True):
550     import scipy
551 
552     prob_matrix_volume=np.load(properties_folder + str(n_bins) +
    '_{}allprob_matrix_volume.npy'.format(0))
553     prob_matrix_num =  np.load(properties_folder + str(n_bins) +
    '_{}allprob_matrix_num.npy'.format(0))*(window_size/100)#/2
554     bin_edges_vols =   np.load(properties_folder + str(n_bins) +
    '_{}allbin_edges_vols.npy'.format(0))
555     bin_edges_anis =   np.load(properties_folder + str(n_bins) +
```

```
    '_{}allbin_edges_anis.npy'.format(0))
556     bin_edges_phis = np.load(properties_folder + str(n_bins) +
    '_{}allbin_edges_phis.npy'.format(0))
557     prob_matrix_phis = np.load(properties_folder + str(n_bins) +
    '_{}allprob_matrix_phis.npy'.format(0))
558
559     prob_matrix_anis =       np.load(properties_folder + str(n_bins) +
    '_{}allprob_matrix_anis.npy'.format(0))
560     bin_edges_orientations = np.load(properties_folder + str(n_bins) +
    '_{}allbin_edges_orientations.npy'.format(0))
561     prob_matrix_orientation= np.load(properties_folder + str(n_bins) +
    '_{}allprob_matrix_orientations.npy'.format(0))
562
563     target_list_size = []
564     target_anis = []
565     target_vols = []
566     actual_vols = []
567     generated_pore_matrix = np.loadtxt('./reconstruction/gan/figures
    /pore_matrix_updated.csv')
568     gt_pore_matrix = np.loadtxt('./analyze_pore_samples/results
    /individual_pore_samples/partsample0/pore_matrix')
569     x_extent = np.linspace(0,pore_part.shape[0], n_bins)
570     y_extent = np.linspace(0, pore_part.shape[1], n_bins)
571     z_extent = np.arange(0, pore_part.shape[2],window_size)
572     gt_pores_used = []
573     gen_pores_used = []
574     gen_losses = []
575     gt_losses = []
576     for idx_x,x_sample in enumerate(x_extent):
577         for idx_y,y_sample  in enumerate(y_extent):
578             for idx_z, z_sample in enumerate(z_extent):
579
580                 vol_hist = prob_matrix_volume[idx_x, idx_y, :]
581
582                 vol_hist_dist = scipy.stats.rv_histogram((vol_hist,
    bin_edges_vols))
583                 num = prob_matrix_num[idx_x, idx_y]
584                 if num == 0:
585                     continue
586                 if num < 1:
587                     unif_sample = np.random.uniform()
588                     if num > unif_sample:
589                         num = 1
590                     else:
591                         num = 0
592                         continue
593                 elif num > 1:
594                     num = int(np.around(num))
595
596                 volumes = vol_hist_dist.rvs(size=int(num))
597                 ani_hist = prob_matrix_anis[idx_x, idx_y]
598                 ani_hist_dist = scipy.stats.rv_histogram((ani_hist,
    bin_edges_anis))
599                 anisotropies = ani_hist_dist.rvs(size=int(num))
600
```

```
601
602                 angle_hist = prob_matrix_orientation[idx_x, idx_y]
603                 angle_hist_dist = scipy.stats.rv_histogram((angle_hist,
    bin_edges_orientations))
604                 angles = angle_hist_dist.rvs(size=int(num))
605                 phi_hist = prob_matrix_phis[idx_x, idx_y]
606                 phi_hist_dist = scipy.stats.rv_histogram((phi_hist,
    bin_edges_phis))
607                 phis = phi_hist_dist.rvs(size = int(num))
608                 target_anis.extend(anisotropies)
609                 target_vols.extend(np.array(volumes)**3)
610                 for idx_pore,gen_pore in enumerate(volumes):
611
612                     gen_pore = np.max([gen_pore, 2])
613
614                     tmp_x =
    np.random.randint(0,int(pore_part.shape[0]/n_bins))
615                     tmp_y =
    np.random.randint(0,int(pore_part.shape[1]/n_bins))
616
617                     curr_x = int(x_sample-tmp_x)
618                     curr_y = int(y_sample - tmp_y)
619
620                     curr_z = z_sample+np.random.randint(0,window_size)
621
622                     target_pore= gen_pore**3#/25
623                     target_list_size.append(target_pore)
624                     anis = anisotropies[idx_pore]
625                     angle = angles[idx_pore]
626                     phi = phis[idx_pore]
627
628                     pore_matrix = generated_pore_matrix
629                     pore_matrix = generated_pore_matrix
630                     pore_sizes = (pore_matrix[:,0])
631
632                     gen_pore_identity = np.argmin(np.abs(target_pore -
    pore_matrix[:,0])/np.mean(pore_matrix[:,0]) + np.abs(anis -
    generated_pore_matrix[:,2])/np.mean(generated_pore_matrix[:,2]) +
    np.abs(angle -
    generated_pore_matrix[:,3])/np.mean(generated_pore_matrix[:,3])+  np.abs(phi
    - generated_pore_matrix[:,4])/np.mean(generated_pore_matrix[:,4]))# +
    np.abs(phi - pore_matrix[:,4])/np.mean(pore_matrix[:,4]))
633                     gen_loss = np.min(np.abs(target_pore -
    pore_matrix[:,0])/np.mean(pore_matrix[:,0]) + np.abs(anis -
    pore_matrix[:,2])/np.mean(pore_matrix[:,2]) + np.abs(angle -
    pore_matrix[:,3])/np.mean(pore_matrix[:,3])+  np.abs(phi -
    generated_pore_matrix[:,4])/np.mean(generated_pore_matrix[:,4]))# +
    np.abs(phi - pore_matrix[:,4])/np.mean(pore_matrix[:,4]))
634
635                     pore_matrix = gt_pore_matrix
636                     gt_pore_identity = np.argmin(np.abs(target_pore -
    pore_matrix[:,0])/np.mean(pore_matrix[:,0]) + np.abs(anis -
    pore_matrix[:,2])/np.mean(pore_matrix[:,2]) + np.abs(angle -
    pore_matrix[:,3])/np.mean(pore_matrix[:,3])+  np.abs(phi -
    pore_matrix[:,4])/np.mean(pore_matrix[:,4]))# + np.abs(phi -
```

```
    pore_matrix[:,4])/np.mean(pore_matrix[:,4]))
637                     gt_loss = np.min(np.abs(target_pore -
    pore_matrix[:,0])/np.mean(pore_matrix[:,0]) + np.abs(anis -
    pore_matrix[:,2])/np.mean(pore_matrix[:,2]) + np.abs(angle -
    pore_matrix[:,3])/np.mean(pore_matrix[:,3])+  np.abs(phi -
    pore_matrix[:,4])/np.mean(pore_matrix[:,4]))# + np.abs(phi -
    pore_matrix[:,4])/np.mean(pore_matrix[:,4]))
638
639                     gen_losses.append(gen_loss)
640                     gt_losses.append(gt_loss)
641
642                     if use_generated and use_gt: # Use both generated and
    ground truth pores
643                         if gen_loss > gt_loss:
644                             generated = False
645                             pore_identity = gt_pore_identity
646                             gt_pores_used.append(pore_identity)
647                         else:
648                             generated = True
649                             pore_identity = gen_pore_identity
650                             gen_pores_used.append(pore_identity)
651                     elif use_generated: # only use generated
652                         generated = True
653                         pore_identity = gen_pore_identity
654                         gen_pores_used.append(pore_identity)
655                     else: # only use ground truth
656
657                         generated = False
658                         pore_identity = gt_pore_identity
659                         gt_pores_used.append(pore_identity)
660
661                     if generated:
662                         filename = './reconstruction
    /gan/saved_generated_pores/generator_' + str(pore_identity) +'.hdf5'
663                     else:
664                         filename   = './analyze_pore_samples/results
    /individual_pore_samples/partsample0/pore_original_' + str(pore_identity) +
    '.hdf5'
665
666                     data = read_file(filename)
667
668                     if len(np.where(data)[0]) == 0:
669                         print('continue 1 activated')
670                         continue
671                     center = 32
672                     size = 32
673                     if generated:
674                         data = np.array(data)#/255
675                     else:
676                         data = np.array(data)
677                     xmin = np.min(np.where(data)[0])
678                     ymin = np.min(np.where(data)[1])
679                     zmin = np.min(np.where(data)[2])
680
681                     xmax = np.max(np.where(data)[0])+1
```

```
682                     ymax = np.max(np.where(data)[1])+1
683                     zmax = np.max(np.where(data)[2])+1
684 
685 
686                     xmin_slice = np.min(np.where(data[:,:, zmin])[0])
687                     ymin_slice = np.min(np.where(data[:,:,zmin])[1])
688                     lowerlim_x =  0
689                     lowerlim_y =  0
690                     data_ylower = 0
691                     data_xlower = 0
692 
693 
694                     data_yupper = 64-np.abs(np.min((0, pore_part.shape[1] -
    (lowerlim_y + 64)))) # in case of negative indices
695                     data_xupper = 64-np.abs(np.min((0, pore_part.shape[0] -
    (lowerlim_x + 64))))
696                     target_z = curr_z
697                     if int(target_z) + (zmax-zmin) > pore_part.shape[2]:
698                         print('continue 2 activated: pore on back surface')
699                         continue
700                     if int(target_z) < 0:
701                         print('continue 3 activated: pore on front surface')
702                         continue
703                     try:
704                         test_window = pore_part[curr_x: curr_x+ int(xmax-
    xmin), curr_y:curr_y + int(ymax-ymin), int(target_z):int(target_z)+(zmax-
    zmin)]  + data[xmin:xmax , ymin:ymax, zmin:zmax]
705                     except Exception as e:
706                         print(e)
707                         print('continue 4 activated: indexing exception')
708                         continue
709                     if 0 in test_window.shape:
710                         print('continue 5 activated: unresolved pore')
711                         continue
712 
713                     elif np.max(test_window) > 1:
714 
715                         print('continue 6 activated: Collision')
716                         collision_z = np.where(test_window >1)[2][0]
717                         continue
718 
719                     try:
720                         xdatastart = np.min(np.where(data)[0])
721                         labelpore = measure.label(data[xmin:xmax , ymin:ymax,
    zmin:zmax])
722 
723                         oldpore =  pore_part[curr_x: curr_x+ int(xmax-xmin),
    curr_y:curr_y + int(ymax-ymin), int(target_z):int(target_z)+(zmax-zmin)]
724 
725                         newpore = data[xmin:xmax , ymin:ymax, zmin:zmax]
    +pore_part[curr_x: curr_x+ int(xmax-xmin), curr_y:curr_y + int(ymax-ymin),
    int(target_z):int(target_z)+(zmax-zmin)]
726                         if len(measure.regionprops(measure.label(newpore)))
    != len(measure.regionprops(measure.label(oldpore))) + 1:
727 
```

```
    print(len(measure.regionprops(measure.label(newpore))),len(measure.regionprop
    s(measure.label(oldpore))), 'cmerged')
728                             print('continue 7 activated: Collision')
729                             continue
730
731                         if generated:
732                             pore_part[curr_x: curr_x+ int(xmax-xmin),
    curr_y:curr_y + int(ymax-ymin), int(target_z):int(target_z)+(zmax-zmin)] =
    2*data[xmin:xmax , ymin:ymax, zmin:zmax]+ pore_part[curr_x: curr_x+ int(xmax-
    xmin), curr_y:curr_y + int(ymax-ymin), int(target_z):int(target_z)+(zmax-
    zmin)]
733                         else:
734                             pore_part[curr_x: curr_x+ int(xmax-xmin),
    curr_y:curr_y + int(ymax-ymin), int(target_z):int(target_z)+(zmax-zmin)] =
    data[xmin:xmax , ymin:ymax, zmin:zmax]+ pore_part[curr_x: curr_x+ int(xmax-
    xmin), curr_y:curr_y + int(ymax-ymin), int(target_z):int(target_z)+(zmax-
    zmin)]
735
736                     except Exception as f:
737                         print(f)
738                         print('continue 8 activated: Insertion process
    failed')
739
740                         continue
741
742
    actual_vols.append(measure.regionprops(measure.label(np.array(data)))
    [0].area)
743     # breakpoint()
744     return pore_part, target_list_size
745
746
747 def replace_sampling_polar(pore_part,  generated_boundary, profile_2d, n_bins
    = 30, window_size = 100, properties_folder = './analyze_pore_samples/results
    /pore_properties/probability_matrices/', folder_index = 0):
748     import scipy#.stats
749     prob_matrix_dir = properties_folder
750     np.load(properties_folder + str(n_bins) +
    '_{}allprob_matrix_volume.npy'.format(0))
751     prob_matrix_volume=np.load(prob_matrix_dir + str(n_bins) +
    '_{}polarprob_matrix_volume.npy'.format(folder_index))
752     prob_matrix_num =  np.load(prob_matrix_dir + str(n_bins) +
    '_{}polarprob_matrix_num.npy'.format(folder_index))*(window_size/100)#/2
753     bin_edges_vols =   np.load(prob_matrix_dir + str(n_bins) +
    '_{}polarbin_edges_vols.npy'.format(folder_index))
754     bin_edges_anis =   np.load(prob_matrix_dir + str(n_bins) +
    '_{}polarbin_edges_anis.npy'.format(folder_index))
755     bin_edges_phis = np.load(prob_matrix_dir + str(n_bins) +
    '_{}polarbin_edges_phis.npy'.format(folder_index))
756     prob_matrix_phis = np.load(prob_matrix_dir + str(n_bins) +
    '_{}polarprob_matrix_phis.npy'.format(folder_index))
757     prob_matrix_anis =       np.load(prob_matrix_dir + str(n_bins) +
    '_{}polarprob_matrix_anis.npy'.format(folder_index))
758     bin_edges_orientations = np.load(prob_matrix_dir + str(n_bins) +
    '_{}polarbin_edges_orientations.npy'.format(folder_index))
```

```
759     prob_matrix_orientation= np.load(prob_matrix_dir + str(n_bins) +
    '_{}polarprob_matrix_orientations.npy'.format(folder_index))
760     target_list_size = []
761     target_anis = []
762     target_vols = []
763     actual_vols = []
764     generated_pore_matrix = np.loadtxt('./reconstruction/gan/figures
    /pore_matrix_updated.csv')
765     gt_pore_matrix = np.loadtxt('./analyze_pore_samples/results
    /individual_pore_samples/partsample0/pore_matrix')
766     r_extent = np.linspace(0,1.1, n_bins)
767     theta_extent = np.linspace(0, np.pi*2, n_bins)
768     z_extent = np.arange(0, pore_part.shape[2],window_size)
769     gt_pores_used = []
770     gen_pores_used = []
771     gen_losses = []
772     gt_losses = []
773     angles = np.linspace(0, np.pi*2, profile_2d.shape[1])
774     radii_totals = []
775     for idx_x,x_sample in enumerate(r_extent):
776         for idx_y,y_sample  in enumerate(theta_extent):
777             for idx_z, z_sample in enumerate(z_extent):
778
779                 vol_hist = prob_matrix_volume[idx_x, idx_y, :]
780
781                 vol_hist_dist = scipy.stats.rv_histogram((vol_hist,
    bin_edges_vols))
782                 num = prob_matrix_num[idx_x, idx_y]
783                 if num == 0:
784                     continue
785                 if num < 1:
786                     unif_sample = np.random.uniform()
787                     if num > unif_sample:
788                         num = 1
789                     else:
790                         num = 0
791                         continue
792                 elif num > 1:
793                     num = int(np.around(num))
794
795                 volumes = vol_hist_dist.rvs(size=int(num))
796                 ani_hist = prob_matrix_anis[idx_x, idx_y]
797                 ani_hist_dist = scipy.stats.rv_histogram((ani_hist,
    bin_edges_anis))
798                 anisotropies = ani_hist_dist.rvs(size=int(num))
799
800
801                 angle_hist = prob_matrix_orientation[idx_x, idx_y]
802                 angle_hist_dist = scipy.stats.rv_histogram((angle_hist,
    bin_edges_orientations))
803                 angles = angle_hist_dist.rvs(size=int(num))
804                 phi_hist = prob_matrix_phis[idx_x, idx_y]
805                 phi_hist_dist = scipy.stats.rv_histogram((phi_hist,
    bin_edges_phis))
806                 phis = phi_hist_dist.rvs(size = int(num))
```

```
807
808                 target_anis.extend(anisotropies)
809                 target_vols.extend(np.array(volumes)**3)
810                 print(x_sample, "RADIUS SAMPLE")
811                 for idx_pore,gen_pore in enumerate(volumes):
812
813                     gen_pore = np.max([gen_pore, 2])
814                     tmp_x = np.random.uniform(0,r_extent[1])
815                     tmp_y = np.random.uniform(0,theta_extent[1])
816                     curr_radius = (x_sample-tmp_x)
817                     curr_angle = (y_sample - tmp_y)
818                     idx_angle = np.argmin(np.abs(curr_angle-
    np.linspace(0,2*np.pi, profile_2d.shape[1])))
819
820                     curr_z = z_sample+np.random.randint(0,window_size)
821                     radius_total = profile_2d[curr_z, idx_angle]
822                     full_radii = curr_radius*radius_total
823                     if x_sample == r_extent[-1]:
824                         print(curr_radius, "Current Radius, final evolution")
825                     curr_x, curr_y = unmap_pixel(full_radii, theta_idx =
    idx_angle, theta = curr_angle,output= pore_part[:,:,0].shape)
826                     radii_totals.append(curr_radius)
827
828                     target_pore= gen_pore**3
829                     target_list_size.append(target_pore)
830                     anis = anisotropies[idx_pore]
831                     angle = angles[idx_pore]
832                     phi = phis[idx_pore]
833
834                     pore_matrix = generated_pore_matrix
835                     pore_matrix = generated_pore_matrix
836                     pore_sizes = (pore_matrix[:,0])
837
838                     gen_pore_identity = np.argmin(np.abs(target_pore -
    pore_matrix[:,0])/np.mean(pore_matrix[:,0]) + np.abs(anis -
    generated_pore_matrix[:,2])/np.mean(generated_pore_matrix[:,2]) +
    np.abs(angle -
    generated_pore_matrix[:,3])/np.mean(generated_pore_matrix[:,3])+  np.abs(phi
    - generated_pore_matrix[:,4])/np.mean(generated_pore_matrix[:,4]))# +
    np.abs(phi - pore_matrix[:,4])/np.mean(pore_matrix[:,4]))
839                     gen_loss = np.min(np.abs(target_pore -
    pore_matrix[:,0])/np.mean(pore_matrix[:,0]) + np.abs(anis -
    pore_matrix[:,2])/np.mean(pore_matrix[:,2]) + np.abs(angle -
    pore_matrix[:,3])/np.mean(pore_matrix[:,3])+  np.abs(phi -
    generated_pore_matrix[:,4])/np.mean(generated_pore_matrix[:,4]))# +
    np.abs(phi - pore_matrix[:,4])/np.mean(pore_matrix[:,4]))
840
841                     pore_matrix = gt_pore_matrix
842                     gt_pore_identity = np.argmin(np.abs(target_pore -
    pore_matrix[:,0])/np.mean(pore_matrix[:,0]) + np.abs(anis -
    pore_matrix[:,2])/np.mean(pore_matrix[:,2]) + np.abs(angle -
    pore_matrix[:,3])/np.mean(pore_matrix[:,3])+  np.abs(phi -
    pore_matrix[:,4])/np.mean(pore_matrix[:,4]))# + np.abs(phi -
    pore_matrix[:,4])/np.mean(pore_matrix[:,4]))
843                     gt_loss = np.min(np.abs(target_pore -
```

```
    pore_matrix[:,0])/np.mean(pore_matrix[:,0]) + np.abs(anis -
    pore_matrix[:,2])/np.mean(pore_matrix[:,2]) + np.abs(angle -
    pore_matrix[:,3])/np.mean(pore_matrix[:,3])+  np.abs(phi -
    pore_matrix[:,4])/np.mean(pore_matrix[:,4]))# + np.abs(phi -
    pore_matrix[:,4])/np.mean(pore_matrix[:,4]))
844                     gen_losses.append(gen_loss)
845                     gt_losses.append(gt_loss)
846
847                     if gen_loss > gt_loss:
848                         generated = False
849                         pore_identity = gt_pore_identity
850                         gt_pores_used.append(pore_identity)
851                     else:
852
853                         generated = True
854                         pore_identity = gen_pore_identity
855                         gen_pores_used.append(pore_identity)
856
857                     if generated:
858                         filename = './reconstruction
    /gan/saved_generated_pores/generator_' + str(pore_identity) +'.hdf5'
859                     else:
860                         filename   = './analyze_pore_samples/results
    /individual_pore_samples/partsample0/pore_original_' + str(pore_identity) +
    '.hdf5'
861                     data = read_file(filename)
862                     if len(np.where(data)[0]) == 0:
863                         print('Empty pore, skipped')
864                         continue
865
866                     center = 32
867                     size = 32
868                     if generated:
869                         data = np.array(data)#/255
870                     else:
871                         data = np.array(data)
872
873                     xmin = np.min(np.where(data)[0])
874                     ymin = np.min(np.where(data)[1])
875                     zmin = np.min(np.where(data)[2])
876
877                     xmax = np.max(np.where(data)[0])+1
878                     ymax = np.max(np.where(data)[1])+1
879                     zmax = np.max(np.where(data)[2])+1
880
881
882                     xmin_slice = np.min(np.where(data[:,:, zmin])[0])
883                     ymin_slice = np.min(np.where(data[:,:,zmin])[1])
884
885
886                     lowerlim_x =  0
887                     lowerlim_y =  0
888                     data_ylower = 0
889                     data_xlower = 0
890
```

```
891
892                     data_yupper = 64-np.abs(np.min((0, pore_part.shape[1] -
    (lowerlim_y + 64)))) # in case of negative indices
893                     data_xupper = 64-np.abs(np.min((0, pore_part.shape[0] -
    (lowerlim_x + 64))))
894                     target_z = curr_z
895                     if int(target_z) + (zmax-zmin) > pore_part.shape[2]:
896                         print('continue 2 activated')
897                         continue
898                     if int(target_z) < 0:
899                         print('continue 3 activated')
900                         continue
901                     try:
902                         test_window = pore_part[curr_x: curr_x+ int(xmax-
    xmin), curr_y:curr_y + int(ymax-ymin), int(target_z):int(target_z)+(zmax-
    zmin)]  + data[xmin:xmax , ymin:ymax, zmin:zmax]
903                     except Exception as e:
904                         print(e)
905                         print('continue 4 activated')
906                         continue
907                     if 0 in test_window.shape:
908                         print('continue 5 activated')
909                         continue
910
911                     elif np.max(test_window) > 1:
912                         print('continue 6 activated')
913                         collision_z = np.where(test_window >1)[2][0]
914                         continue
915                     try:
916                         xdatastart = np.min(np.where(data)[0])
917                         labelpore = measure.label(data[xmin:xmax , ymin:ymax,
    zmin:zmax])
918
919                         oldpore =  pore_part[curr_x: curr_x+ int(xmax-xmin),
    curr_y:curr_y + int(ymax-ymin), int(target_z):int(target_z)+(zmax-zmin)]
920                         newpore = data[xmin:xmax , ymin:ymax, zmin:zmax]
    +pore_part[curr_x: curr_x+ int(xmax-xmin), curr_y:curr_y + int(ymax-ymin),
    int(target_z):int(target_z)+(zmax-zmin)]
921                         if len(measure.regionprops(measure.label(newpore)))
    != len(measure.regionprops(measure.label(oldpore))) + 1:
922
    print(len(measure.regionprops(measure.label(newpore))),len(measure.regionprop
    s(measure.label(oldpore))), 'cmerged')
923                             print('continue 7 activated')
924                             continue
925                         if generated:
926                             pore_part[curr_x: curr_x+ int(xmax-xmin),
    curr_y:curr_y + int(ymax-ymin), int(target_z):int(target_z)+(zmax-zmin)] =
    data[xmin:xmax , ymin:ymax, zmin:zmax]+ pore_part[curr_x: curr_x+ int(xmax-
    xmin), curr_y:curr_y + int(ymax-ymin), int(target_z):int(target_z)+(zmax-
    zmin)]
927                         else:
928                             pore_part[curr_x: curr_x+ int(xmax-xmin),
    curr_y:curr_y + int(ymax-ymin), int(target_z):int(target_z)+(zmax-zmin)] =
    data[xmin:xmax , ymin:ymax, zmin:zmax]+ pore_part[curr_x: curr_x+ int(xmax-
```

```
    xmin), curr_y:curr_y + int(ymax-ymin), int(target_z):int(target_z)+(zmax-
    zmin)]
929                     except Exception as f:
930                         print(f)
931                         print('continue 8 activated')
932                         continue
933 
    actual_vols.append(measure.regionprops(measure.label(np.array(data)))
    [0].area)
934     breakpoint()
935     return pore_part, target_list_size
936 def nan_helper(y):
937     """Helper to handle indices and logical indices of NaNs.
938 
939     Input:
940         - y, 1d numpy array with possible NaNs
941     Output:
942         - nans, logical indices of NaNs
943         - index, a function, with signature indices= index(logical_indices),
944           to convert logical indices of NaNs to 'equivalent' indices
945     Example:
946         >>> # linear interpolation of NaNs
947         >>> nans, x= nan_helper(y)
948         >>> y[nans]= np.interp(x(nans), x(~nans), y[~nans])
949     """
950 
951     return np.isnan(y), lambda z: z.nonzero()[0]
952 def trim_boundary(shell, pore_part):
953     for sample in range(pore_part.shape[2]):
954 
955         oldtime = time.time()
956         coords = np.array(np.where(shell[:,:,sample]))
957         index_anglesort = np.argsort(calc_theta(coords, shell[:,:,sample]))
958         coords_pores = np.array(np.where(pore_part[:,:,sample]))
959         outside_bounds = measure.points_in_poly(coords_pores.T,
    np.array(coords)[:, index_anglesort].T)
960         test = np.copy(pore_part[:,:,sample])
961 
962         test[coords_pores[0], coords_pores[1]] = np.array(outside_bounds,
    dtype = 'int')
963         pore_part[:,:,sample] = test
964 
965     return pore_part
966 def threshold_pore_size(profile_2d, pore_part, boundary = None, name = ''):
967     im_label  = measure.label(pore_part)
968     props = measure.regionprops(im_label)
969     small_props= [prop for prop in props if prop.area < 8]
970     centroids = [prop.centroid for prop in props]
971     polar, rs, ts,o, r, out_h, out_w  = linear_polar(pore_part[:,:, 0],
    verbose = 1)
972     radii = []
973     angles = []
974     for centroid in centroids:
975 
976         x = centroid[0]
```

```
977         y = centroid[1]
978         z = int(centroid[2])
979         r_index, theta_index, theta =
    map_pixel(int(x),int(y),pore_part[:,:,z], o = o, r =r, out_h = out_h, out_w =
    out_w, debug= False )
980         radius = profile_2d[z, theta_index] - r_index
981         oldtime = time.time()
982         r_index, theta_index, theta =
    map_pixel(int(x),int(y),pore_part[:,:,z], o = o, r =r, out_h = out_h, out_w =
    out_w, debug= False )
983         newtime = time.time()
984         print(newtime-oldtime)
985         radii.append(radius)
986         angles.append(theta)
987
988
989
990         plt.imshow(pore_part[:,:,z]+boundary[:,:,z])
991         plt.scatter(int(y), int(x))
992         plt.title(str(radius))
993         plt.savefig('./failures/polar' + name + 'frame'+str(z))
994         plt.clf()
995         plt.imshow(linear_polar(boundary[:,:, z]))
996         plt.scatter(theta_index, r_index)
997         plt.title(str(profile_2d[z, theta_index]) + ' radius: ' +
    str(r_index) +' calc diff: '+ str(radius))
998         plt.savefig('./failures/polar' + name + 'polarframe'+str(z))
999         plt.clf()
1000
1001 def plt_scatter_matrices(prior_realizations, post_realizations,
    variable_list, plt_prior=True, plt_post=True, plt_corr=True,
    print_corr=True):
1002     '''
1003     Plot the prior and posterior scatter matrices (on top of each other) for
    a *variable_list*.
1004     This is from a dictionary of *prior_realizations* and
    *post_realizations*.
1005     '''
1006     from pandas import DataFrame
1007     from pandas.plotting import scatter_matrix
1008     import seaborn as sns
1009
1010     def plt_corr_sns(corr):
1011
1012         f, ax = plt.subplots(figsize=(10, 8))
1013         sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool),
    cmap=sns.diverging_palette(220, 10, as_cmap=True),
1014                     square=True, ax=ax, vmin = -1.0, vmax = 1.0)
1015
1016     shape_prior = np.array(prior_realizations[variable_list[0]]).shape
1017     shape_post = np.array(post_realizations[variable_list[0]]).shape
1018     prior_data = prior_realizations.copy()
1019     prior_data['Case'] = ['Ground
    Truth']*shape_prior[0]#np.zeros(shape_prior)
1020     post_data = post_realizations.copy()
```

```
1021     post_data['Case'] = ['Reconstructed']*shape_post[0]#np.ones(shape_post)
1022     plt.figure(figsize = [8,6], dpi = 150)
1023 
1024     if plt_prior and plt_post:
1025         for key in prior_data:
1026             prior_data[key] = np.append(prior_data[key],post_data[key])
1027         data = DataFrame(prior_data)
1028     elif plt_post:
1029         data = DataFrame(post_data)
1030     else:
1031         data = DataFrame(prior_data)
1032     data.to_csv('Generated_data.csv')
1033     return
1034 def fill_boundary(shell, pore_part):
1035     oldtime = time.time()
1036     test = np.zeros(np.shape(pore_part[:,:,:]))
1037     for sample in range(pore_part.shape[2]):
1038         center = (pore_part.shape[0]//2, pore_part.shape[1]//2)
1039         filled  = np.array(segmentation.flood(shell[:,:,sample], (310,310),
     connectivity = 0), dtype = int)
1040         pores = np.array(pore_part[:,:,sample]>0, dtype = int)
1041         test[:,:,sample] = np.array((filled - pores) > 0, dtype= int)
1042         quadrant_check = len(np.where(test[0,0,:])[0])  +
     len(np.where(test[-1,0,:])[0])+ len(np.where(test[0,-1,:])[0])+
     len(np.where(test[-1,-1,:])[0])
1043 
1044         if quadrant_check > 0:
1045             print("Segmentation didn't work, trying dilation ", sample)
1046 
1047             dilated = morphology.dilation(shell[:,:,sample])
1048             skeleton = morphology.skeletonize(dilated)
1049             filled = morphology.flood(np.array(skeleton,dtype = 'uint8'),
     (310,310), connectivity = 0)
1050             pores = np.array(pore_part[:,:,sample]>0, dtype = int)
1051             test[:,:,sample] = np.array((filled - pores) > 0, dtype= int)
1052             quadrant_check = len(np.where(test[0,0,:])[0])  +
     len(np.where(test[-1,0,:])[0])+ len(np.where(test[0,-1,:])[0])+
     len(np.where(test[-1,-1,:])[0])
1053 
1054             if quadrant_check >0:
1055                 print("Segmentation didn't work, trying angle based method ",
     sample)
1056                 coords = np.array(np.where(shell[:,:,sample]))
1057                 index_anglesort = np.argsort(calc_theta(coords,
     shell[:,:,sample]))
1058                 coords_pores = np.array(np.where(pore_part[:,:,sample]))
1059 
1060                 solid =
     measure.grid_points_in_poly(pore_part[:,:,sample].shape, np.array(coords)[:,
     index_anglesort].T)
1061                 test[:,:,sample] = np.array((np.array(solid, dtype = int) -
     np.array(pore_part[:,:,sample]>0, dtype = int)) > 0, dtype = int)
1062 
1063     return test
1064 
```

```
1065 def n_bins_study(profile_2d, pore_part, generated_boundary, gt_pores):
1066     threshold_pore_size(profile_2d, pore_part, generated_boundary)
1067     list_means = []
1068     for n_bins in [5, 10, 20, 30, 50,100]:
1069         pore_part_test, _ = replace_sampling(np.zeros((pore_part.shape[0],
     pore_part.shape[1], 200)), n_bins = n_bins)
1070         list_means.append(np.mean(pore_part_test, axis = 2))
1071         plt.clf()
1072         plt.figure(figsize = [4,3], dpi = 150)
1073         pore_dataset = measure.regionprops(measure.label(pore_part_test))
1074 
1075         pore_gt_dataset,_,_ =
     extract_pores(gt_pores[:,:,:200])#measure.regionprops(measure.label())
1076         locations = [pore.centroid for pore in pore_dataset]
1077 
1078         loc_pores = np.array(locations)
1079         kdtree = spatial.KDTree(locations)
1080         dd, ii = kdtree.query(locations, len(locations))
1081         voxelsize = 3.49
1082         nearest_neighbor = dd[:,1:]*voxelsize
1083         neighbors = dd[:,1]*voxelsize
1084         edges, hist = np.histogram(dd[:,1:]*voxelsize, bins =
     np.arange(0,800,25)*voxelsize)
1085         plt.plot(hist[:-1],edges, linewidth = 2.0,label= "Original
     Distribution")
1086 
1087 
1088         gtlocations = [pore.centroid for pore in pore_gt_dataset]
1089         gtloc_pores = np.array(gtlocations)
1090 
1091         gtkdtree = spatial.KDTree(gtlocations)
1092         gtdd, gtii = kdtree.query(gtlocations, len(gtlocations))
1093         gtnearest_neighbor = gtdd[:,1:]*voxelsize
1094         gtneighbors = gtdd[:,1]*voxelsize
1095         gtedges, gthist = np.histogram(gtdd[:,1:]*voxelsize, bins =
     np.arange(0,800,25)*voxelsize)
1096         plt.plot(gthist[:-1],gtedges, linewidth = 2.0, label = 'Generated
     Distribution')
1097 
1098 
1099         plt.xlabel(r"Distance [$\mu m$]")
1100         plt.ylabel("Number of Pores")
1101         plt.tight_layout()
1102         legend(location='best')
1103         frame_tick()
1104         plt.savefig('matriximproved_boundaryaddednbins' +"/rdf" + str(n_bins)
     + ".png")
1105         plt.clf()
1106         plt.imsave('matriximproved_boundaryaddednbins'
     +'/truedensity'+str(n_bins)+ 'example.png',
     np.sum(pore_part_test+generated_boundary, axis = 2), cmap= 'binary')
1107         plt.imsave('matriximproved_boundaryaddednbins'
     +'/truedensity'+str(n_bins)+ 'gt.png', np.sum(gt_pores[:,:,:500], axis = 2),
     cmap= 'binary')
1108         plt.colorbar()
```

```
1109         plt.savefig()
1110         plt.clf()
1111 
1112 
1113 def save_binary_segment(segment, fname, voxelsize, index, vname = None):
1114     voxel_conversions=[vname]
1115     if segment.shape[0] > segment.shape[1]:
1116         l_pad = (segment.shape[0] - segment.shape[1])//2
1117         r_pad = (segment.shape[0] - segment.shape[1]) - l_pad
1118         padstack = np.pad(segment,  ((0,0),(l_pad,r_pad), (0,0)))
1119 
1120     elif segment.shape[0] < segment.shape[1]:
1121         l_pad = (segment.shape[1] - segment.shape[0])//2
1122         r_pad = (segment.shape[1] - segment.shape[0]) - l_pad
1123         padstack = np.pad(segment,  ((l_pad,r_pad),(0,0), (0,0)))
1124     else:
1125         padstack = segment
1126     for size in [64, 128, 256, 512, padstack.shape[0]]:
1127 
1128         ratio = size/padstack.shape[0]
1129         z_size = np.max([int(padstack[:,:,:100].shape[2]*ratio),1])
1130         resized_imstack = resize(np.array(padstack[:,:,:100]),
     (size,size,z_size), anti_aliasing=False, order =0 )
1131         print(ratio,padstack.shape[2],resized_imstack.shape,
     int(padstack.shape[0]*ratio), (padstack.shape[0]*ratio),
     size*voxelsize/ratio, "diameter of bounding box")
1132         resized_arr = np.array(resized_imstack>0,dtype='uint8')
1133         if size == padstack.shape[0]:
1134             np.save(fname + str(index) + '_' + "fullres" , resized_arr*255)
1135         else:
1136             np.save(fname + str(index) + '_' +str(size) , resized_arr*255)
1137         voxel_conversions.append(voxelsize/ratio)
1138     return padstack.shape, voxel_conversions
1139 
1140 
1141 def make_caps(segments, capsize = 100):
1142     props = measure.regionprops(measure.label(segments[:,:,0]))
1143     prop_max = props[np.argmax([prop.area for prop in props])]
1144     centroid = prop_max.centroid
1145     plt.plot(centroid[1], centroid[0], 'r.')
1146     plt.plot(centroid[1]+ prop_max.major_axis_length//2, centroid[0] , 'r.')
1147     from skimage.draw import circle
1148 
1149     rr, cc= circle(centroid[0], centroid[1],
     (prop_max.major_axis_length*1.03)//2, shape =segments[:,:,0].shape )
1150     circ_image = np.zeros(segments[:,:,0].shape)
1151     circ_image[rr,cc] = 1
1152     start = np.repeat(circ_image[:,:,np.newaxis], capsize, axis = 2)
1153 
1154 
1155 
1156     props = measure.regionprops(measure.label(segments[:,:,-1]))
1157     prop_max = props[np.argmax([prop.area for prop in props])]
1158     centroid = prop_max.centroid
1159     plt.plot(centroid[1], centroid[0], 'r.')
```

```
1160     plt.plot(centroid[1]+ prop_max.major_axis_length//2, centroid[0] , 'r.')
1161     from skimage.draw import circle
1162     rr, cc= circle(centroid[0], centroid[1],
     (prop_max.major_axis_length*1.03)//2,shape =segments[:,:,0].shape)
1163     circ_image = np.zeros(segments[:,:,-1].shape)
1164     circ_image[rr,cc] = 1
1165     end = np.repeat(circ_image[:,:,np.newaxis], capsize, axis = 2)
1166 
1167     return np.array(start, dtype = 'uint8'), np.array(end, dtype = 'uint8')
1168 
1169 
1170 def combine_segments(fname, num, actual_size,  caps = True, folder_index =
     0):
1171     capsize = 100
1172     for resolution in [64,  128, 256, 512,"fullres"]:
1173 
1174         pname_test = fname+'{}_'.format(0) + str(resolution) + '.npy'
1175         segment_part_test =  np.array(np.load(pname_test, allow_pickle =
     True), dtype = 'uint8')
1176         segments = np.zeros((segment_part_test.shape[0],
     segment_part_test.shape[1], 0), dtype = 'uint8')
1177         for part in range(0, num, 100):
1178             pname = fname+'{}_'.format(part) + str(resolution) + '.npy'
1179             segment_part =  np.array(np.load(pname, allow_pickle = True),
     dtype = 'uint8')
1180             segments = np.dstack((segments, segment_part))
1181         if caps:
1182             start,end = make_caps(segments, capsize = capsize)
1183             full_segments = np.dstack((start, segments, end))
1184         if caps:
1185             save_name = fname.split('partial')[0] +"padded"
1186             if resolution  == "fullres":
1187                 assert full_segments.shape == (actual_size[0],
     actual_size[1], actual_size[2] + capsize*2)
1188         else:
1189             save_name = fname.split('partial')[0]
1190             full_segments = segments
1191             if resolution == 'fullres':
1192                 assert full_segments.shape == actual_size
1193         np.save(save_name+'{}.npy'.format(resolution),np.array(full_segments,
     dtype = 'uint8'))
1194         if resolution == 64:
1195             os.makedirs(save_name + '/64datasamples', exist_ok=True)
1196             np.save(save_name+'/64datasamples
     /{}.npy'.format(folder_index),np.array(full_segments, dtype = 'uint8'))
1197 
1198 
1199 def clean_segments(fname):
1200     command =  'rm ' + fname + '*npy'
1201     print(command)
1202     os.system(command)
1203 
1204 def save_sample(folder_index, generated_dir, imstack_all = None, voxelsize =
     None, generated = True, boundary_stack_all = None):
1205     frame_window = 100
```

```
1206     shift = 100
1207     num = None
1208     num = imstack_all.shape[2]
1209     shell_fragment =  'segment_{}_plane_removed_partial_'
1210     fragment =  'segment_{}_partial__'
1211     index = 0
1212     folder_name = generated_dir + '/Part{}/'.format(folder_index)
1213 
1214     os.makedirs(folder_name, exist_ok=True)
1215     while index < num:
1216         pores_total =[]
1217         print("Processing pores, index = " + str(index) + " out of " +
     str(num) + " ..." )
1218         im = np.copy(imstack_all[:,:, index:index + frame_window])
1219         boundary_stack = np.copy(boundary_stack_all[:,:,
     index:index+frame_window])
1220         fractured = np.array(fill_boundary(boundary_stack, im), 'uint8')
1221         pores_removed = np.array(fill_boundary(boundary_stack,
     np.zeros(im.shape)), 'uint8')
1222 
1223         square_shape, voxel_conversions  =
     save_binary_segment(fractured,folder_name + fragment.format(folder_index),
     voxelsize = voxelsize, index = index )
1224         square_shape, _  = save_binary_segment(pores_removed,folder_name +
     shell_fragment.format(folder_index), voxelsize = voxelsize, index = index)
1225         index = index + shift
1226 
1227     combine_segments(os.path.join(folder_name,
     fragment.format(folder_index)), num = num, actual_size = (square_shape[0],
     square_shape[1], imstack_all.shape[2]),  caps = True, folder_index =
     folder_index)
1228     combine_segments(os.path.join(folder_name,
     fragment.format(folder_index)), num = num,actual_size = (square_shape[0],
     square_shape[1], imstack_all.shape[2]),  caps = False, folder_index =
     folder_index)
1229 
1230     clean_segments(os.path.join(folder_name, fragment.format(folder_index)))
1231 
1232     combine_segments(os.path.join(folder_name,
     shell_fragment.format(folder_index)), num = num,actual_size =
     (square_shape[0], square_shape[1], imstack_all.shape[2]),  caps = True,
     folder_index = folder_index)
1233 
1234     combine_segments(os.path.join(folder_name,
     shell_fragment.format(folder_index)), num = num, actual_size =
     (square_shape[0], square_shape[1],imstack_all.shape[2]), caps = False,
     folder_index = folder_index)
1235     clean_segments(os.path.join(folder_name,
     shell_fragment.format(folder_index)))
1236     return voxel_conversions
1237 
1238 def test_sample(sample, generated_dir):
1239     pdir  = generated_dir + '/Part{}/'.format(sample)
1240     npyfiles = [os.path.join(pdir,file).split('.npy')[0] for file in
     os.listdir(pdir) if file.endswith('npy')]
```

```
1241     for npyfile in npyfiles:
1242         sample_dir = './reconstruction/full/testing_binarization/part{}
     /'.format(sample) + npyfile.split('/')[-1]
1243         os.makedirs(sample_dir, exist_ok = True)
1244         npy = np.load(npyfile+'.npy', allow_pickle = "True")
1245         plt.clf()
1246         plt.title(npyfile.split('/')[-1] + "beginning")
1247         plt.imshow(npy[:,:,0])
1248         plt.savefig(sample_dir +'/' + npyfile.split('/')[-1] + 'begin.png')
1249         plt.clf()
1250         plt.title(npyfile.split('/')[-1] + "middle")
1251         plt.imshow(npy[:,:,npy.shape[2]//2])
1252         plt.savefig(sample_dir +'/' + npyfile.split('/')[-1] + 'middle.png')
1253         plt.clf()
1254         print(npyfile.split('/')[-1])
1255         print("done case " + str(sample))
1256     os.makedirs('./reconstruction/full/testing_binarization/part{}
     /tiff_stack'.format(sample), exist_ok= True)
1257     npy = np.load(generated_dir + '/Part{}
     /segment_{}_fullres.npy'.format(sample, sample), allow_pickle = True)
1258     for i in range(npy.shape[2]):
1259         plt.imsave('./reconstruction/full/testing_binarization/part{}
     /tiff_stack'.format(sample) + "/frame{:05}.png".format(i), npy[:,:, i], cmap
     = 'gist_gray')
1260         plt.clf()
1261     os.system('./Fiji.app/ImageJ-linux64 -macro ./reconstruction
     /full/testing_binarization/Video.ijm '+ str(sample))
1262     os.system('rm ' +'./reconstruction/full/testing_binarization/part{}
     /tiff_stack'.format(sample) + "/frame*png")
1263
1264 def main():
1265     tot_frames= 2000
1266     num_frames = 2000
1267     interval = 1
1268     generated_dir  = './reconstruction/results/GeneratedPartSamples'
1269
1270     groundtruth_stack_shape = (566, 571) # example shape from dataset
1271
1272     n_bins = 30
1273     import time
1274     oldtime = time.time()
1275     start = 0#int(sys.argv[1])
1276     for iter in range(start, start+ 100):
1277
1278
1279        for total_frame in range(0, tot_frames, num_frames):
1280             pore_part =
     np.zeros((groundtruth_stack_shape[0],groundtruth_stack_shape[1], num_frames),
     dtype  = 'uint8')
1281
1282             generated_boundary, profile_2d = load_boundary(num_frames =
     num_frames, start = total_frame, pore_part_shape = (pore_part.shape[0],
     pore_part.shape[1]), return_profile=True)
1283
1284
```

```
1285             before_polar_time = time.time()
1286             pore_part,_ = replace_sampling(pore_part, generated_boundary =
     generated_boundary,n_bins = 30)
1287             after_polar_time = time.time()
1288             print(after_polar_time - before_polar_time, "Time taken polar")
1289
1290
1291
1292             before_cartesian_time = time.time()
1293             pore_part,_ = replace_sampling(pore_part, generated_boundary =
     generated_boundary,n_bins = 30)
1294             after_cartesian_time = time.time()
1295             print(after_cartesian_time - before_cartesian_time, "Time taken
     cartesian")
1296             trim_boundary(generated_boundary, pore_part)
1297
1298             plt.close('all')
1299             voxel_to_micron = {}
1300             voxel_to_micron['Resolution'] = ["Name", 64, 128, 256, 512,
     "Full"]
1301             num_samples = 12
1302
1303             conversion = save_sample(iter, voxelsize = 4.87,
     boundary_stack_all=generated_boundary, imstack_all = pore_part, generated_dir
     = generated_dir)
1304
1305             os.system("7z a "+ generated_dir +  "/Part{}.zip ".format(iter) +
     generated_dir + "/Part{}/".format(iter, iter))
1306
1307             if os.path.exists(generated_dir + '/Part{}.zip'.format(iter)):
1308                 os.system("rm " + generated_dir + "/Part{}
     /*npy".format(iter))
1309             print(time.time() - oldtime, "TIME ELAPSED PER ITERATION")
1310             oldtime = time.time()
1311
1312
1313 if __name__ == "__main__":
1314     main()
1315
```