```
# -*- coding: utf-8 -*-
"""
Created on Sat Jul 16 22:45:11 2016
@author: david

Volume class
custom FUNcitons using SimpleITK
https://itk.org/Wiki/SimpleITK/GettingStarted#Generic_Distribution

works only with cropped CT and MRI images (showing only one rod),
both Volumes should have the same PixelSpacing,
 and x and y PixelSpacing shoult be equal
sitk_write() creates .mha file with pixel values corresponding
 to distortion in pixel distance * PixelSpacing (mm)

important to remember:
    sitk.Image saves Volume like this (x,y,z)
    array returned by sitk.GetArrayFromImage(Image)
    is transposed: (z,y,x)

based on:
https://pyscience.wordpress.com/2014/10/19/image-segmentation-with-python-and-SimpleITK/

"""

import numpy as np
from scipy import ndimage
import SimpleITK as sitk
import matplotlib.pyplot as plt
import os
from skimage.draw import circle


class Volume:
    '''
    Create a Volume (SimpleITK.Image with convenient properties and functions)
    recommended use:
    create new Volume (optional use denoise=True)
    Volume.getThresholds()

    Parameters
    ----------
    path : string_like
        directory containing DICOM data
    method : string_like, recommended
        either "CT or "MR", used for automatic calculations
    denoise : bool, optional
        If true, the imported data will be denoised using
        SimpleITK.CurvatureFlow(image1=self.img,
                                timeStep=0.125,
                                numberOfIterations=5)
    ref : int, optional
        slice used to make calculations (idealy isocenter) e.g. thresholds
        all plots show this slice
        by default it is set to be in the middle of the image (z-axis)
    resample : int, optional
        resample rate, becomes part of title
    seeds : array_like (int,int,int), optional
        coordinates (pixel) of points inside rod, used for segmentation
        by default list of brightest pixel in each slice
    radius: double, optional
        overrides radius value (default CT:4mm, MR:2mm)
    spacing: double, optional
        by default SitpleITK.img.GetSpacing is used to find relation of pixels
        to real length (in mm)
    skip: int, optional
        neglecting first 'skip' number of slices
    leave: int, optional
        neglects last 'leave' number of slices
    rotate: bool, optional
        if True: mirrors x- & z-axis, effectively rotating the image by 180 deg
        (looked at from above), this is applied after skip&size
```

```
73      '''
74      def __init__(self, path=None, method=None, denoise=False, ref=None,
75                   resample=False, seeds='auto', radius=0, spacing=0, skip=0,
76                   leave=False, rotate=False):
77          if(path is None):
78              print("Error: no path given!")
79          else:
80              self.path = path
81              self.method = method
82              self.denoise = denoise
83              self.resample = resample
84              self.centroid = False
85              self.mask = False
86              self.masked = False
87              self.title = method
88              self.radius = radius
89              self.bestRadius = 0
90              self.lower = False
91              self.upper = False
92
93              file_no = len([name for name in os.listdir(path)
94                             if os.path.isfile(os.path.join(path, name))])
95              size = file_no - skip - leave
96              if size <= 0:
97                  print("There nothing left to load after skipping {} file(s) "
98                        "and ignoring the last {} files.".format(skip,leave))
99                  print("The directory only contains {} files!".format(file_no))
100             else:
101
102                 print("Import {} DICOM Files from: {}\n".format(size, path))
103                 shortened_img = sitk_read(path, denoise)[
104                                       :, :, skip:(file_no + skip - leave)]
105                 if rotate is True:
106                     self.img = shortened_img[::-1,:,::-1]
107                 else:
108                     self.img = shortened_img
109
110                 if (self.img and self.denoise):
111                     a = self.title
112                     self.title = a + " denoised"
113
114                 if resample:
115                     a = self.title
116                     self.title = a + ", x" + str(resample)
117
118                 self.xSize, self.ySize, self.zSize = self.img.GetSize()
119                 if spacing == 0:
120                     self.xSpace, self.ySpace, self.zSpace = self.img.GetSpacing()
121
122                 if type(ref) == int:
123                     self.ref = ref
124                 else:
125                     self.ref = int(self.zSize / 2)
126
127                 # niceSlice used to remember which slices show irregularities such
128                 # as parts of plastic pane (CT)
129                 # and should therefore not be used to calculate COM, dice, etc.
130                 self.niceSlice = np.ones((self.zSize, 1), dtype=bool)
131                 self.maxBrightness = np.zeros((self.zSize, 1))
132                 self.meanBrightness = np.zeros((self.zSize, 1))
133                 arr = sitk.GetArrayFromImage(self.img)
134                 average = np.average(arr[ref])
135     #               print("\nAverage @ ref: ", average)
136                 for index in range(self.zSize):
137                     # save value of brightest pixel in each slice
138                     self.maxBrightness[index] = arr[index].max()
139                     self.meanBrightness[index] = np.average(arr[index])
140                     # if average value of slice differs too much -> badSlice
141                     # difference between ref-Slice and current chosen arbitratry
142                     # seems to be big enough not to detect air bubble in MRI
143                     # entire air block (no liquid) should be recognised, though.
144                     # small enough to notice plastic pane
```

```
145                     if np.absolute(self.meanBrightness[index] - average) > 40:
146                         print("Irregularities detected in slice {}".format(index))
147                         self.niceSlice[index] = False
148                         # maybe also set slice prior and after current slice as
149                         # self.niceSlice[index+1] = self.niceSlice[index+1] = False
150                         # because small changes happening around irregularities
151                         # might not have been big enough for detection, but already
152                         # leading to false calculations?
153 
154                 if type(seeds) == list:
155                     self.seeds = seeds
156                 elif seeds == 'auto':
157                     self.seeds = []
158                     for index in range(self.zSize):
159                         yMax = int(arr[index].argmax() / self.xSize)
160                         xMax = arr[index].argmax() - yMax*self.xSize
161                         if self.niceSlice[index] == True:
162                             self.seeds.append((xMax, yMax, index))
163     #                    print("{}: found max at ({},{})"
164     #                          .format(index, xMax, yMax))
165 
166     def show(self, pixel=False, interpolation=None, ref=None, save=False):
167         '''
168         plots ref slice of Volume
169 
170         Parameters
171         ----------
172         pixel: bool, optional
173             if True, changes axis from mm to pixels
174         interpolation: "string", optional, default: 'nearest'
175             using build-in interpolation of matplotlib.pyplot.imshow
176             Acceptable values are 'none', 'nearest', 'bilinear', 'bicubic',
177             'spline16', 'spline36', 'hanning', 'hamming', 'hermite', 'kaiser',
178             'quadric', 'catrom', 'gaussian', 'bessel', 'mitchell', 'sinc',
179             'lanczos'
180         ref: int, optional
181             slice to be plotted instead of self.ref (default: 0)
182         '''
183 
184         if ref is None:
185             ref = self.ref
186 
187         if interpolation is None:
188             a = 'nearest'
189 
190         extent = None
191         if pixel is False:
192             extent = (-self.xSpace/2, self.xSize*self.xSpace - self.xSpace/2,
193                       self.ySize*self.ySpace - self.ySpace/2, -self.ySpace/2)
194 # The location, in data-coordinates, of the lower-left and upper-right corners
195 # (left, right, bottom, top)
196 
197         sitk_show(img=self.img, ref=ref, extent=extent, title=self.title,
198                   interpolation=a, save=save)
199 
200     def showSeed(self, pixel=False, interpolation='nearest', ref=None, save=False):
201         '''
202         plots slice containing seed
203 
204         Parameters
205         ----------
206         pixel: bool, optional
207             if True, changes axis from mm to pixels
208         interpolation: "string", optional, default: 'nearest'
209             using build-in interpolation of matplotlib.pyplot.imshow
210             Acceptable values are 'none', 'nearest', 'bilinear', 'bicubic',
211             'spline16', 'spline36', 'hanning', 'hamming', 'hermite', 'kaiser',
212             'quadric', 'catrom', 'gaussian', 'bessel', 'mitchell', 'sinc',
213             'lanczos'
214         ref: int, optional
215             slice of seed to be plotted instead of self.ref (default: zSize/2)
216         '''
```

```
217             if ref is None:
218                 ref = self.ref
219 
220             if type(self.seeds[ref]) != tuple:
221                 print("No seed found @ slice {}".format(ref))
222                 return None
223 
224             x, y = -1,-1
225             extent = None
226             if pixel is False:
227                 extent = (-self.xSpace/2, self.xSize*self.xSpace - self.xSpace/2,
228                           self.ySize*self.ySpace - self.ySpace/2, -self.ySpace/2)
229                 x = (self.seeds[ref][0] * self.xSpace)
230                 y = (self.seeds[ref][1] * self.xSpace)
231             else:
232                 x, y, z = self.seeds[ref]
233 
234             arr = sitk.GetArrayFromImage(self.img)
235             fig = plt.figure()
236             plt.set_cmap("gray")
237             plt.title(self.title + ", seed @ {}".format(self.seeds[ref]))
238 
239             plt.imshow(arr[ref, :, :], extent=extent, interpolation=interpolation)
240             plt.scatter(x, y)
241             plt.show()
242             if save != False:
243                 fig.savefig(str(save) + ".png")
244 
245         def getThresholds(self, pixelNumber=0, scale=1):
246             '''
247             Calculates threshold based on number of pixels representing rod.
248             If no pixelNumber is given, self.radius is used to get estimated
249             pixelNumber. If self.raduis == 0: use method to get raduis
250              All calculations based on ref-slice.
251 
252             approx. number of pixels being part of rod:
253             pn = realRadius^2 * pi / pixelSpacing^2
254 
255             Parameters
256             ----------
257             pixelNumber: int, optional
258                 if 0, uses self.radis to calculate pixelnumber
259                 if self.radius also 0, uses self.method instead (CT: 4mm, MR: 2mm)
260             scale: double, optional
261                 factor altering pixelNumber
262 
263             Returns
264             -------
265             lower and upper threshold value: (double, double)
266             '''
267 
268             if pixelNumber == 0:
269                 if self.radius != 0:
270                     realRadius = self.radius
271                 else:
272                     if self.method == "CT":
273                         realRadius = 4
274                     if self.method == "MR":
275                         realRadius = 2
276                     if self.method != "MR" and self.method != "CT":
277                         print("method is unknown, please set pixelNumber!")
278                         return None
279                 pixelNumber = (np.power(realRadius, 2) * np.pi
280                                / np.power(self.xSpace, 2) * scale)
281 
282             pn = int(pixelNumber)
283             arr = sitk.GetArrayFromImage(self.img)
284             self.upper = np.double(arr.max())
285 
286         #    hist, bins = np.histogram(arr[self.ref, :, :].ravel(), bins=100)
287         # alternatively, increase number of bins for images with many pixels
288             hist, bins = np.histogram(arr[self.ref, :, :].ravel(), bins=int(pn*2))
```

```
289           self.lower = np.double(bins[np.argmax((np.cumsum(hist[::-1]) < pn)[::-1])])
290           print("number of pixels (pn): {}\n "
291                 "lower: {}\n "
292                 "upper: {}".format(pn, self.lower, self.upper))
293
294           return (self.lower, self.upper)
295
296       def getCentroid(self, threshold='default', pixelNumber=0, scale=1,
297                       percentLimit=False, iterations=5, top = 1,
298                       plot=False, save=False):
299           '''
300           Calculates centroid, either by setting threshold or percentLimit
301
302           Parameters
303           ----------
304           threshold: float or 'auto', default='auto'
305               if 'auto': uses getThreshold(pixelnumber, scale) and then
306                   sitk_centroid(threshold=self.lower)
307                   sets self.lower and self.upper
308           percentLimit: float from 0 to 1 (or "auto" =experimental)
309               if percentLimit is True: used instead of threshold method
310               if 'auto': makes 5 iterations by default, uses getThreshold()
311                   and getDice(), but does NOT set self.mask
312                   sets self.lower and self.upper
313           plot, save: bool, optional
314               plot and save iteration (percentLimit='auto')
315
316           Returns
317           -------
318           self.centroid: numpy.ndarray
319           '''
320
321           if ((threshold is False and percentLimit is False)
322               or (percentLimit == "auto" and threshold is not False
323                   and threshold != 'default')):
324               print("Please use percentLimit or threshold! "
325                     "(default setting: threshold = 'auto')")
326               return None
327
328           if ((percentLimit == "auto" and threshold is False) or
329               (percentLimit == "auto" and threshold == 'default')):
330               # EXPERIMENTAL!!!
331               # looks at whole range of possible percentLimits
332               # reduces range by finding out which half yields higher result
333               # starts at A=25% and B=75% of all pixels
334               # if DC(A) > DC(B): next values come from lower half (0-50%)
335               # else: upper half (50-100%)
336               # calculates 5 centroids with different percentLimits
337               # gets dice coefficient for each centroid percentLimit combination
338               # returns best result
339
340               print("\n\n")
341               arr = sitk.GetArrayFromImage(self.img)
342               direction = np.zeros(iterations)
343               left = np.zeros(iterations)
344               right = np.zeros(iterations)
345               left[0] = 0
346               right[0] = top
347               guess = np.zeros(iterations)
348               guess[0] = (left[0]+right[0])/2
349               thresholdsA = np.zeros((iterations,2))
350               thresholdsB = np.zeros((iterations,2))
351               centroidScoreA = np.zeros(iterations)
352               centroidScoreB = np.zeros(iterations)
353               centroidsA = np.zeros((iterations, self.zSize, 2))
354               centroidsB = np.zeros((iterations, self.zSize, 2))
355               diceA = np.zeros((iterations, self.zSize, 1))
356               diceB = np.zeros((iterations, self.zSize, 1))
357               for index in range(iterations):
358                   print("    ITERATION #{}, current guess: ~{:.4f}\n"
359                         "A @ ~{:.4f}%"
360                         .format(index, guess[index] *100,
```

```
361                                 (guess[index] + left[index]) / 2*100))
362                 thresholdsA[index] = self.getThresholds(pixelNumber=self.xSize
363                     * self.ySize * (guess[index] + left[index]) / 2)
364                 # create mask including all pixels relevant for guess
365                 maskA = sitk.ConnectedThreshold(image1=self.img,
366                                                 seedList=self.seeds,
367                                                 lower=self.lower,
368                                                 upper=self.upper,
369                                                 replaceValue=1)
370                 # shift values so that they're all positive and apply mask
371                 maskedA2 = sitk_applyMask(self.img - arr.min(), maskA)
372                 # now shift values back, this results in all masked pixels
373                 # to be assigned the minimum value
374                 maskedA = maskedA2 + arr.min()
375                 # use all pixels above minimum value for centroid:
376                 centroidsA[index] = (self.xSpace * sitk_centroid(maskedA,
377                                           ref=self.ref, threshold=arr.min() + 1))
378                 diceA[index] = self.getDice(centroidsA[index], maskA)
379                 # all irregular Slices result in DC of -1:
380                 diceA[index][np.where(self.niceSlice == False)] = -1
381                 # for the final DC score it will look only at the niceSlices:
382                 centroidScoreA[index] = np.average(diceA[index,
383                                                    self.niceSlice == True])
384                 # alternatively we could look at those with DC values > 0:
385 #                centroidScoreA[index] = np.average(diceA[index,diceA[index]>-1])
386                 # or just take all into account, regardless of their value
387 #                centroidScoreA[index] = np.average(diceA[index])
388
389
390                 print("\nB @ ~{:.4f}%"
391                       .format((guess[index] + right[index]) / 2*100))
392                 thresholdsB[index] = self.getThresholds(pixelNumber=self.xSize
393                     * self.ySize * (guess[index] + right[index]) / 2)
394                 maskB = sitk.ConnectedThreshold(image1=self.img,
395                                                 seedList=self.seeds,
396                                                 lower=self.lower,
397                                                 upper=self.upper,
398                                                 replaceValue=1)
399                 maskedB2 = sitk_applyMask(self.img - arr.min(), maskB)
400                 maskedB = maskedB2 + arr.min()
401                 centroidsB[index] = (self.xSpace * sitk_centroid(maskedB,
402                                           ref=self.ref, threshold=arr.min() + 1))
403                 diceB[index] = self.getDice(centroidsB[index], maskB)
404                 # all irregular Slices result in DC of -1:
405                 diceB[index][np.where(self.niceSlice == False)] = -1
406                 # for the final DC score it will look only at the niceSlices:
407                 centroidScoreB[index] = np.average(diceB[index,
408                                                    self.niceSlice == True])
409 #                centroidScoreB[index] = np.average(diceB[index,diceB[index]>-1])
410                 # or just take all into account, regardless of their value
411 #                centroidScoreB[index] = np.average(diceB[index])
412
413                 if (centroidScoreA[index] < centroidScoreB[index]
414                     and index < iterations-1):
415
416                     left[index+1] = guess[index]
417                     right[index+1] = right[index]
418                     guess[index+1] = (left[index+1] + right[index+1]) / 2
419                     direction[index] = 1
420                 elif (centroidScoreA[index] > centroidScoreB[index]
421                       and index < iterations-1):
422
423                     right[index+1] = guess[index]
424                     left[index+1] = left[index]
425                     guess[index+1] = (left[index+1] + right[index+1]) / 2
426                     direction[index] = -1
427                 elif (centroidScoreA[index] == centroidScoreB[index]
428                       and index < iterations-1):
429
430                     right[index+1] = (guess[index] + right[index]) / 2
431                     left[index+1] = (guess[index] + left[index]) / 2
432                     guess[index+1] = guess[index]
```

```
433                 else:
434                     break
435 
436                 print("--------------------------")
437                 print("next guess (#{}): ~{:.4f}% \n\n\n\n"
438                       .format(index+1, guess[index+1]*100))
439 
440 
441             if centroidScoreA.max() > centroidScoreB.max():
442                 self.centroid = centroidsA[centroidScoreA.argmax()]
443                 self.lower, self.upper = thresholdsA[centroidScoreA.argmax()]
444                 self.dice = diceA[centroidScoreA.argmax()]
445                 self.diceAverage = centroidScoreA.max()
446                 print("\nmax dice-coefficient obtained "
447                       "during iteration #{}: ~{:.4f}"
448                       .format(centroidScoreA.argmax(), centroidScoreA.max()))
449             elif (centroidScoreA.max() <= centroidScoreB.max()
450                   and centroidScoreB.max() != 0):
451 
452                 self.centroid = centroidsB[centroidScoreB.argmax()]
453                 self.lower, self.upper = thresholdsB[centroidScoreB.argmax()]
454                 self.dice = diceB[centroidScoreB.argmax()]
455                 self.diceAverage = centroidScoreB.max()
456                 print("\nmax dice-coefficient obtained "
457                       "during iteration #{}: ~{:.4f}"
458                       .format(centroidScoreB.argmax(), centroidScoreB.max()))
459             else:
460                 return None
461 
462             print("\n\n-o-o-o-o-- Summary: --o-o-o-o-\n")
463             for index in range(np.size(guess)):
464                 print("\n  Iteration #{}: range({},{})"
465                       .format(index, left[index]*100, right[index]*100))
466                 if centroidScoreA[index] > centroidScoreB[index]:
467                     print("A @ {}%, Score: {}  <---"
468                           .format((guess[index] + left[index]) / 2*100,
469                                   centroidScoreA[index]))
470                     print("B @ {}%, Score: {}"
471                           .format((guess[index] + right[index]) / 2*100,
472                                   centroidScoreB[index]))
473                 if centroidScoreA[index] < centroidScoreB[index]:
474                     print("A @ {}%, Score: {}"
475                           .format((guess[index] + left[index]) / 2*100,
476                                   centroidScoreA[index]))
477                     print("B @ {}%, Score: {}  <---"
478                           .format((guess[index] + right[index]) / 2*100,
479                                   centroidScoreB[index]))
480                 if centroidScoreA[index] == centroidScoreB[index]:
481                     print("A @ {}% same as for B @ {}%, Score: = {}"
482                           .format((guess[index] + left[index]) / 2*100,
483                                   (guess[index] + right[index]) / 2*100,
484                                   centroidScoreA[index]))
485 
486 
487             if plot == True:
488                 fig = plt.figure()
489                 for index in range(iterations):
490                     if guess[index] > 0 and centroidScoreA[index] > 0:
491                         plt.plot((guess[index] + left[index]) / 2*100,
492                                  centroidScoreA[index], 'bo')
493                     if guess[index] > 0 and centroidScoreB[index] > 0:
494                         plt.plot((guess[index] + right[index]) / 2*100,
495                                  centroidScoreB[index], 'go')
496                 plt.show()
497                 if save != False:
498                     fig.savefig(str(save) + ".png")
499 
500 
501         if percentLimit != "auto" and percentLimit is not False:
502             self.centroid = (self.xSpace * sitk_centroid(self.img, ref=self.ref,
503                                                          percentLimit=percentLimit))
504 
```

```
505             if ((threshold == 'auto' or threshold == 'default')
506                 and percentLimit is False):
507 
508                 self.getThresholds(pixelNumber=pixelNumber, scale=scale)
509                 self.centroid = (self.xSpace * sitk_centroid(self.img, ref=self.ref,
510                                                              threshold=self.lower))
511             if ((threshold != "auto" and threshold != 'default')
512                 and threshold is not False and percentLimit is False):
513 
514                 self.centroid = (self.xSpace * sitk_centroid(self.img, ref=self.ref,
515                                                              threshold=threshold))
516             for index in range(self.zSize):
517                 if not self.niceSlice[index]:
518                     self.centroid[index] = -1, -1
519                 if self.centroid[index,0] < 0 or self.centroid[index,1] < 0 :
520                     self.centroid[index] = -1, -1
521             print("\n\n")
522             return self.centroid
523 
524 
525         def showCentroid(self, img=None, com2=0, title=None, pixel=False,
526                          interpolation='nearest', ref=None, save=False):
527             '''
528             shows slice with centroid coordinates
529 
530             Parameters
531             ----------
532             img: SimpleITK.img, optional
533                 slice of this volume will be shown
534                 default: self.img
535             com2: numpy.ndarray
536                 supposed to be of same length as img
537                 will also be shown in plot alongside self.centroid
538                 helps creating nice plot for comparing COM-shift
539             pixel: bool, optional
540                 if True, changes axis from mm to pixels
541             interpolation: "string", optional, default: 'nearest'
542                 using build-in interpolation of matplotlib.pyplot.imshow
543                 Acceptable values are 'none', 'nearest', 'bilinear', 'bicubic',
544                 'spline16', 'spline36', 'hanning', 'hamming', 'hermite', 'kaiser',
545                 'quadric', 'catrom', 'gaussian', 'bessel', 'mitchell', 'sinc',
546                 'lanczos'
547             ref: int, optional
548                 slice to be plotted instead of self.ref (default: 0)
549             save: string, optional
550                 save plot as save + ".png"
551             '''
552 
553             if self.centroid is False:
554                 print("Volume has no centroid yet. use Volume.getCentroid() first!")
555                 return None
556 
557             if title is None:
558                 title = self.title
559             if ref is None:
560                 ref = self.ref
561             if img is None:
562                 img = self.img
563 
564             if pixel is False:
565                 extent = (-self.xSpace/2, self.xSize*self.xSpace - self.xSpace/2,
566                           self.ySize*self.ySpace - self.ySpace/2, -self.ySpace/2)
567                 sitk_centroid_show(img=img, com=self.centroid, com2=com2,
568                                    extent=extent, save=save, title=title,
569                                    interpolation=interpolation, ref=ref)
570             else:
571                 sitk_centroid_show(img=img, com=self.centroid/self.xSpace,
572                                    com2=com2/self.xSpace, save=save, title=title,
573                                    interpolation=interpolation, ref=ref)
574 
575         def getMask(self, lower=False, upper=False):
576 
```

```
577             if lower is False and self.lower is not False:
578                 lower = self.lower
579             if upper is False and self.upper is not False:
580                 upper = self.upper
581 
582             if lower is False:
583                 print("Lower threshold missing!")
584                 return None
585             if upper is False:
586                 print("Upper threshold missing!")
587                 return None
588 
589             self.mask = sitk_getMask(self.img, self.seeds, upper, lower)
590             return self.mask
591 
592         def applyMask(self, mask=0, replaceArray=False, scale=1000):
593             if mask == 0:
594                 if self.mask:
595                     mask = self.mask
596                 else:
597                     print("Volume has no mask yet. use Volume.getMask() first!")
598                     return None
599 
600             self.masked = sitk_applyMask(self.img, mask, scale=scale,
601                                          replaceArray=replaceArray)
602 
603             return self.masked
604 
605         def showMask(self, interpolation=None, ref=None, save=False, pixel=False):
606             if self.mask is False:
607                 print("Volume has no mask yet. use Volume.getMask() first!")
608                 return None
609 
610             if ref is None:
611                 ref = self.ref
612 
613             if interpolation is None:
614                 interpolation = 'nearest'
615 
616             title = self.title + ", mask"
617 
618             extent = None
619             if pixel is False:
620                 extent = (-self.xSpace/2, self.xSize*self.xSpace - self.xSpace/2,
621                           self.ySize*self.ySpace - self.ySpace/2, -self.ySpace/2)
622 
623             sitk_show(img=self.mask, ref=ref, title=title, extent=extent,
624                           interpolation=interpolation, save=save)
625 
626         def showMasked(self, interpolation=None, ref=None, save=False, pixel=False):
627             if self.masked is False:
628                 print("Volume has not been masked yet. use Volume.applyMask() first!")
629                 return None
630             if ref is None:
631                 ref = self.ref
632 
633             if interpolation is None:
634                 interpolation = 'nearest'
635 
636             title = self.title + ", masked"
637 
638             extent = None
639             if pixel is False:
640                 extent = (-self.xSpace/2, self.xSize*self.xSpace - self.xSpace/2,
641                           self.ySize*self.ySpace - self.ySpace/2, -self.ySpace/2)
642 
643             sitk_show(img=self.masked, ref=ref, title=title, extent=extent,
644                       interpolation=interpolation, save=save)
645 
646         def getDice(self, centroid=None, mask=None, iterations=15,
647                     CT_guess=(3.5,5.5), MR_guess=(1.5,4.5),
648                     show=False, showAll=False, plot=False, save=False, pixel=False):
```

```
        '''
        Calculates dice coefficient ('DC') and average DC of the volume
        if iterations > 0: varies radius and finds DC with best average DC
        else: if self.raduis == 0: use method to get raduis for DC calculation
        average DC is mean value of all slices, except those with DC of -1

        slice DC is set to -1 if centroid lies outside image or reference
        circle exceeds image

        Parameters
        ----------
        centroid: numpy.ndarray, optional
            centroid to place circles in instead of self.centroid
        mask: SimpleITK image, optional
            binary image to calculate DC of instead of self.mask
        iterations: int, optional
        show: int, optional
            shows circle used to compare mask to in slice nr. "show"
        showAll: bool, optional
            shows all circles tried during iteration
        plot, save: bool, optional
            plot and save iteration

        Returns
        -------
        self.dice: numpy.ndarray
        '''
        if centroid is None:
            centroid = self.centroid
        # to get from mm to pixel coordinates:
        com = centroid / self.xSpace
        if mask is None:
            if self.mask is False:
                self.getMask()
            mask = self.mask

        extent = None
        if pixel is False:
            extent = (-self.xSpace/2, self.xSize*self.xSpace - self.xSpace/2,
                      self.ySize*self.ySpace - self.ySpace/2, -self.ySpace/2)

        if self.radius != 0:
            print("{}_x{}.radius is {} and will therefore be used to calculate DC."
                  .format(self.method, self.resample, self.radius))
            self.dice = sitk_dice_circle(img=mask, centroid=com, extent=extent,
                                         radius=self.radius/self.xSpace, show=show)

        if self.radius == 0 and iterations == 0:
            if self.method == "CT":
                self.dice = sitk_dice_circle(img=mask, centroid=com, extent=extent,
                                             radius=4/self.xSpace, show=show)
            if self.method == "MR":
                self.dice = sitk_dice_circle(img=mask, centroid=com, extent=extent,
                                             radius=2/self.xSpace, show=show)
            if self.method != "CT" and self.method != "MR":
                print("Unknown method!")
                return None

        if self.radius == 0 and iterations > 0:
            low, up = 0, 0
            if self.method == "CT":
                low, up = CT_guess
                radii = np.linspace(low, up, num=iterations) / self.xSpace
            if self.method == "MR":
                low, up = MR_guess
                radii = np.linspace(low, up, num=iterations) / self.xSpace
            if self.method != "CT" and self.method != "MR":
                # radii = np.linspace(1.5, 4.5, num = 11)
                print("Unknown method!")
                return None

            DCs = np.zeros(len(radii))
```

```
            for index, r in enumerate(radii, start=0):
                dice = sitk_dice_circle(img=mask, centroid=com, radius=r,
                                        show=showAll, extent=extent)
#                DCs[index] = np.average(dice)
#                DCs[index] = np.average(dice[dice>-1])
                DCs[index] = np.average(dice[self.niceSlice==True])

            if plot == True:
                fig = plt.figure()
#                plt.ylim(ymin=0.6, ymax=1)
#                plt.xlim(xmin=(low-.1), xmax=(up+.1))
                plt.plot(radii*self.xSpace, DCs, '+-')
                if save is not False:
                    fig.savefig(str(save) + ".png")

            self.dice = sitk_dice_circle(img=mask, centroid=com, show=show,
                                         extent=extent, radius=radii[DCs.argmax()])
            self.bestRadius = radii[DCs.argmax()]*self.xSpace
            print("max dice-coefficient obtained for {} when "
                  "compared to circle with radius = {}"
                  .format(self.method, self.bestRadius))

# Instead of using only DC > 0 for the average:
#        self.diceAverage = np.average(self.dice[self.dice>-1])
# we include the "-1" values in our overall DC average:
        self.diceAverage = np.average(self.dice)
        print("dice-coefficient average for the whole volume is: {:.4f}"
              .format(self.diceAverage))
        return self.dice



def sitk_read(directory, denoise=False):
    '''
    returns DICOM files as "SimpleITK.Image" data type (3D)
    if denoise is True: uses SimpleITK to denoise data
    '''
    reader = sitk.ImageSeriesReader()
    filenames = reader.GetGDCMSeriesFileNames(directory)
    reader.SetFileNames(filenames)
    if denoise:
        print("\n...denoising...")
        imgOriginal = reader.Execute()
        return sitk.CurvatureFlow(image1=imgOriginal,
                                  timeStep=0.125,
                                  numberOfIterations=5)
    else:
        return reader.Execute()


def sitk_write(image, output_dir='', filename='3DImage.mha'):
    '''
    saves image as .mha file
    '''
    output_file_name_3D = os.path.join(output_dir, filename)
    sitk.WriteImage(image, output_file_name_3D)


def sitk_show(img, ref=0, extent=None, title=None, interpolation='nearest', save=False):
    """
    shows plot of img at z=ref
    """
    arr = sitk.GetArrayFromImage(img)
    fig = plt.figure()
    plt.set_cmap("gray")
    if title:
        plt.title(title)

    plt.imshow(arr[ref], extent=extent, interpolation=interpolation)
    plt.show()
    if save != False:
        fig.savefig(str(save) + ".png")
```

```
793
794 def sitk_centroid(img, ref=False, percentLimit=False, threshold=False):
795     '''
796     returns array with y&x coordinate of centroid for every slice of img
797     centroid[slice, y&x-coordinate]
798     if no pixel has value > threshold:
799         centroid x&y-coordinate of that slice = -1,-1
800     '''
801     if ((threshold is False and percentLimit is False)
802         or (threshold is True and percentLimit is True)):
803
804         print("Please set either percentLimit or threshold!")
805         return None
806
807     arr = sitk.GetArrayFromImage(img)
808     z, y, x = np.shape(arr)
809     # create array with centroid coordinates of rod in each slice
810     com = np.zeros((z, 2))
811
812     if ref is False:
813         ref = int(z/2)
814
815     if threshold is False:
816 #         hist, bins = np.histogram(arr[ref, :, :].ravel(), density=True, bins=100)
817      #  alternatively, increase number of bins for images with many pixels
818         hist, bins = np.histogram(arr[ref, :, :].ravel(), density=True, bins=int(y*x))
819         threshold = bins[np.concatenate((np.array([0]), np.cumsum(hist)))
820                          * (bins[1] - bins[0]) > percentLimit][0]
821
822     for index in range(z):
823         if arr[index].max() > threshold:
824             # structuring_element=[[1,1,1],[1,1,1],[1,1,1]]
825             segmentation, segments = ndimage.label(arr[index] > threshold)
826             # print("segments: {}".format(segments))
827             # add ', structuring_element' to label() for recognising
828             # diagonal pixels as part of object
829             com[index, ::-1] = ndimage.center_of_mass(arr[index, :, :]-threshold,
830                                                       segmentation)
831             # add ', range(1,segments)' to center_of_mass for list of centroids
832             # in each slice (multiple rods!)
833         else:
834             com[index] = (-1,-1)
835
836     return com
837
838 def sitk_centroid_show(img, com, com2=0, extent=None, title=None,
839                        save=False, interpolation='nearest', ref=0):
840
841         arr = sitk.GetArrayFromImage(img)
842         fig = plt.figure()
843         plt.set_cmap("gray")
844         if title:
845             plt.title(title + ", centroid")
846         x = y = 0
847         plt.imshow(arr[ref], extent=extent, interpolation=interpolation)
848         if type(com2) == np.ndarray:
849             x = [com[ref,0],com2[ref,0]]
850             y = [com[ref,1],com2[ref,1]]
851         else:
852             x, y = com[ref]
853         plt.scatter(x, y, c=['b','r'])
854         plt.show()
855         if save != False:
856             fig.savefig(str(save) + ".png")
857
858 def sitk_coordShift(first, second):
859     '''
860     returns array with difference of y&x coordinates for every
861     centroid[slice, y&x-coordinate]
862     '''
863     if (np.shape(first) == np.shape(second)
864         and np.shape((np.shape(first))) == (2,)):
```

```
865
866         z, xy = np.shape(first)
867         diff = np.zeros((z, 2))
868         for slice in range(z):
869             if (    first[slice,0]==-1 or first[slice,1]==-1
870                     or second[slice,0]==-1 or second[slice,0]==-1):
871                 diff[slice, 0] = diff[slice, 1] = -1
872             else:
873                 diff[slice, 0] = first[slice, 0] - second[slice, 0]
874                 diff[slice, 1] = first[slice, 1] - second[slice, 1]
875         return diff
876     else:
877         print("Wrong shape! sitk_coordShift returned 'False'")
878         return False
879
880
881 def sitk_coordDist(shift):
882     '''
883     calculates norm for each entry of array
884     returns array with list of calculated values
885     '''
886     if np.shape(shift)[1] != 2:
887         print("shift has wrong shape!")
888         return False
889
890     dist = np.zeros((len(shift), 1))
891     for slice in range(len(shift)):
892         if shift[slice,0] == -1 or shift[slice,1] == -1:
893             dist[slice,:] =-1
894         else:
895             dist[slice, :] = np.linalg.norm(shift[slice, :])
896     return dist
897
898
899 def sitk_getMask(img, seedList, upper, lower):
900     '''
901     creates new SimpleITK.img using a SimpleITK segmentation function
902     which is made up by all pixels with values between upper and lower and
903     connected to a seed from seedList.
904     Returns binary image (SimpleITK.img)
905     '''
906
907     if seedList is False:
908         print("no seeds given!")
909         return None
910
911     return sitk.ConnectedThreshold(image1=img, seedList=seedList,
912                                    lower=lower, upper=upper,
913                                    replaceValue=1)
914
915
916 def sitk_applyMask(img, mask, replaceArray=False, scale=1000, errorValue=-1):
917     '''
918     masks img (SimpleITK.Image) using mask (SimpleITK.Image)
919     if a replaceArray is given, the values*scale (default scale=1000) of the
920     array will be used as pixel intensity for an entire slice each
921     '''
922     if img.GetSize() != mask.GetSize():
923         print(mask.GetSize())
924         print(img.GetSize())
925
926         print("mask and image are not the same size!")
927         return False
928
929     arr = sitk.GetArrayFromImage(img)
930     maskA = sitk.GetArrayFromImage(mask)
931     xSize, ySize, zSize = img.GetSize()
932
933     imgMaskedA = (arr - arr.min() + 1)*maskA
934
935     if (np.shape(replaceArray) == (img.GetDepth(), 1)
936         or np.shape(replaceArray) == (img.GetDepth(),)):
```

```
937
938             for slice in range(zSize):
939                 imgMaskedA[slice][imgMaskedA[slice] != 0] = replaceArray[slice]*scale
940                 imgMaskedA[slice][imgMaskedA[slice] < 0] = errorValue
941
942
943         return sitk.GetImageFromArray(imgMaskedA)
944
945
946     def sitk_dice_circle(img, centroid, radius=2.1, show=False, extent=None,
947                          interpolation='nearest', save=False):
948         """
949         Dice coefficient, inspired by
950          Medpy (http://pythonhosted.org/MedPy/_modules/medpy/metric/binary.html)
951
952         Computes the Dice coefficient (akas Sorensen index) between a binary
953         object in an image and a circle.
954
955         The metric is defined as:
956
957             DC=\frac{2|A\cap B|}{|A|+|B|}
958
959         where A is the first and B the second set of samples (here: binary objects)
960
961         Parameters
962         ----------
963         input_umg : SimpleITK.Image
964             Input data containing objects. Can be any type but will be converted
965             into binary: background where 0, object everywhere else.
966         centroid : array_like
967             array with coordinates for circle centre
968         radius : float
969             radius for creating reference circles
970
971         Returns
972         -------
973         DC : array_like
974             The Dice coefficient between the object(s) in ```input``` and the
975             created circles. It ranges from 0 (no overlap) to 1 (perfect overlap).
976             if centroid coordinates + radius would create circle exceeding image
977             size: DC of this slice = -1
978             Other errors occuring during the calculation should also result in -1
979         """
980
981         xSize, ySize, zSize = img.GetSize()
982         xSpace, ySpace, zSpace = img.GetSpacing()
983         profile = np.zeros((zSize, ySize, xSize), dtype=np.uint8)
984         DC = np.zeros((zSize, 1))
985         for slice in range(zSize):
986             if (centroid[slice,0]+radius < xSize and centroid[slice, 1]+radius < ySize
987                 and centroid[slice,0]-radius > 0 and centroid[slice, 1]-radius > 0):
988
989                 rr, cc = circle(centroid[slice, 0], centroid[slice, 1], radius, (xSize,ySize))
990                 profile[slice, cc, rr] = 1
991             else:
992                 # print("something's fishy!")
993                 DC[slice]= -1
994
995         input = sitk.GetArrayFromImage(img)
996
997         input = np.atleast_1d(input.astype(np.bool))
998         reference = np.atleast_1d(profile.astype(np.bool))
999
1000        intersection = np.zeros((zSize, 1))
1001        size_input = np.zeros((zSize, 1))
1002        size_reference = np.zeros((zSize, 1))
1003        for slice in range(zSize):
1004            intersection[slice] = np.count_nonzero(input[slice, :, :] & reference[slice, :, :])
1005            size_input[slice] = np.count_nonzero(input[slice, :, :])
1006            size_reference[slice] = np.count_nonzero(reference[slice, :, :])
1007
1008            try:
```

```
1009                 if (DC[slice] == 0) and (float(size_input[slice] + size_reference[slice]) != 0):
1010                     DC[slice] = (2. * intersection[slice] / float(size_input[slice]
1011                                  + size_reference[slice]))
1012 
1013             except ZeroDivisionError:
1014                 DC[slice] = -1
1015 
1016         if show != False:
1017             profile_img = sitk.GetImageFromArray(profile)
1018             sitk_centroid_show(profile_img, centroid*xSpace, extent=extent,
1019                                title="profile, radius: {:03.2f}".format(radius*xSpace),
1020                                ref=show, save=save)
1021 
1022         return DC
1023 
1024 
1025     # to view in 3D Slicer, type this in IPython console or in jupyter notebook:
1026     # %env SITK_SHOW_COMMAND /home/david/Downloads/Slicer-4.5.0-1-linux-amd64/Slicer
1027     # sitk.Show(imgFillingCT)
```