SegTools module User Manual

Pauline Chassonnery[*], Laetitia Pieruccioni, Mathieu Vigneau
Restore Institute, Toulouse III University, France

October 16, 2023

Introduction

The objective of the SegTools Python module is to provide non-specialists in image analysis with an integrated, user-friendly, automatic segmentation algorithm to identify clusters of spherical objects in a 3D system composed of both spherical and rod-like elements. It takes as input either a pandas.DataFrame or a comma-separated values (csv) file describing the dataset to clusterize, and returns another pandas.DataFrame containing the resulting clusters. It is aimed in particular at modellers, who may generate such an *in silico* dataset through numerical simulations, and at biologists who may retrieve it from *in vivo* or *in vitro* experiments (for example, by detecting spherical cells in 3D microscopic images using the Imaris software).

Objects segmentation and clustering is a wide-spread problem in the domain of image analysis and many methods have been developed to solve it, of which one of the most widely used is the watershed segmentation. But these are generally not well-known to researchers outside this field and the procedures provided by image analysis software usually require a number of preprocessing steps. Moreover, data produced by a mathematical simulation will usually not even be in the form of images and will thus require an extra processing step that can be quite time-consuming if not optimized.

The main function of the SegTools module, described in section 2, is called **ClusterizeSphereObjects**. It takes as input a pandas.DataFrame describing a set of spherical objects as well as a number of optional parameters (see section 2.2 for more details), groups these objects into clusters based on spatial proximity using the watershed transformation, and returns a copy of the pandas.DataFrame with an additional column containing the index of the cluster each object pertains to.

Because conventional clustering software are made for biological or physical images, they never include tools to treat the case of periodic boundary conditions. However, this is a very common hypothesis in computational models that needs to be taken into account for clustering. The SegTools module thus includes an option for periodic boundary conditions.

The SegTools module is free of use, upon proper citation of [2] and of this user manual. The zipped package containing all the necessary files is downloadable here.

1 Installation

Describe usage of a python module : place .py files somewhere in your path, or place them in a new folder and add this folder to path. Then, in a python script, import the module to be able to call functions from it.

2 Function ClusterizeSphereObjects

The main function of the SegTools module, described in this section, is called **ClusterizeSphereObjects**. It takes as input a pandas.DataFrame describing a set of spherical objects as well as a number of optional parameters (see section 2.2 for more details), groups these objects into clusters based on spatial proximity using the watershed transformation, and returns a copy of the pandas.DataFrame with an additional column containing the index of the cluster each object pertains to.

2.1 Brief description of the algorithm

The clustering process can be broken down as follows (the name of the related sub-functions are given in bold text) :

1. Retrieve the objects' data.
   - ↳ **RetrieveSpheresData**
   - ↳ **RetrieveRodsData**
   - ↳ **ParametersForClustering**
2. Create a 3D binary image where objects are white and background is black.
   - ↳ **Create3Dbinaryimage**
3. Apply euclidean distance transform to create a grayscale map indicating the distance of each white pixel of the binary to the background.
   - ↳ **scipy.ndimage.distance_transform_edt**
4. Filter the distance map to remove shallow local minima.

[*] E-mail: pauline.chassonnery@ens-cachan.fr

Name	Description	Python type	Default value
`InputSpheres`	Set of spherical objects to clusterize.	*string* or *pandas.DataFrame*	NA : it is a mandatory parameter
`InputRods`	Set of rod-like objects acting as cluster separators.	*string* or *pandas.DataFrame*	None
`Periodic Boundary Condition`	Whether the border of the computational domain are considered periodic.	*bool*	False
`xmax`, `ymax`, `zmax`	Size of the computational domain.	*float*	None
`dil_coeff`	Scaling to apply to the spherical objects before clustering.	*float*	1
`resolution`	Image resolution, in pixel per radius of the smallest object.	*int*	10
`smooth_coeff`	Smoothing of the cluster edges.	*float*	3
`MinNumber OfObject InCluster`	Minimal number of objects that a cluster must contain to be retained as valid.	*int*	5
`header`	Header for the column containing the clusterization result in the output *pandas.DataFrame*.	*string*	"Cluster"

Table 1: List of all the parameters

↳ **imhmin_redefined**

5. Identify and label local minima in the filtered distance map.

   ↳ **skimage.morphology.local_maxima**

   ↳ **scipy.ndimage.label**

6. Apply watershed algorithm to the filtered distance map, using local minima as seeds.

   ↳ **skimage.segmentation.watershed**

7. Identify the cluster index of each object based on the clustermap generated at the previous step.

   ↳ **ClusterizeFromMap**

Steps 3 through 6 are combined in a function **MapRegionsUsingWatershed**.

2.2 Mandatory parameters

2.2.1 Set of spherical objects

The only mandatory argument of the function **ClusterizeSphereObjects** is a set of spherical objects `InputSpheres`. This set must be provided in one of the two following formats.

(i) A *pandas.DataFrame* with at least two rows (i.e. it must describe at least two objects otherwise there is nothing to clusterize) and at least four columns with headers "X", "Y", "Z" and "R" containing respectively the x-, y- and z-coordinate of the object's center and its radius. The dataset may contain any number of additional columns but these will not be processed.

(ii) A *string* with the path to a csv file containing the data, which will be read to a *pandas.DataFrame* using the function **pandas.read_csv**. The file must use the default csv format, that is comma ',' for column separator and dot '.' for decimal point. The first line must contain column headers among which "X", "Y", "Z" and "R" pointing respectively the x-, y- and z-coordinate of the object's center and its radius. The order of the columns does not matter and the file may contain any number of additional columns. The folder demo_data/ contains an example of such a file, named "spheres.csv".

2.3 Optional parameters

2.3.1 Set of rod-like objects

The function **ClusterizeSphereObjects** may be provided with a set of rod-like or spherocylindrical objects `InputRods` which will be considered as "cluster separators". These objects will be used as masks during the creation the binary image : that is, pixels located inside a rod-like object will be considered as background even if they are also inside a spherical object. This may help to segment clusters in the set of spherical objects.

Similarly to the parameter `InputSpheres`, the parameter `InputRods` must be provided in one of the two following formats.

(i) A *pandas.DataFrame* with at least eight columns with headers "X", "Y", "Z", "wX", "wY", "wZ", "L" and "R" containing respectively the x-, y- and z-coordinate of the object's center, the x-, y- and z-component of its unitary directional vector, its length and its radius. The dataset may contain any number of additional columns but these will not be processed.

(ii) A *string* with the path to a csv file containing the data, which will be read to a *pandas.DataFrame* using the function **pandas.read_csv**. The file must use the default csv format, that is comma ',' for column separator and dot '.' for decimal point. The first line must contain column headers among which "X", "Y", "Z", "wX", "wY", "wZ", "L" and "R" pointing respectively the x-, y- and z-coordinate of the object's center, the x-, y- and

z-component of its unitary directional vector, its length and its radius. The order of the columns does not matter and the file may contain any number of additional columns. The folder demo_data/ contains an example of such a file, named "rods.csv"

By default, the parameter `InputRods` is equal to "None".

2.3.2 Periodic boundary condition

The parameter `PeriodicBoundaryCondition` allows the user to specify whether the domain boundary conditions are periodic or not.

If the input data is derived from a mathematical model including periodic boundary condition (any object exiting the domain from one side re-enter from the opposite side), then the segmentation process should also include periodic boundary condition. This is done by setting the parameter `PeriodicBoundaryCondition` to True and providing the function with the exact size of the computational domain (see next section and Figure 2).

Python type is *bool*, default value is False.

2.3.3 Size of the computation domain

The algorithm assumes the data to be contained in a 3D symmetrical domain $\Omega = [-\mathtt{xmax}, \mathtt{xmax}] \times [-\mathtt{ymax}, \mathtt{ymax}] \times [-\mathtt{zmax}, \mathtt{zmax}]$ and creates a 3D binary image spanning this whole domain.

If the user did not provide a value for any of the half-length parameters `xmax`, `ymax` and `zmax`, the algorithm will estimate the missing value(s) as the smallest value(s) allowing the domain to enclose all the spherical objects referenced in `InputSpheres` :

```
xmax = np.max(np.abs(InputSpheres["X"]) +
                     InputSpheres["R"])
ymax = np.max(np.abs(InputSpheres["Y"]) +
                     InputSpheres["R"])
zmax = np.max(np.abs(InputSpheres["Z"]) +
                     InputSpheres["R"])
```

Figure 1 gives a visual example of how these length are computed.

User-provided values are only needed if `PeriodicBoundaryCondition` is True, to find out which part of the objects cross the border of the domain and must be accounted for on the opposite side. Note that, with the default values, no objects will cross the border and periodic boundary condition will thus have no effect. For this reason, if `PeriodicBoundaryCondition` is True but no value is provided for `xmax` and/or `ymax` and/or `zmax`, computation will run with default value(s) but a warning will be issued to the user.

If `PeriodicBoundaryCondition` is False, the user may provide its own values or let the computation run with the default values. This should not make any difference to the segmentation result.

Python types are *float*. Default values are None, leading to auto-computation.

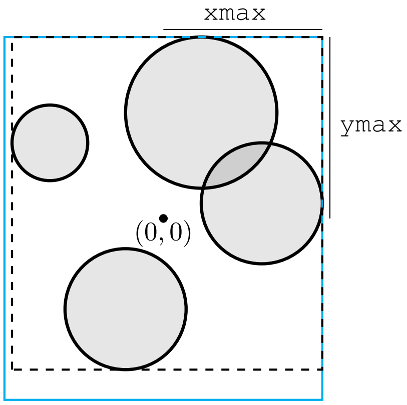


Figure 1: Example of computation of the domain size from the data (in 2D). The bounding box of the data is drawn in black dashed lines and the border of the symmetrical domain Ω in blue solid lines.

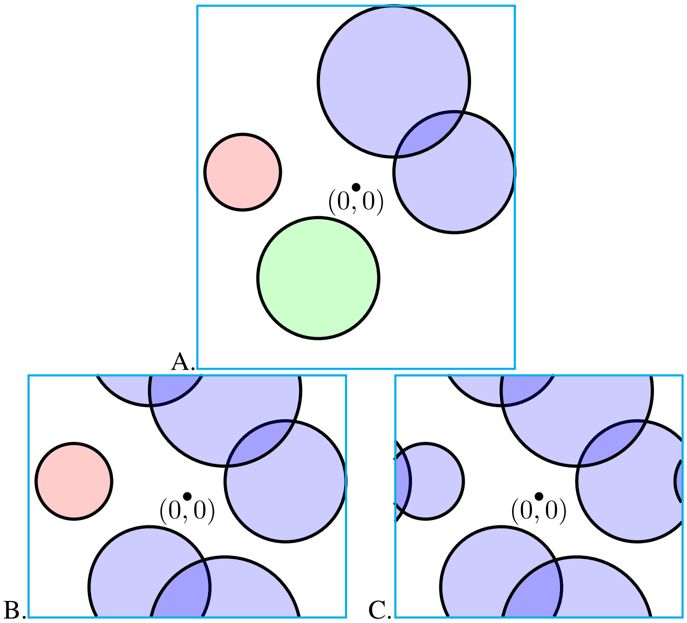


Figure 2: Example of the application of periodic boundary condition to the same data with different domain size. **A** : auto-computation of the domain (`xmax` = `ymax` = None), the periodic boundary conditions have no impact. The algorithm will identify three clusters, indicated in color. **B** : user-defined value for `ymax` and auto-computation of `xmax`, two of the previous clusters are now merged through the periodic y-border. **C** : user-defined value for `xmax` and `ymax`, all clusters are now merged through the periodic borders.

2.3.4 Objects scaling

To make clustering easier, it may be useful to scale the objects up/down by a factor `dil_coeff`, that is to multiply the radius of the objects by `dil_coeff` when inserting them in the 3D binary image (see Figure 3 and 4 for examples).

For instance, if there are small gaps between objects pertaining to the same cluster, the distance transform will display a local minima at the center of each object and watershed clustering will thus segment the individual objects instead of the cluster (see Figure 3.A). In this case, scaling up the objects by a small factor may fill the holes without unduly connecting separate clusters (see Figure 3.B). This option is especially intended for cases where the user also provides a set of rod-like objects to serve as separators between clusters, the risk of cluster merging being then reduced.

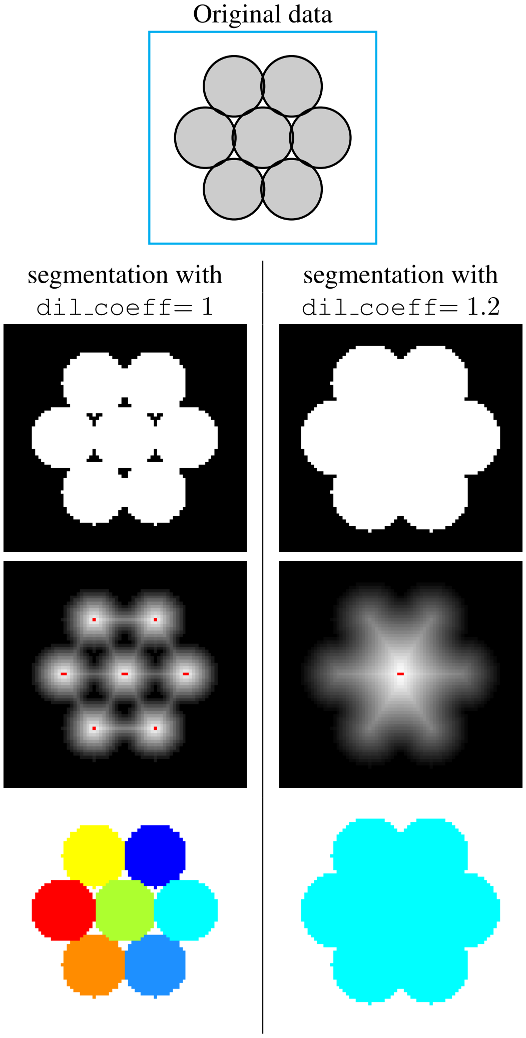


Figure 3: Example (in 2D) of data scaling up. The original data is displayed at the top. Each column shows first the binary image produced from the data, then its euclidean distance transform (with local minima indicated in red) and last the segmentation result in the form of a region map. **Left column :** The original data shows a cluster of objects with small gaps between them. The euclidean distance transform thus presents a local minima at the center of each object, leading to the segmentation of 7 clusters constituted of only one object. **Right column :** Data scaled by a factor `dil_coeff`= 1.2, closing the gaps between the objects. The euclidean distance transform presents now only one local minimum, leading to the segmentation of one cluster containing all the objects.

If, on the other hand, the user wants to divide a continuous cluster into multiple components based on a notion of objects density, scaling down may allow it by creating holes only in the low density regions (see Figure 4). This option is intended for cases where there are large discrepancies in the objects local density.

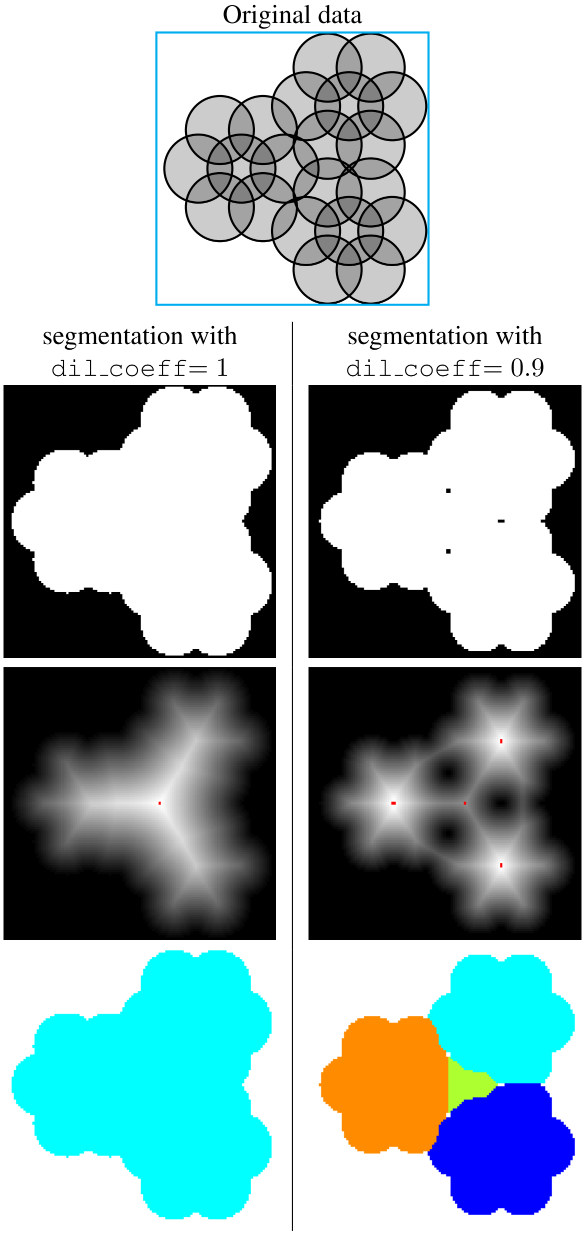


Figure 4: Example (in 2D) of data scaling down. The original data is displayed at the top. Each column shows first the binary image produced from the data, then its euclidean distance transform (with local minima indicated in red) and last the segmentation result in the form of a region map. **Left column :** The original data shows a unique cluster with three denser areas. The euclidean distance transform presents a single local minimum at the center of the whole structure, leading to the segmentation of one cluster containing all the objects. **Right column :** Data scaled by a factor `dil_coeff`= 0.9, creating gaps between the three denser areas. The euclidean distance transform presents now 4 local minima, leading to the segmentation of the 4 regions : the three dense areas and a miscellaneous zone at their junction. This last region covers part of three objects but, because objects are assigned a cluster based on the majority value of their pixels, it does not really "own" any object and will therefore not constitute a cluster.

In both case, it is best to keep the factor `dil_coeff` close to

1 as values too high or too low are likely to produce unwanted results.

Python type is *float*, default value is 1 (i.e. no scaling).

2.3.5 Image resolution

The parameter resolution represents the number of pixels of the 3D binary image that will be covered by the diameter of the smallest spherical object referenced in InputSpheres. In other words, the binary image will be composed of cubic pixels of side-length equal to the diameter of this smallest spherical object divided by resolution.

The minimal value is 1, which ensures that all objects are represented at least by one pixel in the binary image. Setting a high value of resolution ensure that small objects are well outlined, but increase the total size of the 3D binary image and thus the computational time. Please note that, from the perspective of cluster segmentation, a high resolution is usually necessary only if the clusters are very close to each other.

Python type is *int*, default value is 10.

2.3.6 Region smoothing (or peak filtering)

The watershed algorithm segment clusters in a binary image based on seeds. These seeds are usually defined as the local minima in the euclidean distance transform of the image. However, taking all local minima into account is likely to result in over-segmentation, since clusters with irregular edges will generate irregular depth-patterns with multiple local minima (see first row of Figure 5). Hence, it is common to filter such shallow minima by applying to the distance transform a smoothing function.

For this we used a modified version of the h-minima smoothing function, designed to reproduce the behaviour of Matlab's imhmin function. This function erase all local minima whose depth relative to their surroundings is less than the distance smooth_coeff (see second and third rows of Figure 5).

Python type is *float*, default value is 3.

2.3.7 Minimal number of objects in a cluster

Clusters containing too few objects may not interest users. Hence, the parameter MinNumberOfObjectInCluster allows the user to define the minimum number of objects a cluster must contain to be considered valid. Appropriate values may vary widely depending on the user purpose.

Please note that this option does not affect the segmentation process in itself : it is only a post-processing step which erase small clusters. It is mainly intended for users who plan to do statistical computation on the segmented clusters and do not want to take into account small, miscellaneous clusters. It *does not* merge multiple small clusters into a large cluster or blend a small cluster into a neighbouring large cluster (such results could however be achieved using the parameters dil_coeff and smooth_coeff, see corresponding sections).

Objects pertaining to invalid clusters will get the cluster index -1, while valid clusters are numbered starting from 0 to be consistent with Python indexing.

Python type is *int*, default value is 5.

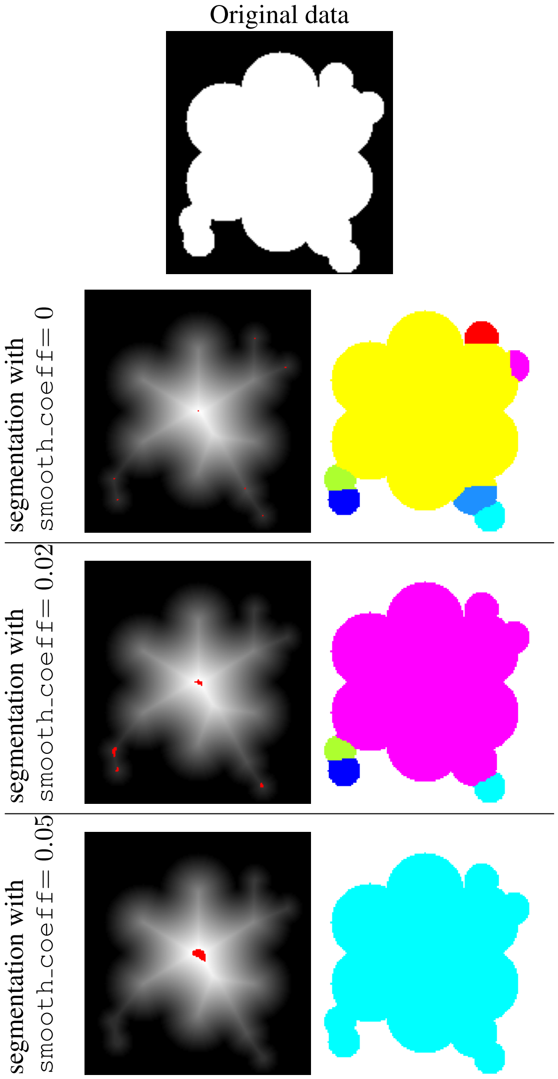

Original data
segmentation with smooth_coeff= 0
segmentation with smooth_coeff= 0.02
segmentation with smooth_coeff= 0.05

Figure 5: Example (in 2D) of data smoothing with three different values of smooth_coeff. The original data is displayed at the top. Each row shows first the smoothed euclidean distance transform of the binary image, with local minima indicated in red, then the segmentation result in the form of a region map. In each cases, whether the data are over-segmented or not depends on the user's objective. **First row :** The non-smoothed euclidiean distance transform presents local minima at the center of each object protruding from the cluster's core, leading to segmentation of one cluster and six isolated objects. **Second row :** Smoothing the euclidean distance transform by a factor smooth_coeff= 0.02 reduces the number of local minima to 4. Note how the remaining local minima span more pixels due to the smoothing operation. **Third row :** Smoothing the euclidean distance transform by a factor smooth_coeff= 0.05 reduces the number of local minima to 1, leading to the segmentation of one cluster containing all the objects.

2.3.8 Result header

The parameter `header` allows the user to define the label of the column containing the result of the clusterization process in the output *pandas.DataFrame*.

Python type is *string*, default value is “Cluster”.

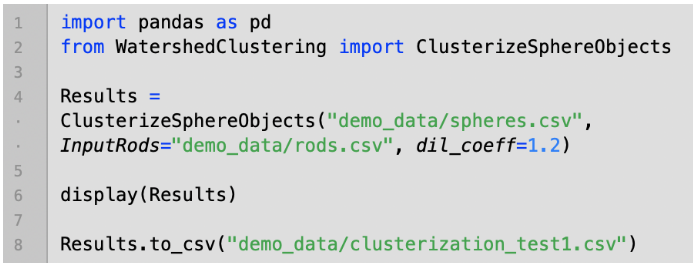
2.4 Output data

The function **ClusterizeSphereObjects** returns a *pandas.DataFrame* containing the original set of spherical objects plus the result of the clusterization process.

If the original set `InputSpheres` was a *pandas.DataFrame*, then **ClusterizeSphereObjects**’s output is a deep copy of it. If `InputSpheres` was a *string* with the path to a csv file, then **ClusterizeSphereObjects**’s output contains all the data read from this file using the function **pandas.read_csv**. In both cases, the cluster index of each object will be indicated in an additional column labeled `header`.

2.5 Practical examples

N.B. : Examples would be more understandable if coupled with paraview visualization, but this is outside the scope of the present manual. To be expanded later.



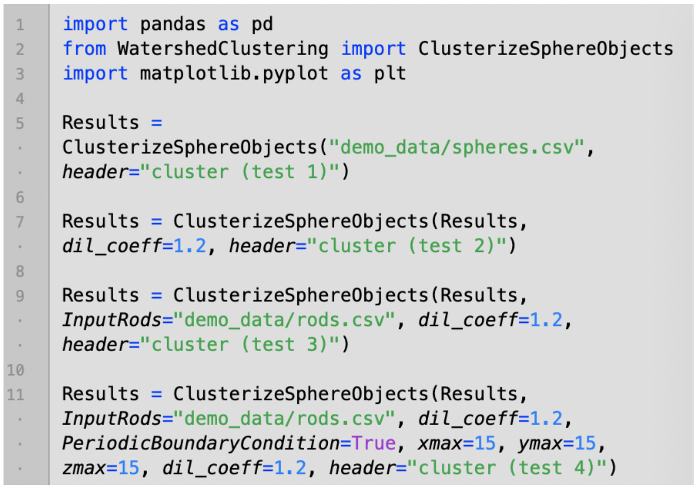
```python
import pandas as pd
from WatershedClustering import ClusterizeSphereObjects

Results = ClusterizeSphereObjects("demo_data/spheres.csv", InputRods="demo_data/rods.csv", dil_coeff=1.2)

display(Results)

Results.to_csv("demo_data/clusterization_test1.csv")
```

Figure 6: Example of use of the function **ClusterizeSphereObjects** to clusterize a set of spherical objects whose data is contained in the file ”spheres.csv” in the folder ”demo_data/”. The resulting *pandas.DataFrame* is displayed on screen using function **display**, then saved in a new file spheres_clusterized.csv.



```python
import pandas as pd
from WatershedClustering import ClusterizeSphereObjects
import matplotlib.pyplot as plt

Results = ClusterizeSphereObjects("demo_data/spheres.csv", header="cluster (test 1)")

Results = ClusterizeSphereObjects(Results, dil_coeff=1.2, header="cluster (test 2)")

Results = ClusterizeSphereObjects(Results, InputRods="demo_data/rods.csv", dil_coeff=1.2, header="cluster (test 3)")

Results = ClusterizeSphereObjects(Results, InputRods="demo_data/rods.csv", dil_coeff=1.2, PeriodicBoundaryCondition=True, xmax=15, ymax=15, zmax=15, dil_coeff=1.2, header="cluster (test 4)")
```

Figure 7: Example of use of the function **ClusterizeSphereObjects** to clusterize the same set of spherical objects with various clusterization parameters.

Acknowledgements

References

[1] F. Meyer. Topographic distance and watershed lines. Signal Processing, 38(1):113–125, July 1994

[2]