1 Python correction

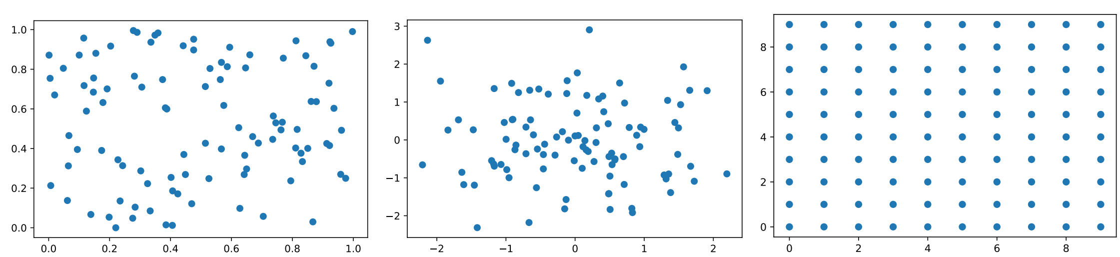
The different libraries will be used. shapely deals with polygons (area and perimeter), scipy.sparse.csgraph with the minimum spanning tree.

```python
from scipy.spatial import Voronoi, voronoi_plot_2d, Delaunay, distance
import numpy as np

import matplotlib.pyplot as plt
from shapely import geometry
from  scipy.sparse import csgraph
```

1.1 Random tessellations

Random tessellations are generated following normal standard and uniform distribution. A regular pattern is also employed. They are illustrated in Fig.1.

```python
def dist_poisson(N=100):
    points = np.random.rand(N, 2)
    return points

def dist_gaussienne(N=100):
    points = np.random.randn(N, 2)
    return points

def dist_regular(N=100):
    c = np.floor(np.sqrt(N));
    x2, y2 = np.meshgrid(range(int(c)), range(int(c)));
    points = np.vstack([x2.ravel(), y2.ravel()])
    return points.transpose();
```



(a) Uniform distribution. (b) Gaussian distribution. (c) Regular distribution.

Figure 1: Different point patterns.

1.2 Voronoi diagram and analysis

The Voronoi diagram is simply generated via the following command, with points being generated with the previous functions:

```
vor = Voronoi(points);
```

The two characterization functions RFH and AD are defined on the Voronoi cells.

```
def RFH(vor):
    """
    Evaluates Round Factor Homogeneity from voronoi diagram
    """
    rfs=[];
    for cell in vor.regions:
        if cell and -1 not in cell:
            poly = geometry.Polygon([(vor.vertices[p-1]) for p in cell]);
            rfs.append(4*np.pi*poly.area/(poly.length**2));
    res = 1 - np.std(rfs) / np.mean(rfs);
    return res;
```

```
def AD(vor):
    """
    Evaluates Area Disorder from voronoi diagram
    """
    areas=[];
    for cell in vor.regions:
        if cell and -1 not in cell:
            poly = geometry.Polygon([(vor.vertices[p-1]) for p in cell]);
            areas.append(poly.area);
    res = 1 - 1/(1+np.std(areas) / np.mean(areas));
    return res;
```

1.3 Delaunay triangulation and minimum spanning tree

The Delaunay triangulation is computed with

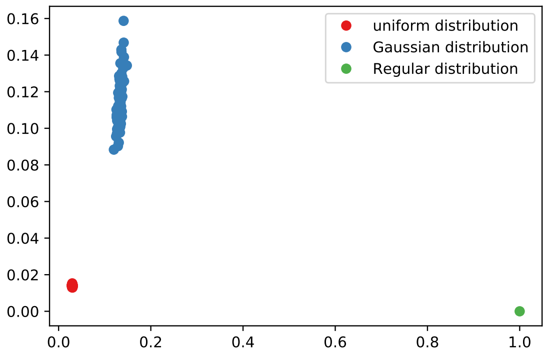
```
tri = delaunay(points);
```

Then, the characterization of the triangulation is done by measuring the distances of the edges.

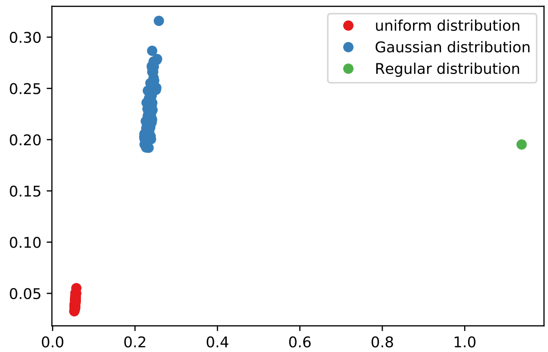
```python
def characterization(tri):
    """
    Characterization of the Delaunay triangulation (mean and std dev of
        ↪ edges)
    """
    M = triToMat(tri);
    m = np.mean(M[M>0]);
    s = np.std(M[M>0]);
    return m, s;

def triToMat(tri, value=0.):
    """
    Transforms the triangulation into a matrix representation,
    for simplicity
    """
    M = np.full((tri.npoints, tri.npoints), value);
    d = distance.pdist(tri.points);
    distances = distance.squareform(d);
    for s in tri.simplices:
        M[s[0], s[1]] = distances[s[0], s[1]];
        M[s[1], s[2]] = distances[s[1], s[2]];
        M[s[2], s[0]] = distances[s[2], s[0]];
    return M;
```

1.4 Characterization of different realizations

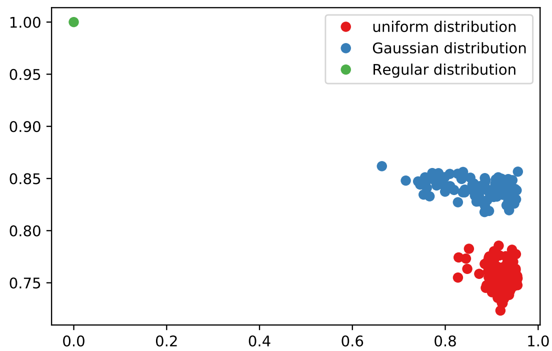
n realizations of the two distributions (uniform and Gaussian) are simulated. Then, the Voronoi diagram and the Delaunay triangulation are computed and characterized. The results are presented in Fig.2.



(a) Characterization of the Delaunay trianglulation



(b) Characterization of the minimum spanning tree of the Delaunay triangulation.



(c) AD and RFH on the Voronoi diagram.

Figure 2: Characterization of several point processes. Each color represent a different process, and each point represent one realization.