In [1]:

```
import sys
from sklearn.neighbors import KernelDensity

sys.path.insert(0, "gudhi-devel-multi/build/src/python/gudhi/")
import gudhi as gd
import numpy as np
import matplotlib.pyplot as plt
```

From scratch

Definition of a simplextree

In [2]:

```
simplextree = gd.SimplexTreeMulti(num_parameters=2)
simplextree.insert([0,1,2], [0,0])        # we add the triangle 0-1-2 to the complex.

#    0---2
#    |**/
#    |*/
#    |/
#    1
```

Out[2]:

```
True
```

In [3]:

```
simplextree.num_parameters
```

Out[3]:

```
2
```

Filtration assignment

In [4]:

```
#simplextree.assign_filtration(simplex, bifiltration value)
simplextree.assign_filtration([0], np.array([1,2], dtype=int)) # The vertices appear
simplextree.assign_filtration([1], np.array([1,2], dtype=np.float128))
simplextree.assign_filtration([2], [1,2])
simplextree.assign_filtration([0,1], [1,2]) # The edges (and the cycle) appear at [1,
simplextree.assign_filtration([0,2], [1,2])
simplextree.assign_filtration([1,2], [1,2])
simplextree.assign_filtration([0,1,2], [3,3]) # The triangle appears at [3,3], and ki
```

In [5]:

```
simplextree2 = gd.SimplexTreeMulti(num_parameters = 2)
simplextree2.insert([0,1,2], [0,0])
for s,f in simplextree2.get_skeleton(1):
        simplextree2.assign_filtration(s, [1,2])
simplextree2.assign_filtration([0,1,2], [3,3])
print(simplextree == simplextree2)
simplextree2.assign_filtration([0], [1,1])
print(simplextree == simplextree2)
```

```
True
False
```

Multi-critical filtrations

In [6]:

```
simplextree.filtration([0])
```

Out[6]:

```
[1.0, 2.0]
```

In [7]:

```
simplextree.insert([0], [1,3]) # (1,3) >= (1,2), simplex already appeared
```

Out[7]:

```
False
```

In [8]:

```
# (2, 1) is incomparable to previous filtrations, it can be added as a new birth
simplextree.insert([0], [2,1])
```

Out[8]:

```
True
```

In [9]:

```
# Vector on which all filtration values of this simplex are stacked.
f = simplextree.filtration([0])
np.array_split(f, len(f) // simplextree.num_parameters) # They can be unstack using a
```

Out[9]:

```
[array([1., 2.]), array([2., 1.])]
```

In [10]:

```
## Prevents assigning filtration values
#  that don't have the same shape as self.num_parameter
try:
    simplextree.assign_filtration([0],[1,1,1])
except AssertionError:
    print("len([1,1,1]) % 2 != 0")
    # I don't know how to make the assert message print with cython
```

```
len([1,1,1]) % 2 != 0
```

Example of (useful) bifiltration : Rips + Density

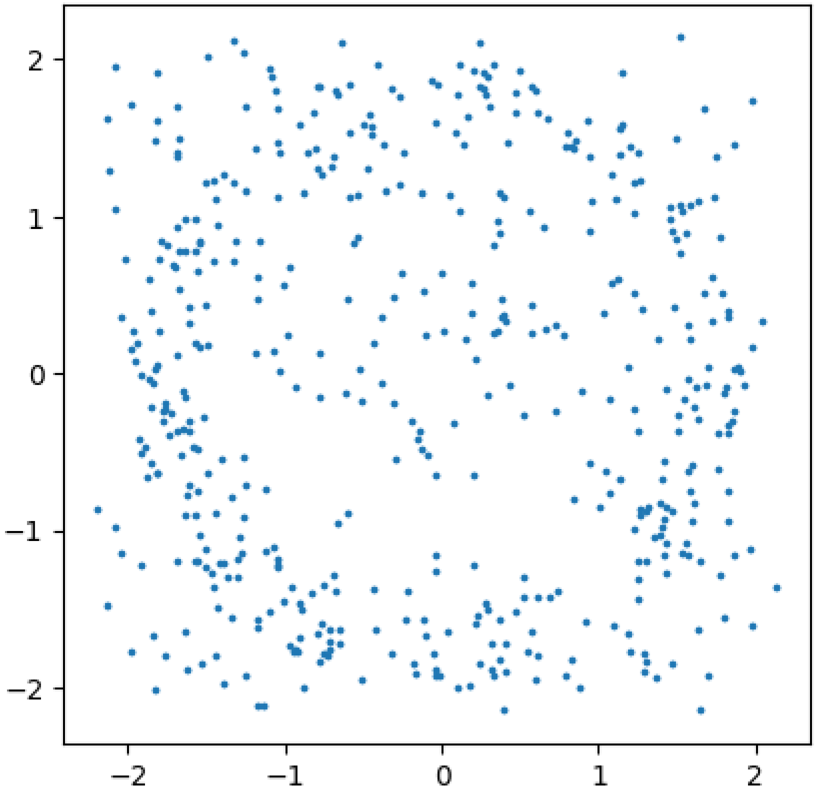
In [11]:

```
def noisy_annulus(r1:float=1, r2:float=2, n1:int=1000,n2:int=200, dim:int=2, center:n
        """Generates a noisy annulus dataset.

        Parameters
        ----------
        r1 : float.
                Lower radius of the annulus.
        r2 : float.
                Upper radius of the annulus.
        n1 : int
                Number of points in the annulus.
        n2 : int
                Number of points in the square.
        dim : int
                Dimension of the annulus.
        center: list or array
                center of the annulus.

        Returns
        -------
        numpy array
                Dataset. size : (n1+n2) x dim

        """
        from numpy.random import uniform
        from numpy.linalg import norm

        set =[]
        while len(set)<n1:
                draw=uniform(low=-r2, high=r2, size=dim)
                if norm(draw) > r1 and norm(draw) < r2:
                        set.append(draw)
        annulus = np.array(set) if center == None else np.array(set) + np.array(cente
        diffuse_noise = uniform(size=(n2,dim), low=-1.1*r2,high=1.1*r2)
        if center is not None:  diffuse_noise += np.array(center)
        return np.vstack([annulus, diffuse_noise])
```

In [12]:

```
npts = 200
noutliers = (int)(npts * 0.4)
np.random.seed(100)
```

```
points = np.block([[np.array(noisy_annulus(1.5,2, npts))], [np.random.uniform(low=-2,
fig, ax = plt.subplots()
ax.set_aspect('equal')
plt.scatter(points[:,0], points[:,1],s=3)
plt.show()
```



In [13]:
```
simplextree = gd.RipsComplex(points = points).create_simplex_tree(max_dimension=1)
## Creates a SimplexTreeMulti from a standard one. The first filtration is the same
simplextree = gd.SimplexTreeMulti(simplextree, num_parameters=2)
simplextree.num_simplices()
```

Out[13]:
```
115440
```

In [14]:
```
kde = KernelDensity(bandwidth=0.25)
density = kde.fit(points).score_samples(points)
simplextree.fill_lowerstar(-density, parameter=1)
```

Out[14]:
```
<gudhi.simplex_tree_multi.SimplexTreeMulti at 0x7f11ba922810>
```

In [15]:
```
# Example of filtration
print(next(simplextree.get_simplices()))
```

```
([0, 1], [0.351698269996502, 2.6640137794676924])
```

In [16]:
```
# Edge collapses from filtration_domination
simplextree.collapse_edges(strong=True, num=100, progress=1) # Takes time to get an e
simplextree.collapse_edges(strong=False, num=100, progress=1)
simplextree.expansion(2)
simplextree.num_simplices()
```

```
Removing edges:  13%|████                              | 13/100 [00:02<00:17,  4.84it/s]
Removing edges:  11%|███                               | 11/100 [00:01<00:14,  6.09it/s]
```

Out[16]:
```
7791
```

In [17]:
```
## The eulerchar of a list of points of a simplextree can be computed using this meth
# NOTE: Makes no sense on rips complexes, as they are cut in dimension.
# I have another implementation, relying on computing the rank invariant on a grid.
simplextree.euler_char([[0.1,2.5], [1,2.2]])
```

```
Out[17]: array([54,  6])
```