Actividad 1 Rips, Cech y Alpha.

In [ ]:
```
import numpy as np # Se necesitan arrays
import pandas as pd # Para trabajar con bases de datos
import matplotlib.pyplot as plt # Para graficar
import matplotlib as cm #Para manejar colores
from scipy.spatial.distance import squareform, pdist #Para calcular matrices de distancias e hacer inferencia de los
import matplotlib.patches as mpatches #Para hacer elipses
from matplotlib.collections import PatchCollection #Para hacer elipses
import gudhi #Para hacer la filtracion de complejos simpliciales de Rips y Alpha
```

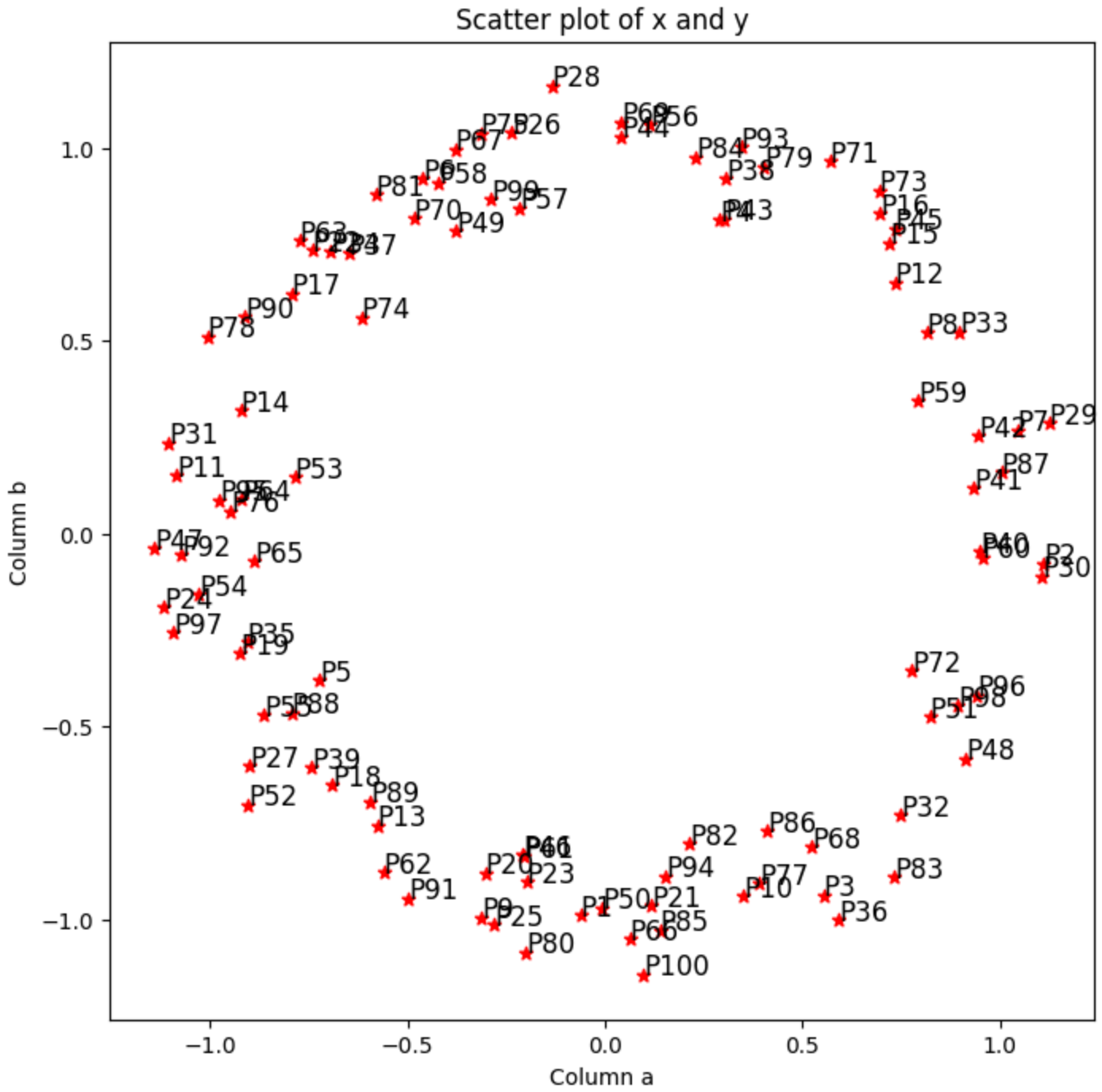
Dataset 1

In [ ]:
```
df1 = pd.read_csv('Datos/Activity1.csv') #Se lee la base de datos
df1.head()
num = np.arange(1,len(df1)+1) #Se crea un array con el numero de la fila
num = ['P'+str(i) for i in num] #Se convierte a string
df1['Punto'] = num #Se agrega al dataframe
df1.set_index('Punto',inplace=True) #Se cambia el indice
df1.rename(columns = {'0':'a','1':'b'},inplace=True) #Se cambian los nombres de las columnas
df1.head()
```

Out[ ]:

Punto	a	b
P1	-0.062332	-0.990463
P2	1.109356	-0.077222
P3	0.553080	-0.938321
P4	0.290183	0.813677
P5	-0.722770	-0.380330

In [ ]:
```
# Grafiuemos los datos
plt.figure(figsize=(8,8))
plt.scatter(df1['a'],df1['b'],c='r',marker='*')
plt.xlabel('Column a')
plt.ylabel('Column b')
plt.title('Scatter plot of x and y')
for j in df1.itertuples():
    plt.annotate(j.Index,(j.a,j.b),fontsize=12)
```


Scatter plot of x and y
Column a
Column b
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
P28
P67
P84
P93
P79
P71
P38
P73
P16
P15
P12
P17
P90
P74
P78
P8
P33
P59
P14
P31
P11
P53
P87
P41
P65
P54
P24
P97
P72
P96
P48
P27
P39
P18
P89
P52
P13
P32
P86
P68
P82
P62
P91
P94
P77
P83
P3
P36
P50
P21
P85
P80
P100

In [ ]:
```
#Calculemos la matriz de distancias
dist = pd.DataFrame(squareform(pdist(df1,metric='euclidean')),columns = num,index = num) #Calculamos la matriz de dis
m = dist.values.max()
m
```

Out[ ]:
```
2.316694694471048
```

Rips 1

Radio = 2.5

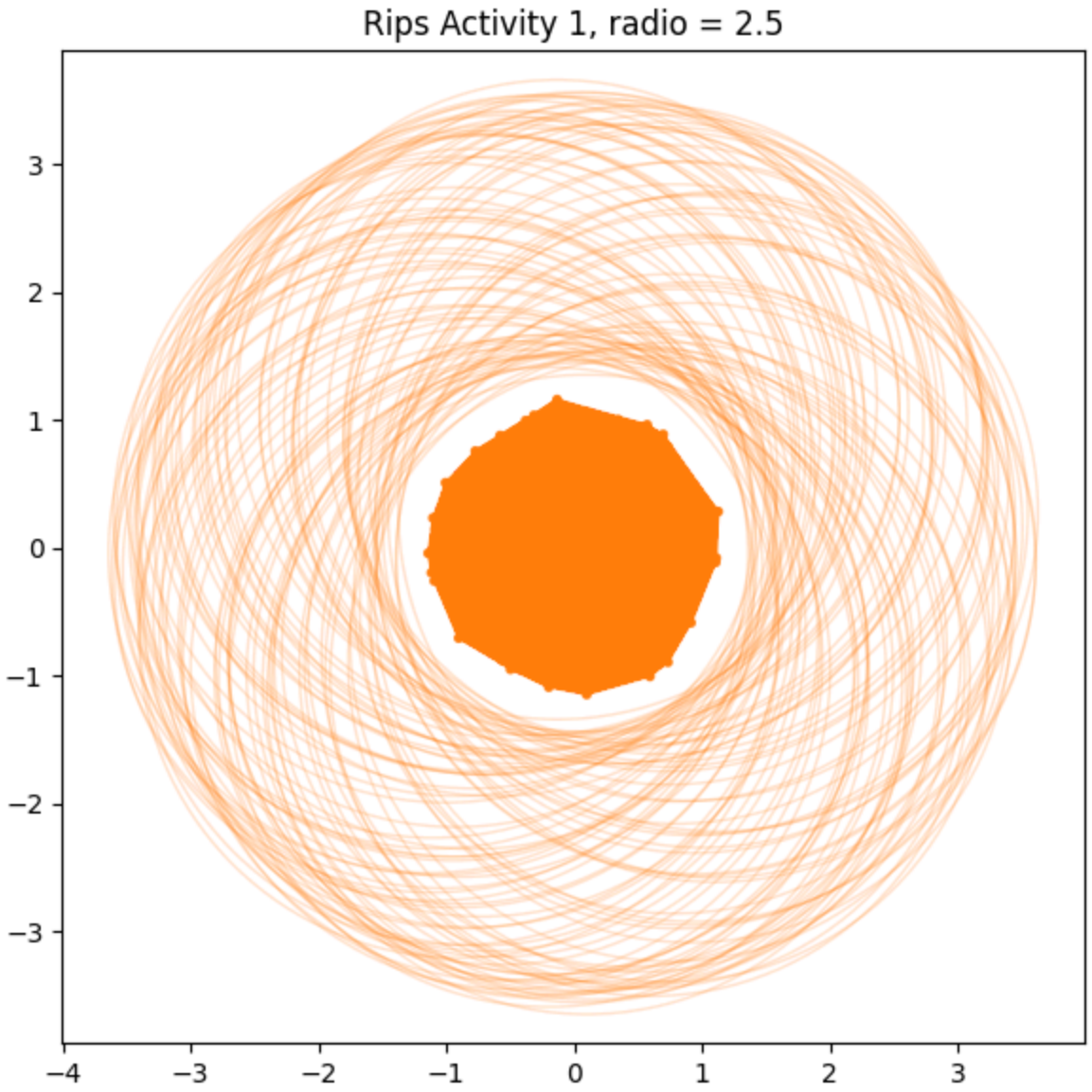
In [ ]:
```
# Calculamos la filtracion de Rips con un radio de 2.5
rips_complex = gudhi.RipsComplex(distance_matrix=dist.values, max_edge_length=2.5)
```

In [ ]:
```
simplex_tree = rips_complex.create_simplex_tree(max_dimension=2)
result_str = 'Rips complex is of dimension ' + repr(simplex_tree.dimension()) + ' - ' + \
               repr(simplex_tree.num_simplices()) + ' simplices - ' + \
               repr(simplex_tree.num_vertices()) + ' vertices.'
print(result_str)
fmt = '%s -> %.2f'
#for filtered_value in simplex_tree.get_filtration():
 #   print(fmt % tuple(filtered_value))
```

```
Rips complex is of dimension 2 - 65577 simplices - 100 vertices.
```

In [ ]:
```
def plot_rips_complex(df1, R, label="df1", col=1, maxdim=2):
    tab10 = plt.get_cmap('tab10')

    fig, ax = plt.subplots(figsize=(6, 6))
    ax.set_title(label)
    ax.scatter(
        df1[:, 0], df1[:, 1], label=label,
        s=8, alpha=0.9, c=np.array(tab10([col] * len(df1)))
    )

    for xy in df1:
        ax.add_patch(mpatches.Circle(xy, radius=R, fc='none', ec=tab10(col), alpha=0.2))

    for i, xy in enumerate(df1):
        if maxdim >=1:
```

```
            for j in range(i + 1, len(df1)):
                pq = df1[j]
                if (xy != pq).all() and (np.linalg.norm(xy - pq) <= R):
                    pts = np.array([xy, pq])
                    ax.plot(pts[:, 0], pts[:, 1], color=tab10(col), alpha=0.6, linewidth=1)
                if maxdim == 2:
                    for k in range(j + 1, len(df1)):
                        ab = df1[k]
                        if ((ab != pq).all()
                                and (np.linalg.norm(xy - pq) <= R)
                                and (np.linalg.norm(xy - ab) <= R)
                                and (np.linalg.norm(pq - ab) <= R)
                        ):
                            pts = np.array([xy, pq, ab])
                            ax.fill(pts[:, 0], pts[:, 1], facecolor=tab10(col), alpha=0.1)
                        pass

    plt.axis('equal')
    plt.tight_layout()
    plt.show()
    pass
plot_rips_complex(df1.values,R=2.5,label='Rips Activity 1, radio = 2.5',maxdim=2)
```



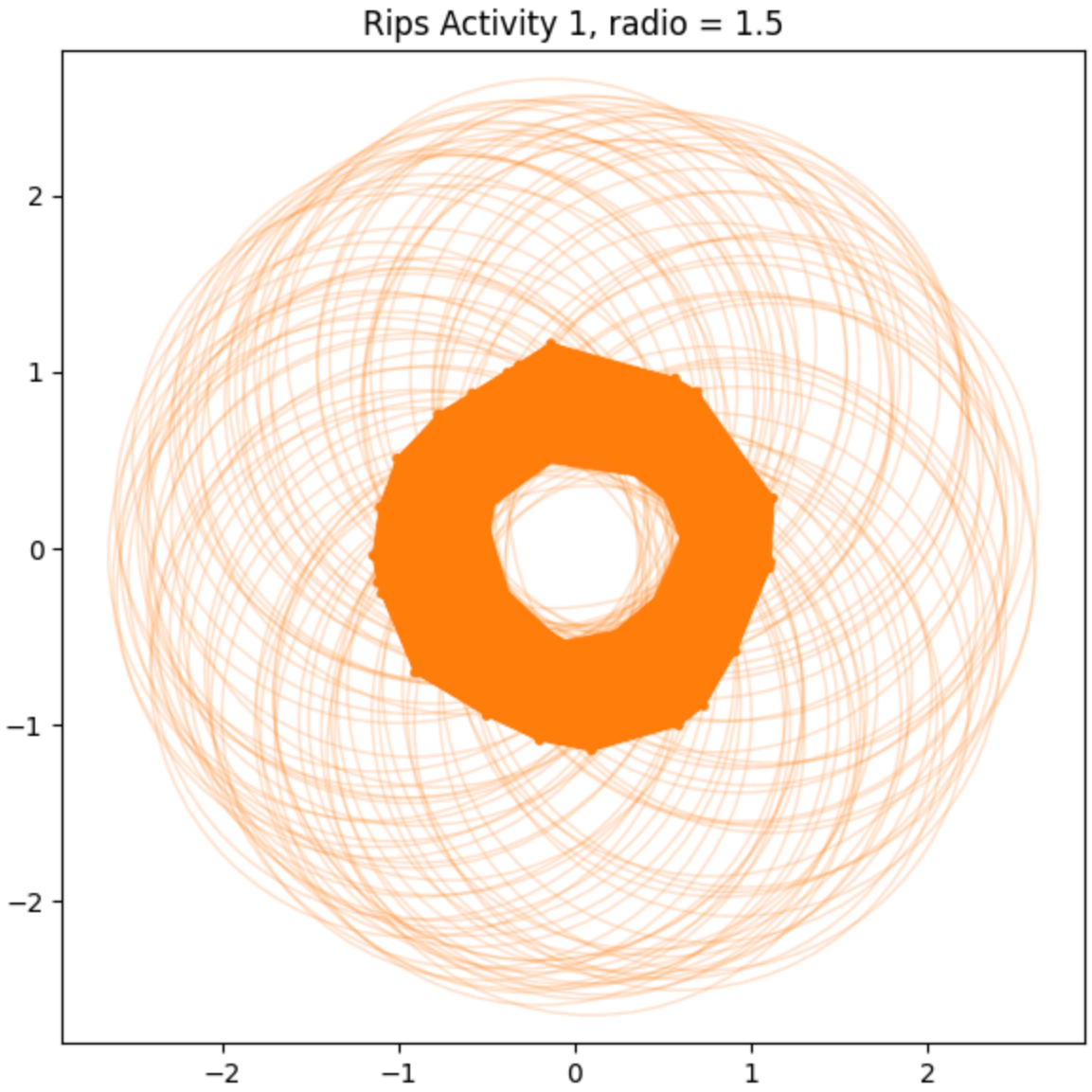
Radio = 1.5

In [ ]:
```
rips_complex = gudhi.RipsComplex(distance_matrix=dist.values, max_edge_length=1.5)
simplex_tree = rips_complex.create_simplex_tree(max_dimension=2)
result_str = 'Rips complex is of dimension ' + repr(simplex_tree.dimension()) + ' - ' + \
               repr(simplex_tree.num_simplices()) + ' simplices - ' + \
```

```
                repr(simplex_tree.num_vertices()) + ' vertices.'
print(result_str)
fmt = '%s -> %.2f'
#for filtered_value in simplex_tree.get_filtration():
 #   print(fmt % tuple(filtered_value))
```

```
Rips complex is of dimension 2 - 324 simplices - 100 vertices.
```

In [ ]:
```
plot_rips_complex(df1.values,R=1.5,label='Rips Activity 1, radio = 1.5',maxdim=2)
```



Alpha 1

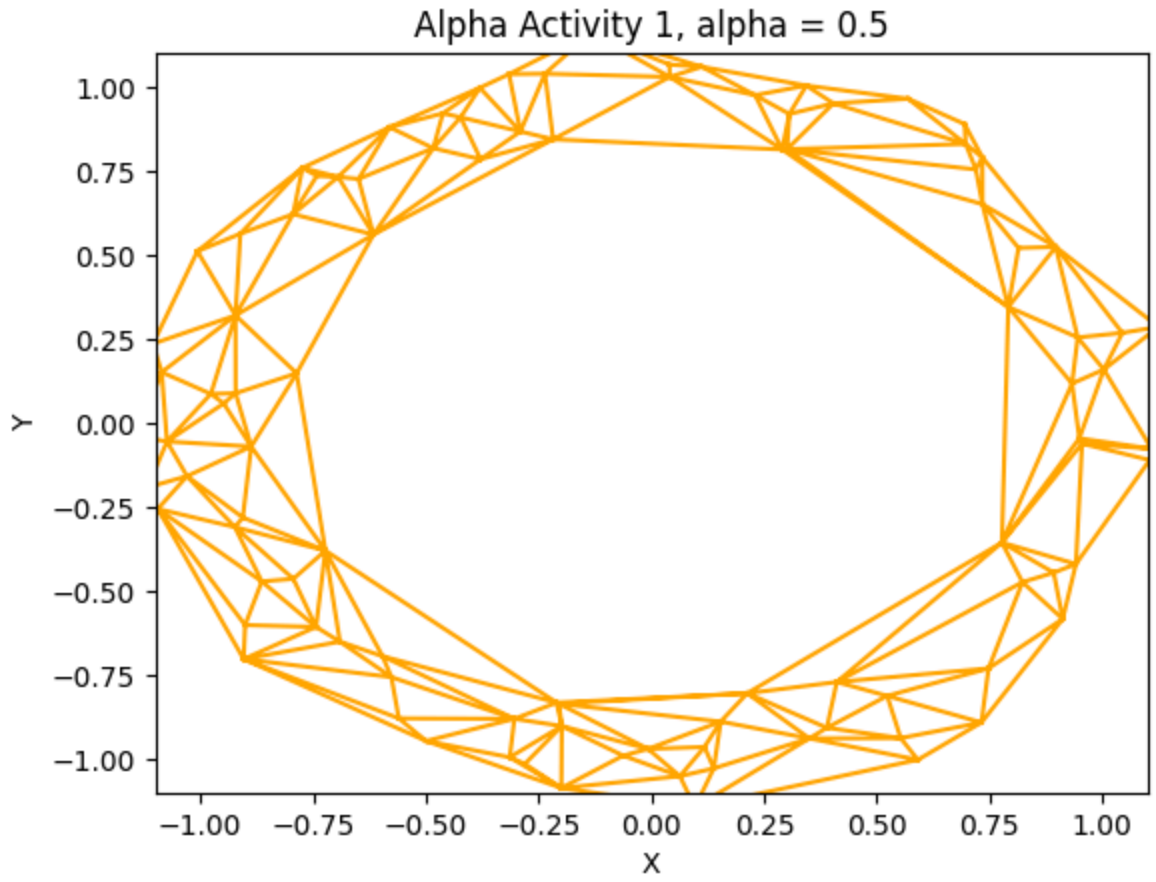
Alpha = 0.5

In [ ]:
```
alpha_complex = gudhi.AlphaComplex(points=df1.values)
simplex_tree = alpha_complex.create_simplex_tree(max_alpha_square=0.5)
result_str = 'Alpha complex is of dimension ' + repr(simplex_tree.dimension()) + ' - ' + \
               repr(simplex_tree.num_simplices()) + ' simplices - ' + \
               repr(simplex_tree.num_vertices()) + ' vertices.'
print(result_str)
fmt = '%s -> %.2f'
#for filtered_value in simplex_tree.get_filtration():
 #   print(fmt % tuple(filtered_value))
```

```
Alpha complex is of dimension 2 - 528 simplices - 100 vertices.
```

In [ ]:
```
def plot_alpha_complex(alpha_complex, label="df1", col=1, maxdim=2):
    points = np.array([alpha_complex.get_point(i) for i in range(simplex_tree.num_vertices())])
    triangles = np.array([s[0] for s in simplex_tree.get_skeleton(2) if len(s[0]) == 3 and s[1] <= 0.5])

    fig, ax = plt.subplots()
    ax.triplot(points[:, 0], points[:, 1], triangles=triangles, color='orange')
    ax.set_xlim(-1.1, 1.1)
    ax.set_ylim(-1.1, 1.1)
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.title(label)

    plt.show()
```

In [ ]:
```
plot_alpha_complex(alpha_complex, label='Alpha Activity 1, alpha = 0.5', maxdim=2)
```

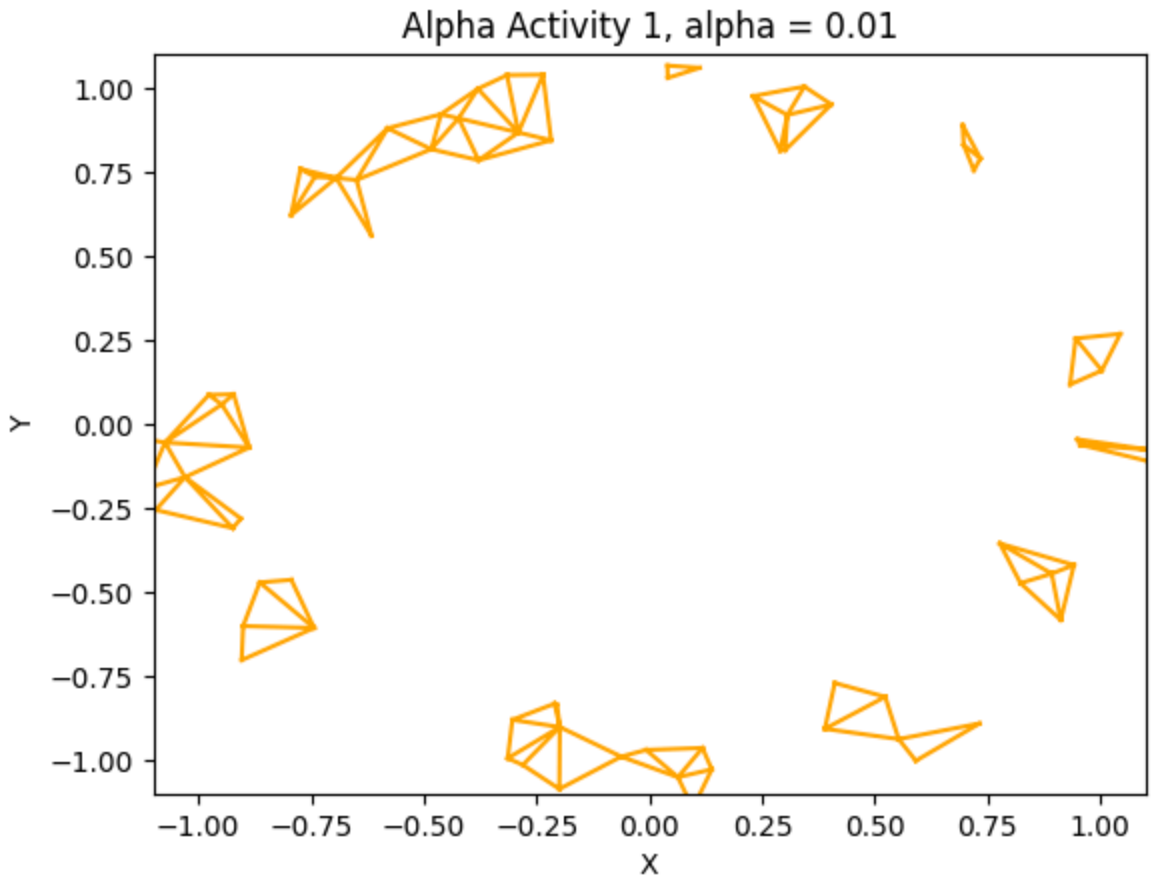


Alpha = 0.01

In [ ]:
```
alpha_complex = gudhi.AlphaComplex(points=df1.values)
simplex_tree = alpha_complex.create_simplex_tree(max_alpha_square=0.01)
result_str = 'Alpha complex is of dimension ' + repr(simplex_tree.dimension()) + ' - ' + \
               repr(simplex_tree.num_simplices()) + ' simplices - ' + \
               repr(simplex_tree.num_vertices()) + ' vertices.'
print(result_str)
fmt = '%s -> %.2f'
#for filtered_value in simplex_tree.get_filtration():
#    print(fmt % tuple(filtered_value))
```

```
Alpha complex is of dimension 2 - 314 simplices - 100 vertices.
```

In [ ]:
```
plot_alpha_complex(alpha_complex, label='Alpha Activity 1, alpha = 0.01', maxdim=2)
```



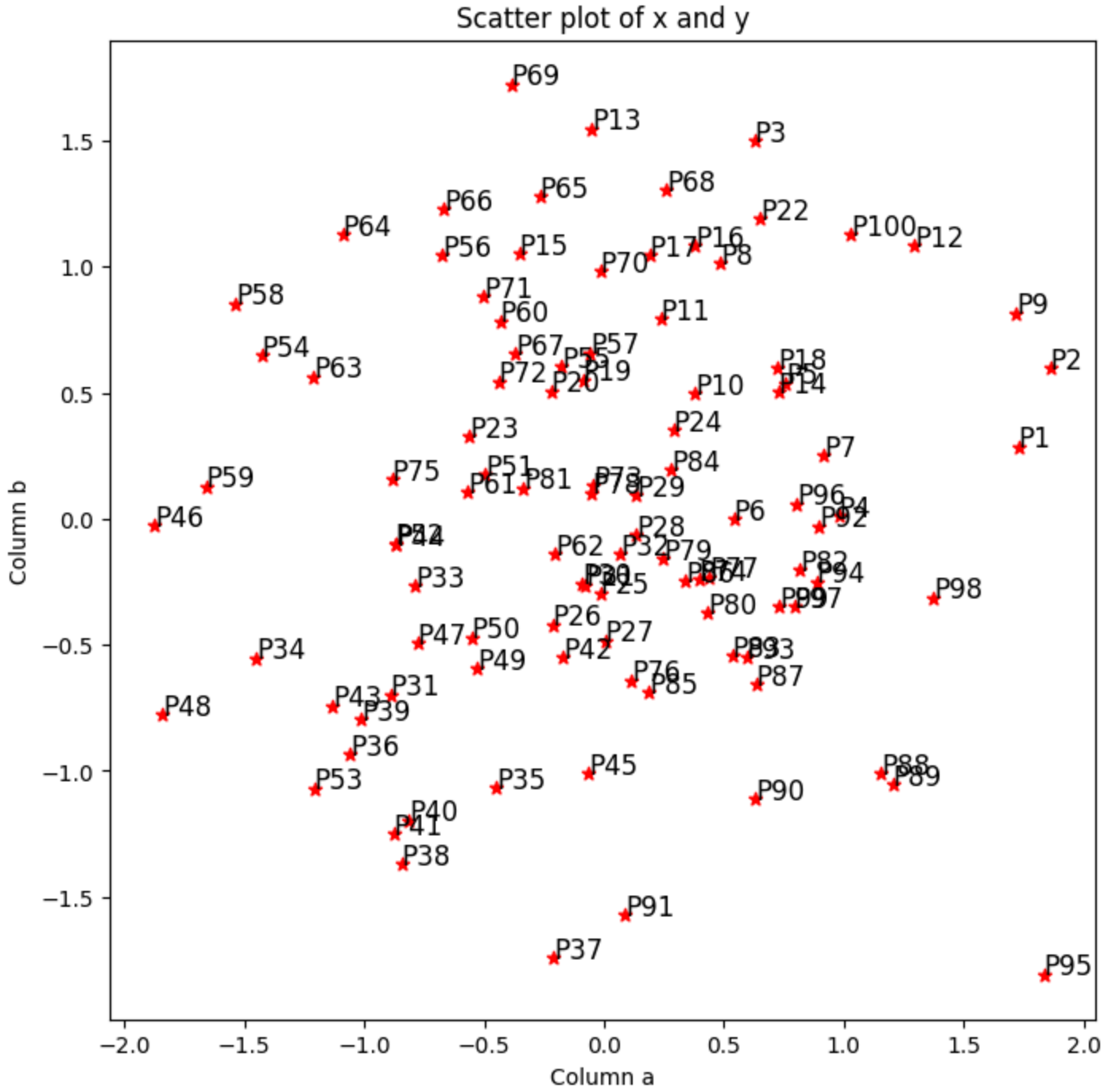
Dataset 2

In [ ]:
```
df2 = pd.read_csv('Datos/Activity2.csv') #Se lee la base de datos
df2.head()
num = np.arange(1,len(df2)+1) #Se crea un array con el numero de la fila
num = ['P'+str(i) for i in num] #Se convierte a string
df2['Punto'] = num #Se agrega al dataframe
df2.set_index('Punto',inplace=True) #Se cambia el indice
df2.rename(columns = {'0':'a','1':'b'},inplace=True) #Se cambian los nombres de las columnas
df2.head()
```

Out[ ]:

Punto	a	b
P1	1.727350	0.285771
P2	1.861161	0.597764
P3	0.627520	1.497373
P4	0.979559	0.008873
P5	0.756204	0.536461

In [ ]:
```
# Grafiuemos los datos
plt.figure(figsize=(8,8))
plt.scatter(df2['a'],df2['b'],c='r',marker='*')
plt.xlabel('Column a')
plt.ylabel('Column b')
plt.title('Scatter plot of x and y')
for j in df2.itertuples():
    plt.annotate(j.Index,(j.a,j.b),fontsize=12)
```


Scatter plot of x and y
Column a
Column b
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
P69
P13
P3
P65
P68
P66
P22
P64
P100
P12
P56
P15
P17
P16
P70
P8
P71
P58
P9
P60
P11
P54
P67
P57
P18
P2
P63
P72
P19
P10
P14
P23
P24
P1
P7
P59
P75
P51
P84
P61
P81
P29
P96
P46
P6
P28
P62
P32
P79
P82
P33
P25
P94
P98
P80
P26
P47
P50
P27
P34
P42
P49
P76
P85
P87
P31
P48
P43
P39
P36
P45
P53
P35
P90
P40
P41
P38
P91
P37
P95

In [ ]:
```
#Calculemos la matriz de distancias
dist = pd.DataFrame(squareform(pdist(df2,metric='euclidean')),columns = num,index = num) #Calculamos la matriz de dis
m = dist.values.max()
m
```

Out[ ]:
```
4.293391062091306
```

Rips 2

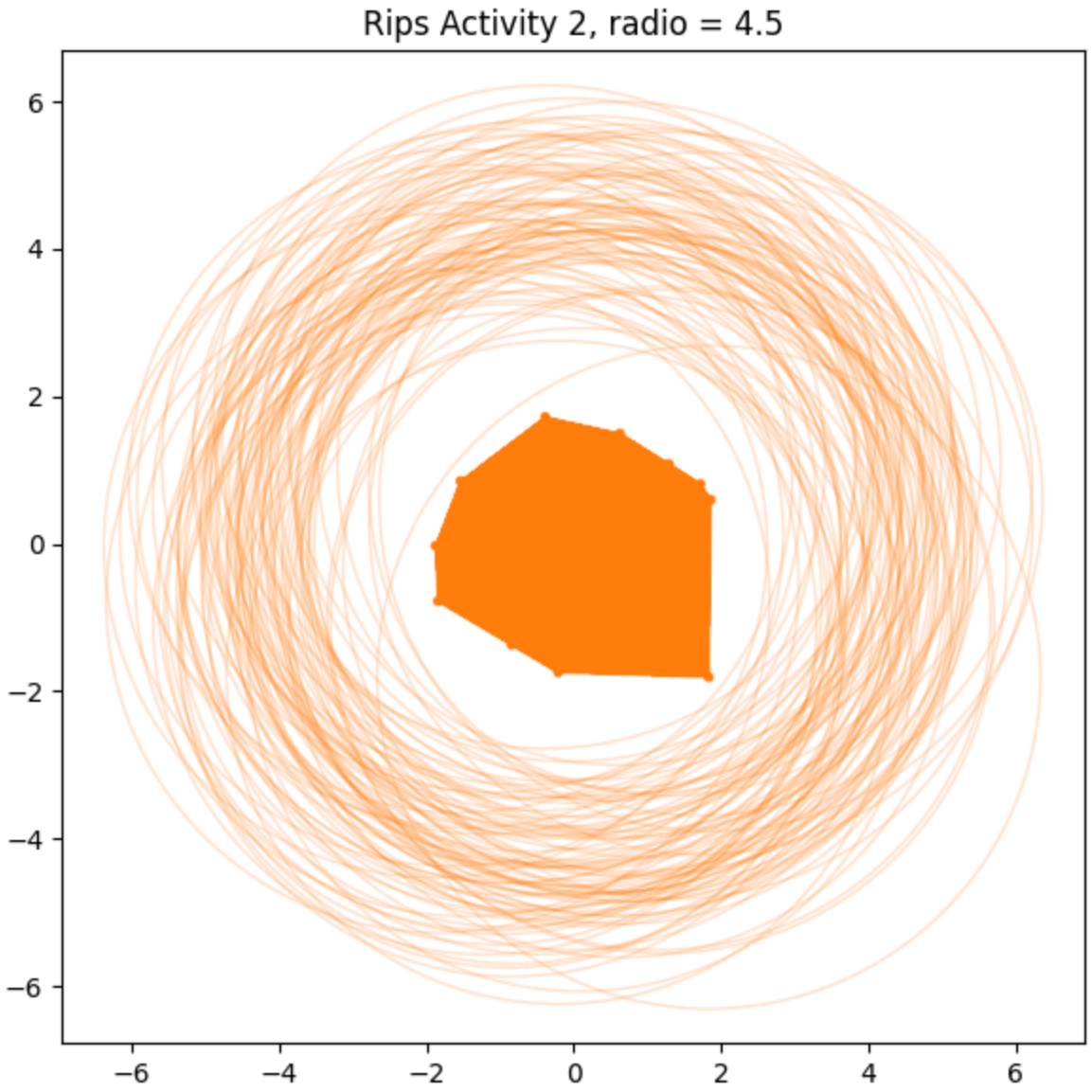
Radio = 4.5

In [ ]:
```
# Calculamos la filtracion de Rips con un radio de 4.5
rips_complex = gudhi.RipsComplex(distance_matrix=dist.values, max_edge_length=4.5)
```

In [ ]:
```
# Calculamos la filtracion de Rips con un radio de 4.5
simplex_tree = rips_complex.create_simplex_tree(max_dimension=2)
result_str = 'Rips complex is of dimension ' + repr(simplex_tree.dimension()) + ' - ' + \
               repr(simplex_tree.num_simplices()) + ' simplices - ' + \
               repr(simplex_tree.num_vertices()) + ' vertices.'
print(result_str)
fmt = '%s -> %.2f'
#for filtered_value in simplex_tree.get_filtration():
#    print(fmt % tuple(filtered_value))
```

```
Rips complex is of dimension 2 - 324 simplices - 100 vertices.
```

In [ ]:
```
# Graficamos el complejo de Rips
plot_rips_complex(df2.values,R=4.5,label='Rips Activity 2, radio = 4.5',maxdim=2)
```



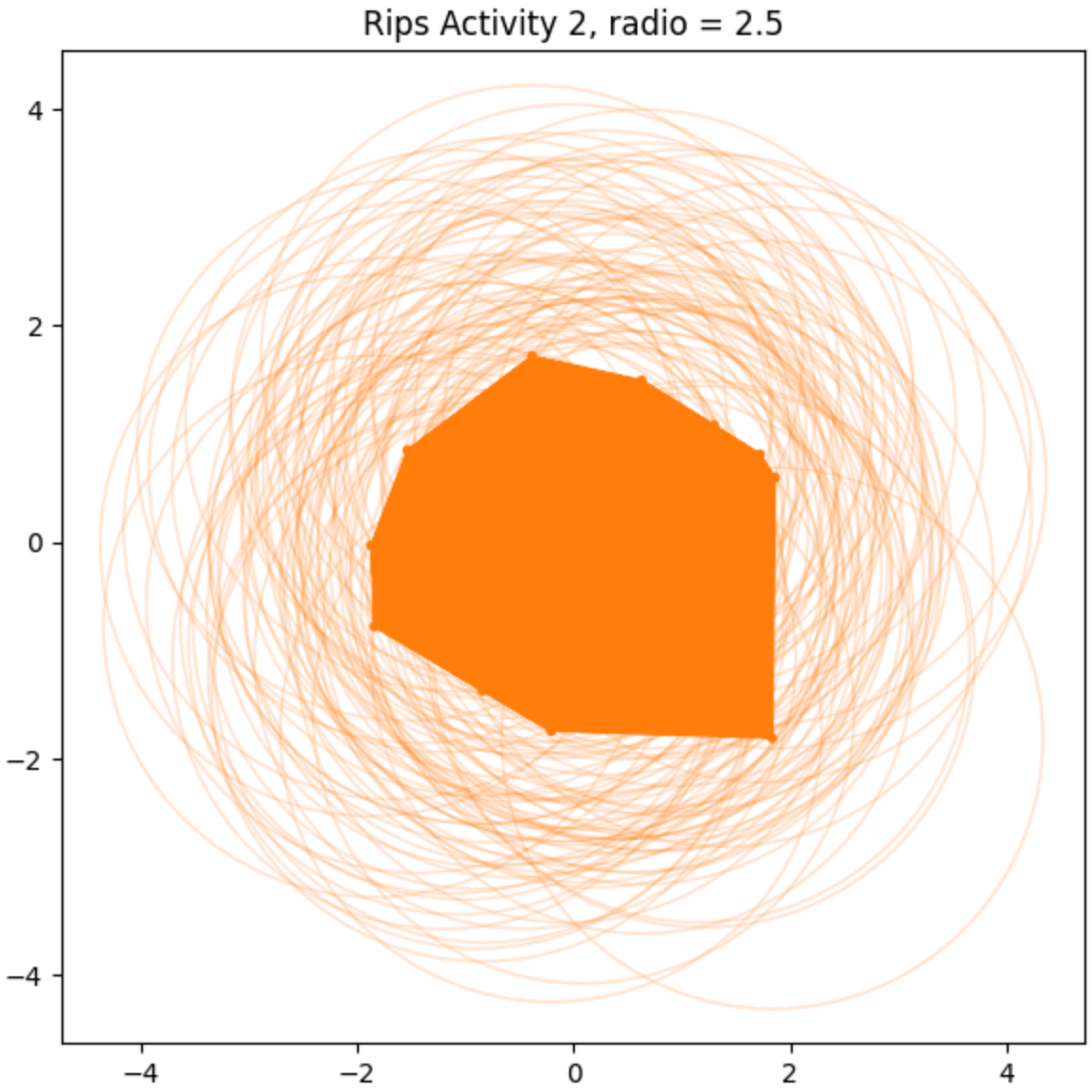
Radio = 2.5

In [ ]:
```
# Calculamos la filtracion de Rips con un radio de 2.5
rips_complex = gudhi.RipsComplex(distance_matrix=dist.values, max_edge_length=2.5)
simplex_tree = rips_complex.create_simplex_tree(max_dimension=2)
```

```
result_str = 'Rips complex is of dimension ' + repr(simplex_tree.dimension()) + ' - ' + \
               repr(simplex_tree.num_simplices()) + ' simplices - ' + \
               repr(simplex_tree.num_vertices()) + ' vertices.'
print(result_str)
fmt = '%s -> %.2f'
#for filtered_value in simplex_tree.get_filtration():
 #   print(fmt % tuple(filtered_value))
```

```
Rips complex is of dimension 2 - 593 simplices - 100 vertices.
```

In [ ]:
```
# Graficamos el complejo de Rips
plot_rips_complex(df2.values,R=2.5,label='Rips Activity 2, radio = 2.5',maxdim=2)
```



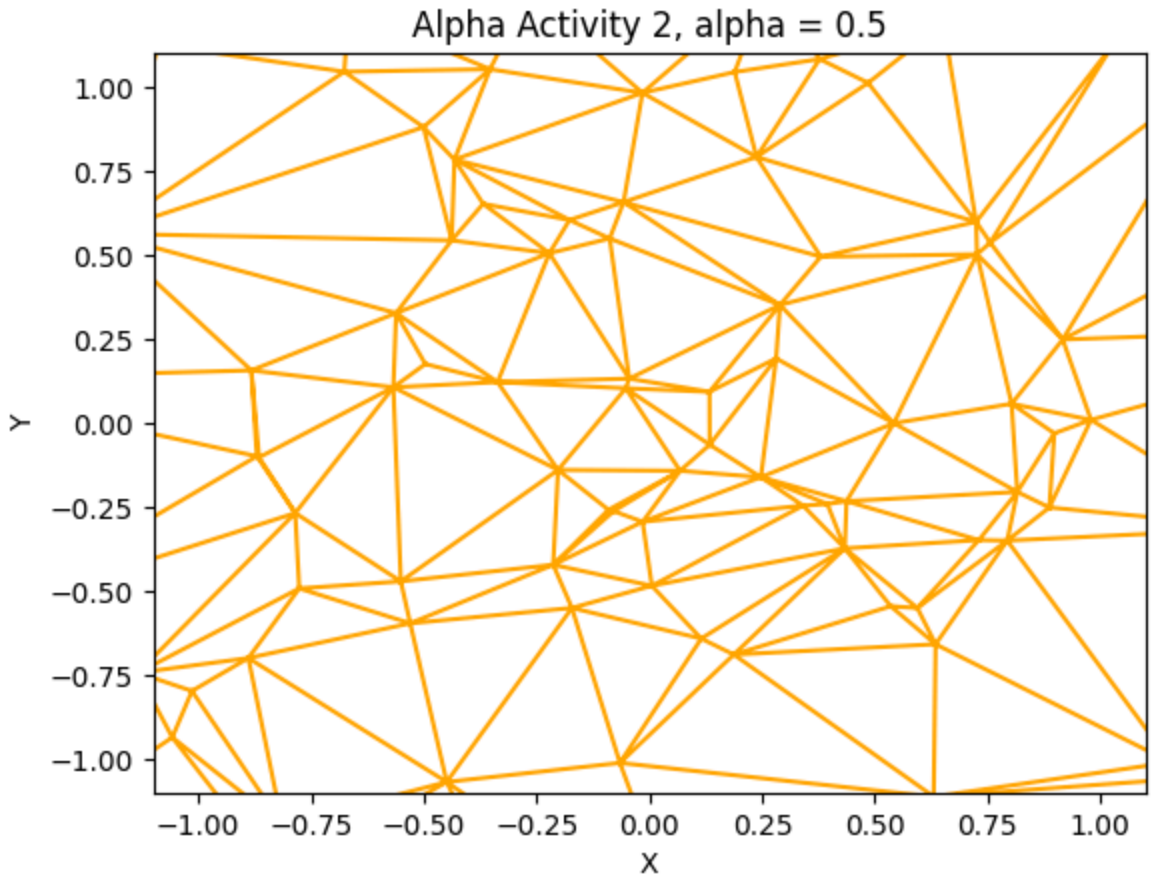
Alpha 2

Alpha de 0.5

In [ ]:
```
# Calculamos la filtracion de Alpha con un alpha de 0.5
alpha_complex = gudhi.AlphaComplex(points=df2.values)
simplex_tree = alpha_complex.create_simplex_tree(max_alpha_square=0.5)
result_str = 'Alpha complex is of dimension ' + repr(simplex_tree.dimension()) + ' - ' + \
               repr(simplex_tree.num_simplices()) + ' simplices - ' + \
               repr(simplex_tree.num_vertices()) + ' vertices.'
print(result_str)
fmt = '%s -> %.2f'
#for filtered_value in simplex_tree.get_filtration():
#    print(fmt % tuple(filtered_value))
```

```
Alpha complex is of dimension 2 - 553 simplices - 100 vertices.
```

In [ ]:
```
# Graficamos el complejo de Alpha
plot_alpha_complex(alpha_complex, label='Alpha Activity 2, alpha = 0.5', maxdim=2)
```

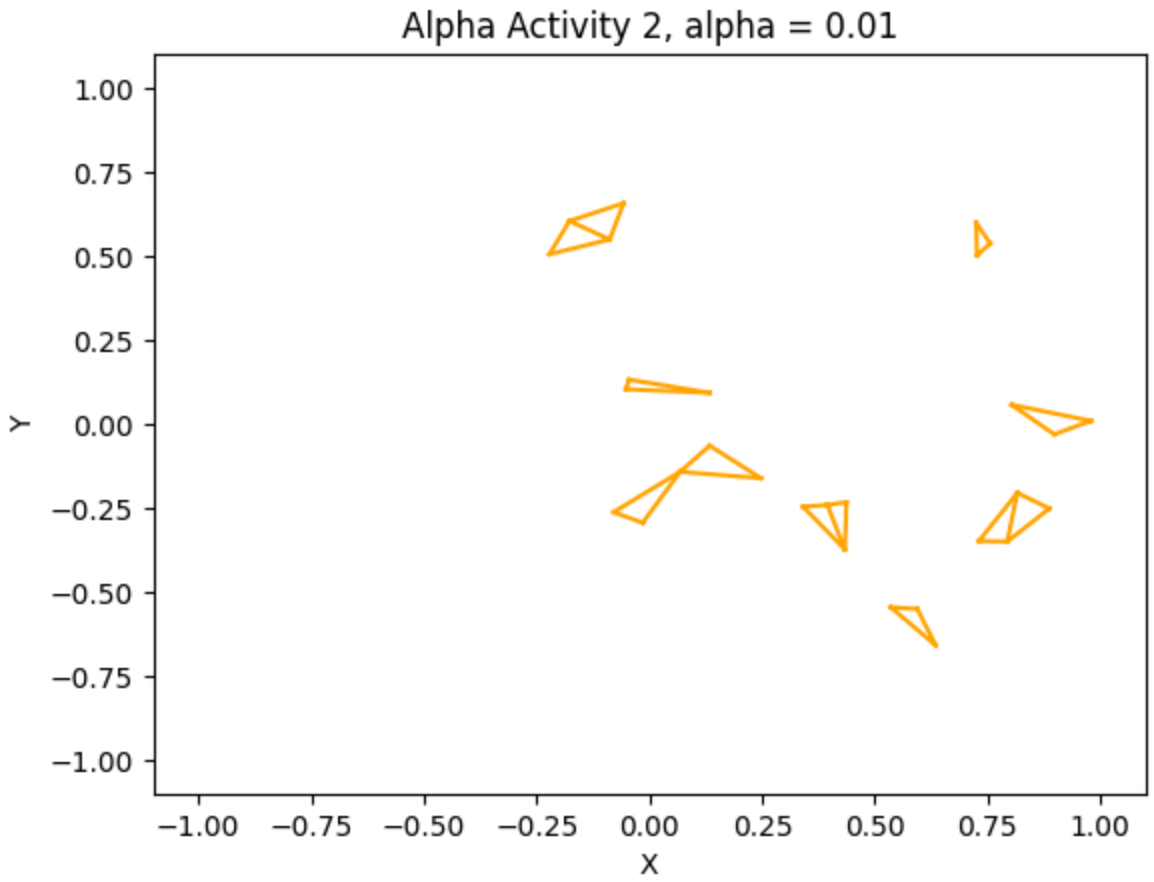


Alpha de 0.01

In [ ]:
```
# Calculamos la filtracion de Alpha con un alpha de 0.01
alpha_complex = gudhi.AlphaComplex(points=df2.values)
simplex_tree = alpha_complex.create_simplex_tree(max_alpha_square=0.01)
result_str = 'Alpha complex is of dimension ' + repr(simplex_tree.dimension()) + ' - ' + \
               repr(simplex_tree.num_simplices()) + ' simplices - ' + \
               repr(simplex_tree.num_vertices()) + ' vertices.'
print(result_str)
fmt = '%s -> %.2f'
#for filtered_value in simplex_tree.get_filtration():
 #   print(fmt % tuple(filtered_value))
```

```
Alpha complex is of dimension 2 - 179 simplices - 100 vertices.
```

In [ ]:
```
# Graficamos el complejo de Alpha
plot_alpha_complex(alpha_complex, label='Alpha Activity 2, alpha = 0.01', maxdim=2)
```



Dataset 3

In [ ]:
```
#Se lee la base de datos
df3 = pd.read_csv('Datos/Activity3.csv')
df3.head()
#Se crea un array con el numero de la fila
num = np.arange(1,len(df3)+1)
#Se convierte a string
```

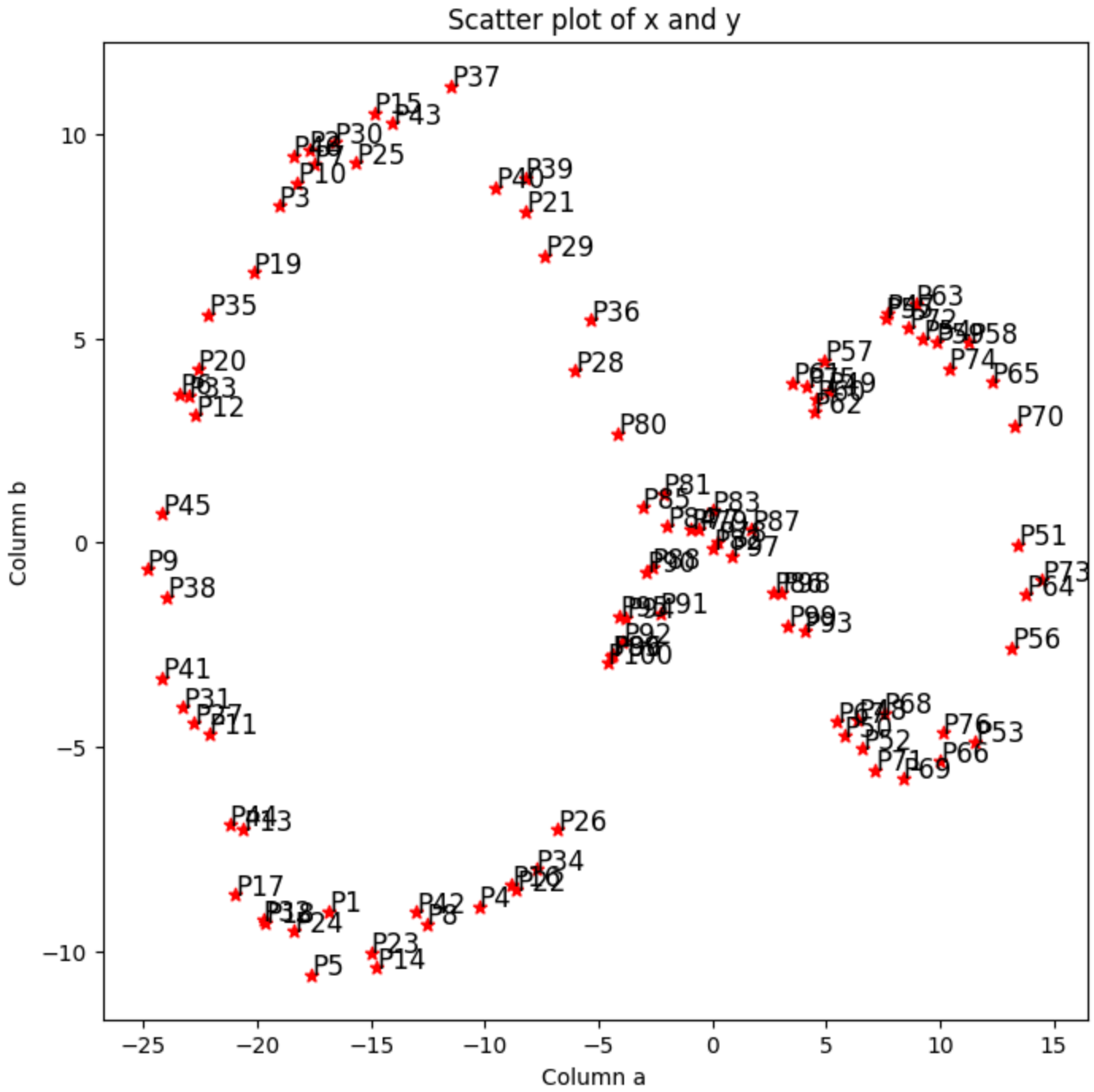
```
num = ['P'+str(i) for i in num]
#Se agrega al dataframe
df3['Punto'] = num
#Se cambia el indice
df3.set_index('Punto',inplace=True)
#Se cambian los nombres de las columnas
df3.rename(columns = {'0':'a','1':'b'},inplace=True)
df3.head()
```

Out[ ]:

Punto	a	b
P1	-16.865905	-9.030913
P2	-17.721247	9.601550
P3	-19.047260	8.233957
P4	-10.252216	-8.925738
P5	-17.636004	-10.591064

In [ ]:

```
# Grafiuemos los datos
plt.figure(figsize=(8,8))
plt.scatter(df3['a'],df3['b'],c='r',marker='*')
plt.xlabel('Column a')
plt.ylabel('Column b')
plt.title('Scatter plot of x and y')
for j in df3.itertuples():
    plt.annotate(j.Index,(j.a,j.b),fontsize=12)
```


Scatter plot of x and y
Column a
Column b
10
5
0
−5
−10
−25
−20
−15
−10
−5
0
5
10
15
P37
P15
P43
P30
P25
P10
P3
P40
P39
P21
P29
P19
P35
P36
P28
P20
P12
P80
P63
P58
P57
P74
P65
P62
P70
P81
P83
P87
P45
P51
P9
P38
P73
P64
P91
P93
P56
P41
P31
P11
P68
P76
P53
P66
P69
P26
P34
P17
P1
P42
P4
P24
P23
P14
P5

In [ ]:
```
#Calculemos la matriz de distancias
dist = pd.DataFrame(squareform(pdist(df3,metric='euclidean')),columns = num,index = num)
m = dist.values.max()
m
```

Out[ ]:
```
39.34119889823739
```

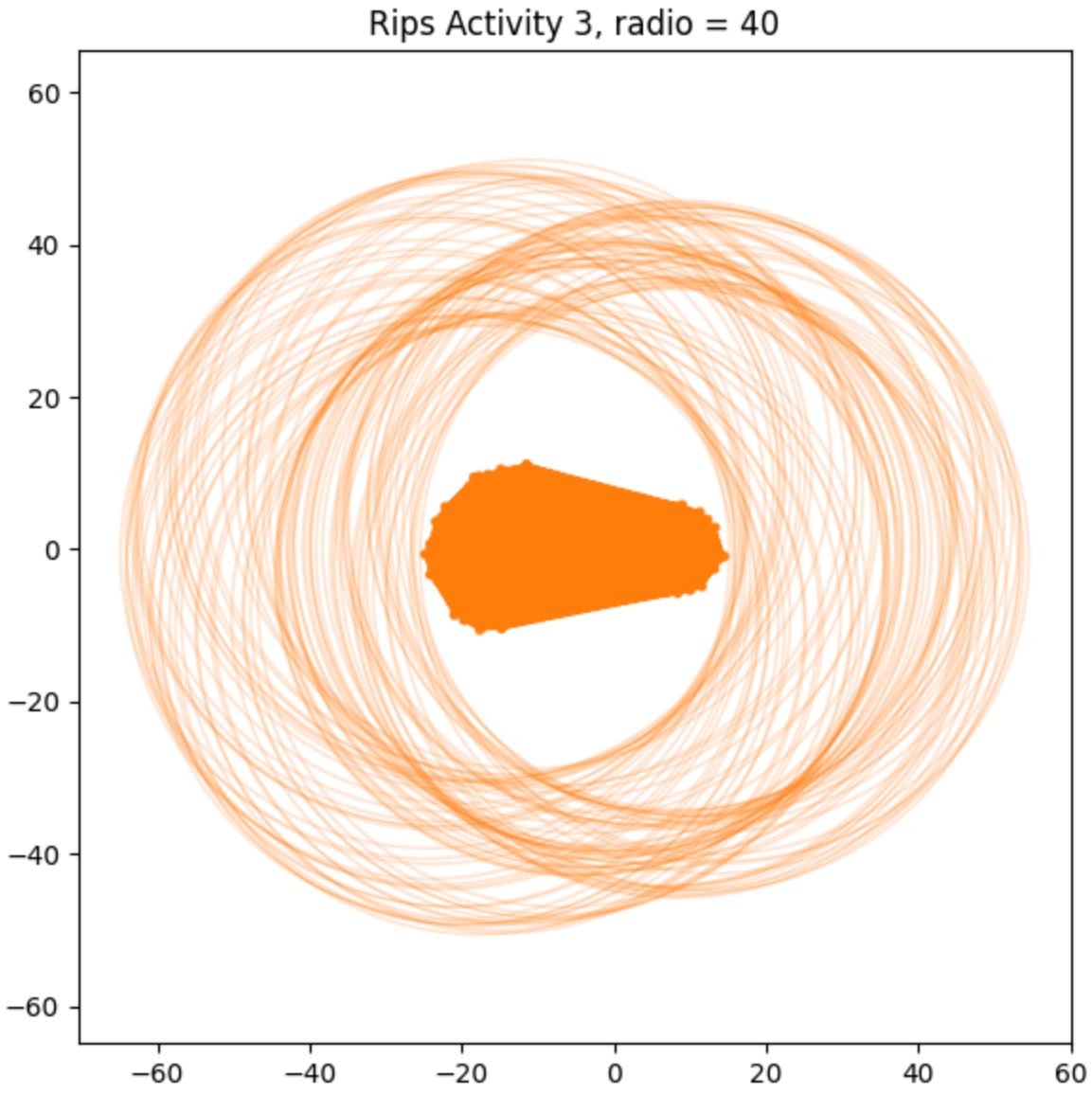
Rips 3

Radio = 40

In [ ]:
```
# Calculamos la filtracion de Rips con un radio de 40
rips_complex = gudhi.RipsComplex(distance_matrix=dist.values, max_edge_length=40)
simplex_tree = rips_complex.create_simplex_tree(max_dimension=2)
result_str = 'Rips complex is of dimension ' + repr(simplex_tree.dimension()) + ' - ' + \
               repr(simplex_tree.num_simplices()) + ' simplices - ' + \
               repr(simplex_tree.num_vertices()) + ' vertices.'
print(result_str)
fmt = '%s -> %.2f'
#for filtered_value in simplex_tree.get_filtration():
 #   print(fmt % tuple(filtered_value))
```

```
Rips complex is of dimension 2 - 166750 simplices - 100 vertices.
```

In [ ]:
```
# Graficamos el complejo de Rips
plot_rips_complex(df3.values,R=40,label='Rips Activity 3, radio = 40',maxdim=2)
```



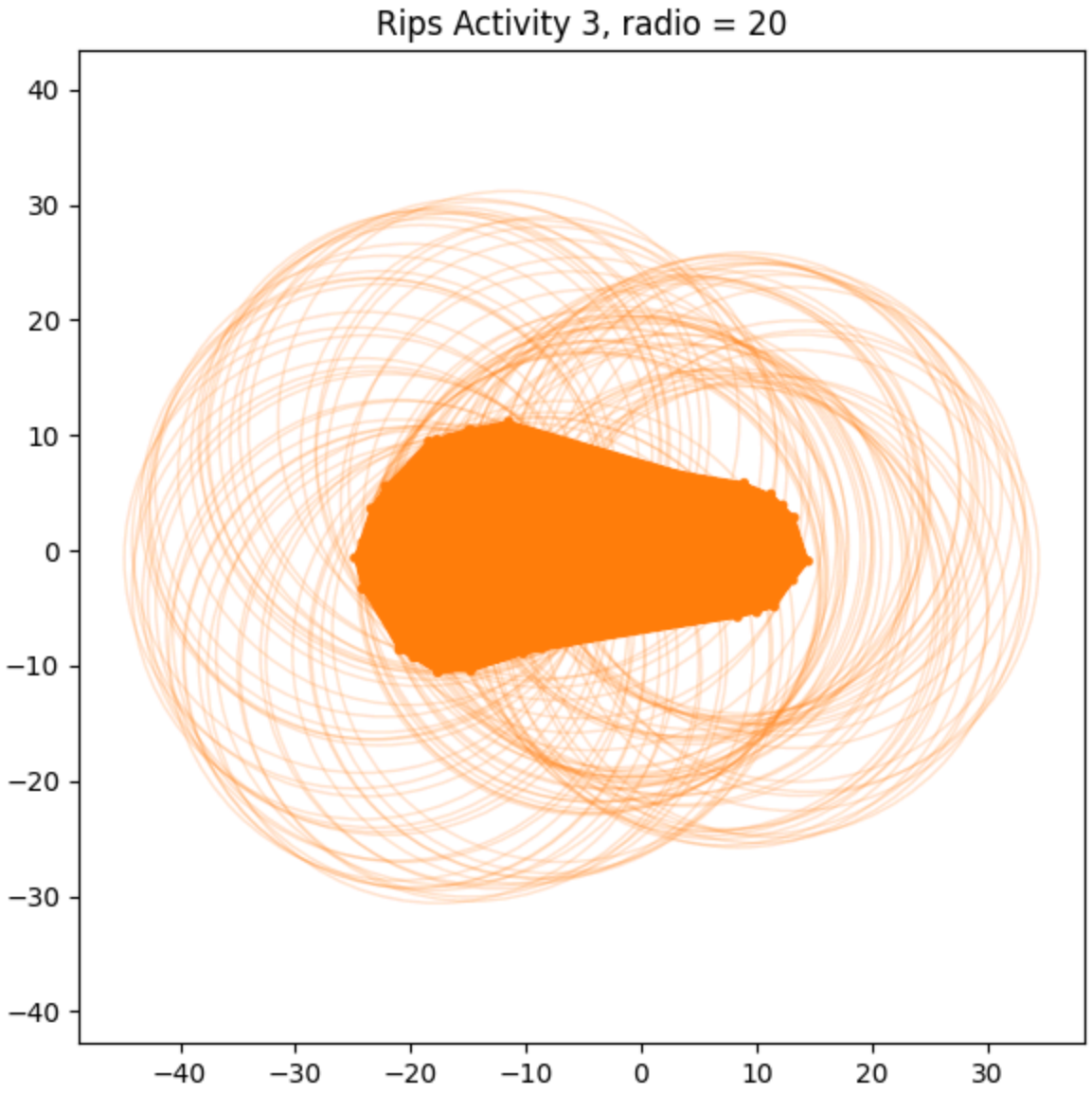
Radio = 20

In [ ]:
```
# Calculamos la filtracion de Rips con un radio de 20
rips_complex = gudhi.RipsComplex(distance_matrix=dist.values, max_edge_length=20)
simplex_tree = rips_complex.create_simplex_tree(max_dimension=2)
```

```
result_str = 'Rips complex is of dimension ' + repr(simplex_tree.dimension()) + ' - ' + \
               repr(simplex_tree.num_simplices()) + ' simplices - ' + \
               repr(simplex_tree.num_vertices()) + ' vertices.'
print(result_str)
fmt = '%s -> %.2f'
#for filtered_value in simplex_tree.get_filtration():
 #   print(fmt % tuple(filtered_value))
```

```
Rips complex is of dimension 2 - 65577 simplices - 100 vertices.
```

In [ ]:
```
# Graficamos el complejo de Rips
plot_rips_complex(df3.values,R=20,label='Rips Activity 3, radio = 20',maxdim=2)
```



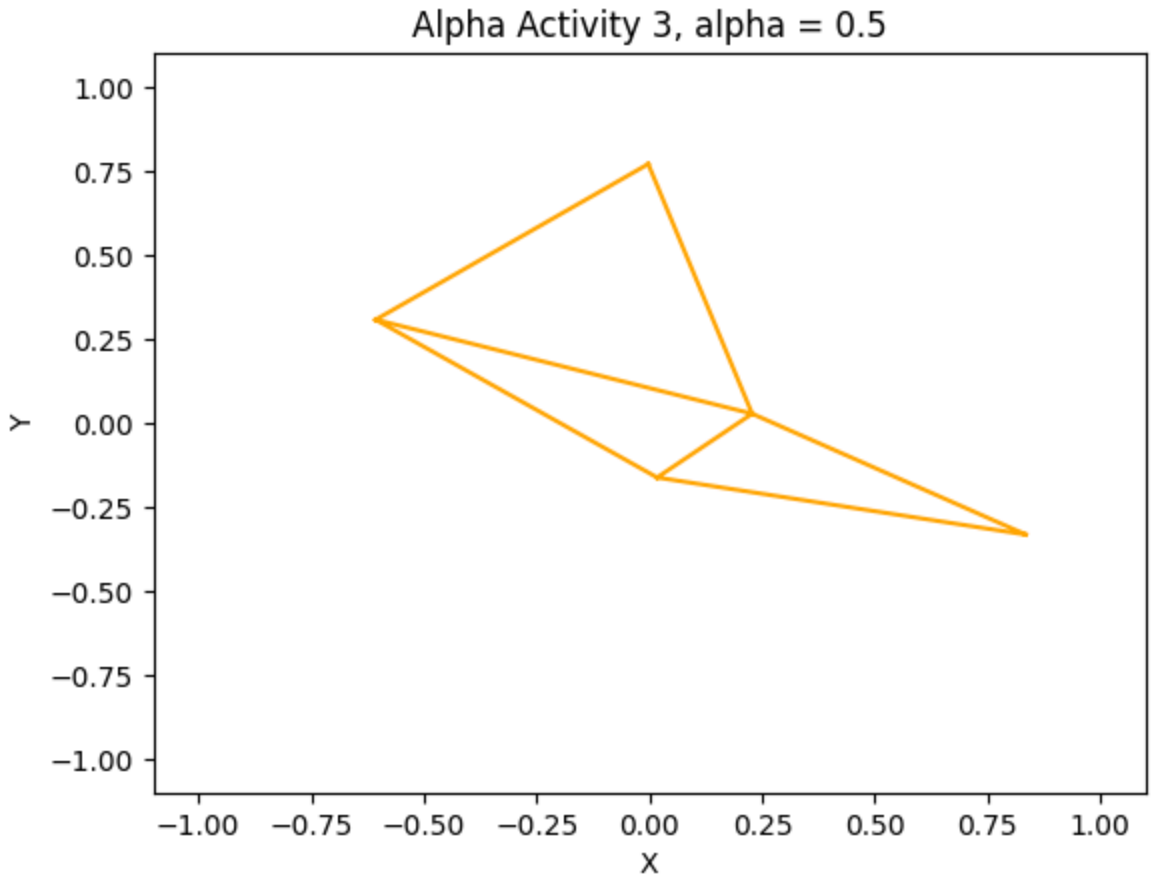
Alpha 3

Alpha de 0.5

In [ ]:
```
# Calculamos la filtracion de Alpha con un alpha de 0.5
alpha_complex = gudhi.AlphaComplex(points=df3.values)
simplex_tree = alpha_complex.create_simplex_tree(max_alpha_square=0.5)
result_str = 'Alpha complex is of dimension ' + repr(simplex_tree.dimension()) + ' - ' + \
               repr(simplex_tree.num_simplices()) + ' simplices - ' + \
               repr(simplex_tree.num_vertices()) + ' vertices.'
print(result_str)
fmt = '%s -> %.2f'
#for filtered_value in simplex_tree.get_filtration():
    #print(fmt % tuple(filtered_value))
```

```
Alpha complex is of dimension 2 - 225 simplices - 100 vertices.
```

In [ ]:
```
# Graficamos el complejo de Alpha
plot_alpha_complex(alpha_complex, label='Alpha Activity 3, alpha = 0.5', maxdim=2)
```



Alpha de 0.01

In [ ]:
```
# Calculamos la filtracion de Alpha con un alpha de 0.01
alpha_complex = gudhi.AlphaComplex(points=df3.values)
simplex_tree = alpha_complex.create_simplex_tree(max_alpha_square=0.01)
result_str = 'Alpha complex is of dimension ' + repr(simplex_tree.dimension()) + ' - ' + \
               repr(simplex_tree.num_simplices()) + ' simplices - ' + \
               repr(simplex_tree.num_vertices()) + ' vertices.'
print(result_str)
fmt = '%s -> %.2f'
#for filtered_value in simplex_tree.get_filtration():
    #print(fmt % tuple(filtered_value))
```

```
Alpha complex is of dimension 1 - 103 simplices - 100 vertices.
```

In [ ]:
```
# Graficamos el complejo de Alpha
plot_alpha_complex(alpha_complex, label='Alpha Activity 3, alpha = 0.01', maxdim=2)
```