Actividad 1: Rips, Cech, Alpha

Annette Pamela Ruiz Abreu

A01423595

In [ ]:
```
# Librerías
import numpy as np  # Para arreglos
import pandas as pd # Para bases de datos
import matplotlib.pyplot as plt # Para graficar
from matplotlib import cm # Para colores
from scipy.spatial.distance import squareform, pdist # Para matrices de distancias
import matplotlib.patches as mpatches # Para elipses
from matplotlib.collections import PatchCollection # Para elipses
import gudhi # Para hacer la filtración de complejos cimpliciales de Rips y Alpha
import seaborn as sns
```

Base de Datos 1

In [ ]:
```
df1 = pd.read_csv('Activity1.csv') # Cargamos la base de datos

# Mostramos la base de datos
display(df1.head())
display(df1.describe())
print("Tamaño del dataframe:", df1.shape)
print("Tipos de datos del dataframe:")
display(df1.dtypes)
```

	0	1
0	-0.062332	-0.990463
1	1.109356	-0.077222
2	0.553080	-0.938321
3	0.290183	0.813677
4	-0.722770	-0.380330

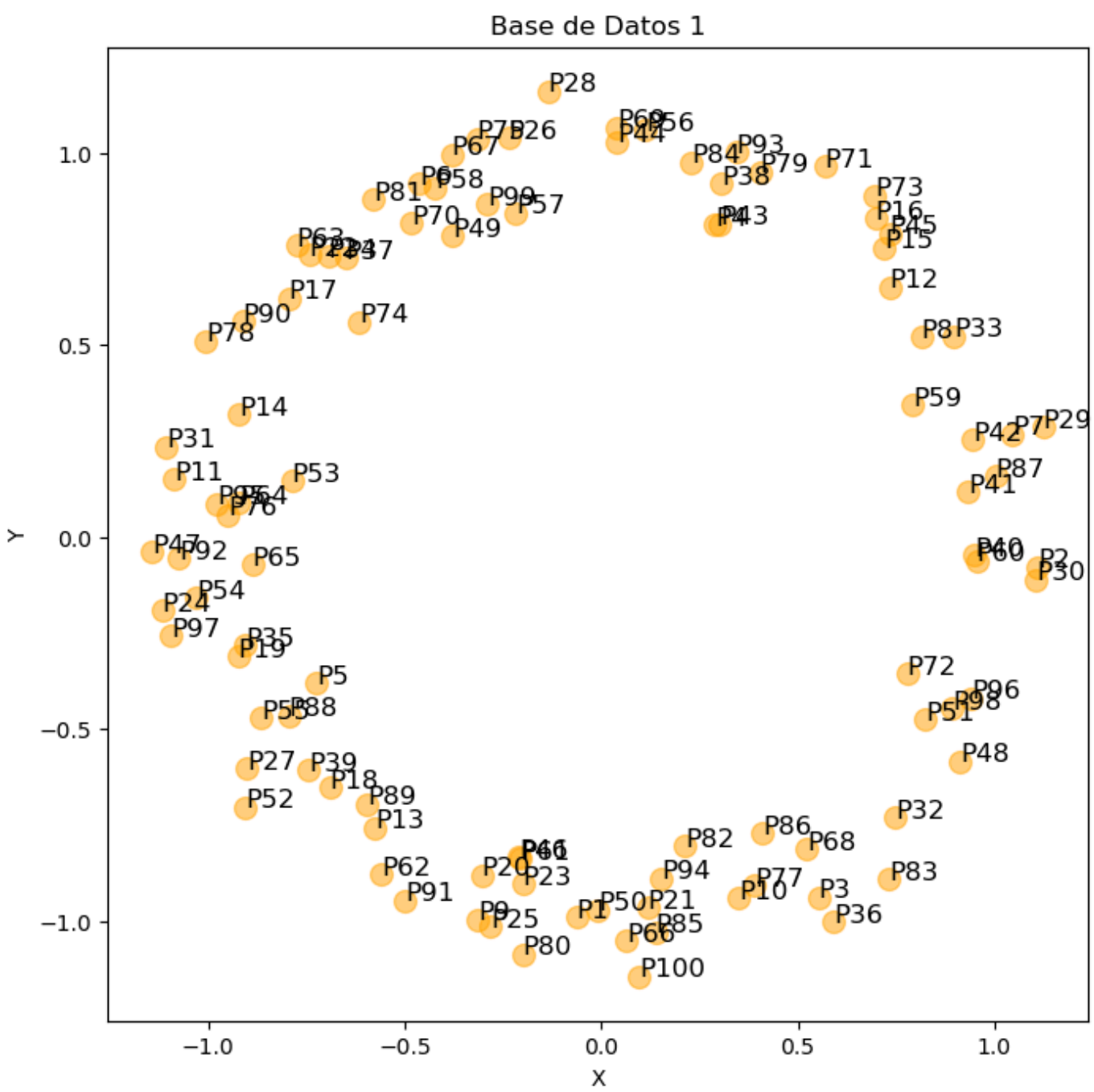
	0	1
count	100.000000	100.000000
mean	-0.073169	0.000463
std	0.700289	0.732798
min	-1.142208	-1.144156
25%	-0.727509	-0.737348
50%	-0.198949	-0.043047
75%	0.575066	0.755372
max	1.124707	1.160988

```
Tamaño del dataframe: (100, 2)
Tipos de datos del dataframe:
```

```
0    float64
1    float64
dtype: object
```

In [ ]:

```python
# Creamos una columna nueva con el número del punto (P1)
num = np.arange(1,len(df1)+1,1)
num = ['P'+str(i) for i in num]
df1["Point"] = num

# Convertimos la columna "Point" en el índice de la tabla
df1.set_index('Point', inplace=True)
df1.head()

# Graficamos los puntos con el índice como etiqueta
plt.figure(figsize=(8,8))
plt.scatter(df1['0'], df1['1'], s=100, c='orange', alpha=0.5)
for i, txt in enumerate(df1.index):
    plt.annotate(txt, (df1['0'][i], df1['1'][i]), fontsize=12)
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Base de Datos 1')
plt.show()
```



A primera vista después de graficar todos los puntos de la base de datos 1, se puede observar que los puntos están distribuidos en un círculo, por lo que se espera que los diagramas de persistencia de Rips, Cech y Alpha sean similares.

Complejo de Rips

Nuestra base de datos está en el plano R^2, por lo que la dimensión máxima del complejo simplicial de Rips debe ser 2. El parámetro r es la distancia máxima entre dos puntos para que se conecten por una arista.

In [ ]:
```
dist = pd.DataFrame(squareform(pdist(df1), "euclidian"), columns=df1.index.values, index

max_dist = dist.values.max()
print("La distancia máxima entre los puntos es:", round(max_dist, 5))
```

```
La distancia máxima entre los puntos es: 2.31669
```

Dada la distancia máxima, podemos elegir $r = 2.5$

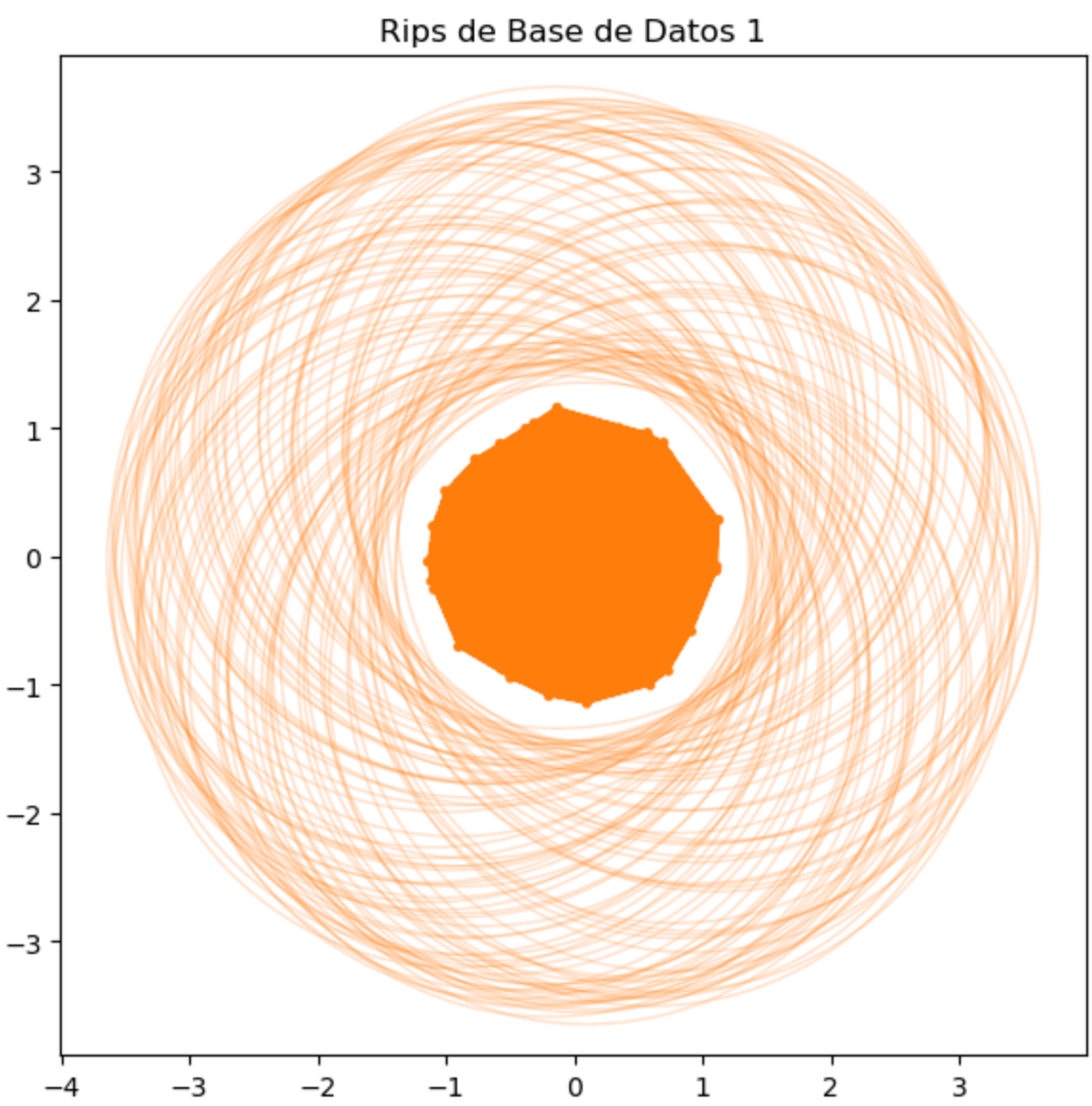
In [ ]:
```
rips_complex = gudhi.RipsComplex(distance_matrix=dist.values, max_edge_length=2.5)

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 - 166750 simplices - 100 vertices.
```

In [ ]:
```
# Rips complex graph function
def plot_rips_complex(data, R, label="data", col=1, maxdim=2):
    tab10 = cm.get_cmap('tab10')

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

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

    for i, xy in enumerate(data):
        if maxdim >=1:
            for j in range(i + 1, len(data)):
                pq = data[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
                if maxdim == 2:
                    for k in range(j + 1, len(data)):
                        ab = data[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.
                        pass

    plt.axis('equal')
    plt.tight_layout()
    plt.show()
    pass
```

In [ ]:
```
plot_rips_complex(df1.values, R=2.5, label="Rips de Base de Datos 1", col=1, maxdim=2)
```

```
/var/folders/dd/fhmd_dws0_d412cq_s690s3c0000gn/T/ipykernel_61183/3813265713.py:3: Matplo
tlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib 3.7 and will
be removed two minor releases later. Use ``matplotlib.colormaps[name]`` or ``matplotlib.
colormaps.get_cmap(obj)`` instead.
  tab10 = cm.get_cmap('tab10')
```



Observando la gráfica del complejo simplicial de Rip, se puede observar que el complejo simplicial de Rips es un círculo, lo cual es esperado.

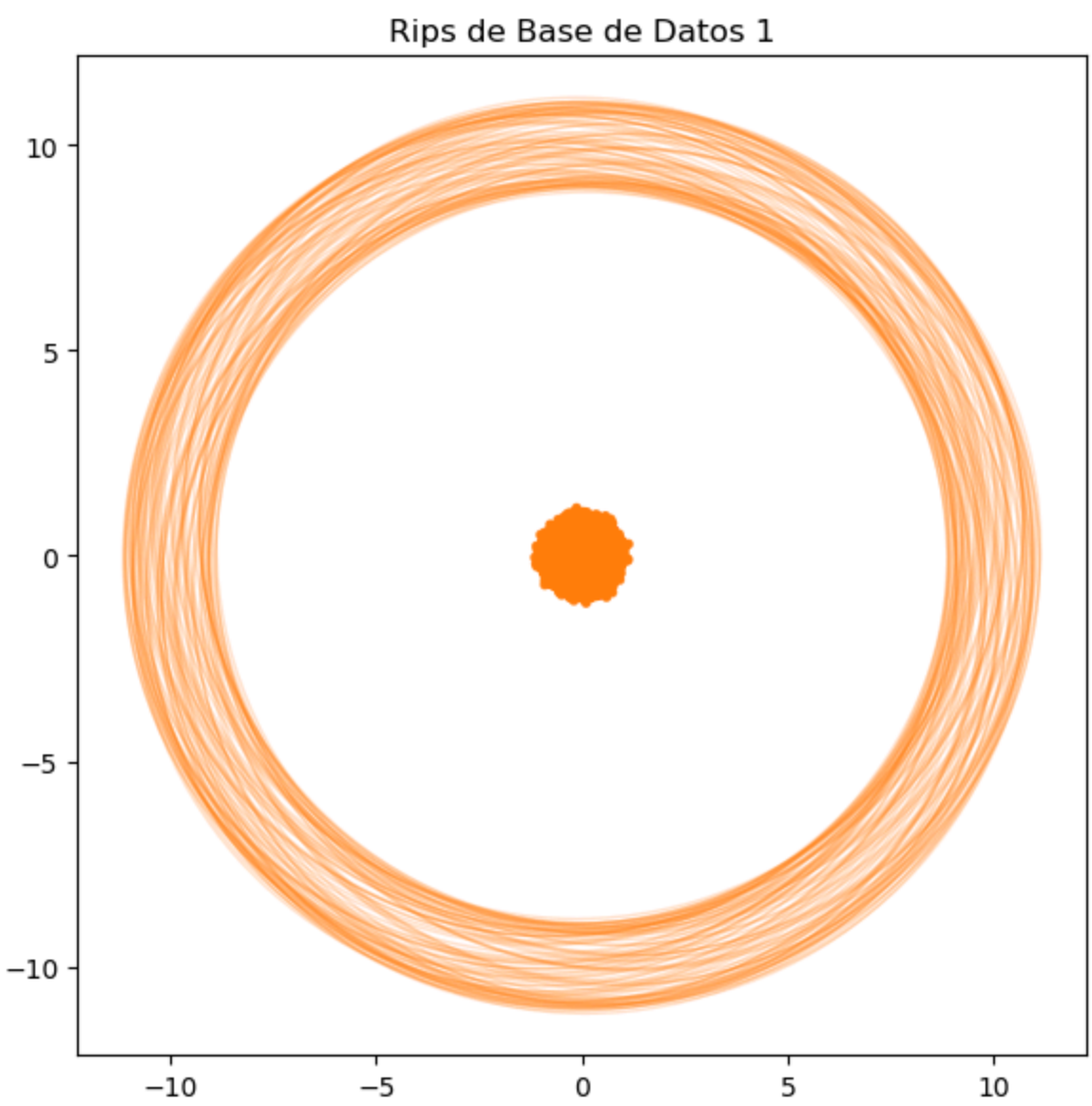
Si experimentamos con otros valores de r, podemos observar que el complejo simplicial de Rips se mantiene como un círculo. Por ejemplo, probaremos $r = 10$.

In [ ]:
```
rips_complex = gudhi.RipsComplex(distance_matrix=dist.values, max_edge_length=10)

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 [ ]:
```
plot_rips_complex(df1.values, R=10, label="Rips de Base de Datos 1", col=1, maxdim=2)
```

```
/var/folders/dd/fhmd_dws0_d412cq_s690s3c0000gn/T/ipykernel_61183/3813265713.py:3: Matplo
tlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib 3.7 and will
be removed two minor releases later. Use ``matplotlib.colormaps[name]`` or ``matplotlib.
colormaps.get_cmap(obj)`` instead.
  tab10 = cm.get_cmap('tab10')
```



Al probar con un valor de $r = 10$, se puede observar que el complejo simplicial de Rips sigue siendo un círculo; sin embargo, la distancia entre el centro y las cirfunferencias es mucho mayor que en el caso anterior. Entre más grande sea el radio, más grande será el círculo y más se "perfeccionará".

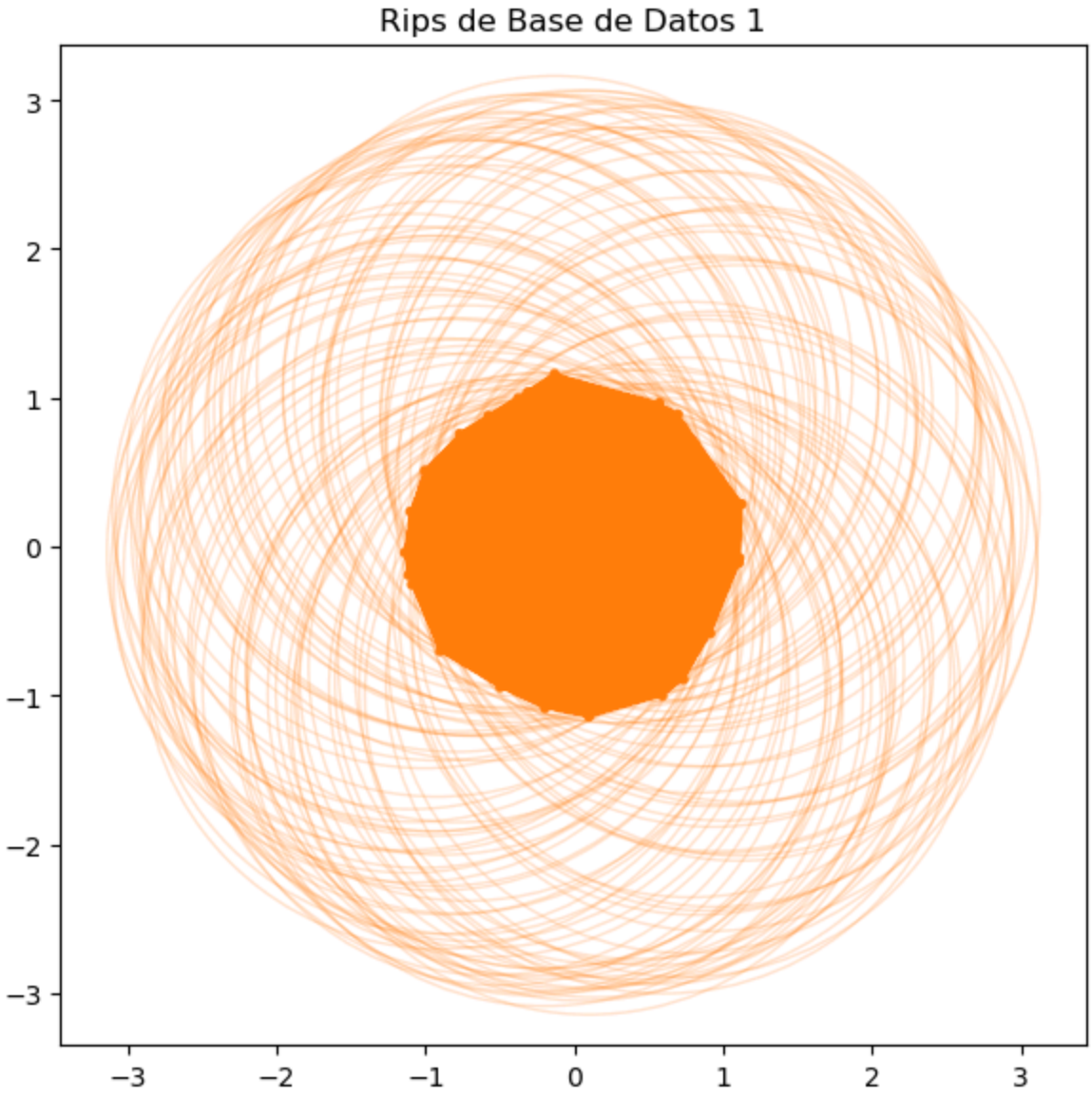
Si probamos un valor de $r = 2$, que es menor que la distancia máxima, se puede observar que el complejo simplicial de Rips sigue siendo un círculo, pero ahora los puntos están más juntos y que la cantidad de simplejos que se forman es menor.

In [ ]:
```
rips_complex = gudhi.RipsComplex(distance_matrix=dist.values, max_edge_length=2)

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 - 122305 simplices - 100 vertices.
```

In [ ]:
```
plot_rips_complex(df1.values, R=2, label="Rips de Base de Datos 1", col=1, maxdim=2)
```

```
/var/folders/dd/fhmd_dws0_d412cq_s690s3c0000gn/T/ipykernel_61183/3813265713.py:3: Matplo
tlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib 3.7 and will
be removed two minor releases later. Use ``matplotlib.colormaps[name]`` or ``matplotlib.
colormaps.get_cmap(obj)`` instead.
  tab10 = cm.get_cmap('tab10')
```



Complejo de Alpha

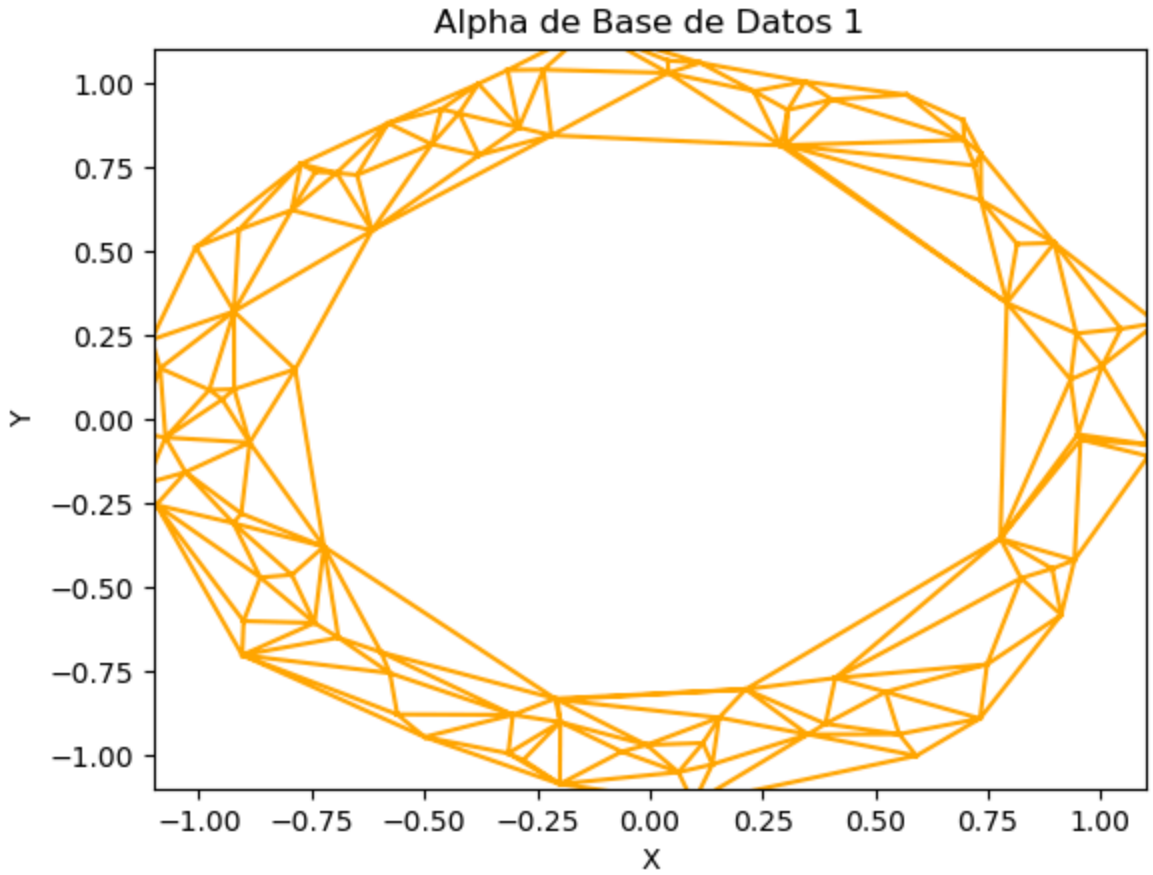
El complejo de Alpha es un complejo simplicial que se construye a partir de un conjunto de puntos en el plano. La idea es que el complejo de Alpha es un subcomplejo del complejo de Delaunay, y se construye a partir de la intersección de los discos de radio α centrados en cada punto.

Entre más grande el valor de α, más simplejos se formarán y mejor será la visualización.

In [ ]:
```
alpha_complex = gudhi.AlphaComplex(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 [ ]:
```
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

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("Alpha de Base de Datos 1")

plt.show()
```



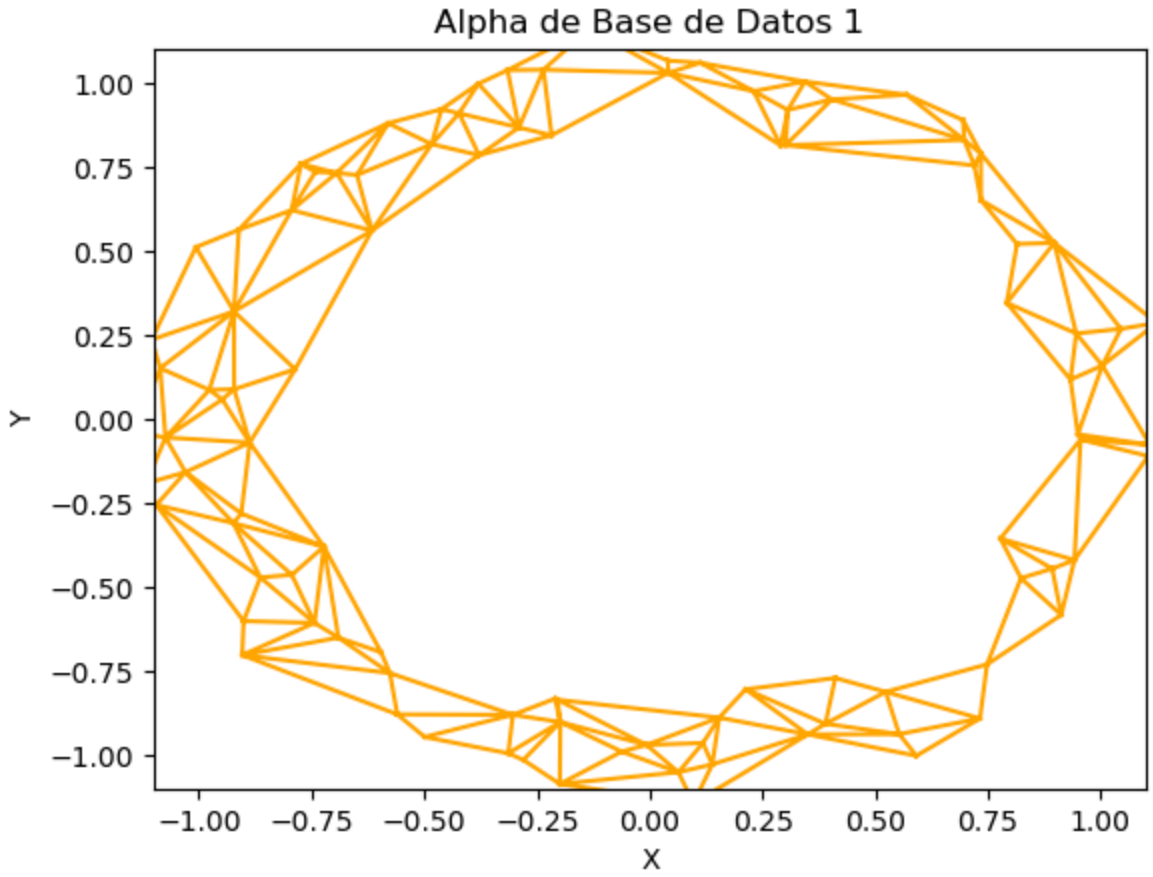
Con $\alpha = 0.5$ podemos observar que el complejo simplicial de Alpha es un círculo, lo cual es esperado. Es un poco diferente al complejo de Rips porque el complejo de Alpha es más sensible a la densidad de los puntos mientras que el complejo de Rips solo se basa en la distancia entre los puntos.

Si tomamos un valor de α muy pequeño, se forman menos simplejos y puede empeorar la visualización si no hay suficientes.

In [ ]:
```
alpha_complex = gudhi.AlphaComplex(df1.values)
simplex_tree = alpha_complex.create_simplex_tree(max_alpha_square = 0.05)
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 - 470 simplices - 100 vertices.
```

In [ ]:
```
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

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("Alpha de Base de Datos 1")

plt.show()
```



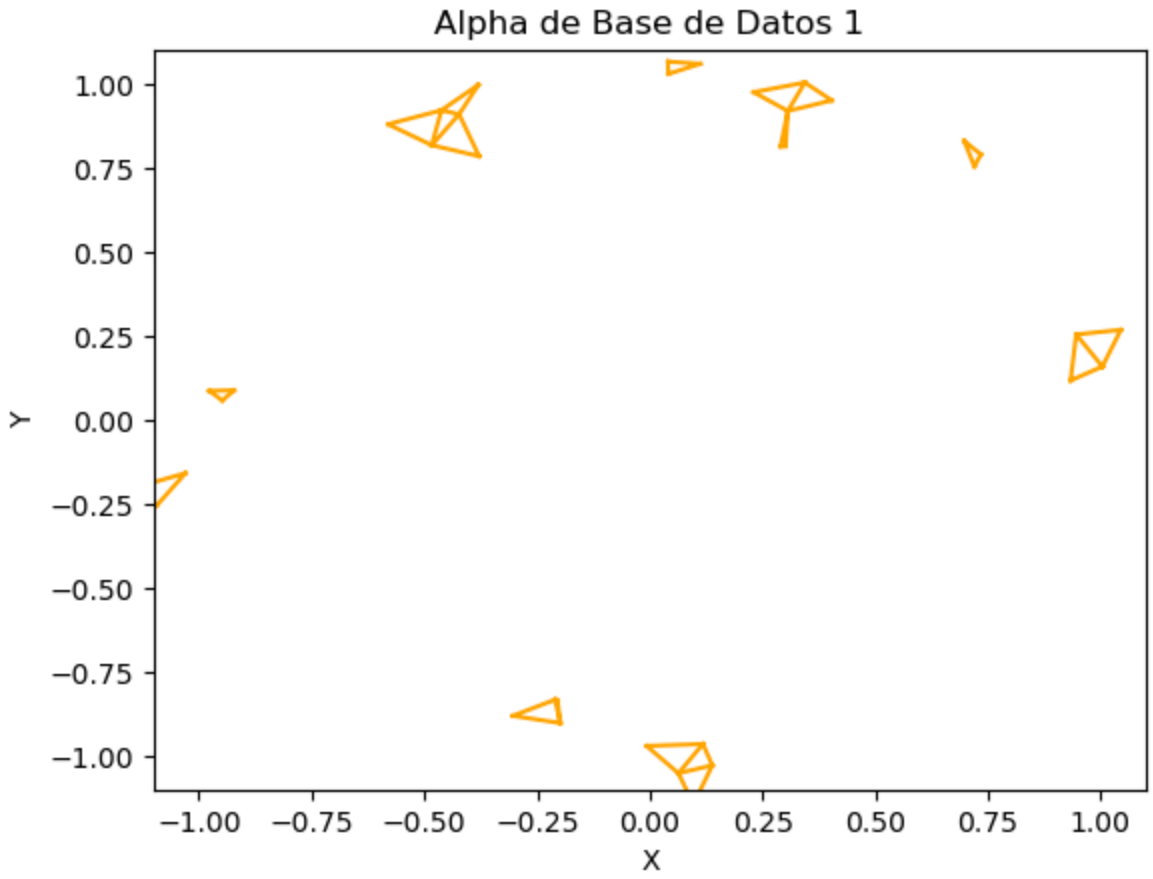
Con 400 simplejos todavía se forma una figura de circulo; sin embargo, si disminuimos aún más alpha, y por ende el número de simplejos, obtenemos una gráfica que no asemeja ninguna figura geométrica.

In [ ]:
```
alpha_complex = gudhi.AlphaComplex(df1.values)
simplex_tree = alpha_complex.create_simplex_tree(max_alpha_square = 0.005)
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 - 212 simplices - 100 vertices.
```

In [ ]:
```
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

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("Alpha de Base de Datos 1")

plt.show()
```



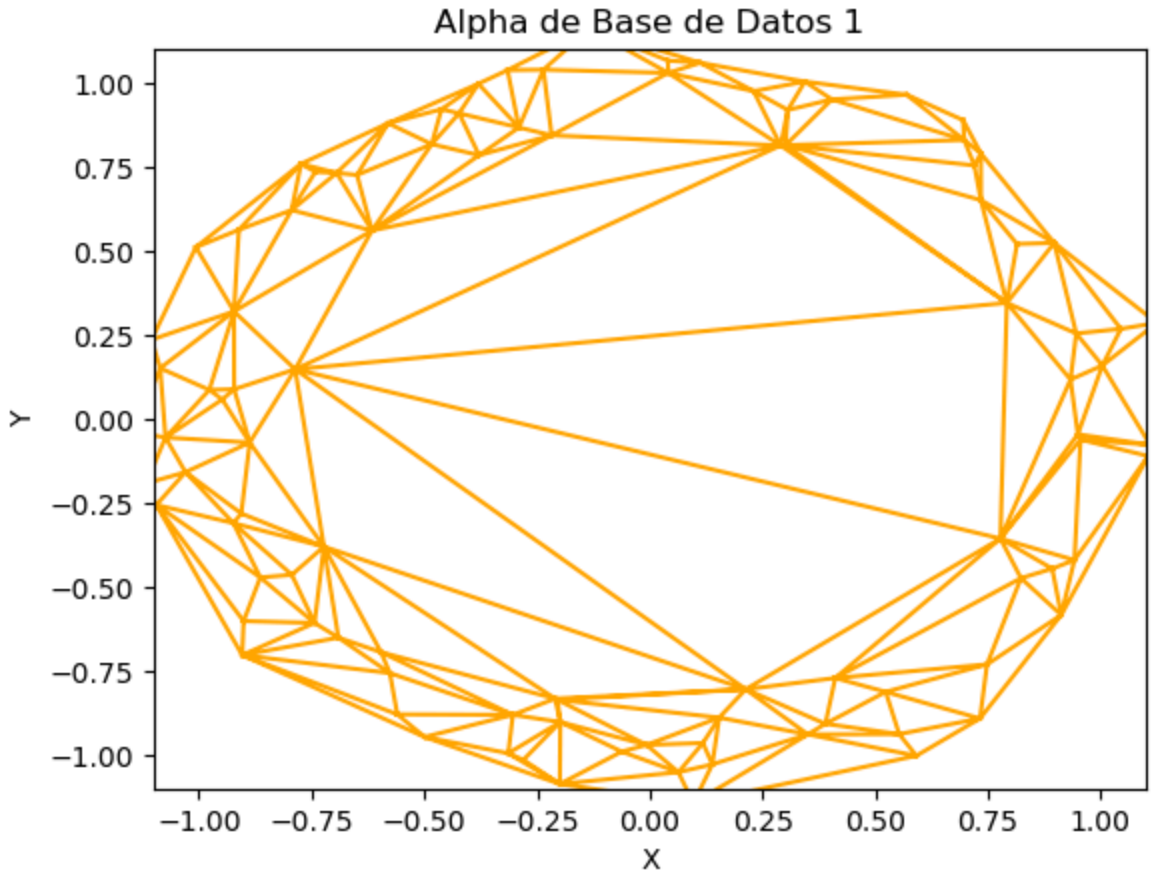
Aunque ya vimos que un alpha muy pequeño no es ideal, un alpha muy grande tampoco lo es. Si tomamos un valor de $\alpha = 1$, se puede observar que el complejo simplicial de Alpha es un círculo, pero ya no es perfecto. En la figura de abajo se puede observar que hay aristas que traspasan el centro del círculo, deformándolo.

In [ ]:
```
alpha_complex = gudhi.AlphaComplex(df1.values)
simplex_tree = alpha_complex.create_simplex_tree(max_alpha_square = 1)
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 - 543 simplices - 100 vertices.
```

In [ ]:
```
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

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("Alpha de Base de Datos 1")
```

```
plt.show()
```



Base de Datos 2

In [ ]:
```
df2 = pd.read_csv('Activity2.csv') # Cargamos la base de datos

# Mostramos la base de datos
display(df2.head())
display(df2.describe())
print("Tamaño del dataframe:", df2.shape)
```

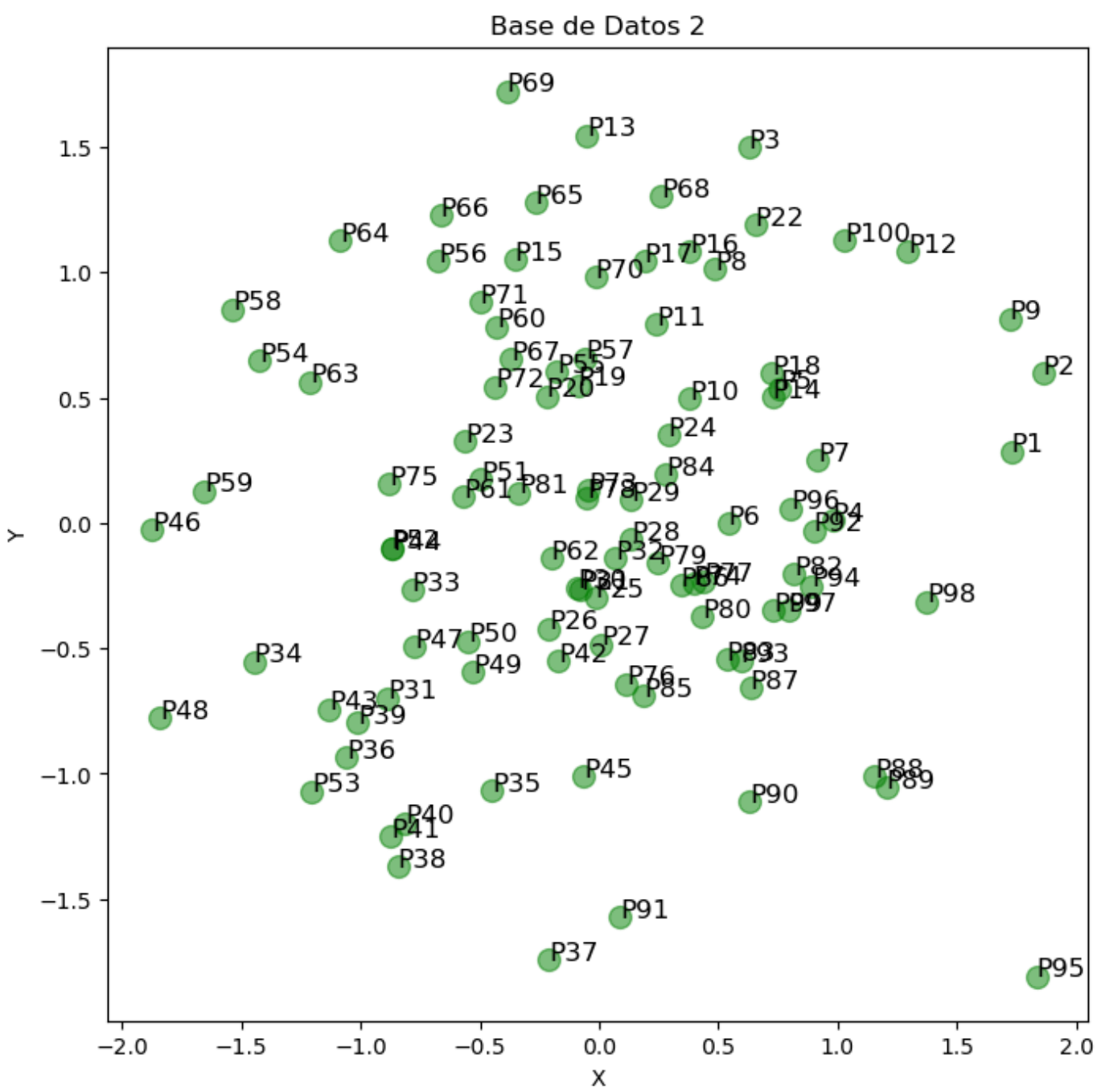
	0	1
0	1.727350	0.285771
1	1.861161	0.597764
2	0.627520	1.497373
3	0.979559	0.008873
4	0.756204	0.536461

	0	1
count	100.000000	100.000000
mean	-0.018978	0.030507
std	0.816764	0.782374
min	-1.874888	-1.808748
25%	-0.553641	-0.506078
50%	-0.048096	-0.014768
75%	0.555272	0.599522
max	1.861161	1.721905

```
Tamaño del dataframe: (100, 2)
```

In [ ]:

```python
# Creamos una columna nueva con el número del punto (P1)
num = np.arange(1,len(df2)+1,1)
num = ['P'+str(i) for i in num]
df2["Point"] = num

# Convertimos la columna "Point" en el índice de la tabla
df2.set_index('Point', inplace=True)
df2.head()

# Graficamos los puntos con el índice como etiqueta
plt.figure(figsize=(8,8))
plt.scatter(df2['0'], df2['1'], s=100, c='green', alpha=0.5)
for i, txt in enumerate(df2.index):
    plt.annotate(txt, (df2['0'][i], df2['1'][i]), fontsize=12)
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Base de Datos 2')
plt.show()
```



A primera vista después de graficar todos los puntos de la base de datos 2, se puede observar que los puntos están distribuidos practicamente aleatoriamente y no se forma una figura geométrica muy clara. Se podría hacer la suposición que forman un círculo, pero no es tan claro como en la base de datos 1. Es por ello que en este caso es incluso más importante realizar los diagramas de persistencia para poder observar la estructura de los datos.

Complejo de Rips

Nuestra base de datos está en el plano R^2, por lo que la dimensión máxima del complejo simplicial de Rips debe ser 2. El parámetro r es la distancia máxima entre dos puntos para que se conecten por una arista.

In [ ]:
```
dist = pd.DataFrame(squareform(pdist(df2), "euclidian"), columns=df2.index.values, index

max_dist = dist.values.max()
print("La distancia máxima entre los puntos es:", round(max_dist, 5))
```

```
La distancia máxima entre los puntos es: 4.29339
```

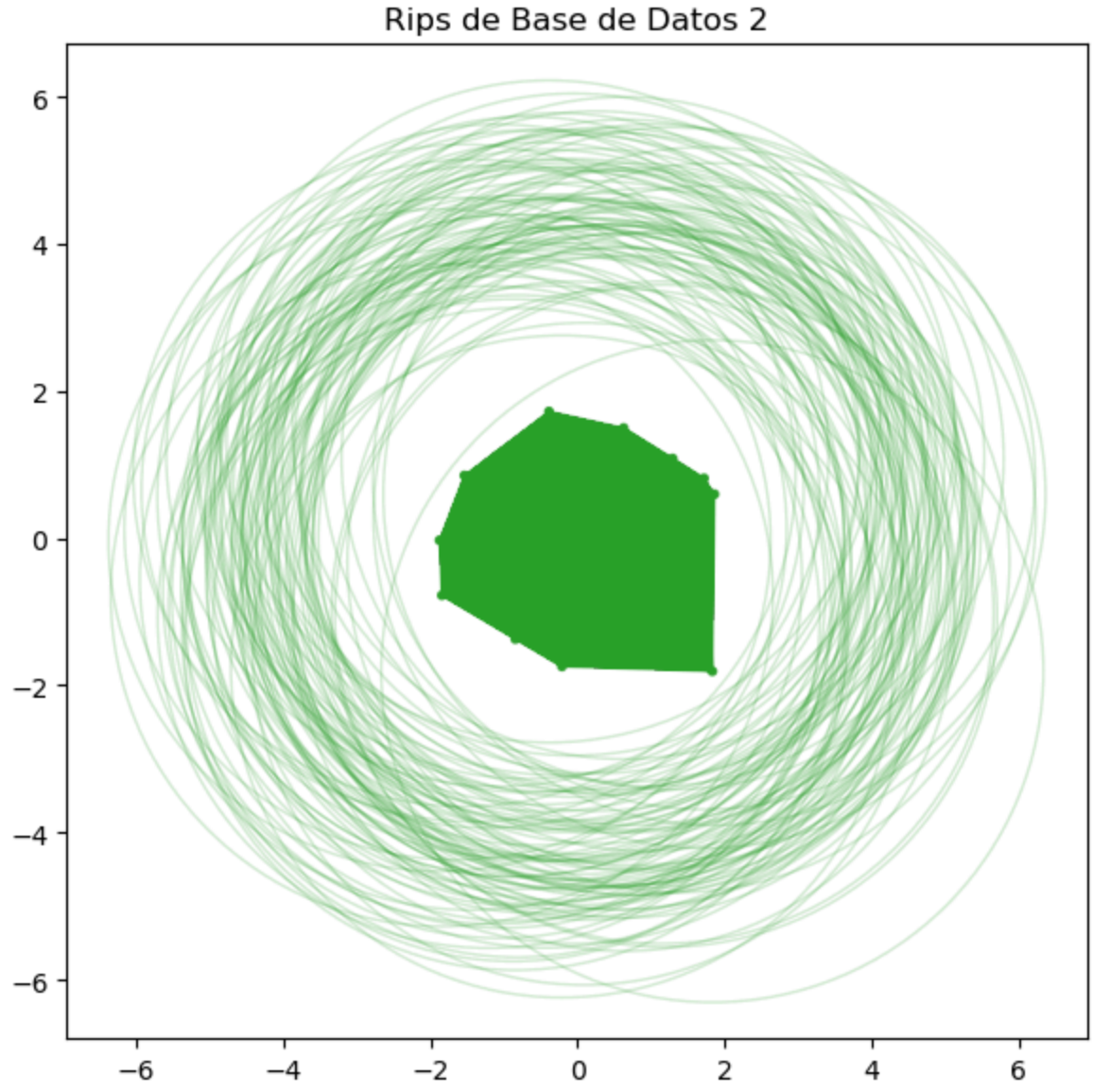
Dada la distancia máxima, podemos elegir $r = 4.5$

In [ ]:
```
rips_complex = gudhi.RipsComplex(distance_matrix=dist.values, max_edge_length=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 - 166750 simplices - 100 vertices.
```

In [ ]:
```
plot_rips_complex(df2.values, R=4.5, label="Rips de Base de Datos 2", col=2, maxdim=2)
```

```
/var/folders/dd/fhmd_dws0_d412cq_s690s3c0000gn/T/ipykernel_61183/3813265713.py:3: Matplo
tlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib 3.7 and will
be removed two minor releases later. Use ``matplotlib.colormaps[name]`` or ``matplotlib.
colormaps.get_cmap(obj)`` instead.
  tab10 = cm.get_cmap('tab10')
```



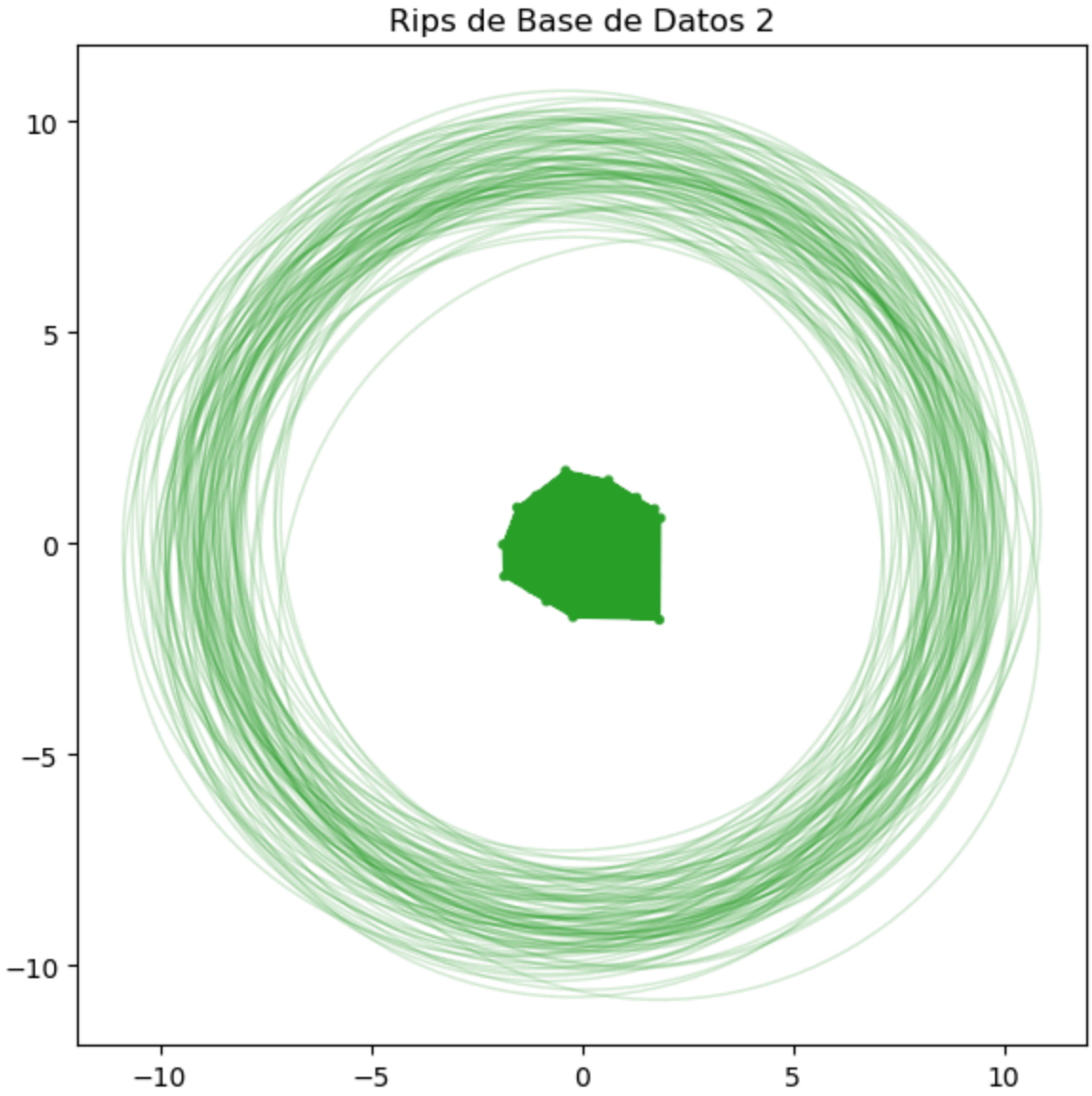
In [ ]:
```
rips_complex = gudhi.RipsComplex(distance_matrix=dist.values, max_edge_length=9)

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 [ ]:
```
plot_rips_complex(df2.values, R=9, label="Rips de Base de Datos 2", col=2, maxdim=2)
```

```
/var/folders/dd/fhmd_dws0_d412cq_s690s3c0000gn/T/ipykernel_61183/3813265713.py:3: Matplo
tlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib 3.7 and will
be removed two minor releases later. Use ``matplotlib.colormaps[name]`` or ``matplotlib.
colormaps.get_cmap(obj)`` instead.
  tab10 = cm.get_cmap('tab10')
```



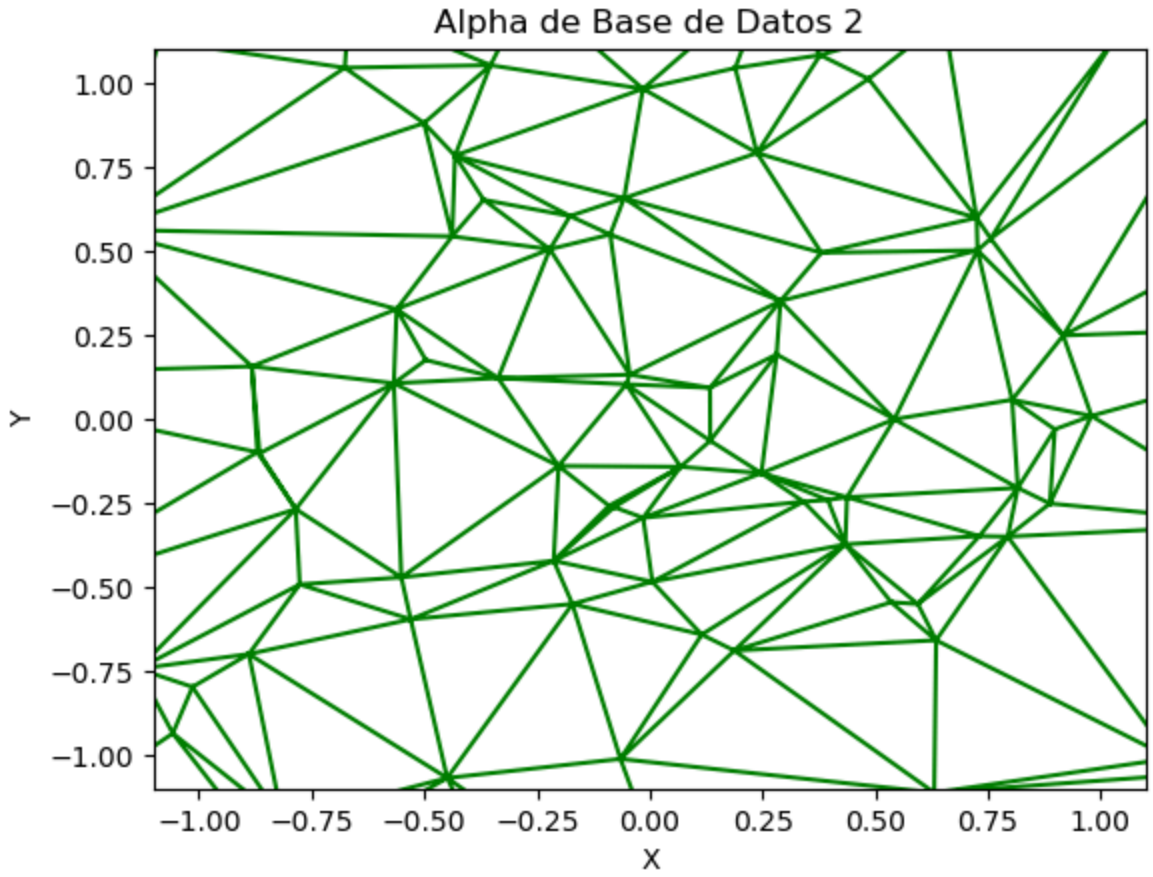
Como se puede observar en la gráfica del complejo simplicial de Rip, el complejo simplicial de Rips es un círculo, así como se había predicho. Esto es interesante, ya que a pesar de que los puntos no forman un círculo, el complejo simplicial de Rips sí lo forma. Sin embargo, a diferencia de la base de datos 1, en este caso el círculo no es tan perfecto, tiene una pequeña "deformación".

Complejo de Alpha

In [ ]:
```
alpha_complex = gudhi.AlphaComplex(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 [ ]:
```
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

fig, ax = plt.subplots()
ax.triplot(points[:, 0], points[:, 1], triangles=triangles, color='green')
ax.set_xlim(-1.1, 1.1)
ax.set_ylim(-1.1, 1.1)
plt.xlabel('X')
plt.ylabel('Y')
plt.title("Alpha de Base de Datos 2")

plt.show()
```



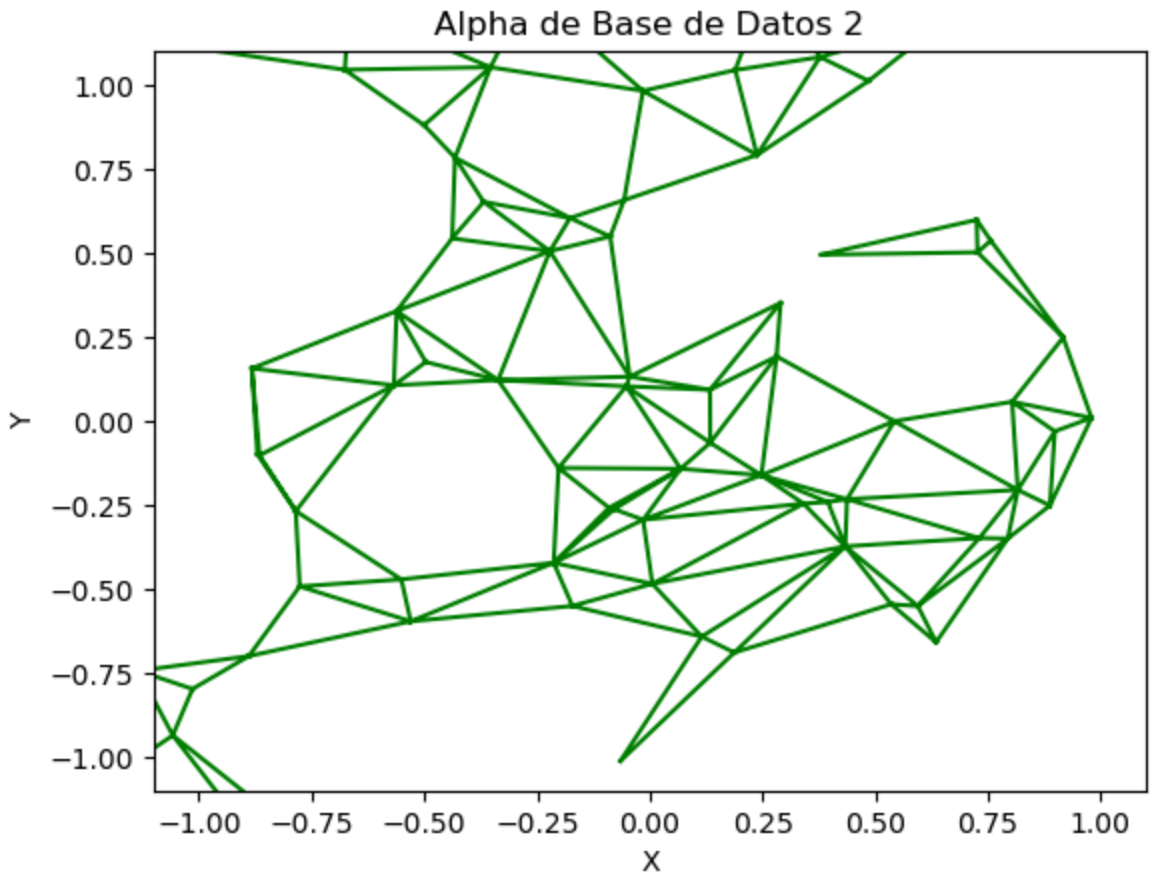
Dado que los datos originales estaban distribuidos prácticamente aleatoriamente, se esperaría que el complejo simplicial de Alpha no forme una figura perfecta. Evidentemente, esto se cumple. Incluso disminuyendo el valor de α, el complejo simplicial de Alpha no forma una figura geométrica clara.

In [ ]:
```
alpha_complex = gudhi.AlphaComplex(df2.values)
simplex_tree = alpha_complex.create_simplex_tree(max_alpha_square = 0.05)
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 - 389 simplices - 100 vertices.
```

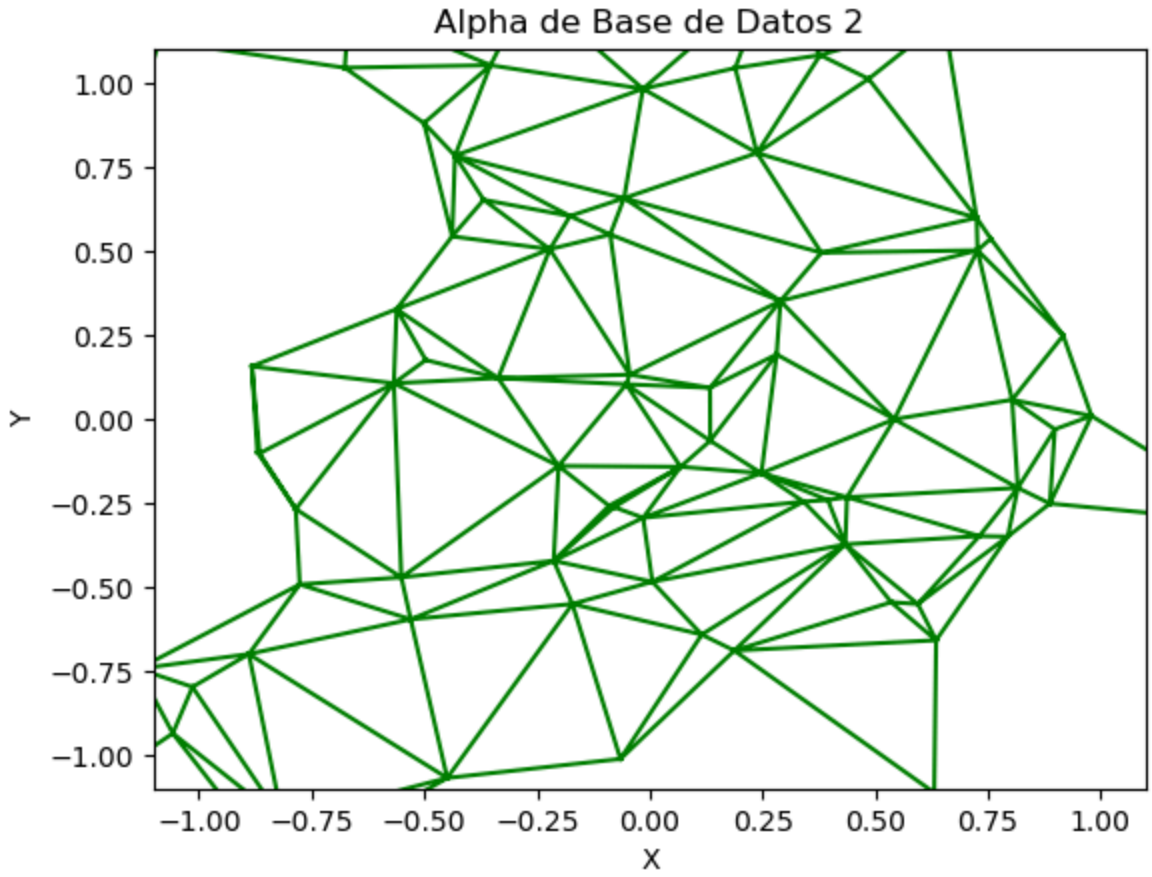
In [ ]:
```
points = np.array([alpha_complex.get_point(i) for i in range(simplex_tree.num_vertices()
triangles = np.array([s[0] for s in simplex_tree.get_skeleton(2) if len(s[0]) == 3 and s

fig, ax = plt.subplots()
ax.triplot(points[:, 0], points[:, 1], triangles=triangles, color='green')
ax.set_xlim(-1.1, 1.1)
ax.set_ylim(-1.1, 1.1)
plt.xlabel('X')
plt.ylabel('Y')
plt.title("Alpha de Base de Datos 2")

plt.show()
```



In [ ]:
```
alpha_complex = gudhi.AlphaComplex(df2.values)
simplex_tree = alpha_complex.create_simplex_tree(max_alpha_square = 0.1)
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 - 477 simplices - 100 vertices.
```

In [ ]:
```
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

fig, ax = plt.subplots()
ax.triplot(points[:, 0], points[:, 1], triangles=triangles, color='green')
ax.set_xlim(-1.1, 1.1)
ax.set_ylim(-1.1, 1.1)
plt.xlabel('X')
plt.ylabel('Y')
plt.title("Alpha de Base de Datos 2")

plt.show()
```



Base de Datos 3

In [ ]:
```
df3 = pd.read_csv('Activity3.csv') # Cargamos la base de datos

# Mostramos la base de datos
display(df3.head())
display(df3.describe())
print("Tamaño del dataframe:", df3.shape)
```

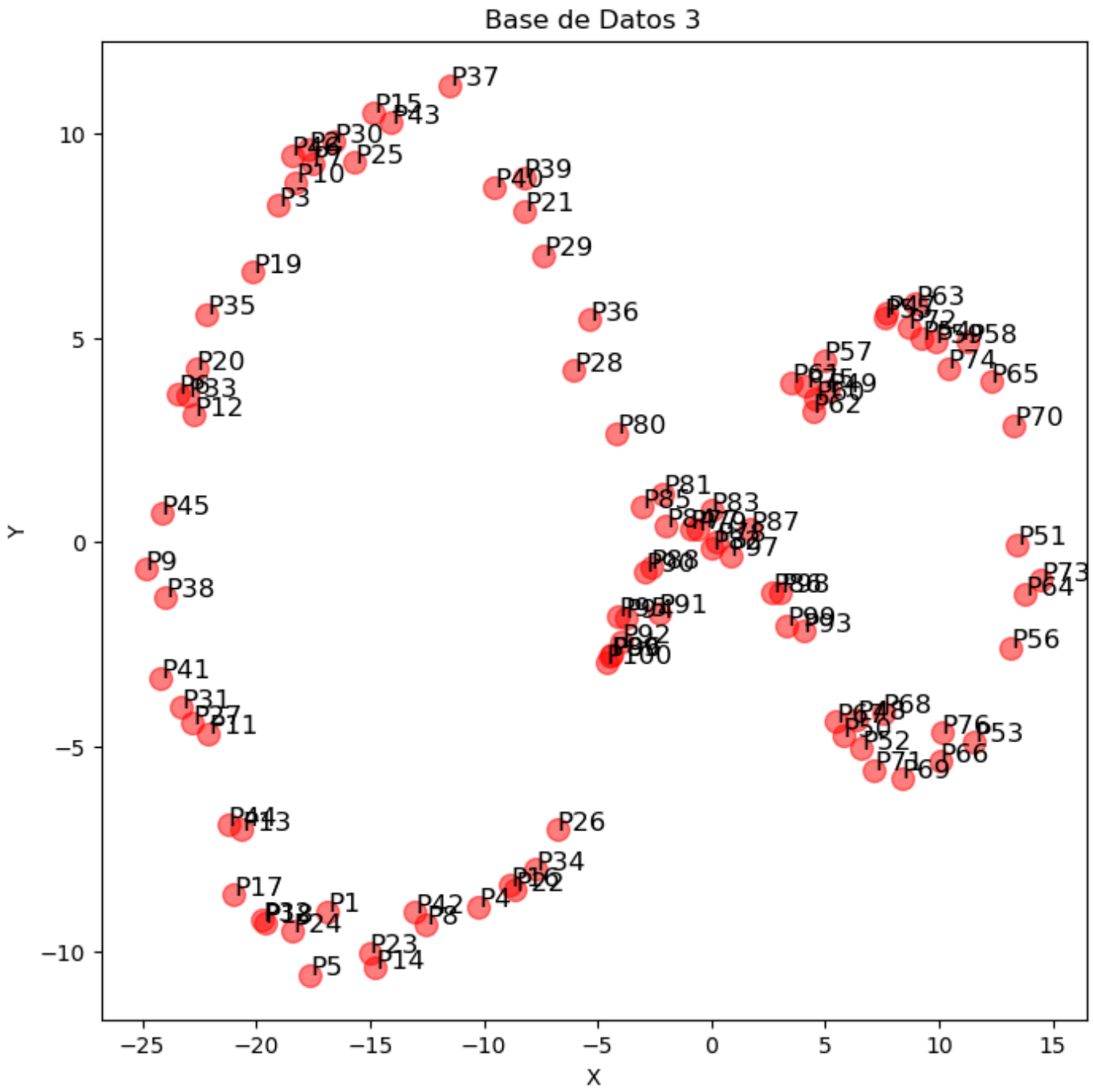
	0	1
0	-16.865905	-9.030913
1	-17.721247	9.601550
2	-19.047260	8.233957
3	-10.252216	-8.925738
4	-17.636004	-10.591064

	0	1
count	100.000000	100.000000
mean	-5.308112	-0.035194
std	11.931284	5.888771
min	-24.843552	-10.591064
25%	-17.030821	-4.482070
50%	-4.112474	-0.247647
75%	4.982264	4.304558
max	14.496662	11.175014

```
Tamaño del dataframe: (100, 2)
```

In [ ]:

```python
# Creamos una columna nueva con el número del punto (P1)
num = np.arange(1,len(df3)+1,1)
num = ['P'+str(i) for i in num]
df3["Point"] = num

# Convertimos la columna "Point" en el índice de la tabla
df3.set_index('Point', inplace=True)
df3.head()

# Graficamos los puntos con el índice como etiqueta
plt.figure(figsize=(8,8))
plt.scatter(df3['0'], df3['1'], s=100, c='red', alpha=0.5)
for i, txt in enumerate(df1.index):
    plt.annotate(txt, (df3['0'][i], df3['1'][i]), fontsize=12)
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Base de Datos 3')
plt.show()
```



A primera vista después de graficar todos los puntos de la base de datos 3, se puede observar que los puntos están distribuidos como en forma de un infinito. Esto podría interpretarse como dos círculos y se esperaría que los diagramas de persistencia de Rips, Cech y Alpha muestren una estructura similar a la de dos círculos.

Complejo de Rips

In [ ]:
```
dist = pd.DataFrame(squareform(pdist(df3), "euclidian"), columns=df3.index.values, index

max_dist = dist.values.max()
print("La distancia máxima entre los puntos es:", round(max_dist, 5))
```

```
La distancia máxima entre los puntos es: 39.3412
```

Dada la distancia máxima, podemos elegir $r = 40$

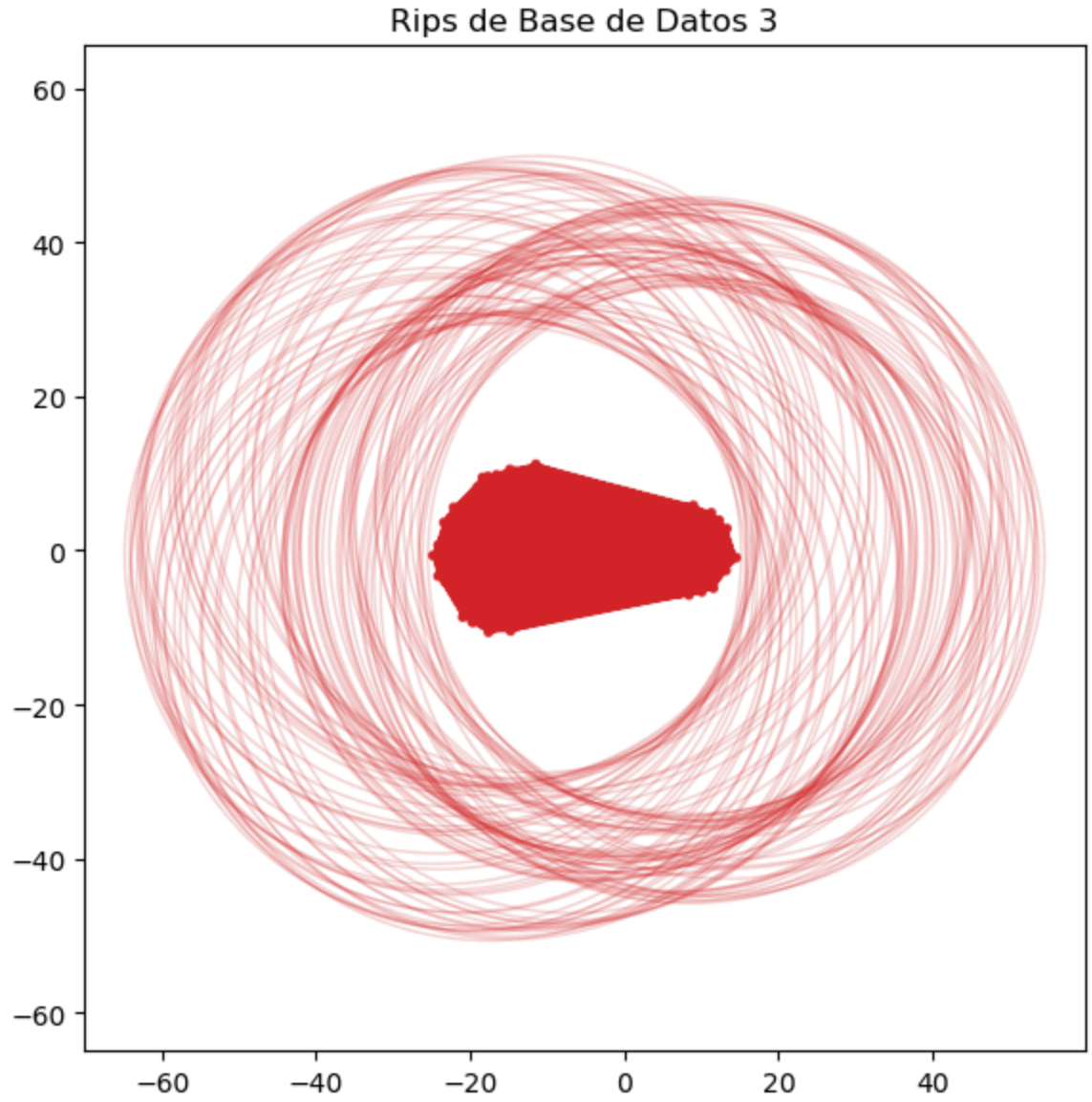
In [ ]:
```
rips_complex = gudhi.RipsComplex(distance_matrix=dist.values, max_edge_length=40)
```

```
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 [ ]:
```
plot_rips_complex(df3.values, R=40, label="Rips de Base de Datos 3", col=3, maxdim=2)
```

```
/var/folders/dd/fhmd_dws0_d412cq_s690s3c0000gn/T/ipykernel_61183/3813265713.py:3: Matplo
tlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib 3.7 and will
be removed two minor releases later. Use ``matplotlib.colormaps[name]`` or ``matplotlib.
colormaps.get_cmap(obj)`` instead.
  tab10 = cm.get_cmap('tab10')
```



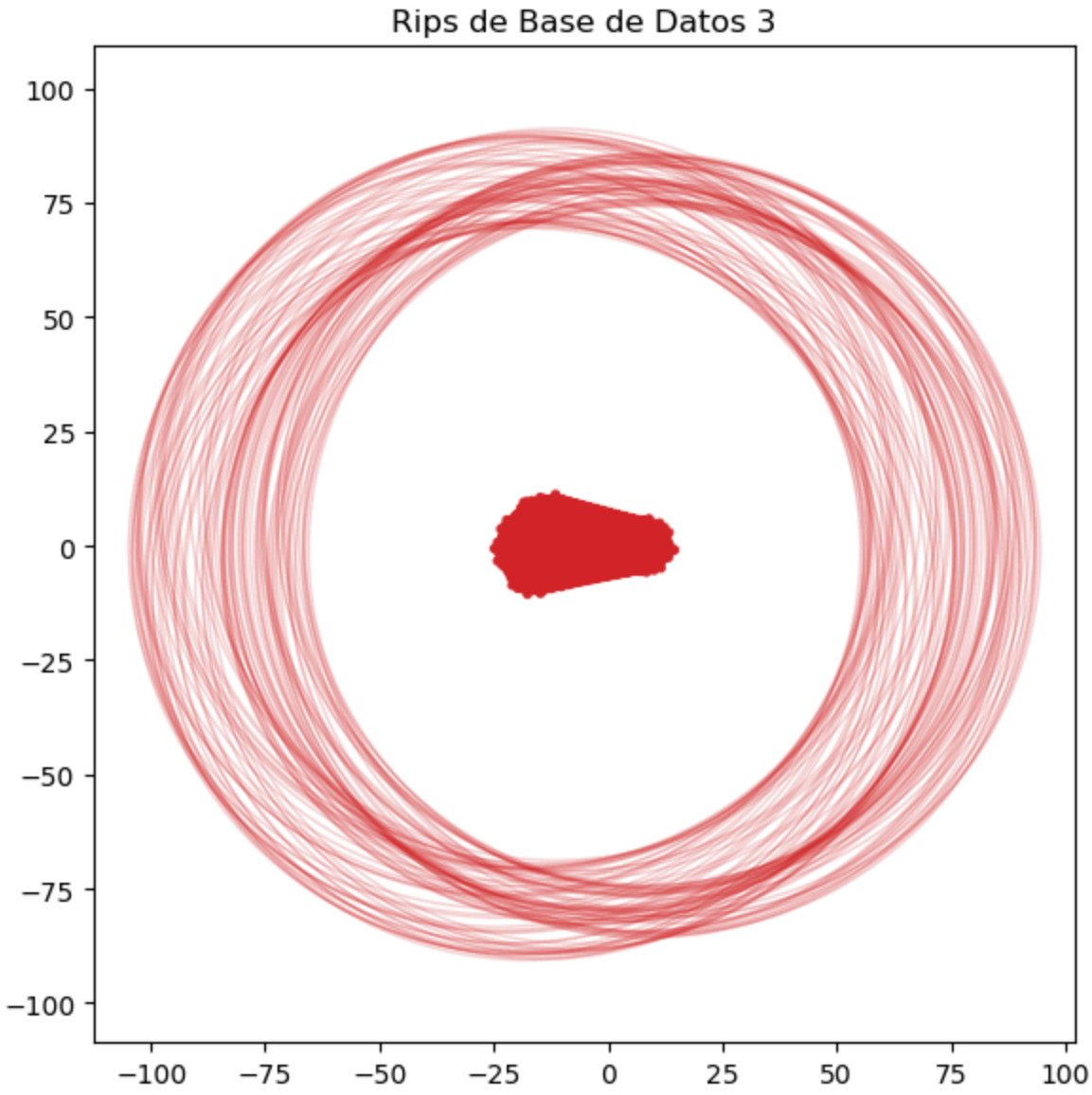
In [ ]:
```
rips_complex = gudhi.RipsComplex(distance_matrix=dist.values, max_edge_length=80)

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 [ ]:
```
plot_rips_complex(df3.values, R=80, label="Rips de Base de Datos 3", col=3, maxdim=2)
```

```
/var/folders/dd/fhmd_dws0_d412cq_s690s3c0000gn/T/ipykernel_61183/3813265713.py:3: Matplo
tlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib 3.7 and will
be removed two minor releases later. Use ``matplotlib.colormaps[name]`` or ``matplotlib.
colormaps.get_cmap(obj)`` instead.
  tab10 = cm.get_cmap('tab10')
```



Como se puede observar, el complejo simplicial de Rips son dos círculos que están un poco sobrepuestos, así como un infinito. Esto tiene sentido, ya que los puntos originales están distribuidos como un infinito.

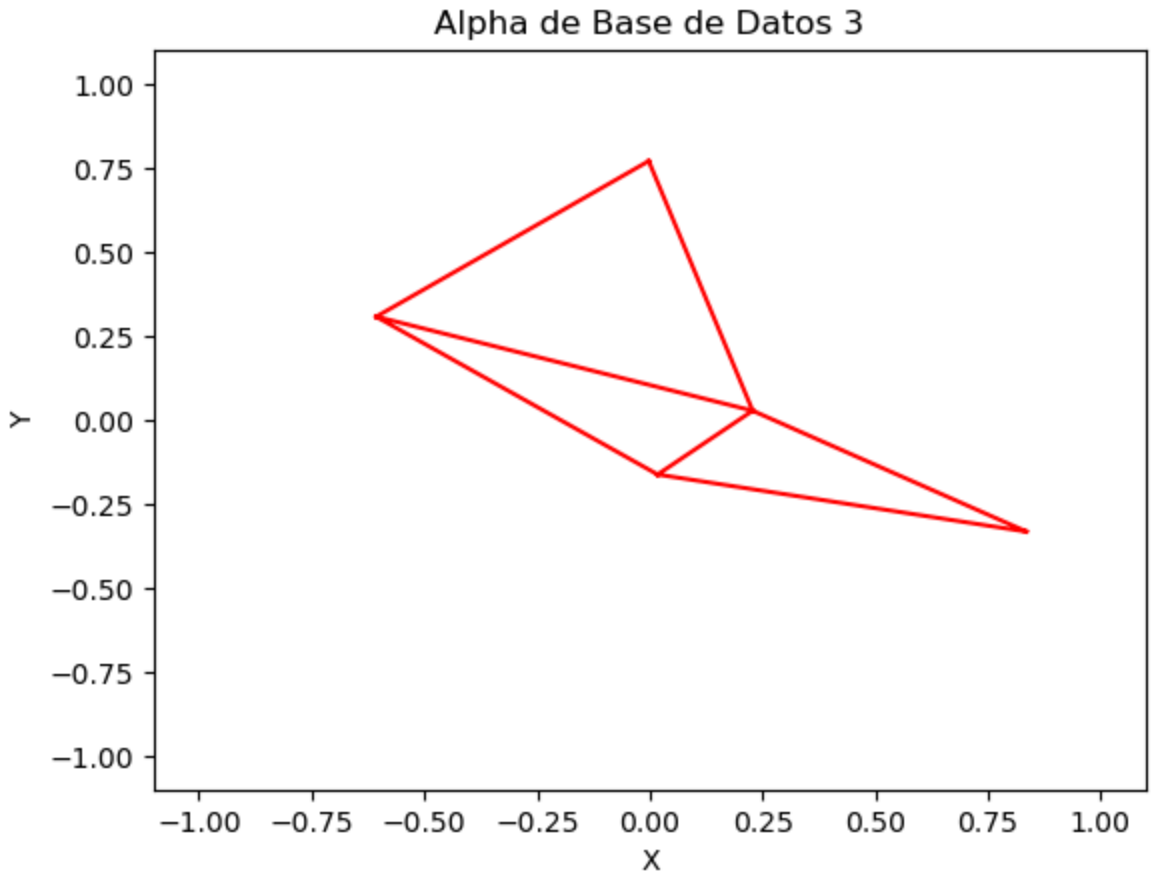
Complejo de Alpha

In [ ]:
```
alpha_complex = gudhi.AlphaComplex(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 [ ]:
```
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

fig, ax = plt.subplots()
ax.triplot(points[:, 0], points[:, 1], triangles=triangles, color='red')
ax.set_xlim(-1.1, 1.1)
ax.set_ylim(-1.1, 1.1)
plt.xlabel('X')
plt.ylabel('Y')
plt.title("Alpha de Base de Datos 3")

plt.show()
```

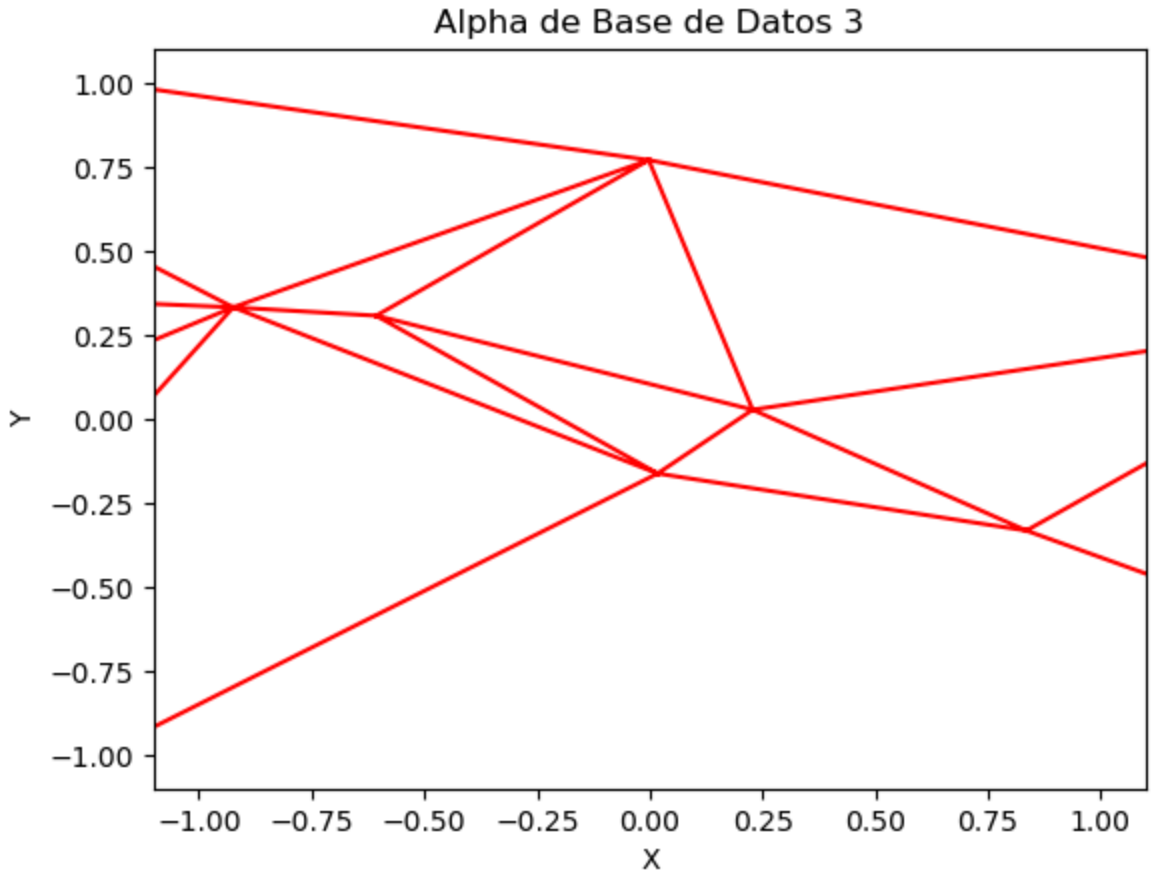


En este caso, el valor de alpha base que habíamos usado en las bases de datos anteriores ($alpha = 0.5$) no es suficiente para poder visualizar la estructura de los datos. Por lo tanto, se decidió aumentar el valor de alpha.

In [ ]:
```
alpha_complex = gudhi.AlphaComplex(df3.values)
simplex_tree = alpha_complex.create_simplex_tree(max_alpha_square = 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 - 392 simplices - 100 vertices.
```

In [ ]:
```
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

fig, ax = plt.subplots()
ax.triplot(points[:, 0], points[:, 1], triangles=triangles, color='red')
ax.set_xlim(-1.1, 1.1)
ax.set_ylim(-1.1, 1.1)
plt.xlabel('X')
plt.ylabel('Y')
plt.title("Alpha de Base de Datos 3")

plt.show()
```



Evidentemente, el complejo simplicial de Alpha no forma una figura geométrica clara sin importar el valor de alpha, indicando que en este caso, es mejor usar el complejo simplicial de Rips para visualizar la estructura de los datos.