Universidad Nacional Autónoma de México

Facultad de Ciencias

Introducción a la homología persistente mediante las gráficas Mapper

T E S I S

que para obtener el título de:

Matemática

P R E S E N T A :

Ariadna Olvera Sampieri

TUTOR

Dr. Vinicio Antonio Gómez Gutiérrez

Ciudad Universitaria, CDMX, 2022

1. Datos del alumno.
Olvera
Sampieri
Ariadna
Universidad Nacional Autónoma de México
Facultad de Ciencias
Matemáticas
315298905

2. Datos del asesor.
Dr.
Vinicio Antonio
Gómez
Gutiérrez

3. Datos del sinodal 1
Dr.
José Ángel
Frías
García

4. Datos del sinodal 2
Dr.
Jesús
Rodríguez
Viorato

5. Datos del sinodal 3
Dra.
Araceli
Guzmán
Tristán

6. Datos del sinodal 4
Dr.
Luis Jorge
Sánchez
Saldaña

7. Datos de la tesis
Introducción a la homología persistente mediante las gráficas Mapper
70 p.
2022

«Decía que me gustaban las matemáticas porque consisten en resolver problemas, y esos problemas son difíciles e interesantes, pero siempre hay una respuesta sencilla al final. Y lo que quería decir era que las matemáticas no son como la vida, porque al final en la vida no hay respuestas sencillas.»

Mark Haddon, en *"El curioso incidente del perro a medianoche"*.

Agradecimientos

Quiero agradecer al Dr. Vinicio Antonio Gómez Gutiérrez por la dirección de esta tesis y por todo el apoyo brindado.

A los sinodales, Dr. José Ángel Frías García, Dr. Jesús Rodríguez Viorato, Dra. Araceli Guzmán Tristán y al Dr. Luis Jorge Sánchez Saldaña por su valiosa revisión, así como por todos los comentarios y observaciones realizadas para la mejora de este trabajo.

A la Facultad de Ciencias y a la Universidad Nacional Autónoma de México por permitirme cursar estudios desde educación media superior hasta posgrado, por la oportunidad laboral, y por todas las actividades en las que me ha permitido participar.

Por último, quiero agradecer a Roberto Manríquez y a Luis Ochoa por todo el apoyo brindado en la realización de este trabajo.

Introducción

En los últimos años los notables avances tecnológicos han permitido el desarrollo de diversos métodos computacionales, programas y softwares, además de varias IAs y redes neuronales, las cuales fungen como herramientas que coadyuvan en el estudio y análisis de amplios conjuntos de datos.

El análisis topológico de datos es un campo que busca aplicar métodos rigurosos de topología, como gráficas de Reeb y gráficas Mapper, así como homología persistente, con el objetivo de analizar conjuntos grandes de datos complicados, con ruido y de alta dimensión.

Los métodos topológicos frecuentemente requieren un complejo simplicial como entrada o *input*. Sin embargo, los conjuntos u objetos no siempre se encuentran expresados de esta forma. Por lo que, un paso para realizar un análisis topológico del conjunto es poder expresarlo como complejo simplicial.

Basándonos en Singh, Mémoli, Carlsson [2] y Dlotko [1], en este trabajo abordaremos distintas construcciones de complejos simpliciales a partir de una nube de puntos, es decir, un conjunto finito de puntos en un espacio métrico. Este conjunto de puntos puede representar o muestrear algún objeto, superficie, variedad, un conjunto de datos numéricos, entre otros, del que queramos obtener información matemática para su análisis.

En el primer capítulo abordamos el tema de algoritmos de clustering o agrupamiento, el cual busca dividir y clasificar elementos de un conjunto de datos, tomando en cuenta la cercanía o proximidad entre ellos. Estos algoritmos tendrán relevancia en el siguiente capítulo.

En el segundo capítulo se aborda el método de Mapper, el cual hace uso de herramientas topológicas para el estudio de grandes conjuntos de datos. Este método se basa en clusters parciales de datos, como se trabajó en el primer capítulo, e introduce un conjunto de funciones definidas sobre el conjunto de datos a estudiarse, mediante las cuales se extrae información útil del conjunto para ser traducida a complejos simpliciales con el fin de poder ser analizada. Además de buscar simplificar dicha información para hacer posible su estudio, y de una forma más asequible, ya que en muchos casos, se dificulta el análisis de los datos o información del conjunto debido

al denso volumen de los datos. De esta forma, el método de Mapper puede reducir conjuntos de datos de altas dimensiones en complejos simpliciales más simples que el conjunto de datos, y que contengan información geométrica y topológica relevante para su análisis.

En el tercer capítulo abordamos el método Ball mapper, el cual se basa en la idea del método de Mapper, incluyendo los parámetros del método original para construir una cubierta con traslapes de la nube de puntos. En este caso la cubierta de la nube de puntos consistirá de bolas de radio ε, mediante la construcción de una ε-red.

Parte importante de este trabajo fue la implementación en Python de los algoritmos computacionales para obtener la gráfica Ball mapper de una nube de puntos X en el plano complejo. Los códigos en Python de esta implementación se encuentran en la sección de apéndices. Asimismo se pusieron en práctica estos programas para cuatro nubes de puntos obtenidas a partir de los ceros del polinomio de Jones de nudos tóricos, y para diversos valores de ε. Los resultados de este experimento se muestran en el apéndice C.

Finalmente, en el cuarto capítulo introducimos el tema de homología persistente y su relación con la gráfica Ball mapper, tomando como base lo construido en el capítulo anterior.

Índice general

1 Algoritmos de clustering

En este capítulo abordamos el tema de algoritmos de clustering, primero desde una perspectiva general, describiendo algunos tipos de éstos, y basándonos en las definiciones de [3] y [7] para posteriormente hablar de un algoritmo de clustering basado en la densidad de una nube de puntos, DBSCAN, obtenido en [4]. Finalmente, concluimos el capítulo describiendo el clustering de enlace simple, basándonos en [3] y [8], el cual será usado para el método de Mapper.

1.1. Clustering o agrupamiento de datos

Un algoritmo de clustering sirve para clasificar un conjunto finito. Informalmente, un algoritmo de clustering o agrupamiento de datos se refiere al proceso de particionar un conjunto de datos en un número de partes o clusters, que son distinguibles entre sí.

En el contexto de espacios métricos finitos, a grandes rasgos podemos decir que los puntos en un mismo cluster están más cerca entre sí que puntos en otros clusters. Un algoritmo de clustering puede pensarse como la contraparte estadística a la construcción geométrica de las componentes arcoconexas de un espacio, es decir, podemos pensar a cada cluster como una discretización sobre un espacio métrico del concepto de componente arcoconexa, el cual es un bloque constructivo fundamental en el que la topología algebraica se basa.

Existen varias formas de construir clusters basados en información métrica, como cluster de enlace simple, promedio o completo, cluster por k-medias, clustering espectral, entre otros.

El clustering, o agrupamiento, jerárquico es un método de análisis de datos o nubes de puntos en el cual se busca construir una jerarquía de grupos de puntos. El clustering jerárquico se divide principalmente en dos tipos: clustering aglomerativo, en donde se inicia con puntos por separado y en cada paso se van uniendo los clusters cercanos entre sí; y clustering divisivo, en el que se inicia con todos los puntos en un mismo cluster y se van dividiendo en cada paso. Esta clasificación jerárquica puede representarse de forma diagramática, y a esta representación le llamamos dendograma.

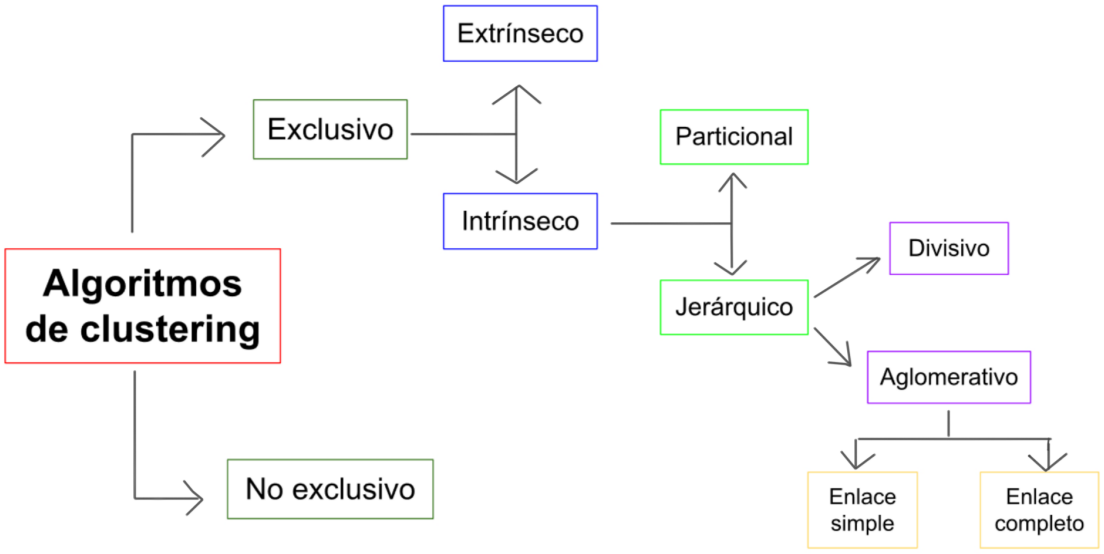


Figura 1.1: Clasificación de algoritmos de clustering

En los métodos de clustering aglomerativo, inicialmente cada elemento de la nube de puntos es un cluster por sí mismo. En cada paso, los clusters con menor distancia entre sí son combinados para obtener nuevos clusters más grandes, y así sucesivamente hasta que todos los puntos conformen un único cluster. La función que se usa para determinar la distancia entre dos clusters se le llama función de enlace y es lo que diferencia a los distintos métodos de clustering aglomerativo.

Como comentamos en el párrafo anterior, en los métodos de clustering aglomerativo, en cada paso se unen los clusters con menor distancia entre sí. El cómo definimos la " menor distancia" entre los clusters es lo que diferenciará entre un algoritmo y otro. Por ejemplo, en el clustering de enlace simple, la función distancia entre los clusters X y Y estará dada por $d(X,Y) = \min_{x\in X, y\in Y} d(x,y)$, mientras que en el clustering de enlace completo, la función distancia estará dada por $d(X,Y) = \max_{x\in X, y\in Y} d(x,y)$.

El clustering de enlace completo presenta algunas ventajas sobre el clustering de enlace simple, como el fenómeno de encadenamiento, es decir, el clustering de enlace completo tiende a formar grupos de diámetros aproximadamente iguales, mientras que el clustering de enlace simple puede generar clusters delgados y largos, donde elementos cercanos del cluster pueden tener distancias pequeñas entre sí, pero elementos en extremos opuestos dentro de un mismo cluster pueden tener distancias grandes entre sí.

1.2. DBSCAN, un algoritmo de clustering

A continuación presentamos un algoritmo de clustering o agrupamiento de datos, el algoritmo DBSCAN, *Density Based Spatial Clustering of Applications with Noise*, basado en la densidad de una nube de puntos D en un espacio euclidiano k-dimensional.

Definición 1.2.1 *Definimos la ε-vecindad (eps-vecindad) $N_\varepsilon(p)$ de un punto p como el conjunto: $N_\varepsilon(p) = \{q \in D | d(p,q) \leq \varepsilon\}$.*

Este algoritmo reconoce dos tipos de puntos en un cluster: puntos núcleo o *core points* que se encuentran al interior del cluster y puntos frontera o *border points*, que se encuentran en el borde del cluster. Intuitivamente podemos ver que una ε-vecindad de un punto frontera tendrá menos puntos que una ε-vecindad de un punto núcleo, por lo cual, es necesario establecer que para cada punto en un cluster, haya al menos una cantidad mínima de puntos ($MinPts$) en una ε-vecindad de él.

Definición 1.2.2 *Un punto p es directamente (densamente-) alcanzable desde un punto q, (con respecto a los parámetros ε y MinPts), si cumple que:*
1) $p \in N_\varepsilon(q)$, y
2) $|N_\varepsilon(q)| \geq MinPts$.

Un punto p es entonces un punto núcleo si hay una cantidad mínima de puntos ($MinPts$) en una ε-vecindad de p, $N_\varepsilon(p)$, y además esos puntos son directamente alcanzables desde p. Un punto q será punto frontera si es directamente alcanzable desde un punto p, pero no cumple la condición de $MinPts$.

Observamos que la propiedad de ser directamente alcanzable desde un punto es simétrica para dos puntos núcleo, pero en general, no es simétrica para un punto núcleo y un punto frontera.

Definición 1.2.3 *Un punto q es (densamente-) alcanzable desde un punto p (con respecto a los parámetros ε y MinPts) si existe una sucesión de puntos $p_1, ..., p_n$, donde $p_1 = p$ y $p_n = q$ tal que cada punto p_{i+1} es directamente alcanzable desde p_i.*

Notemos que todos los puntos en dicha sucesión deben ser puntos núcleo, salvo posiblemente q. Asimismo, cabe mencionar que la relación densamente-alcanzable es transitiva, y no es simétrica en general, pero sí lo es para puntos núcleo. Pues por definición, ningún punto puede ser alcanzable desde otro punto que no sea punto núcleo sin importar la distancia entre ellos.

Definición 1.2.4 *Un punto p está (densamente-) conectado a un punto q (con respecto a los parámetros ε y MinPts) si existe un punto r tal que tanto p como q son alcanzables desde r (con respecto a los parámetros ε y MinPts).*

Observamos que el estar densamente-conectado es una relación simétrica y para puntos densamente-alcanzables, también es reflexiva. Un punto q que no es alcanzable desde ningún otro punto p, será considerado ruido.

Ejemplo 1 *Sean A, B, C, N puntos en una nube de puntos como en la Figura 1.2. La ε-vecindad de cada punto está determinada por el círculo con centro en cada punto y radio ε como se marca en la figura. $MinPts = 4$.*

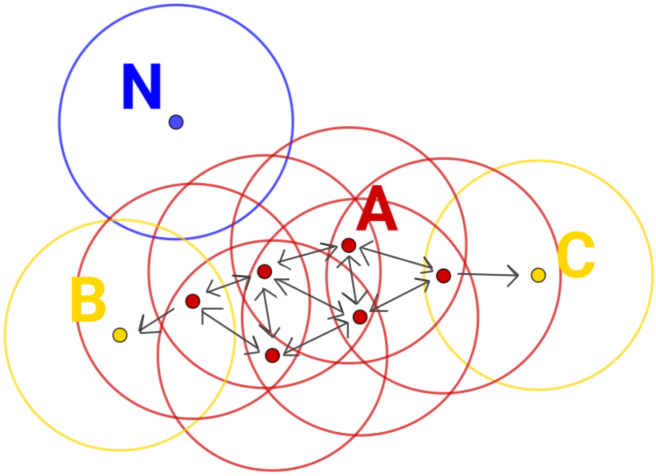


Figura 1.2: Nube de puntos

Observamos que tanto B como C son alcanzables desde A, pues existen sucesiones de puntos p_1, p_2, p_3, p_4, donde $p_1 = A$ y $p_4 = B$, y q_1, q_2, q_3, donde $q_1 = A$ y $q_3 = C$ tal que cada punto p_{i+1} es directamente alcanzable desde p_i y cada punto q_{i+1} es directamente alcanzable desde q_i. Con esto, B y C están densamente-conectados, pues ambos son alcanzables desde el punto A. Observamos también que A es punto núcleo, al igual que el resto de los puntos en color rojo, mientras que B y C son puntos frontera, y N es ruido, pues no es alcanzable desde ningún otro punto. Asimismo notemos que B y C son alcanzables desde A como ya comentamos; sin embargo, A no es alcanzable desde B ni desde C, pues no hay puntos directamente alcanzables desde B o C, ya que no cumplen la condición del $MinPts$: $|N_\varepsilon(B)| = |N_\varepsilon(C)| = 2 < 4 = MinPts$.

De forma que ahora podemos proceder a dar la definición de cluster en este algoritmo. Intuitivamente, un cluster puede pensarse como un conjunto maximal de puntos densamente-conectados con respecto a la relación densamente-alcanzables. Y el ruido, es entonces el conjunto de puntos que no pertenecen a ningún cluster.

Definición 1.2.5 *Dada una nube de puntos D, definimos un cluster C (con respecto a los parámetros ε y MinPts) como un subconjunto no vacío de D que cumple lo siguiente:*
*1) **(Maximalidad)** si un punto q es alcanzable desde cualquier otro punto p del cluster entonces q también pertenece al cluster: $\forall\ p, q$, si $p \in C$ y q es alcanzable desde p (con respecto a ε y MinPts), entonces $q \in C$.*
*2) **(Conectividad)** todos los puntos del cluster están densamente-conectados entre sí: $\forall\ p, q \in C$, p está densamente-conectado con q (con respecto a ε y MinPts).*

Definición 1.2.6 *Si $C_1, ..., C_k$ son los clusters de la nube de puntos D con parámetros ε_i y $MinPts_i$, i=1,...,k, entonces definimos el ruido como el conjunto de puntos en la base de datos D que no pertenecen a ningún cluster C_i, es decir, ruido = $\{p \in D | \forall i = 1, ..., k : p \notin C_i\}$.*

Observemos que un cluster C con respecto a ε y $MinPts$ contiene al menos $MinPts$ puntos ya que C contiene al menos un punto p que debe estar densamente-conectado a sí mismo a partir de un punto r (donde puede ser que $r = p$) y entonces o debe satisfacer la condición (2) de la definición 1.2.2, y por lo tanto, la ε-vecindad de r contiene al menos $MinPts$ puntos.

Observemos que pueden darse casos como los siguientes:
Caso 1. $MinPts = 5$ y que haya puntos que no sean alcanzables desde otro punto y por lo que no pertenezcan a ningún cluster, pero que cumplan que $|N_\varepsilon(p)| = 3$, de forma que dichos puntos serían etiquetados como ruido.
Caso 2. $MinPts = 1$, en el cual todo punto sería un cluster y no habría ruido, debido a la observación anterior, un punto p está conectado a sí mismo desde un punto o (donde $o = p$), y claramente se cumple la condición de $MinPts = 1$.

Algunas propiedades importantes derivadas de esta definición de clusters son:

Lema 1 *Sea p un punto en D y $|N_\varepsilon(p)| \geq MinPts$, entonces el conjunto $O = \{o \mid o \in D$ y o es densamente-alcanzable desde p (con respecto a ε y $MinPts)\}$ es un cluster con respecto a ε y MinPts.*

Demostración. Veamos que el conjunto O cumple las dos condiciones de la definición 1.2.5. Para la maximalidad, sea $q \in D$ tal que q es alcanzable desde $o \in O$, veamos que $q \in O$. Como $o \in O$, entonces o es alcanzable desde p, y como la propiedad densamente-alcanzable es transitiva, entonces q es alcanzable desde p, por lo que q pertenece a O. Para ver la conectividad, sean $o_1, o_2 \in O$, queremos ver que o_1 y o_2 están densamente-conectados entre sí. Como $o_1, o_2 \in O$, entonces ambos son alcanzables desde p, lo que significa que o_1 y o_2 están densamente-conectados. □

Además, un cluster C con respecto a ε y $MinPts$ está determinado de forma única por cualquier punto núcleo en él. Sin embargo, cada punto en C es densamente-alcanzable desde cualquiera de los puntos núcleo de C, y por lo tanto, un cluster C contiene exactamente a los puntos que son densamente-alcanzables desde algún punto núcleo arbitrario en C. Lo que se lee como sigue:

Lema 2 *Sea C un cluster con respecto a ε y MinPts y sea p un punto arbitrario en C tal que $|N_\varepsilon(p)| \geq MinPts$, entonces $C = O = \{o \mid o$ es densamente-alcanzable desde p (con respecto a ε y MinPts)$\}$.*

Demostración. Veamos que $C = O$. Para ver que $O \subseteq C$, sea $o \in O$, veamos que $o \in C$. Como $o \in O$, o es densamente-alcanzable desde p (con respecto a ε y $MinPts$), por lo que por la maximalidad de C, $o \in C$. Ahora, para ver que $C \subseteq O$,

vemos que si $c \in C$, lo cual implica que c y p están densamente-conectados entre sí, es decir, existe $a \in C$ tal que c y p son alcanzables desde a. Además, como $|N_\varepsilon(p)| \geq \varepsilon$, es decir p es punto núcleo, y como c es alcanzable desde a, y por definición ningún punto puede ser alcanzable desde otro punto que no sea punto núcleo, a también es punto núcleo. Luego, como la propiedad densamente-alcanzable es simétrica para puntos núcleo, a es alcanzable desde p, y como esta misma propiedad es transitiva en general, c es alcanzable desde p, lo que significa que $c \in O$. Lo que concluye la prueba. □

Notemos que entonces para el algoritmo DBSCAN, se toman en cuenta los dos parámetros ε (eps), y el mínimo número de puntos ($MinPts$) que deben estar en la ε-vecindad de un punto para que ésta sea considerada densa. El algoritmo comienza en algún punto arbitrario p y se fija en su ε-vecindad, y si ésta contiene suficientes puntos, entonces construye un cluster, es decir, reúne todos los puntos alcanzables desde p con respecto a ε y $MinPts$. Si p es punto núcleo, entonces se conforma un cluster con respecto a ε y $MinPts$. Si p es punto frontera, entonces no hay puntos alcanzables desde él, así que el algoritmo pasa al siguiente punto.

Aplicando el algoritmo al ejemplo de la Figura 1, con ε y $MinPts = 4$ dados, el algoritmo empezaría recorriendo un punto arbitrario, digamos N, como la ε-vecindad de N no cumple la condición de $MinPts$, lo etiqueta como ruido y pasa al siguiente punto, digamos B, que como no cumple $MinPts$, pasa al siguiente punto, digamos A, y como cumple $MinPts$ en su ε-vecindad, construye el cluster a partir de A, que consiste en los puntos de color rojo y amarillo en la figura 1.2, debido a que todos éstos son alcanzables desde A.

A continuación presentamos una versión básica de DBSCAN obtenida en [4]:

```
DBSCAN (SetOfPoints, Eps, MinPts)

// SetOfPoints is UNCLASSIFIED
   ClusterId := nextId(NOISE);
   FOR i FROM 1 TO SetOfPoints.size DO
      Point := SetOfPoints.get(i);
      IF Point.ClId = UNCLASSIFIED THEN
       IF ExpandCluster(SetOfPoints, Point,
              ClusterId, Eps, MinPts) THEN
          ClusterId := nextId(ClusterId)
       END IF
      END IF
   END FOR
END; // DBSCAN
```

Asimismo, en el Anexo A mostramos una versión del algoritmo DBSCAN en Python.

1.3. Clustering de enlace simple

El algoritmo de clustering de enlace simple pertenece a la clasificación de clustering aglomerativo dentro de los clusters jerárquicos, y consiste en combinar en cada paso dos clusters que contengan al par de puntos más cercanos entre sí que aún no pertenecen al mismo cluster que el otro.

Este método fue presentado por primera vez en [6] como una forma conveniente de resumir relaciones taxonómicas en forma de dendrogramas. Se busca que las relaciones entre n muestras sean expresadas en términos de las distancias taxonómicas entre cada par de éstas.

La distancia en este algoritmo de clustering está determinada por dos puntos, en clusters distintos que sean los más cercanos entre sí. Es decir, está determinada por la función distancia: $d(X,Y) = min\{d(x,y) \mid x \in X, y \in Y\}$, que denota la distancia entre los clusters X y Y.

Existen varias formas de abordar este algoritmo, por ejemplo, usando teoría de gráficas y mínimos árboles generadores para representar los dendrogramas, así como matrices de distancias.

A continuación mostramos un breve algoritmo esquemático presentado en [3] en el que se usan matrices de distancia para este algoritmo de clustering. De forma intuitiva, este algoritmo borra en cada paso filas y columnas de la matriz mientras los clusters son unidos para formar uno nuevo.

Consideremos $M \in M_{nxn}(\mathbb{R})$ tal que la entrada (i,j) de la matriz es precisamente la distancia entre los puntos i y j. Esto es $M_{i,j} = d(i,j)$. A cada paso del algoritmo se le asigna un número $0, 1, 2, ..., n-1$, y denotamos por (m) al cluster formado en el paso $m \in \{0, 1, 2, ..., n-1\}$. Denotamos por $d[(r),(s)]$ la distancia entre los clusters (r) y (s).

Entonces el algoritmo se lee como sigue a continuación:

1. Como todos los puntos inicialmente consisten en un cluster por sí mismos, todos los clusters son disjuntos entre sí y a todos se les asigna nivel $L(0) = (0)$ y valor asociado $m = 0$.

2. Consideramos el par (r), (s) tales que $d[(r),(s)] = min(d[(i),(j)])$. Con i, j clusters en el paso actual, $i \neq j$.

3. Redefinimos $m = m + 1$.

4. Unimos los clusters (r) y (s) del paso anterior en un mismo cluster y obtenemos un nuevo cluster m al cual denotamos por (r,s).

5. Rescribimos la matriz M eliminando las filas y columnas de los clusters (r) y (s) y agregamos una fila y columna entre ellos que corresponderá al cluster $(m) = (r,s)$ formado por la unión de (r) y (s) del paso anterior.

6. Definimos entonces la distancia entre $(m) = (r, s)$ y un cluster previo (k), como sigue: $d[(r, s), (k)] = min\{d[(k), (r)], d[(k), (s)]\}$.

7. Detener cuando todos los puntos estén en un mismo cluster *(if/break)*. De otra forma, repetir el algoritmo desde el paso 2 *(repeat)*.

A continuación presentamos un ejemplo de este algoritmo basado en el modelo de Jukes y Cantor de 1969 presentado en [17], en el que expresamos la distancia genética de cinco especies de bacterias: (a) *Bacillus* , (b) *Bacillus stearothermophilus*, (c) *Lactobacillus viridescens*, (d) *Acholeplasma modicum*, (e) *Micrococcus luteus.*

Ejemplo 2 *Consideremos la siguiente matriz de distancia M donde la entrada $M_{i,j} = M_{j,i}$ denota la distancia genética entre las especies i y j mencionadas arriba.*

$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{pmatrix} 0 & 17 & 21 & 31 & 23 \\ 17 & 0 & 30 & 24 & 21 \\ 21 & 30 & 0 & 28 & 39 \\ 31 & 24 & 28 & 0 & 43 \\ 23 & 21 & 39 & 43 & 0 \end{pmatrix} \end{array}$$

A continuación mencionamos una propiedad obtenida en la que se incorpora teoría de gráficas y ultrametricidad, lo cual puede consultarse en [3]. Si u es un nodo o vértice tal que a y b son adyacentes a u, tomamos $d(a, u) = d(b, u) = d(a, b)/2 = 17/2 = 8.5$, de forma que u equidista de a y de b. Por lo que las ramas en los dendrogramas que unen a a y b con u tendrán dicha longitud.

Notemos que $M_{a,b} = d(a, b) = 17$ es el menor valor en M, entonces para el primer paso, unimos los elementos a y b en un mismo cluster para formar una nueva matriz N de tamaño 4×4 como sigue:

Calculamos las nuevas distancias entre (a, b) y los puntos restantes c, d, e usando la propiedad mencionada previamente de tomar la mínima distancia entre los elementos de (a, b) y el tercer elemento:

$d((a, b), c) = min\{d(a, c), d(b, c)\} = min\{21, 30\} = 21 = N_{(a,b),c} = N_{c,(a,b)}$.
$d((a, b), d) = min\{d(a, d), d(b, d)\} = min\{31, 24\} = 31 = N_{(a,b),d} = N_{d,(a,b)}$.
$d((a, b), e) = min\{d(a, e), d(b, e)\} = min\{23, 21\} = 21 = N_{(a,b),e} = N_{e,(a,b)}$.

Observamos que las distancias $d(c, d) = d(d, c)$, $d(c, e) = d(e, c)$ y $d(d, e) = d(e, d)$ no se ven afectadas al unir los clusters a y b en uno mismo. De forma que la matriz N queda de la siguiente forma:

$$\begin{array}{c} \\ (a,b) \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{cccc} (a,b) & c & d & e \end{array} \\ \begin{pmatrix} 0 & 21 & 31 & 21 \\ 21 & 0 & 28 & 39 \\ 31 & 28 & 0 & 43 \\ 21 & 39 & 43 & 0 \end{pmatrix} \end{array}$$

Repitiendo el proceso, observamos que $N_{(a,b),c} = d((a,b),c) = 21 = N_{(a,b),e} = d((a,b),e)$ *son los valores mínimos de la matriz* N*, de forma que unimos los cluster* (a,b)*,* c *y* e *en uno mismo:*

$d(((a,b),c,e),d) = min\{d((a,b),d), d(c,d), d(e,d)\} = min\{31,28,43\} = 28 = L_{((a,b),c,e),d} = L_{d,((a,b),c,e)}$.

De esta forma, la matriz L *queda de la siguiente manera:*

$$\begin{array}{c} \\ ((a,b),c,e) \\ d \end{array} \begin{array}{c} \begin{array}{cc} ((a,b),c,e) & d \end{array} \\ \begin{pmatrix} 0 & 28 \\ 28 & 0 \end{pmatrix} \end{array}$$

Como paso último, unimos los clusters $((a,b),c,e)$ *y* d *en uno mismo, de forma que todos los puntos que inicialmente formaban un cluster por sí mismos, ahora se encuentran todos en un mismo cluster.*

Asimismo, podemos representar esta sucesión de clustering de la siguiente manera:

$C_1 = \{a,b,c,d,e\}$
$C_2 = \{\{a,b\},c,d,e\}$
$C_3 = \{\{\{a,b\},c,e\},d\}$
$C_4 = \{\{\{\{a,b\},c,e\},d\}\}$

Finalmente, a continuación presentamos la representación en forma de dendrograma:

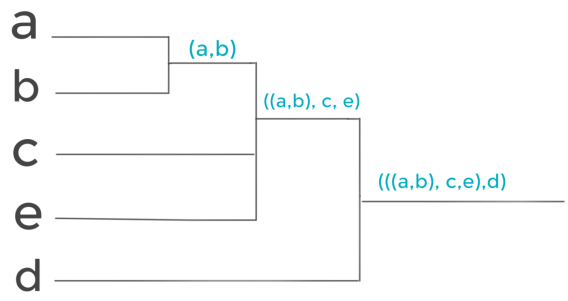


Figura 1.3: Representación del clustering en forma de dendrograma.

2 El método de Mapper

En este capítulo abordaremos el tema de la gráfica Mapper, primero revisaremos algunos conceptos y definiciones básicas de topología, como homotopía y el nervio de una cubierta, basándonos en [10] y [7], así como la definición de Gráfica de Reeb como motivación para concluir el capítulo con la definición y construcción de la gráfica Mapper, desde las perspectivas de [9] y [2].

2.1. El nervio de una cubierta

2.1.1. Homotopía

Recordemos las siguientes definiciones:

Definición 2.1.1 *Decimos que dos funciones continuas $f, g : X \to Y$ son homotópicas si existe una función continua $H : X \times [0,1] \to Y$ tal que $H(x,0) = f(x)$ y $H(x,1) = g(x)$.*

Ejemplo 3 *Consideremos $f, g : \mathbb{R} \to \mathbb{R}^2$ dadas por $f(x) = (x, x^2)$ y $g(x) = (x, e^x)$. Vemos que f y g son homotópicas vía $H : \mathbb{R} \times [0,1] \to \mathbb{R}^2$ dada por $H(x,t) = (x, (1-t)x^2 + te^x)$, que claramente cumple $H(x,0) = f(x)$ y $H(x,1) = g(x)$.*

Definición 2.1.2 *Una función continua $f : X \to Y$ es una equivalencia homotópica si existe una función continua $g : Y \to X$ tal que $f \circ g$ es homotópica a la identidad en Y y $g \circ f$ es homotópica a la identidad en X.*

Definición 2.1.3 *Dos espacios X y Y son homotópicamente equivalentes si existe una equivalencia homotópica $f : X \to Y$ entre ellos. Se dice que un espacio es contráctil si es homotópicamente equivalente a un punto.*

Ejemplo 4 *Veamos que $\mathbb{R}^n$ y un punto $\{0\}$ son homotópicamente equivalentes. Sean $f : \mathbb{R}^n \to \{0\}$ y $g : \{0\} \to \mathbb{R}^n$ tales que $f(x) = g(0) = 0$. Vemos que $f \circ g : \{0\} \to \{0\}$, $g \circ f : \mathbb{R}^n \to \mathbb{R}^n$ son tales que $(f \circ g)(0) = 0$ y $(g \circ f)(x) = 0$. De forma que $f \circ g = Id_{\{0\}}$ y $g \circ f$ es homotópica a la identidad en $\mathbb{R}^n$ vía $H : \mathbb{R}^n \times [0,1] \to \mathbb{R}^n$ dada por $H(x,t) = tx$, entonces $H(x,0) = 0 = (g \circ f)(x)$ y $H(x,1) = x = Id_{\mathbb{R}^n}(x)$.*

Con esto, también vemos que $\mathbb{R}^n$ es contráctil, al ser homotópicamente equivalente a un punto.

Definición 2.1.4 *Un retracto por deformación, de un espacio X sobre un espacio A, es una familia de funciones $f_t : X \to X$, $t \in I$, tales que $f_0 = Id$, $f_1(X) = A$ y $f_t|A = Id$ para toda t.*

Ejemplo 5 *S^n es un retracto por deformación de $\mathbb{R}^{n+1}\backslash\{0\}$, mediante $f(x,t) = \left((1-t) + \frac{t}{\|x\|}\right)x$.*

Nervio de una cubierta

Definición 2.1.5 *Una cubierta abierta finita de un espacio topológico X es una colección de conjuntos abiertos $\mathcal{U} = \{U_1, ..., U_n\}$, tales que $X = \bigcup_{i=1}^{n} U_i$.*

Definición 2.1.6 *Una cubierta abierta finita $\mathcal{U}$ de un espacio topológico es una buena cubierta si cada intersección no vacía $\bigcap_{i\in\sigma} U_i$, para $\sigma \subseteq \{1, ..., n\}$, es contráctil.*

Definición 2.1.7 *Definimos el nervio de una cubierta abierta finita $\mathcal{U}$ como el complejo simplicial abstracto $\mathcal{N}(\mathcal{U}) := \{\sigma \subset [n] | \bigcap_{i\in\sigma} U_i \neq \varnothing\}$, donde $[n] = \{1, ..., n\}$.*

Ejemplo 6 *Consideremos la cubierta $\mathcal{U}$ como se muestra en la figura 2.1, la cual no es una buena cubierta. Entonces se tiene que $\mathcal{N}(\mathcal{U})$ será también como en la figura 2.1:*

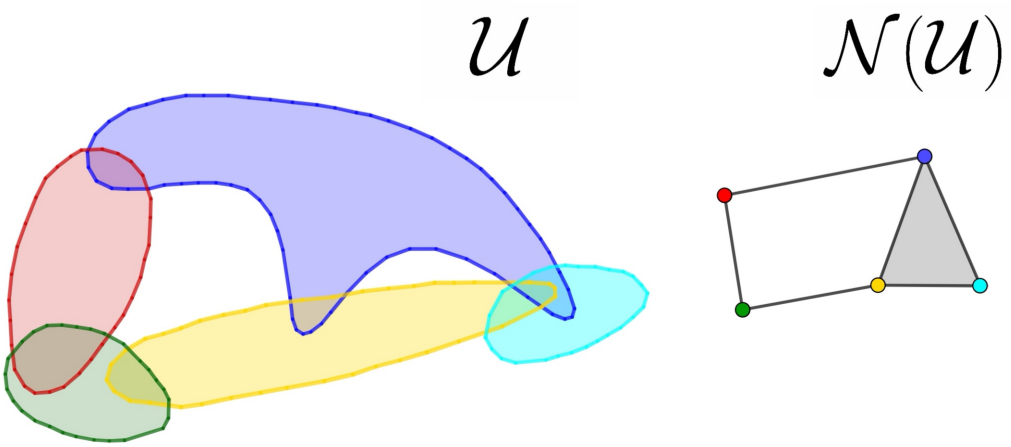


Figura 2.1: $\mathcal{U}$ y $\mathcal{N}(\mathcal{U})$

2.2. Teorema del nervio

Teorema 1 *(Teorema del nervio). Sea $\mathcal{U}$ una buena cubierta de un espacio topólogico X. Entonces $\mathcal{N}(\mathcal{U})$ es homotópicamente equivalente a X. En particular, $\mathcal{N}(\mathcal{U})$ y X tienen exactamente los mismos grupos de homología.*

Ejemplo 7 *Consideremos la circunferencia unitaria S^1, y las cubiertas $U = \{U_1, U_2, U_3\}$ y $W = \{W_1, W_2\}$ como en la figura. Vemos que U es una buena cubierta, mientras que W no lo es, pues $W_1 \cap W_2$ consiste de dos componentes conexas, lo cual no es contráctil.*

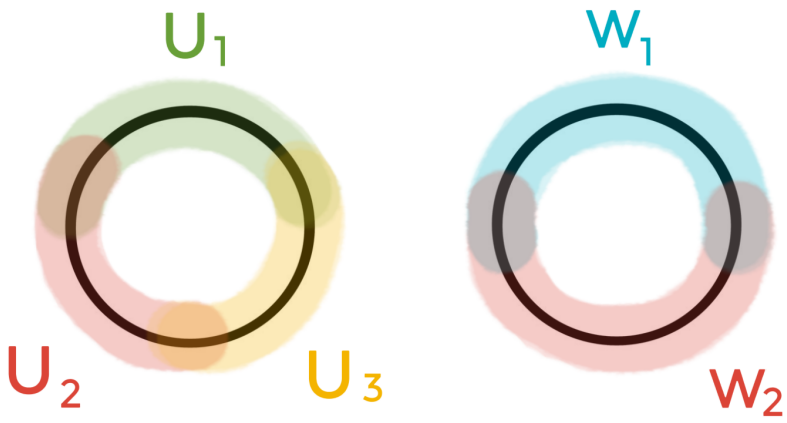


Figura 2.2: Cubiertas U y W de S^1

Entonces se tiene que $N(U)$ y $N(W)$ son también como en la figura:

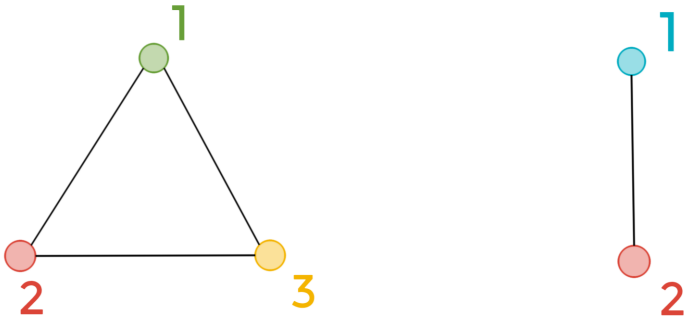


Figura 2.3: Nervio de la cubierta U (izquierda) y nervio de la cubierta W (derecha)

Que claramente $N(U)$ es homotópicamente equivalente a S^1, mientras que $N(W)$ es homotópicamente equivalente a un punto, lo cual no es homotópicamente equivalente a S^1.

Cabe mencionar que en este trabajo, ilustramos a los elementos de una cubierta $\mathcal{U}$ como subconjuntos del plano para mayor claridad; sin embargo, es importante recalcar que formalmente $\mathcal{U}$ consiste de los abiertos relativos obtenidos de la intersección entre elementos de U' y X, donde los elementos de U' son abiertos del plano ilustrados en la figura.

A continuación mostramos la idea de la demostración del Teorema del Nervio, como se muestra en [15]. El caso general puede encontrarse en el capítulo 4 de [11].

Idea de la demostración. A continuación realizamos el caso de la demostración cuando el nervio es de dimensión 1, i.e., no hay triples intersecciones de elementos de la cubierta U. La demostración en general es más técnica pero se siguen en términos generales los mismos pasos que mostramos a continuación.

Definimos $Z \subset X \times \mathcal{N}(\mathcal{U})$ como: $Z = \bigcup_{\sigma \in \mathcal{N}(\mathcal{U})} \left(\bigcap_{s \in \sigma} U_s \times \sigma \right)$.

Demostraremos que $Z \simeq X$ y $Z \simeq \mathcal{N}(\mathcal{U})$. Para $Z \simeq X$, observemos que para cada $x \in X$ el conjunto $(x \times \mathcal{N}(\mathcal{U})) \cap Z$ es un simplejo, que para el caso 1-dimensional es un vértice o una arista, en el nervio generado por todo $s \in \sigma$, para el cual $x \in U_s$. Contrayendo simultáneamente cada uno de estos simplejos a un punto para cada $x \in X$ proporciona un retracto por deformación de Z a X. Por tanto, $Z \simeq X$.

Para ver que $Z \simeq \mathcal{N}(\mathcal{U})$, observemos que para cada $y \in \mathcal{N}(\mathcal{U})$, el conjunto $(X \times y) \cap Z$ es contráctil por hipótesis. Entonces contraemos primero los conjuntos de esta forma para todos los y que no son vértices, y después contraemos todos los conjuntos para los vértices. Obtenemos entonces un retracto por deformación de Z a $\mathcal{N}(\mathcal{U})$, y por lo tanto, $Z \simeq \mathcal{N}(\mathcal{U})$.

Ejemplo 8 *Retomando lo explicado en el ejemplo 7, consideremos nuevamente $X = S^1$ y la buena cubierta U como se muestra en la figura 2.4. De esta forma, $N(U)$ y el espacio Z construido en la idea de la demostración son como sigue:*

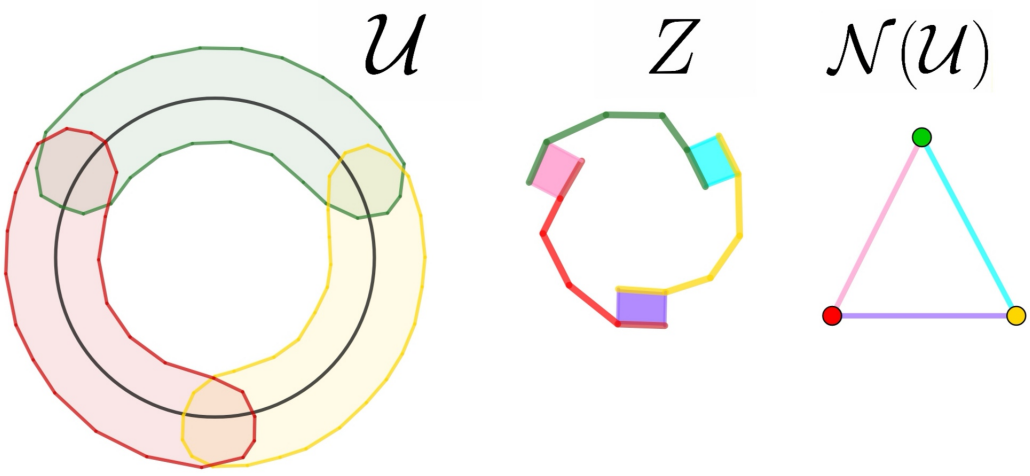


Figura 2.4: La cubierta U, $N(U)$ y el espacio Z.

Para ver que $Z \simeq S^1 = X$, contraemos las secciones como muestran las flechas en la figura 2.5 sobre los puntos $x \in X$ en Z, que corresponden a los aristas en $\mathcal{N}(\mathcal{U})$ para obtener S^1.

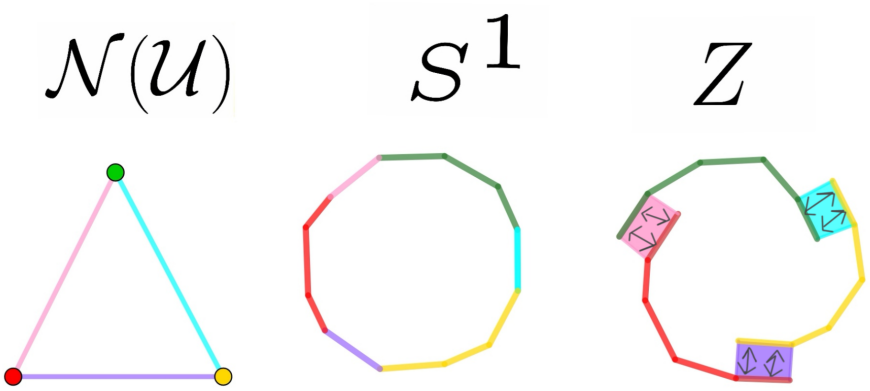


Figura 2.5: El espacio Z y S^1.

Para ver que $Z \simeq \mathcal{N}(\mathcal{U})$, contraemos primero las secciones que no son vértices $y \in \mathcal{N}(\mathcal{U})$ como muestran las flechas en Z en la figura 2.6, y se obtiene el espacio que también se muestra en la figura 2.6. Finalmente contraemos las secciones de vértices $y \in \mathcal{N}(\mathcal{U})$ como indican las flechas al centro de la figura 2.6:

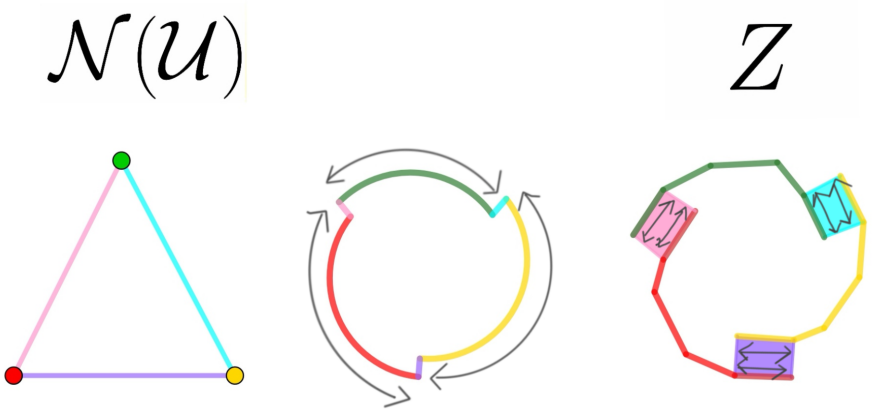


Figura 2.6: Z, el espacio obtenido al contraer las secciones, y $\mathcal{N}(\mathcal{U})$.

(Figuras modificadas de [15]).

2.3. Gráficas de Reeb

La gráfica de Reeb de una función escalar, o que toma valores reales, en una variedad diferenciable describe la conectividad de sus conjuntos de nivel.

Llamamos conjunto de nivel a la preimagen de un valor real bajo una función escalar o altura.

La gráfica de Reeb de una función escalar f definida en una variedad es obtenida al colapsar cada componente conexa de un conjunto de nivel a un punto.

Definición 2.3.1 *Sea M una variedad topológica y sea $f : M \to \mathbb{R}$ una función continua que toma valores reales, definimos una relación de equivalencia $\sim$ en M como sigue: para $x, y \in M$, $x \sim y$ si $f(x) = f(y)$ y x y y pertenecen a la misma componente conexa de un mismo conjunto de nivel $f^{-1}(f((x)) = f^{-1}(f((y))$. La gráfica de Reeb es entonces el espacio cociente $M/\sim$ con la topología cociente.*

Como información adicional, cabe mencionar la relación del concepto de gráfica de Reeb con teoría de Morse, ya que en el caso en que f es un función de Morse con distintos valores críticos, la gráfica de Reeb se puede describir más. Los nodos o vértices de la gráfica que corresponden a los conjuntos de nivel de valores críticos $f^{-1}(c)$ expresarán el cambio en la topología de las componentes conexas de los conjuntos de nivel.

Ejemplo 9 *Consideremos el siguiente toro vertical como se muestra en la figura 2.7, y consideremos la función altura $f : \mathbb{R}^3 \to \mathbb{R}$ dada por $f(x, y, z) = z$. Entonces, la gráfica de Reeb se ve como sigue, al colapsar cada componente conexa a un punto:*

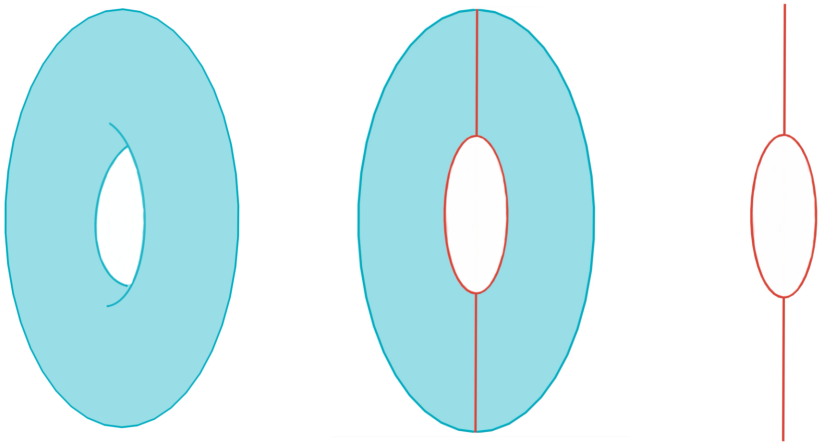


Figura 2.7: Toro vertical y su Gráfica de Reeb

2.4. La gráfica Mapper

2.4.1. La gráfica pre-Mapper

La idea de la gráfica de Mapper se basa en la consideración de si es posible construir un análogo a una gráfica de Reeb, para un conjunto finito X.

Definición 2.4.1 *Sea $L \subset \mathbb{R}^n$ una variedad y sea $f : L \to \mathbb{R}$ una función continua. Dada una cubierta abierta $U = \{U_i\}_{i \in I}$ de L, se define el nervio 1-dimensional de U, $N_1(U)$, como la gráfica cuyos vértices están dados por los elementos de la cubierta abierta, es decir, para cada $U_i \in U$, existe un vértice V_i en $N_1(U)$, y la arista V_iV_j pertenece a $N_1(U)$ si y sólo si $U_i \cap U_j$ es no vacía.*

Ejemplo 10 *Sea $U = \{U_1, U_2, U_3\}$ una cubierta abierta de $L = [0, 10] \subset \mathbb{R}$, donde $U_1 = (-1, 4)$, $U_2 = (3, 7)$, $U_3 = (6, 11)$.*

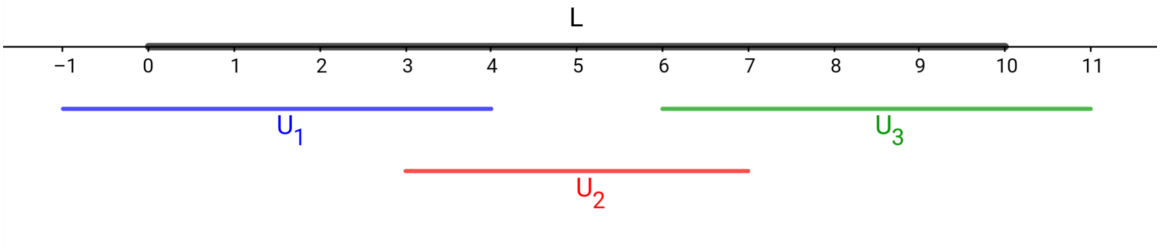


Figura 2.8: L y su cubierta abierta U.

Los vértices de $N_1(U)$ son entonces el conjunto $V(N_1(U)) = \{V_1, V_2, V_3\}$, y vemos que como $U_1 \cap U_2 \neq \varnothing$, $U_2 \cap U_3 \neq \varnothing$ y $U_1 \cap U_3 = \varnothing$, los aristas de $N_1(U)$ son el conjunto $E(N_1(U)) = \{V_1V_2, V_2V_3\}$.

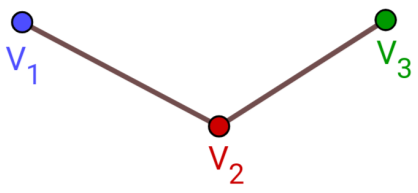


Figura 2.9: Nervio de la cubierta U.

Consideremos una cubierta abierta finita $W = \{W_j\}_{j\in J}$ de $f(L) \subset \mathbb{R}$, la imagen de la función f, y con J un conjunto de índices, y donde cada $W_j \subset \mathbb{R}$ es un intervalo abierto. Es decir, $f(L) \subseteq \bigcup W_j$. Observemos que $f^{-1}(W_j)$ es abierto pero puede no ser conexo, en cuyo caso, es unión de abiertos conexos U_i, y entonces $U = \{U_i\}_{i\in I}$ es una cubierta abierta de L donde cada $U_i \subset f^{-1}(W_j)$ y además U_i es conexo para toda $i \in I$, donde es posible que $\mid I \mid \geq \mid J \mid$.

Definición 2.4.2 *Definimos la gráfica pre-Mapper como el nervio 1-dimensional de U, $M := N_1(U)$, de (L, f, W).*

Notemos que M también depende de W, pero sólo la denotaremos como la gráfica pre-Mapper de (L, f).

Ejemplo 11 *Sea $L = Fr([0,1]\times[0,1])$, el cuadrado unitario, sea $f : L \to \mathbb{R}$ dada por $f(x, y) = x$ la función proyección de la primer coordenada, y sea $W = \{W_1, W_2, W_3\}$ cubierta abierta de $f[L]$, dada por $W_1 = (-\frac{1}{2}, \frac{1}{3})$, $W_2 = (\frac{1}{4}, \frac{3}{4})$, $W_3 = (\frac{2}{3}, 2)$, y como se muestra en la figura 2.10.*

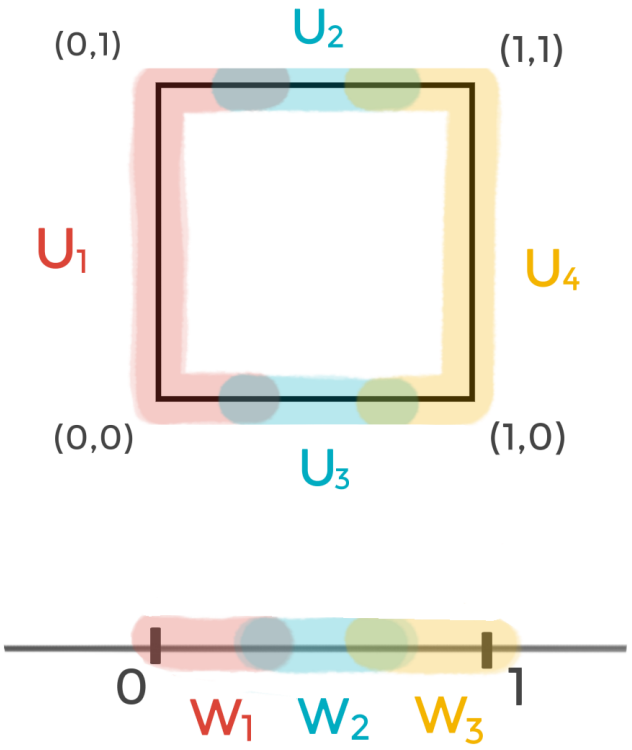


Figura 2.10: L y $f[L]$ con sus respectivas cubiertas.

Entonces $U = \{U_1, U_2, U_3, U_4\}$, donde $U_1 = f^{-1}(W_1) = ([0, \frac{1}{3}) \times \{0, 1\}) \cup (\{0\} \times [0, 1])$, $f^{-1}(W_2) = U_2 \cup U_3 = ((\frac{1}{4}, \frac{3}{4}) \times \{0\}) \cup ((\frac{1}{4}, \frac{3}{4}) \times \{1\})$, $U_2 = (\frac{1}{4}, \frac{3}{4}) \times \{0\}$ y $U_3 = (\frac{1}{4}, \frac{3}{4}) \times \{0\}$, y $U_4 = f^{-1}(W_3) = ((\frac{2}{3}, 1] \times \{0, 1\}) \cup (\{1\} \times [0, 1])$.

Donde U_1, U_2, U_3, U_4 son abiertos relativos en L.

De forma que la gráfica pre-mapper $M := N_1(U)$ de L, estará dada por los vértices $V(M) = \{V_1, V_2, V_3, V_4\}$, uno por cada U_i, y las aristas $E(M) = \{V_1V_2, V_1V_3, V_2V_3, V_3V_4\}$. Ya que $U_1 \cap U_2 \neq \varnothing$, $U_2 \cap U_3 = \varnothing$, $U_2 \cap U_4 \neq \varnothing$ y $U_3 \cap U_4 \neq \varnothing$.

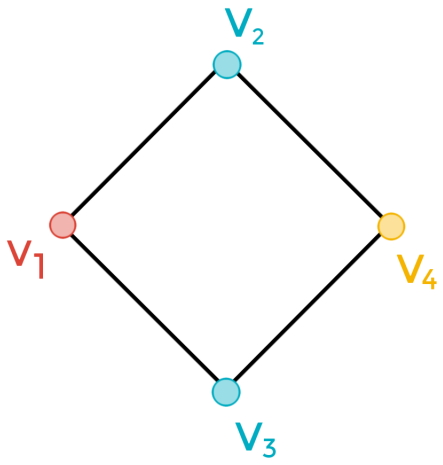


Figura 2.11: Gráfica pre-Mapper de L.

2.4.2. La gráfica Mapper

A continuación presentamos una definición básica de la gráfica Mapper, obtenida en [9]:

Definición 2.4.3 *Dada una nube de puntos $X \subset \mathbb{R}^n$, una función $f : X \to \mathbb{R}$ y una cubierta abierta $U = \{U_i\}_{i \in I}$ de X, se define el nervio 1-dimensional de U, $N_1(U)$, como la gráfica cuyos vértices están dados por los elementos de la cubierta abierta, es decir, para cada $U_i \in U$, existe un vértice V_i en $N_1(U)$, y el arista V_iV_j pertenece a $N_1(U)$ si y sólo si $U_i \cap U_j$ es no vacía.*

Para la construcción de la gráfica Mapper, consideramos una cubierta abierta finita $V = \{V_j\}_{j \in J}$ de $f(X) \subset \mathbb{R}$, la imagen de la función f, y con J un conjunto de índices. Es decir, $f(X) \subseteq U(V_j)$. Ahora, sea U la cubierta abierta de X que se obtiene al tomar los clusters inducidos por los puntos en $f^{-1}(V_j)$ para cada $j \in J$.

Definición 2.4.4 *El nervio 1-dimensional de U, $M := N_1(U)$, se llama la gráfica Mapper de (X, f).*

Notemos que dicha construcción recae en los parámetros de la función f, la cubierta V y el algoritmo de clustering.

A continuación presentamos una definición más general, como se muestra en [2]:

Recordemos que dada una cubierta abierta $U = \{U_i\}_{i\in I}$ de un espacio X, definimos el nervio de la cubierta U como el complejo simplicial $N(U)$ cuyo conjunto de vértices es el conjunto de índices I y donde una familia $\{i_0, i_1, ..., i_k\}$ genera un k-simplejo en $N(U)$ si y sólo si $U_{i_0} \cap U_{i_1} \cap \ldots \cap U_{i_k} \neq \varnothing$.

Cabe mencionar que si $\{V_0, V_1, ..., V_k\}$ son los vértices de un simplejo, entonces los puntos V en el simplejo, tienen una correspondencia uno a uno con el conjunto de k-tuplas ordenadas de números reales $(r_0, r_1, ..., r_k)$ que satisfacen que $0 \leq r_i \leq 1$, y $\sum_{i=0}^{k} r_i = 1$. A esta correspondencia se le llama las coordenadas baricéntricas, y los números r_i son las coordenadas baricéntricas del punto V.

Ahora, supongamos que dado un espacio equipado con una función continua $f : X \to Z$, donde Z es un espacio de parametrización y está equipado con una cubierta $U = \{U_\alpha\}_{\alpha\in A}$, para algún conjunto de índices finito A. Como f es continua, los conjuntos $f^{-1}(U_\alpha)$ también forman una cubierta abierta de X. Para cada α, podemos considerar la descomposición de $f^{-1}(U_\alpha)$ en sus componentes arcoconexas, entonces se tiene que $f^{-1}(U_\alpha) = \bigcup_{i=1}^{j_\alpha} V(\alpha, i)$, donde j_α es el número de componentes conexas en $f^{-1}(U_\alpha)$. Sea $\overline{U}$ la cubierta de X obtenida de esta forma a partir de la cubierta U de Z.

Luego, si tenemos dos cubiertas $U = \{U_\alpha\}_{\alpha\in A}$ y $V = \{V_\beta\}_{\beta\in B}$ de un espacio X, una función de cubiertas de U a V es una función $f : A \to B$, de forma que para todo $\alpha \in A$, se tiene que $U_\alpha \subseteq V_{f(\alpha)}$ para todo $\alpha \in A$.

Ejemplo 12 *Sea $X = [0, 5] \subseteq \mathbb{R}$ y $\varepsilon > 0$. Los conjuntos $I_l^\varepsilon = (l-\varepsilon, l+\varepsilon+1)\cap X$, con $l = 0, 1, ..., 4$ son una cubierta abierta de X. Estas cubiertas, para diferentes valores de ε, tienen el mismo conjunto de índices, y si $\varepsilon \leq \varepsilon'$, la función identidad en este conjunto de índices es una función de cubiertas, ya que $I_l^\varepsilon \subseteq I_l^{\varepsilon'}$.*

Ejemplo 13 *Sea $X = [0, 4] \times [0, 4] \subseteq \mathbb{R}^2$ como en el ejemplo 11. Sea $\varepsilon > 0$, y sea $B_\varepsilon(i, j) = (i - \varepsilon, i + 1 + \varepsilon) \times (j - \varepsilon, j + 1 + \varepsilon)$. El conjunto $\{B_e(i, j)\}_{0\leq i,j\leq 3}$ es una cubierta B_ε de X, y nuevamente, si $\varepsilon \leq \varepsilon'$, la función identidad en el conjunto de índices $\{(i, j)|0 \leq i, j \leq 3\}$ es una función de cubiertas $B_\varepsilon \to B_{\varepsilon'}$.*

Notemos que si tenemos una función $f : A \to B$ entre las cubiertas U y V que cumple estas condiciones, existe una función inducida de complejos simpliciales $N(f) : N(U) \to N(V)$, dada en vértices por la función f. Por lo tanto, si tenemos una familia de cubiertas $\{Ui\}_{i=0}^{n}$ y una función de cubiertas $f_i : U_i \to U_{i+1}$ para cada i, obtenemos la siguiente sucesión de complejos simpliciales y funciones simpliciales: $N(U_0) \overset{N(f_0)}{\to} N(U_1) \overset{N(f_1)}{\to} ... \overset{N(f_{n-1})}{\to} N(U_n)$.

Si consideramos un espacio X equipado con una función $f : X \to Z$, donde Z es un espacio de parametrización, y dada una función de cubiertas $g : U \to V$ en Z,

existe una función de cubiertas $h : \overline{U} \to \overline{V}$ del espacio X, pues si $U \subseteq V$, entonces $f^{-1}(U) \hookrightarrow f^{-1}(V)$, y por lo tanto, cada componente conexa de $f^{-1}(U)$ está contenida en exactamente una componente conexa de $f^{-1}(V)$. De forma que h está dada al pedir que el conjunto $U_\alpha(i)$ vaya al único conjunto de la forma $V_{f(\beta)}(j)$, de forma que $U_\alpha(i) \subseteq V_{h(\alpha)}(j)$.

Ejemplo 14 *Sea* $X = [-1.5, 1.5] \subseteq \mathbb{R}$, *sea* $Z = [0, +\infty)$ *el espacio de parametrización y sea* $f : X \to \mathbb{R}$ *dada por* $f(x) = \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}}$, *la función de densidad de probabilidad de la distribución normal (también llamada distribución gaussiana), con media* $\mu = 0$ *y varianza* $\sigma^2 = 1$. *Sea* $U = \{[0, 0.15), (0.13, 0.22), (0.2, 0.32), (0.3, +\infty)\}$ *cubierta de* Z.

Figura 2.12: Función de distribución gaussiana y la cubierta U de Z

Vemos que $f^{-1}((0.3, +\infty))$ *consiste de una sola componente conexa, mientras que* $f^{-1}([0, 0.15))$, $f^{-1}((0.13, 0.22))$, *y* $f^{-1}((0.2, 0.32))$ *consisten cada una de dos componentes conexas (una por la parte positiva y una por la parte negativa en* X*).*

De forma que el complejo simplicial asociado a U *será el siguiente:*

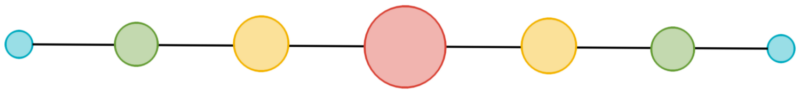


Figura 2.13: Nervio de U.

Cabe mencionar que es posible representar los vértices o nodos de cada complejo simplicial con color y tamaño. El color, como hasta ahora, puede representar cada conjunto correspondiente en la cubierta $\overline{U}$, o como en este ejemplo, donde el rojo represente que el valor de f es alto, y el azul, que el valor de f es bajo en puntos representativos de cada conjunto de $\overline{U}$ o en un promedio adecuado tomado sobre cada conjunto. Mientras que el tamaño del nodo indica la cantidad de puntos en X representados por cada nodo. De esta forma, el complejo brinda información sobre la función f.

Ejemplo 15 *De forma similar a los ejemplos 7 y 11, consideremos $S^1 = \{(x, y)|x^2 + y^2 = 1\}$ en $\mathbb{R}^2$. Sea $f(x, y) = y$, la segunda proyección canónica, $Z = [-1, 1]$ el espacio de parametrización y $U = \{[-1, -\frac{1}{3}), (-\frac{1}{2}, \frac{1}{2}), (\frac{1}{3}, 1]\}$ la cubierta de Z. De forma que similarmente a los ejemplos 7 y 11, la cubierta $\overline{U}$ será como se muestra en la figura 2.14*

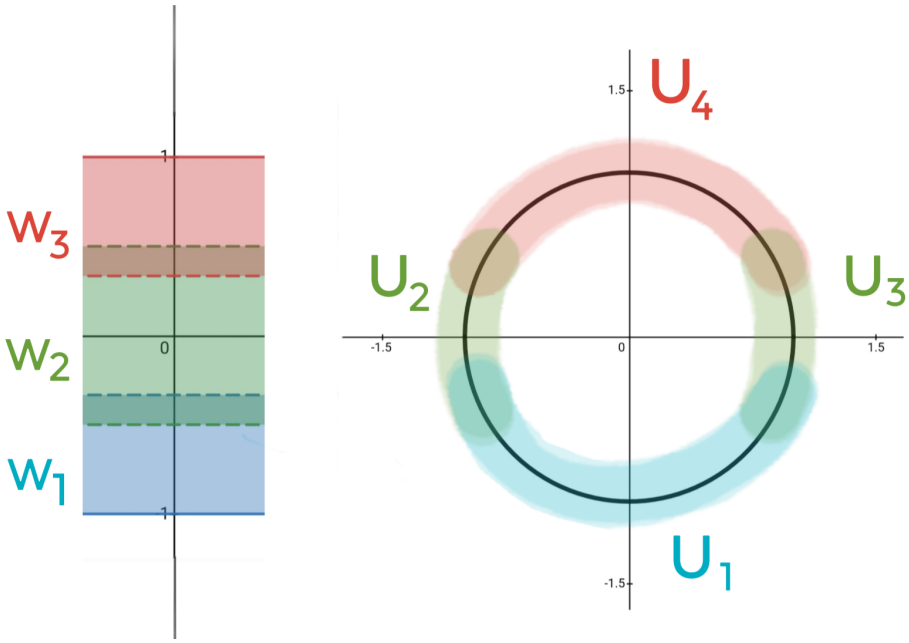


Figura 2.14: X, $Z = [-1, 1]$ y las cubiertas U de Z y $\overline{U}$ de X

Dado que $f^{-1}([-1, -\frac{1}{3}))$ y $f^{-1}((\frac{1}{3}, 1])$ consisten de una componente conexa, mientras que $f^{-1}((-\frac{1}{2}, \frac{1}{2}))$ consiste de dos componentes conexas, se tiene que nuevamente el complejo simplicial será como sigue:

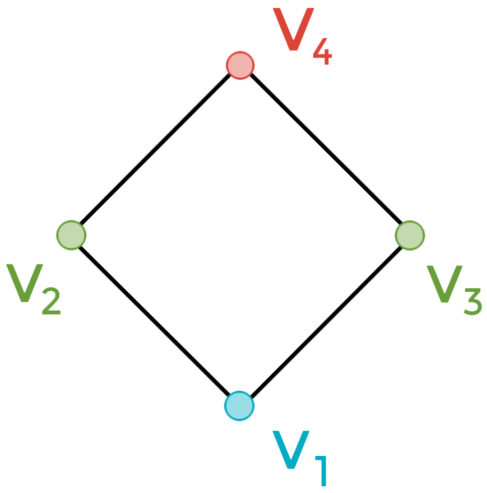


Figura 2.15: Nervio de $\overline{U}$

A continuación describimos la implementación de una versión estadística de Mapper, que fue desarrollada para una nube de puntos, mostrado en [2]. La idea principal se basa en que al pasar de la versión topológica a la versión estadística es que el clustering debe ser considerado como la versión estadística de la noción geométrica de particionar un espacio en sus componentes conexas.

Supongamos que la nube de puntos N contiene puntos $x \in X$ y que se tiene una función $f : X \to \mathbb{R}$ cuyo valor se conoce para los puntos de la nube N. A esta función le llamamos filtro. Asumimos también que es posible calcular distancias entre los puntos de la nube. Más específicamente, debe ser posible construir una matriz de distancia dadas estas distancias entre conjuntos de puntos.

Comenzamos encontrando el rango de la función restringida a los puntos dados. Para encontrar una cubierta de la nube dada, dividimos este rango en un conjunto de intervalos pequeños que se traslapan. Esto nos da dos parámetros que pueden utilizarse para controlar la resolución, es decir, la longitud de los intervalos más pequeños y el porcentaje de traslape entre intervalos sucesivos.

Ahora, para cada intervalo $I_j \in S$, encontramos el conjunto de puntos $X_j = \{x | f(x) \in I_j\}$ que conforman el dominio. Entonces se tiene que el conjunto $\{X_j\}$ es una cubierta de X y $X \subseteq \bigcup_j X_j$. Para cada conjunto X_j encontramos clusters X_{j_k}. Cada vértice en el complejo simplicial representará un cluster, y la arista entre dos vértices pertenecerá a la gráfica si $X_{j_k} \cap X_{l_m} \neq \varnothing$, es decir, cuando los clusters correspondientes a esos dos vértices, tengan intersección no vacía.

Ejemplo 16 *Retomando el ejemplo 9, consideremos el caso en el que nuestro conjunto es un toro vertical T, y por otro lado X es una nube de puntos como se muestra en la figura 2.17 y $f : \mathbb{R}^3 \to \mathbb{R}$ tal que $f(x, y, z) = z$, la función altura. Por lo que, en ambos casos, la gráfica Mapper de T y X, respectivamente, será como sigue:*

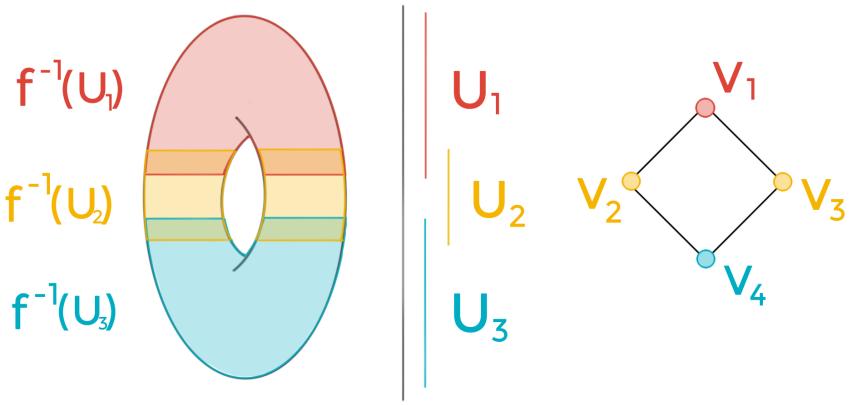


Figura 2.16: T y su gráfica Mapper.

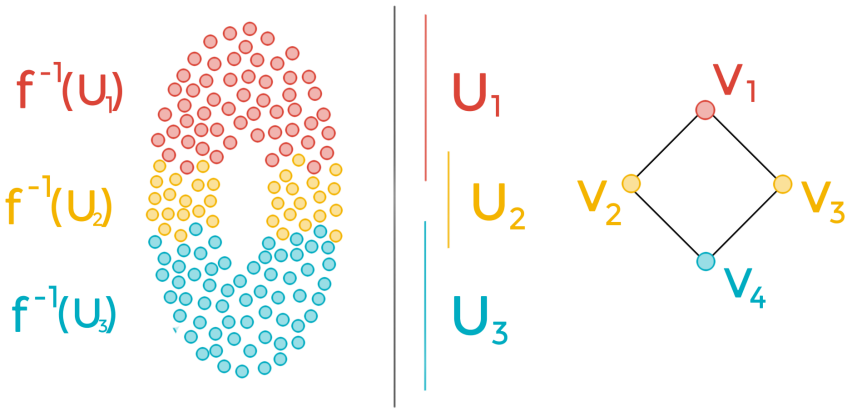


Figura 2.17: X y su gráfica Mapper.

Cabe mencionar que encontrar un buen algoritmo de clustering es fundamental para obtener el complejo simplicial correspondiente. Mapper no establece condiciones en el uso de algoritmos de clustering, por lo que cualquier algoritmo puede ser usado. Sin embargo, en [2] se especifica que fue implementado el siguiente algoritmo de clustering, con las siguientes características:

1. Se toma la matriz de distancia $M \in \mathbb{R}^{N \times N}$ entre puntos, como entrada.

2. No es necesario especificar de forma previa el número de clusters.

Asimismo, en [2] se menciona que fue implementado un algoritmo basado en el algoritmo de clustering de enlace simple que se menciona en la sección 1.3. Este algoritmo proporciona un vector $C \in \mathbb{R}^{N-1}$ que contiene la longitud del arista que fue agregado para reducir el número de clusters uno a uno por cada paso en el algoritmo.

Ahora, para encontrar el número de clusters, se usa la longitud de la arista en la cual fue unido cada cluster. Se tendrá que la distancia entre los puntos en cada cluster será más pequeña que la distancia entre los clusters, entonces las aristas más pequeñas serán usadas para conectar puntos en cada cluster, pero aristas relativamente más grandes para unir los clusters.

También cabe mencionar que es posible ampliar el método a espacios de parametrización de dimensiones mayores. Al usar una función a modo de filtro, obtenemos como *output* del algoritmo un complejo simplicial donde la dimensión de los simplejos es a lo más uno, es decir, aristas en una gráfica. Cualitativamente, la única información que se puede obtener de esto es el número de componentes, número de lazos, entre otros. Para obtener más información de dimensiones mayores, es necesario construir un complejo simplicial de mayor dimensión, utilizando más funciones.

En general, recordemos que la construcción de Mapper requiere como entradas o *Input*, un espacio de parametrización definido por las funciones, y una cubierta de este espacio. Notemos que cualquier cubierta del espacio puede ser usada. Una forma natural de construir complejos simpliciales de dimensiones mayores es asociar funciones con cada punto en vez de una única función. Si usáramos M funciones con $\mathbb{R}^M$ el espacio de parametrización, sería necesario encontrar una cubierta de un hipercubo de dimensión M, definido por el rango de las M funciones.

Finalmente, a continuación describimos el algoritmo de Mapper usando dos funciones f_1 y f_2, y $\mathbb{R}^2$ como espacio de parametrización. Consideremos dos funciones en cada punto, y que el rango de estas funciones sea cubierto por rectángulos. Definimos la región $R = [minf_1, maxf_1] \times [minf_2, maxf_2]$. Y sea $\bigcup_{i,j} A_{ij}$ una cubierta tal que cada $A_{i,j}$, $A_{i+1,j}$ y $A_{i,j}$, $A_{i,j+1}$ se intersectan respectivamente.

Un algoritmo para construir un complejo simplicial reducido es el siguiente:

1. Para cada i, j, seleccionar todos los puntos para los cuales los valores de las funciones f_1 y f_2 están en $A_{i,j}$. Encontrar un clustering de puntos para este conjunto y considerar cada cluster para representar un simplejo de dimensión 0 (o un vértice, como se refiere en este algoritmo). Asimismo, tener una lista de vértices para cada $A_{i,j}$ y un conjunto de índices de los puntos miembros del cluster asociados con cada vértice.

2. Para todos los vértices en los conjuntos $\{A_{i,j}, A_{i+1,j}, A_{i,j+1}, A_{i+1,j+1}\}$, si la intersección de los clusters asociados a los vértices es no vacía, agregar un 1-simplejo (o un arista, como se refiere en este algoritmo).

3. Cuando los clusters correspondientes a cualesquiera tres vértices tengan intersección no vacía, agregar el 2-simplejo correspondiente (o un triángulo, como se refiere

en este algoritmo), con los tres vértices conformando su conjunto de vértices.

4. Cuando los clusters correspondientes a cualesquiera cuatro vértices tengan intersección no vacía, agregar un 3-simplejo (o un tetraedro, como se refiere en este algoritmo), con los cuatro vértices conformando su conjunto de vértices.

Ejemplo 17 *Consideremos S^2, la esfera unitaria en $\mathbb{R}^3$, y sean $f, g : S^2 \to \mathbb{R}$, $f(x, y, z) = z$ y $g(x, y, z) = x$ las funciones tercera y primera proyección canónica, respectivamente. Al recorrer los intervalos en el rango de las funciones f y g, seleccionamos puntos del conjunto de datos cuyas imágenes bajo las funciones están en ambos intervalos de éstas, para aplicar el algoritmo de clustering. Por lo que, en el caso de la esfera unitaria, tenemos tres posibilidades, como se muestra en las siguientes figuras:*

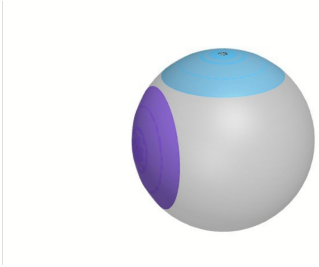


Figura 2.18: Los intervalos tienen intersección vacía y no se obtienen clusters.

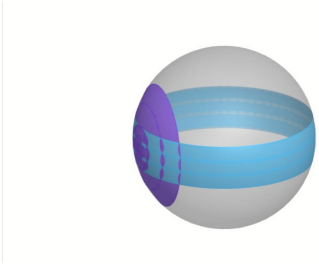


Figura 2.19: La intersección de los intervalos consiste de una componente conexa y obtenemos un cluster.

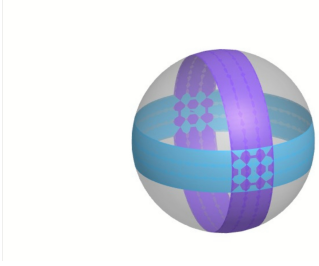


Figura 2.20: La intersección de los intervalos consiste de dos componentes conexas, y obtenemos dos clusters.

Después de obtener los clusters a partir de la cubierta, construimos simplejos como se mencionó en el algoritmo.

Es posible extender el algoritmo de Mapper a $\mathbb{R}^M$ como espacio de parametrización de una forma similar.

Como información adicional, cabe mencionar que los resultados del algoritmo de Mapper son altamente dependientes de la función o funciones usadas para particionar el conjunto de datos. En [2] se menciona que hay algunas funciones en particular que pueden brindar información geométrica de conjuntos de datos en general. Estas funciones recaen en la capacidad de calcular distancias entre puntos. Entonces, supongamos que se tiene una colección de N puntos en una nube de puntos X equipada con una función de distancia $d(x, y)$, que denota la distancia entre los puntos $x, y \in X$.

En primer lugar, consideremos una función de estimación de la densidad. Las funciones de estimación de la densidad se utilizan para estimar densidades de datos que no tienen comportamientos estadísticos paramétricos, es decir, no se agrupan en distribución normal, exponencial, etc. Para esto, sea $\varepsilon > 0$, podemos hacer una estimación de la densidad usando un núcleo Gaussiano como: $f_\varepsilon(x) = C_\varepsilon \sum_y e^{\frac{-d(x,y)^2}{\varepsilon}}$, donde $x, y \in X$ y C_ε es una constante tal que $\int f_\varepsilon(x)dx = 1$. En este caso, ε controla la suavidad en la estimación de la función de densidad en el conjunto de datos, estimaciones que usan valores grandes de ε corresponden a versiones más suaves que las que usan valores pequeños de este parámetro. Existen más métodos que recaen en el cálculo de distancias entre elementos de la nube de puntos y que producen funciones que brindan información geométrica del conjunto. Para más información recomendamos consultar [12].

Ejemplo 18 *Consideremos* $X = \{0, 1, i, -1\}$*, y sea* ε*>0, entonces, como* $f_\varepsilon(x) = C_\varepsilon \sum_{y\in X} e^{\frac{-d(x,y)^2}{\varepsilon}}$*, se tiene que* $f_\varepsilon(0) = C_\varepsilon(e^{\frac{-d(0,0)^2}{\varepsilon}} + e^{\frac{-d(0,-1)^2}{\varepsilon}} + e^{\frac{-d(0,1)^2}{\varepsilon}} + e^{\frac{-d(0,i)^2}{\varepsilon}}) = C_\varepsilon(1 + 3e^{-\frac{1}{\varepsilon}})$.

Análogamente,

$f_\varepsilon(1) = C_\varepsilon(1 + e^{-\frac{1}{\varepsilon}} + e^{-\frac{2}{\varepsilon}} + e^{-\frac{4}{\varepsilon}})$

$f_\varepsilon(-1) = C_\varepsilon(1 + e^{-\frac{1}{\varepsilon}} + e^{-\frac{2}{\varepsilon}} + e^{-\frac{4}{\varepsilon}})$

$f_\varepsilon(i) = C_\varepsilon(1 + e^{-\frac{1}{\varepsilon}} + 2e^{-\frac{2}{\varepsilon}})$

Por lo que, como $1 = \int f_\varepsilon(x)dx = \sum_{x\in X} f_\varepsilon(x) = C_\varepsilon(4 + 6e^{-\frac{1}{\varepsilon}} + 4e^{-\frac{2}{\varepsilon}} + 2e^{-\frac{4}{\varepsilon}})$*, se tiene que* $C_\varepsilon = 1/(4 + 6e^{-\frac{1}{\varepsilon}} + 4e^{-\frac{2}{\varepsilon}} + 2e^{-\frac{4}{\varepsilon}})$.

En segundo lugar, existe otra familia de funciones de excentricidad que también brindan información de la geometría del conjunto de datos. La idea intuitiva se basa en identificar puntos que se encuentran lejos del centro, sin identificar en realidad un punto central. Dada p con $1 \leq p < +\infty$, definimos $E_p(x) = (\frac{\sum_{y\in X} d(x,y)^p}{N})^{\frac{1}{p}}$, donde $x, y \in X$. Es posible extender la definición a $p = +\infty$ definiendo $E_\infty(x) = max_{x'\in X}\{d(x, x')\}$. En el caso de la función de distribución Gaussiana, esta función está correlacionada negativamente con la densidad. Y por lo general, la función tiende a tomar valores mayores en puntos alejados del *centro*.

Ejemplo 19 *Sea nuevamente* $X = \{0, 1, i, -1\}$*, y sea* $1 \leq p < +\infty$*, entonces* $E_p(x) = (\frac{\sum_{y\in X} d(x,y)^p}{N})^{\frac{1}{p}}$*, donde:*

$E_p(0) = (\frac{\sum_{y\in X} d(0,y)^p}{4})^{\frac{1}{p}} = (\frac{3}{4})^{\frac{1}{p}}$
$E_p(1) = (\frac{\sum_{y\in X} d(1,y)^p}{4})^{\frac{1}{p}} = (\frac{1+2^p+\sqrt{2^p}}{4})^{\frac{1}{p}}$
$E_p(-1) = (\frac{\sum_{y\in X} d(-1,y)^p}{4})^{\frac{1}{p}} = (\frac{1+2^p+\sqrt{2^p}}{4})^{\frac{1}{p}}$
$E_p(i) = (\frac{\sum_{y\in X} d(i,y)^p}{4})^{\frac{1}{p}} = (\frac{1+2\sqrt{2^p}}{4})^{\frac{1}{p}}$

Por último, la familia de funciones de Laplacianos de gráfica, que se origina de considerar un operador Laplaciano en una gráfica definida. El operador Laplaciano se relaciona con problemas de minimización de ciertas magnitudes sobre un dominio dado. El operador Laplaciano de una gráfica definida se origina como sigue: el conjunto de vértices de la gráfica es el conjunto de puntos de la nube de puntos X y el peso de la arista ponderada entre puntos $x, y \in X$ está dado por $w(x, y) = k(d(x, y))$, donde d denota la distancia entre puntos de la nube de puntos X y k es a grandes rasgos un *núcleo para suavizar* como un núcleo o kernel Gaussiano. Finalmente, se calcula la matriz (normalizada) Laplaciana de la gráfica con: $L(x, y) = \frac{w(x,y)}{\sqrt{\sum_z w(x,z)}\sqrt{\sum_z w(y,z)}}$. De forma que los eigenvectores de la matriz Laplaciana normalizada de la gráfica proporciona un conjunto de vectores ortogonales que brindan información geométrica importante del conjunto. Para mayor profundidad en el tema, se puede consultar [13].

Ejemplo 20 *Nuevamente consideremos* $X = \{0, 1, i, -1\}$*, y de forma similar al ejemplo 14 consideremos* $k(x) = \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}}$*, entonces* $w(x, y) = \frac{1}{\sqrt{2\pi}} e^{\frac{-d(x,y)^2}{2}}$*.*

De esta forma, $w(0,0) = w(1,1) = w(-1,-1) = w(i,i) = \frac{1}{\sqrt{2\pi}}$
$w(0,1) = w(1,0) = w(0,i) = w(i,0) = w(0,-1) = w(-1,0) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}}$
$w(1,-1) = w(-1,1) = \frac{1}{\sqrt{2\pi}} e^{-2}$
$w(i,1) = w(1,i) = w(i,-1) = w(-1,i) = \frac{1}{\sqrt{2\pi}} e^{-1}$

Por lo que las entradas de la matriz Laplaciana quedarán como sigue:
$L(0,0) = \frac{1/\sqrt{2\pi}}{1/\sqrt{2\pi}+3\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}}} = \frac{1}{1+3e^{-\frac{1}{2}}}$
$L(1,1) = L(-1,-1) = \frac{1}{1+e^{-\frac{1}{2}}+e^{-2}+e^{-1}}$
$L(i,i) = \frac{1}{1+e^{-\frac{1}{2}}+2e^{-1}}$
$L(0,1) = L(1,0) = L(0,-1) = L(-1,0) = \frac{e^{-\frac{1}{2}}}{\sqrt{1+3e^{-\frac{1}{2}}}\sqrt{1+e^{-\frac{1}{2}}+e^{-2}+e^{-1}}}$
$L(i,0) = L(0,i) = \frac{e^{-\frac{1}{2}}}{\sqrt{1+3e^{-\frac{1}{2}}}\sqrt{1+e^{-\frac{1}{2}}+2e^{-1}}}$
$L(i,-1) = L(-1,i) = L(i,1) = L(1,i) = \frac{e^{-1}}{\sqrt{1+e^{-\frac{1}{2}}+e^{-2}+e^{-1}}\sqrt{1+e^{-\frac{1}{2}}+2e^{-1}}}$
$L(1,-1) = L(-1,1) = \frac{e^{-2}}{1+e^{-\frac{1}{2}}+e^{-2}+e^{-1}}$

3 El método Ball Mapper

En este capítulo abordamos el tema de la gráfica Ball Mapper, basándonos principalmente en [1] y [2], y retomando varios conceptos del capítulo 2. Abordaremos primero la motivación y construcción de la gráfica Ball Mapper, junto con diversos algoritmos usados para su construcción. Finalmente revisaremos qué relación existe entre la gráfica Mapper y la gráfica Ball Mapper, basándonos en [1] y [2].

3.1. La gráfica Ball Mapper

Para la construcción de la gráfica Mapper tomamos en cuenta cuatro parámetros: el conjunto de datos X, la función $f : X \to \mathbb{R}$ (o Z, donde Z es un espacio de parametrización), la cubierta U en $\mathbb{R}$ o Z, y el algoritmo de clustering.

A continuación mostramos una forma alternativa de obtener cubiertas con elementos que se traslapan. Para esto, dada $\varepsilon > 0$ se busca encontrar una colección de puntos $C \subset X$, que la colección de bolas $\{B(c, \varepsilon)\}_{c \in C}$ sea una cubierta de X, es decir, que $X \subset \bigcup_{c \in C} B(c, \varepsilon)$. Para cada $x \in X$ denotaremos por $B(X, \varepsilon)[x]$ al conjunto de $c \in C$ tales que x está en $B(c, \varepsilon)$. Denominaremos vector cubriente al conjunto $\{B(X, \varepsilon)[x]\}_{x \in X}$ y lo denotaremos por $B(X, \varepsilon)$.

Ejemplo 21 *Consideremos ε=1 y sean las raíces del polinomio de Jones $-q^8+q^5+q^3$ del nudo tórico $T_{(4,3)}$ la nube de puntos $X = \{x_1, x_2, x_3, x_4, x_5, x_6\}$, donde:*

$x_1 = 0$
$x_2 = 1.19386$
$x_3 = -0.751519 - 0.784616i$
$x_4 = -0.751519 + 0.784616i$
$x_5 = 0.15459 - 0.828074i$
$x_6 = 0.15459 + 0.828074i$

Si el algoritmo recorre la nube de puntos en el orden mencionado, para $\varepsilon = 1$*, se tiene que:*

- $\{x_1, x_5, x_6\} \subset B(x_1, 1)$
- $B(x_2, 1) \cap X = x_2$
- $\{x_3, x_5\} \subset B(x_3, 1)$
- $\{x_4, x_6\} \subset B(x_4, 1)$

como se muestra en la figura 3.1.

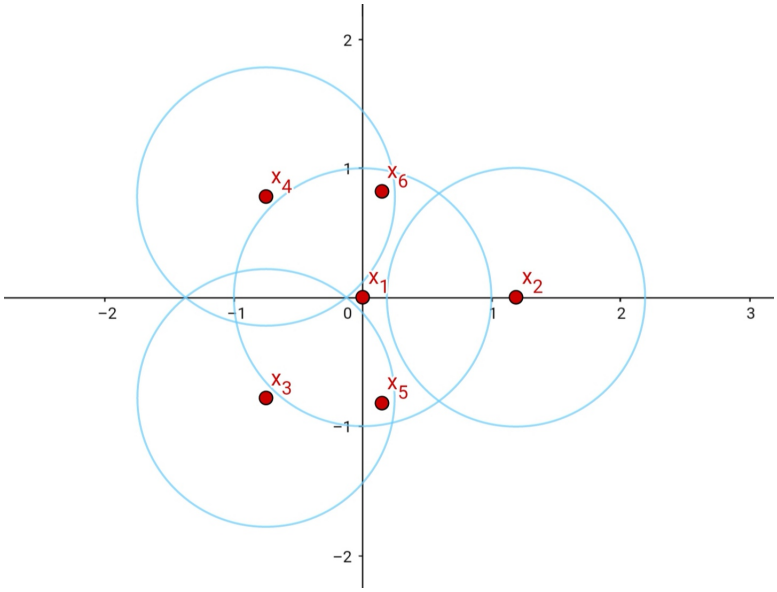


Figura 3.1: X y su ε-red.

De forma que el vector cubriente será el conjunto $B(X, \varepsilon) = \{B(X, \varepsilon)[x_1], B(X, \varepsilon)[x_2], B(X, \varepsilon)[x_3], B(X, \varepsilon)[x_4], B(X, \varepsilon)[x_5], B(X, \varepsilon)[x_6]\} = \{\{x_1\}, \{x_2\}, \{x_3\}, \{x_4\}, \{x_1, x_3\}, \{x_1, x_4\}\}$.

x	$B(X, \varepsilon)[x]$
x_1	x_1
x_2	x_2
x_3	x_3
x_4	x_4
x_5	x_1, x_3
x_6	x_1, x_4

Asimismo, el conjunto de centros de bolas será $C = \{x_1, x_2, x_3, x_4\} \subseteq X$*. Tal que se cumple que las bolas de radio* $\varepsilon = 1$ *con centro en los puntos de* C *conforman una cubierta abierta de* X *ya que* $X \subseteq B(C) = \bigcup_{x \in C} B(x, \varepsilon)$.

Existen varias formas de seleccionar el conjunto C de los centros de las bolas a partir de X: construyendo una $\varepsilon - red$ en X, donde ε es un parámetro; usando clusters con k-medias, o usando un Modelo de Mezcla Gaussiano (GMM), donde el parámetro del método es el número de clusters, y posteriormente encontrando una distancia ε de un punto más lejano del conjunto datos, a los centros de los clusters seleccionados.

A continuación presentamos una definición de ε-red obtenida en [14]:

Definición 3.1.1 *Sea M un conjunto de un espacio métrico R y $\varepsilon > 0$. Se dice que el conjunto A de R es una $\varepsilon - red$ de M cuando para todo punto $x \in M$ existe al menos un punto $a \in A$ tal que $d(x, a) \leq \varepsilon$.*

Ejemplo. Los puntos de coordenadas enteras del plano forman una $1/\sqrt{2}$ - red del plano.

En este caso, obtenemos el conjunto C de centros de las bolas construyendo una $\varepsilon - red$ de las siguientes formas.

A continuación introducimos dos algoritmos presentados en [1] para construir una ε-red:

Algorithm 1. Greedy ε–net
Input: Point cloud X, $\varepsilon > 0$
Mark all points in X as not covered.
Create initially empty cover vector B(X,ε).
repeat
  Pick a first point p $\in$ X that is not covered.
  For every point in x $\in$ B(p,ε) $\cap$ X, add p to B(X,ε)[x].
until All elements of X are covered.
return B(X,ε).

Algorithm 2. Max–min ε–net
Input: Point cloud X, $\varepsilon > 0$
Pick arbitrary p $\in$ X. C = {p}
repeat
  Find point p $\in$ X $\setminus$ C that is farthest away from C
  (if there is more than one, pick any).
  d = dist(p,C)
  C = C $\cup$ {p}.
until d $\leq \varepsilon$
Create initially empty cover vector B(X, ε).
for Every point p $\in$ C **do**
  For every x $\in$ B(p,ε) $\cap$ X, add p to B(X,ε)[x].
Return B(X,ε).

El algoritmo 1 es bastante sencillo de usar e implementar y presenta una versión a buena escala, mientras que una ventaja del algoritmo 2 es que puede modificarse para ser detenido después de un número fijo de iteraciones de forma que proporcione una colección de puntos de forma distribuida.

Asimismo, presentamos una versión desarrollada en Python de los mismos en el Anexo B.

Dado un vector cubriente $B(X, \varepsilon)$, proporcionado por alguno de estos algoritmos, es posible construir un nervio de la cubierta $N(B(X, \varepsilon))$ donde los centros de las bolas $c_1, ..., c_n$ corresponden a los vértices de $N(B(X, \varepsilon))$ y un simplejo $[c_{i0}, ..., c_{ik}]$ es formado si existe un $x \in X$ tal que $c_{i0}, ..., c_{ik}$ lo cubren. Es decir, x es el punto que pertenece a la intersección $B(c_{i0}, \varepsilon) \cap ... \cap B(c_{ik}, \varepsilon)$.

Definición 3.1.2 *La gráfica o el complejo obtenido se llamará la gráfica Ball Mapper.*

Ejemplo 22 *Considerando la nube de puntos X, el parámetro $\varepsilon = 1$, el vector cubriente $B(X, \varepsilon)$ y el conjunto C del ejemplo 21, como $B(x_1, 1) \cap B(x_4, 1) \cap X \neq \varnothing$, $B(x_1, 1) \cap B(x_3, 1) \cap X \neq \varnothing$, y $B(x_2, 1) \cap X = x_2$, los vértices de la gráfica Ball Mapper corresponden precisamente al conjunto C de centros de las bolas, y los aristas x_1x_3 y x_1x_4 pertenecen a la gráfica. Por lo tanto, la gráfica Ball Mapper de X será como sigue:*

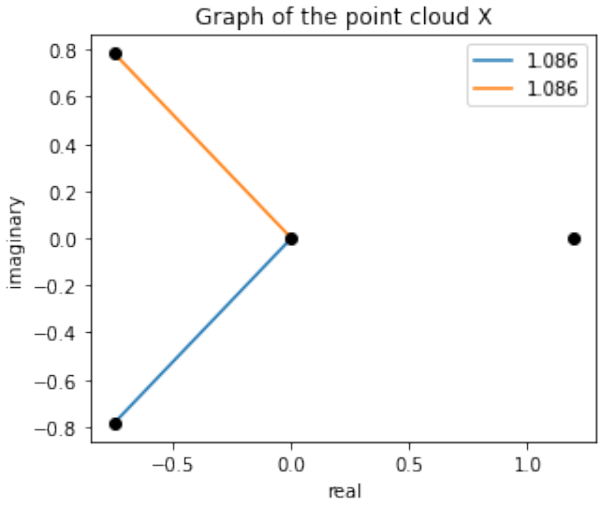


Figura 3.2: Gráfica Ball Mapper de X.

A continuación presentamos un algoritmo para la construcción de la gráfica Ball Mapper, como se muestra en [1]:

Algorithm 3. Construction of Ball Mapper graph
Input: Cover vector B(X,ε) from Algorithm 1 or 2.

```
V = cover elements in B(X,ε), E = ∅,
 for Every point p ∈ X do
    For every pair of cover elements c1 ≠ c2 in B(X,ε)[p],
    add a (weighted) edge between vertices corresponding
    to the cover elements c1 and c2.
    Formally, E = E ∪ {c1, c2}
Return G = (V,E)
```

La versión implementada en Python de este algoritmo se encuentra en el Anexo B.

Definición 3.1.3 *Sea K un complejo simplicial, definimos la filtración de K como una sucesión de subcomplejos $K_1 \leq K_2 \leq ... \leq K_m = K$.*

La definición 3.1.2 induce naturalmente una filtración de nervios de una cubierta, que ocuparemos posteriormente en el capítulo 4.

Cabe mencionar que los algoritmos de Mapper y Ball Mapper no están restringidos a nubes de puntos muestreados a partir de variedades.

3.2. Relación entre la gráfica Mapper y la gráfica Ball Mapper

Una diferencia entre las gráficas Mapper y Ball Mapper es que la gráfica Mapper puede tener aristas múltiples, según se defina su construcción o dependiendo el algoritmo de clustering mientras que la gráfica Ball Mapper no las puede tener, por ser un complejo simplicial.

Por otro lado, un factor fundamental en el método de Mapper es la función $f : X \to \mathbb{R}$, si ésta no es continua, entonces no hay correspondencia entre puntos cercanos en el dominio y el rango de f. De esta forma, las gráficas Mapper y Ball Mapper pueden ser muy diferentes entre sí y no se podría obtener mucha información útil en este caso.

Supongamos entonces que la función $f : X \to \mathbb{R}$ es continua. Entonces para todo $x \in X$ y para toda $\varepsilon > 0$ existe $\delta > 0$ tal que $f(B(x, \delta)) \subset B(f(x), \varepsilon)$. Es decir, los puntos cubiertos por cada bola en la gráfica Ball Mapper deben tener valores similares bajo la función f. Y en este caso, sería posible encontrar cierta correspondencia entre las gráficas Mapper y Ball Mapper.

Cabe mencionar que la conectividad de la gráfica Ball Mapper puede ser muy diferente si el crecimiento de la función es muy grande en comparación con la densidad del conjunto. En dicho caso, pueden existir regiones disconexas en la gráfica Mapper que sean conexas en la gráfica Ball Mapper, ya que el algoritmo de Ball Mapper sólo depende de la proximidad de los puntos en X y no de la función f como parámetro.

Ejemplo 23 *Consideremos la nube de puntos* $X = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$, *la función* $f(x) = 1/(10x)$, *la cubierta* $\mathcal{U} = \{U_1, U_2, U_3, U_4, U_5\}$ *donde* $U_1 = (0.92, 1.08)$, $U_2 = (0.42, 0.58)$, $U_3 = (0.25333, 0.41333)$, $U_4 = (0.12, 0.28)$, *y* $U_5 = (0.03111, 0.19111)$, *y* ε=*0.16.*

Entonces se tiene lo siguiente:

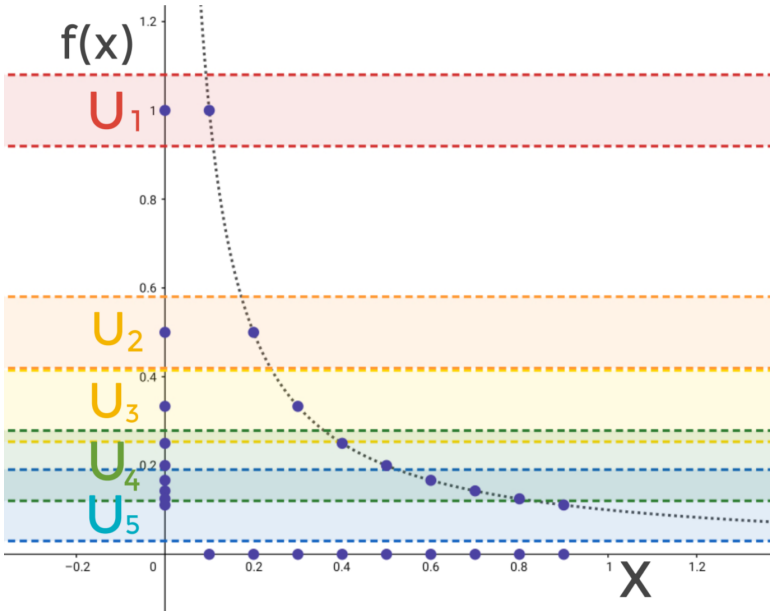


Figura 3.3: Nube de puntos X, imagen $f[X]$, y cubierta abierta $\mathcal{U}$.

De forma que la gráfica Mapper de X *será como sigue:*

Figura 3.4: Gráfica Mapper de X con respecto a $\mathcal{U}$ y f.

Y la gráfica Ball Mapper de X será como se muestra en la siguiente figura:

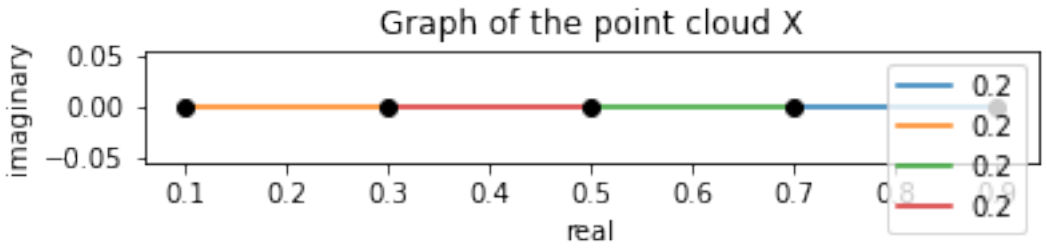


Figura 3.5: Gráfica Ball Mapper de X con parámetro ε.

De esta forma, vemos que puntos cercanos en X cuyos vértices correspondientes en la gráfica Ball Mapper están conectados, no lo están en la gráfica Mapper.

Para evitar este caso, asumamos que la función f es uniformemente continua, es decir, que $\forall \varepsilon > 0$ existe $\delta > 0$, tal que para toda $x, y \in X$, si $|x - y| < \delta$ entonces $|f(x) - f(y)| < \varepsilon$, más aún, supongamos que ε es el radio inicial de las bolas en la construcción de la gráfica Ball Mapper. Asimismo, asumamos que la cubierta en $\mathbb{R}$ en la gráfica Mapper consiste de intervalos de longitud 6ε que se intersectan en subintervalos de longitud ε y el algoritmo de clustering usado es un clustering de enlace simple con el parámetro δ, donde estas ε y δ son las de la continuidad uniforme.

Primero, sean $x, y \in X$ tales que en la gráfica Ball Mapper éstos son cubiertos por las bolas $B(c_x, \delta)$ y $B(c_y, \delta)$, respectivamente, de forma que $B(c_x, \delta) \cap B(c_y, \delta) \neq \varnothing$, en la gráfica Ball Mapper. Es decir, existe un punto en X que está cubierto por ambas bolas, y por lo tanto, existe un arista entre c_x y c_y en la gráfica Ball Mapper.

Proposición 3.1 *En este caso, tanto x como y pertenecen al mismo vértice o pertenecen a dos vértices conectados en la gráfica Mapper.*

Demostración. Como $x \in B(c_x, \delta)$, y $y \in B(c_y, \delta)$, $d(x, y) < 4\delta$, se tiene que $d(f(x), f(y)) < 4\varepsilon$. Entonces las imágenes de x y y bajo f estarán en uno, o a lo más dos elementos de la cubierta abierta del condominio de f, y en el caso de que sea en dos elementos U_1, U_2, estos tendrán intersección no vacía. Supongamos primero que sus imágenes se encuentran en el mismo conjunto abierto (o intervalo) de la cubierta. Entonces, como $x \in B(c_x, \delta)$, se tiene que $d(x, c_x) < \delta$. Análogamente, $d(y, c_y) < \delta$. Y como $B(c_x, \delta) \cap B(c_y, \delta) \neq \varnothing$, también se tiene que $d(c_x, c_y) < 2\delta$. Por lo tanto, el clustering de enlace simple con parámetro δ, colocará a x en el mismo cluster que y, y por tanto, estarán en el mismo vértice de la gráfica Mapper.

Ahora supongamos que las imágenes de x y y bajo f pertenecen a dos elementos distintos U_1 y U_2 de la cubierta abierta. Asimismo, U_1 y U_2 tienen intersección no vacía. Si x y y son enviados a $U_1 \cup U_2$, entonces regresamos al caso anterior y x y y terminarán en el mismo vértice de la gráfica Mapper, correspondiente a los conjuntos U_1 y U_2. Entonces, sin pérdida de generalidad, supongamos que $f(x) \in U_1 \setminus U_2$ y

$f(y) \in U_2 \setminus U_1$. Como las distancias $d(x, c_x)$, $d(c_x, c_y)$ y $d(c_y, y)$ están acotadas por δ, la imagen de al menos uno, c_x o c_y, estará en $U_1 \cap U_2$, y este vértice estará conectado a x y a y. Por lo tanto, los vértices de la gráfica Mapper a la que pertenecen x y y, estarán conectados. Lo que termina la prueba. □

Bajo las mismas hipótesis, podemos preguntarnos si dados dos vértices $x, y \in X$, que estén agrupados en uno o dos nodos de la gráfica Mapper correspondientes a U_1 y U_2, tal que existe un arista entre dichos nodos, también se encontrarán en nodos adyacentes de la gráfica Ball Mapper.

Supongamos primero que x y y pertenecen al mismo vértice U_1 de la gráfica Mapper, es decir que las imágenes de x y y bajo f pertenecen el abierto U_1. Entonces $d(f(x), f(y)) < 6\varepsilon$, es decir, tendrán valores cercanos. Como ambos, x y y, están en el mismo nodo de la gráfica Mapper, entonces existe un camino compuesto de los aristas de longitud a lo más δ en $f^{-1}(C)$ que unen x y y. A pesar de que la distancia entre ellos puede ser arbitrariamente larga, lo único que sabemos es que tanto x como y están conectados de acuerdo a un algoritmo de clustering de enlace simple con un parámetro δ en $f^{-1}(C)$.

El caso en el que x y y pertenecen a dos vértices adyacentes de la gráfica Mapper puede abordarse usando el mismo argumento de antes, y se llega a la misma conclusión. Por lo tanto, también en este caso, los vértices en la gráfica Ball Mapper podrían estar muy alejados, pero sí estarán conectados.

Cuando la gráfica Ball Mapper esté coloreada usando una función f, es posible determinar que x y y pertenecen a la misma componente conexa de la fibra de f.

En conclusión, se espera que puntos cercanos en la gráfica Ball Mapper sean cercanos en la gráfica Mapper. Mientras que puntos cercanos en la gráfica Mapper, estarán conectados en la gráfica Ball Mapper pero podrían estar arbitrariamente lejos entre sí. Consecuentemente, la gráfica Ball Mapper puede brindar información más adecuada de los datos; sin embargo, esta información podría ser más difícil de interpretar. Por tanto, el algoritmo Ball Mapper debe ser considerado como una técnica adicional que puede ser usada junto con el método Mapper para análisis de exploración de datos.

Ejemplo 24 *Retomando el ejemplo 23, consideremos nuevamente* $X = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$, ε*=0.16, pero* $f(x) = x$*. De esta forma, tomamos* $\varepsilon=\delta$*, y la cubierta* $\mathcal{U}$ *que consiste de intervalos de longitud* 6ε*=0.96 que se intersectan en subintervalos de longitud* ε*=0.16. Entonces se tiene que tanto la gráfica Mapper como la gráfica Ball Mapper serán como en la figura 3.5, y se cumple que puntos cercanos en la gráfica Ball Mapper puede son cercanos en la gráfica Mapper y puntos cercanos en la gráfica Mapper, estarán conectados en la gráfica Ball Mapper*

4 Homología y homología persistente en las Gráficas Mapper

En este capítulo abordamos primero algunas construcciones de complejos simpliciales. Principalmente complejos de Vietoris-Rips, basándonos en [15] y [7]. Posteriormente introducimos el tema de homología persistente y su relación con la gráfica Ball Mapper, con base en [7] y [1].

4.1. Complejos Vietoris-Rips

Definición 4.1.1 *Sea M un espacio métrico con métrica d y $X \subset M$ un conjunto finito. Entonces el complejo Vietoris-Rips de X, asociado al parámetro ε, denotando por $VR(X, \varepsilon)$, es el complejo simplicial cuyo conjunto de vértices es X y en el cual $\{x_0, x_1, ..., x_k\}$ genera un k-simplejo si y sólo si $d(x_i, x_j) \leq \varepsilon$ para $0 \leq i, j \leq k$.*

Ejemplo 25 *Consideremos la siguiente nube de puntos X en el plano, como se muestra en la figura 4.1. Entonces los complejos $VR(X, \varepsilon_i)$ de X con parámetros $\varepsilon_1 < \varepsilon_2 < \varepsilon_3$ son como sigue:*

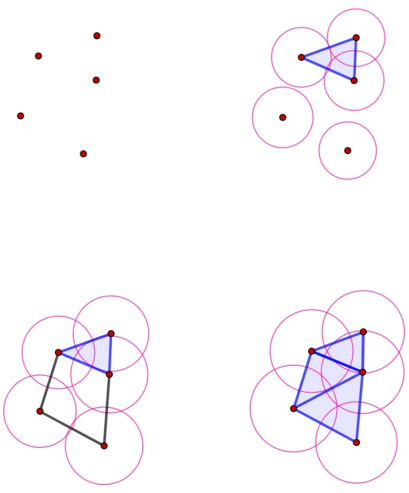


Figura 4.1: X y Complejos $VR(X, \varepsilon)$.

Como observación, tenemos que si r es menor al mínimo de las distancias entre dos puntos de X, entonces el complejo $VR(X, r)$ es un conjunto discreto, sin aristas o simplejos de dimensiones mayores. Asimismo, observemos que si $r_1 \leq r_2$, entonces $VR(X, r_1) \subseteq VR(X, r_2)$.

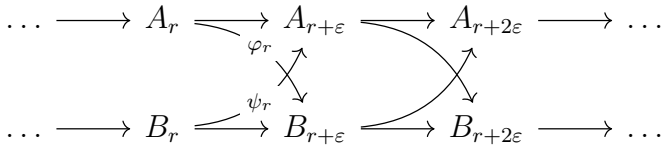
Definición 4.1.2 *Sea M un espacio métrico y sea $X \subset M$ un conjunto finito. La filtración de Vietoris-Rips de X es una colección de complejos simpliciales $\{VR(X, r)\}_{r\geq 0}$ con inclusiones $i_{r_1,r_2} : VR(X, r_1) \subseteq VR(X, r_2)$ para toda $r_1 \leq r_2$.*

Una filtración proporciona una colección de todos los complejos VR en X. Mientras que un complejo VR depende de la elección de r, la filtración no. Estas filtraciones tendrán un papel fundamental después en la definición de homología persistente.

Supongamos que tomamos dos muestras X, Y de un mismo espacio métrico, y que a partir de cada una construimos una filtración de complejos VR. ¿Qué tan diferentes serán estas filtraciones? Para responder esta pregunta, a continuación introducimos el concepto de epsilon-intercalamiento.

Definición 4.1.3 *Sea ε>0. Las filtraciones $\{A_r\}_{r\geq 0}$ y $\{B_r\}_{r\geq 0}$ obtenidas mediante una construcción VR están ε-intercaladas si existen funciones entre complejos $\phi_r : A_r \to B_{r+\varepsilon}$ y $\psi_r : B_r \to A_{r+\varepsilon}$ tales que $\phi_{r+\varepsilon} \circ \psi_r : B_r \to B_{r+2\varepsilon}$ y $\psi_{r+\varepsilon} \circ \phi_r : A_r \to A_{r+2\varepsilon}$ son iguales a las inclusiones correspondientes.*

Es posible visualizar estas funciones mediante el siguiente diagrama conmutativo:



Definición 4.1.4 *Dadas dos filtraciones, definimos su distancia de intercalamiento como el ínfimo de todos los valores ε>0 tales que las filtraciones están ε-intercaladas.*

Ejemplo 26 *Sea $X = \{0, 1\} \subset \mathbb{R}$ y sea $Y = \{0.1, 1.2\} \subset \mathbb{R}$.*

El complejo VR(X,r) consiste de:

- *Dos puntos si r <1;*
- *Un arista si $r \geq 1$.*

El complejo VR(Y,r) consiste de:

- *Dos puntos si r <1.1;*
- *Un arista si $r \geq 1.1$.*

Estas filtraciones están 0.1-intercaladas.

Se tiene que para las filtraciones VR en colecciones finitas de puntos, la distancia de intercalamiento resulta ser una métrica en el conjunto de filtraciones. El concepto de intercalamiento juega un papel fundamental en la estabilidad de homología persistente.

Teorema 2 *(Estabilidad con respecto a muestras de un mismo espacio métrico). Sea $\varepsilon>0$ y sean $X = \{x_1, x_2, ..., x_k\}$ y $Y = \{y_1, y_2, ..., y_k\}$ tales que $d(x_i, y_i) \leq \varepsilon$, $\forall i$, es decir, los conjuntos X y Y consisten cada uno de k puntos, tales que las distancias correspondientes son a lo más ε. Entonces las filtraciones VR de X y Y están 2ε-intercaladas.*

Demostración. Se sigue de la desigualdad del triángulo: si $d(x_1, x_2) \leq r$ y $d(x_i, y_i) \leq \varepsilon$ entonces $d(y_1, y_2) \leq r + 2\varepsilon$. Si un subconjunto $\sigma \subset X$ tiene diámetro r, entonces el conjunto correspondiente $\tau \subset Y$, que resulta de tomar los puntos de Y con mismos índices que los puntos de σ, tiene diámetro a lo más $r + 2\varepsilon$. De forma que si σ es un complejo $VR(X, r)$, entonces τ es un complejo $VR(Y, r + 2\varepsilon)$. De este modo, tenemos que: las funciones $VR(X, r) \to VR(Y, r + 2\varepsilon)$ dadas por $x_i \mapsto y_i$ son funciones de complejos; de misma forma las funciones $VR(Y, r) \to VR(X, r+2\varepsilon)$ dadas por $y_i \mapsto x_i$ son funciones de complejos. Estas funciones conmutan con las inclusiones, por lo que podemos concluir que las filtraciones VR de X y Y están $2\varepsilon-intercaladas$.

En conclusión, si modificamos ligeramente el conjunto de puntos, la filtración resultante no presentará muchos cambios con respecto a la distancia de intercalamiento, es decir, la construcción de la filtración es estable.

4.2. Homología persistente

La homología persistente es un método para estudiar características topológicas de un espacio a diferentes resoluciones. Intuitivamente, la homología persistente sigue la evolución del número de componentes conexas y hoyos en complejos filtrados.

Matemáticamente hablando, la homología persistente es una extensión obvia de la homología: la funcionalidad de la homología se aplica a una filtración o sucesión de inclusiones. Sorprendentemente, la estructura resultante no es más difícil de calcular que la homología ordinaria.

Para obtener los grupos de homología persistente de un espacio, éste debe estar representado como un complejo simplicial K. Una función distancia en el espacio corresponderá a una filtración de K, es decir una sucesión anidada ascendente de contenciones de conjuntos.

Sea $f : K \to \mathbb{R}$ una función real sobre un complejo simplicial K y que es no decreciente en una sucesión ascendente de caras de K. Es decir $f(\sigma) \leq f(\tau)$ para σ una cara de τ en K. Entonces para todo $a \in \mathbb{R}$, el conjunto de nivel $K(a) = f^{-1}(-\infty, a]$ es un subcomplejo de K, y el orden de los valores de f en los simplejos de K induce un orden en los subcomplejos de nivel que definen una filtración:

$$\varnothing = K_0 \subseteq K_1 \subseteq \ldots \subseteq K_n = K$$

Si $0 \leq i \leq j \leq n$, la inclusión $Ki \hookrightarrow Kj$ induce un homomorfismo $f_p^{i,j} : H_p(K_i) \to H_p(K_j)$ en los grupos de homología simplicial de K para cada p.

Definición 4.2.1 *Los p-ésimos grupos de homología persistente son las imágenes de cada homomorfismo $f_p^{i,j}$, es decir $H_p^{i,j} = Im(f_p^{i,j})$. Respectivamente los p-ésimos números persistentes de Betti $\beta_p^{i,j}$ son los rangos de cada p-ésimo grupo de homología persistente.*

4.3. Grupos de homología en las gráficas Ball Mapper

La homología persistente es usada como herramienta para estudiar propiedades teóricas del algoritmo de Ball Mapper. Para esto, se toma un complejo simplicial K, donde es un conjunto de conjuntos tales que para todo $s \in K$ y todo $t \subset s$, $t \in K$.

A partir de una nube de puntos X, es posible obtener un complejo simplicial usando dicha construcción de complejos Vietoris-Rips (VR). Se colocan bolas de radio r con centro en todo $x \in X$. Un complejo VR es colocado en bolas con centros en puntos $x_1, ..., x_n \in X$ si $B(x_i, r) \cap B(x_j, r) \neq \varnothing$ para cada par de $i, j \in \{1, ..., n\}$. $VR(X, r)$ representa el complejo VR de radio r colocado en X. Entonces, como mencionamos en la sección anterior, se tiene que $VR(X, r) \subset VR(X, r')$ si $r < r'$, lo que induce una filtración de complejos VR, o sucesión anidada. La sucesión de inclusiones entre complejos induce una sucesión de isomorfismos en grupos de homología $f_{r,r'} : H_p(VR(X, r)) \to H_p(VR(X, r'))$. Los grupos de homología persistente son las imágenes de $f_{r,r'}$.

Conectividad en la gráfica Ball Mapper y su relación con Homología Persistente

Dada una nube de puntos X y una sucesión ascendente de radios $\varepsilon < \varepsilon_1 < ... < \varepsilon_n$, consideremos la gráfica Ball Mapper a multi-escala asociada a dicha sucesión. Sea C la colección de centros de bolas usados en la construcción de la gráfica. Entonces se tiene la siguiente sucesión de inclusiones:

$$\bigcup_{c \in C} B(c, \varepsilon) \subset \bigcup_{x \in X} B(x, \varepsilon) \subset \bigcup_{c \in C} B(c, 2\varepsilon)$$

La primera inclusión se sigue de que $C \subset X$. La segunda inclusión se sigue de que $C \subset X$ es una $\varepsilon - red$ en X, es decir, que para todo $x \in X$ existe $c \in C$ tal que $d(x, c) < \varepsilon$. Luego, sea $y \in \bigcup_{x \in X} B(x, \varepsilon)$. Entonces existe $x \in X$ tal que $d(x, y) < \varepsilon$. Además, para x, existe $c \in C$ tal que $d(x, c) < \varepsilon$. Entonces $d(y, c) \leq d(y, x) + d(x, c) < 2\varepsilon$. Consecuentemente, $y \in \bigcup_{c \in C} B(c, 2\varepsilon)$, y por lo tanto, se da la segunda inclusión.

De forma que, en la gráfica Ball Mapper a multi escala, si una característica es visible en los niveles ε y 2ε, también estará presente en la ε-vecindad de todos los puntos de X. Y usando el mismo argumento, la sucesión de inclusiones puede extenderse a:

$$\bigcup_{c\in C} B(c,\varepsilon) \subset \bigcup_{x\in X} B(x,\varepsilon) \subset \bigcup_{c\in C} B(c,2\varepsilon) \subset \bigcup_{x\in X} B(x,2\varepsilon) \subset \bigcup_{c\in C} B(c,3\varepsilon) \subset \dots$$

Entonces esta cadena induce la siguiente sucesión de grupos de homología:

$$\begin{array}{ccc} H\left(\bigcup_{x\in X} B(x,(i)\varepsilon)\right) & \longrightarrow & H\left(\bigcup_{x\in X} B(x,(i+1)\varepsilon)\right) \\ \uparrow & \nearrow & \uparrow \\ H\left(\bigcup_{c\in C} B(c,(i)\varepsilon)\right) & \longrightarrow & H\left(\bigcup_{c\in C} B(c,(i+1)\varepsilon)\right) \end{array}$$

De esta forma, al calcular el nervio de la cubierta U a dimensiones mayores, es posible aproximar los grupos de homología persistente del espacio X a partir del nervio obtenido de puntos en C. Esto muestra una relación entre el método Ball Mapper y la homología persistente, y que el procedimiento usado en la construcción de la gráfica Ball Mapper puede usarse para aproximar grupos de homología persistente.

Ejemplo 27 *Consideremos la siguiente nube de puntos X que se acumula en torno a la figura de una letra A:*

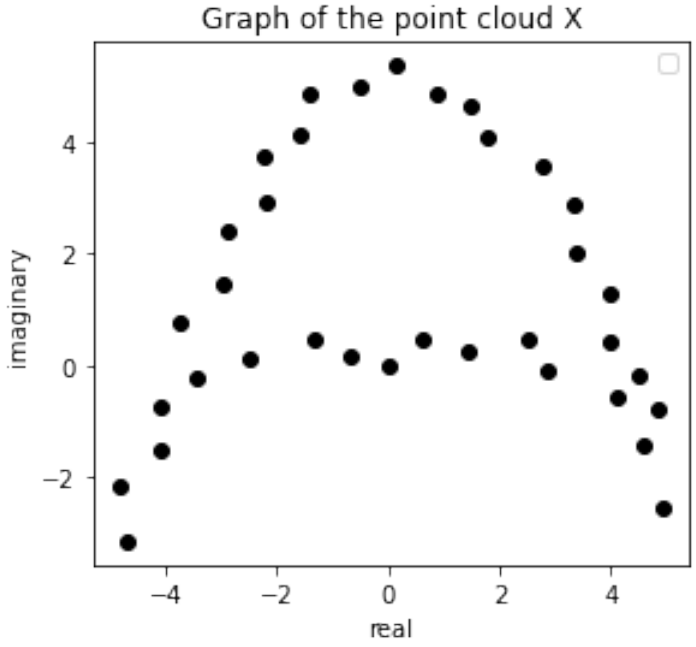


Figura 4.2: Nube de puntos X.

A continuación mostramos la gráfica Ball Mapper de X obtenida con los algoritmos mostrados en el Apéndice B para $\varepsilon_1 = 1$, $\varepsilon_2 = 2$, $\varepsilon_3 = 3$, $\varepsilon_4 = 4$.

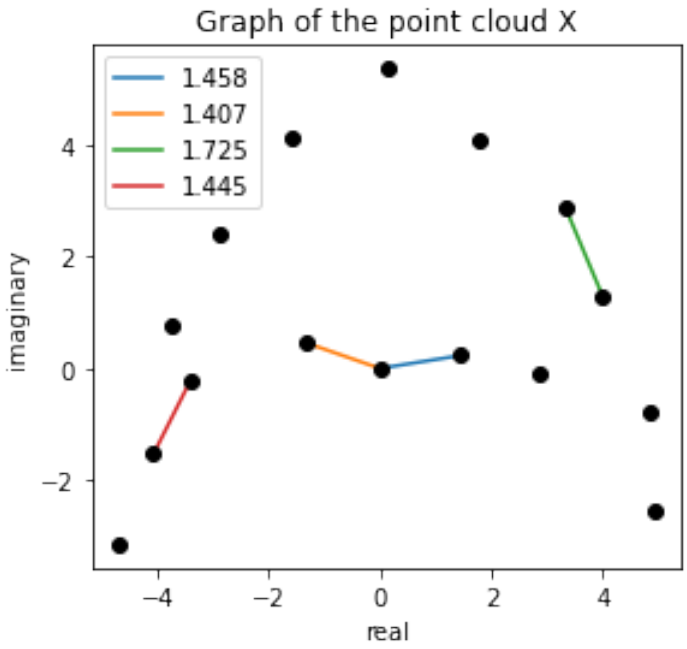


Figura 4.3: Gráfica Ball Mapper de X para ε=1.

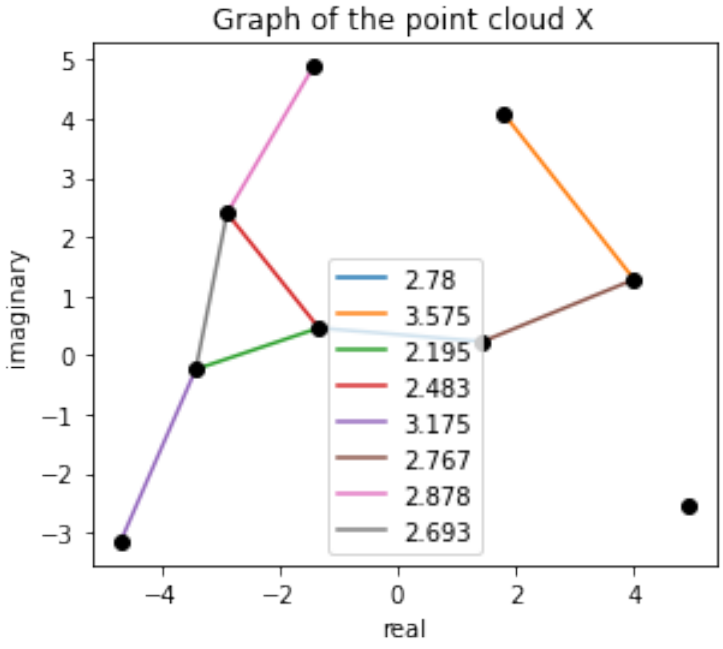


Figura 4.4: Gráfica Ball Mapper de X para ε=2.

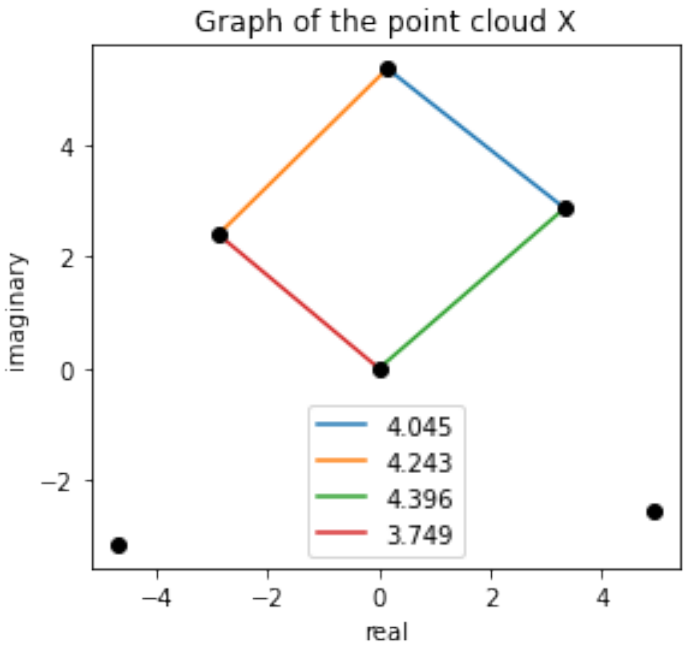


Figura 4.5: Gráfica Ball Mapper de X para ε=3.

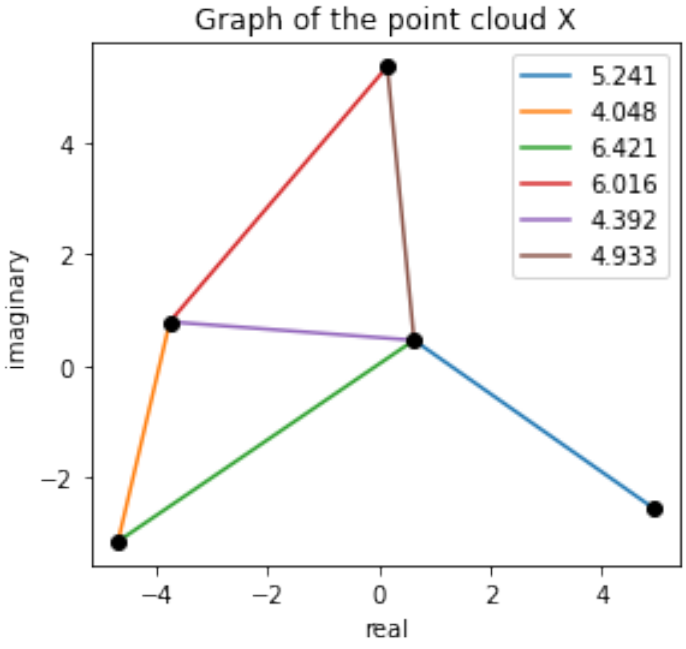


Figura 4.6: Gráfica Ball Mapper de X para ε=4.

Como información adicional, es posible analizar la sensibilidad de las gráficas Ball Mapper con respecto a ruido uniforme y ruido Hausdorff-acotado. El ruido Hausdorff-acotado está relacionado con el concepto de métrica Hausdorff:

Definición 4.3.1 *Sean X y Y dos nubes de puntos, la distancia Hausdorff entre X y Y está dada por $d_H(X,Y) = \max\{\sup_{x\in X} \inf_{y\in Y} d(x,y), \sup_{y\in Y} \inf_{x\in X} d(x,y)\}$, donde d(x,y) es la distancia entre los puntos x y y, usualmente es la distancia euclidiana.*

Esta definición implica que si $d_H(X,Y) < \delta$, entonces para todo $x \in X$ existe $y \in Y$ tal que está a lo más a distancia δ de x y viceversa. Dada esta observación, supongamos que C es una $\varepsilon - red$ en X, por lo que tenemos la siguiente sucesión natural de inclusiones para las nubes de puntos X y Y:

$$\bigcup_{c\in C} B(c,\varepsilon) \subset \bigcup_{x\in X} B(x,2\varepsilon) \subset \bigcup_{y\in Y} B(y,2\varepsilon+\delta) \subset \bigcup_{x\in X} B(x,2\varepsilon+2\delta) \subset \bigcup_{c\in C} B(c,3\varepsilon+2\delta)$$

De esta forma, si se tiene un valor δ en la métrica Hausdorff sobre el conjunto de ruido, las características que persisten entre el nivel ε y $3\varepsilon + \delta$ de la gráfica Ball Mapper son estables. Este tipo de ruido puede ser resultado de una reconstrucción o error numérico.

Existe otro tipo de ruido que es el ruido uniforme. Al ejecutar el Algoritmo 3 de la gráfica Ball Mapper para una nube de puntos con ruido uniforme, se obtiene el conjunto C de los centros de las bolas a lo largo de las regiones con señal y con ruido. Sin embargo, también podemos observar que la densidad de los puntos en las bolas que pertenecen a esta región con señal es superior a la densidad de los puntos en la región con ruido. Por lo que, es posible usar esta observación para eliminar las regiones de menor densidad.

A Algoritmo DBSCAN en Python

```
# A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise

# Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu

# dbscan: density based spatial clustering of applications with noise

import numpy as np
import math

UNCLASSIFIED = False
NOISE = None

#Euclidean distance
def dist(p,q):
    return math.sqrt(np.power(p-q,2).sum())

def eps_neighborhood(p,q,eps):
    return dist(p,q) < eps

def region_query(m, point_id, eps):
    n_points = m.shape[1]  #number of points in a vector
    seeds = []
    for i in range(0, n_points):
        #if not i == point_id:
        if eps_neighborhood(m[:,point_id], m[:,i], eps):
            seeds.append(i)
    return seeds

def _expand_cluster(m, classifications, point_id, cluster_id, eps, min_points):
    seeds = region_query(m, point_id, eps)
    if len(seeds) < min_points:
        classifications[point_id] = NOISE
        return False
    else:
        classifications[point_id] = cluster_id
        for seed_id in seeds:
            classifications[seed_id] = cluster_id

```

```
39          while len(seeds) > 0: # takes care of all connected/
    clustered points
40              current_point = seeds[0]
41              results = region_query(m, current_point, eps)
42              if len(results) >= min_points:
43                  for i in range(0, len(results)):
44                      result_point = results[i]
45                      if classifications[result_point] == UNCLASSIFIED
     or \
46                         classifications[result_point] == NOISE:
47                          if classifications[result_point] ==
    UNCLASSIFIED:
48                              seeds.append(result_point)
49                          classifications[result_point] = cluster_id
50              seeds = seeds[1:]
51          return True
52
53  def main_dbscan(m, eps, min_points):
54      cluster_id = 1
55      n_points = m.shape[1] #number of points in a vector
56      classifications = [UNCLASSIFIED] * n_points
57      for point_id in range(0, n_points):
58          point = m[:,point_id]
59          if classifications[point_id] == UNCLASSIFIED:
60              if _expand_cluster(m, classifications, point_id,
    cluster_id, eps, min_points):
61                  cluster_id = cluster_id + 1
62      return classifications
63
64  def test_main_dbscan():
65      m = np.matrix('1 1.2 0.8 3.7 3.9 3.6 10; 1.1 0.8 1 4 3.9 4.1 10'
    )
66      eps = 0.5
67      min_points = 2
68      output = [1, 1, 1, 2, 2, 2, None]
69      assert main_dbscan(m, eps, min_points) == output, "Oh no! This
    assertion failed!"
70      print("Everything's alright, this means the inputs: {}, {} and
    {} have the output: {}.".format(m, eps, min_points, output))
71
72  def test_main_dbscan2():
73      m = np.matrix('10 1 1 4 10 4 1 4 10; 10 1 1 4 10 4 1 4 10; 10 1
    1 4 10 4 1 4 10; 10 1 1 4 10 4 1 4 10')
74      eps = 0.5
75      min_points = 2
76      output = [1, 2, 2, 3, 1, 3, 2, 3, 1]
77      assert main_dbscan(m, eps, min_points) == output, "Oh no! This
    assertion failed!"
78      print("Everything's alright, this means the inputs: {}, {} and
    {} have the output: {}.".format(m, eps, min_points, output))
79
80  test_main_dbscan()
81  test_main_dbscan2()
82
```

```
m = np.matrix('3 7 3 5 11 3 3 5 11 5 11 7 11 5 7; 3 7 3 5 11 3 3 5
    11 5 11 7 11 5 7; 3 7 3 5 11 3 3 5 11 5 11 7 11 5 7')
eps = 0.1
min_points = 3
output = main_dbscan(m, eps, min_points)
print(output)
```

Listing A.1: Algoritmo DBSCAN en Python.

B Algoritmos Ball Mapper en Python

```
#!/usr/bin/env python3

#@title Algorithm 1: Greedy ε-net ##### { form-width: "25%" }

"""
Algorithm 1 Greedy ε-net
Input: Point cloud X, ε > 0
Mark all points in X as not covered.
Create initially empty cover vector B(X, ε).
repeat
Pick a first point p ∈ X that is not covered.
For every point in x ∈ B(p, ε) ∩ X, add p to B(X, ε)[x].
until All elements of X are covered.
return B(X, ε).
"""

import cmath
import math

##### input #####
X = [complex(0, 0), complex(1.19386, 0), complex(-0.751519,
    -0.784616),
      complex(-0.751519, 0.784616), complex(0.15459, -0.828074),
      complex(0.15459, 0.828074) ]
ε = 1
B_X = [] # B(X, ε)

##### functions #####
def distance(x, y): #Euclidean distance between points (complex
    numbers)
     return math.sqrt((x.real - y.real) ** 2  + (x.imag - y.imag) **
    2)

def B(p, ε): #considering the center (p) of open balls
     B = []
     B.append("B({}, {}) ∩ X =".format(p, ε)) # just for legibility
     for x in X:
         if distance(x, p) < ε:
              B.append(x)
     return B

##### main ####
```

```
#if __name__ == "__main__":

for p in X: # while
    B_X.append(B(p, ε)) #B(X, ε)

print("ε =", ε)
print("X =", X)
print("B(X, ε) =", B_X) #B(X, ε)
```

Listing B.1: Algoritmo 1: greedy ε-red

```

#!/usr/bin/env python3

#@title Algorithm 2: Max-min ε-net #Note: It seems to take double the time and also double the code XP { form-width: "25%" }

"""
Input: Point cloud X, ε > 0
Pick arbitrary p ∈ X. C = {p}
repeat
Find point p ∈ X \ C that is farthest away from C (if there is more than one, pick any).
d = dist(p, C)
C = C ∪ {p}.
until d ≤ ε
Create initially empty cover vector B(X, ε).
for Every point p ∈ C do
For every x ∈ B(p, ε) ∩ X, add p to B(X, ε)[x].
Return B(X, ε).
"""

import random
import cmath
import math

##### input #####
X = [complex(0, 0), complex(1.19386, 0), complex(-0.751519, -0.784616),
     complex(-0.751519, 0.784616), complex(0.15459, -0.828074),
     complex(0.15459, 0.828074) ]
ε = 1

C = []
C.append(random.choice(X))

X_minus_C = X.copy()
X_minus_C.remove(C[0])
inf = 100000 # infinity

##### functions #####
def distance(*args): #Euclidean distance between a point and a another point or a set of points (complex numbers)
    if len(args) == 2:
```

```
        if isinstance(args[0], complex): # x
            if isinstance(args[1], complex): # y
                return math.sqrt((args[0].real - args[1].real) ** 2
    + (args[0].imag - args[1].imag) ** 2)
            elif isinstance(args[1], list): # C
                min = inf
                for c in args[1]:
                    if isinstance(c, complex): # c
                        aux = distance(args[0], c)
                        if aux < min:
                            min = aux
                    else:
                        raise TypeError
                return min
            else:
                raise TypeError
        else:
            raise TypeError("The first entry must be a complex class
    type.")
    else:
        raise SyntaxError("Wrong number of arguments. \n distance
   takes exactly 2 arguments.")


def B(p, ε): #considering the center (p) of open balls
    B = []
    #B.append("B({}, {}) ∩ X =".format(p, ε)) # just for legibility
    #B.append(p) # the first element is the center p
    for x in X:
        if distance(x, p) < ε:
            B.append(x)
    return B

##### main ####
#if __name__ == "__main__":

while True: # do
    dist = 0
    p = None
    for q in X_minus_C:
        aux = distance(q, C)

        if dist < aux:
            dist = aux
            p = q

    if dist <= ε: # until
        break
    C.append(p)
    X_minus_C.remove(p)



B_X = [] # B(X, ε)
```

```
aux = []
for x in X:
    aux.append(x)
    B_X.append(aux)
    aux = []

#print("", B_X)

B_C = [] # B(C, ε)
B_p = []
for p in C:
    B_p = B(p, ε)
    B_C.append(B_p) #B(C, ε)
    #print("B_p", B_p)
    for x in B_p:
        B_X[X.index(x)].append(p)
    #print("B_X", B_X)

print("ε =", ε)
print("X =", X)
print("C =", C)
print("B(C, ε) =", B_C) #B(C, ε)

print("B_X = \n", end = '')
for x in B_X:
    print(x)
```

Listing B.2: Algoritmo 2: ε-red de máximos y mínimos

```

#!/usr/bin/env python3

#@title Algorithm 3: Construction of Ball Mapper graph { form-width: "25%" }

"""
Algorithm 3 Construction of Ball Mapper graph
Input: Cover vector B(X, ε) from Algorithm 1 or 2.
V = cover elements in B(X, ε), E = ∅,
for Every point p ∈ X do
For every pair of cover elements c1 different to c2 in B(X, ε)[p], add a (weighted) edge between vertices
corresponding to the cover elements c1 and c2. Formally, E = E ∪ {c1, c2}
Return G = (V, E)
"""

##### libraries #####
import numpy as np
import matplotlib.pyplot as plt
#import matplotlib as mpl

##### input #####
E = []
```

```
from a2_max_min_ε_net import * # Cover vector B(X, ε), Point cloud X, ε > 0
# if you cannot run this line, simply copy & paste Algorithm 2 instead.

##### main #####
#if __name__ == "__main__":

for B_p in B_X: # do
    # B(p, ε)
    print(B_p)
    length = len(B_p)
    for i in range(1, length):
        for j in range(i + 1, length):
            E.append(tuple([B_p[i], distance(B_p[i], B_p[j]), B_p[j]]))

E = set(E)

"""
print()

for edge in E:
    print("({}, {}) = {}".format(edge[0], edge[2], round(edge[1], 3)))
"""


##### Plotting graph #####
# Note that even in the OO-style, we use `.pyplot.figure` to create the figure.
fig, ax = plt.subplots()  # Create a figure and an axes.

ax.set_aspect('equal', adjustable='box')

print()

for edge in E:
    run = 1
    print("{} <---{}---> {}".format(edge[0], round(edge[1], 3), edge[2]))
    if edge[0].real < edge[2].real:
        y = np.linspace(edge[0].real, edge[2].real, 2)
    elif edge[0].real > edge[2].real:
        y = np.linspace(edge[2].real, edge[0].real, 2)
    else:
        run = 0
        print("run =", run)

    if run != 0:
        m = (edge[2].imag - edge[0].imag) / (edge[2].real - edge[0].real)
        b = edge[0].imag - edge[0].real * m
        ax.plot(y, m * y + b, label=str(round(edge[1], 3)))
```

```
    else:
        if edge[0].imag < edge[2].imag:
            plt.vlines(edge[0].real, edge[0].imag, edge[2].imag,
    label=str(round(edge[1], 3)))
        elif edge[0].imag > edge[2].imag:
            plt.vlines(edge[0].real, edge[2].imag, edge[0].imag,
    label=str(round(edge[1], 3)))
        else:
            print("WHAT?!!!!!!!")

for c in C:
    plt.plot(c.real, c.imag, 'ko')

ax.set_xlabel('real')  # Add an x-label to the axes.
ax.set_ylabel('imaginary')  # Add a y-label to the axes.
ax.set_title("Graph of the point cloud X")  # Add a title to the
    axes.
ax.legend()  # Add a legend.
# "Graph of the point cloud X"
```

Listing B.3: Algoritmo 3: Construcción de la gráfica Ball Mapper

C Gráfica Ball Mapper a multi-escala de nubes de puntos

En esta sección, de forma similar al ejemplo 21, consideramos en cada caso a la nube de puntos X como las raíces del polinomio de Jones de los nudos tóricos $T(4,3)$, $T(11,2)$, $T(21,2)$ y $T(23,2)$.

En cada caso, usando los códigos mostrados en el Anexo B, mostramos las gráficas Ball Mapper para su respectiva nube de puntos con distintos valores de ε, tales que $\varepsilon_1 < \varepsilon_2 < \varepsilon_3 < \varepsilon_4$.

Nudo T(4,3)
Polinomio de Jones: $-q^8 + q^5 + q^3$.

Raíces (Nube de puntos X):

- $x_1 = 0$
- $x_2 = 1.19386$
- $x_3 = -0.751519 - 0.784616i$
- $x_4 = -0.751519 + 0.784616i$
- $x_5 = 0.15459 - 0.828074i$
- $x_6 = 0.15459 + 0.828074i$

$\varepsilon_1 = 1/2$
$\varepsilon_2 = 1$
$\varepsilon_3 = 5/4$
$\varepsilon_4 = 3/2$

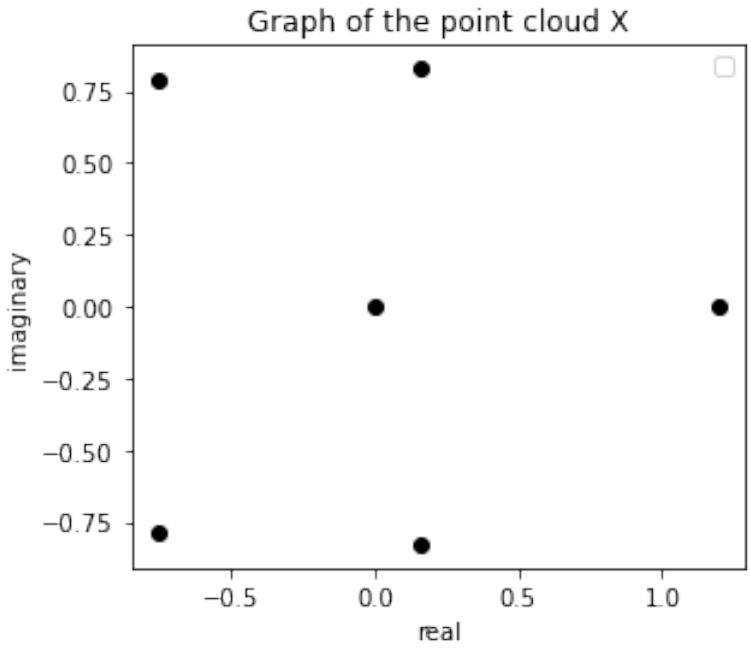


Figura C.1: Gráfica Ball Mapper de X. $\varepsilon = 1/2$.

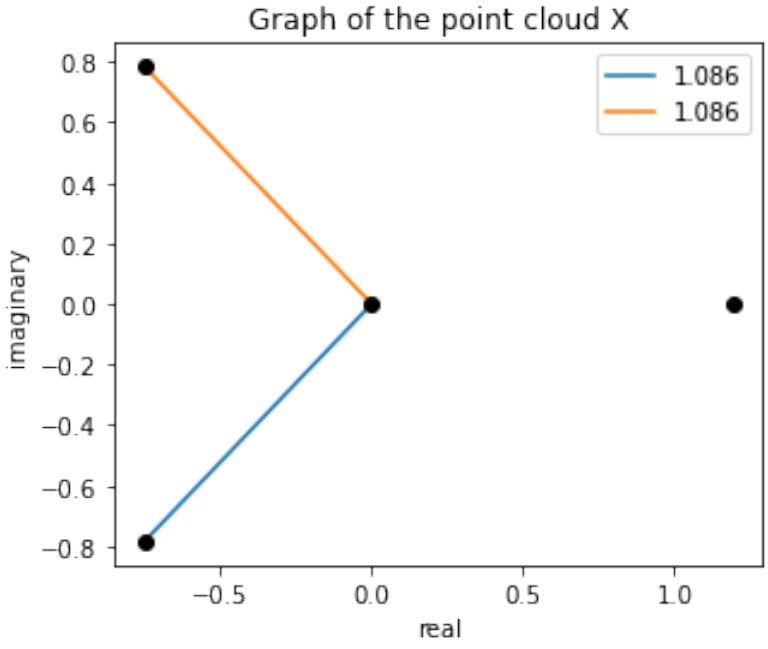


Figura C.2: Gráfica Ball Mapper de X. $\varepsilon = 1$.

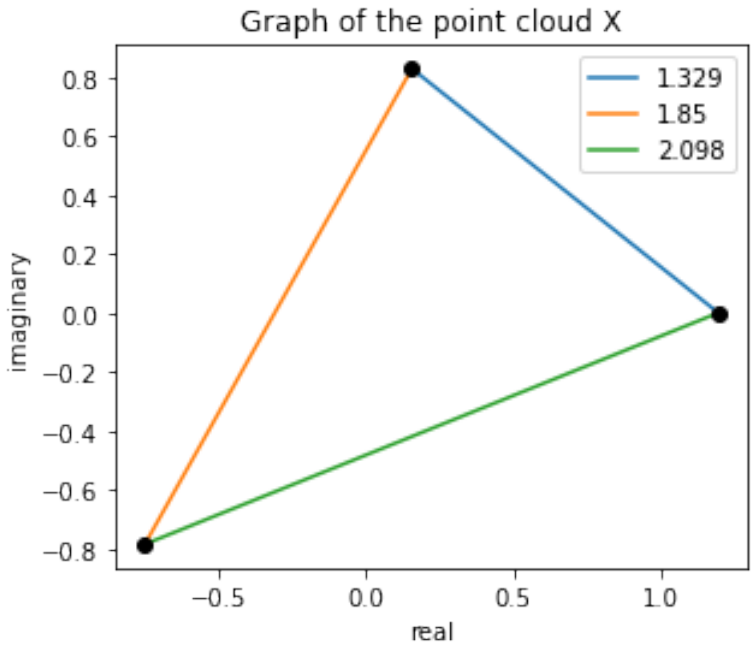


Figura C.3: Gráfica Ball Mapper de X. $\varepsilon = 5/4$.

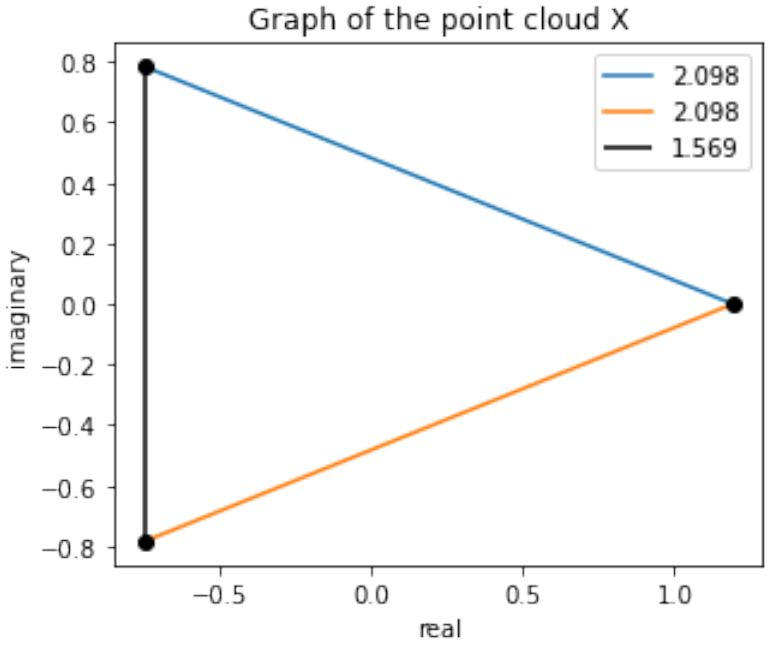


Figura C.4: Gráfica Ball Mapper de X. $\varepsilon = 3/2$.

Nudo T(11,2)
Polinomio de Jones: $-q^16 + q^15 - q^14 + q^13 - q^12 + q^11 - q^10 + q^9 - q^8 + q^7 + q^5$.

Raíces (Nube de puntos X):

- $x_1 = 0$
- $x_2 = 1.10537$
- $x_3 = -0.815795 - 0.518782i$
- $x_4 = -0.815795 + 0.518782i$
- $x_5 = -0.457714 - 0.746842i$
- $x_6 = -0.457714 + 0.746842i$
- $x_7 = -0.131572 - 0.96301i$
- $x_8 = -0.131572 + 0.96301i$
- $x_9 = 0.433998 - 0.96297i$
- $x_{10} = 0.433998 + 0.96297i$
- $x_{11} = 0.918399 - 0.594523i$
- $x_{12} = 0.918399 + 0.594523i$

$\varepsilon_1 = 1/2$
$\varepsilon_2 = 1/3$
$\varepsilon_3 = 1/4$
$\varepsilon_4 = 1$

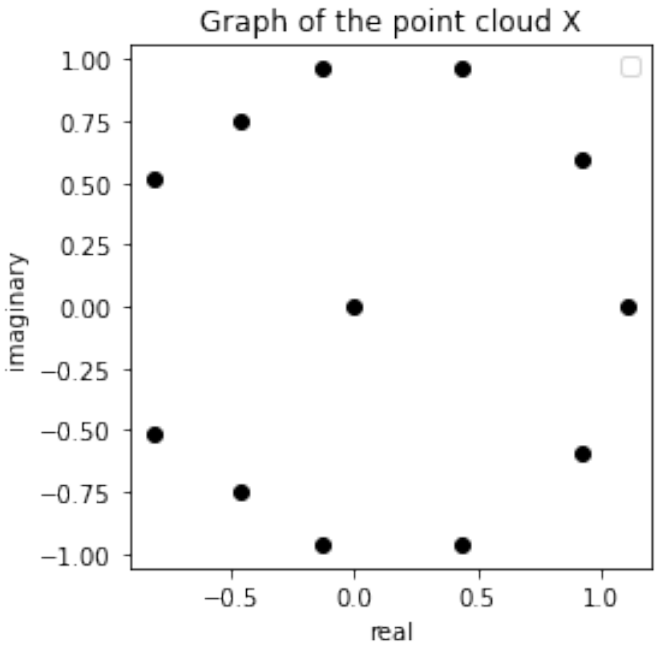


Figura C.5: Gráfica Ball Mapper de X. $\varepsilon = 1/4$.

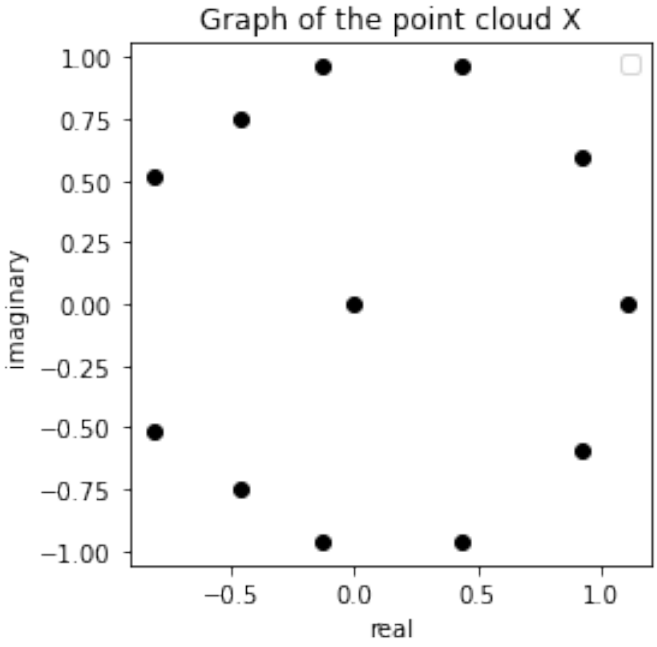


Figura C.6: Gráfica Ball Mapper de X. $\varepsilon = 1/3$.

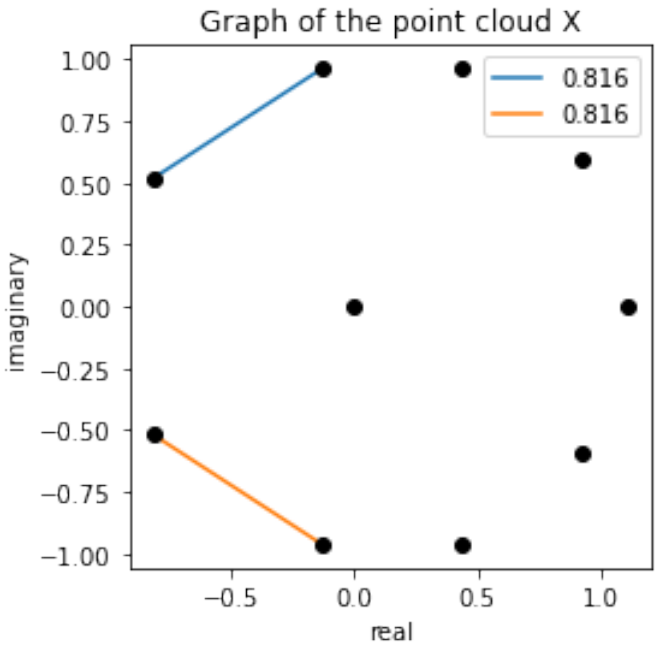


Figura C.7: Gráfica Ball Mapper de X. $\varepsilon = 1/2$.

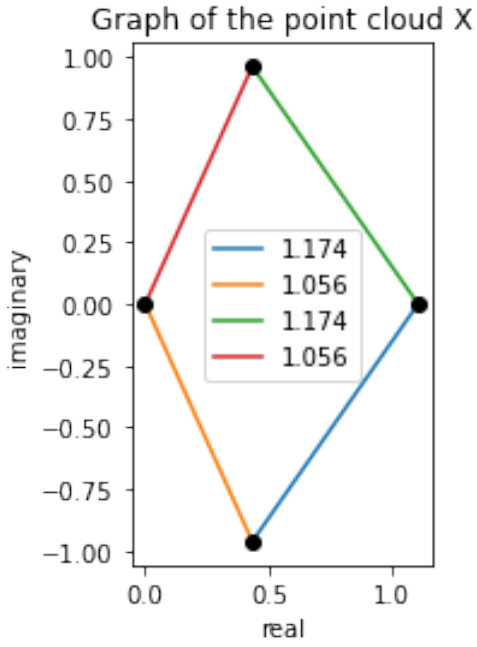


Figura C.8: Gráfica Ball Mapper de X. $\varepsilon = 1$.

Nudo T(21,2) Polinomio de Jones: $-q^{3}1+q^{3}0-q^{2}9+q^{2}8-q^{2}7+q^{2}6-q^{2}5+q^{2}4-q^{2}3+q^{2}2-q^{2}1+q^{2}0-q^{1}9+q^{1}8-q^{1}7+q^{1}6-q^{1}5+q^{1}4-q^{1}3+q^{1}2+q^{1}0$.

Raíces (Nube de puntos X):

- $x_1 = 0$
- $x_2 = 1.05375$
- $x_3 = 0.64695 + 0.81338i$
- $x_4 = 0.64695 - 0.81338i$
- $x_5 = -0.41925 + 0.81843i$
- $x_6 = -0.41925 - 0.81843i$
- $x_7 = -0.59889 + 0.72771i$
- $x_8 = -0.59889 - 0.72771i$
- $x_9 = 0.37371 + 0.95597i$
- $x_{10} = 0.37371 - 0.95597i$
- $x_{11} = -0.81067 + 0.55075i$
- $x_{12} = -0.81067 - 0.55075i$
- $x_{13} = -0.21234 + 0.94981i$
- $x_{14} = -0.21234 - 0.94981i$
- $x_{15} = 0.86499 + 0.59093i$
- $x_{16} = 0.86499 - 0.59093i$
- $x_{17} = 0.07468 + 1.00384i$
- $x_{18} = 0.07468 - 1.00384i$
- $x_{19} = -0.95139 + 0.29332i$
- $x_{20} = 0.95139 - 0.29332i$
- $x_{21} = 1.00534 + 0.31065i$
- $x_{22} = 1.00534 - 0.31065i$

$\varepsilon_1 = 1/4$
$\varepsilon_2 = 1/3$
$\varepsilon_3 = 1/2$
$\varepsilon_4 = 1$

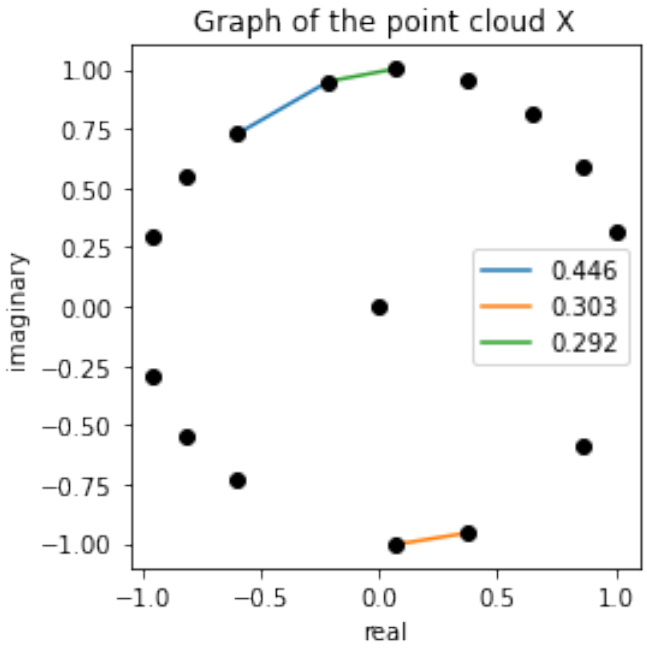


Figura C.9: Gráfica Ball Mapper de X. $\varepsilon = 1/4$.

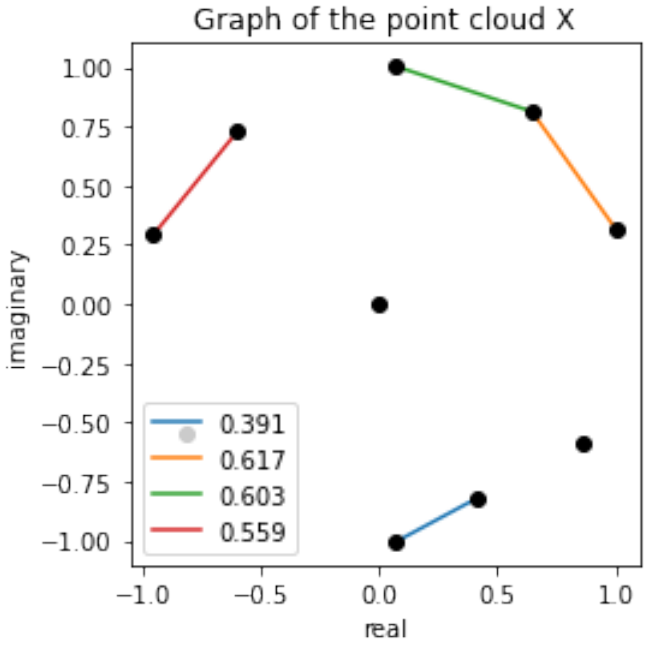


Figura C.10: Gráfica Ball Mapper de X. $\varepsilon = 1/3$.

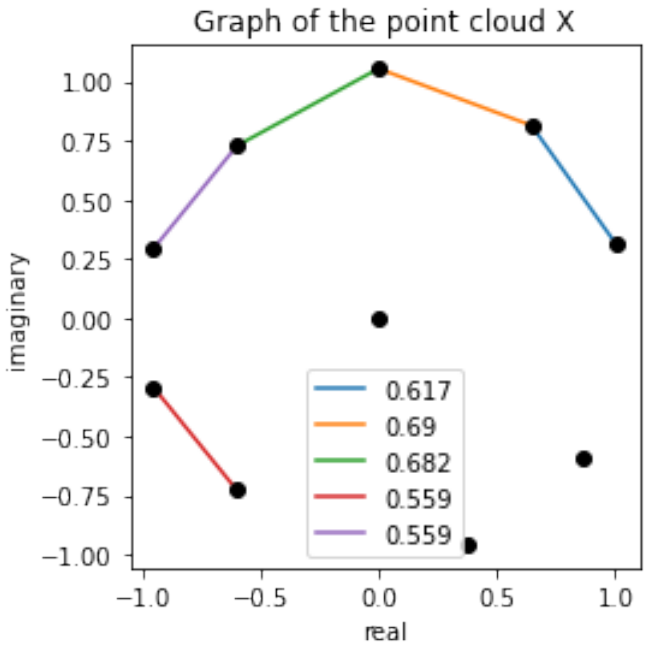


Figura C.11: Gráfica Ball Mapper de X. $\varepsilon = 1/2$.

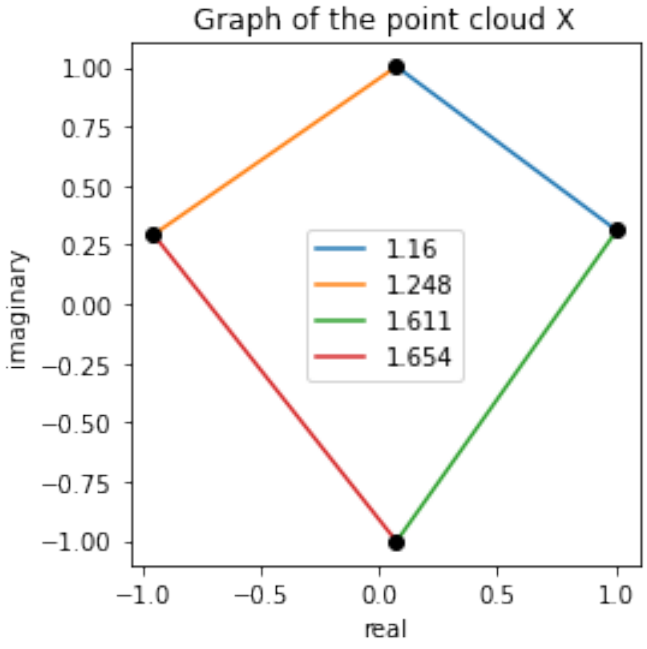


Figura C.12: Gráfica Ball Mapper de X. $\varepsilon = 1$.

Nudo T(23,2)

Polinomio de Jones: $-q^34+q^33-q^32+q^31-q^30+q^29-q^28+q^27-q^26+q^25-q^24+q^23-q^22+q^21-q^20+q^19-q^18+q^17-q^16+q^15-q^14+q^13+q^11$.

Raíces (Nube de puntos X):

- $x_1 = 0$
- $x_2 = 1.04896$
- $x_3 = 0.47225 + 0.91395i$
- $x_4 = 0.47225 - 0.91395i$
- $x_5 = -0.47129 + 0.78995i$
- $x_6 = -0.47129 - 0.78995i$
- $x_7 = -0.311 + 0.90447i$
- $x_8 = -0.311 - 0.90447i$
- $x_9 = 0.2056 + 0.99387i$
- $x_{10} = 0.2056 - 0.99387i$
- $x_{11} = -0.65941 + 0.69626i$
- $x_{12} = -0.65941 - 0.69626i$
- $x_{13} = -0.06719 + 0.99143i$
- $x_{14} = -0.06719 - 0.99143i$
- $x_{15} = 0.70782 + 0.75942i$
- $x_{16} = 0.70782 - 0.75942i$
- $x_{17} = -0.84228 + 0.51111i$
- $x_{18} = -0.84228 - 0.51111i$
- $x_{19} = 0.89188 + 0.54325i$
- $x_{20} = 0.89188 - 0.54325i$
- $x_{21} = -0.95972 + 0.26881i$
- $x_{22} = -0.95972 - 0.26881i$
- $x_{23} = 1.00885 + 0.28308i$
- $x_{24} = 1.00885 - 0.28308i$

$\varepsilon_1 = 1/4$
$\varepsilon_2 = 1/3$
$\varepsilon_3 = 1/2$
$\varepsilon_4 = 1$

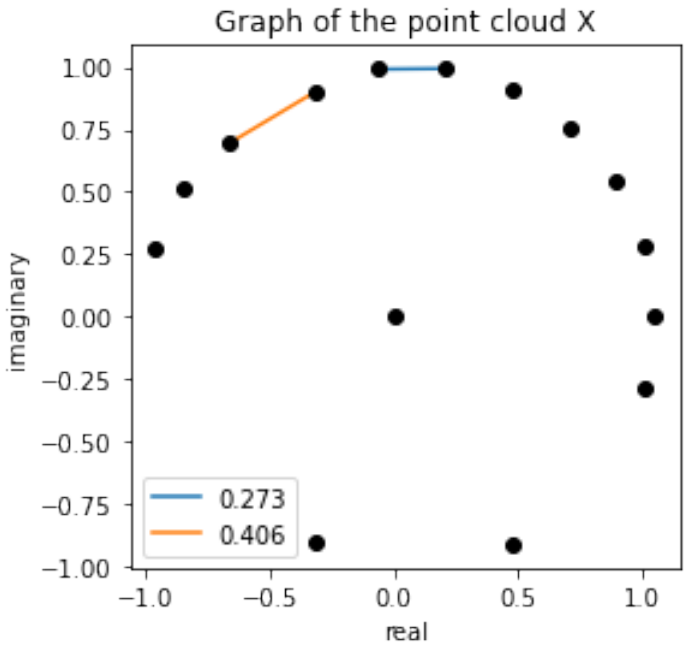


Figura C.13: Gráfica Ball Mapper de X. $\varepsilon = 1/4$.

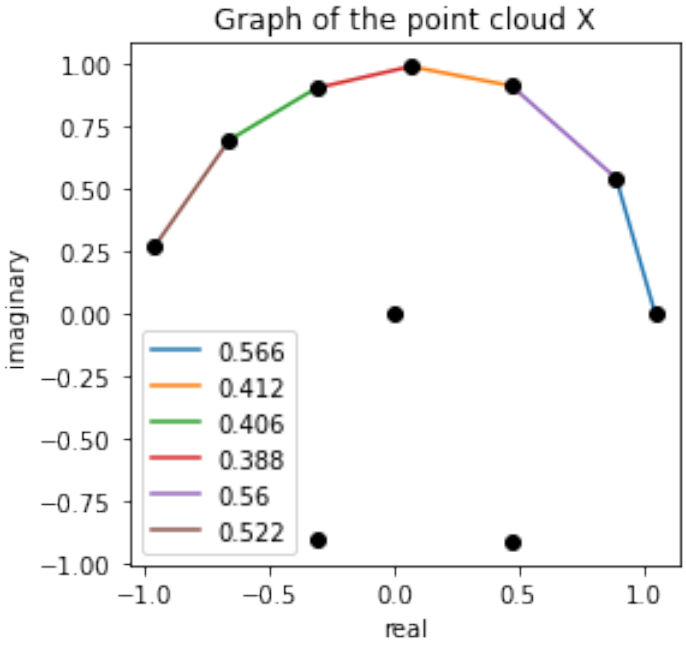


Figura C.14: Gráfica Ball Mapper de X. $\varepsilon = 1/3$.

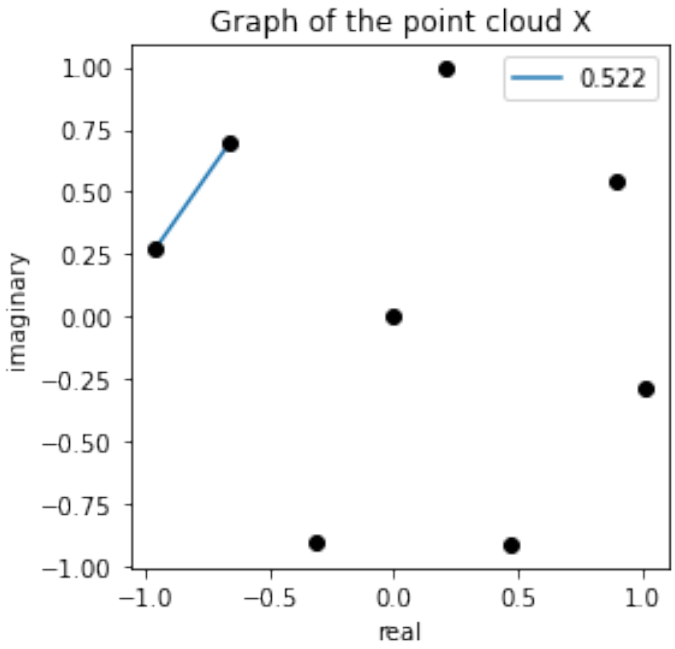


Figura C.15: Gráfica Ball Mapper de X. $\varepsilon = 1/2$.

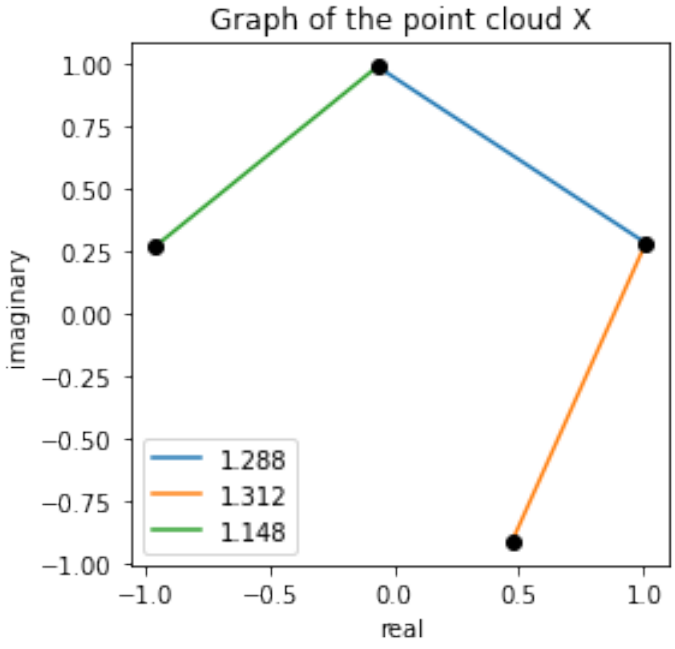


Figura C.16: Gráfica Ball Mapper de X. $\varepsilon = 1$.

Como información adicional, cabe mencionar que las raíces complejas del polinomio de Jones de los nudos tóricos, tienden a acumularse alrededor de la circunferencia unitaria. Para mayor información de ésta y otras características y propiedades de los ceros de los polinomios de Jones, recomendamos revisar [16].

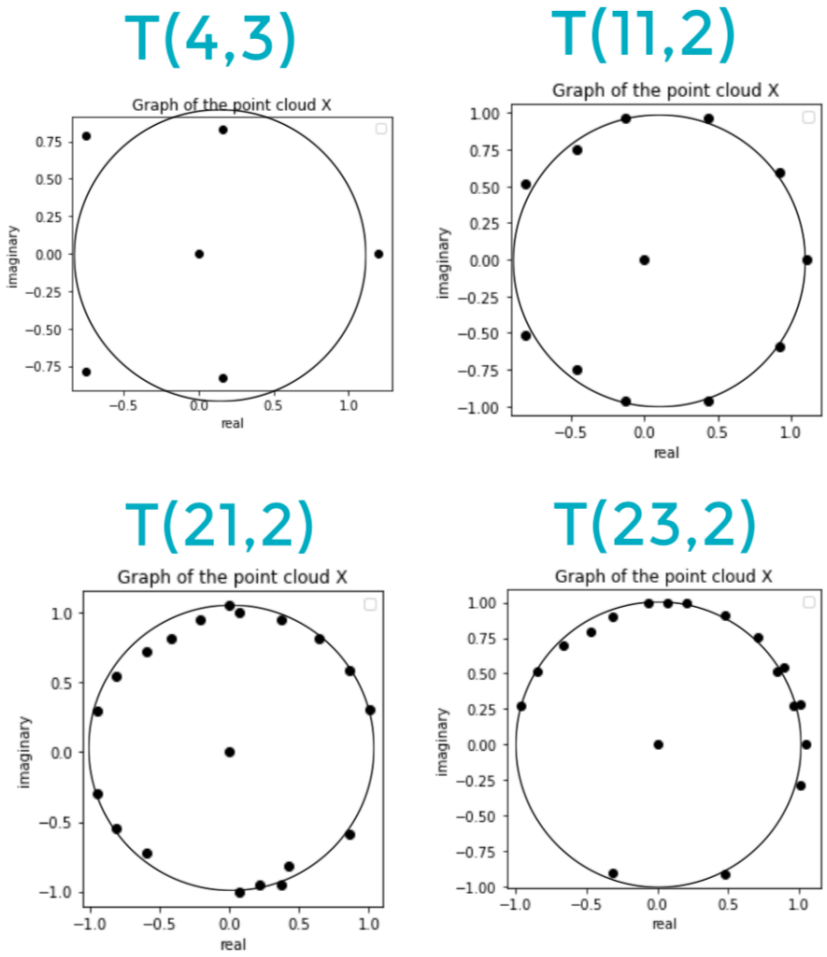


Figura C.17: Ceros del polinomio de Jones de los nudos tóricos $T(4,3)$, $T(11,2)$, $T(21,2)$, $T(23,2)$.

Bibliografía

[1] Dlotko, P. (2019). *Ball mapper: a shape summary for topological data analysis.* arXiv:1901.07410v1w2

[2] Singh, G., Mémoli, F., Carlsson, G. (2007). *Topological methods for the analysis of high dimensional data sets and 3d object recognition.* Eurographics Symposium on Point-Based Graphics 22. http://dx.doi.org/10.2312/SPBG/SPBG07/091-100

[3] Jain A. K., Dubes R. C. (1988). *Algorithms for clustering data. Prentice Hall Advanced Reference Series.* Prentice Hall Inc., Englewood Cliffs, NJ.

[4] Ester M., Kriegel H.P., Sander J., Xu X. (1996). *A density-based algorithm for discovering clusters in large spatial databases with noise.* International Conference on Knowledge Discovery and Data Mining, 226–231.

[5] Johnson S. C. (1967). *Hierarchical clustering schemes.* Psychometrika 2, 241–254.

[6] Sneath, P. H. (1957). *Computers in taxonomy.* J. gen. Microbiol., 17, 201-226.

[7] Carlsson, Gunnar (2009). *Topology and data.* Bulletin of the American Mathematical Society. 46 (2): 255–308. doi:10.1090/S0273-0979-09-01249-X. ISSN 0273-0979

[8] Gower, J. C. Ross, G. J. (1969). *Minimum spanning trees and single linkage cluster analysis.* Journal of the Royal Statistical Society, Series C. 18 (1): 54?64.

[9] Rathore, A., Chalapathi, N., Palande, S., Wang, B. (2020). *TopoAct: Visually Exploring the Shape of Activations in Deep Learning.* School of Computing, Scientific Computing and Imaging (SCI) Institute, University of Utah, USA.

[10] Curto, C. (2017). *What can topology tell us about the neural code?.* Bulletin of the American Mathematical Society, 54(1), 63-78. https://doi.org/10.1090/bull/1554

[11] Hatcher A. (2002). *Algebraic topology.* Cambridge University Press, Cambridge.

[12] Silverman B. W. (1986). *Density estimation for statistics and data analysis. Monographs on Statistics and Applied Probability* . Chapman & Hall, London.

[13] Lafon S., Lee A. B. (2006). *Diffusion maps and coarse-graining: A unified framework for dimensionality reduction, graph partitioning, and data set parameterization.* IEEE Transactions on Pattern Analysis and Machine Intelligence 28, 9, 1393–1403.

[14] Kolmogorov A. N., Fomin S. V. (1996). *Elements of the Theory of Functions and Functional Analysis*. Dover Publications, Incorporated.

[15] Virk, Z. (2021). *Lectures on persistent homology*. University of Ljubljana, Faculty of computer science and informatics.

[16] Wu F. Y., Wang J. (2001). *Zeros of the Jones polynomial*. Physica A 296, 483-494. arXiv:cond-mat/0105013. https://doi.org/10.48550/arXiv.cond-mat/0105013

[17] Jukes T.H., Cantor C.R. (1969). *Evolution of Protein Molecules*. New York: Academic Press. pp. 21–132.