



Universidad Politécnica
de Madrid

**Escuela Técnica Superior de
Ingenieros Informáticos**



Grado en Matemáticas e Informática

Trabajo Fin de Grado

Robustez de la Homología Persistente: el Teorema de Estabilidad

Autor: Alejandro García Castellanos
Tutor: Héctor Barge Yañez

Madrid, 06 - 2021

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Matemáticas e Informática

Título: Robustez de la Homología Persistente: el Teorema de Estabilidad
06 - 2021

Autor: Alejandro García Castellanos
Tutor: Héctor Barge Yañez
Departamento de Matemática Aplicada
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

Sección por hacer

«Aquí va el resumen del TFG. Extensión máxima 2 páginas.»

Abstract

Sección por hacer

«Abstract of the Final Degree Project. Maximum length: 2 pages.»

Tabla de contenidos

1. Introducción	1
1.1. Motivación	1
1.2. Descripción general del trabajo	1
1.3. Estructura del trabajo	2
2. Desarrollo	3
2.1. Conocimientos previos y definiciones	3
2.1.1. Complejos Simpliciales	3
2.1.2. Complejos simpliciales de nubes de puntos	9
2.1.3. Homología	13
2.1.4. Persistencia	19
2.2. Teorema de estabilidad	27
2.2.1. Proposición del teorema	27
2.2.2. Estabilidad para la distancia Hausdorff	28
2.2.3. Estabilidad para la distancia bottleneck	35
2.3. Implementaciones y cálculos	40
2.3.1. Cálculo de la distancia Hausdorff	40
2.3.2. Cálculo de la distancia bottleneck	41
2.3.3. Pruebas	45
2.3.3.1. Ejemplo 1: Elipse	47
2.3.3.2. Ejemplo 2: Rosa polar	48
2.3.3.3. Ejemplo 3: Elecciones generales noviembre 2019	50
3. Resultados y conclusiones	55
4. Análisis de impacto	57
Bibliografía	60
Anexo	61
.1. Implementación de las distancias Hausdorff y bottleneck	61
.2. Visualización de la función distancia a un conjunto	65
.3. Implementación de los ejemplos	66

Capítulo 1

Introducción

1.1. Motivación

El crecimiento de la cantidad de datos que producimos y almacenamos es exponencial, estimándose que para 2025 llegaremos a haber creado un total de 160 zettabytes [1]. En este contexto surge la necesidad de analizar y comprender las características de grandes conjuntos de datos. Así pues, nace la disciplina del *Análisis Topológico de Datos* con el fin de responder a las siguientes preguntas sobre las propiedades cualitativas geométricas de nuestros conjuntos de datos: ¿Cuáles son las características topológicas de mi conjunto de datos? Si hemos introducido complejidad adicional a nuestros datos debido a problemas de medición o de discretización, ¿cómo medimos la relevancia de las características observadas?

Para poder intentar contestar a estas preguntas se hace uso de la *homología persistente*, y más concretamente de los *diagramas de persistencia*. Estos diagramas son multiconjuntos de puntos en el plano extendido, donde cada punto representa una característica cualitativa de nuestros datos y la diferencia en valor absoluto de sus coordenadas cuantifica su relevancia. Algunas de las características usuales que se miden son el número de componentes conexas, el número de agujeros y el número de cavidades.

1.2. Descripción general del trabajo

El trabajo se basa en el estudio y exposición del Teorema de estabilidad, que establece, a grandes rasgos, que pequeñas perturbaciones en los datos implican pequeñas perturbaciones en la homología persistente.

Para ello he buscado y estudiado diversas referencias acerca de este resultado y su demostración [2, 3, 4].

Adicionalmente, he implementado en Python las distancias *Bottleneck* y *Hausdorff* para poder ilustrar este resultado haciendo uso distintos conjuntos de datos.

Las pruebas se sustentan en el *pipeline* usual del Análisis Topológico de Datos, ilustrado en la figura 1.1. Partiremos de un conjunto de datos, en este caso, una nube de puntos. Posteriormente se obtendrá una familia de espacios topológicos sobre los

obtendremos la homología persistente que será representada en el diagrama de persistencia. Para estudiar la estabilidad, se introducirán perturbaciones en conjunto de puntos inicial y se compararán los diagramas de persistencia.

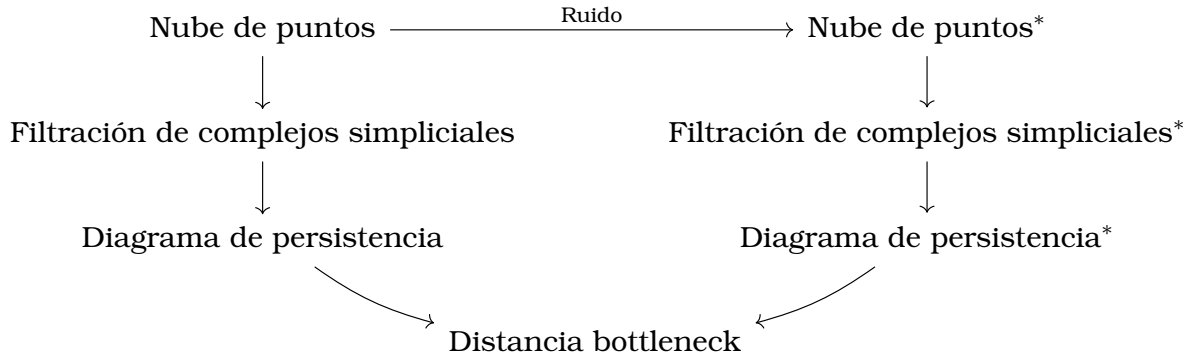


Figura 1.1: Pipeline para la comprobación del Teorema de estabilidad

1.3. Estructura del trabajo

En la sección 2.1 se introducirán las nociones topológicas básicas para hacer autocontenido el trabajo. En ella introduciremos los *complejos simpliciales*, la *homología* y la *persistencia*. Continuando con la sección 2.2, donde enunciaremos y demostraremos el *Teorema de estabilidad*, usando como referencia el siguiente artículo [4]. Y terminando con la sección 2.3, en la que entraremos en detalle en los algoritmos propuestos para la implementación de las distancias *Bottleneck* y *Hausdorff*, mostrando diversas pruebas para ilustrar los resultados del teorema.

Capítulo 2

Desarrollo

2.1. Conocimientos previos y definiciones

En esta sección se introducirán las nociones topológicas básicas para hacer autocontenido este trabajo. Dichas nociones nos darán el contexto y conocimientos necesarios para poder profundizar en el *Teorema de Estabilidad* y ser capaces de abordar su demostración.

2.1.1. Complejos Simpliciales

Una forma de representar algunos espacios topológicos es a través de su descomposición en piezas más sencillas. Una descomposición de estas características se denomina complejo si sus piezas son topológicamente simples y sus intersecciones son piezas del mismo tipo, pero de dimensión inferior [2]. Existe una gran variedad de complejos con distintos grados de abstracción. En este trabajo nos centraremos en los complejos simpliciales, que permiten representar una gran variedad de espacios y son especialmente adecuados para cuestiones computacionales.

Los complejos simpliciales pueden ser estudiados desde un enfoque geométrico y desde un enfoque combinatorio. Partiremos de la definición de complejo simplicial desde el punto de vista geométrico. Para ello recordaremos algunos conceptos de geometría afín.

Definición 2.1.1. El conjunto de puntos $\{u_0, u_1, \dots, u_k\}$ de \mathbb{R}^d es *afinmente independiente* si los vectores $\{\overrightarrow{u_0 u_1}, \dots, \overrightarrow{u_0 u_k}\}$ son linealmente independientes.

Definición 2.1.2. Diremos que $x \in \mathbb{R}^d$ es *combinación convexa* de los puntos u_0, u_1, \dots, u_k si $x = \sum_{i=0}^k \lambda_i u_i$ con $\lambda_i \geq 0$ para todo $i \in \{0, \dots, k\}$ y $\sum_{i=0}^k \lambda_i = 1$.

Definición 2.1.3. Llamaremos *envolvente convexa* de u_0, u_1, \dots, u_k , denotado por $\text{conv}\{u_0, u_1, \dots, u_k\}$, al conjunto de todas las combinaciones convexas de dichos puntos.

Haciendo uso de este conjunto podremos definir nuestras piezas de la descomposición de la siguiente manera:

Definición 2.1.4. Un k -*simplex* σ en \mathbb{R}^d con $d \geq k$ es la envolvente convexa de $k + 1$ puntos afínmente independientes $u_0, u_1, \dots, u_k \in \mathbb{R}^d$, es decir, $\sigma := \text{conv}\{u_0, u_1, \dots, u_k\}$.

2.1. Conocimientos previos y definiciones

Diremos que el k -símplice σ tiene dimensión k y llamaremos *vértices de σ* a los puntos u_0, u_1, \dots, u_k .

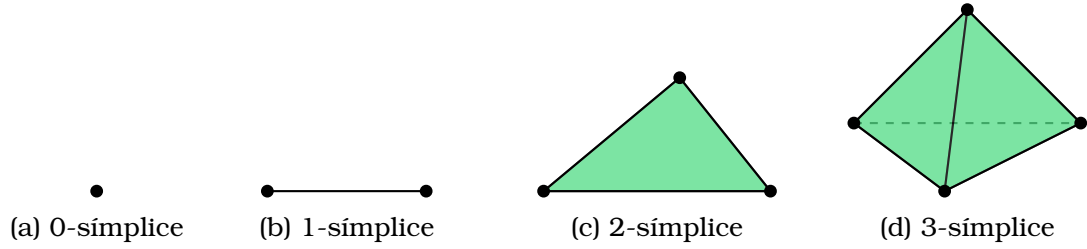


Figura 2.1: Representación de los símlices de dimensión 0, 1, 2 y 3

Se puede observar que cualquier subconjunto de los vértices de σ será afinmente independiente y por lo tanto definirá un símplex τ de dimensión inferior. De esta forma diremos que τ es una cara de σ si es una combinación convexa de un subconjunto no vacío de los vértices de σ , y lo denotaremos por $\tau \leq \sigma$. Si el subconjunto es propio, diremos que τ es cara propia de σ , y lo denotaremos por $\tau < \sigma$. Por otro lado, diremos que σ es cocara (propia) de τ si $\sigma \geq \tau$ ($\sigma > \tau$).

Haciendo uso de la definición de caras de un símplex σ podemos definir el borde y el interior de σ .

Definición 2.1.5. Sea σ un símplex. Entonces

- Se define el borde de σ como

$$\text{bd } \sigma = \bigcup_{\tau < \sigma} \tau.$$

- Se define el interior de σ como

$$\text{int } \sigma = \sigma - \text{bd } \sigma.$$

Observación. Se sigue directamente de la definición que un punto $x \in \sigma$ pertenece al interior de σ si y sólo si todos sus coeficientes λ_i de la combinación convexa son positivos. Se sigue que cada punto $x \in \sigma$ pertenece únicamente al interior de la cara generada por los puntos con coeficientes λ_i positivos.

Una vez que ya conocemos las piezas de nuestra descomposición vamos a ver como tenemos que unir las y cuáles son las principales propiedades de los complejos resultantes.

Como ya hemos visto al principio de la sección, para que una descomposición sea un complejo sus piezas tienen que ser topológicamente simples y sus intersecciones tienen que ser piezas de dimensión inferior del mismo tipo. La manera natural de hacer esto es pegar unos símlices con otros por sus caras.

Definición 2.1.6. Un *complejo simplicial* es una colección finita de símlices K que satisface las siguientes propiedades:

- Si $\sigma \in K$ y $\tau \leq \sigma$ entonces $\tau \in K$.
- Si $\sigma_0, \sigma_1 \in K$ y $\sigma_0 \cap \sigma_1 \neq \emptyset$ entonces $\sigma_0 \cap \sigma_1 \leq \sigma_i$ para $i = 0, 1$.

Se define la dimensión de como el máximo de las dimensiones de sus símlices.

Un ejemplo de complejo simplicial es lo que se muestra en la figura 2.2, mientras que en la figura 2.3 muestra un ejemplo que no es complejo simplicial.

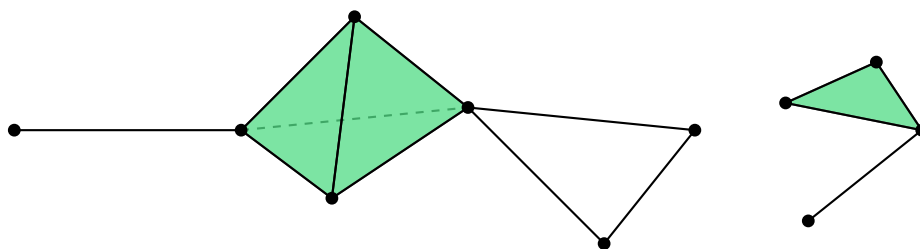


Figura 2.2: Ejemplo de complejo simplicial

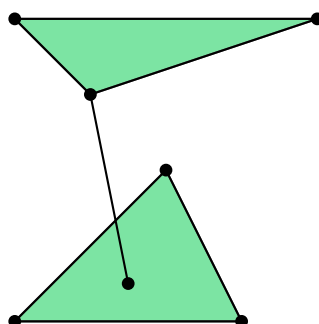


Figura 2.3: Ejemplo de conjunto de simples que no cumplen las condiciones de complejo simplicial

Definición 2.1.7. El *espacio subyacente* de un complejo simplicial K , denotado $|K|$, es la unión de los simples de K con la topología heredada del \mathbb{R}^d donde viven sus simples. Este espacio subyacente también es llamado *poliedro*.

Como se puede observar, el espacio subyacente de un complejo simplicial es compacto, siendo unión finita de simples. El siguiente resultado caracteriza los abiertos y cerrados del espacio subyacente $|K|$ de un complejo simplicial K .

Proposición 2.1.1 ([2]). Sea K un complejo simplicial y $A \subset |K|$ un subconjunto. Entonces A es un abierto (cerrado) en K si y sólo si para cada $\sigma \in K$, $A \cap |\sigma|$ es un abierto (cerrado) de $|\sigma|$.

Definición 2.1.8. Una *triangulación* de un espacio topológico X es un par (K, h) donde K es un complejo simplicial y $h : X \rightarrow |K|$ es un homeomorfismo (h continua, biyectiva y h^{-1} continua).

Diremos que un espacio topológico es *triangulable* si admite una triangulación.

También nos será de utilidad poder estudiar los complejos simpliciales contenidos en otro complejo simplicial.

Definición 2.1.9. Un *subcomplejo* L de un complejo simplicial K es un complejo simplicial $L \subseteq K$.

Un subcomplejo de gran interés son los *j-esqueletos*, definidos de la siguiente forma:

$$K^{(j)} = \{\sigma \in K \mid \dim \sigma \leq j\}.$$

Otro subconjunto de símlices que nos será de gran ayuda más adelante es la *estrella de un símlice* τ , la cual consiste de las cocaras de τ , denotado por $\text{St } \tau$. Este conjunto no será siempre un complejo simplicial, así que se define la *estrella cerrada* $\overline{\text{St}} \tau$ como el menor subcomplejo de K que contiene a $\text{St } \tau$. Adicionalmente, se define el *link* de τ como: $\text{Lk } \tau = \{v \in \overline{\text{St}} \tau \mid v \cap \tau = \emptyset\}$.

Complejos simpliciales abstractos

Una vez que ya conocemos los complejos simpliciales desde el punto de vista geométrico, vamos a abordarlos desde un enfoque combinatorio, el cual nos será de gran ayuda para poder programar los complejos simpliciales.

Definición 2.1.10. Un *complejo simplicial abstracto* A es una colección finita de conjuntos finitos tal que si $\alpha \in A$ y $\beta \subset \alpha$ entonces $\beta \in A$.

De esta forma se cumple que

- Los conjuntos en A no vacíos se denominan *símlices abstractos*.
- La *dimensión* de un símlice abstracto $\alpha \in A$ es $\dim \alpha = \text{card}(\alpha) - 1$. Y la dimensión del complejo es el máximo de las dimensiones de sus símlices.
- Una *cara* de $\alpha \in A$ es cualquier subconjunto no vacío de $\beta \subset \alpha$.
- El *conjunto de vértices* de A , denotado por $\text{Vert } A$, es la unión de todos sus símlices.
- Un *subcomplejo* B de un complejo simplicial abstracto A es un complejo simplicial abstracto $B \subset A$.

Ejemplo 2.1.1. Un ejemplo de complejo simplicial abstracto es el siguiente conjunto

$$A = \{\{0\}, \{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}, \{0, 1\}, \{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}, \{4, 5\}, \{4, 6\}, \{5, 6\}, \{1, 2, 3\}, \{1, 2, 4\}, \{1, 3, 4\}, \{2, 3, 4\}, \{1, 2, 3, 4\}\}.$$

Donde el conjunto de vértices es: $\text{Vert } A = \{0, 1, 2, 3, 4, 5, 6\}$.

Definición 2.1.11. Sean A y B dos complejos simpliciales abstractos. Diremos que A y B son *isomorfos* si existe una biyección

$$b : \text{Vert } A \rightarrow \text{Vert } B$$

tal que $\alpha \in A$ si y sólo si $b(\alpha) \in B$.

Cada complejo geométrico induce de manera natural un complejo abstracto de la siguiente forma:

Definición 2.1.12. Sea K un complejo simplicial y V el conjunto de vértices de K . Llamaremos *esquema de vértices* al complejo simplicial abstracto A formado por todos aquellos subconjuntos de V que generan símlices en K .

Y bajo ciertas circunstancias podremos hacer el paso opuesto de construir un complejo simplicial (geométrico) a partir de otro abstracto:

Definición 2.1.13. Sean A un complejo simplicial abstracto y K un complejo simplicial. Diremos que K es una *realización geométrica* de A , si A es isomorfo al esquema de vértices de K .

Teorema 2.1.1 ([2]). *Todo complejo simplicial abstracto de dimensión d admite una realización geométrica en \mathbb{R}^{2d+1} .*

Así pues, los complejos simpliciales abstractos son una representación fiel de un complejo simplicial (geométrico).

Aplicaciones simpliciales

Una vez que ya conocemos las principales propiedades de los complejos simpliciales, veremos cuales son las aplicaciones que preservan la estructura de complejo simplicial. Como vimos anteriormente, cada punto de un k -símplice pertenece al interior de exactamente una cara. Por lo tanto, todo punto $x \in |K|$, siendo K un complejo simplicial de vértices u_0, u_1, \dots, u_n , pertenece al interior de uno de los símlices de K . Si $\sigma = \text{conv}\{u_0, u_1, \dots, u_k\}$ es dicho símplex, entonces $x = \sum_{i=0}^k b_i(x)u_i$, donde

$$b_i(x) = \begin{cases} \lambda_i & \text{si } 0 \leq i \leq k \\ 0 & \text{si } k+1 \leq i \leq n \end{cases}, \text{ con } \lambda_i \text{ tal que } x = \sum_{i=0}^k \lambda_i u_i$$

se denominan *coordenadas baricéntricas* de x en K .

Haremos uso de estas coordenadas para construir una función, lineal a trozos inducida por una función entre los vértices de dos complejos simpliciales, denominada *aplicación de vértices*

Definición 2.1.14. Sean K y L complejos simpliciales y $\varphi : \text{Vert } K \rightarrow \text{Vert } L$ una aplicación. Diremos que φ es una *aplicación de vértices* si satisface que para cada $\sigma \in K$ su imagen $\varphi(\sigma) \in L$.

Una aplicación de vértices $\varphi : \text{Vert } K \rightarrow \text{Vert } L$ induce una aplicación, lineal a trozos $f : |K| \rightarrow |L|$ dada por

$$f(x) = f\left(\sum_{i=0}^n b_i(x)u_i\right) = \sum_{i=0}^n b_i(x)\varphi(u_i)$$

a la que llamaremos *aplicación simplicial* asociada a φ . Para enfatizar que es una aplicación lineal en cada símplex del complejo, se suele notar la aplicación de la siguiente forma $f : K \rightarrow L$.

Subdivisiones

Veremos que hay ocasiones que nos interesará controlar el tamaño de los símlices de nuestro complejo simplicial conservando el espacio subyacente. Por esta razón, se introduce la noción de *subdivisión de un complejo simplicial*.

Definición 2.1.15. Sea K un complejo simplicial. Diremos que un complejo simplicial L es una *subdivisión* de K si:

- $|K| = |L|$.
- Cada símplex de L está contenido en un símplex de K .

2.1. Conocimientos previos y definiciones

Hay muchas maneras de obtener subdivisiones de un complejo simplicial, pero un tipo particular de subdivisión que es muy utilizada es la *subdivisión baricéntrica*, denotada por $L = \text{Sd}K$. Para la construcción de esta subdivisión, introducimos el *baricentro* de un símple y el *cono* de un símple de vértice v .

Definición 2.1.16. Sea σ un k -símple, tal que $\sigma = \text{conv}\{v_0, v_1, \dots, v_k\}$. Llamaremos *baricentro* de σ al punto

$$b_\sigma = \sum_{i=0}^k \frac{v_i}{k+1} \in \text{int } \sigma.$$

Definición 2.1.17. Sea σ un k -símple, tal que $\sigma = \text{conv}\{v_0, v_1, \dots, v_k\}$ y v un punto no contenido en el subespacio afín generado por $\{v_0, v_1, \dots, v_k\}$. Se define el *cono* de σ con vértice v y se denota por $\sigma * v$ como el $k+1$ -símple generado por $\{v, v_0, v_1, \dots, v_k\}$.

Definición 2.1.18. Sea K un complejo simplicial. Se define la *subdivisión baricéntrica* de K como el complejo simplicial $\text{Sd}K$ que se construye inductivamente sobre el j -esqueleto como sigue:

- A. $\text{Sd}K^{(0)} = K^{(0)}$.
- B. $\text{Sd}K^{(j)}$ es la unión de $\text{Sd}K^{(j-1)}$ con el conjunto de todos los símple de la forma $b_\sigma * \tau$, donde σ es un j -símple y τ es cualquier símple de $\text{Sd}K^{(j-1)}$ contenido en una cara de σ .

En la figura 2.4 se muestra la primera y segunda subdivisión baricéntrica de un complejo simplicial.

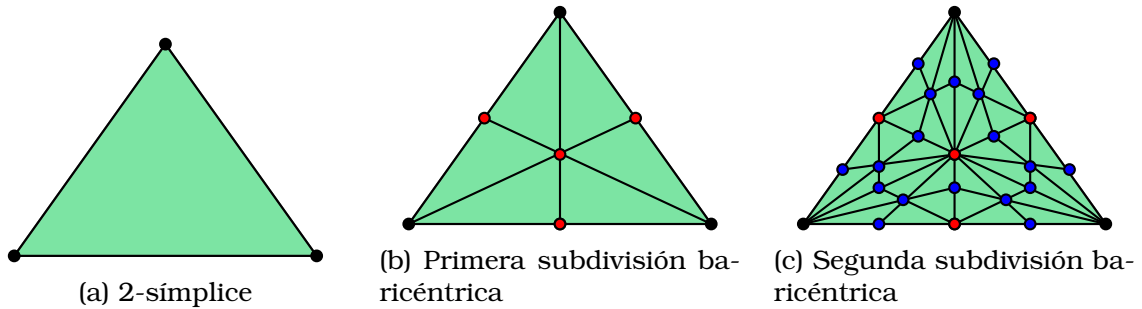


Figura 2.4: Primera y segunda subdivisión baricéntrica de un 2-símple

Recordemos que el *diámetro* de un subconjunto $A \subset \mathbb{R}^d$ es el supremo sobre las distancias entre sus puntos.

Lema 2.1.2 ([2]). Si σ es un k -símple, entonces el diámetro de cada símple en la subdivisión baricéntrica de σ es como máximo $\frac{k}{k+1} \text{diam } \sigma$.

De forma que gracias al lema anterior podremos hacer el diámetro de los símple de los complejos simpliciales tan pequeño como queramos, ya que el diámetro de los símple de la n -ésima subdivisión baricéntrica del complejo simplicial K , denotado por $\text{Sd}^n K = \text{Sd}(\text{Sd}^{n-1} K)$, es

$$\left(\frac{k}{k+1} \right)^n \text{diam } \sigma \xrightarrow{n \rightarrow \infty} 0, \text{ con } \sigma \in K \text{ y } k = \dim \sigma.$$

Aproximaciones simpliciales

Para estudiar la topología de los poliedros es fundamental aproximar funciones continuas por aplicaciones simpliciales. Para poder definir estas aproximaciones primero vamos a definir un tipo de entorno de los vértices de un complejo como se puede ver en la figura 2.5.

Definición 2.1.19. Sea K un complejo simplicial y v un vértice de K . El conjunto

$$N(v) = \bigcup_{\sigma \in \text{St } v} \text{int } \sigma$$

es un entorno abierto de v en $|K|$ al que llamaremos *entorno estrellado* de v .

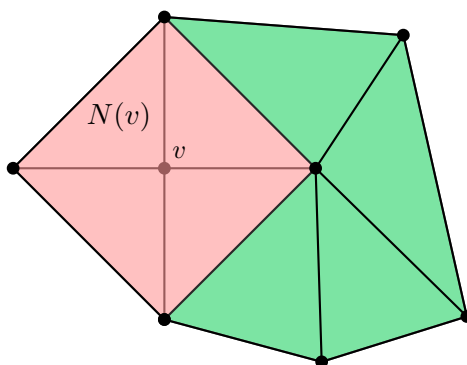


Figura 2.5: Entorno estrellado de v marcado en color rojo

Así pues, definimos una aproximación simplicial de la siguiente forma:

Definición 2.1.20. Sean K y L complejos simpliciales, $g : |K| \rightarrow |L|$ una aplicación continua y $f : K \rightarrow L$ una aplicación simplicial. Diremos que f es una *aproximación simplicial* de g si verifica la *condición de estrella*, es decir, si para cada vértice $v \in K$ se tiene que $g(N(v)) \subset N(f(v))$.

Además, la condición de estrella será una condición suficiente para garantizar la existencia de una aproximación simplicial:

Lema 2.1.3 ([2]). Sean K y L complejos simpliciales, $g : |K| \rightarrow |L|$ una aplicación continua que satisface la condición de estrella. Entonces g tiene una aproximación simplicial $f : K \rightarrow L$.

En la figura 2.6 podemos ver un ejemplo de aproximación simplicial de una aplicación continua.

Teorema 2.1.4 (Aproximación simplicial [2]). Sean K y L complejos simpliciales, $g : |K| \rightarrow |L|$ una aplicación continua. Entonces existe $n \in \mathbb{N}$ tal que g tiene una aproximación simplicial $f : \text{Sd}^n K \rightarrow L$.

2.1.2. Complejos simpliciales de nubes de puntos

Desde el punto de vista computacional nos encontramos con el problema de que tenemos una representación de un espacio topológico a través de una discretización finita de los puntos de dicho espacio, y nuestro objetivo es poder recuperar propiedades del espacio topológico original a partir de esta nube de puntos. Para ello asociaremos complejos simpliciales a dicha nube de puntos.

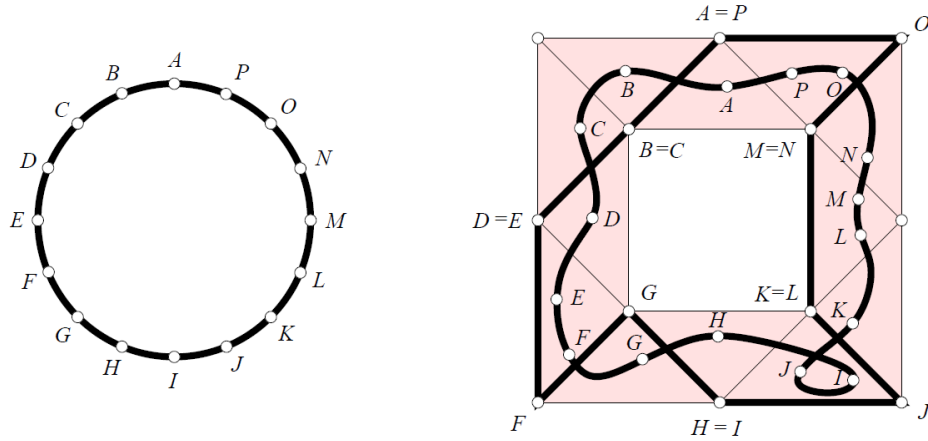


Figura 2.6: Aplicación continua del círculo en una corona circular y una aproximación simplicial de dicha aplicación. Fuente: [2]

Complejo de Čech

El complejo de Čech se define a partir de la intersección de una colección de bolas cerradas. La idea que subyace a esta construcción es la del nervio de una colección, que se introduce a continuación.

Definición 2.1.21. Sea F una colección finita de conjuntos. Se define el *nervio* de F como el complejo simplicial abstracto

$$\text{Nrv } F = \left\{ X \subseteq F \mid \bigcap X \neq \emptyset \right\}.$$

Lema 2.1.5 (Lema del nervio). Si F es una colección finita de subconjuntos cerrados y convexos en el espacio Euclídeo, entonces el nervio de F tiene el mismo tipo de homotopía que la unión de los conjuntos de F .

Consideramos el caso particular en el que los conjuntos de la familia son las bolas cerradas $\overline{B}_r(x) = \{y \in \mathbb{R}^d \mid d(x, y) \leq r\}$ en \mathbb{R}^d .

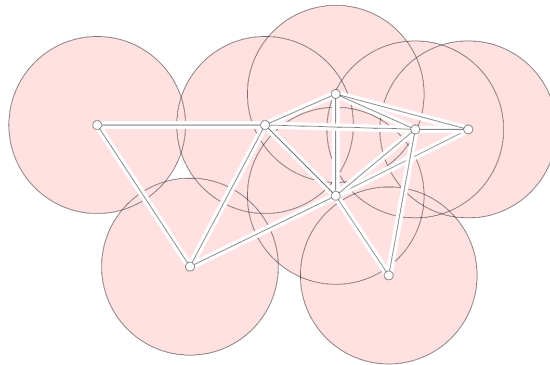


Figura 2.7: Complejo de Čech para un conjunto de nueve puntos y un radio r . Fuente: [2]

Definición 2.1.22. Sea $S \subset \mathbb{R}^d$ un conjunto finito de puntos. Llamaremos *complejo*

de Čech de S de radio r al complejo simplicial abstracto

$$\check{\text{Cech}}(r) = \left\{ \sigma \subset S \mid \bigcap_{u \in \sigma} \overline{B}_r(u) \neq \emptyset \right\}.$$

El complejo de Čech es isomorfo al nervio de la colección de las bolas cerradas de radio r centrada en los puntos de S . En la figura 2.7 podemos observar un ejemplo de complejo de Čech.

Podemos comprobar [2] que para valores de r lo suficientemente grandes, $\check{\text{Cech}}(r)$ es un simple de dimensión $\text{card}(S) - 1$, por lo que el complejo de Čech es poco eficiente desde el punto de vista computacional.

Además, en general, el complejo de Čech de un conjunto de puntos $S \subset \mathbb{R}^d$ no posee una realización geométrica en \mathbb{R}^d .

Complejo de Vietoris-Rips

Definición 2.1.23. Sea $S \subset \mathbb{R}^d$ un conjunto finito de puntos. Llamamos *complejo de Vietoris-Rips* de S de radio r al complejo simplicial abstracto

$$\text{VR}(r) = \{ \sigma \subseteq S \mid \text{diam } \sigma \leq 2r \}$$

donde $\text{diam } \sigma$ denota el diámetro del subconjunto σ .

En la figura 2.8 podemos observar como se generan los diversos complejos de VR a medida que se va aumentando el radio.

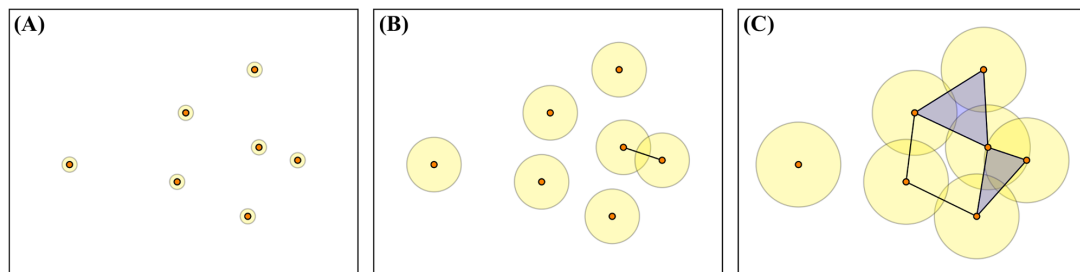


Figura 2.8: Complejos de Vietoris-Rips para un conjunto de siete puntos a medida que aumentamos el radio de izquierda a derecha. Fuente: [5]

Sea $\sigma \subset S$, entonces recordamos que el diámetro se define como

$$\text{diam } \sigma = \max_{u, v \in \sigma} d(u, v).$$

Esta observación garantiza que $\sigma \in \text{VR}(r)$ si y sólo si todas sus aristas están en $\text{VR}(r)$. Dicho de otro modo, $\text{VR}(r)$ está completamente determinado por su 1-esqueleto. Esto hace que el complejo de Vietoris-Rips sea mucho más eficiente que el complejo de Čech desde el punto de vista computacional. Sin embargo, al igual que ocurre con el complejo de Čech, no admite una realización geométrica en \mathbb{R}^d .

Por otro lado, el complejo de Vietoris-Rips no es el nervio de ningún recubrimiento. Sin embargo, el siguiente resultado garantiza que el complejo de VR aproxima al complejo de Čech.

Lema 2.1.6 (Lema de Vietoris-Rips [2]). Sea $S \subset \mathbb{R}^d$ un conjunto finito de puntos y sea $r \geq 0$. Entonces,

$$\check{\text{Cech}}(r) \subset \text{VR}(r) \subset \check{\text{Cech}}(\sqrt{2}r).$$

Complejo de Delaunay

En esta sección introduciremos construcciones geométricas que nos limitarán la dimensión de los símplexes que obtenemos del nervio de una colección finita de conjuntos.

Definición 2.1.24. Sea $S \subset \mathbb{R}^d$ un conjunto finito. Se define la *celda de Voronoi* de un punto $u \in S$ como el conjunto de los puntos

$$V_u = \{x \in \mathbb{R}^d \mid d(x, u) \leq d(x, v), \text{ para todo } v \in S\}.$$

La colección de las celdas de Voronoi de los puntos de S se denomina *diagrama de Voronoi* de S .

En la figura 2.9 se puede ver el diagrama de Voronoi de un conjunto de puntos. Nótese que las celdas de Voronoi recubren todo el espacio.

Definición 2.1.25. Sea $S \subset \mathbb{R}^d$ un conjunto finito. Se define el *complejo de Delaunay* de S como el complejo simplicial abstracto

$$\text{Del} = \left\{ \sigma \subseteq S \mid \bigcap_{u \in \sigma} V_u \neq \emptyset \right\}.$$

Definición 2.1.26 ([6]). Un conjunto de puntos en un espacio afín d -dimensional está en *posición general* si ningún subconjunto de k puntos está contenido en un subespacio afín $(k - 2)$ -dimensional, para $k = 2, 3, \dots, d + 1$.

El complejo de Delaunay es un complejo isomorfo al nervio del diagrama de Voronoi. Además, si los puntos de S están en posición general, se obtiene una realización del complejo de Delauney en \mathbb{R}^d considerando envolventes convexas de los símplexes abstractos. Esta realización geométrica se denomina *triangulación de Delaunay*.

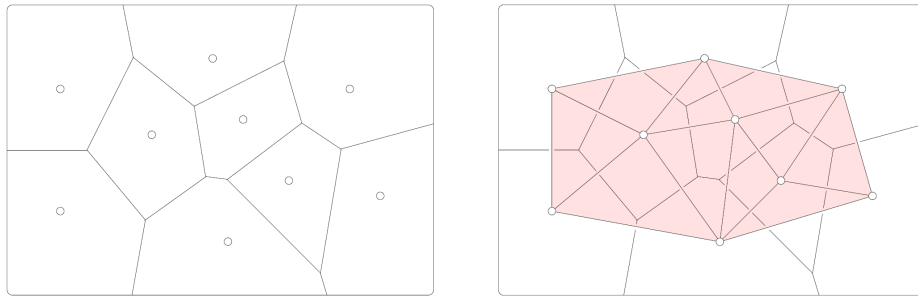


Figura 2.9: A la izquierda tenemos el Diagrama de Voronoi de un conjunto de nueve puntos en el plano, y a la derecha triangulación de Delaunay superpuesta al diagrama de Voronoi. Fuente: [2]

Alfa complejo

Sea $S \subset \mathbb{R}^d$ un conjunto finito de puntos y $r \geq 0$. Para cada $u \in S$ consideramos la región $R_u(r) = \overline{B}_r(u) \cap V_u$, es decir, la intersección de la región de Voronoi de u con la bola cerrada de centro u y radio r .

Definición 2.1.27. Sea $S \subset \mathbb{R}^d$ un conjunto finito de puntos y $r \geq 0$. Se define el *Alfa complejo* de radio r asociado a S como el complejo simplicial abstracto

$$\text{Alpha}(r) = \left\{ \sigma \in S \mid \bigcap_{u \in \sigma} R_u(r) \neq \emptyset \right\}.$$

En la figura 2.10 se puede observar la unión de dichas regiones y su correspondiente alfa complejo. Se puede observar que el alfa complejo es isomorfo al nervio de la colección formada por los $R_u(r)$.

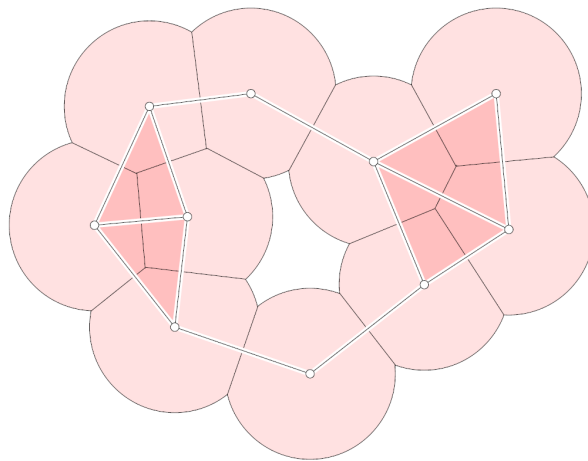


Figura 2.10: Unión de las regiones $R_u(r)$ asociadas a un radio r y un conjunto finito de puntos S . El correspondiente alfa complejo es superpuesto a esta unión de regiones. Fuente: [2]

Puesto que $R_u(r) \subset \overline{B}_r(u)$ para cada $u \in S$, se tiene que $\text{Alpha}(r) \subset \check{\text{Cech}}(r)$. Del mismo modo, dado que $R_u(r) \subset V_u$ para cada $u \in S$, se tiene que $\text{Alpha}(r) \subset \text{Del}(S)$.

Propiedad 2.1.1 ([2]). $|\text{Alpha}(r)|$ y $|\check{\text{Cech}}(r)|$ tienen el mismo tipo de homotopía.

Además, el alfa complejo tiene menos simplices que el complejo de Čech. Y como es un subcomplejo del complejo de Delaunay, admite de manera natural una realización en \mathbb{R}^d . Por lo que hace que los alfa complejos sean una buena opción desde el punto de vista computacional.

2.1.3. Homología

Como se puede ver en [7], la homotopía es una herramienta algebraica para poder obtener propiedades de los espacios topológicos. Sin embargo, los métodos para el cálculo de la homotopía no son manejables computacionalmente. Así pues, se propone la homología como formalismo algebraico, que, aunque no es capaz de obtener tanta información topológica sobre el espacio como con otros formalismos, es muy computable.

Comenzaremos estudiando los diversos grupos que están involucrados en la definición de la homología.

Grupos de cadenas

Sea K un complejo simplicial y p un número entero no negativo. Una p -cadena en K es una suma formal de p -símplices en K . Más concretamente, c es una p -cadena en K si

$$c = \sum a_i \sigma_i$$

con σ_i es un p -símplice para cada i y a_i son los *coeficientes*. Estos coeficientes pueden tomarse de cualquier anillo conmutativo, sin embargo, nosotros usaremos con coeficientes en el cuerpo de dos elementos, es decir, $a_i \in \mathbb{Z}_2$.

Ejemplo 2.1.2. Escribiremos los símplices como la lista de sus vértices, $\sigma = [u_0, u_1, \dots, u_p]$.

- En la figura 2.11 se muestra en rojo la 0-cadena $c = [0] + [2] + [6] + [9]$.

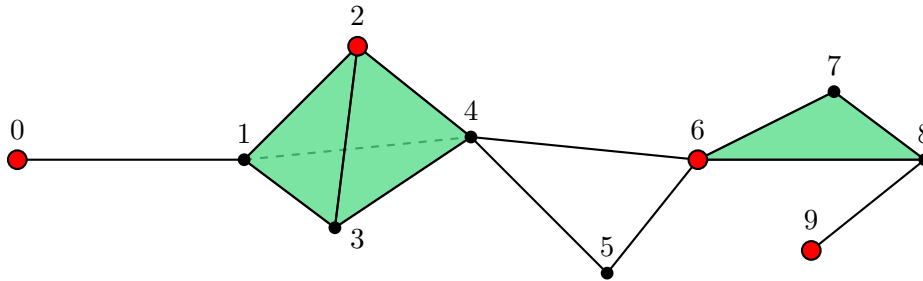


Figura 2.11: Ejemplo de 0-cadena

- En la figura 2.12 se muestra en rojo la 1-cadena $c = [0, 1] + [1, 2] + [2, 4] + [8, 9]$.

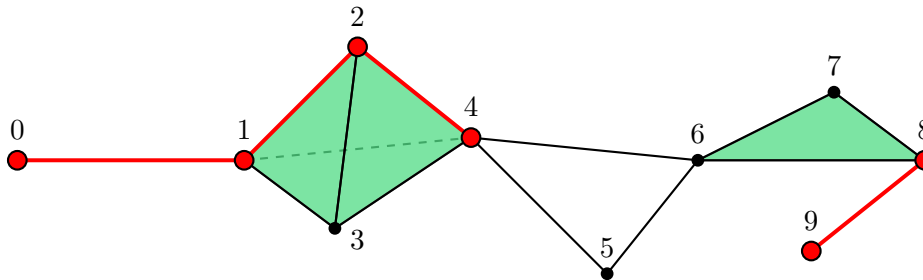


Figura 2.12: Ejemplo de 1-cadena

- En la figura 2.13 se muestra en rojo la 2-cadena $c = [1, 2, 3] + [2, 3, 4] + [6, 7, 8]$.

Dadas dos p -cadenas $c = \sum a_i \sigma_i$ y $c' = \sum b_i \sigma_i$, se define su suma como

$$c + c' = \sum (a_i + b_i) \sigma_i.$$

Las p -cadenas con la operación suma $+$ forman el *grupo de p -cadenas* denotado por $(C_p, +)$, pero como la operación se sobrentiende, se suele nombrar como $C_p = C_p(K)$.

Este grupo es un grupo abeliano, y como en nuestro caso los coeficientes están tomados en el cuerpo \mathbb{Z}_2 , $C_p(K)$ es un espacio vectorial sobre \mathbb{Z}_2 . Fijado $p \in \mathbb{Z}$, una base

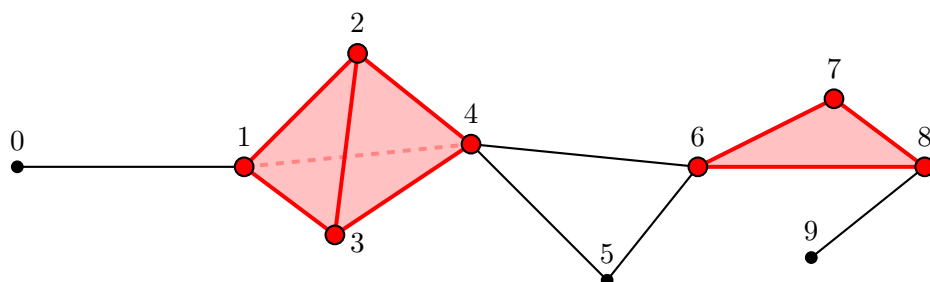


Figura 2.13: Ejemplo de 2-cadena

del espacio vectorial $C_p(K)$ es el conjunto $\{\sigma_i^p \mid i = 1, \dots, s_p\}$ formado por los símlices de dimensión p de K . Como consecuencia $C_p(K) = \{0\}$, siendo $0 = \sum 0 \cdot \sigma_i$, si $p < 0$ ó $p > \dim(K)$.

Operador borde

Para poder relacionar estos grupos definiremos el *operador borde*, así pues, partiremos con la definición del borde de un símlice.

Definición 2.1.28. Sea p un número entero y $\sigma \in K$ un p -símlice $\sigma = [v_0, v_1, \dots, v_p]$ se define su *borde*, $\partial_p \sigma$, como la suma formal de sus caras $(p-1)$ -dimensionales, es decir,

$$\partial_p \sigma = \sum_{j=0}^p [v_0, \dots, \hat{v}_j, \dots, v_p]$$

donde \hat{v}_j denota que v_j se omite.

En general, dada una p -cadena $c = \sum a_i \sigma_i$, se define su borde mediante la extensión lineal como $\partial_p c = \sum_{j=0}^p a_i \partial_p \sigma_i$. Como consecuencia, el borde define una aplicación lineal $\partial_p : C_p \rightarrow C_{p-1}$ entre espacios vectoriales de cadenas denominada *operador borde*. Para simplificar la notación suele omitirse el subíndice p del operador borde, ya que siempre coincide con la dimensión de la cadena a la que se le aplica.

Ejemplo 2.1.3. Sea la 2-cadena $c = [0, 1] + [4, 5]$, entonces el borde de c es:

$$\partial c = \partial[0, 1] + \partial[4, 5] = [0] + [1] + [4] + [5].$$

Ciclos y bordes

Distinguiremos dos tipos de cadenas, las cuales usaremos para poder definir los grupos de homología.

Definición 2.1.29. Diremos que una p -cadena c es un p -ciclo si

$$\partial c = 0$$

o, equivalentemente, si $c \in \ker \partial$.

Debido a que ∂ conmuta con la suma $+$, el conjunto de p -ciclos $Z_p = \ker \partial_p$ es un subgrupo (subespacio vectorial en nuestro caso) de C_p .

Ejemplo 2.1.4. Veremos que geoméricamente los p -ciclos representan ciclos en el complejo simplicial. Estos a su vez pueden ser agujeros de dimensión p . En la figura

2.1. Conocimientos previos y definiciones

2.14 se muestra en rojo el 1-ciclo $[4, 5] + [4, 6] + [5, 6]$, el cual es un agujero. Mientras que en azul se representa el 1-ciclo $[6, 7] + [6, 8] + [7, 8]$, que no es un agujero.

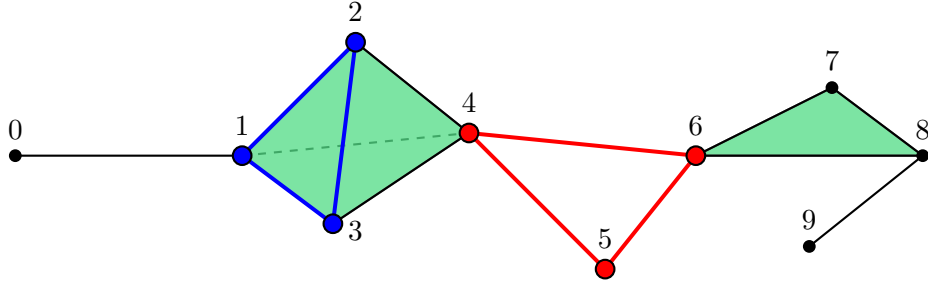


Figura 2.14: Ejemplos de 1-ciclos

Definición 2.1.30. Diremos que una p -cadena c es un p -borde si existe una $(p+1)$ -cadena c' tal que

$$\partial c' = c$$

o, equivalentemente, si $c \in \text{im } \partial_{p+1}$.

Debido a que ∂ conmuta con la suma $+$, el conjunto de p -bordes $B_p = \text{im } \partial_{p+1}$ es un subespacio vectorial de C_p .

Ejemplo 2.1.5. El 1-ciclo que habíamos destacado en azul en la figura 2.14 es un 1-borde.

Probaremos que los p -bordes son p -ciclos, como ocurre en el ejemplo. Para ello enunciaremos el siguiente lema.

Lema 2.1.7 (Lema fundamental de la homología [2]). $\partial_p \partial_{p+1} c = 0$ para todo entero p y toda $(p+1)$ -cadena c .

Se sigue que B_p es un subespacio vectorial de Z_p , es decir $B_p \subset Z_p$. Además, podemos definir el *complejo de cadenas* asociado a un complejo simplicial K como la sucesión de grupos de cadenas conectados por los operadores borde

$$\dots \xrightarrow{\partial_{p+2}} C_{p+1} \xrightarrow{\partial_{p+1}} C_p \xrightarrow{\partial_p} C_{p-1} \xrightarrow{\partial_{p-1}} \dots$$

La figura 2.15 muestra esta relación entre el grupo de cadenas C_p , el grupo de ciclos Z_p y el grupo de bordes B_p ; y sus conexiones generadas por el operador borde.

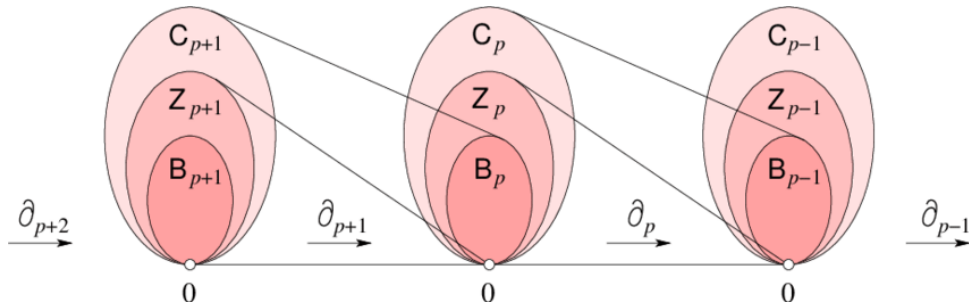


Figura 2.15: Complejo de cadenas representando el grupo de cadenas, el grupo de ciclos y el grupo de bordes. Fuente: [2]

Grupos de homología simplicial

La idea general de los grupos de homología es poder encontrar los agujeros a partir de los ciclos. Para ello tendremos que “descartar” aquellos ciclos que son bordes. Es por esto por lo que cocientaremos el grupo de los ciclos por el grupo de bordes, ya que así todos los bordes serán triviales en homología.

Definición 2.1.31. Dado un complejo simplicial K se define su *grupo de homología p -dimensional* como el cociente

$$H_p(K) = \frac{Z_p}{B_p}.$$

El *número de Betti p -dimensional* $\beta_p(K)$ como la dimensión de $H_p(K)$.

Luego los elementos $z \in H_p = H_p(K)$ son de la forma $z = c + B_p$ con $c \in Z_p$, donde $c + B_p$ es la *clase lateral* de B_p en Z_p . Dos ciclos $c_1, c_2 \in Z_p$ representan la misma *clase de homología* $z \in H_p$ si y sólo si $z = c_1 + B_p = c_2 + B_p$; lo que equivale a que $(c_1 - c_2) \in B_p$.

Definición 2.1.32. Diremos que dos ciclos $c_1, c_2 \in Z_p$ son *homólogos* si existe $b \in B_p$ tal que

$$c_1 = c_2 + b.$$

Como $H_p(K)$ es un grupo finito, por el *teorema de Lagrange* sabemos que el número de clases de homología es

$$\text{ord } H_p(K) = \frac{\text{ord } Z_p}{\text{ord } B_p}.$$

Además, como Z_p, B_p y H_p son espacios vectoriales sobre \mathbb{Z}_2 se sigue que

$$\beta_p = \dim H_p = \dim Z_p - \dim B_p.$$

Aplicaciones inducidas

Veremos que una aplicación simplicial entre dos complejos simpliciales lleva ciclos a ciclos y bordes a bordes. Luego, esta aplicación induce una aplicación entre grupos de homología.

Sean K y L complejos simpliciales y $f : K \rightarrow L$ una aplicación simplicial. Para cada p -símplice σ^p se define

$$f_{\#}(\sigma^p) = \begin{cases} f(\sigma^p) & \text{si } \dim f(\sigma^p) = p \\ 0 & \text{en otro caso} \end{cases}$$

Puesto que los símlices forman una base de los espacios vectoriales $C_p(K)$ y $C_p(L)$, mediante una extensión lineal se obtiene una aplicación lineal $f_{\#} : C_p(K) \rightarrow C_p(L)$.

Propiedad 2.1.2 ([2]). Sean ∂_K y ∂_L los operadores borde de K y L respectivamente. Entonces $f_{\#} \circ \partial_K = \partial_L \circ f_{\#}$.

La propiedad anterior garantiza que $f_{\#}(Z_p(K)) \subset Z_p(L)$ y $f_{\#}(B_p(K)) \subset B_p(L)$. Por tanto $f_{\#}$ induce una aplicación lineal $f_* : H_p(K) \rightarrow H_p(L)$, que denominaremos *homomorfismo inducido por f* .

Utilizando aproximaciones simpliciales podemos ver que aplicaciones continuas entre poliedros inducen aplicaciones lineales en homología. Para ello definiremos el siguiente operador:

2.1. Conocimientos previos y definiciones

Definición 2.1.33. Sea K un complejo simplicial y consideremos la aplicación $\lambda : C_p(K) \rightarrow C_p(\text{Sd}^n K)$ definida sobre los p -simplices como

$$\lambda_p(\sigma^p) = \sum_{\tau^p \in \text{Sd}^n \sigma^p} \tau^p.$$

La aplicación λ_p se denomina *operador subdivisión*.

Sean K y L complejos simpliciales y $f : |K| \rightarrow |L|$ una aplicación continua y $g : \text{Sd}^n K \rightarrow L$ una aproximación simplicial de f . Se define el *homomorfismo inducido* por la aplicación f como la aplicación lineal $f_* : H_p(K) \rightarrow H_p(L)$ dada por

$$f_* = g_* \circ \lambda_{p*}.$$

Donde g_* es el homomorfismo inducido por g y $\lambda_{p*} : H_p(K) \rightarrow H_p(\text{Sd}^n K)$ es el isomorfismo inducido por λ_p .

Teorema 2.1.8. Sean K y L dos complejos simpliciales y $f : |K| \rightarrow |L|$ un homeomorfismo. Entonces $f_* : H_p(K) \rightarrow H_p(L)$ es un isomorfismo para todo p .

Propiedades topológicas

En esta sección veremos algunas propiedades topológicas que podemos obtener del estudio de la homología de un complejo simplicial.

Definición 2.1.34. La característica de Euler de un complejo simplicial K es

$$\chi(K) = \sum_{p=0}^{\dim K} (-1)^p s_p$$

donde $s_p = \dim C_p(K)$.

La podremos calcular a partir de los números de Betti:

Teorema 2.1.9 ([2]). $\chi(K) = \sum_{p=0}^{\dim K} (-1)^p \beta_p(K)$.

Por el teorema 2.1.8 sabemos que si los espacios subyacentes de dos complejos simpliciales son homeomorfos, entonces sus grupos de homología son isomorfos, y por tanto tendrán la misma dimensión.

Corolario. Sean K y L dos complejos simpliciales tales que $|K| \approx |L|$. Entonces, $\chi(K) = \chi(L)$.

Uno de los valores más importantes que obtenemos al calcular los grupos de homología son sus correspondientes números de Betti, ya que estos nos darán mucha información sobre el espacio subyacente.

Teorema 2.1.10. Sea K un complejo simplicial. Entonces $\beta_0(K)$ coincide con el número de componentes conexas de $|K|$.

Corolario. $|K|$ es conexo si y sólo si $\beta_0(K) = 1$.

El Teorema de dualidad de Alexander [2] nos permite interpretar los números de Betti de un poliedro contenido en \mathbb{R}^3 :

- $\beta_0(K)$ nos indica el número de componentes conexas.
- $\beta_1(K)$ nos indica el número de túneles.
- $\beta_2(K)$ nos indica el número de cavidades.

Homología singular

Hay una gran variedad de teorías de homología en topología. La homología que hemos definido, denominada *homología simplicial*, supone que nuestro espacio está expresado como el poliedro subyacente de un complejo simplicial. La *homología singular* generaliza la homología simplicial y permite estudiar otros espacios no triangulables [8]. Este tipo de homología tiene la ventaja que existe para cualquier espacio topológico y que facilita definir conceptos como las aplicaciones inducidas. Sin embargo, los grupos de cadenas singulares tienen dimensión infinita, lo que hace que no sea una buena opción desde el punto de vista computacional. Cabe destacar que, sobre poliedros ambas teorías coinciden [3].

Además, para el *teorema de estabilidad* no nos hará falta hacer uso de la homología singular, ya que se parte de la hipótesis de que el espacio es triangulable.

2.1.4. Persistencia

Introduciremos el concepto de persistencia primero para funciones de una variable. Después veremos en el caso de funciones morse, luego profundizaremos en el caso de los complejos simpliciales y por último para funciones tame. En esta sección seguiremos [3] como referencia.

Funciones reales de una variable

Sea $f : \mathbb{R} \rightarrow \mathbb{R}$ una función suave. Recordemos que x es un *punto crítico* y $f(x)$ un *valor crítico de f* si $f'(x) = 0$. Además, un punto crítico x es *no degenerado* si $f''(x) \neq 0$. Así pues, supongamos que f sólo contiene puntos críticos no degenerados con valores críticos distintos.

Si consideramos el *conjunto de subnivel* $\mathbb{R}_t = f^{-1}(-\infty, t]$ para cada $t \in \mathbb{R}$, entonces veremos que a medida que incrementemos t , el número de componentes conexas de \mathbb{R}_t permanecerá constante hasta que pasemos por un t_0 valor crítico de f . Como podemos ver en la figura 2.16, cuando pasamos por un mínimo local se crea una nueva componente conexa y cuando pasamos por un máximo local se combinan dos componentes conexas en una.

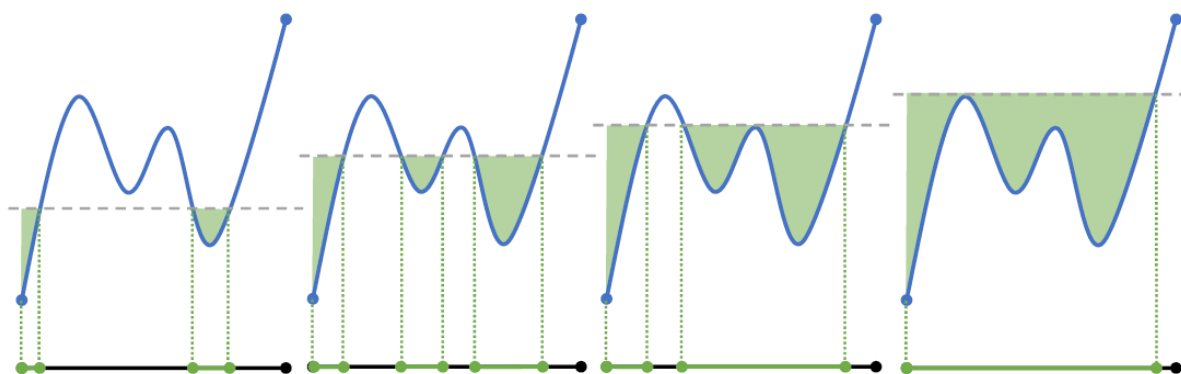


Figura 2.16: Componentes conexas en \mathbb{R} en las diferentes filtraciones. Fuente: [9]

Los puntos críticos de f se emparejan de la siguiente forma:

2.1. Conocimientos previos y definiciones

- A. Cuando aparece una nueva componente conexa, diremos que el mínimo local que lo crea *representa* esa componente.
- B. Cuando pasamos por un máximo local y se juntan dos componentes, emparejamos el máximo con el mayor (el más joven) de los dos mínimos locales que representan dichas componentes. El otro mínimo (el más antiguo) pasa a ser el representante de la nueva componente resultante de juntar las dos anteriores.

Cuando los puntos x_1 y x_2 se emparejan siguiendo este método, definimos la *persistencia* del par como $f(x_2) - f(x_1)$. Esta persistencia es codificada a través del *diagrama de persistencia*, representando cada par con el punto $(f(x_1), f(x_2))$, como se puede ver en la figura 2.17. Se puede observar que todos los puntos se encontrarán por encima de la diagonal $y = x$, y que la persistencia es la distancia vertical de un punto a la diagonal. Por razones que explicaremos después se añadirán los puntos de la diagonal al diagrama de persistencia.

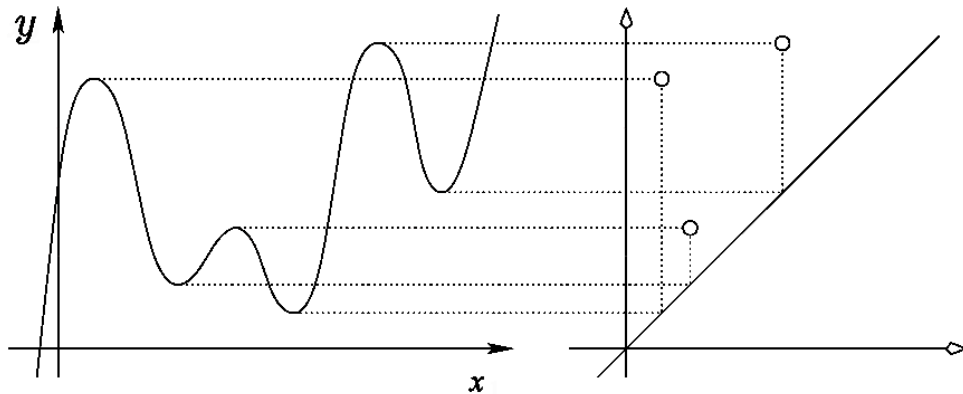


Figura 2.17: Emparejamiento de los puntos críticos de la función de la función de la izquierda representados como puntos en el diagrama de persistencia de la derecha. Fuente: [3]

Funciones Morse

Vamos a generalizar lo visto con funciones de una variable en \mathbb{R} a funciones suaves sobre *variedades diferenciables* con ciertas propiedades que explicaremos más adelante. Primero recordaremos que son las variedades diferenciables.

Definición 2.1.35. Una *variedad diferenciable* un espacio topológico \mathbb{M} que satisface:

- A. \mathbb{M} es Hausdorff (T_2).
- B. \mathbb{M} es segundo numerable, es decir, su topología tiene una base numerable.
- C. Todo punto de \mathbb{M} posee un entorno abierto difeomorfo a \mathbb{R}^n .

Sea $f : \mathbb{M} \rightarrow \mathbb{R}$ una aplicación suave. En este caso, un *punto crítico* es un punto $p \in \mathbb{M}$ tal que $\frac{\partial f}{\partial x_i}(p) = 0$ para $i = 1, \dots, n$. Un punto crítico p es *no degenerado* si la matriz Hessiana de las segundas derivadas parciales,

$$(H_f)_{i,j} = \left(\frac{\partial^2 f}{\partial x_i \partial x_j} \right)_{i,j}$$

es no singular. Si p es un punto crítico no degenerado se define su *índice* como el número de autovalores negativos de la matriz Hessiana en p .

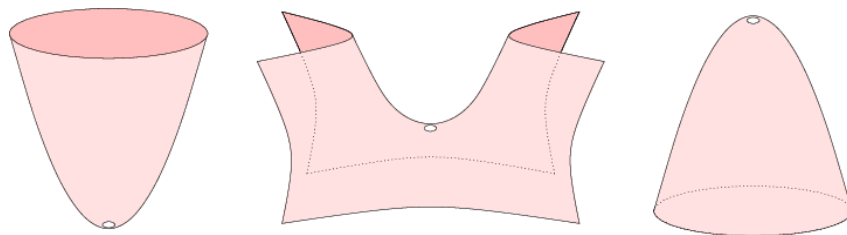


Figura 2.18: De izquierda a derecha tenemos: un punto crítico no degenerado de índice 0, 1 y 2. Fuente: [3]

Definición 2.1.36. Sea $f : \mathbb{M} \rightarrow \mathbb{R}$ una aplicación diferenciable. Diremos que f es una *función Morse* si todos sus puntos críticos son no degenerados y tienen distintos valores críticos.

Se puede demostrar que las funciones Morse poseen un número finito de puntos críticos. Elegimos los valores regulares $t_0 < t_1 < \dots < t_m$ tal que existe un único punto crítico $p_i \in (t_i, t_{i+1})$ para todo $i = 0, \dots, m-1$. Sea $\mathbb{M}_j = f^{-1}(-\infty, t_j]$ el *conjunto de subnivel* que contiene los primeros j puntos críticos.

Cuando pasamos de \mathbb{M}_{j-1} a \mathbb{M}_j la homología (singular) puede cambiar de dos formas distintas:

- A) H_p incrementa la dimensión en uno, es decir, $\beta_p(\mathbb{M}_j) = \beta_p(\mathbb{M}_{j-1}) + 1$.
- B) H_{p-1} disminuye la dimensión en uno, es decir, $\beta_{p-1}(\mathbb{M}_j) = \beta_{p-1}(\mathbb{M}_{j-1}) - 1$.

Donde p es el índice del j -ésimo punto crítico. En el primer caso denotaremos a ese punto crítico como *punto crítico positivo* y en el segundo como *punto crítico negativo*.

La persistencia nos dará un emparejamiento de algunos de los puntos críticos positivos de índice p con puntos críticos negativos de índice $p+1$. La idea es determinar el “momento” en el que nace una clase de homología y cuando muere, de forma que la persistencia será la diferencia de los tiempos. Para ello haremos uso de funciones entre grupos de homología inducidos por la inclusión de los conjuntos de subnivel $\mathbb{M}_i \subseteq \mathbb{M}_j$ para $i \leq j$. Definiremos de forma más precisa los conceptos de nacimiento y muerte de una clase de homología de la siguiente forma:

- Una clase de homología α *nace* en \mathbb{M}_i si no existe en \mathbb{M}_{i-1} .
- Una clase de homología α nacida en \mathbb{M}_i *morirá al entrar en \mathbb{M}_j* si la imagen de la función inducida por $\mathbb{M}_{i-1} \subseteq \mathbb{M}_{j-1}$ no contiene a la imagen de α pero la imagen de la función inducida por $\mathbb{M}_{i-1} \subseteq \mathbb{M}_j$ si. Siguiendo lo que vimos en funciones de una variable, lo que ocurre es que al entrar en \mathbb{M}_j se junta la clase α con una clase que ya existía en \mathbb{M}_{i-1} .

Si α nace en \mathbb{M}_i y muere al entrar \mathbb{M}_j , entonces emparejaremos sus puntos críticos correspondientes, x e y , y diremos que su persistencia es $j - i$ ó $f(y) - f(x)$ según convenga. Esta persistencia es codificada a través de los *diagramas de persistencia*, $\text{Dgm}_p(f)$, representando cada emparejamiento de un punto crítico positivo de índice p con un punto crítico negativo de índice $p+1$ añadiendo el punto $(f(x), f(y))$ al diagrama. Al igual que hicimos en el caso de funciones reales de una variable, añadiremos

los puntos de la diagonal en el diagrama de persistencia.

Funciones tame

Se puede comprobar que las funciones Morse sobre variedades diferenciables limitarán demasiado para algunas aplicaciones. Es por ello que consideraremos un tipo de función $f : \mathbb{X} \rightarrow \mathbb{R}$, donde f y \mathbb{X} cumplen una serie de propiedades menos restrictivas. Empezaremos extendiendo la noción de punto crítico de la siguiente forma:

Definición 2.1.37. Sea \mathbb{X} un espacio topológico, f una función real en \mathbb{X} y $\mathbb{X}_t = f^{-1}(-\infty, t]$ el conjunto de subnivel definido para el valor t . Un *valor crítico de homología* de f es un número real a para el cual existe un entero k tal que para todo $\epsilon > 0$ lo suficientemente pequeño, el homomorfismo $H_k(\mathbb{X}_{a-\epsilon}) \rightarrow H_k(\mathbb{X}_{a+\epsilon})$ ¹ inducido por la inclusión, $\mathbb{X}_{a-\epsilon} \subseteq \mathbb{X}_{a+\epsilon}$, no es un isomorfismo.

En otras palabras, los valores críticos de homología son los niveles en los cuales la homología de los conjuntos de subnivel cambia. Como ya hemos visto, en el caso de las funciones Morse, estos puntos críticos de homología coinciden con los valores críticos de la función.

Definición 2.1.38. Una función $f : \mathbb{X} \rightarrow \mathbb{R}$ es *tame* si los grupos de homología de cada conjunto de subnivel son finito-dimensionales y f posee un número finito de valores críticos de homología.

En particular, las funciones Morse sobre variedades compactas son funciones tame, ya que la compacidad y el carácter aislado de los puntos críticos garantizan que estas funciones posean un número finito de puntos críticos. Para simplificar la notación, para cada entero k fijo, escribimos $F_x = H_k(f^{-1}(-\infty, x])$, y para $x < y$, denotamos como $f_x^y : F_x \rightarrow F_y$ la aplicación lineal inducida por la inclusión $\mathbb{X}_x \subseteq \mathbb{X}_y$. Una vez establecida la notación, probaremos el lema 2.1.11, que nos será de gran ayuda para la demostración del *teorema de estabilidad*.

Propiedad 2.1.3. La familia de aplicaciones $(f_x^y)_{x \leq y}$ satisface las siguientes propiedades:

- $f_x^x = \text{id}_{F_x}$.
- $f_m^y \circ f_x^m = f_x^y$, con $x \leq m \leq y$.

Lema 2.1.11 (Lema del valor crítico). Si un intervalo cerrado $[x, y]$ no contiene ningún valor crítico de homología de f , entonces f_x^y es un isomorfismo para todo entero k .

Demostración. Sea $m = (x + y)/2$, tenemos que $f_x^y = f_m^y \circ f_x^m$. Supongamos que f_x^y no es un isomorfismo. Entonces, al menos una de las funciones f_m^y y f_x^m no es un isomorfismo.

Repetiendo este argumento sobre las funciones no isomorfas de la composición obtenemos una sucesión de intervalos encajados cerrados y acotados, $I_n = [x_n, y_n]$, con

$$\lim_{n \rightarrow \infty} |y_n - x_n| = 0 \text{ y tal que } f_{x_n}^{y_n} \text{ no es un isomorfismo para todo } n \in \mathbb{N}$$

por lo que, aplicando el principio de intervalos encajados en \mathbb{R} , sabemos que su intersección es un punto $a \in \mathbb{R}$, que verifica que $f_{a-\epsilon}^{a+\epsilon}$ no es un isomorfismo para todo $\epsilon > 0$.

¹En esta sección consideraremos la *homología singular* como teoría de homología, dado que los espacios topológicos \mathbb{X} no requieren ser triangulables.

Luego, el punto a es un valor crítico de homología en $[x, y]$, contradiciendo nuestra hipótesis inicial. \square

Definición 2.1.39. Sea $f_x^y : F_x \rightarrow F_y$ la aplicación lineal inducida por la inclusión $\mathbb{X}_x \subseteq \mathbb{X}_y$. Se definen los *grupos de homología persistente* como la imagen de F_x en F_y de la aplicación f_x^y , es decir,

$$F_x^y = \text{im } f_x^y.$$

Los correspondientes *números de Betti persistentes* se definen como los rangos de estos grupos, es decir, $\beta_x^y = \dim F_x^y$, para todo $-\infty \leq x \leq y \leq +\infty$.

Por convención, se establece que $F_x^y = \{0\}$ cuando x ó y son infinito. El grupo de homología persistente consiste de las clases que han nacido antes de x y siguen vivas en y .

Observación. Si analizamos las aplicaciones f_x^y , observamos que el $\ker f_x^y$ son aquellos elementos $\gamma \in F_x$ tales que $f_x^y(\gamma) = 0$. Esto significa que si c es un ciclo representando a γ , $c \in B_k(\mathbb{X}_y)$. Como consecuencia

$$\ker f_x^y = \frac{Z_k(\mathbb{X}_x) \cap B_k(\mathbb{X}_y)}{B_k(\mathbb{X}_x)}$$

para cada dimensión k fija.

Sea $f : \mathbb{X} \rightarrow \mathbb{R}$ una función tame, $(a_i)_{i=1..n}$ sus valores críticos homológicos y se considera la sucesión entrelazada $(b_i)_{i=0..n}$, tal que $b_{i-1} < a_i < b_i$ para $1 \leq i \leq n$. Para capturar la homología a lo largo de todo el proceso hacemos $b_{-1} = a_0 = -\infty$ y $b_{n+1} = a_{n+1} = +\infty$. Entonces,

Definición 2.1.40. Se define la multiplicidad del par (a_i, a_j) como

$$\mu_i^j = \beta_{b_{i-1}}^{b_j} - \beta_{b_i}^{b_j} + \beta_{b_i}^{b_{j-1}} - \beta_{b_{i-1}}^{b_{j-1}}, \text{ para todo } i, j \in \mathbb{Z} \text{ tal que } 0 \leq i < j \leq n+1.$$

Podemos visualizar la multiplicidad, μ_i^j , como se muestra en la figura 2.19. Donde, considerando β_x^y como una función sobre el plano real extendido $\overline{\mathbb{R}}^2$, donde $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$; entonces, μ_i^j es la suma alternada de los números de Betti persistentes en las esquinas del cuadrado $[b_{i-1}, b_i] \times [b_{j-1}, b_j]$.

Observación. Si x y x' se encuentran dentro del intervalo (a_i, a_{i+1}) , e y e y' en el intervalo (a_{j-1}, a_j) , entonces $\beta_x^y = \beta_{x'}^{y'}$. Este resultado se sigue como consecuencia del *Lema del valor crítico*, que garantiza que F_x^y y $F_{x'}^{y'}$ son isomorfos.

Definición 2.1.41. El *diagrama de persistencia* $\text{Dgm}(f) \subset \overline{\mathbb{R}}^2$ de f es el multiconjunto de puntos (a_i, a_j) con multiplicidad μ_i^j para todo $0 \leq i < j \leq n+1$, unión los puntos de la diagonal, $\Delta = \{(x, y) \in \overline{\mathbb{R}}^2 \mid y = x\}$, con multiplicidad infinito.

Denotaremos por $\#(A)$ la *multiplicidad total* de un multiconjunto A , que, por definición es la suma de las multiplicidades de los elementos de A . Por tanto, la multiplicidad total del diagrama de persistencia menos la diagonal es

$$\#(\text{Dgm}(f) \setminus \Delta) = \sum_{i < j} \mu_i^j.$$

Esta multiplicidad se denomina *tamaño del diagrama de persistencia*.

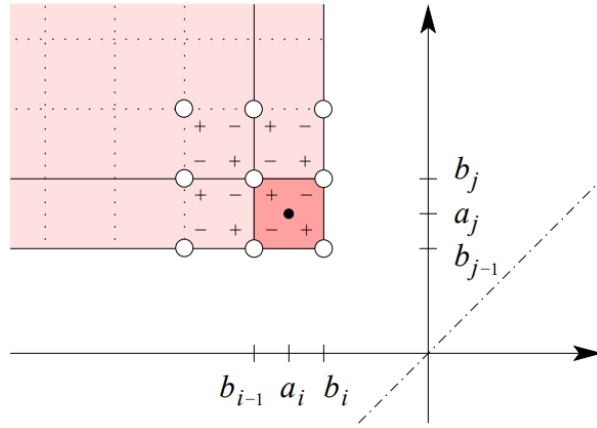


Figura 2.19: La multiplicidad del punto (a_i, a_j) es la suma alternada de los números de Betti persistentes en las esquinas del cuadrado $[b_{i-1}, b_i] \times [b_{j-1}, b_j]$. Fuente: [4]

Denotaremos el cuadrante superior izquierda cerrado con vértice en el punto (x, y) como $Q_x^y = [-\infty, x] \times [y, \infty]$.

Lema 2.1.12 (Lema del k -Triángulo). *Sea f una función tame y $x < y$ diferentes de los valores críticos homológicos de f . Entonces, la multiplicidad total del diagrama de persistencia en el cuadrante superior izquierdo con vértice (x, y) es*

$$\#(\text{Dgm}(f) \cap Q_x^y) = \beta_x^y.$$

Demostración. Podemos asumir sin pérdida de generalidad que $x = b_i$ y $y = b_{j-1}$. Por definición, la multiplicidad total en el cuadrante superior izquierdo es igual a la suma de las multiplicidades de los puntos contenidos en dicho cuadrante, luego

$$\#(\text{Dgm}(f) \cap Q_x^y) = \sum_{k \leq i} \sum_{l > j} \mu_k^l = \sum_{k \leq i} \sum_{l > j} (\beta_{b_{k-1}}^{b_l} - \beta_{b_k}^{b_l} + \beta_{b_k}^{b_{l-1}} - \beta_{b_{k-1}}^{b_{l-1}}).$$

Como se muestra en la figura 2.19, cuando se suman las multiplicidades, ocurre la cancelación entre signos positivos y negativos de las esquinas de los cuadrados. Entonces:

$$\begin{aligned} \#(\text{Dgm}(f) \cap Q_x^y) &= \beta_{b_{-1}}^{b_{i+1}} - \beta_{b_i}^{b_{i+1}} + \beta_{b_i}^{b_{j-1}} - \beta_{b_{-1}}^{b_{j-1}} = \\ &= \beta_{-\infty}^{+\infty} - \beta_{b_i}^{+\infty} + \beta_{b_i}^{b_{j-1}} - \beta_{-\infty}^{b_{j-1}} = \beta_{b_i}^{b_{j-1}} = \beta_x^y \end{aligned}$$

ya que $F_x^y = \{0\}$ cuando x ó y son infinito, y por lo tanto su dimensión, es decir, su número de Betti persistente, es cero. \square

Este lema nos garantiza que el diagrama de persistencia codifica toda la información sobre los grupos de homología persistente [2].

Persistencia en complejos simpliciales

Veremos que podemos particularizar la persistencia vista para funciones tame a complejos simpliciales. Para ello utilizaremos las *filtraciones* de un complejo simplicial como conjuntos de subnivel y haremos uso de la *homología simplicial* como teoría de homología.

Definición 2.1.42. Sea K un complejo simplicial y $f : K \rightarrow \mathbb{R}$ una función. Se dice que, f es *monótona* si $f(\sigma) \leq f(\tau)$ si σ es una cara de τ .

La monotonía de f garantiza que para cada $a \in \mathbb{R}$, el conjunto de subnivel $K(a) = f^{-1}(-\infty, a]$ es un subcomplejo de K .

Definición 2.1.43. Sean $a_1 < a_2 < \dots < a_n$ los valores que toma la función en los simplices y sea $a_0 = -\infty$. Entonces f induce una *filtración*

$$\emptyset = K_0 \subseteq K_1 \subseteq \dots \subseteq K_n = K, \text{ con } K_i = K(a_i).$$

De esta forma, al igual que vimos con las funciones Morse, una clase de homología α nace en K_i si no está en la imagen de la función inducida por la inclusión $K_{i-1} \subseteq K_i$. Además, una clase α que nace en K_i muere al entrar en K_j si la imagen de la función inducida por $K_{i-1} \subseteq K_{j-1}$ no contiene la imagen de α , pero la imagen de la función inducida por $K_{i-1} \subseteq K_j$ sí.

Introduciremos los grupos de homología persistente, reduciendo la notación de la siguiente forma: $F_i = F_{b_i}$, $F_i^j = F_{b_i}^{b_j}$ y $\beta_i^j = \beta_{b_i}^{b_j}$. Así podemos redefinir la noción de nacimiento y muerte de una clase de homología como sigue

- Una clase $\gamma \in F_i$ *nace* en K_i si $\gamma \notin F_{i-1}^i$.
- Una clase $\gamma \in F_i$ nacida en K_i *muere* al entrar en K_j si $f_i^{j-1}(\gamma) \notin F_{i-1}^{j-1}$, pero $f_i^j(\gamma) \in F_{i-1}^j$.

Definición 2.1.44. Sea γ una clase de homología que nace en K_i y muere al entrar en K_j . Se define la *persistencia* de γ como $\text{pers}(\gamma) = a_j - a_i$. Asimismo, la diferencia $j - i$ se denomina *índice de persistencia* de la clase γ . Si una clase γ nace en K_i pero nunca muere, entonces diremos que su persistencia, al igual que su índice, es infinito.

Siguiendo esta notación, se define la multiplicidad como

$$\mu_i^j = (\beta_i^{j-1} - \beta_i^j) - (\beta_{i-1}^i - \beta_{i-1}^{i-1}).$$

Donde β_i^{j-1} se puede interpretar como el número de clases de homología que están vivas en K_i y siguen vivas en K_{j-1} . Por lo tanto, la primera diferencia de la igualdad se interpreta como el número de clases independientes que están vivas en K_i y mueren en K_j , mientras que la segunda diferencia son el número de clases independientes que nacen antes de K_i y mueren en K_j . En conclusión, la multiplicidad, μ_i^j , se interpreta como el número de clases de homología que nacen en K_i y mueren en K_j .

Cada punto (a_i, a_j) representa μ_i^j clases de homología independientes cuya persistencia coincide con la distancia del punto (a_i, a_j) a su proyección vertical sobre la diagonal Δ . Por razones técnicas, los puntos de la diagonal se añaden al diagrama de persistencia con multiplicidad infinito.

Adicionalmente de los diagramas de persistencia, podemos codificar la información sobre la homología persistente a través de los denominados *códigos de barras*. Estas representaciones se pueden obtener a partir del diagrama de persistencia dibujando por cada punto (a_i, a_j) con $a_i < a_j$ de dicho diagrama μ_i^j intervalos semiabiertos $[a_i, a_j)$, como se muestra en la figura 2.20.

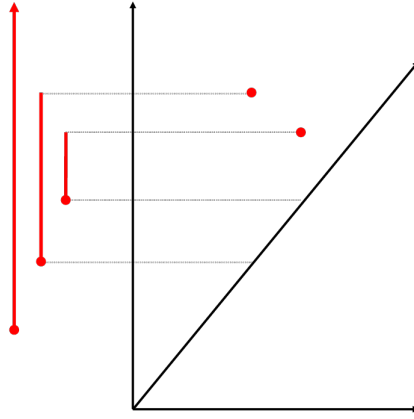


Figura 2.20: Código de barras asociado a un diagrama de persistencia. Fuente: [10]

Funciones PL

Un caso especial de las funciones tame son las *funciones lineales a trozos* (en inglés: *piecewise linear function*) que asocian valores reales al espacio subyacente de un complejo simplicial.

Definición 2.1.45. Sea K un complejo simplicial con valores reales asignados en todos sus vértices. Se define la función lineal a trozos $f : |K| \rightarrow \mathbb{R}$ como la extensión lineal de los valores de los vértices sobre los símlices, es decir,

$$f(x) = \sum_i b_i(x) f(u_i)$$

donde u_i son los vértices de K y $b_i(x)$ son las coordenadas baricéntricas de x .

Por simplicidad se asume que $f|_{\text{Vert } K}$ es inyectiva. Reindexando los vértices de forma que $f(u_1) < f(u_2) < \dots < f(u_n)$, definimos K_i como el subcomplejo definido por los primeros i vértices.

Definición 2.1.46. La *estrella inferior* de un vértice $u_i \in \text{Vert } K$ se define como el subconjunto de símlices para los cuales u_i es el vértice de mayor valor de f :

$$\text{St}_- u_i = \{\sigma \in \text{St } u_i \mid x \in \sigma \Rightarrow f(x) \leq f(u_i)\}.$$

Como ocurría con la estrella, la estrella inferior generalmente no es un subcomplejo. Añadiendo las caras restantes a los símlices en $\text{St}_- u_i$, obtenemos la *estrella inferior cerrada* $\overline{\text{St}}_- u_i$, que es el menor subcomplejo de K que contiene a $\text{St}_- u_i$. Como f es inyectiva en sus vértices, cada símlice tiene un único vértice con valor máximo, y por tanto pertenece a una única estrella inferior. Luego, K_i es la unión de las primeras i estrellas inferiores; obteniendo la siguiente filtración de K :

Definición 2.1.47. Sea K un complejo simplicial y $f_i : |K| \rightarrow \mathbb{R}$ una función PL. Se define la *filtración de K por las estrellas inferiores de f* como la filtración de subcomplejos $\emptyset = K_0 \subseteq K_1 \subseteq \dots \subseteq K_n = K$, donde $K_i = K_{i-1} \cup \overline{\text{St}}_- u_i$.

Esta filtración cumple las siguientes propiedades:

Propiedad 2.1.4 ([2]). K_i tiene el mismo tipo de homotopía que el subnivel $|K|_t = f^{-1}(-\infty, t]$, para todo $f(u_i) \leq t < f(u_{i+1})$.

Propiedad 2.1.5 ([2]). La la variación de la homología en los conjuntos de subnivel $|K|_t = f^{-1}(-\infty, t]$ es la misma que la homología de la filtración por las estrellas inferiores de f .

Propiedad 2.1.6 ([3]). Sea \mathbb{X} un espacio topológico triangulable. Entonces podemos aproximar toda función tame en \mathbb{X} a partir de una función PL en su triangulación.

2.2. Teorema de estabilidad

En esta sección enunciamos y demostraremos el *teorema de estabilidad de los diagramas de persistencia*, siguiendo [4]. Primero estudiaremos la estabilidad para la *distancia de Hausdorff* y, después, reforzaremos el resultado estudiando la estabilidad con la *distancia bottleneck*.

2.2.1. Proposición del teorema

El teorema de estabilidad nos va a garantizar la robustez de los diagramas de persistencia. Dicho de otro modo, que “pequeñas” perturbaciones en las funciones, dan lugar a diagramas de persistencia “cercaños”. Así pues, primero precisaremos el concepto de cercanía entre funciones y diagramas de persistencia.

Sean X e Y dos diagramas de persistencia. Recordamos que X e Y son dos multiconjuntos de puntos del plano extendido $\overline{\mathbb{R}}^2$, constituidos por un número finito de puntos sobre la diagonal, y por los puntos de la diagonal con multiplicidad infinito.

Definición 2.2.1. Sean los puntos $p = (p_1, p_2)$ y $q = (q_1, q_2)$ en $\overline{\mathbb{R}}^2$. Entonces, la distancia infinito entre los puntos es:

$$d_\infty(p, q) = \|p - q\|_\infty = \max\{|p_1 - q_1|, |p_2 - q_2|\}.$$

Definición 2.2.2. Sean $f, g : \mathbb{X} \rightarrow \mathbb{R}$ dos funciones continuas. Entonces, la distancia infinito entre las funciones es:

$$d_\infty(f, g) = \|f - g\|_\infty = \sup_{x \in \mathbb{X}} |f(x) - g(x)|.$$

Definiremos las distancias Hausdorff y bottleneck sobre multiconjuntos (diagramas de persistencia en nuestro caso) de la siguiente forma

Definición 2.2.3. La *distancia Hausdorff* y la *distancia bottleneck* entre X e Y son, respectivamente

$$H(X, Y) = \max \left\{ \sup_{x \in X} \inf_{y \in Y} \|x - y\|_\infty, \sup_{y \in Y} \inf_{x \in X} \|y - x\|_\infty \right\},$$

$$W_\infty(X, Y) = \inf_{\eta: X \rightarrow Y} \sup_{x \in X} \|x - \eta(x)\|_\infty$$

siendo $\eta : X \rightarrow Y$ las biyecciones de X a Y .

Las biyecciones entre dos diagramas de persistencia generan tres tipos de emparejamientos:

- Ambos puntos fuera de la diagonal.

- Un punto fuera de la diagonal y otro en la diagonal.
- Ambos puntos en la diagonal.

Se puede observar que los puntos que determinan en mayor escala la distancia bottleneck son los del primer tipo, y los que menor importancia tienen son los del último tipo, ya que completarán el emparejamiento sin afectar en el cálculo de las distancias.

Observación. Debido a que la distancia bottleneck satisface una restricción adicional respecto a la distancia Hausdorff, es decir, la biyección entre los puntos; entonces, se cumple $H(X, Y) \leq W_\infty(X, Y)$.

Teorema 2.2.1 (Teorema de estabilidad para funciones tame). *Sea \mathbb{X} un espacio topológico triangulable y sea $f, g : \mathbb{X} \rightarrow \mathbb{R}$ dos funciones tame continuas. Entonces, para cada dimensión k , la distancia bottleneck entre los diagramas de persistencia esta acotada por la distancia infinito entre las funciones, es decir,*

$$W_\infty(\text{Dgm}(f), \text{Dgm}(g)) \leq \|f - g\|_\infty.$$

Luego, se garantiza que los diagramas de persistencia son estables bajo perturbaciones de baja amplitud. Este resultado se puede observar gráficamente en la figura 2.21, donde se observa que los valores críticos “superfluos” de la función perturbada definen puntos en el diagrama muy próximos a la diagonal, y los valores críticos “relevantes” definen puntos muy próximos a los puntos del diagrama asociados a la función original.

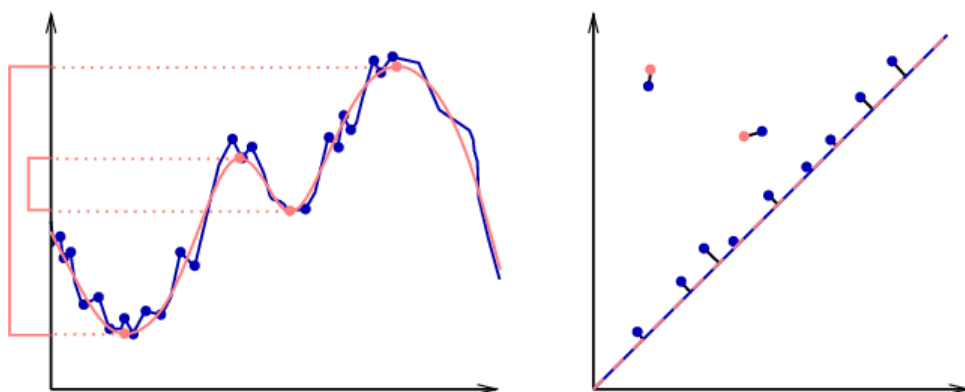


Figura 2.21: A la izquierda se muestran dos funciones cercanas, una con muchos valores críticos y otra con cuatro. A la derecha se muestran los diagramas de persistencia superpuestos, con la biyección que da lugar a la distancia bottleneck. Fuente: [4]

2.2.2. Estabilidad para la distancia Hausdorff

Partiremos de la demostración de la estabilidad con la distancia Hausdorff, que sigue así

Teorema 2.2.2 (Teorema de estabilidad con la distancia Hausdorff para funciones tame). *Sea \mathbb{X} un espacio topológico triangulable y sea $f, g : \mathbb{X} \rightarrow \mathbb{R}$ dos funciones tame continuas. Entonces, para cada dimensión k , la distancia Hausdorff entre los diagramas de persistencia esta acotada por la distancia L_∞ entre las funciones, es decir,*

$$H(\text{Dgm}(f), \text{Dgm}(g)) \leq \|f - g\|_\infty.$$

Relaciones entre cuadrantes superiores izquierdos

Primero estudiaremos la relación entre las multiplicidades de los cuadrantes superiores izquierdos de dos diagramas de persistencia.

Proposición 2.2.1. *Sean $f, g : \mathbb{X} \rightarrow \mathbb{R}$ dos funciones tame continuas. Si denotamos $\epsilon = \|f - g\|_\infty$, entonces $f^{-1}(-\infty, x] \subseteq g^{-1}(-\infty, x + \epsilon]$ para todo $x \in \mathbb{R}$*

Demostración. Sea $y \in \mathbb{X}$ tal que $y \in f^{-1}(-\infty, x] = \{x \in \mathbb{X} \mid f(x) \in (-\infty, x]\}$. Como $\|f - g\|_\infty = \sup_{x \in \mathbb{X}} |f(x) - g(x)| = \epsilon$, entonces $|f(y) - g(y)| < \epsilon$ por lo que $g(y) \in (-\infty, x + \epsilon]$, de donde se sigue que $y \in g^{-1}(-\infty, x + \epsilon]$. \square

Denotamos por $\varphi_x : F_x \rightarrow G_{x-\epsilon}$ a la aplicación inducida por esta inclusión. La inclusión análoga, $g^{-1}(-\infty, x] \subseteq f^{-1}(-\infty, x + \epsilon]$, induce la aplicación $\psi_x : G_x \rightarrow F_{x+\epsilon}$. Sea $b < c$, estas dos aplicaciones dan lugar a los siguientes diagramas conmutativos:

$$\begin{array}{ccc} F_{b-\epsilon} & \xrightarrow{f_{b-\epsilon}^{c+\epsilon}} & F_{c+\epsilon} \\ \varphi_{b-\epsilon} \downarrow & & \uparrow \psi_c \\ G_b & \xrightarrow{g_b^c} & G_c \end{array} \quad \begin{array}{ccc} F_{b+\epsilon} & \xrightarrow{f_{b+\epsilon}^{c+\epsilon}} & F_{c+\epsilon} \\ \psi_b \uparrow & & \uparrow \psi_c \\ G_b & \xrightarrow{g_b^c} & G_b^c \end{array}$$

Los diagramas son conmutativos, dado que están inducidos por los diagramas de inclusiones que son conmutativos.

Del primer diagrama tenemos que $f_{b-\epsilon}^{c+\epsilon} = \psi_c \circ g_b^c \circ \varphi_{b-\epsilon}$. Sea $\xi \in F_{b-\epsilon}^{c+\epsilon} = \text{im } f_{b-\epsilon}^{c+\epsilon}$, de forma que $\xi = f_{b-\epsilon}^{c+\epsilon}(\eta)$ para un $\eta \in F_{b-\epsilon}$. Luego, $\xi = \psi_c(\zeta)$, con $\zeta = g_b^c(\varphi_{b-\epsilon}(\eta)) \in G_b^c$, por tanto $F_{b-\epsilon}^{c+\epsilon} \subseteq \psi_c(G_b^c)$.

Del segundo diagrama, tenemos que $\psi_c(G_b^c) = \psi_c \circ g_b^c(G_b)$, ya que $G_b^c = \text{im } g_b^c = g_b^c(G_b)$. A su vez, se cumple que $\psi_c \circ g_b^c(G_b) = f_{b+\epsilon}^{c+\epsilon} \circ \psi_b(G_b) \subseteq F_{b+\epsilon}^{c+\epsilon}$ de donde se sigue que $\psi_c(G_b^c) \subseteq F_{b+\epsilon}^{c+\epsilon}$.

Así pues, se cumple:

$$F_{b-\epsilon}^{c+\epsilon} \subseteq \psi_c(G_b^c) \subseteq F_{b+\epsilon}^{c+\epsilon}. \quad (2.1)$$

De manera análoga podemos demostrar que se cumple que

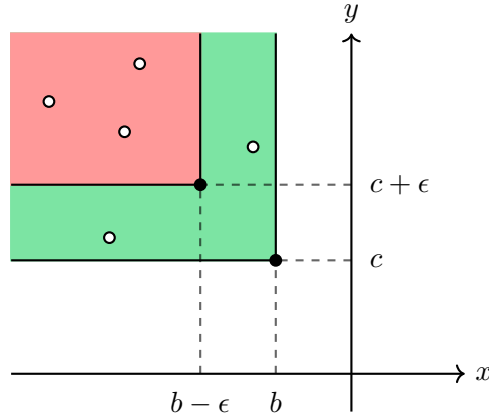
$$G_{b-\epsilon}^{c+\epsilon} \subseteq \varphi_c(F_b^c) \subseteq G_{b+\epsilon}^{c+\epsilon}$$

intercambiando F_x y G_y en los diagramas y sustituyendo las aplicaciones inducidas correctamente.

Recordamos que por la *Fórmula de las dimensiones*[11], si una aplicación $f : U \rightarrow V$ es lineal entonces se cumple que

$$\dim \ker f + \dim \text{im } f = \dim U. \quad (2.2)$$

De la primera inclusión de 2.1 obtenemos que $\dim F_{b-\epsilon}^{c+\epsilon} \leq \dim \psi_c(G_b^c) \stackrel{(2.2)}{\leq} \dim G_b^c$. Aplicando el *Lema del k -Triángulo* a la anterior desigualdad y denotando a los cuadrantes superiores izquierdos como $Q = Q_b^c$ y $Q_\epsilon = Q_{b-\epsilon}^{c+\epsilon}$, se obtiene el siguiente resultado:


 Figura 2.22: Representación del *Lema del cuadrante*

Lema 2.2.3 (Lema del cuadrante). $\#(\text{Dgm}(f) \cap Q_\epsilon) \leq \#(\text{Dgm}(g) \cap Q)$.

Demostración. Si b y c no son valores críticos homológicos de g y $b - \epsilon$, $c + \epsilon$ no son valores críticos homológicos de f , entonces por el Lema del k -Triángulo

$$\#(\text{Dgm}(g) \cap Q) = \beta_b^c = \dim G_b^c \text{ y } \#(\text{Dgm}(f) \cap Q_\epsilon) = \beta_{b-\epsilon}^{c+\epsilon} = \dim F_{b-\epsilon}^{c+\epsilon}.$$

Y como se tiene $\dim F_{b-\epsilon}^{c+\epsilon} \leq \dim G_b^c$, entonces $\#(\text{Dgm}(f) \cap Q_\epsilon) \leq \#(\text{Dgm}(g) \cap Q)$.

En el caso que los puntos b y c sean valores críticos homológicos de g y $b - \epsilon$, $c + \epsilon$ valores críticos homológicos de f , entonces podemos engordar los cuadrantes una cantidad $0 < \delta < \epsilon$, tal que

$$\#(\text{Dgm}(f) \cap Q_\epsilon) = \#(\text{Dgm}(f) \cap Q_{b-\epsilon+\delta}^{c+\epsilon-\delta}) \text{ y } \#(\text{Dgm}(g) \cap Q) = \#(\text{Dgm}(g) \cap Q_{b+\delta}^{c-\delta}),$$

siendo estas nuevas coordenadas distintas de los valores críticos de f y g respectivamente. \square

Este lema nos garantiza que la multiplicidad total de $\text{Dgm}(g)$ en el cuadrante superior izquierda con vértice en el punto (b, c) esta acotada inferiormente por la multiplicidad total de $\text{Dgm}(f)$ en el cuadrante superior izquierda reducida por ϵ . Esto se puede observar en la figura 2.22.

Regiones como subespacios vectoriales

Sin embargo, el *Lema del cuadrante* no es lo suficientemente fuerte para nuestros propósitos. Vamos a obtener un resultado similar al *Lema del cuadrante*, pero en este caso para cajas encajadas. Esto se debe a que si se cumple $H(\text{Dgm}(f), \text{Dgm}(g)) \leq \|f - g\|_\infty = \epsilon$, entonces para todo punto $(x, y) \in \text{Dgm}(f)$ debe haber un punto en $\text{Dgm}(g)$ a distancia menor o igual que ϵ . Lo que significa que debe haber un punto $q \in \text{Dgm}(g)$ dentro del cuadrado $[x - \epsilon, x + \epsilon] \times [y - \epsilon, y + \epsilon]$ [12].

Para definir estas regiones introduciremos subespacios vectoriales de \mathbb{R}^2 y haremos uso del *Lema del k -triángulo* para poder expresar sus dimensiones como la multiplicidad total del diagrama de persistencia en dichas regiones.

Desarrollo

Sean $w < x < y < z \in \mathbb{R}$ puntos diferentes a los valores críticos homológicos de $f : \mathbb{X} \rightarrow \mathbb{R}$. Recordamos que la dimensión del grupo de homología F_x es igual a la multiplicidad total en el cuadrante superior izquierdo de vértice el punto de la diagonal (x, x) , y la dimensión del grupo de persistencia F_x^y es igual a la multiplicidad total en el cuadrante superior izquierdo de vértice el punto (x, y) . Estas regiones se pueden observar en las figuras 2.23 (a),(b).

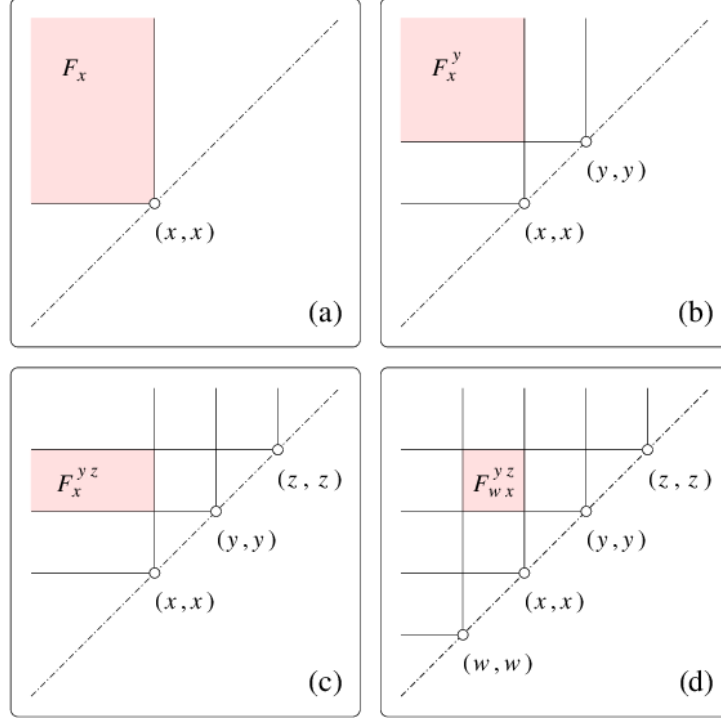


Figura 2.23: (a) Grupo de homología del conjunto de subnivel $f^{-1}(-\infty, x]$. (b) Imagen de F_x en F_y . (c) Núcleo de la sobreyección $F_x^y \rightarrow F_x^z$. (d) Cociente de F_x^{yz} y F_w^{yz} . Fuente: [4]

Si restringimos $f_x^y : F_y \rightarrow F_z$ al espacio vectorial F_x^y tenemos la epimorfismo $f_x^{yz} : F_x^y \rightarrow F_x^z$, ya que todas las clases de homología que están vivas en x y siguen vivas en z , deben de seguir vivas en $y < z$. Denotando por F_x^{yz} al núcleo de dicha aplicación, tenemos que $\dim F_x^{yz} = \dim F_x^y - \dim F_x^z$. Lo que corresponde con la sección marcada en la figura 2.23 (c), que contiene las clases que nacen antes de x y mueren entre y y z .

Además, podemos observar que se cumple $F_w^y \subseteq F_x^y$, ya que todo elemento de F_w^y , que es la imagen de un $\xi \in F_w$ por la aplicación f_w^y , es también la imagen de $f_w^x(\xi)$ por la aplicación f_x^y . Como consecuencia, $F_w^{yz} \subseteq F_x^{yz}$, y por tanto podemos definir el siguiente cociente

$$F_{wx}^{yz} = \frac{F_x^{yz}}{F_w^{yz}}.$$

Al ser un cociente de subespacios vectoriales, su dimensión es la diferencia de los dos núcleos, es decir, $\dim F_{wx}^{yz} = \dim F_x^{yz} - \dim F_w^{yz}$, que equivale a la multiplicidad total en el diagrama de persistencia en la caja $[w, x] \times [y, z]$; como se puede observar en la figura 2.23 (d). Por tanto, este rectángulo contiene las clases de equivalencia que nacen entre w y x y mueren entre y y z .

Relaciones entre cajas encajadas

Lema 2.2.4 (Lema de la caja). Sean $a < b < c < d \in \mathbb{R}$, $R = [a, b] \times [c, d]$ una caja en $\overline{\mathbb{R}}^2$ y $R_\epsilon = [a + \epsilon, b - \epsilon] \times [c + \epsilon, d - \epsilon]$ la caja obtenida de reducir R en todos sus lados. Entonces se cumple

$$\#(\text{Dgm}(f) \cap R_\epsilon) \leq \#(\text{Dgm}(g) \cap R).$$

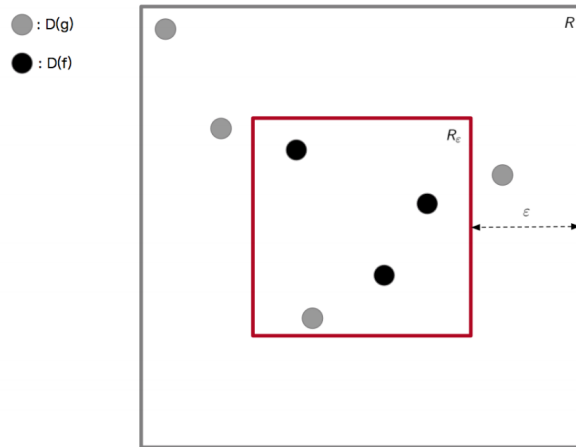


Figura 2.24: Representación del Lema de la caja. Fuente: [12]

Para poder demostrar el Lema de la caja primero recordemos el Segundo teorema de isomorfía:

Teorema 2.2.5 (Segundo teorema de isomorfía [13, Theorem 14.3]). Sea V un espacio vectorial y sean S y T dos subespacios de V , entonces

$$S/(S \cap T) \cong (S + T)/T.$$

Demostración del Lema 2.2.4 (Lema de la caja). Podemos asumir sin perder generalidad que a , b , c y d no son valores críticos homológicos de g y $a + \epsilon$, $b - \epsilon$, $c + \epsilon$ y $d - \epsilon$ no son valores críticos homológicos de f . Además consideraremos que $a + \epsilon < b - \epsilon$ y $c + \epsilon < d - \epsilon$, de forma que R_ϵ este bien definido.

Para el cálculo de las multiplicidades totales dentro de las cajas haremos uso de las dimensiones de los subespacios vectoriales asociados, es decir,

$$\dim F_{a+\epsilon, b-\epsilon}^{c+\epsilon, d-\epsilon} = \#(\text{Dgm}(f) \cap R_\epsilon), \quad (2.3)$$

$$\dim G_{a, b}^c, d = \#(\text{Dgm}(g) \cap R). \quad (2.4)$$

Para demostrar que $\dim F_{a+\epsilon, b-\epsilon}^{c+\epsilon, d-\epsilon} \leq \dim G_{a, b}^c, d$, buscaremos un epimorfismo entre un subespacio vectorial de $G_{a, b}^c, d$ y $F_{a+\epsilon, b-\epsilon}^{c+\epsilon, d-\epsilon}$. Para la construcción de dicha aplicación haremos uso del diagrama que se muestra en la figura 2.25, que, como veremos, esta bien definido y es conmutativo.

Definimos E_a^c como la preimagen, por la restricción de ψ_c a G_b^c , del núcleo de u_3 (ver figura 2.25), es decir, $E_b^c = \psi_c^{-1}(F_{b-\epsilon}^{c+\epsilon, d-\epsilon}) \cap G_b^c$. Por (2.1) se cumple que $F_{b-\epsilon}^{c+\epsilon} \subseteq \psi_c(G_b^c)$, por lo que $s_3 = \psi_c|_{E_b^c}$ tiene al núcleo de u_3 , $F_{b-\epsilon}^{c+\epsilon, d-\epsilon}$, como su imagen.

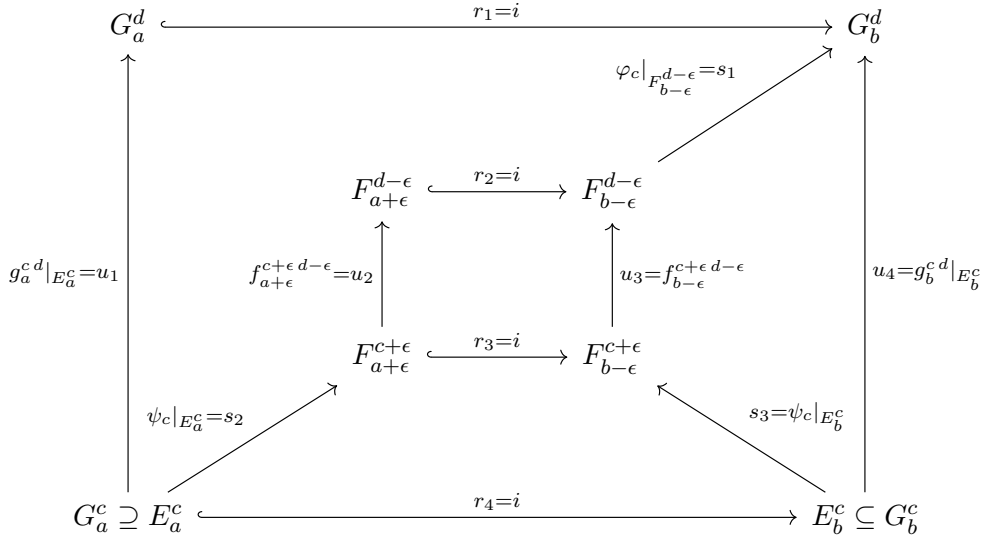


Figura 2.25: Diagrama conmutativo con la notación reducida explicada.

También definimos $E_a^c = G_a^c \cap E_b^c$. Veremos posteriormente que E_b^c/E_a^c es el subespacio de $G_a^c d$ del que podremos encontrar un epimorfismo a $F_{a+\epsilon b-\epsilon}^{c+\epsilon d-\epsilon}$.

Continuando la descripción del diagrama conmutativo tenemos las aplicaciones r_1 , r_2 , r_3 y r_4 que son las inclusiones entre los respectivos espacios vectoriales. Además, u_1 es la restricción de g_a^{cd} en E_a^c y u_2 es la restricción de g_b^{cd} en E_b^c . También tenemos $s_2 = \psi_c|_{E_a^c}$ y por (2.1) se cumple que $\psi_c(G_a^c) \subseteq F_{a+\epsilon}^{c+\epsilon}$, lo que garantiza que s_2 está bien definido ya que su imagen está contenida en $F_{a+\epsilon}^{c+\epsilon}$. Finalmente, $s_1 = \varphi_c|_{F_{b-\epsilon}^{d-\epsilon}}$ y por (2.1) con F y G intercambiados se cumple que $\varphi_{d-\epsilon}(F_{b-\epsilon}^{d-\epsilon}) \subseteq G_b^d$, lo que garantiza que s_1 está bien definido.

Por tanto, el diagrama está bien definido y es conmutativo (ya que las aplicaciones son inclusiones o bien aplicaciones inducidas por inclusiones).

Como se puede observar en la figura 2.26a, $u_4 = s_1 \circ u_3 \circ s_3$, lo que implica que $E_b^c = \ker u_4$, ya que $u_3 \circ s_3$ es cero. Además, como se puede observar en la figura 2.26b, $r_1 \circ u_1 = u_4 \circ r_4$, lo que implica que $E_a^c = \ker u_1$, ya que $u_4 \circ r_4$ es cero y r_1 es inyectivo al ser una inclusión. Expresamos estas relaciones denotando $E_b^c = E_b^{cd} \subseteq G_b^{cd}$ y $E_a^c = E_a^{cd} \subseteq G_a^{cd}$.

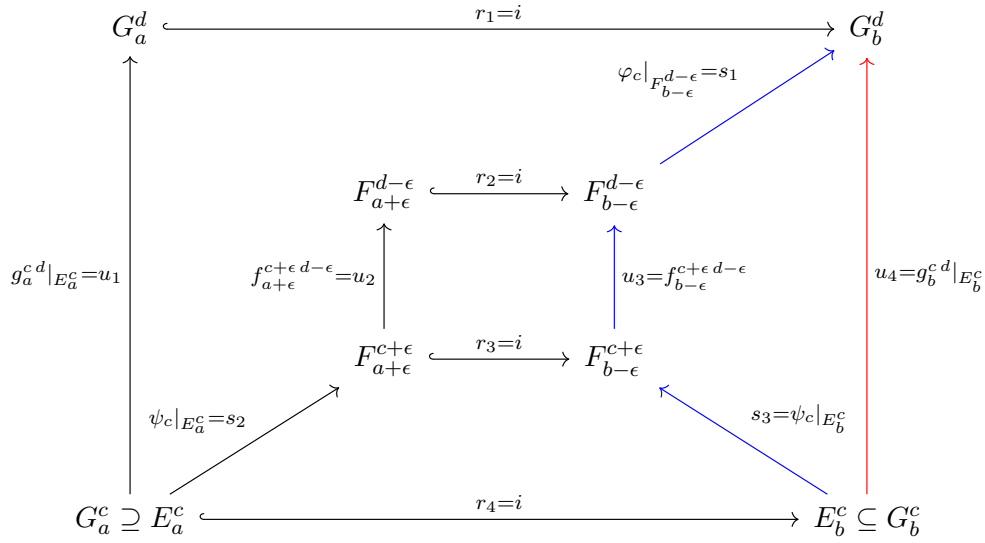
Como $E_a^{cd} = E_b^{cd} \cap G_a^{cd}$, el cociente

$$E_{ab}^{cd} = \frac{E_b^{cd}}{E_a^{cd}} = \frac{E_b^{cd}}{E_b^{cd} \cap G_a^{cd}} \stackrel{Th. 2.2.5}{\cong} \frac{E_b^{cd} + G_a^{cd}}{G_a^{cd}},$$

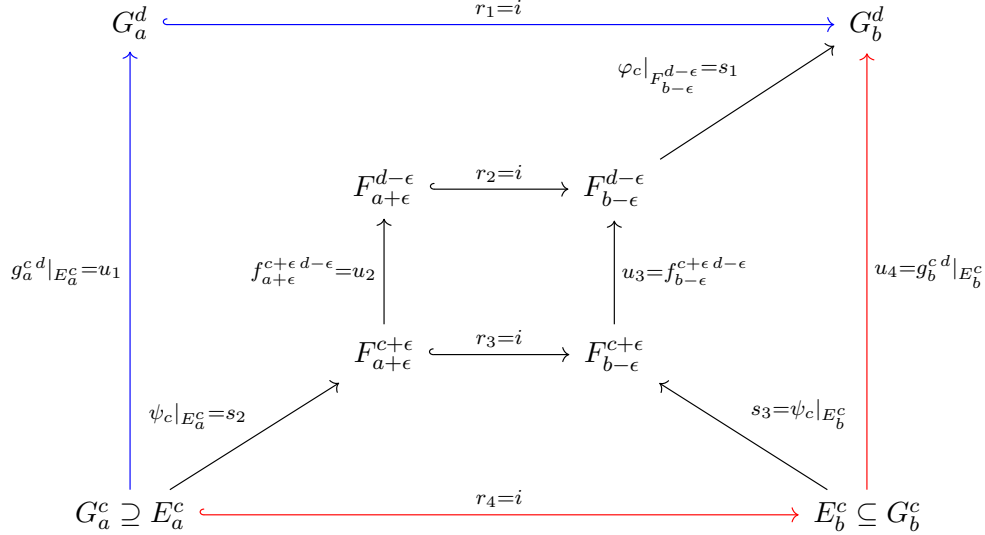
es decir, es un conjunto de clases laterales de elementos en $E_b^{cd} \subseteq G_b^{cd}$ módulo G_a^{cd} , por tanto $E_{ab}^{cd} \subseteq G_{ab}^{cd}$. Luego,

$$\dim E_{ab}^{cd} \leq \dim G_{ab}^{cd}. \quad (2.5)$$

Recordemos que $E_{ab}^{cd} = \ker u_4 / \ker u_1$ y que $F_{a+\epsilon b-\epsilon}^{c+\epsilon d-\epsilon} = \ker u_3 / \ker u_2$. Además, hemos observado que $s_3(\ker u_4) = s_3(E_b^c) = \ker u_3$. Así pues, para demostrar que s_3 induce un epimorfismo entre los cocientes E_{ab}^{cd} y $F_{a+\epsilon b-\epsilon}^{c+\epsilon d-\epsilon}$ sólo quedaría por garantizar que $s_3(\ker u_1) = s_2(\ker u_1)$ está incluida en $\ker u_2$. Sin embargo, esto se cumple, ya que



(a) El camino en azul representa $s_1 \circ u_3 \circ s_3$ y el camino en rojo u_4 .



(b) El camino en azul representa $r_1 \circ u_1$ y el camino en rojo $u_4 \circ r_4$.

Figura 2.26: Representación de las composiciones como caminos en el diagrama conmutativo.

como se puede observar en la figura 2.27, $u_3 \circ s_3 \circ r_4(\xi) = r_2 \circ u_2 \circ s_2(\xi) = 0$ para todo $\xi \in \ker u_1$, y r_2 es inyectiva al ser una inclusión.

Como consecuencia, aplicando la fórmula de las dimensiones tenemos que $\dim E_{a+b}^{c,d} = \dim F_{a+\epsilon}^{c+\epsilon,d-\epsilon} + \dim \ker s_3$, entonces

$$\dim F_{a+\epsilon}^{c+\epsilon,d-\epsilon} \leq \dim E_{a+b}^{c,d}. \quad (2.6)$$

Finalmente, obtenemos la desigualdad al concatenar (2.3), (2.6), (2.5) y (2.4), en este orden, es decir,

$$\#(\text{Dgm}(f) \cap R_\epsilon) = \dim F_{a+\epsilon}^{c+\epsilon,d-\epsilon} \leq \dim E_{a+b}^{c,d} \leq \dim G_{a+b}^{c,d} = \#(\text{Dgm}(g) \cap R).$$

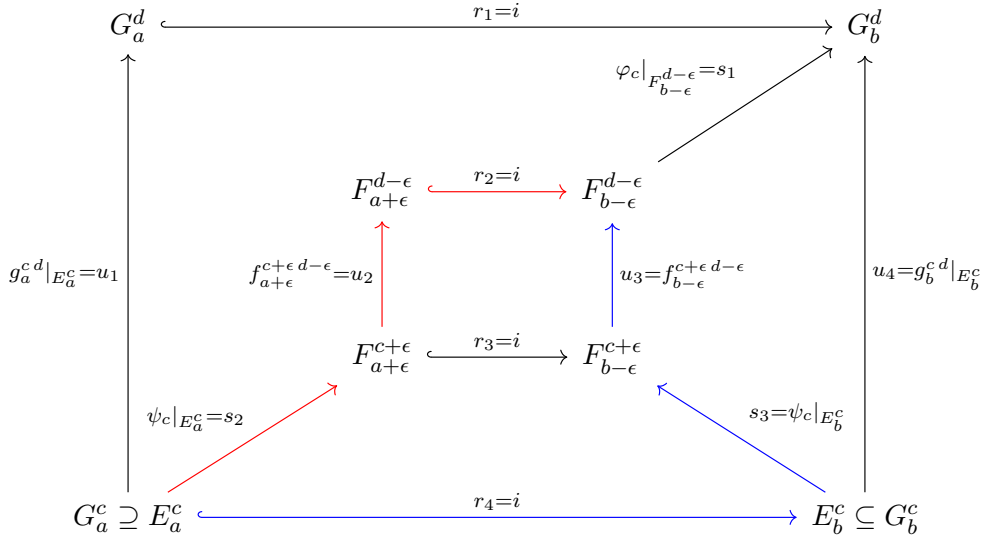


Figura 2.27: El camino en azul representa $u_3 \circ s_3 \circ r_4$ y el camino en rojo $r_2 \circ u_2 \circ s_2$.

□

Como comentábamos previamente, una consecuencia inmediata del *Lema de la caja* es que la distancia Hausdorff entre $\text{Dgm}(f)$ y $\text{Dgm}(g)$ no es mayor que ϵ . Ya que si $R_\epsilon = [x, x] \times [y, y] = (x, y)$ es un punto de $\text{Dgm}(f)$, entonces debe haber un punto de $\text{Dgm}(g)$ a distancia menor o igual que ϵ , porque la multiplicidad total de $\text{Dgm}(g)$ en la caja $R = [x - \epsilon, x + \epsilon] \times [y - \epsilon, y + \epsilon]$ es mayor o igual que uno.

2.2.3. Estabilidad para la distancia bottleneck

Una vez demostrada que la distancia Hausdorff entre dos diagramas de persistencia esta acotada por las distancia infinito de las funciones tame, vamos a probar el resultado para la distancia bottleneck.

Estabilidad para la distancia bottleneck en un caso sencillo

Empezaremos demostrando la estabilidad para un caso concreto que tiene una demostración sencilla. Dada una función tame $f : \mathbb{X} \rightarrow \mathbb{R}$, consideramos la mínima distancia entre dos puntos fuera de la diagonal o bien entre un punto fuera de la diagonal y otro en la diagonal:

$$\delta_f = \min\{\|p - q\|_\infty \mid (\text{Dgm}(f) \setminus \Delta) \ni p \neq q \in \text{Dgm}(f)\}.$$

Si dibujamos cuadrados de radio $\epsilon = \delta_f/2$ centrados en los puntos de $\text{Dgm}(f)$, obtenemos una colección de cuadrados disjuntos entre ellos y disjuntos de la diagonal engordada; ver figura 2.28a. Añadiremos otra función tame $g : \mathbb{X} \rightarrow \mathbb{R}$ que es *muy cercana* a f ; ver figura 2.28b. Lo que significa que f y g satisfacen que $\|f - g\|_\infty < \delta_f/2$.

Así pues, probaremos el teorema de estabilidad para la distancia bottleneck añadiendo la condición que las funciones tame tienen que *estar muy cerca*.

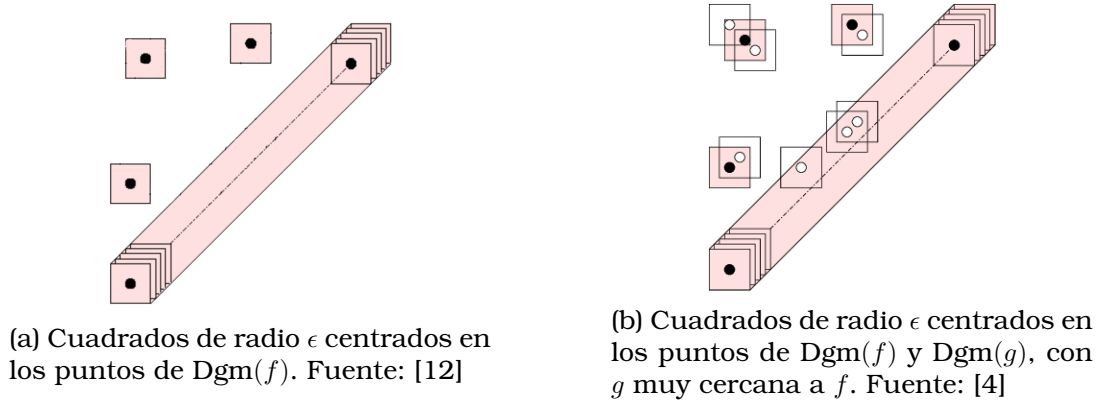


Figura 2.28

Lema 2.2.6 (Lema de la biyección sencilla). Sean $f, g : \mathbb{X} \rightarrow \mathbb{R}$ dos funciones tame, y supongamos que g es muy cercana a f . Entonces los diagramas de persistencia satisfacen

$$W_{\infty}(\text{Dgm}(f), \text{Dgm}(g)) \leq \|f - g\|_{\infty}.$$

Demostración. Denotamos μ a la multiplicidad del punto $p \in (\text{Dgm}(f) \setminus \Delta)$ y \square_{ϵ} al cuadrado de centro p y radio $\epsilon = \|f - g\|_{\infty}$. Entonces, aplicando el Lema de la caja obtenemos que

$$\mu = \#(\text{Dgm}(f) \cap \square_0) \leq \#(\text{Dgm}(g) \cap \square_{\epsilon}) \leq \#(\text{Dgm}(f) \cap \square_{2\epsilon}).$$

Como g es muy cercana a f entonces $\epsilon = \|f - g\|_{\infty} < \delta_f/2$ por lo que $2\epsilon < \delta_f$. Como consecuencia p es el único punto de $\text{Dgm}(f) \cap \square_{2\epsilon}$ como se puede ver en la figura 2.29. Por tanto

$$\#(\text{Dgm}(f) \cap \square_{2\epsilon}) = \mu \Rightarrow \mu \leq \#(\text{Dgm}(g) \cap \square_{\epsilon}) \leq \mu \Rightarrow \#(\text{Dgm}(g) \cap \square_{\epsilon}) = \mu.$$

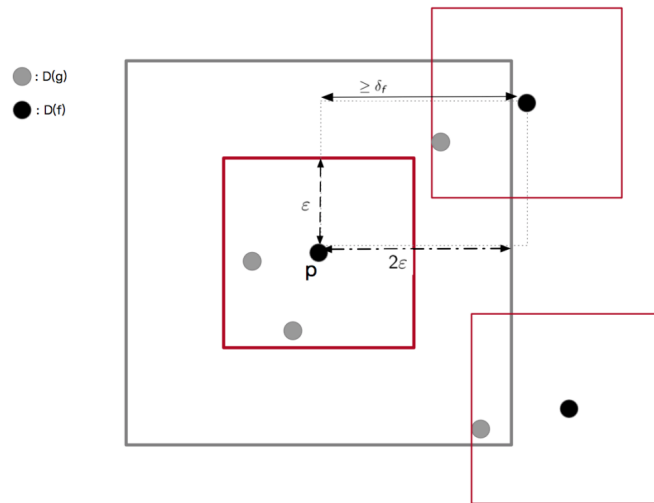


Figura 2.29: Se observa que al ser g muy cercana a f , entonces p es el único punto de $\text{Dgm}(f) \cap \square_{2\epsilon}$. Fuente: [12]

Entonces, podemos emparejar todos los puntos de $\text{Dgm}(g) \cap \square_\epsilon$ con p . Repitiendo este proceso para todos los puntos de $\text{Dgm}(f)$ fuera de la diagonal, emparejaremos todos los puntos de $\text{Dgm}(g)$ excepto aquellos que su distancia a $\text{Dgm}(f) \setminus \Delta$ sea mayor que ϵ . Sin embargo, debido a que $H(\text{Dgm}(f), \text{Dgm}(g)) \leq \epsilon$, estos puntos de $\text{Dgm}(g)$ deben estar a distancia menor o igual que ϵ de la diagonal. Por lo que emparejando estos puntos restantes a su proyección ortogonal sobre la diagonal obtenemos una biyección entre $\text{Dgm}(f)$ y $\text{Dgm}(g)$, tal y como se muestra en la figura 2.30.

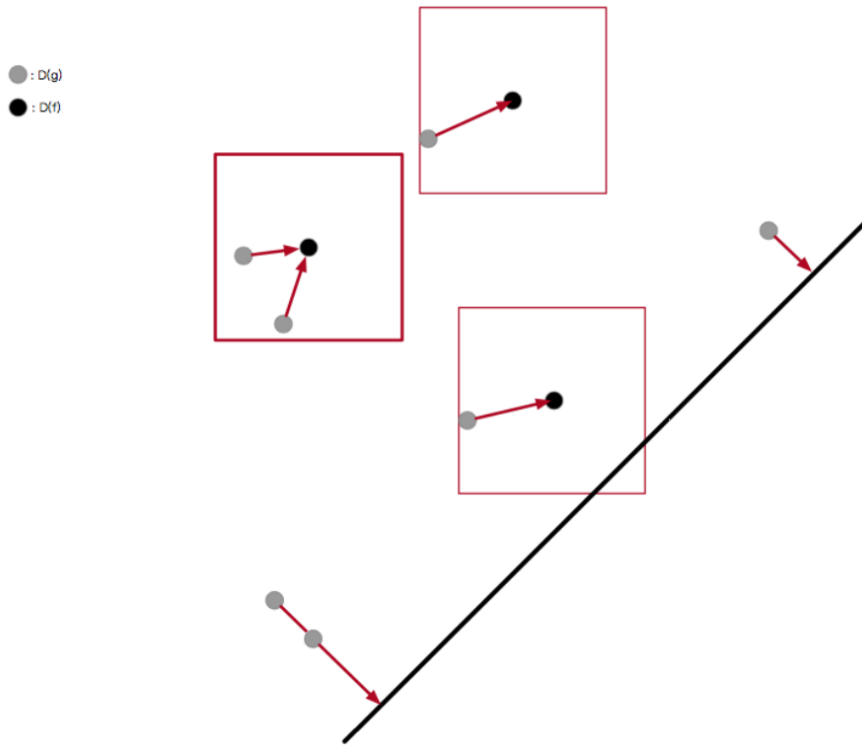


Figura 2.30: Emparejamiento de distancia menor que ϵ entre los puntos de $\text{Dgm}(f)$ y $\text{Dgm}(g)$, para el caso en el que g es muy cercano a f . Fuente: [12]

Como la biyección empareja puntos que están a distancia menor o igual que ϵ , concluimos que la distancia bottleneck entre $\text{Dgm}(f)$ y $\text{Dgm}(g)$ es menor o igual que ϵ . \square

Estabilidad para la distancia bottleneck con funciones PL

Nos acercaremos un poco más a la demostración para el caso general, comprobando primero la estabilidad para dos funciones PL \hat{f} y \hat{g} definidas en un complejo simplicial K . Recordamos que vimos en la sección 2.1.4 que las funciones PL eran tame.

Definimos la *combinación convexa* de \hat{f} y \hat{g} como $h_\lambda : (1 - \lambda)\hat{f} + \lambda\hat{g}$, para $\lambda \in [0, 1]$. Esta familia uniparamétrica de combinaciones convexas forma una interpolación lineal entre las funciones $h_0 = \hat{f}$ y $h_1 = \hat{g}$.

Lema 2.2.7 (Lema de la interpolación). *Sea K un complejo simplicial y sean $\hat{f}, \hat{g} : K \rightarrow \mathbb{R}$ dos funciones PL. Entonces,*

$$W_\infty(\text{Dgm}(\hat{f}), \text{Dgm}(\hat{g})) \leq \|\hat{f} - \hat{g}\|_\infty.$$

Demostración. Descompondremos esta interpolación lineal en suficientes pequeños pasos de forma que podamos usar el *Lema de la biyección sencilla* en cada uno de ellos. Sea $c = \|\hat{f} - \hat{g}\|_\infty$.

Podemos observar que para todo $\lambda \in [0, 1]$, h_λ es tame y que $\delta(\lambda) = \delta_{h_\lambda} > 0$. Para garantizar que h_λ es tame veremos que h_λ es una función PL. Como \hat{f} y \hat{g} son PL, entonces $\hat{f}(x) = \sum_i b_i(x)\hat{f}(u_i)$ y $\hat{g}(x) = \sum_i b_i(x)\hat{g}(u_i)$, donde u_i son los vértices de K y $b_i(x)$ son las coordenadas baricéntricas de x . Luego,

$$\begin{aligned} h_\lambda(x) &= (1 - \lambda)\hat{f}(x) + \lambda\hat{g}(x) \\ &= (1 - \lambda)\sum_i b_i(x)\hat{f}(u_i) + \lambda\sum_i b_i(x)\hat{g}(u_i) \\ &= \sum_i b_i(x)\hat{f}(u_i) - \lambda\sum_i b_i(x)\hat{f}(u_i) + \lambda\sum_i b_i(x)\hat{g}(u_i) \\ &= \sum_i b_i(x)((1 - \lambda)\hat{f}(u_i) + \lambda\hat{g}(u_i)), \end{aligned}$$

por lo que h_λ es PL.

Se sigue que el conjunto de los intervalos abiertos $J_\lambda = (\lambda - \delta(\lambda)/4c, \lambda + \delta(\lambda)/4c) \subset \mathbb{R}$ forman un recubrimiento abierto del intervalo $[0, 1]$. Como $[0, 1]$ es compacto, entonces un subrecubrimiento minimal C' de C será finito. Sean $\lambda_1 < \lambda_2 < \dots < \lambda_n$ los puntos medios de los intervalos de C' . Como C' es minimal y los intervalos son abiertos, entonces cualquier par de intervalos consecutivos J_{λ_i} y $J_{\lambda_{i+1}}$ tienen intersección no vacía. Luego,

$$\lambda_{i+1} - \lambda_i \leq \frac{\delta(\lambda_{i+1}) + \delta(\lambda_i)}{4c} \leq \frac{\max\{\delta(\lambda_{i+1}), \delta(\lambda_i)\}}{2c}.$$

Por definición de c , se cumple que $\|h_{\lambda_i} - h_{\lambda_{i+1}}\|_\infty = c(\lambda_{i+1} - \lambda_i)$, ya que

$$\begin{aligned} \|h_{\lambda_i} - h_{\lambda_{i+1}}\|_\infty &= \|(1 - \lambda_i)\hat{f} + \lambda_i\hat{g} - (1 - \lambda_{i+1})\hat{f} - \lambda_{i+1}\hat{g}\|_\infty = \\ &= \|(\lambda_{i+1} - \lambda_i)\hat{f} - (\lambda_{i+1} - \lambda_i)\hat{g}\|_\infty = |\lambda_{i+1} - \lambda_i|\|\hat{f} - \hat{g}\|_\infty = c(\lambda_{i+1} - \lambda_i). \end{aligned}$$

Como consecuencia, $\|h_{\lambda_i} - h_{\lambda_{i+1}}\|_\infty \leq \max\{\delta(\lambda_{i+1}), \delta(\lambda_i)\}/2$, lo que implica que h_{λ_i} *esta muy cercana a* $h_{\lambda_{i+1}}$ o al revés. Entonces, aplicando el *Lema de la biyección sencilla*, se sigue que $W_\infty(\text{Dgm}(h_{\lambda_i}), \text{Dgm}(h_{\lambda_{i+1}})) \leq \|h_{\lambda_i} - h_{\lambda_{i+1}}\|_\infty$, para todo $1 \leq i \leq n - 1$. Siendo $\lambda_0 = 0$ y $\lambda_{n+1} = 1$, se sigue dando la desigualdad anterior para $i = 0$ e $i = n$, ya que h_0 *es muy cercana a* h_1 y h_1 *es muy cercana a* h_n .

Haciendo uso de la *desigualdad triangular de* W_∞ obtenemos el resultado,

$$\begin{aligned} W_\infty(\text{Dgm}(\hat{f}), \text{Dgm}(\hat{g})) &\stackrel{\text{Des. Triang.}}{\leq} \sum_{i=0}^n W_\infty(\text{Dgm}(h_{\lambda_i}), \text{Dgm}(h_{\lambda_{i+1}})) \leq \sum_{i=0}^n \|h_{\lambda_i} - h_{\lambda_{i+1}}\|_\infty = \\ &= c \sum_{i=0}^n (\lambda_{i+1} - \lambda_i) = c(\lambda_{n+1} - \lambda_0) = c(1 - 0) = \|\hat{f} - \hat{g}\|_\infty. \end{aligned}$$

□

Estabilidad para la distancia bottleneck con funciones tame

Tenemos todos los resultados necesarios para poder demostrar el *Teorema de estabilidad para la distancia bottleneck con funciones tame*, el cual recordamos a continuación:

Teorema 2.2.1 (Teorema de estabilidad para funciones tame). *Sean \mathbb{X} un espacio topológico triangulable y $f, g : \mathbb{X} \rightarrow \mathbb{R}$ dos funciones tame continuas. Entonces,*

$$W_\infty(\text{Dgm}(f), \text{Dgm}(g)) \leq \|f - g\|_\infty.$$

Adicionalmente, necesitaremos recordar un par de nociones de análisis matemático:

Definición 2.2.4. Dados dos espacios métricos (X, d_X) y (Y, d_Y) , y $M \subset X$ entonces una función $f : M \rightarrow Y$ se llama *uniformemente continua* en M cuando, para cada $\epsilon > 0$, puede encontrarse $\delta > 0$ tal que, si $x_1, x_2 \in M$ verifican que $d_X(x_1, x_2) < \delta$, entonces $d_Y(f(x_1), f(x_2)) < \epsilon$.

Teorema 2.2.8 (Teorema de Heine–Cantor [14]). *Sean E y F espacios métricos y $f : E \rightarrow F$ una función continua. Si E es compacto, entonces f es uniformemente continua.*

Demostración del teorema 2.2.1 (Teorema de estabilidad para funciones tame).

Por la definición de espacio triangulable, existe un complejo simplicial (finito) L y un homeomorfismo $\Phi : |L| \rightarrow \mathbb{X}$. Se observa que el diagrama de persistencia es invariante por este tipo de cambio de variable, es decir, $f \circ \Phi : |L| \rightarrow \mathbb{R}$ es tame y tiene el mismo diagrama de persistencia que f .

Sea $\delta > 0$ lo suficientemente pequeño. Como f y g son continuas, L es compacto, entonces por el *Teorema de Heine-Cantor* f y g son uniformemente continuas, lo que nos garantiza que existe una subdivisión K de L tal que

$$\begin{aligned} |f \circ \Phi(u) - f \circ \Phi(v)| &\leq \delta, \\ |g \circ \Phi(u) - g \circ \Phi(v)| &\leq \delta, \end{aligned}$$

cuando u y v son puntos de un mismo símplice de K , ya que podemos obtener esta subdivisión K de L con la propiedad de que el diámetro de cada símplice sea tan pequeño como sea necesario para cumplir la condición de la continuidad uniforme de f y g .

Sean $\hat{f}, \hat{g} : \text{Sd } K \rightarrow \mathbb{R}$ funciones PL que aproximan a $f \circ \Phi$ y $g \circ \Phi$ en K . Por construcción de K , estas funciones satisfacen $\|\hat{f} - f \circ \Phi\|_\infty \leq \delta$ y $\|\hat{g} - g \circ \Phi\|_\infty \leq \delta$.

Para terminar utilizaremos la *desigualdad triangular de W_∞* para acotar $W_\infty(\text{Dgm}(f), \text{Dgm}(g))$ superiormente por la suma de las distancias entre los diagramas de persistencia de las funciones adyacentes en la secuencia f, \hat{f}, \hat{g}, g .

Para el par \hat{f} y \hat{g} tenemos

$$W_\infty(\text{Dgm}(\hat{f}), \text{Dgm}(\hat{g})) \leq \|\hat{f} - \hat{g}\|_\infty \tag{2.7}$$

$$\leq \|\hat{f} - f \circ \Phi\|_\infty + \|f \circ \Phi - g \circ \Phi\|_\infty + \|\hat{g} - g \circ \Phi\|_\infty \tag{2.8}$$

$$\leq \|f - g\|_\infty + 2\delta \tag{2.9}$$

El punto 2.7 se debe al *Lema de la interpolación*, el 2.8 a la *desigualdad triangular de* d_∞ y por último, el punto 2.9 se debe a que \hat{f} y \hat{g} difieren como mucho δ de $f \circ \Phi$ y $g \circ \Phi$ respectivamente, y que $\|f - g\|_\infty = \|f \circ \Phi - g \circ \Phi\|_\infty$.

Para poder acotar la distancia bottleneck entre f y \hat{f} , supondremos que $\delta < \delta_f/2$, de forma que podemos aplicar el *Lema de la biyección sencilla*. Como el cambio de variables no afecta al diagrama de persistencia obtenemos que

$$W_\infty(\text{Dgm}(f), \text{Dgm}(\hat{f})) = W_\infty(\text{Dgm}(f \circ \Phi), \text{Dgm}(\hat{f})) \leq \delta.$$

Análogamente podemos acotar la distancia bottleneck entre g y \hat{g} si asumimos que $\delta < \min\{\delta_f/2, \delta_g/2\}$.

Luego, en total tenemos

$$\begin{aligned} W_\infty(\text{Dgm}(f), \text{Dgm}(g)) &\stackrel{\text{Des. Triang.}}{\leq} W_\infty(\text{Dgm}(f), \text{Dgm}(\hat{f})) + W_\infty(\text{Dgm}(\hat{f}), \text{Dgm}(\hat{g})) \\ &\quad + W_\infty(\text{Dgm}(\hat{g}), \text{Dgm}(g)) \\ &\leq \|f - g\|_\infty + 4\delta \xrightarrow{\delta \rightarrow 0} \|f - g\|_\infty \end{aligned}$$

Como la condición es cierta para todo $\delta > 0$, podemos hacer δ tan pequeño como queramos. Por tanto queda demostrado el *Teorema de estabilidad*. \square

2.3. Implementaciones y cálculos

En esta sección daremos algunas evidencias computacionales de la estabilidad de los diagramas de persistencias, centrándonos filtraciones de complejos simpliciales asociadas a nubes de puntos con un cierto ruido. Para ello comenzaremos estudiando posibles algoritmos para calcular tanto la *distancia Hausdorff* como la *distancia bottleneck*.

2.3.1. Cálculo de la distancia Hausdorff

Definición 2.3.1. Sea A y B dos conjuntos de puntos. Se define la *distancia Hausdorff directa* entre A y B como el máximo de las distancias entre cada punto $x \in A$ y el punto $y \in B$ más cercano a x . Es decir,

$$\check{H}(A, B) = \sup_{x \in A} \inf_{y \in B} \|x - y\|_\infty.$$

Observación. $\check{H}(A, B) \neq \check{H}(B, A)$ y por tanto la distancia Hausdorff directa no es simétrica.

Luego, la distancia de Hausdorff es el máximo de las distancias Hausdorff directas en ambas direcciones, es decir

$$H(A, B) = \max\{\check{H}(A, B), \check{H}(B, A)\}.$$

Sea $A = \{x_1, x_2, \dots, x_m\}$ y $B = \{y_1, y_2, \dots, y_m\}$ los dos conjuntos de puntos en \mathbb{R}^k y sea $\|x - y\|_\infty$ la distancia infinito entre x y y . Por lo tanto, podemos calcular de manera sencilla la distancia Hausdorff directa entre A y B de la siguiendo los pasos del algoritmo 1.

Desarrollo

Algoritmo 1 Cálculo de la distancia Hausdorff directa

Entrada: Dos conjuntos finitos de puntos A y B

Salida: Distancia Hausdorff directa entre A y B

```
1:  $cmax \leftarrow 0$ 
2: for  $x \in A$  do
3:    $cmin \leftarrow \infty$ 
4:   for  $y \in B$  do                                 $\triangleright$  Calculamos  $d_\infty(x, B) = \inf_{y \in B} d_\infty(x, y)$ 
5:      $d \leftarrow \|x - y\|_\infty$ 
6:     if  $d < cmin$  then
7:        $cmin \leftarrow d$ 
8:     end if
9:   end for
10:  if  $cmin > cmax$  then                              $\triangleright$  Recalculamos el supremo
11:     $cmax \leftarrow cmin$ 
12:  end if
13: end for
14: return  $cmax$ 
```

Obviamente, la complejidad del algoritmo 1 es del orden de $\mathcal{O}(n * m)$, donde $m = |A|$ y $n = |B|$. La distancia Hausdorff entre A y B será el máximo de los resultados de ejecutar el algoritmo 1 en ambas direcciones, y por lo tanto la complejidad de calcular la distancia Hausdorff de este modo es de $\mathcal{O}(n * m)$.

Sin embargo, existen implementaciones del cálculo de la distancia Hausdorff que tienen complejidad del orden de $\mathcal{O}(m)$ en el mejor de los casos y $\mathcal{O}(n * m)$ en el peor de los casos [15].

2.3.2. Cálculo de la distancia bottleneck

En esta sección veremos los algoritmos propuestos en [2], donde el cálculo de la distancia bottleneck entre dos diagramas de persistencia se reduce a la obtención de un emparejamiento óptimo en un grafo bipartido.

Obtención de la distancia a partir de emparejamientos

Empezaremos viendo cómo podemos obtener la distancia bottleneck entre diagramas de persistencia a partir de emparejamientos de un grafo bipartido.

Sea X e Y dos diagramas de persistencia, para los que asumimos que están formados por un número finito de puntos fuera de la diagonal e infinitos puntos en ella. Denotamos X_0 al multiconjunto finito de los puntos fuera de la diagonal en X y X'_0 a la proyección ortogonal de X_0 sobre la diagonal. Por tanto, construimos el grafo bipartido completo

$$G = (U \dot{\cup} V, A), \text{ con } U = X_0 \dot{\cup} Y'_0, V = Y_0 \dot{\cup} X'_0, \text{ y } A = U \times V,$$

donde $U \dot{\cup} V$ denota la unión disjunta de los conjuntos U y V .

En este grafo introducimos la función de coste $c : A \rightarrow \mathbb{R}$ donde a cada arista $uv \in A$

se le asigna la la distancia infinito entre los puntos u y v :

$$c(uv) = \begin{cases} \|u - v\|_\infty & \text{si } u \in X_0 \text{ ó } v \in Y_0 \\ 0 & \text{si } u \in Y'_0 \text{ y } v \in X'_0 \end{cases}$$

Observación. Por construcción, la arista de coste mínimo que conecta un punto u fuera de la diagonal con un punto de la diagonal es uu' , donde u' es la proyección ortogonal de u sobre la diagonal. Además, el coste de esta arista es la mitad de la persistencia de u .

Definición 2.3.2. Un *emparejamiento* en G es un subconjunto $M \subseteq A$ tal que dos aristas de M no tienen un vértice en común. Diremos que

- M es *maximal* si no existe un emparejamiento M' en G con $M \subset M'$.
- M es *máximo* si no existe un emparejamiento M' en G con $\text{card } M < \text{card } M'$.
- M es *perfecto* si todos los vértices de G son extremo de alguna arista de M .

Como G es un grafo bipartido completo, todo emparejamiento máximo es también un emparejamiento perfecto.

Definición 2.3.3. Se define $G(\epsilon) = (U \dot{\cup} V, A_\epsilon)$ como el subgrafo de G que se obtiene al eliminar todas las aristas $uv \in A$ con coste $c(uv) > \epsilon$.

En este caso, todo emparejamiento perfecto en $G(\epsilon)$ es máximo, sin embargo, el opuesto no siempre es cierto.

Definición 2.3.4. Un *emparejamiento de coste mínimo* es un emparejamiento máximo que minimiza la suma de los costes de las aristas del emparejamiento. Denotaremos a esta suma como el *coste total* del emparejamiento.

Lema 2.3.1 (Lema de reducción [2]). Sean X e Y dos diagramas de persistencia y sea $G = (U \dot{\cup} V, A)$ su correspondiente grafo bipartido. Entonces la distancia bottleneck entre X e Y es el menor ϵ tal que el subgrafo $G(\epsilon)$ tiene un emparejamiento perfecto.

Por lo tanto, el cálculo de la distancia bottleneck entre diagramas de persistencia se reduce a la obtención de emparejamientos perfectos con coste mínimo en grafos bipartidos.

Emparejamientos en grafos bipartidos

Comenzaremos viendo cómo podemos obtener emparejamientos máximos en el grafo bipartido $G(\epsilon) = (U \dot{\cup} V, A_\epsilon)$. Para ello haremos uso de algoritmos iterativos, donde en cada paso mejoraremos el emparejamiento, hasta que no sea posible aumentarlo.

Definición 2.3.5. Sea M_i el emparejamiento tras realizar i iteraciones. Se define $D_i = (P, Q)$ como el digrafo tal que

- $P = (U \dot{\cup} V) \cup \{s, t\}$, donde s se denota como fuente y t como sumidero.
- $Q = Q_1 \cup Q_2$, donde
 - Q_1 son las aristas $x \in A_\epsilon$ tal que x va de V a U si pertenece al emparejamiento M_i , y x va de U a V en caso contrario.

Desarrollo

- Q_2 son las aristas que van desde s a los vértices no emparejados $u \in U$, más las aristas que van desde los vértices no emparejados $v \in V$ a t .

En la figura 2.31 podemos observar un ejemplo del digrafo D_i asociado a un emparejamiento M_i .

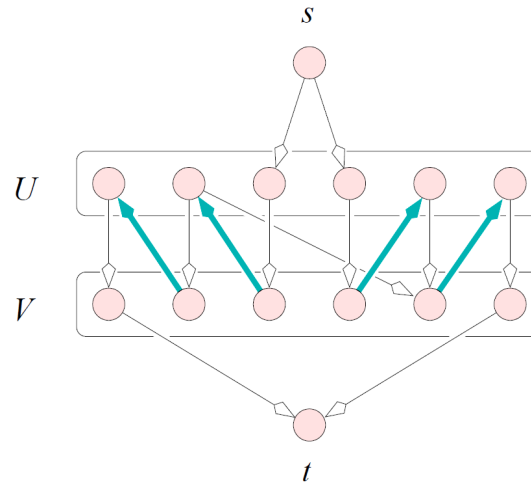


Figura 2.31: Digrafo asociado a un emparejamiento de cuatro aristas. Fuente: [2]

Definición 2.3.6. Un *camino de M_i -aumento* es un camino dirigido desde s hasta t el cual visita un vértice de D_i como máximo una vez.

Claramente, si tenemos un camino de M_i -aumento con k vértices no contenidos en M_i y $k - 1$ vértices en M_i , entonces podemos mejorar el emparejamiento sustituyendo los k vértices que no estaban en M_i por los $k - 1$ vértices que sí estaban en M_i . Cuando hacemos esta mejora, decimos que hemos *aumentado* el emparejamiento usando el camino.

Lema 2.3.2 (Lema de Berge). M_i es un emparejamiento máximo de $G(\epsilon)$ si y sólo si $G(\epsilon)$ no contiene caminos de M_i -aumento.

Luego, para obtener un emparejamiento máximo de $G(\epsilon)$ seguiremos los siguientes pasos:

Algoritmo 2 Obtención de emparejamientos máximos

Entrada: $G(\epsilon) = (U \cup V, A_\epsilon)$ grafo bipartido

Salida: M_i es un emparejamiento máximo de $G(\epsilon)$

- 1: $M_0 \leftarrow \emptyset$
 - 2: $i \leftarrow 0$
 - 3: **while** existe un camino de M_i -aumento en D_i **do**
 - 4: aumentar M_i usando el camino para obtener M_{i+1}
 - 5: $i \leftarrow i + 1$
 - 6: **end while**
 - 7: **return** M_i
-

Este algoritmo terminará como mucho en n iteraciones, siendo $n = \text{card } U = \text{card } V$, ya que en cada iteración se aumenta el tamaño del emparejamiento en uno. Podemos hacer uso de la *búsqueda en anchura* y la *búsqueda en profundidad* para encontrar

caminos de M_i -aumento en un tiempo proporcional al número de aristas en A_ϵ . Por lo que la complejidad del algoritmo es del orden de $\mathcal{O}(n^3)$.

Se puede obtener una complejidad del orden de $\mathcal{O}(n^{5/2})$ implementando el algoritmo que se muestra en [2]. Este hace uso de la *búsqueda en anchura* para etiquetar los vértices con su distancia a s y después usa la *búsqueda en profundidad* para construir un conjunto maximal de múltiples caminos de M_i -aumento.

Emparejamientos de coste mínimo en grafos bipartidos

Para calcular el menor ϵ tal que $G(\epsilon)$ tiene un emparejamiento perfecto, seguiremos una variante del método húngaro, el cual se utiliza para resolver problemas de asignación [16].

Propiedad 2.3.1 ([2]).

- A. Si el subgrafo $G(0)$, que consiste en las aristas de coste cero de G , tiene un emparejamiento perfecto, entonces es un emparejamiento de coste mínimo. Es más, su coste total es cero.
- B. Restar la misma cantidad al coste de todas las aristas incidentes a un vértice de G afecta a todos los emparejamientos perfectos de la misma forma. En particular, un emparejamiento perfecto minimiza el coste total antes de las restas de la cantidades si y sólo si sigue minimizándolo tras las restas de las cantidades.

Así pues, empezaremos construyendo un emparejamiento máximo en $G(0)$. Si es un emparejamiento perfecto ya hemos acabado y por lo tanto la distancia bottleneck entre los diagramas de persistencia es 0. En otro caso, cambiaremos los costes de las aristas de G preservando el orden de los emparejamientos perfectos en G por coste total. Para ello introducimos las *funciones de reducción* $d_i : U \times V \rightarrow \mathbb{R}$. Partiendo de $d_0(x) = 0$ para todos los vértices de G , el algoritmo cambiará el valor de la función de reducción en cada iteración i .

Definición 2.3.7. Sea $c(xy)$ el coste original de la arista $xy \in G$. Se define el *coste modificado* tras i iteraciones como

$$c_i(xy) = c(xy) - d_i(x) - d_i(y) > 0.$$

Sea G_i el grafo G con los costes modificados por d_i , entonces el algoritmo construirá iterativamente emparejamientos máximos en $G_i(0)$, que coincide con el subgrafo resultante al eliminar las aristas con peso no nulo de G_i . Incrementando el número de aristas del emparejamiento máximo en uno por cada iteración, obtendremos el emparejamiento perfecto en n iteraciones.

Análogo al método Húngaro, iremos añadiendo aristas de coste modificado cero al emparejamiento en cada iteración y para generar ceros adicionales en los costes modificados de las aristas seleccionaremos el menor de los costes totales de los caminos de M_i -aumento como cantidad que variará la función de reducción.

Sea M_i un emparejamiento máximo en $G_i(0)$ y sea D_i el digrafo asociado al emparejamiento M_i y G_i . Si M_i no es un emparejamiento perfecto en G_i , entonces no es un emparejamiento máximo en G_i , y por lo tanto existirá un camino de M_i -aumento en D_i .

Por definición $c_i(sy) = c_i(xt) = 0$ para todo $x \in U$ e $y \in V$. Se denota como *coste total* de un camino de M_i -aumento como la suma de los costes modificados de sus aristas. Obtendremos el camino de M_i -aumento π que minimiza el coste total, a través del *algoritmo de Dijkstra* con una complejidad del orden de $\mathcal{O}(n^2)$.

Como hacíamos en el algoritmo 2, aumentamos M_i usando π para obtener M_{i+1} . Vamos a garantizar que podemos cambiar la función de reducción de forma que todas las aristas del emparejamiento M_{i+1} tienen coste modificado cero. Para ello definimos $\gamma_i(x)$ como el coste total mínimo de los caminos desde s hasta x .

De esta forma, actualizamos las funciones de reducción a

$$d_{i+1} = \begin{cases} d_i(x) - \gamma_i(x) & \text{si } x \in U \\ d_i(x) + \gamma_i(x) & \text{si } x \in V \end{cases}$$

Luego, para todos los vértices $u \in U$ y $v \in V$, el nuevo coste modificado de la arista uv es:

$$c_{i+1}(uv) = c(uv) - d_i(u) - d_i(v) + \gamma_i(u) - \gamma_i(v).$$

Propiedad 2.3.2 ([2]). Sea M_{i+1} el emparejamiento máximo obtenido al aumentar M_i . Entonces, $c_{i+1}(uv) \geq 0$ para toda arista uv en G_i , y $c_{i+1}(uv) = 0$ para toda arista $uv \in M_{i+1}$.

La propiedad anterior garantiza que en la última iteración obtenemos el emparejamiento perfecto de coste total mínimo, y por tanto la distancia bottleneck entre los diagramas de persistencia X e Y es igual al máximo de los costes originales de las aristas de dicho emparejamiento perfecto, es decir

$$W_\infty(X, Y) = \max_{xy \in M_n} c(xy), \text{ siendo } n = \text{card } U = \text{card } V.$$

Como tenemos n iteraciones en las cuales cada una aplicamos el algoritmo de Dijkstra, entonces la complejidad del cálculo de la distancia bottleneck siguiendo el algoritmo comentado es del orden de $\mathcal{O}(n^3)$.

2.3.3. Pruebas

La implementación del cálculo de las distancias se ha realizado en Python y el código se puede encontrar en el Anexo 1.

Para comprobar el resultado del problema estudiaremos la robustez de los diagramas de persistencia para un conjunto finito de puntos y una variación de dicho conjunto dada por un cierto ruido. Obtendremos filtraciones de alfa complejos para ambos conjuntos de puntos, y calcularemos la distancia bottleneck entre sus diagramas de persistencia. Veremos que para el caso de los alfa complejos de nubes de puntos obtenemos el siguiente teorema de estabilidad

Teorema 2.3.3 (Teorema de estabilidad para alfa complejos de nubes de puntos). Sean A y B subconjuntos finitos de \mathbb{R}^d y $\text{Dgm}_k(A)$ el diagrama de persistencia de dimensión k de las filtraciones del alfa complejo asociado a A . Entonces, para cada k

$$W_\infty(\text{Dgm}_k(A), \text{Dgm}_k(B)) \leq H(A, B).$$

Para su demostración haremos uso de los siguientes resultados:

Teorema 2.3.4 (Teorema de la invariancia homotópica [7, Theorem 2.10]). Si dos funciones $f, g : X \rightarrow Y$ son homotópicas, entonces inducen el mismo homomorfismo $f_* = g_* : H_n(X) \rightarrow H_n(Y)$.

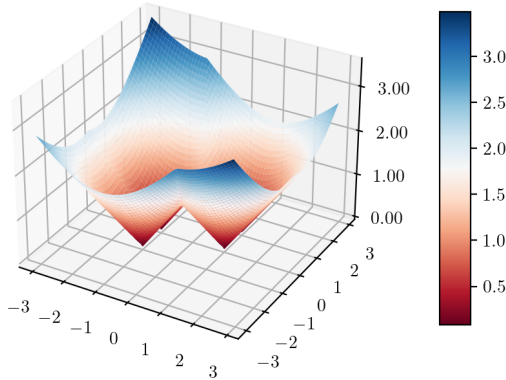
Y su correspondiente corolario,

Corolario ([7, Corollary 2.11]). Las funciones $f_* : H_n(X) \rightarrow H_n(Y)$ inducidas por la equivalencia homotópica $f : X \rightarrow Y$, son isomorfismos para todo n .

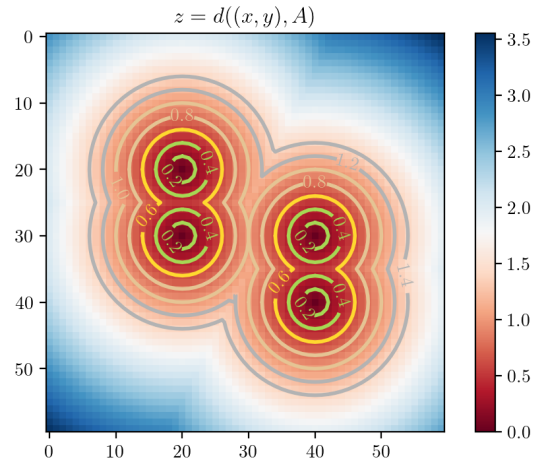
Demostración del Teorema 2.3.3. Sea $d^A : \mathbb{R}^d \rightarrow \mathbb{R}$ la función que asigna a cada punto $p \in \mathbb{R}^d$ su distancia a A . Entonces, por la definición de la distancia Hausdorff,

$$\begin{aligned} H(A, B) &= \max \left\{ \sup_{x \in A} \inf_{y \in B} \|x - y\|_\infty, \sup_{y \in B} \inf_{x \in A} \|y - x\|_\infty \right\} \\ &= \sup_{p \in A \cup B} \left| \inf_{x \in A} d_\infty(p, x) - \inf_{y \in B} d_\infty(p, y) \right| \\ &= \sup_{p \in \mathbb{R}^d} \left| \inf_{x \in A} d_\infty(p, x) - \inf_{y \in B} d_\infty(p, y) \right| \\ &= \sup_{p \in \mathbb{R}^d} |d_\infty(p, A) - d_\infty(p, B)| \\ &= \|d^A - d^B\|_\infty. \end{aligned}$$

Por otro lado si denotamos al conjunto de subnivel $\hat{L}_\epsilon = \{x \mid d^A \leq \epsilon\}$, que es la unión de las bolas cerradas centradas en los puntos de A y radio ϵ , como podemos ver en la figura 2.32. Como consecuencia del *Lema del Nervio* $H_k(\hat{L}_r) \cong H_k(\check{\text{Cech}}(r))$, para todo k debido a que el complejo de Čech tiene el mismo tipo de homotopía que la unión de las bolas cerradas de radio r centradas en los puntos de A .



(a) Representación del grafo de d^A en \mathbb{R}^3 .



(b) Representación de los conjuntos de subnivel \hat{L}_r .

Figura 2.32: Grafo de d^A y sus conjuntos de subnivel, siendo el conjunto de puntos $A = \{(-1, 0), (1, 0), (1, 1), (-1, -1)\}$. El código para la obtención de las figuras se encuentra en el Anexo 2.

Y como el complejo de Čech tiene el mismo tipo de homotopía que el alfa complejo entonces, como consecuencia del corolario 2.3.3,

$$H_k(\text{Alpha}(r)) \cong H_k(\check{\text{Cech}}(r)) \cong H_k(\widehat{L}_r), \text{ para todo } k.$$

□

El código de las siguientes pruebas se puede encontrar en el Anexo 3.

2.3.3.1. Ejemplo 1: Elipse

Empezaremos viendo un ejemplo sencillo de la curva de una elipse dada por la siguiente parametrización:

$$\gamma(t) = (4 \sin(t), 9 \cos(t)), \text{ con } t \in [0, 2\pi]$$

De esta curva obtenemos 30 puntos y obtenemos las filtraciones de alfa complejos asociadas a dichos puntos. Además introducimos una pequeña cantidad de ruido en los puntos a través de una distribución normal $N(2, 0.09)$, y obtenemos sus filtraciones de alfa complejos asociadas. Una vez obtenidas las filtraciones obtendremos los diagramas de persistencia correspondientes como se muestra en la figura 2.33.

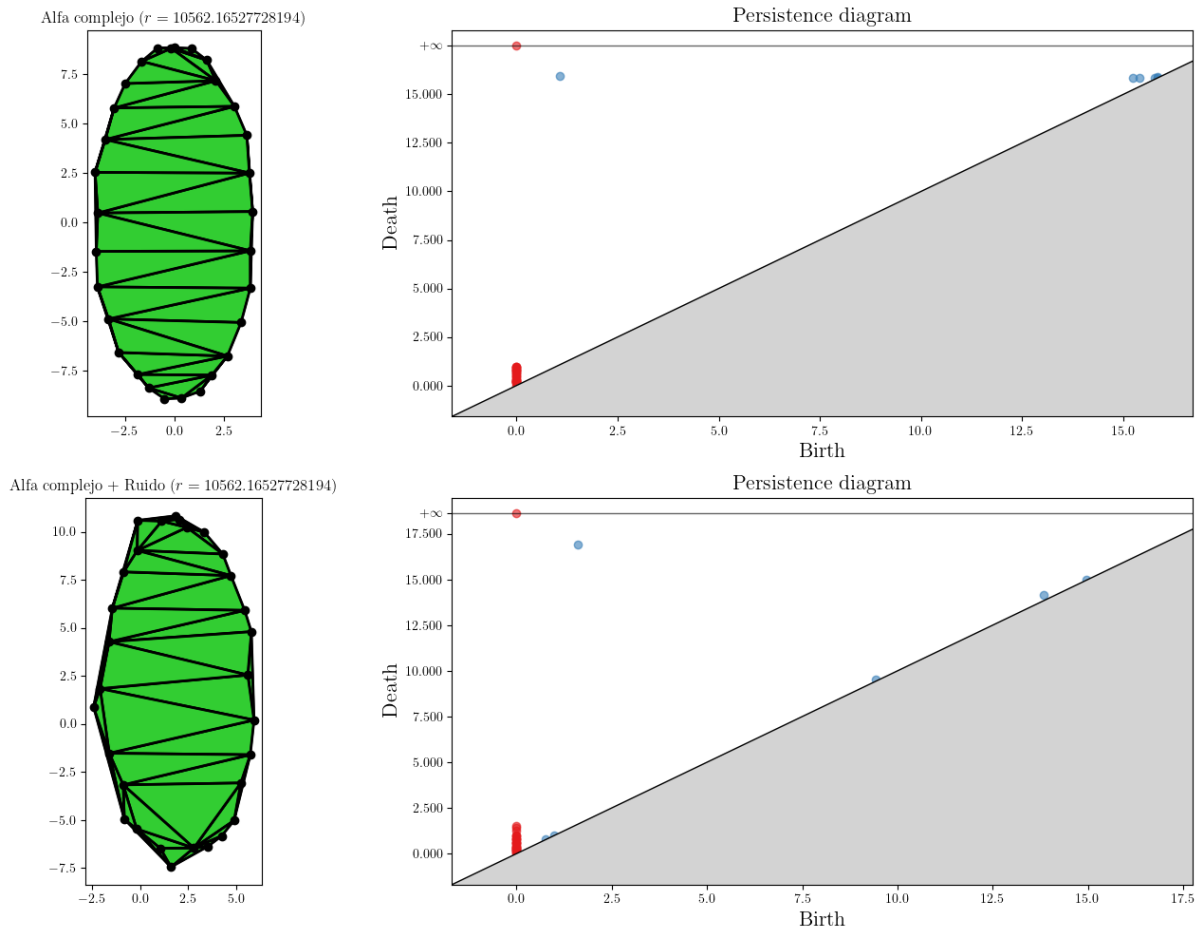


Figura 2.33: Alfa complejos y diagramas de persistencias asociados a las filtraciones de los alfa complejos. En las figuras superiores se toma como entrada una elipse y en las inferiores la misma elipse mas cierto ruido.

Podemos observar a simple vista en la figura 2.33 que los diagramas de persistencia varían poco, y las características más importantes se mantienen. Se ve que en ambos diagramas para la dimensión cero la mayoría de las clases mueren muy pronto y una de ellas nunca muere, lo que refleja que la elipse es conexa. Por otro lado, en las clases de dimensión 1 todas están muy cercanas a la diagonal excepto una que se mantiene durante un largo periodo de tiempo, lo cual refleja el agujero que contiene el interior de la elipse.

Esta similitud entre los diagramas queda demostrada al calcular la distancia bottleneck entre ellos para las dos dimensiones. Adicionalmente, se puede observar que se cumple el *Teorema de estabilidad* debido a que la distancia bottleneck es inferior a la distancia Hausdorff entre los conjuntos de puntos, es decir, la variación entre los diagramas es inferior a la variación entre los puntos de la curva.

```

1 Alfa complejo de dimensión 2 - 133 símlices - 30 vértices.
2 Alfa complejo de dimensión 2 - 145 símlices - 30 vértices.
3
4 Distancia Hausdorff: 2.4398
5 Distancia bottleneck dimensión 0: 0.55573 (Implementación GUDHI: 0.55573)
6 Distancia bottleneck dimensión 1: 0.98572 (Implementación GUDHI: 0.98572)

```

2.3.3.2. Ejemplo 2: Rosa polar

Continuaremos viendo un ejemplo un poco más elaborado. Tomaremos la curva de una rosa polar dada por la siguiente parametrización:

$$\alpha(t) = (10 \cos(2t) \cos(t), 10 \cos(2t) \sin(t)), \text{ con } t \in [0, 2\pi]$$

De esta curva obtenemos 50 puntos y obtenemos las filtraciones de alfa complejos asociadas a dichos puntos. En este caso introducimos una cantidad de ruido en los puntos superior respecto al anterior ejemplo. Estas perturbaciones se generan a través de una distribución normal $N(2, 0.25)$. Nuevamente, obtenemos sus filtraciones de alfa complejos asociadas a estos puntos. Una vez obtenidas las filtraciones obtendremos los diagramas de persistencia correspondientes como se muestra en la figura 2.34.

De nuevo, podemos observar a simple vista en la figura 2.34 que los diagramas de persistencia varían poco, y las características más importantes se mantienen. Se ve que en ambos diagramas para la dimensión cero la mayoría de las clases mueren muy pronto y una de ellas nunca muere, lo que refleja que la rosa polar es conexa. Por otro lado, en las clases de dimensión 1 se observa que únicamente hay 4 clases en cada diagrama que se encuentran lejos de la diagonal, lo que refleja los 4 agujeros de esta rosa polar.

Cabe destacar que para el conjunto de datos inicial, estas cuatro clases mueren y nacen en momentos similares. Esto se debe a que, como podemos ver en la figura, los agujeros están bien definidos. Mientras que en el caso de los datos perturbados estos agujeros no se ven tan claros, lo cual se refleja en el diagrama al ver que 3 de los cuatros puntos comentados se encuentran mucho más cercanos a la diagonal.

Esta diferencias comentadas se pueden ver en el cálculo de la distancia bottleneck, ya que se observa que en la dimensión 1 tenemos una distancia superior que en el ejemplo anterior. Sin embargo se observa que la distancia bottleneck es inferior a la

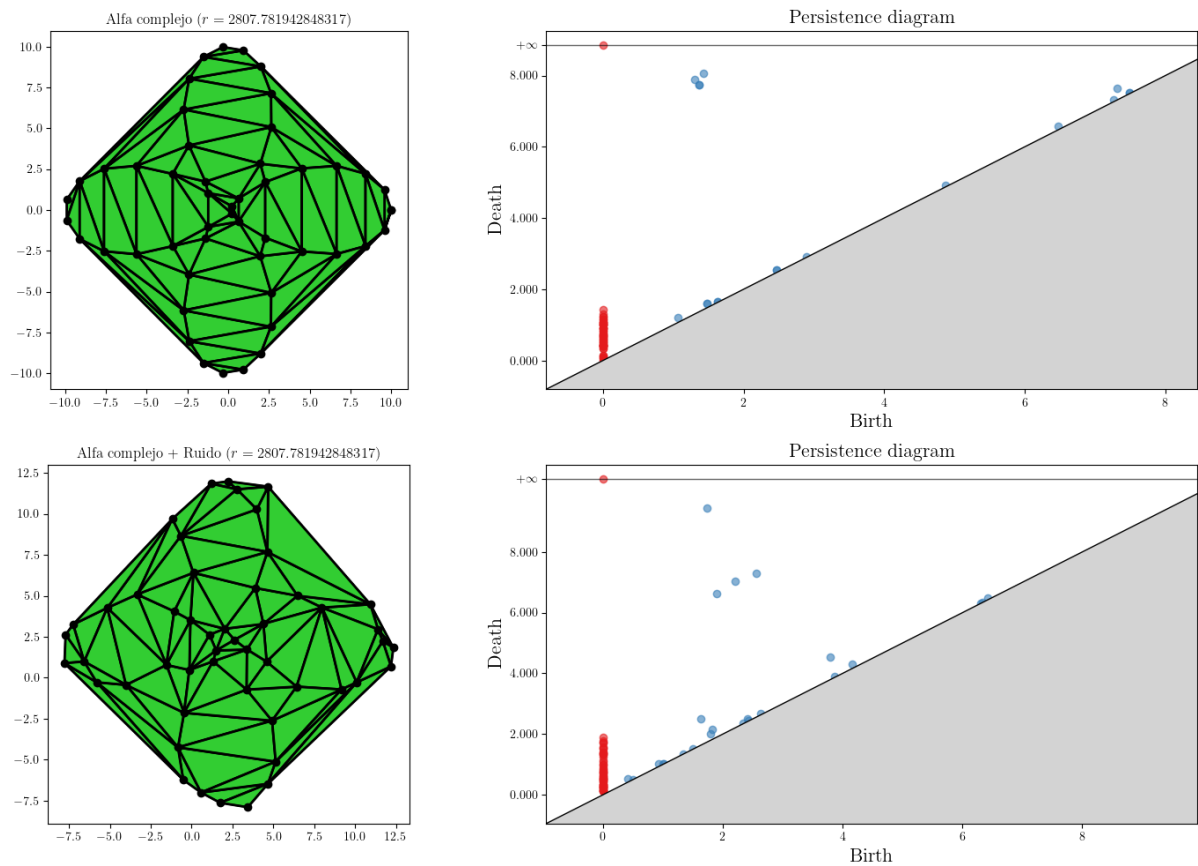


Figura 2.34: Alfa complejos y diagramas de persistencias asociados a las filtraciones de los alfa complejos. En las figuras superiores se toma como entrada una rosa polar y en las inferiores la misma rosa polar mas cierto ruido.

distancia Hausdorff entre los conjuntos de puntos, dando lugar a la robustez que nos garantiza el *Teorema de estabilidad*.

```

1 Alfa complejo de dimensión 2 - 263 símlices - 50 vértices.
2 Alfa complejo de dimensión 2 - 267 símlices - 50 vértices.
3
4 Distancia Hausdorff: 2.66069
5 Distancia bottleneck dimensión 0: 0.50235 (Implementación GUDHI: 0.50235)
6 Distancia bottleneck dimensión 1: 1.39537 (Implementación GUDHI: 1.39537)

```

Observación. En los ejemplos se indica la distancia bottleneck calculada tanto por mi implementación como la implementación de la librería GUDHI [17] para poder garantizar que esta bien calculada, ya que se ha observado que mi implementación obtiene erróneamente el resultado en algunas ocasiones. Esto se debe a que como hemos visto en los dos últimos ejemplos, los puntos de los diagramas se encuentran muy cercanos los unos a los otros en algunas ocasiones, lo que ocasiona en distancias de orden inferior a la precisión del tipo *float*. Esto causa que se arrastren errores de número máquina a lo largo del proceso, ocasionando que se calculen erróneamente las distancias con el algoritmo de Dijkstra y por tanto que se obtenga un emparejamiento que no sea mínimo.

2.3.3.3. Ejemplo 3: Elecciones generales noviembre 2019

Por último veremos un caso más práctico. Para ello simulamos el procedimiento descrito en [18], donde se estudia como se puede hacer uso de la homología persistente para poder determinar si hay distritos electorales donde se ha votado diferente respecto al resto de distritos que lo rodean. Este tipo de estudio nos permitiría poder observar a gran escala si los votos condados en los condados son uniformes, o bien hay zonas donde se vota diferente.

Como se comenta en [18], este trabajo es especialmente adecuada para la homología persistente, ya que identificaremos condados con “distritos isla” como agujeros. Para ello a cada distrito se le asigna una coordenada y en el caso de Estados Unidos se le asigna un color representando el partido ganador (rojo republicano y azul democrata). En dicho artículo se plantean diversos complejos simpliciales y sus ventajas e inconvenientes para este tipo de tarea.

Así pues, he recreado este estudio para el caso de las elecciones generales del noviembre de 2019, donde he marcado a cada provincia del color del partido que obtuvo mas votos en dicha región. Nuestro interés desde el punto de vista de la estabilidad es poder garantizar que los resultados obtenidos en [18] son robustos independientemente de la coordenada elegida como representante de cada distrito/provincia.

Una vez obtenido el mapa toca decidir que coordenadas se toman para cada una de las provincias. Como coordenadas “canonicas” he usado las coordenadas que aparecen en la página de la Wikipedia de cada una de las provincias. Y como datos perturbados he elegido unas coordenadas arbitrarias de cada una de las provincias.

Cabe destacar que en este ejemplo no se han utilizado filtraciones de alfa complejos para obtener los diagramas de persistencia, sino que se han utilizado filtraciones de complejos de Vietoris-Rips. Esto se debe a que los alfa complejos requieren que los puntos se encuentren en posición general para obtener el complejo de Delaunay, lo cual podía no ocurrir al elegir puntos aleatorios.

Sin embargo el siguiente teorema nos garantiza la estabilidad para los diagramas de persistencia asociado a filtraciones de complejos de Vietoris-Rips:

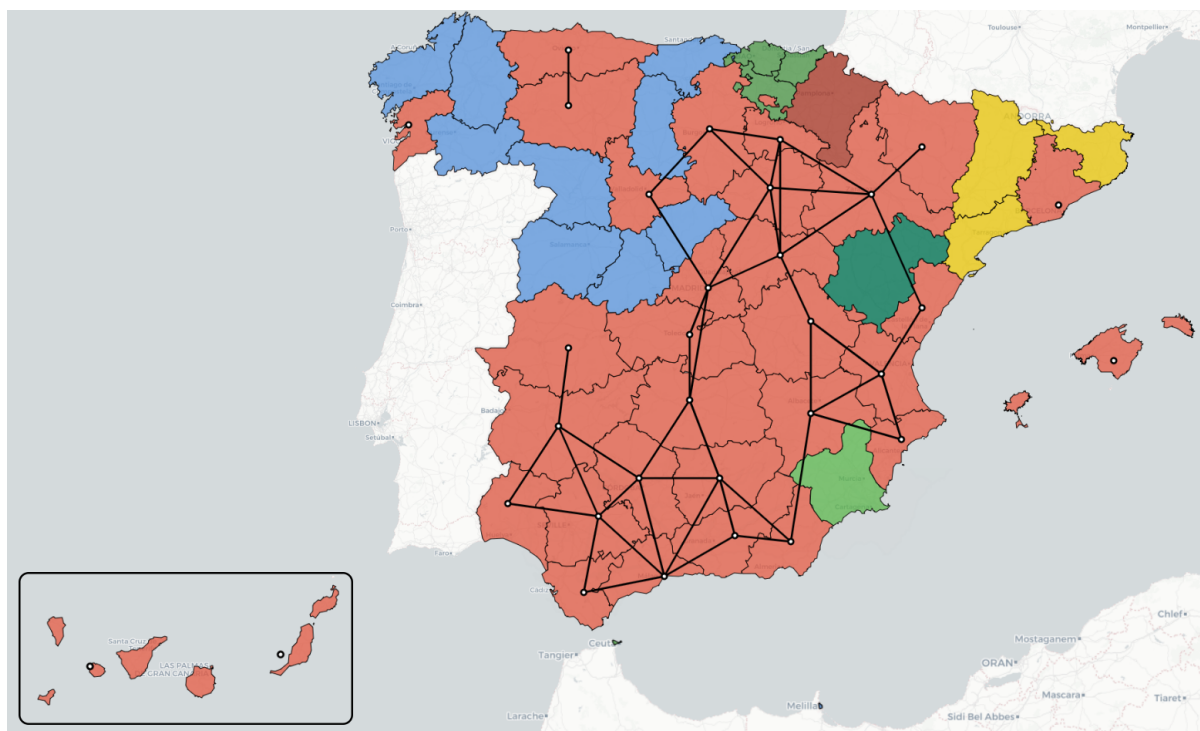
Teorema 2.3.5 (Teorema de estabilidad para complejos de Vietoris-Rips de espacios métricos acotados [19, Theorem 5.2]).

Sean X, Y dos espacios métricos acotados y $\text{Dgm}_k(X)$ el diagrama de persistencia de dimensión k de las filtraciones del complejo de Vietoris-Rips asociado a X . Entonces, para cada k

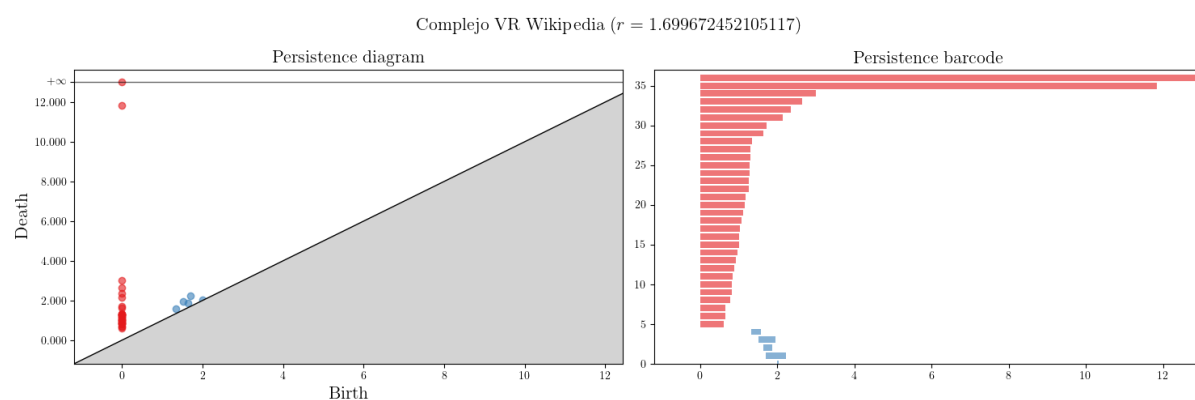
$$W_\infty(\text{Dgm}_k(X), \text{Dgm}_k(Y)) \leq 2H(X, Y).$$

De esta forma obtenemos el siguiente diagrama de persistencia y códigos de barras representados en la figura 2.35, al estudiar las filtraciones de los complejos de VR con vértices en las coordenadas dadas por Wikipedia de las provincias donde el PSOE ganó en las elecciones generales del noviembre de 2019.

Además, obtenemos el siguiente diagrama de persistencia y códigos de barras representados en la figura 2.36, al estudiar las filtraciones de los complejos de VR con vértices en unas coordenadas aleatorias sobre las provincias donde el PSOE ganó en las elecciones generales del noviembre de 2019.

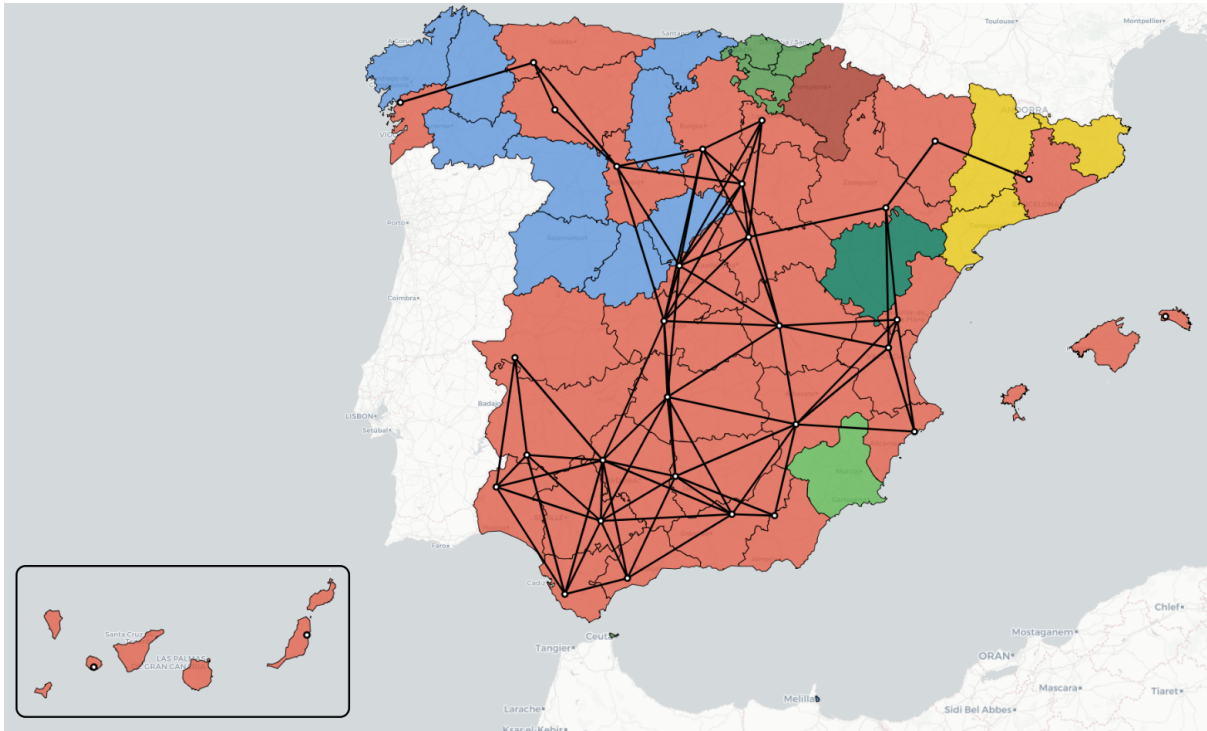


(a) Filtraciones del complejo de Vietoris-Rips con vértices en las coordenadas dadas por Wikipedia de las provincias donde el PSOE ganó en las elecciones generales del noviembre de 2019.



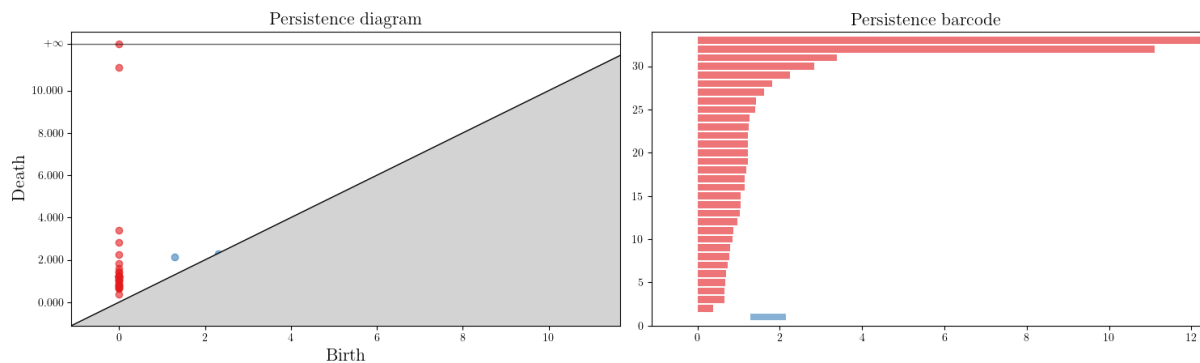
(b) Diagrama de persistencia y código de barras asociado a las filtraciones de dicho complejo de Vietoris-Rips.

Figura 2.35: Estudio de la persistencia sobre los datos originales.



(a) Filtración del complejo de Vietoris-Rips con vértices en unas coordenadas aleatorias sobre las provincias donde el PSOE ganó en las elecciones generales del noviembre de 2019.

Complejo VR Random ($r = 2.299065825181417$)



(b) Diagrama de persistencia y código de barras asociado a las filtraciones de dicho complejo de Vietoris-Rips.

Figura 2.36: Estudio de la persistencia al aplicar ruido a los datos.

Como hemos podido observar en las figuras 2.35b y 2.36b, los diagramas de persistencia son muy cercanos. Esto se observa también al calcular las distancias bottleneck. Cabe destacar que los diagramas de persistencia nos dan a pensar que nuestro mapa (restringido a las provincias que ganó el PSOE) sólo tiene dos componentes conexas, la zona peninsular (península y baleares) y las islas canarias, sin embargo sería más correcto si pudiéramos obtener 3 componentes conexas bien identificables como mínimo. Siguiendo el estudio de las componentes conexas también se muestran unas clases que sobreviven un poco más que el resto, las cuales hacen referencia a Pontevedra y Barcelona, las cuales no se encuentran conectadas con el bloque central.

Desarrollo

Desde el punto de vista de los agujeros podemos observar que en ambos casos se muestra a Teruel como agujero, sin embargo en el caso de los puntos obtenidos de las coordenadas de Wikipedia la clase de homología de dimensión 1 que más tiempo sobrevive realmente no representa ninguna isla de voto, simplemente es producto del complejo de Vietoris-Rips, lo que nos indica (al igual que se observa en [18]) que este tipo de complejo simplicial no sea el más apropiado para esta tarea.

Desde el punto de vista de la estabilidad podemos observar que la distancia bottleneck es inferior al doble de la distancia Hausdorff entre los conjuntos de puntos, dando lugar a la robustez que nos garantiza el *Teorema de estabilidad*.

```
1 Complejo de Vietoris-Rips de dimensión 2 - 5488 símlices - 32 vértices.  
2 Arista que genera la clase de homología de dimensión 1 que más vive: (Albacete-Almeria)  
3  
4 Complejo de Vietoris-Rips de dimensión 2 - 5488 símlices - 32 vértices.  
5 Arista que genera la clase de homología de dimensión 1 que más vive: (Guadalajara-Zaragoza)  
6  
7 Distancia Hausdorff: 0.89583  
8 Distancia bottleneck dimensión 0: 0.7323 (Implementación GUDHI: 0.7323)  
9 Distancia bottleneck dimensión 1: 0.2682 (Implementación GUDHI: 0.2682)
```


Capítulo 3

Resultados y conclusiones

Sección por hacer

Resumen de resultados obtenidos en el TFG. Y conclusiones personales del estudiante sobre el trabajo realizado.

Capítulo 4

Análisis de impacto

Sección por hacer

En este capítulo se realizará un análisis del impacto potencial de los resultados obtenidos durante la realización del TFG, en los diferentes contextos para los que se aplique:

- Personal
- Empresarial
- Social
- Económico
- Medioambiental
- Cultural

En dicho análisis se destacarán los beneficios esperados, así como también los posibles efectos adversos.

Se recomienda analizar también el potencial impacto respecto a los Objetivos de Desarrollo Sostenible (ODS), de la Agenda 2030, que sean relevantes para el trabajo realizado (ver enlace)

Además, se harán notar aquellas decisiones tomadas a lo largo del trabajo que tienen como base la consideración del impacto.

Bibliografía

- [1] D. Reinsel, J. Gantz, and J. Rydning. Data age 2025: The evolution of data to life-critical. Seagate. [Online]. Available: <https://www.import.io/wp-content/uploads/2017/04/Seagate-WP-DataAge2025-March-2017.pdf>
- [2] H. Edelsbrunner and J. Harer, *Computational Topology: An Introduction*. American Mathematical Society, 01 2010.
- [3] H. Edelsbrunner and J. Harer, “Persistent homology—a survey,” *Discrete & Computational Geometry - DCG*, vol. 453, 01 2008.
- [4] D. Cohen-Steiner, H. Edelsbrunner, and J. Harer, “Stability of persistence diagrams,” *Discrete & Computational Geometry*, vol. 37, no. 1, pp. 103–120, Jan 2007. [Online]. Available: <https://doi.org/10.1007/s00454-006-1276-5>
- [5] M. Ulmer, L. Ziegelmeier, and C. M. Topaz, “A topological approach to selecting models of biological experiments,” *PLOS ONE*, vol. 14, no. 3, p. e0213679, Mar. 2019. [Online]. Available: <https://doi.org/10.1371/journal.pone.0213679>
- [6] P. Yale, *Geometry and Symmetry*, ser. Dover books on advanced mathematics. Dover Publications, 2014. [Online]. Available: <https://books.google.es/books?id=PjOIBQAAQBAJ>
- [7] A. Hatcher, *Algebraic topology*. Cambridge: Cambridge University Press, 2002.
- [8] M. D. Crossley, *Essential Topology*. Springer London, 2005.
- [9] J. Curry. Counting embedded spheres with the same persistence. University at Albany SUNY. [Online]. Available: <http://www.fields.utoronto.ca/talks/Counting-Embedded-Spheres-same-Persistence>
- [10] J. Curry, “The fiber of the persistence map for functions on the interval,” 2019.
- [11] H. Ricardo, *A Modern Introduction to Linear Algebra*. Chapman and Hall/CRC, Oct. 2009. [Online]. Available: <https://doi.org/10.1201/b16027>
- [12] X. Kong. Stability theorem. Eindhoven, university of Technology. [Online]. Available: <https://www.win.tue.nl/~kbuchin/teaching/2IMA00/2018/Slides/stability.pdf>
- [13] B. Binegar. Lecture 14: The isomorphism theorems. Oklahoma State University. [Online]. Available: <https://math.okstate.edu/people/binegar/4063-5023/4063-5023-114.pdf>

- [14] W. Rudin, *Principles of mathematical analysis*, 3rd ed. McGraw-Hill New York, 1976. [Online]. Available: <http://www.loc.gov/catdir/toc/mh031/75017903.html>
- [15] A. A. Taha and A. Hanbury, “An efficient algorithm for calculating the exact hausdorff distance,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 11, pp. 2153–2163, Nov. 2015. [Online]. Available: <https://doi.org/10.1109/tpami.2015.2408351>
- [16] H. W. Kuhn, “The hungarian method for the assignment problem,” *Naval Research Logistics Quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800020109>
- [17] F. Godi, “Bottleneck distance,” in *GUDHI User and Reference Manual*, 3rd ed. GUDHI Editorial Board, 2021. [Online]. Available: https://gudhi.inria.fr/doc/3.4.1/group__bottleneck__distance.html
- [18] M. Feng and M. A. Porter, “Persistent homology of geospatial data: A case study with voting,” 2019.
- [19] F. Chazal, V. de Silva, and S. Oudot, “Persistence stability for geometric complexes,” 2013.

Anexos

.1. Implementación de las distancias Hausdorff y bottleneck

Anexo 1: Implementación de las distancias Hausdorff y bottleneck

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Tue May  4 15:45:56 2021.
4
5 @author: Alejandro
6 """
7
8 import numpy as np
9 import networkx as nx
10 from networkx.algorithms import bipartite
11 import matplotlib.pyplot as plt
12 import sympy as sy
13 import gudhi
14
15 # variable curvas
16 t = sy.symbols('t', real=True)
17
18
19 def distanciaEuclidea(p1, p2):
20     """
21     Distancia euclídea entre los puntos p1 y p2.
22
23     p1: list.
24     p2: list.
25     """
26     p1Np = np.array(p1)
27     p2Np = np.array(p2)
28     return np.sqrt(np.dot(p1Np - p2Np, p1Np - p2Np))
29
30
31 def distanciaInf(x, y):
32     """
33     Calcular la distancia infiniro entre los puntos x e y.
34
35     x: numpy.ndarray.
36     y: numpy.ndarray.
37     """
38     res = 0
39     if x[1] == float('inf') and y[1] == float('inf'):
40         res = abs(x[0]-y[0])
41     elif x[1] == float('inf') and y[1] != float('inf'):
42         res = float('inf')
43     elif x[1] != float('inf') and y[1] == float('inf'):
44         res = float('inf')
45     else:
46         res = max(abs(x[0]-y[0]), abs(x[1]-y[1]))
47
```

.1. Implementación de las distancias Hausdorff y bottleneck

```
48     return res
49
50
51 def hausdorffDir(A, B):
52     """
53     Calcular la distancia Hausdorff directa entre los conjuntos de puntos A y B.
54
55     A: numpy.array.
56     B: numpy.array.
57     """
58     cmax = 0
59     for x in A:
60         cmin = float('inf')
61         for y in B:
62             d = distanciaInf(x, y)
63             if d < cmin:
64                 cmin = d
65
66         if cmin > cmax:
67             cmax = cmin
68
69     return cmax
70
71
72 def hausdorff(A, B):
73     """
74     Calcular la distancia Hausdorff entre los conjuntos de puntos A y B.
75
76     A: numpy.array.
77     B: numpy.array.
78     """
79     return max(hausdorffDir(A, B), hausdorffDir(B, A))
80
81
82 def grafoBottleneck(X, Y):
83     """
84     Obtener el grafo bipartido de para el emparejamiento de la distancia bottleneck.
85
86     X: numpy.array.
87     y: numpy.array.
88     """
89     X0 = list()
90     X0_inf = list()
91
92     Y0 = list()
93     Y0_inf = list()
94     for x in X:
95         if x[0] != x[1]:
96             if x[1] == float("inf"):
97                 X0_inf.append(x[0])
98             else:
99                 X0.append((x[0], x[1]))
100
101     for y in Y:
102         if y[0] != y[1]:
103             if y[1] == float("inf"):
104                 Y0_inf.append(y[0])
105             else:
106                 Y0.append((y[0], y[1]))
107
108     X0_ = [(x[0]+x[1])/2, (x[0]+x[1])/2] for x in X0
109     Y0_ = [(y[0]+y[1])/2, (y[0]+y[1])/2] for y in Y0
110     U = X0 + Y0_
111     V = Y0 + X0_
112     n = len(U)
113     G = nx.Graph()
114     G.add_nodes_from([(f"u{i}", {'coord': U[i]}) for i in range(0, n)], bipartite=0)
115     G.add_nodes_from([(f"v{i}", {'coord': V[i]}) for i in range(0, n)], bipartite=1)
116     edges = list()
117     for i in range(0, n):
```

```
118         for j in range(0, n):
119             d = 0.0
120             if U[i] in X0 or V[j] in Y0:
121                 d = distanciaInf(U[i], V[j])
122
123             edges.append((f"u{i}", f"v{j}", d))
124
125         G.add_weighted_edges_from(edges)
126
127         distPinf = 0
128
129         if len(X0_inf) != len(Y0_inf):
130             distPinf = float("inf")
131         else:
132             distPinf = max([abs(x-y) for x, y in zip(sorted(X0_inf), sorted(Y0_inf))]+[0])
133
134         return G, U, V, distPinf
135
136
137 def subgrafoG(G, i):
138     """
139     Obtener el subgrafo de G formado por las aristas que tienen peso menor o igual a i.
140
141     G: networkx.Graph.
142     i: float.
143     """
144     G0 = nx.Graph([(u, v, d) for u, v, d in G.edges(data=True) if d['weight'] <= i])
145     for n, d in G0.nodes(data=True):
146         d["bipartite"] = G.nodes[n]["bipartite"]
147
148     return G0
149
150
151 def subgrafoG_todosV(G, i):
152     """
153     Obtener el subgrafo generador de G formado por las aristas que tienen peso menor o igual a
154     i.
155
156     G: networkx.Graph.
157     i: float.
158     """
159     G0 = nx.Graph()
160     G0.add_nodes_from(G.nodes(data=True))
161     G0.add_edges_from([(u, v, d) for u, v, d in G.edges(data=True) if d['weight'] <= i])
162
163     return G0
164
165 def cambiarPesos(Gi, length):
166     """
167     Obtener los pesos de la nueva iteración a partir del diccionario de distancia obtenido de
168     Dijkstra.
169
170     Gi: networkx.Graph.
171     length: dict.
172     """
173     for u, v, d in Gi.edges(data=True):
174         d['weight'] += length[u] - length[v]
175         if d['weight'] < 0:
176             d['weight'] = 0
177
178     return 0
179
180 def digrafoAsociado(G, M, U=None, V=None):
181     """
182     Obtener el digrafo asociado al emparejamiento M y el grafo bipartido M con vertices U+V.
183
184     Gi: networkx.Graph.
185     M: dict.
```

.1. Implementación de las distancias Hausdorff y bottleneck

```
186 U: list.
187 V: list.
188 """
189 if U is None or V is None:
190     U, V = bipartite.sets(G)
191
192 D = nx.DiGraph()
193 keysM = M.keys()
194
195 aristas = list()
196 for u, v, d in G.edges(data='weight'):
197     if u in U:
198         if u not in keysM or M[u] != v:
199             aristas.append((u, v, d))
200         else:
201             aristas.append((v, u, d))
202     else:
203         if u not in keysM or M[u] != v:
204             aristas.append((v, u, d))
205         else:
206             aristas.append((u, v, d))
207
208 D.add_weighted_edges_from(aristas)
209 D.add_weighted_edges_from([("s", u, 0) for u in U if u not in keysM])
210 D.add_weighted_edges_from([(v, "t", 0) for v in V if v not in keysM])
211
212 return D
213
214
215 def plotMatching(G, M, U):
216     """
217     Representar el emparejamiento M en el grafo bipartido G con vertices superiores en U.
218
219     G: networkx.Graph.
220     M: dict.
221     U: list.
222     """
223     plt.figure(figsize=(8, 6))
224     pos = nx.drawing.layout.bipartite_layout(G, U)
225     nx.draw_networkx(G, pos=pos)
226     edgelist = [(u, M[u]) for u in U if u in M.keys()]
227     nx.draw_networkx_edges(G, pos, edgelist=edgelist, width=2.5, edge_color='blue')
228
229
230 def bottleneck(X, Y, plot=False, debug=False):
231     """
232     Calcular la distancia bottleneck entre los conjuntos de puntos X y Y.
233
234     Si se quiere que se muestren los emparejamientos poner plot=True.
235     Si se quiere información de la traza debug=True.
236
237     A: numpy.array.
238     B: numpy.array.
239     plot: boolean.
240     debug: boolean.
241     """
242     G, _, _, distPinf = grafoBottleneck(X, Y)
243     if distPinf == float("inf"):
244         return float("inf")
245     else:
246         U, V = bipartite.sets(G)
247         Gi = G.copy()
248         Mi = dict()
249
250         if plot:
251             plotMatching(Gi, Mi, U)
252
253         while not nx.is_perfect_matching(G, Mi):
254             Di = digrafoAsociado(Gi, Mi, U, V)
255             length, path = nx.single_source_dijkstra(Di, "s")
```



```
256     caminoAumento = path["t"][1:-1]
257
258     for i in range(0, len(caminoAumento)):
259         x = caminoAumento[i]
260
261         if i % 2 == 0:
262             Mi[x] = caminoAumento[i+1]
263         else:
264             Mi[x] = caminoAumento[i-1]
265
266     if plot:
267         plotMatching(Gi, Mi, U)
268
269     if debug:
270         print("\n", {u: Mi[u] for u in U if u in Mi}, "\n")
271         print(nx.is_perfect_matching(subgrafoG_todosV(Gi, 0), Mi), "\n")
272         print("MAX", max([G[u][Mi[u]]["weight"] for u in U if u in Mi.keys()]+[
273             distPinf]), "\n")
274         print(caminoAumento, "\n")
275         print([(u, v, d) for u, v, d in Gi.edges(data=True) if d["weight"] < 0], "\n")
276
277     cambiarPesos(Gi, length)
278
279     return max([G[u][Mi[u]]["weight"] for u in U if u in Mi.keys()]+[distPinf])
280
281 if __name__ == "__main__":
282
283     A = np.array([(2, 4), (3, 2), (0, 0), (0, 0.8), (4, 5.2)])
284     B = np.array([(2.8, 4), (3, 3), (4.2, 5.8)])
285     C = np.array([(2, 4), (4, 2), (0, 0)])
286     D = np.array([(2.8, 4), (4.8, 2.8), (0, 0.8)])
287
288     diag1 = np.array([(2.7, 3.7), (9.6, 14.), (34.2, 34.974), (3., float('inf'))])
289     diag2 = np.array([(2.8, 4.45), (9.5, 14.1), (3.2, float('inf'))])
290
291     G, _, _, d = grafoBottleneck(diag1, diag2)
292
293     print("Hausdorff: ", hausdorffDir(diag1, diag2), "\n")
294     # print(G.nodes(data=True), "\n")
295     # print(G.edges(data=True), "\n")
296     print("Bottleneck: ", bottleneck(diag1, diag2), "=", gudhi.bottleneck_distance(diag1,
297         diag2))
```

.2. Visualización de la función distancia a un conjunto

Anexo 2: Visualización de la función distancia a un conjunto

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Sat May 22 16:54:05 2021.
4
5 @author: Alejandro
6 """
7
8 from numpy import arange
9 from pylab import meshgrid, cm, imshow, contour, clabel, colorbar, title
10 import math
11 import numpy as np
12 import matplotlib as mpl
13 from matplotlib.ticker import FormatStrFormatter
14 import matplotlib.pyplot as plt
15
16 # Directorios datos
17 pathOut = r"output/"
18
```

```
19
20 def z_func(x, y, A):
21     """
22     Calcular la distancia entre el punto (x, y) y el conjunto A.
23
24     x: float.
25     y: float.
26     A: list.
27     """
28     p1 = (x, y)
29     return min([math.sqrt(sum([(a - b) ** 2 for a, b in zip(p1, p2)])) for p2 in A])
30
31
32 mpl.rcParams['figure.dpi'] = 100
33
34 A = [(-1.0, 0.0), (1.0, 0.0), (1.0, 1.0), (-1.0, -1.0)]
35 x = arange(-3.0, 3.0, 0.1)
36 y = arange(-3.0, 3.0, 0.1)
37 # grid of point
38 X, Y = meshgrid(x, y)
39 # evaluation of the function on the grid
40 Z = np.array([z_func(x1, y1, A) for x1 in x for y1 in y])
41 # drawing the function
42 im = imshow(Z, cmap=cm.RdBu)
43 # adding the Contour lines with labels
44 cset = contour(Z, arange(-1, 1.5, 0.2), linewidths=2, cmap=cm.Set2)
45 clabel(cset, inline=True, fmt='%1.1f', fontsize=10)
46 # adding the colobar on the right
47 colorbar(im)
48 # latex fashion title
49 title('$z=d((x,y), A)$')
50 plt.savefig(pathOut + 'subnivelDistancias.png', dpi=200, bbox_inches='tight')
51 plt.show()
52
53 fig = plt.figure(dpi=200)
54 ax = fig.gca(projection='3d')
55 surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
56                       cmap=cm.RdBu, linewidth=0, antialiased=True)
57
58 ax.zaxis.set_major_formatter(FormatStrFormatter('%0.2f'))
59 fig.colorbar(surf, shrink=0.8, aspect=10, pad=0.1)
60
61 fig.savefig(pathOut + 'grafoDistancias.png', dpi=200, bbox_inches='tight')
62 plt.show()
```

.3. Implementación de los ejemplos

Anexo 3: Implementación de los ejemplos

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Fri May 14 23:34:04 2021.
4
5 @author: Alejandro
6 """
7 import numpy as np
8 import pandas as pd
9 import matplotlib.pyplot as plt
10 import gudhi
11 import folium
12 import sympy as sy
13 import matplotlib.colors
14 import distancias
15 import sys
16
17 # variable curvas
```

```
18 t = sy.symbols('t', real=True)
19
20 # Colores partidos España
21 coloresPartidos = {
22     "PSOE": "#DD5C47",
23     "PP": "#5C95DA",
24     "ERC": "#E6C410",
25     "VOX": "#5CB451",
26     "JxC": "#B6194D",
27     "Bildu": "#6BE955",
28     "PNV": "#4F954A",
29     "NA+": "#A83B2A",
30     "UP": "#8961AA",
31     "PRC": "#269A1B",
32     "CC": "#FFEF4B",
33     "Cs": "#DF9B07",
34     "TEX": "#037252"
35 }
36
37 # Directorios datos
38 pathIn = r"input/"
39 pathOut = r"output/"
40
41 def plotalpha(ac, st, k, ax):
42     """
43     Representar el alpha complejo del complejo K.
44
45     puntos: np.array.
46     K: Complejo.
47     """
48     puntos = np.array([ac.get_point(i) for i in range(st.num_vertices())])
49
50     triangulos = [s[0] for s in st.get_skeleton(2) if len(s[0]) == 3 and s[1] <= k]
51     if triangulos:
52         c = np.ones(len(puntos))
53         cmap = matplotlib.colors.ListedColormap("limegreen")
54         ax.tripcolor(puntos[:, 0], puntos[:, 1], [s[0] for s in st.get_skeleton(2) if len(s[0]) == 3 and s[1] <= k],
55                     c, edgecolor="k", lw=2, cmap=cmap)
56
57     ax.plot(puntos[:, 0], puntos[:, 1], 'ko')
58
59     aristas = [s[0] for s in st.get_skeleton(2) if len(s[0]) == 2 and s[1] <= k]
60     if aristas:
61         for arista in aristas:
62             p1 = puntos[arista[0]]
63             p2 = puntos[arista[1]]
64             ax.plot([p1[0], p2[0]], [p1[1], p2[1]], 'k')
65
66     ax.axis('scaled')
67
68
69 def puntosCurvaRuido(curva, t, t0, t1, numPuntos=10, mu=0, sigma=0.1):
70     """
71     Obtener conjunto discretos de puntos de una curva con ruido.
72
73     curva: list.
74     t: t sympy symbol
75     t0: float
76         Inicio intervalo.
77     t1: float
78         Final intervalo.
79     numPuntos: int. Por defecto 10.
80     mu: float. Por defecto 0
81         Media para la distribución normal.
82     sigma: float. Por defecto 0.1
83         Desviación típica para la distribución normal.
84     """
85     valores = np.linspace(t0, t1, num=numPuntos)
86     puntosCurva = np.array([[x.subs(t, v) for x in curva] for v in valores], dtype=np.float64)
```

```
87     ruido = np.random.normal(mu, sigma, [numPuntos, len(curva)])
88
89     return puntosCurva + ruido
90
91
92 def ejemploPlano(points1, points2, indice, mapa=None, provincias=None):
93     """
94     Calcular la distancia Hausdorff entre los conjuntos de puntos y la distancia bottleneck
95     entre sus diagramas de persistencia.
96
97     Si se va a usar el mapa se utilizarán complejos de Vietoris-Rips y en otro caso alfa
98     complejos
99
100    points1: numpy.array.
101    points2: numpy.array.
102    indice: String.
103            String indicativo del ejemplo a ejecutar.
104    mapa: folium.map.
105    provincias: list.
106    """
107    # -----Primer conjunto de puntos-----
108
109    if mapa is None:
110        # Inicializamos la figura
111        fig, axs = plt.subplots(2, 2, dpi=100)
112        fig.set_size_inches(15, 10)
113
114        # Calculamos el alfa complejo
115        complex1 = gudhi.AlphaComplex(points=points1)
116        simplex_tree1 = complex1.create_simplex_tree()
117        nameComp = "Alfa complejo"
118    else:
119        # Inicializamos la figura
120        fig, axs = plt.subplots(1, 2, dpi=100)
121        fig.set_size_inches(15, 5)
122        fig2, axs2 = plt.subplots(1, 2, dpi=100)
123        fig2.set_size_inches(15, 5)
124
125        # Calculamos el complejo de Vietoris-Rips
126        complex1 = gudhi.RipsComplex(points=points1)
127        simplex_tree1 = complex1.create_simplex_tree(max_dimension=2)
128        nameComp = "Complejo de Vietoris-Rips"
129
130    print(nameComp + ' de dimensión ' + repr(simplex_tree1.dimension()) + ' - ' +
131          repr(simplex_tree1.num_simplices()) + ' simplices - ' +
132          repr(simplex_tree1.num_vertices()) + ' vértices.'
133          )
134
135    # Obtenemos el diagrama de persistencia
136    diag1 = simplex_tree1.persistence()
137
138    if mapa is None:
139        # Obtener valor filtración
140        value = max(set([filtered_value[1] for filtered_value in simplex_tree1.get_filtration()]))
141
142        # Representar alfa complejo de dicha filtración
143        plotalpha(complex1, simplex_tree1, value, axs[0, 0])
144        axs[0, 0].title.set_text(r"Alfa complejo $(r = {})$".format(str(value)))
145
146        # Representar el diagrama de persistencia
147        gudhi.plot_persistence_diagram(diag1, axes=axs[0, 1])
148    else:
149        # Obtener valor filtración
150        valueM = max([s[1][1] for s in diag1 if s[0] == 1])
151        value = max([s[1][0] for s in diag1 if s[1][1] == valueM])
152        arista = [s for s, v in simplex_tree1.get_filtration() if v == value][0]
153        print("Arista que genera la clase de homología de dimensión 1 que más vive:",
154              "(" + provincias[arista[0]] + "-" + provincias[arista[1]] + ")\n")
155
156    # Representar complejo de Vietoris-Rips de dicha filtración sobre el mapa
```

```

154     sub_group1 = folium.FeatureGroup(name=f"Complejo VR Wikipedia (r = {value})", control=
155         True, show=True)
156     plotAlphaMapa(points1, simplex_tree1, value, mapa, sub_group1)
157
158     # Representar el diagrama de persistencia y codigo de barras
159     fig.suptitle(r"Complejo VR Wikipedia $(r = {})$".format(str(value)), fontsize=16)
160     gudhi.plot_persistence_diagram(diag1, axes=axes[0])
161     gudhi.plot_persistence_barcode(diag1, axes=axes[1])
162
163     # -----Segundo conjunto de puntos-----
164
165     if mapa is None:
166         # Calculamos el alfa complejo
167         complex2 = gudhi.AlphaComplex(points=points2)
168         simplex_tree2 = complex2.create_simplex_tree()
169     else:
170         # Calculamos el complejo de Vietoris-Rips
171         complex2 = gudhi.RipsComplex(points=points2)
172         simplex_tree2 = complex2.create_simplex_tree(max_dimension=2)
173
174     print(nameComp + ' de dimensión ' + repr(simplex_tree2.dimension()) + ' - ' +
175           repr(simplex_tree2.num_simplices()) + ' simplices - ' +
176           repr(simplex_tree2.num_vertices()) + ' vértices.'
177           )
178
179     # Obtenemos el diagrama de persistencia
180     diag2 = simplex_tree2.persistence()
181     if mapa is None:
182         # Obtener valor filtración
183         value = max(set([filtered_value[1] for filtered_value in simplex_tree1.get_filtration
184             ()]))
185
186         # Representar alfa complejo de dicha filtración
187         plotalpha(complex2, simplex_tree2, value, axes[1, 0])
188         axes[1, 0].title.set_text(r"Alfa complejo $+ $ Ruido $(r = {})$".format(str(value)))
189
190         # Representar el diagrama de persistencia
191         gudhi.plot_persistence_diagram(diag2, axes=axes[1, 1])
192     else:
193         # Obtener valor filtración
194         valueM = max([s[1][1] for s in diag2 if s[0] == 1])
195         value = max([s[1][0] for s in diag2 if s[1][1] == valueM])
196         arista = [s for s, v in simplex_tree2.get_filtration() if v == value][0]
197         print("Arista que genera la clase de homología de dimensión 1 que más vive:",
198               (" + provincias[arista[0]] + "-" + provincias[arista[1]] + ")")
199
200         # Representar complejo de Vietoris-Rips de dicha filtración sobre el mapa
201         sub_group2 = folium.FeatureGroup(name=f"Complejo VR Random (r = {value})", control=
202             True, show=False)
203         plotAlphaMapa(points2, simplex_tree2, value, mapa, sub_group2)
204
205         # Representar el diagrama de persistencia y codigo de barras
206         fig2.suptitle(r"Complejo VR Random $(r = {})$".format(str(value)), fontsize=16)
207         gudhi.plot_persistence_diagram(diag2, axes=axes2[0])
208         gudhi.plot_persistence_barcode(diag2, axes=axes2[1])
209
210         # Ajustar figura y guardarla en directorio de trabajo
211         fig.tight_layout()
212         if mapa is not None:
213             fig.subplots_adjust(top=0.85)
214             fig2.tight_layout()
215             fig2.subplots_adjust(top=0.85)
216             fig2.savefig(pathOut + f'ejemplo{indice}.png', dpi=100, bbox_inches='tight')
217
218         fig.savefig(pathOut + f'ejemplo{indice}.png', dpi=100, bbox_inches='tight')
219         plt.show()
220
221     # -----Calculo de las distancias-----
222
223     # Obtener los puntos de dimensión 0 de los diagramas

```

.3. Implementación de los ejemplos

```
221 diag1_0 = np.array([s[1] for s in diag1 if s[0] == 0])
222 diag2_0 = np.array([s[1] for s in diag2 if s[0] == 0])
223
224 print("\nDistancia Hausdorff: ", round(distancias.hausdorffDir(points1, points2), 5))
225
226 print("Distancia bottleneck dimensión 0:", round(distancias.bottleneck(diag1_0, diag2_0),
227     5),
228     "(Implementación GUDHI:", str(round(gudhi.bottleneck_distance(diag1_0, diag2_0), 5))
229     + ")")
230
231 # Obtener los puntos de dimensión 1 de los diagramas
232 diag1_1 = np.array([s[1] for s in diag1 if s[0] == 1])
233 diag2_1 = np.array([s[1] for s in diag2 if s[0] == 1])
234 print("Distancia bottleneck dimensión 1:", round(distancias.bottleneck(diag1_1, diag2_1),
235     5),
236     "(Implementación GUDHI:", str(round(gudhi.bottleneck_distance(diag1_1, diag2_1), 5))
237     + ")")
238
239
240 def ejemplo1(backup=True):
241     """
242     Ejecutar el ejemplo 1: Estudio de la estabilidad en una elipse.
243
244     backup: boolean.
245         True si quieres usar los puntos guardados en los ficheros puntos1_1.npy y puntos1_2.
246         npy.
247     """
248     if backup:
249         with open(pathIn + 'puntos1_1.npy', 'rb') as f:
250             points1 = np.load(f)
251
252         with open(pathIn + 'puntos1_2.npy', 'rb') as f:
253             points2 = np.load(f)
254     else:
255         curva = [4 * sy.sin(t), 9 * sy.cos(t)]
256         points1 = puntosCurvaRuido(curva, t, 0, 2*np.pi, numPuntos=30)
257         points2 = puntosCurvaRuido(curva, t, 0, 2*np.pi, numPuntos=30, mu=2, sigma=0.3)
258
259     print("-----INICIO:Ejemplo1-----")
260     ejemploPlano(points1, points2, 1)
261     print("-----FIN:Ejemplo1-----\n")
262
263     return points2
264
265 def ejemplo2(backup=True):
266     """
267     Ejecutar el ejemplo 2: Estudio de la estabilidad en una rosa polar.
268
269     backup: boolean.
270         True si quieres usar los puntos guardados en los ficheros puntos2_1.npy y puntos2_2.
271         npy.
272     """
273     if backup:
274         with open(pathIn + 'puntos2_1.npy', 'rb') as f:
275             points1 = np.load(f)
276
277         with open(pathIn + 'puntos2_2.npy', 'rb') as f:
278             points2 = np.load(f)
279     else:
280         curva = [10 * sy.cos(2 * t) * sy.cos(t),
281                 10 * sy.cos(2 * t) * sy.sin(t)]
282         points1 = puntosCurvaRuido(curva, t, 0, 2*np.pi, numPuntos=50, sigma=0.00)
283         points2 = puntosCurvaRuido(curva, t, 0, 2*np.pi, numPuntos=50, mu=2, sigma=0.5)
284
285     print("-----INICIO:Ejemplo2-----")
286     ejemploPlano(points1, points2, 2)
287     print("-----FIN:Ejemplo2-----\n")
288
289     return points2
```

```
285
286
287 def estiloMapa(feature, elecciones):
288     """
289     Representación de las provincias segun el color del partido politico ganador.
290
291     feature: GeoJson feature.
292     elecciones: pandas.DataFrame.
293     """
294     provincia = feature['properties']['texto']
295     partido = elecciones.loc[elecciones["Provincia"] == provincia, "Partido"].values[0]
296     provincia_style = {
297         'weight': 1,
298         'fillOpacity': 0.8,
299         'fillColor': coloresPartidos[partido],
300         'color': '#000000'
301     }
302
303     return provincia_style
304
305
306 def plotAlphaMapa(puntos, st, k, m, sub_group):
307     """
308     Representar el alpha complejo del complejo K.
309
310     puntos: np.array.
311     K: Complejo.
312     """
313     aristas = [s[0] for s in st.get_skeleton(2) if len(s[0]) == 2 and s[1] <= k]
314     if aristas:
315         for arista in aristas:
316             p1 = puntos[arista[0]].tolist()
317             p2 = puntos[arista[1]].tolist()
318             folium.PolyLine([p1, p2], weight=3, color='black').add_to(sub_group)
319
320     for p in puntos:
321         folium.CircleMarker(p.tolist(), radius=4, fill=True, fill_opacity=1,
322                             fill_color='white', color='black').add_to(sub_group)
323     sub_group.add_to(m)
324
325
326 def ejemploMapa():
327     """Ejecutar el ejemplo 3: Estudio de la estabilidad en el mapa de España de las elecciones
328     nacionales de Nov 2019."""
329     m = folium.Map(
330         location=[36.2, -4],
331         tiles="CartoDB Positron",
332         zoom_start=6,
333     )
334
335     elecciones = pd.read_csv(pathIn + "elecciones2019Nov.csv", index_col="id_provincia")
336     eleccionesPSOE = elecciones[elecciones["Partido"] == "PSOE"]
337     coordenadasWiki = np.array([tuple(elem) for elem in eleccionesPSOE[["Latitud", "Longitud"]].values.tolist()])
338     coordenadasRandom = np.array([tuple(elem) for elem in eleccionesPSOE[["LatitudR", "LongitudR"]].values.tolist()])
339
340     folium.GeoJson(pathIn + r"spain_provincias.geojson", name="Partidos Ganadores", control=False,
341                   style_function=(lambda feature: estiloMapa(feature, elecciones))).add_to(m)
342
343     print("-----INICIO:Ejemplo3-----")
344     ejemploPlano(coordenadasWiki, coordenadasRandom, "Mapa", mapa=m, provincias=eleccionesPSOE[["Provincia"]].values.tolist())
345     print("-----FIN:Ejemplo3-----\n")
346
347     folium.LayerControl(collapsed=False).add_to(m)
348     m.save(pathOut + "mapSpain.html")
349     return elecciones, coordenadasRandom
```

.3. Implementación de los ejemplos

```
350
351 if __name__ == "__main__":
352     orig_stdout = sys.stdout
353     log_file = open(r"ejemplos.log", 'w', encoding='utf-8')
354     sys.stdout = log_file
355
356     points1 = ejemplo1()
357     points2 = ejemplo2()
358     elecciones, coordenadasRandom = ejemploMapa()
359
360     sys.stdout = orig_stdout
361     log_file.close()
362
363     print ("Completado!")
```