Métodos adaptativos para el cálculo de homología persistente cúbica

Jaime Alconchel Gallego

Trabajo de fin de grado
Universidad de Zaragoza

Directores del trabajo:
Álvaro Lozano Rojo
Miguel Ángel Marco Buzunáriz

Fecha: 11 de julio de 2025

Abstract

Topological data analysis is a discipline that employs techniques from algebraic topology to study point clouds in metric spaces. Despite being fairly young for mathematical time standards, it has proven itself capable of providing relevant and useful information in machine learning that was unattainable through classical methods.

The most used tool in TDA is persistent homology, to the extent that it has become completeley identified with the whole discipline. However, other resources such as the mapper algorithm are also frequently used.

Even though homology theory is broad enough to allow the develpoment of different procedures for analyzing point clouds, only two of them are usually employed nowadays: persistent homology and zigzag homology, both over filtrations and principally over simplicial complexes (these notions will be introduced in the first chapter of the present work).

Our purpose is to develop a new TDA methodology that, without abstract simplicial complexes and filtrations, enables us to analyze n-dimensional point clouds taking advantage of their binary representation in a computer. Specifically, we employ various cubical complexes and a novel way of linking them with less trivial maps than the usual inclusions.

The developed implementation also has another advantage over the common ones: it allows us to begin at a minimum precision level and to increase it dynamically, minimizing the number of performed operations. Meanwhile, traditional implementations go the opposite way and adaptive calculus becomes in them more complex and expensive.

This proposal can be succesfully applied to cases with many points, obtaining in some of them a better performance than traditional methods based on simplicial complexes. Moreover, it opens the door (practically, for it was already opened theoretically) to the development of new methods for constructing completely different persistence complexes that will be able to benefit from part of the results obtained here.

This work presents the following structure:

- The first chapter serves as a general introduction to the theory of persistent homology and establishes the theoretical basis for the second and third chapters. It harmonizes the definitions and notation of the foundational works of TDA and of other classical texts of algebraic topology.

- The second chapter briefly describes cubical complexes and develops extensively the method we have conceived for generating and representing them. We also dedicate part of this chapter to prove that the generated complexes satisfy the properties required for calculating their persistent homology.

- The third chapter covers the two steps needed for the extraction of the barcode of a sequence of cubical complexes linked by maps. The first section explains an elemental method for computing homology groups and induced maps. The second one, in turn, exposes the method employed for computing the barcode from the induced maps.

- Finally, the fourth chapter focuses on the results obtained with the algorithm implementation, applied to point clouds generated from simple figures.

This documents includes numerous graphical examples of many of the definitions we introduce, for we believe that they greatly contribute to the comprehension of some concepts that, abtractly, result confusing. That happens mainly in the second chapter, where the computational nature of many of its definitions renders them difficult to address. We have also accompanied, with a similar purpose, most algorithms with simple examples.

Whether this method produces results comparable to traditional ones in more applied areas of TDA, such as medical image classification[1], biological structure[2], shape recognition[3], or the analysis of meteorological maps[4], remains an open question. The cases in which the analyzed data is in a certain way cubical or squared, such as bitmaps, result particularly interesting for further research.

Índice general

Capítulo 1

Introducción a la homología persistente

1.1. Símplices y complejos simpliciales

Definición 1.1 (Símplice)**.** Sean $k \in \{0,1,\dots\}$ y $d \in \{1,2,\dots\}$ con $k \leq d$. Llamamos k-símplice geométrico en $\mathbb{R}^d$ a la envolvente convexa de una familia afínmente independiente de $k+1$ puntos $V = \{v_0, v_1, \cdots, v_k\} \subset \mathbb{R}^d$. A un símplice generado por un subconjunto de vértices de otro símplice σ lo llamamos **cara** de σ.

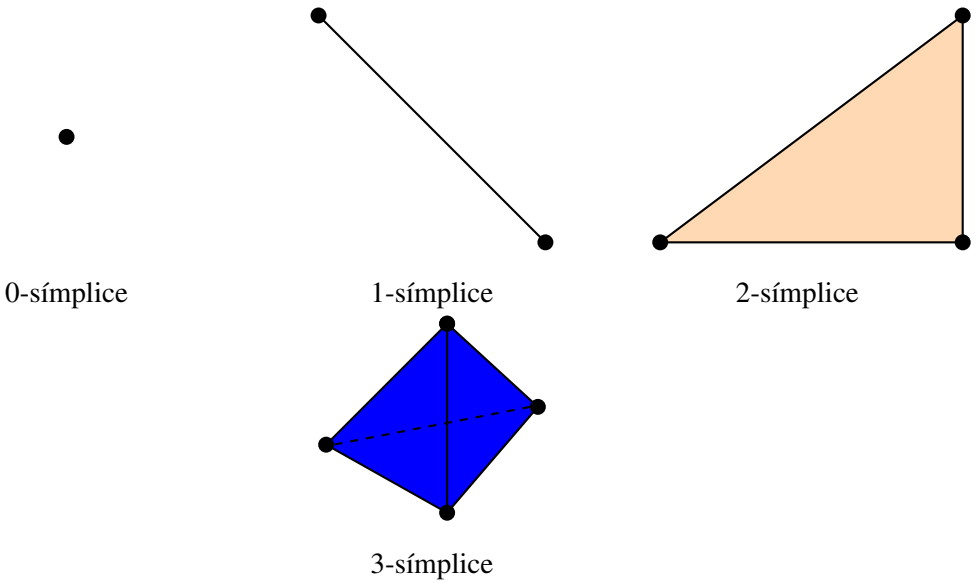


Figura 1.1: Ejemplos de símplices

Definición 1.2 (Complejo simplicial)**.** Sea $d \in \{0,1,\dots\}$. Un complejo simplicial geométrico $K \subseteq \mathbb{R}^d$ es una colección finita de símplices geométricos tal que:

1. Si $\sigma \in K$, y τ es una cara de σ, entonces $\tau \in K$.
2. Si $\sigma, \tau \in K$, entonces $\sigma \cap \tau$ es o bien vacía o una cara común de ambos.

Definición 1.3 (Complejo simplicial abstracto)**.** Un complejo simplicial abstracto K es una unión finita no vacía de subconjuntos para la que se cumple que para todo $\sigma_1 \in K$ y todo $\sigma_2 \subseteq \sigma_1,\ \sigma_2 \in K$.

La dimensión de un símplice abstracto es su cardinalidad como conjunto menos 1. Denotamos por $K^{(i)}$ al conjunto de símplices de dimensión i de K.

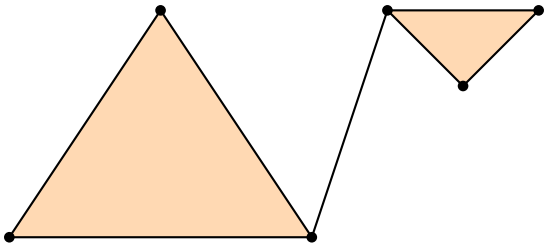


Figura 1.2: Complejo simplicial

Los complejos simpliciales abstractos permiten representar de forma combinatoria complejos simpliciales, eliminando la necesidad de considerar los símplices en el espacio euclídeo. Esto los hace muy interesantes en cálculos computacionales. Además, todo complejo simplicial geométrico admite una única representación como complejo simplicial abstracto, salvo reordenación de los vértices: se representa cada envolvente convexa como el subconjunto que contiene los vértices que la engendran.

Definición 1.4 (Aplicaciones simpliciales)**.** Sean K y L dos complejos simpliciales abstractos. Una aplicación simplicial entre K y L es una aplicación $f : K^{(0)} \to L^{(0)}$ que actúa sobre los vértices de K tal que para cada símplice $\sigma = \{u_0, u_1, \dots u_k\}$, $f(\sigma) = \{f(u_0), f(u_1), \dots f(u_k)\}$ forma un símplice abstracto en L. La imagen de una aplicación simplicial puede ser un símplice de dimensión menor que el de partida, si se da $f(u_i) = f(u_j)$ para algún $i \neq j$.

1.1.1. Construcción de complejos simpliciales

En el análisis topológico de datos, partimos generalmente de una nube de puntos en un espacio de medida a partir de la cual pretendemos generar un complejo simplicial abstracto.

Definición 1.5 (Nervios)**.** Sea F una colección de subconjuntos, definimos el nervio de F como:

$$\mathcal{N}(F) = \{X \subseteq F | \bigcap_{Y \in X} Y \neq \emptyset\}.$$

Si tratamos los elementos de F como vértices, es directo que su nervio es un complejo simplicial abstracto.

Definición 1.6 (Complejo de Čech)**.** El complejo de Čech es un complejo simplicial abstracto muy interesante teóricamente, cuya construcción a partir de la definición anterior es inmediata. Sea $r \in \mathbb{R}$ y $S \subseteq X$ un subconjunto finito de un espacio métrico X:

$$C_r(S) = \mathcal{N}\left(\left\{\overline{B(x, r/2)} | x \in S\right\}\right).$$

Teorema 1.1 (Teorema del nervio)**.** *Sean $U = \{U_1, \dots U_k\}$ una colección de subconjuntos cerrados y convexos de R^n. Se tiene que $U_1 \cup \dots \cup U_k$ es homotópicamente equivalente a $\mathcal{N}(U)$.*

Existen múltiples variaciones de este resultado, Teorema 2.2.1 de [5] ofrece una demostración consistente con la teoría ya introducida.

Corolario 1.2. *$C_r(S)$ es homotópicamente equivalente a $\bigcup_{s \in S} \overline{B(s, r/2)}$.*

La demostración es inmediata de Teorema 1.1 y del hecho de que las bolas cerradas son convexas.

Definición 1.7 (Complejo de Vietoris-Rips)**.** Sea X un espacio métrico y $S \subseteq X$ un subconjunto finito de X. El complejo de Vietoris-Rips para $r \in \mathbb{R}_{\geq 0}$ $R_r(S)$ se define de la siguiente manera:

1. $S = R_r(S)^{(0)}$ y
2. $\sigma \subseteq S \in R_r(S) \iff \text{máx}_{x,y \in \sigma} d(x, y) \leq r$.

Lema 1.3. $R_r(S) \subseteq C_{r\sqrt{2}}(S) \subseteq R_{r\sqrt{2}}(S)$.

Demostración. Se trata de un caso particular de Teorema 2.5 de [6]. □

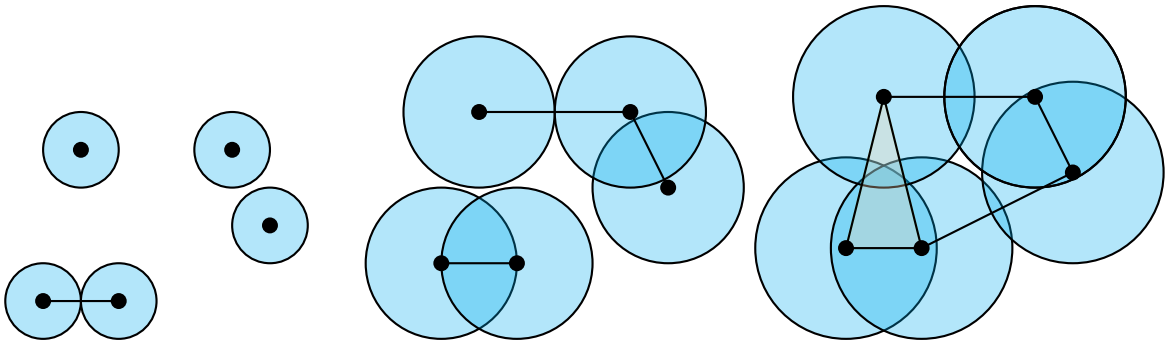


Figura 1.3: Complejos de Čech

1.2. Homología

Definición 1.8 (Grupo de cadenas)**.** El k-ésimo grupo de cadenas $C_k(K,P)$ de un complejo simplicial abstracto K en el dominio de ideales principales P es el grupo abeliano libre de k-símplices orientados. Es decir, un elemento de $C_k(K)$ es de la forma

$$\sum_i \lambda_i \sigma_i \text{ con } \lambda_i \in P \text{ y } \sigma_i \in K^{(k)} \text{ para cada } i.$$

Para la homología persistente, tomaremos siempre P un cuerpo finito de la forma $\mathbb{F} = \mathbb{Z}/n\mathbb{Z}$ con n primo. Las complicaciones de no tomar un cuerpo se exponen en [7].

Definición 1.9 (Aplicaciones de cadenas)**.** Las aplicaciones entre complejos simpliciales se extienden a aplicaciones entre k-grupos de cadenas de la siguiente manera: dado $\sigma = [u_0, \dots, u_k] \in K^{(k)}$,

$$f_k(\sigma) = \begin{cases} f(\sigma) & \text{si } f(u_i) \neq f(u_j)\ \forall i \neq j, \\ 0 & \text{en caso contrario.} \end{cases}$$

Finalmente, observemos que que f_k es una aplicación lineal:

$$f_k\left(\sum_i \lambda_i \sigma_i\right) = \sum_i \lambda_i f_k(\sigma_i).$$

Definición 1.10 (Aplicaciones frontera)**.** Sea $\sigma = [u_0, \dots, u_k]$, definimos su frontera $\partial_k(\sigma)$ como

$$\partial_k(\sigma) = \sum_{i=0}^{k} (-1)^i [u_0, \dots, u_{i-1}, u_{i+1}, \dots u_k].$$

Esto nos define una aplicación $\partial_k : C_k(K,\mathbb{F}) \to C_{k-1}(K,\mathbb{F})$.

Lema 1.4. $\partial_k \partial_{k+1} \equiv 0$.

Demostración. Sea K un complejo simplicial abstracto y $\sigma = [u_0, \dots, u_k] \in K$. Tenemos

$$\begin{aligned} \partial_k \partial_{k+1}(\sigma) =& \partial_k\Big(\sum_{i=0}^{k} (-1)^i [u_0, \dots, u_{i-1}, u_{i+1}, \dots u_k]\Big) = \\ & \sum_{j<i} (-1)^i (-1)^j [u_0, \dots, u_{j-1}, u_{j+1}, \dots\dots, u_{i-1}, u_{i+1}, \dots u_k] + \\ & \sum_{j>i} (-1)^i (-1)^{j-1} [u_0, \dots, u_{i-1}, u_{i+1}, \dots\dots, u_{j-1}, u_{j+1}, \dots u_k] = 0. \end{aligned} \tag{1.1}$$

□

Definición 1.11 (Complejos de cadenas)**.** La aplicación de Definición 1.10 nos permite definir el complejo de cadenas $C_*(K,\mathbb{F})$ como:

$$\ldots \xrightarrow{\partial_{k+2}} C_{k+1}(K,\mathbb{F}) \xrightarrow{\partial_{k+1}} C_k(K,\mathbb{F}) \xrightarrow{\partial_k} C_{k-1}(K,\mathbb{F}) \xrightarrow{\partial_{k-1}} \ldots$$

Definición 1.12 (Grupos de ciclos y grupos de frontera)**.** Definimos el grupo de ciclos $Z_k \simeq Z_k(K,\mathbb{F}) = \ker \partial_k$ y el grupo de frontera $B_k \simeq B_k(K,\mathbb{F}) = \mathrm{im}\partial_{k+1}$.

Nota 1.1. *Por Lema 1.4, tenemos que $B_k \subseteq Z_k \subseteq C_k$.*

Definición 1.13 (Grupo de homología)**.** Con todo lo anterior, podemos definir finalmente el k-grupo de homología de K:

$$H_k \simeq H_k(K,\mathbb{F}) = Z_k/B_k.$$

Como la homología es invariante homotópicamente, Corolario 1.2 nos asegura que para calcular la homología en un espacio métrico X, es suficiente con calcularla para su complejo de Čech.

Definición 1.14 (Número de Betti)**.** A la dimensión de H_k como espacio vectorial la denominamos número de Betti: $\beta_k = \dim H_k$.

Los números de Betti nos informan intuitivamente de cuántos agujeros de cada dimensión tiene el complejo sobre el que los hemos calculado. Así, β_0 es el número de componentes conexas, β_1 es el número de agujeros de dimensión 1, etc.

Teorema 1.5 ([8] IV.1.3)**.** *Sean A y A' dos grupos abelianos y $f : A \to A'$ un homomorfismo. Para $B \subseteq A$ y $B' \subseteq A'$ de forma que $f(B) \subseteq B'$, f induce un homomorfismo $(f)_* : A/B \to A'/B'$.*

Lema 1.6. *Sean K_1 y K_2 dos complejos simpliciales y $f : K_1 \to K_2$ un homomorfismo entre ellos. Entonces f induce un homomorfismo $(f)_* : H_k(K_1) \to H_k(K_2)$ para cada grupo de homología.*

Demostración. Directo de Teorema 1.5 y Nota 1.1. □

1.3. Homología persistente

Definición 1.15 (Complejo de persistencia)**.** Llamamos complejo de persistencia a una familia de complejos de cadenas junto con aplicaciones de cadenas entre ellos:

$$C_*^0 \xrightarrow{f^0} C_*^1 \xrightarrow{f^1} C_*^2 \xrightarrow{f^2} \cdots$$

Considerando todas las dimensiones y las aplicaciones fronteras, obtenemos la siguiente estructura:

$$\begin{array}{ccccccc}
\cdots & \xrightarrow{\partial_3} & C_2^0 & \xrightarrow{\partial_2} & C_1^0 & \xrightarrow{\partial_1} & C_0^0 \\
 & & \downarrow{\scriptstyle f^0} & & \downarrow{\scriptstyle f^0} & & \downarrow{\scriptstyle f^0} \\
\cdots & \xrightarrow{\partial_3} & C_2^1 & \xrightarrow{\partial_2} & C_1^1 & \xrightarrow{\partial_1} & C_0^1 \\
 & & \downarrow{\scriptstyle f^1} & & \downarrow{\scriptstyle f^1} & & \downarrow{\scriptstyle f^1} \\
\cdots & \xrightarrow{\partial_3} & C_2^2 & \xrightarrow{\partial_2} & C_1^2 & \xrightarrow{\partial_1} & C_0^2 \\
 & & \downarrow{\scriptstyle f^2} & & \downarrow{\scriptstyle f^2} & & \downarrow{\scriptstyle f^2} \\
 & & \vdots & & \vdots & & \vdots
\end{array}$$

Definición 1.16 (Módulo de persistencia)**.** Fijando una dimensión k, podemos calcular los grupos de homología en cada dimensión y obtenemos un módulo de persistencia. Denotamos por $(f^i)_*$ las aplicaciones inducidas en los grupos de homología y $H_k(K_i) = H^i$.

$$H^0 \xrightarrow{(f^0)_*} H^1 \xrightarrow{(f^1)_*} H^2 \xrightarrow{(f^2)_*} \dots$$

Definición 1.17 (Módulo de persistencia finito)**.** Un módulo de persistencia es finito si cada una de sus componentes es un módulo finitamente generado y las aplicaciones f^i son isomorfismos para todo $i > m$ para algún m.

En la práctica, la homología persistente se calcula sobre filtraciones.

Definición 1.18 (Filtración)**.** Sea $\{K_t\}_{t\in T}$ una familia finita subconjuntos del complejo simplicial abstracto K de forma que $K_t \subseteq K_{t+1} \forall t \in T$.

Tomando en Definición 1.15 como aplicaciones simpliciales las inclusiones, obtenemos una filtración.

$$K_1 \xrightarrow{i_{1,2}} K_2 \xrightarrow{i_{2,3}} K_3 \xrightarrow{i_{3,4}} \cdots \xrightarrow{i_{m-1,m}} K_m = K$$

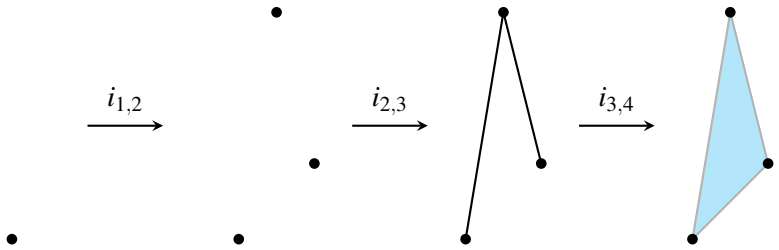


Figura 1.4: Ejemplo de filtración

Definición 1.19 (Homología persistente)**.** Para $i < j$, la k-homología persistente en (i, j) en el complejo de Definición 1.15, $H_k^{i,j}$, se define como la imagen del homomorfismo inducido $f_* : H_k^i \to H_k^j$. Si i y j no son consecutivas en el módulo de persistencia, la aplicación f_* anterior surge de la composición de las aplicaciones de H_k^i a H_k^j.

Al rango del subgrupo libre de $H_k^{i,j}$ lo denominamos número de Betti persistente $\beta_k^{i,j}$.

Tomemos un cuerpo $\mathbb{F}$ para dotar a un complejo de persistencia C de estructura de $\mathbb{F}[x]$-módulo en el que x actúa de la siguiente manera:

$$x \cdot \left(c^0, c^1, c^2, \dots\right) = \left(0, f_0(c^1), f_1(c^2), \dots\right).$$

De forma análoga, dotamos de estructura de $\mathbb{F}[x]$-módulo al módulo de persistencia M correspondiente al complejo C:

$$x \cdot \left(m^0, m^1, m^2, \dots\right) = \left(0, (f_0)_*(m^1), (f_1)_*(m^2), \dots\right).$$

La homología persistente nos informa de cómo evolucionan los grupos de homología en una serie de etapas. En el caso de filtraciones, estas etapas suelen corresponder a aumentos graduales del radio con el que se construyen los complejos simpliciales.

Teorema 1.7 (Teorema de correspondencia [7])**.** *Para un complejo de persistencia finito C_k con coeficientes en un cuerpo $\mathbb{F}$ y su módulo de persistencia asociado $H_k(C_k, \mathbb{F})$:*

$$H_k\left(C_k, \mathbb{F}\right) \cong \left(\bigoplus_i x^{t_i} \cdot \mathbb{F}[x]\right) \oplus \left(\bigoplus_j x^{r_j} \cdot \left(\mathbb{F}[x] / \left(x^{s_j} \cdot \mathbb{F}[x]\right)\right)\right).$$

Podemos interpretar el teorema de correspondencia de la siguiente manera:

- Existe una correspondencia biyectiva entre las partes libres de la ecuación anterior y los generadores de homología que surgen en t_i y permanecen para todos los valores posteriores.
- Los elementos de torsión corresponden a los generadores de homología que nacen en r_j y mueren en $r_j + s_j$.

1.3.1. Códigos de barras

Gracias a Teorema 1.7, podemos introducir ya uno de los elementos fundamentales de la homología persistente, los códigos de barras.

Para cada dimensión de homología, podemos representar el nacimiento y muerte de todos los k-agujeros del módulo de persistencia mediante $\mathscr{P}$-intervalos, esto es, pares ordenados (i, j) tal que $0 \leq i < j \in \mathbb{Z} \cup \{+\infty\}$.

Generalmente, los representamos como $[i, j)$, correspondiente a una componente que nace en i y muere en j. Si j es infinito, significa que la componente no muere nunca.

Una definición más exhaustiva de los códigos de barras puede encontrarse en [7].

1.3.2. Diagramas de persistencia

Los códigos de barras en seguida se vuelven difíciles de visualizar para nubes de puntos grandes o demasiado complejas. Por ello, una representación muy común de los mismos es la de diagrama de persistencia. Un diagrama de persistencia es una gráfica en dos dimensiones donde cada intervalo $[i, j)$ se representa en el plano $\mathbb{R}^2$ como el punto (i, j). Naturalmente, todos los puntos del diagrama se sitúan por encima de la diagonal. Además, para representar las componentes que no mueren nunca se suele fijar una altura superior a todas las demás como la altura infinito. Una interpretación tradicional de los diagramas de persistencia es que aquellos puntos más alejados de la diagonal corresponden a las características más relevantes, mientras que los más próximos a la diagonal son ruido.

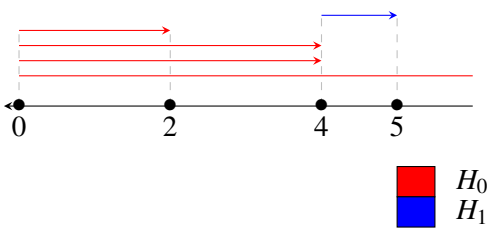


Figura 1.5: Ejemplo de código de barras

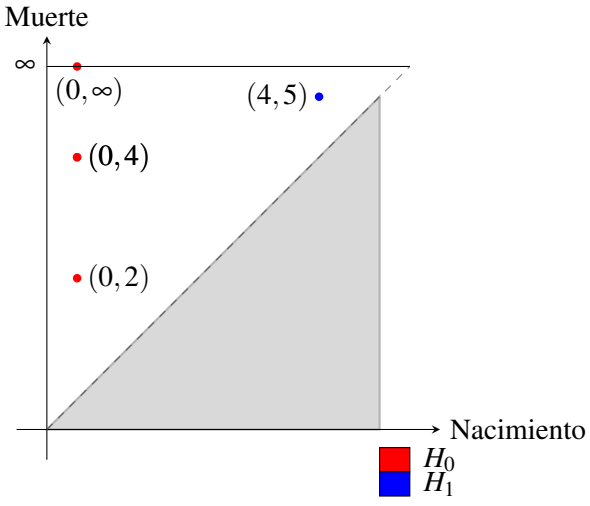


Figura 1.6: Ejemplo de diagrama de persistencia

Capítulo 2

Complejos expansivos

2.1. Homología cúbica

Definición 2.1. Para un $l \in \mathbb{Z}$, llamamos intervalo elemental a $[l, l+1]$ o $[l, l]$. Al segundo tipo de intervalos lo denominamos degenerado.

Para simplificar la notación, $[l, l] = [l]$.

Definición 2.2 (Cubo elemental)**.** Un cubo elemental en $m \in \mathbb{N}$ dimensiones es el producto de m intervalos elementales, $Q = I_1 \times I_2 \times \cdots \times I_m$. Conviene observar que cualquiera de los I_i puede ser degenerado, de forma que cubos de dimensión menor que m pueden vivir en un espacio de dimensión m.

En este capítulo, nos referiremos también a los cubos elementales como símplices (cúbicos).

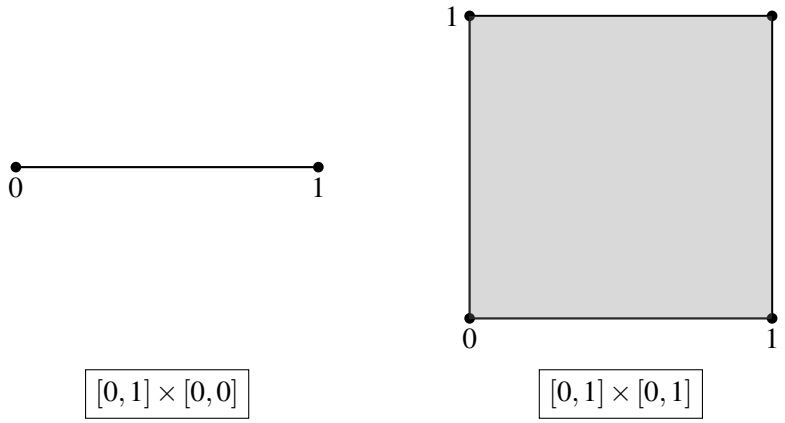


Figura 2.1: Ejemplos de cubos elementales en $\mathbb{Z}^2$

Definición 2.3. Dado un cubo Q, la dimensión del mismo es su número de intervalos no degenerados.

Definición 2.4 (Cara de un cubo)**.** Sea $Q = I_1 \times I_2 \times \cdots \times I_m$, un cubo elemental R es una cara de Q si $R \subset Q$.

Definición 2.5 (Complejo cúbico)**.** Un complejo cúbico K es un conjunto de cubos elementales que satisface los mismos requisitos que en Definición 1.2:

1. Si $Q \in K$ y R es una cara de Q, entonces $R \in K$.
2. Si $Q, R \in K$, entonces $Q \cap R$ es o bien vacía o una cara común de ambos.

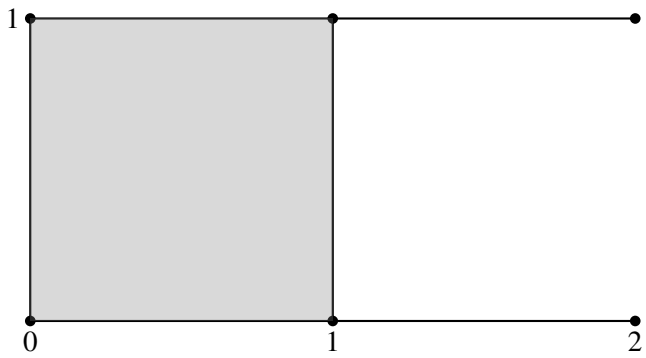


Figura 2.2: Complejo cúbico formado por $[0,1]\times[0,1]$, $[1,2]\times[0]$, $[1,2]\times[1]$ y sus caras.

El segundo requisito es de hecho inmediato del primero y de la intersección de conjuntos de la forma $[l,l+1]$ o $[l,l]$ para $l\in\mathbb{Z}$.

Definición 2.6 (Cadenas cúbicas)**.** Si consideramos los cubos elementales algebraicamente en lugar de geométricamente, podemos obtener una estructura de grupo de cadenas análoga a la de Definición 1.8. Mantenemos la notación $C_k(K,\mathbb{F})$ para los k-ésimos grupos de cadenas cúbicos en un cuerpo $\mathbb{F}$.

Definición 2.7 (Aplicaciones cúbicas sobre cadenas)**.** Considerando de nuevo los cubos como objetos algebraicos, podemos definir aplicaciones entre complejos de cadenas como la extensión de aplicaciones que llevan cubos de un complejo cúbico a otro.

Hasta el momento hemos seguido la teoría de homología cúbica de [9], pero decidimos redefinir las aplicaciones frontera para simplificarlas.

Definición 2.8. (Fronteras en complejos cúbicos) Dado un cubo $Q=[a_1,b_1]\times\cdots\times[a_m,b_m]$ de dimensión k, definimos su k-operador frontera como:

$$\partial_k(Q)=\sum_{i\in I}(-1)^{\rho(i)}\left([a_1,b_1]\times\cdots\times[a_i,a_i]\times\cdots\times[a_m,b_m]-[a_1,b_1]\times\cdots\times[b_i,b_i]\times\cdots\times[a_m,b_m]\right),$$

donde I son los índices de los elementos no degenerados y $\rho(i)$ es su posición relativa en I.

Lema 2.1. $\partial_{k-1}\partial_k\equiv 0$.

Demostración. Por simplicidad, asumimos sin pérdida de generalidad que no hay elementos degenerados intermedios, es decir, que de haberlos estos se encontrarían en los intervalos del final. De esta forma no es necesario considerar la posición relativa de los índices. En los sumatorios, las i y j se toman sobre los intervalos no degenerados. Sea $Q=[a_1,b_1]\times\cdots\times[a_m,b_m]$ un cubo, definimos, para simplificar la notación, $Q_{c_i}=[a_1,b_1]\times\cdots\times[c_i,c_i]\times\cdots\times[a_m,b_m]$:

$$\begin{aligned}\partial_{k-1}\circ\partial_k(Q)=&\partial_{k-1}\left(\sum_i(-1)^i(Q_{a_i}-Q_{b_i})\right)=\\ =&\sum_{i<j}(-1)^{i+j+1}Q_{a_i,a_j}+\sum_{i>j}(-1)^{i+j}Q_{a_i,a_j}+\sum_{i<j}(-1)^{i+j+1}Q_{b_i,b_j}+\sum_{i>j}(-1)^{i+j}Q_{b_i,b_j}+\\ &+\sum_{i<j}(-1)^{i+j}Q_{a_i,b_j}+\sum_{i>j}(-1)^{i+j+1}Q_{a_i,b_j}+\sum_{i<j}(-1)^{i+j}Q_{b_i,a_j}+\sum_{i>j}(-1)^{i+j+1}Q_{b_i,a_j}=0.\end{aligned}\tag{2.1}$$

En el anexo puede consultarse una demostración del teorema con otra notación. □

Corolario 2.2. *Dada una familia de complejos cúbicos relacionados por aplicaciones de la forma* $\{K_0 \xleftarrow{f_1} K_1 \xleftarrow{f_2} K_2 \xleftarrow{f_3} \dots \xleftarrow{f_{m-1}} K_{m-1}\}$*, se puede construir de forma única un complejo de persistencia como el de Definición 1.15 y un módulo de persistencia como el de Definición 1.16.*

Demostración. La construcción del complejo es inmediata y la del módulo resulta posible porque la composición de aplicaciones frontera es nula como en Lema 1.6. □

2.2. Complejo binario expansivo

Existen diversas formas de representar un complejo cúbico computacionalmente. En el presente trabajo, hemos optado por una representación unipuntual en el espacio de $[0,2^n)^m$, con $n,m \in \mathbb{N}$. Este método ofrece varias ventajas:

- Cada símplice se representa únicamente por un punto en el espacio, eliminando cualquier forma de representación combinatoria (con el consiguiente ahorro de memoria).
- La representación unipuntual surgirá directamente del método de construcción de secuencias complejos cúbicos en distintas profundidades que hemos desarrollado.
- La representación mediante *bits* permite emplear instrucciones lógicas del procesador, mucho más rápidas que las instrucciones aritméticas que necesitaríamos habitualmente para el cálculo de un complejo simplicial.

La forma en la que, en cada profundidad, cada punto será interpretado como un símplice en función de su posición en el espacio se expondrá a lo largo de esta sección. La idea general es que construiremos mallas cada vez más finas en las que tanto la dimensión como la posición del símplice derivarán de su posición como punto en la cuadrícula.

Para iniciar la construcción del complejo, se parte de una nube de puntos $P \subset \mathbb{R}^m$ finita. En primer lugar, transformamos la nube de puntos para representarlos en el espacio $Z^m_{2^n}$, a la que llamamos P_n. Puede optarse por cualquier algoritmo de escalado a $[0,2^n)$.

Una vez normalizados, asumiendo una representación entera sin signo, cada punto corresponderá directamente a una secuencia de n bits en cada componente m.

Definición 2.9. Por simplicidad, definimos las constantes $\text{salto}_i = 2^{n-i-2} = \left(00\dots\overset{i}{1}0\dots0\right)_2$. Por ejemplo, $\text{salto}_0 = (010\dots0)$ y $\text{salto}_1 = (001\dots0)$.

También definimos salto_i^k como el salto de profundidad i en la posición k de las m dimensiones, de forma que $\text{salto}_i^1 = (\text{salto}_i, 0, \dots, 0)$, $\text{salto}_i^2 = (0, \text{salto}_i, 0, \dots, 0)$, etc.

Definición 2.10. Definimos recursivamente los conjuntos S_i, donde la i corresponde a lo que llamaremos **profundidad**:

$$S_i = \begin{cases} \{2^{n-1} = (10\dots0)_2\} & \text{si } i = 0, \\ \{s \pm \text{salto}_{i-1} | s \in S_{i-1}\} & \text{si } i > 0. \end{cases}$$

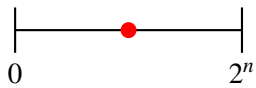


Figura 2.3: Conjunto S_0

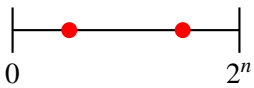


Figura 2.4: Conjunto S_1

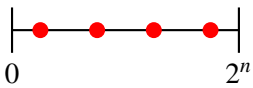


Figura 2.5: Conjunto S_2

Los conjuntos S_i proporcionarán la estructura de complejo cúbico a los complejos de nuestra construcción. En cada profundidad i, los intervalos elementales serán de la forma $[l,l]$ o $[l,l+2\cdot\text{salto}_{i-1}]$ para algunos $l \in S_i$. Seguirán tratándose de complejos cúbicos, ya que simplemente corresponden a desplazamientos y escalados de los intervalos elementales $[l,l]$ y $[l,l+1]$ de Definición 2.1.

Definición 2.11. Ahora podemos construir

$$\text{suelo}_i(x) := \text{máx}\{s \in S_i | s \leq x\} \text{ y } \text{techo}_i(x) := \text{mín}\{s \in S_i | s \geq x\},$$

que definen funciones $\mathbb{Z}_{2^n} \to \mathbb{Z}_{2^n}$.

Definición 2.12 (Intervalos no degenerados)**.** Dado un $Q = (q_1, \dots q_m) \in \mathbb{Z}_{2^n}^m$, definimos $\text{NoDeg}_i(Q) = \{i \in \{1, \dots, m\} \, | q_i \notin S_i\}$. Análogamente, $\text{Deg}_i(Q) = \{i \in \{1, \dots, m\} \, | q_i \in S_i\}$.

Definición 2.13 (Representación puntual de símplices)**.** Finalmente, dado un punto $Q = (q_1, \dots, q_m) \in \mathbb{Z}_{2^n}^m$ en el que $\text{mín}\, S_i \leq q_k \leq \text{máx}\, S_i$ para todo k, definimos su símplice asociado

$$[\text{suelo}_i(q_1), \text{techo}_i(q_1)] \times \cdots \times [\text{suelo}_i(q_m), \text{techo}_i(q_m)] \, .$$

Expresado en palabras, estamos asignando los puntos intermedios a intervalos no degenerados y conservando los puntos en cada componente degenerada. La dimensión de Q en profundidad i es $\dim_i(Q) = |\text{NoDeg}_i(Q)|$.

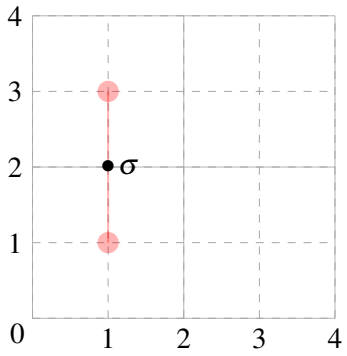


Figura 2.6: Ejemplo de símplice σ de dimensión 1 en profundidad 1. $\sigma = (1,2)$ y $[\text{suelo}_1(1), \text{techo}_1(1)] \times [\text{suelo}_1(2), \text{techo}_1(2)] = [1,1] \times [1,3] = [1] \times [1,3]$.

Definición 2.14 (Funciones salto)**.** Resultará muy útil definir la siguiente función salto:

$$\begin{aligned} \mathfrak{s} : \mathscr{P}(\{1, \cdots, m\}) \times \mathbb{Z}_{2^m}^m \times \mathscr{P}(\mathbb{Z}_{2^n}) &\to \mathscr{P}(\mathbb{Z}_{2^n}^m) \\ (I, Q, P) &\mapsto \{Q + (x_1, \dots, x_m) | x_i \in P \text{ si } i \in I \text{ y } x_i = 0 \text{ si } i \notin I\} \, . \end{aligned}$$

La definición de las funciones salto puede resultar demasiado computacional. Su finalidad es generar todos los posibles desplazamientos de un punto en el espacio en unas dimensiones concretas y para varios desplazamientos. La utilidad de estas aplicaciones se comprenderá al llegar a la generación del complejo.

Definición 2.15 (Función expansión)**.** Una noción fundamental en la creación de un complejo binario expansivo es la de expansión:

$$\mathfrak{e}_i(Q) = \mathfrak{s}(\text{Deg}_i(Q), Q, \{\text{salto}_i, -\text{salto}_i, 0\}),$$

es decir, un símplice puede expandirse a otro desplazándose sumando o restando saltos en las coordenadas degeneradas.

Podemos definir así una familia de funciones

$$\{\mathfrak{e}_i : \mathbb{Z}_{2^n}^m \to \mathscr{P}(\mathbb{Z}_{2^n}^m)\}_{i \in \{0,1,\cdots,n-1\}} \, .$$

Las funciones expansión son la herramienta que nos permitirá generar complejos cúbicos de mayor dimensión partiendo de otros menores. Se la aplicaremos a todos los símplices del complejo y luego decidiremos cuáles de ellos cumplen las condiciones para formar parte del nuevo.

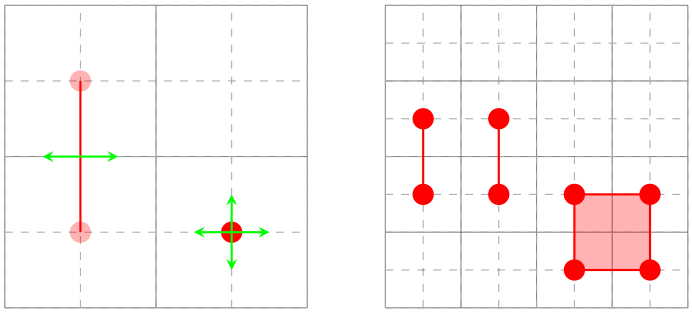


Figura 2.7: Ejemplo de expansiones para dos símplices (sin considerar la expansión de sus caras)

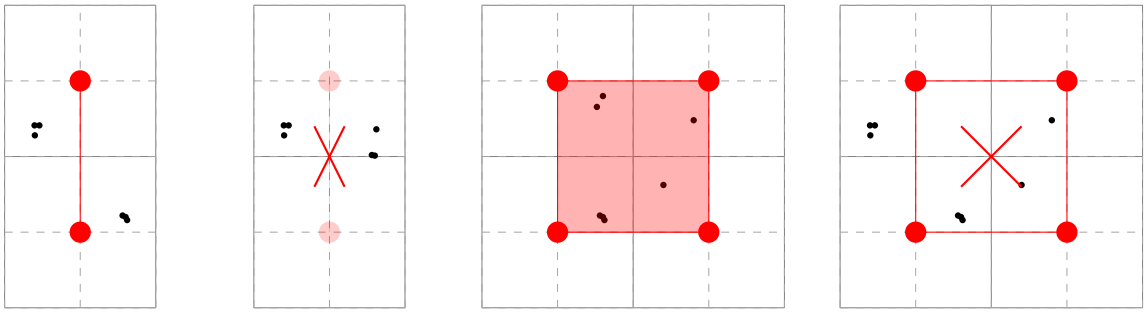


Figura 2.8: Ejemplos de expansiones que se construyen o no dependiendo de la posición de los puntos de P_n

Definición 2.16. Dado un punto cualquiera y un $i \in \{0,1,\cdots,n-2\}$, definimos

$$\begin{aligned}\text{ExisteSímplice}_i(Q) = 1 \iff & \text{ para todo } Q_1 \in \mathfrak{s}(\text{NoDeg}_i(Q), Q, \{\pm salto_i\}),\\ & \text{existe } Q_2 \in \mathfrak{s}(\text{Deg}_i(Q), Q_1, \{\pm salto_i\}) \text{ tal que } d_\infty(Q_2, P_n) \leq salto_i.\end{aligned}$$

Si $\text{NoDeg}_i(Q) = \emptyset$,

$$\text{ExisteSímplice}_i(Q) = 1 \iff \text{ existe } Q_2 \in \mathfrak{s}(\text{Deg}_i(Q), Q, \{\pm salto_i\}) \text{ tal que } d_\infty(Q_2, P_n) \leq salto_i.$$

Si $\text{Deg}_i(Q) = \emptyset$,

$$\text{ExisteSímplice}_i(Q) = 1 \iff \text{ para todo } Q_1 \in \mathfrak{s}(\text{NoDeg}_i(Q), Q, \{\pm salto_i\}), d_\infty(Q_1, P_n) \leq salto_i.$$

Es decir, ExisteSímplice_i es una función booleana que determinará si una expansión debe convertirse en símplice o descartarse.

Podemos definir así una familia de aplicaciones

$$\{\text{ExisteSímplice}_i : \mathbb{Z}_{2^n}^m \to \mathscr{F}_2\}_{i \in \{0,1,\cdots,n-2\}}.$$

En cada dimensión, ExisteSímplice formaliza la idea intuitiva de que, para que se pueda formar un símplice, debe haber puntos de P_n en lugares adecuados cercanos a la representación puntual del símplice: para la formación de un 0-símplice bastará con la presencia de un punto alrededor del símplice; para la de un 1-símplice, con la presencia de un punto a ambos lados de la cuadricula que lo rodea; para la de un 2-símplice, de uno en cada uno de los cuartos; y así en todas las dimensiones posibles.

Construcción del complejo binario expansivo

Una vez introducidas todas las herramientas necesarias, pasamos a exponer el algoritmo que se empleará para la construcción de un complejo a partir de una nube de puntos.

Algoritmo 1 Construcción del complejo

Require: $|P_n| > 0$
  /* E_i son los símplices en representación puntual para profundidad i */
  /* F_i son las representaciones de las aplicaciones colapso f_i en forma de diccionario */
  $E_0 \leftarrow [(1000\ldots^{n)}\,0)_2]^m$
  $F_0 \leftarrow$ [ ]
  depth $\leftarrow 1$
  while depth $< n-1$ **do**
    $E_{\text{depth}} \leftarrow$ [ ]
    $F_{\text{depth}} \leftarrow$ [ ]
    for complex in $E_{\text{depth-1}}$ **do**
      for exp in $\mathfrak{e}_{\text{depth}-1}$(complex) **do**
        if ExisteSímplice$_{\text{depth}}$(exp) **then**
          $E_{\text{depth}} \leftarrow$ exp
          if $\dim_{\text{depth}}(\text{exp}) = \dim_{\text{depth}-1}(\text{complex})$ **then**
            $F_{\text{depth}}[\text{exp}] = \text{complex}$
          end if
        end if
      end for
    end for
    depth $\leftarrow$ depth $+ 1$
  end while

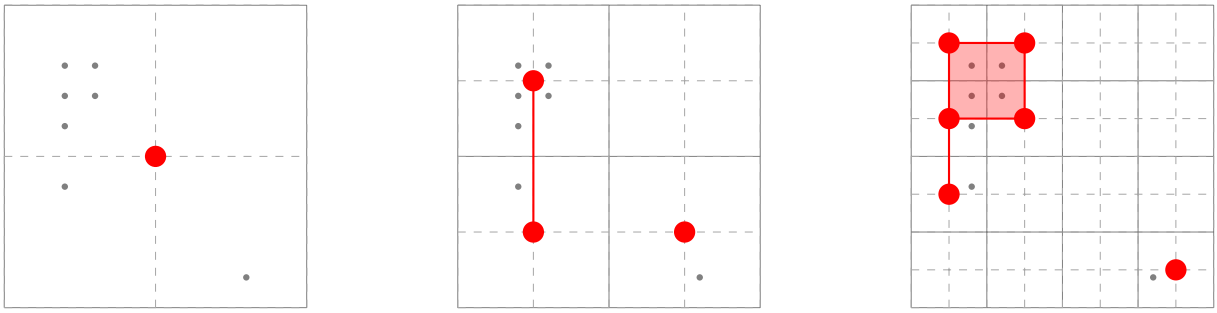


Figura 2.9: Ejemplo de complejos construidos en distintas profundidades sobre la misma nube de puntos

Teorema 2.3. *Se puede dotar a* $\{E_i, F_i\}_i$ *de una estructura de i complejos cúbicos relacionados mediante aplicaciones:*

$$\{K_0 \xleftarrow[f_1]{} K_1 \xleftarrow[f_2]{} K_2 \xleftarrow[f_3]{} \ldots \xleftarrow[f_{n-2}]{} K_{n-2} | K_i \text{ complejo cúbico en } \mathbb{Z}_{2^n}^m \text{ construido a partir de } E_i\}.$$

Demostración. Las aplicaciones f entre complejos surgen naturalmente de las aplicaciones de expansión generadas durante el algoritmo, consideradas únicamente entre símplices de igual dimensión, ya que para el cálculo de la homología únicamente nos interesan las aplicaciones entre grupos de cadenas.

Finalmente, es necesario probar que, para cada símplice en K_i, sus caras están contenidas en K_i. Para ello, es suficiente ver que en Definición 2.16 los intervalos no degenerados de un símplice de menor dimensión R contenido en Q están contenidos en los de Q, y los intervalos degenerados de Q están contenidos en los de R. Por tanto, la expresión booleana de R será verdadera, pues lo es la de Q.

Es importante observar que este método garantiza que **un símplice nunca puede proceder de la expansión de varios símplices al mismo tiempo**, por lo que las aplicaciones f están bien definidas. □

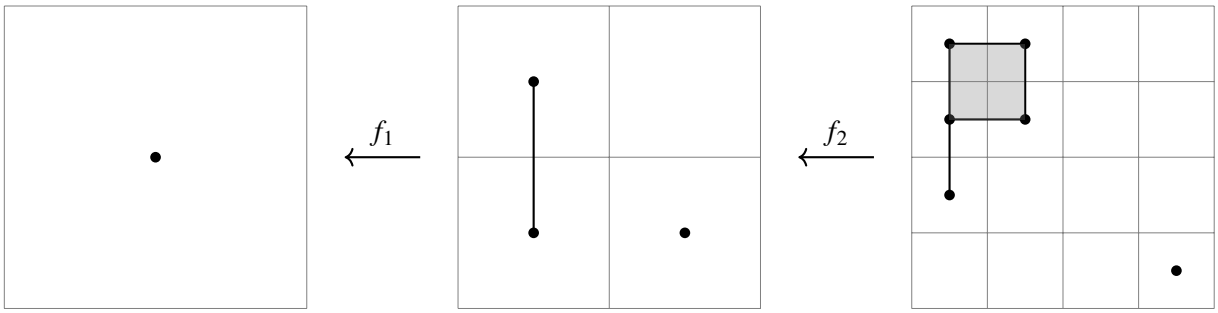


Figura 2.10: Ejemplo de aplicaciones de colapso entre complejos cúbicos

Lema 2.4. *Si $Q = (q_1, \cdots, q_k, \cdots, q_n)$ es un símplice en profundidad i y q_k corresponde a un intervalo no degenerado, $(suelo_i(q_k), techo_i(q_k))$ es equivalente a $(q_k - salto_{i-1}, q_k + salto_{i-1})$.*

El lema anterior, que nos permite obtener las caras de un símplice mediante sumas y restas de constantes, se empleará extensamente el resto del capítulo.

Corolario 2.5. *Con la estructura de complejos cúbicos del teorema anterior, para $Q \in C_i^j$, podemos definir directamente:*

$$\partial_i(Q) = \sum_{k \in NoDeg_i(Q)} (-1)^{\rho(k)} [(Q - salto_{j-1}^k) - (Q + salto_{j-1}^k)],$$

donde $\rho(k)$ es la posición relativa de k en $NoDeg_i(Q)$.

Lema 2.6. *El diagrama*

$$\begin{array}{ccc} C_i^j & \xrightarrow{\partial_i^j} & C_{i-1}^j \\ \downarrow{\scriptstyle f_i^j} & & \downarrow{\scriptstyle f_{i-1}^j} \\ C_i^{j-1} & \xrightarrow{\partial_i^{j-1}} & C_{i-1}^{j-1} \end{array}$$

es conmutativo, donde los subíndices corresponden a la dimensión de los símplices y los superíndices al nivel de expansión o profundidad.

Demostración. Lo probaremos para símplices. Por linealidad de los operadores frontera y las f, la extensión de la conmutatividad al complejo de cadenas es inmediata.

Sea Q un símplice de dimensión i en profundidad j, existen tres posibilidades:

- $f^j(Q) = R$ de dimensión $i-1$.

  En primer lugar, $f_i^j(Q) = 0$. Tenemos además que

  $$f^j(Q) = R \Rightarrow Q = R + (x_1, \cdots, x_n) \text{ con } x_l = \begin{cases} 0 & \text{si } l \in \mathrm{NoDeg}_{j-1}(R) \cup \{k\} \text{ y} \\ \pm \mathrm{salto}_{j-1} & \text{si } l \in \mathrm{Deg}_{j-1}(R) \setminus \{k\}, \end{cases}$$

  para $k \in \mathrm{Deg}_{j-1}(R)$. Por Lema 2.4, si T es una cara Q:

  $$T = Q \pm salto_{j-1}^l = R + (x_1, \cdots, x_n) \pm \mathrm{salto}_{j-1}^l \text{ con } l \in \mathrm{NoDeg}_j(Q) = \mathrm{NoDeg}_{j-1}(R) \cup \{k\}.$$

  - Si $l \in \mathrm{NoDeg}_{j-1}(R)$, T provendría de una cara de R de dimensión $i-2$, ya que $R \pm \mathrm{salto}_{j-1}^l = (R \pm \mathrm{salto}_{j-2}^l) \mp \mathrm{salto}_{j-1}^l$. Así que $f_{i-1}^j(T) = 0$.
  - Si $l = k$, tendríamos que $f_{i-1}^j(T) = R$ y $f_{i-1}^j(Q - \mathrm{salto}_{j-1}^l) - f_{i-1}^j(Q + \mathrm{salto}_{j-1}^l) = R - R = 0$.

En definitiva,

$$\partial_i^{j-1} \circ f_i^j(Q) = f_{i-1}^j \circ \partial_i^j(Q) = 0.$$

- $f^j(Q) = R$ de dimensión menor que $i-1$.

  Por su similitud con el caso anterior, se incluye en el anexo.

- $f^j(Q) = R$ de dimensión i, luego $f_i^j(Q) = R \neq 0$.

  De momento, supondremos que Q es un símplice de dimensión máxima n, de forma que $R = Q$ considerados únicamente como puntos en $\mathbb{Z}_{2^n}^m$.

  Tenemos:

$$\partial_i^{j-1}(f_i^j(Q)) = \partial_i^{j-1}(Q) = \sum_k (-1)^k [(Q - \text{salto}_{j-2}^k) - (Q + \text{salto}_{j-2}^k)] \text{ y}$$

$$\partial_i^j(Q) = \sum_k (-1)^k [(Q - \text{salto}_{j-1}^k) - (Q + \text{salto}_{j-1}^k)].$$

  Por linealidad de las f,

$$f_{i-1}^j(\partial_i^j(Q)) = \sum_k (-1)^k [f_{i-1}^j(Q - \text{salto}_{j-1}^k) - f_{i-1}^j(Q + \text{salto}_{j-1}^k)].$$

  Veamos que $f_{i-1}^j(Q + \text{salto}_{j-1}^k) = Q + \text{salto}_{j-2}^k$:

$$f_{i-1}^j(Q + \text{salto}_{j-1}^k) = Q + \text{salto}_{j-2}^k \iff Q + \text{salto}_{j-1}^k \in \text{expandir}_{j-1}(Q + \text{salto}_{j-2}^k).$$

  Como k es un intervalo degenerado en $j-1$ de $(Q + \text{salto}_{j-2}^k)$ y $\text{salto}_{j-2}^k = 2 \cdot \text{salto}_{j-1}^k$,

$$(Q + \text{salto}_{j-2}^k) - \text{salto}_{j-1}^k = Q + \text{salto}_{j-1}^k \in \text{expandir}_{j-1}(Q + \text{salto}_{j-2}^k).$$

  Análogamente, $f_{i-1}^j(Q - \text{salto}_{j-1}^k) = Q - \text{salto}_{j-2}^k$.

  Queda probado $\partial_i^{j-1} \circ f_i^j(Q) = f_{i-1}^j \circ \partial_i^j(Q)$.

  El resultado se mantiene aunque Q no sea de dimensión máxima, ya que basta considerarlo dentro de una dimensión más pequeña.

□

Nota 2.1. *En definitiva, nuestro complejo de persistencia corresponde al siguiente diagrama conmutativo, análogo al de Definición 1.15, aunque aparentemente cambie la dirección de las f.*

$$\begin{array}{ccccccc}
\cdots & \xrightarrow{\partial_3} & C_2^0 & \xrightarrow{\partial_2} & C_1^0 & \xrightarrow{\partial_1} & C_0^0 \\
 & & \uparrow{\scriptstyle f_2^1} & & \uparrow{\scriptstyle f_1^1} & & \uparrow{\scriptstyle f_0^1} \\
\cdots & \xrightarrow{\partial_3} & C_2^1 & \xrightarrow{\partial_2} & C_1^1 & \xrightarrow{\partial_1} & C_0^1 \\
 & & \uparrow{\scriptstyle f_2^2} & & \uparrow{\scriptstyle f_1^2} & & \uparrow{\scriptstyle f_0^2} \\
\cdots & \xrightarrow{\partial_3} & C_2^2 & \xrightarrow{\partial_2} & C_1^2 & \xrightarrow{\partial_1} & C_0^2 \\
 & & \uparrow{\scriptstyle f_2^3} & & \uparrow{\scriptstyle f_1^3} & & \uparrow{\scriptstyle f_0^3} \\
 & & \vdots & & \vdots & & \vdots
\end{array}$$

A partir de aquí, podemos obtener un módulo de persistencia exactamente igual que expusimos en el primer capítulo.

Capítulo 3

Métodos adaptativos para el cálculo de los grupos de homología y los códigos de barras

A partir del esquema

$$\begin{array}{ccccc} C_{i+1}^{j} & \xrightarrow{\partial_{i+1}} & C_{i}^{j} & \xrightarrow{\partial_{i}} & C_{i-1}^{j} \\ \downarrow{\scriptstyle f_{i+1}^{j}} & & \downarrow{\scriptstyle f_{i}^{j}} & & \downarrow{\scriptstyle f_{i-1}^{j}} \\ C_{i+1}^{j-1} & \xrightarrow{\partial_{i+1}} & C_{i}^{j-1} & \xrightarrow{\partial_{i}} & C_{i-1}^{j-1} \end{array}$$

queremos obtener $H_i^j \xrightarrow{(f_i^j)_*} H_i^{j-1}$.

3.1. Cálculo de los grupos de homología

Sean $A = \big(\partial_{i+1}(C_{i+1})\big)$ y $B = \big(\partial_i(C_i)\big)$, es decir, la representación matricial habitual del operador frontera, donde los índices de columna corresponden a los símplices de partida y los de fila a los símplices del espacio de llegada.

La obtención del núcleo de ∂_i podría lograrse mediante un método de reducción por columnas, pero, dado que también resultará necesario expresar la imagen de ∂_{i+1} en una base del núcleo anterior, será necesario efectuar dos reducciones distintas.

Por Lema 2.1, partimos de:

$$BA = 0.$$

En primer lugar, calculamos una forma escalonada por columnas de B.

$$\left(\frac{B}{I}\right) \longrightarrow \left(\frac{B \cdot Q}{Q}\right).$$

De esta manera:

$$BA = BQQ^{-1}A = 0.$$

Si nos centramos únicamente en los cambios efectuados sobre B, las columnas nulas de BQ corresponderán a vectores de una base del núcleo. De hecho, los índices de columnas nulas de BQ corresponden

a una base del núcleo tomando esas columnas en Q:

$$\left(\frac{B\cdot Q}{Q}\right) = \left(\begin{array}{ccccc} * & * & 0 & \cdots & 0 \\ & & & & \\ \hline * & * & v_1 & \cdots & v_l \end{array}\right) \text{ y } \ker\partial_i = \langle v_1,\cdots,v_l\rangle.$$

Es necesario considerar que estos vectores pueden variar en la reducción posterior.

Queremos saber cuáles de estos vectores corresponden a una base de $\mathrm{im}\partial_{i+1}$. Para ello, calculamos una forma escalonada por filas de $Q^{-1}A$, de tal forma que las filas pivote corresponderán a los vectores que forman una base de la imagen (y que están necesariamente en el núcleo de ∂_i).

$$\left(\begin{array}{c|c} Q^{-1}A & I \end{array}\right) \longrightarrow \left(\begin{array}{c|c} PQ^{-1}A & P \end{array}\right).$$

Finalmente,

$$BA = BQQ^{-1}A = (BQP^{-1})(PQ^{-1}A) = 0.$$

Efectuadas todas las reducciones, definimos los siguientes conjuntos, matrices y vectores:

$$R = QP^{-1} \Rightarrow BA = BRR^{-1}A = 0,$$

$$I = \left\{ j \text{ tal que } (BR)_{ij} = 0 \text{ para todo } i \right\} \text{ y}$$

$$J = \left\{ i \text{ tal que } \left(R^{-1}A\right)_{i*} \text{ es una fila pivote } \right\}.$$

Por las reducciones efectuadas, tenemos además que $J \subset I$ y podemos definir:

$$U = \{u_1,\dots u_k\} = \{R_{*i} \text{ tal que } i \in J\} \text{ y}$$

$$W = \{w_1,\dots w_l\} = \{R_{*i} \text{ tal que } i \in I \setminus J\}.$$

De esta forma, obtenemos $\mathrm{im}\partial_{i+1} = \langle u_1,\dots,u_k\rangle$. Los vectores de W completan una base del núcleo: $\langle u_1,\dots,u_k,w_1,\dots,w_l\rangle = \ker\partial_i$.

Finalmente, resulta inmediato cocientar el núcleo por la imagen y hemos obtenido los representantes del grupo de homología, estos son, $\{[w_1],\dots,[w_l]\}$.

3.2. Cálculo de las aplicaciones inducidas sobre los grupos de homología

En la sección anterior, para cada i y j hemos calculado las siguientes matrices de cambio de base para el núcleo de cada complejo de cadenas: R_i^j y R_i^{j-1}. Llamamos β_i^j a la base del H_i^j resultado de los cambios efectuados al realizar las reducciones en C_i^j (análogamente β_i^{j-1}) y M a la representación matricial de f_i^j. Por tanto, tenemos la siguiente estructura, donde las R son las matrices de cambio de base.

$$\begin{array}{ccc} C_i^j & \xrightarrow{\quad M \quad} & C_i^{j-1} \\ {\scriptstyle R_i^j}\uparrow & & {\scriptstyle R_i^{j-1}}\uparrow \\ \left[C_i^j\right]_{\beta_i^j} & & \left[C_i^{j-1}\right]_{\beta_i^{j-1}} \end{array}$$

Para obtener la aplicación inducida, calculamos $M' = (R_i^{j-1})^{-1}MR_i^j$ y elegimos la submatriz correspondiente a los pivotes de los elementos de las bases W_j y W_{j-1}, es decir, aquellos que extienden la base de la imagen a una base del núcleo.

Tomando los subconjuntos I_j, J_j, I_{j-1} y J_{j-1} definidos en la sección anterior,

$$(M)_* = \left\{ (M')_{lm} \text{ tal que } l \in I_{j-1} \setminus J_{j-1} \text{ y } m \in I_j \setminus J_j \right\}.$$

3.2.1. Ejemplo

Calcularemos el número de 1-agujeros que persisten y mueren en el siguiente paso de colapso. Los símplices en cada nivel corresponden al algoritmo de expansión que hemos explicado, pero limitando la dimensión de los símplices a 1.

Por la simplicidad del ejemplo escogido, para la operaciones y reducciones matriciales podríamos coger cualquier cuerpo primo: supongamos que todas se realizan en $\mathbb{Z}/3\mathbb{Z}$.

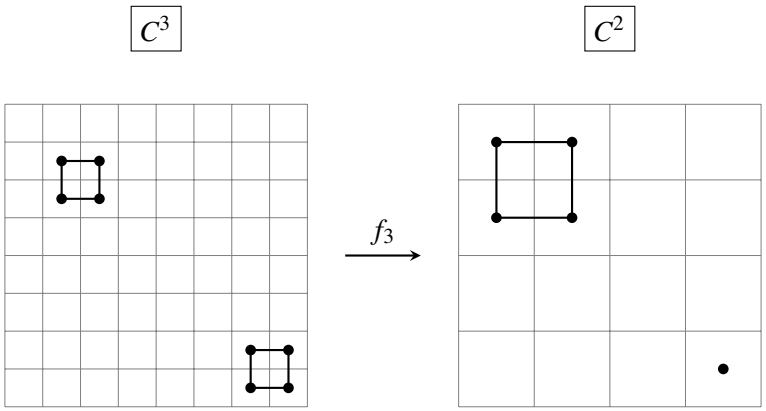


Figura 3.1: Ejemplo de colapso en profundidad 3

Lo vemos en forma de diagrama a continuación.

$$\begin{array}{ccccc} C_2^3 = 0 & \xrightarrow{\partial_2} & C_1^3 & \xrightarrow{\partial_1} & C_0^3 \\ & & \downarrow{\scriptstyle f_3} & & \\ C_2^2 = 0 & \xrightarrow{\partial_2} & C_1^2 & \xrightarrow{\partial_1} & C_0^2 \end{array}$$

Con

$$M := f_3 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Homología en C^2

En este caso, al no haber 2-símplices, $\partial_2 = 0$, es decir, estamos cocientando por cero y sólo debemos calcular la forma escalonada por columnas de ∂_1.

$$\left(\frac{\partial_1}{I}\right) = \left(\begin{array}{cccc} -1 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 1 \\ \hline 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right) \longrightarrow \left(\begin{array}{cccc} -1 & 0 & 0 & \overset{\downarrow}{0} \\ 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & 0 \\ \hline 1 & -1 & 1 & -1 \\ 0 & 1 & -1 & 1 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{array}\right) = \left(\frac{\partial_1 Q}{Q}\right).$$

Por lo ya explicado en secciones anteriores, la cuarta columna de $\partial_1 Q$ se ha anulado, indicando que corresponde a un elemento de núcleo. Como estamos cocientando el núcleo por 0, sabemos que esta columna corresponderá a un elemento del grupo de homología.

Finalmente, llamamos $R_2 = Q$ y

$$R_2^{-1} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Homología en C^3

De nuevo, al no haber 2-símplices, $\partial_2 = 0$.

$$\left(\frac{\partial_1}{I}\right) = \left(\begin{array}{cccccccc} -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ \hline 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right) \longrightarrow \left(\begin{array}{cccccccc} -1 & 0 & 0 & 0 & 0 & 0 & \overset{\downarrow}{0} & \overset{\downarrow}{0} \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ \hline 1 & -1 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & -1 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right) = \left(\frac{\partial_1 Q}{Q}\right).$$

Llamamos $R_3 = Q$.

Función inducida y persistencia

$$R_2^{-1} \cdot M \cdot R_3 = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & -1 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & -1 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} =$$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \boxed{1} & \boxed{0} \end{pmatrix}.$$

Si tomamos la submatriz determinada por las columnas correspondientes a representantes del grupo de homología de C_3 y las filas correspondientes a los del grupo de homología de C_2, obtenemos la siguiente submatriz:

$$(M)_* = \begin{pmatrix} 1 & 0 \end{pmatrix}.$$

Como esta matriz tiene rango 1 y la dimensión de su núcleo es 1, sabemos que uno de los 1-agujeros en profundidad 3 persiste al colapsar en profundidad 2 y otro muere. Esto puede comprobarse inmediatamente en la representación gráfica.

3.3. Cálculo del código de barras

Fijada una dimensión k, tenemos $0 \xleftarrow[(M_0)_*]{} H^0 \xleftarrow[(M_1)_*]{} H^1 \xleftarrow[(M_2)_*]{} H^2 \xleftarrow[(M_3)_*]{} \cdots$.

En los lemas que siguen, el colapso trivial en cero corresponde a la etapa -1.

Lema 3.1. *El número de k-agujeros que existen en una etapa i del módulo y persisten en una* $j-1$ *es igual a rango* $(M_j \cdot M_{j+1} \cdot \cdots \cdot M_{i-1} \cdot M_i)$.

Demostración. Inmediato de la definición de número de Betti persistente en Definición 1.19. □

Lema 3.2. *El número de k-agujeros que existen en una etapa i del módulo y han muerto en una* $j-1$ *es igual a nulidad* $(M_j \cdot M_{j+1} \cdot \cdots \cdot M_{i-1} \cdot M_i)$.

Demostración. Directo de Lema 3.1 y el teorema del rango. □

Los dos lemas anteriores nos permiten obtener de las matrices $(M)_*$ inducidas ya calculadas toda la información del código de barras. Además, en todo momento existe la posibilidad de añadir una profundidad superior recalculando adecuadamente. Conviene observar que los códigos de barras generados de esta manera siguen la dirección contraria a los códigos de barras habituales, es decir, $(i, j]$ en lugar de $[i, j)$.

Algoritmo 2 Cálculo del código de barras

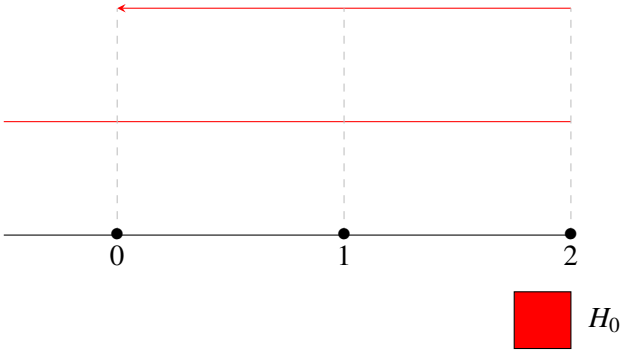
Require: Existen $M_0, \ldots M_n$
/ $D_{i,j}$ son las barras que existen en i y han muerto en j-1 */*
for i in $0 \ldots n$ **do**
  for j in $0, \ldots, i$ **do**
    $D_{i,j} \leftarrow \text{nullity}(M_j \cdot M_{j+1} \cdot \cdots \cdot M_i)$
  end for
end for
/ En cada profundidad de nacimiento (fila), descartamos las barras que han muerto en profundidades anteriores y estaban acumuladas, de forma que el número de la columna j pasa a representar las barras que mueren exactamente en* $j-1$ *y existen (que no necesariamente nacen) en la profundidad correspondiente a la fila. */*
for i in $0, \ldots, n$ **do**
  for j in $i, i-1, \ldots, 0$ **do**
    $D_{i,j} \leftarrow D_{i,j} - \sum_{k>j} D_{i,k}$
  end for
end for
/ En cada profundidad de muerte (columna), descartamos las barras que no han nacido en ese nivel de profundidad de nacimiento (fila). */*
for i in $n, n-1, \ldots, 0$ **do**
  for j in $0, \ldots, i-1$ **do**
    for k in $j, \ldots, i-1$ **do**
      $D_{k,j} \leftarrow D_{k,j} - D_{i,j}$
    end for
  end for
end for
/ Cada $D_{i,j}$ corresponde al número de barras nacidas en profundidad i que mueren en j-1. */*
  return D

3.3.1. Ejemplo

Veamos los pasos que habría que realizar sobre la matriz de D de la homología persistente en dimensión 0 correspondiente a Ejemplo 3.2.1.

$$\begin{pmatrix} 1 & & \\ 2 & 1 & \\ 2 & 1 & 0 \end{pmatrix} \longrightarrow \begin{pmatrix} 1 & & \\ 1 & 1 & \\ 1 & 1 & 0 \end{pmatrix} \longrightarrow \begin{pmatrix} 0 & & \\ 0 & 0 & \\ 1 & 1 & 0 \end{pmatrix}.$$

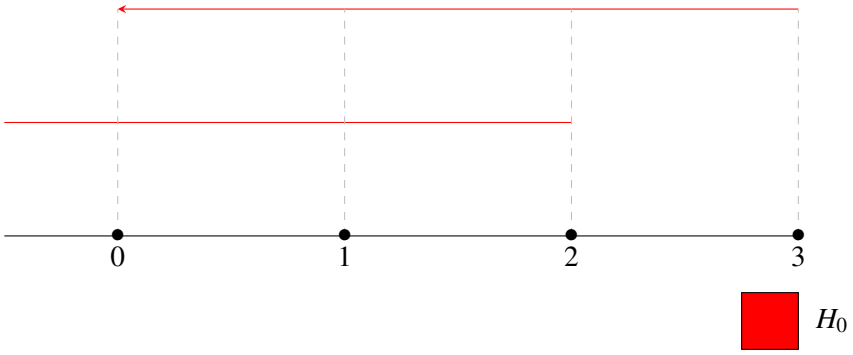
Y obtendríamos el siguiente código de barras.



Observemos que, si aumentamos la dimensión y añadimos una nueva profundidad, sólo es necesario recalcular en las dos últimas filas de la nueva matriz.

$$\begin{pmatrix} 0 & & & \\ 0 & 0 & & \\ 1 & 1 & 0 & \\ 1 & 1 & 0 & 0 \end{pmatrix} \longrightarrow \begin{pmatrix} 0 & & & \\ 0 & 0 & & \\ 1 & 1 & 0 & \\ 0 & 1 & 0 & 0 \end{pmatrix} \longrightarrow \begin{pmatrix} 0 & & & \\ 0 & 0 & & \\ 1 & 0 & 0 & \\ 0 & 1 & 0 & 0 \end{pmatrix}.$$

El código de barras actualizado quedaría:



Capítulo 4

Resultados y conclusiones

4.1. Implementación

Los capítulos 2 y 3 han sido implementados en un único programa en *C++* que acepta como entrada nubes de puntos ya normalizadas en formato *txt*. Los algoritmos 1 y 2 han sido casi traducidos del pseudocódigo al lenguaje de la implementación. Las funciones expansión y ExisteSímplice se han implementado de forma recursiva.

Las reducciones a forma escalonada tanto por filas como por columnas han sido implementadas manualmente tomando como base los algoritmos de [10].

Para la aritmética y el cálculo matricial en cuerpos finitos, se ha utilizado la librería *LinBox*[11] y su librería hija *Givaro*.

Todo el código fuente está disponible en [12].

4.2. Resultados

Veamos la información obtenida de unos ejemplos gráficos con figuras sencillas.

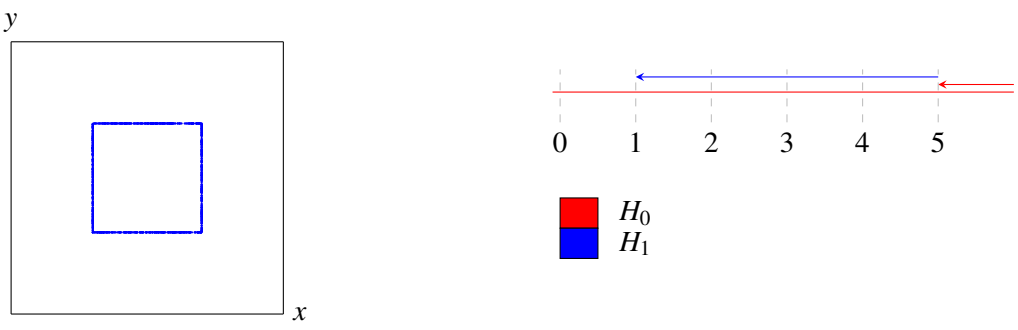


Figura 4.1: Código de barras de una nube de 1000 puntos cuadrada

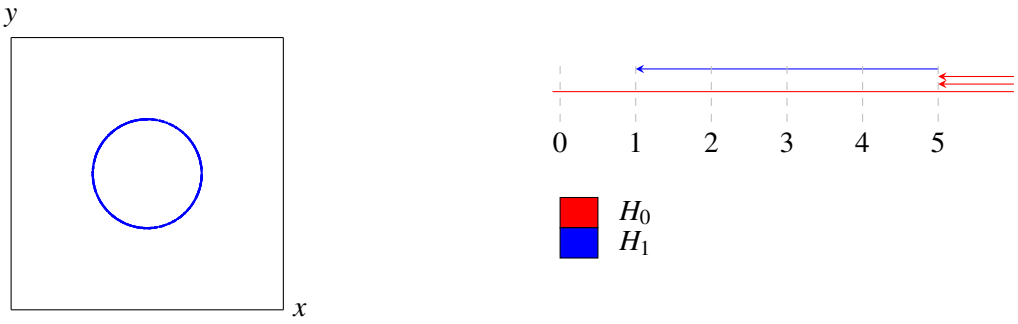


Figura 4.2: Código de barras de una nube de 1000 puntos circular

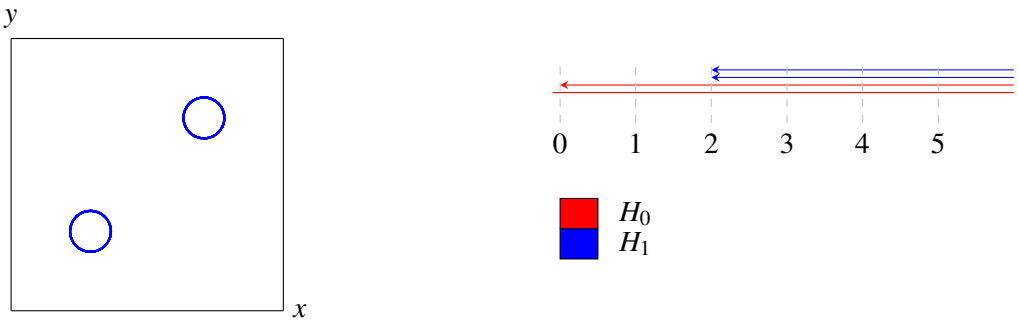


Figura 4.3: Código de barras de dos nubes circulares de 1000 puntos

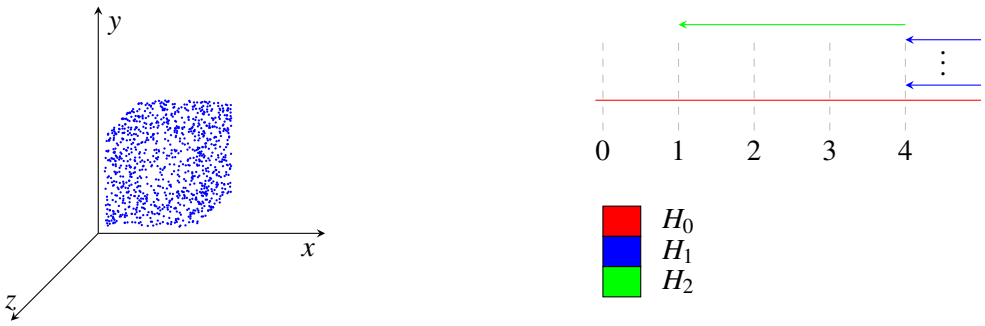


Figura 4.4: Código de barras de una nube de 10000 puntos de un cubo

Figura	Dimensión del espacio	Número de puntos	Resultados relevantes
Circunferencia	2	100	1-agujero [2,1)
Circunferencia	2	1000	1-agujero [5,1)
Circunferencia	2	10000	1-agujero [7,1)
2 circunferencias	2	100	2 1-agujeros [3,2)
2 circunferencias	2	1000	2 1-agujeros [6,2)
2 circunferencias	2	10000	2 1-agujeros [7,2)
Cuadrado	2	100	1-agujero [2,1)
Cuadrado	2	1000	1-agujero [5,1)
Cuadrado	2	10000	1-agujero [8,1)
Cubo	3	100	Ningún 2-agujero
Cubo	3	1000	2-agujero [2,1)
Cubo	3	10000	2-agujero [4,1)

Figura 4.5: Resultados de ejecución hasta profundidad 8

4.3. Conclusiones

Los resultados obtenidos demuestran que el método propuesto en este trabajo para el cálculo de códigos de barras es capaz de detectar agujeros relevantes en nubes de puntos. A diferencia de las soluciones ya existentes, que construyen complejos partiendo de resoluciones muy pequeñas hasta llegar al colapso trivial de todos los puntos, nuestro método parte de este colapso y recorre el camino contrario de forma controlada. Creemos que esta idea presenta varias ventajas sobre el método anterior, ya que reduce bastante el coste computacional en nubes de puntos grandes y permite parar en una profundidad cualquiera.

El almacenamiento unipuntual de símplices del capítulo 2 presenta otra ventaja frente a los métodos de representación combinatoria, ya que cada símplice, en lugar de ser conjunto de vértices abstractos de tamaño variable, se representa únicamente como un punto.

Puede resultar confusa la notación de barras decrecientes $[i, j)$ con $i > j$, que parece contraria a la habitual, pero es en realidad casi equivalente a una barra estándar $[2^j, 2^i)$, considerando aproximaciones. Así, resulta fácil de interpretar por cualquier librería de análisis topológico de datos.

En definitiva, creemos que este método está especialmente indicado para nubes de puntos grandes en resoluciones limitadas.

Bibliografía

[1] Talha Qaiser et al. «Fast and accurate tumor segmentation of histology images using persistent homology and deep convolutional features». En: *Medical image analysis* 55 (2019), págs. 1-14.

[2] Kelin Xia y Guo-Wei Wei. «Persistent homology analysis of protein structure, flexibility, and folding». En: *International journal for numerical methods in biomedical engineering* 30.8 (2014), págs. 814-844.

[3] Frédéric Chazal et al. «Gromov-Hausdorff stable signatures for shapes using persistence». En: *Computer Graphics Forum*. Vol. 28. 5. Wiley Online Library. 2009, págs. 1393-1403.

[4] Rui Zhang et al. «Predicting Tropical Cyclone Rapid Intensification in Western North Pacific Basin using a TDA-RI Model from Digital Typhoon Dataset». En: *IEEE Transactions on Geoscience and Remote Sensing* (2024).

[5] Mohnhaupt Mona. *The Nerve Theorem and its Applications in Topological Data Analysis*. 2023.

[6] Vin De Silva y Robert Ghrist. «Coverage in sensor networks via persistent homology». En: *Algebraic & Geometric Topology* 7.1 (2007), págs. 339-358.

[7] Afra Zomorodian y Gunnar Carlsson. «Computing persistent homology». En: *Proceedings of the twentieth annual symposium on Computational geometry*. 2004, págs. 347-356.

[8] Horst Schubert. *Topology*. 1968.

[9] Tomasz Kaczynski, Konstantin Mischaikow y Marian Mrozek. *Computational homology*. Vol. 157. Springer Science & Business Media, 2006.

[10] Jeremy Kun. *Computing homology*. 2013. URL: https://www.jeremykun.com/2013/04/10/computing-homology/.

[11] The LinBox group. *LinBox*. v1.6.3. 2019. URL: http://github.com/linbox-team/linbox.

[12] *Código de Métodos adaptativos para el cálculo de homología persistente cúbica*. 2025. URL: https://github.com/845977/cubitos.

[13] Allen Hatcher. *Algebraic Topology*. Cambridge University Press, 2002.

[14] Robert Ghrist. «Barcodes: the persistent topology of data». En: *Bulletin of the American Mathematical Society* 45.1 (2008), págs. 61-75.

[15] Cheyne Glass y Elizabeth Vidaurre. «Topological Data Analysis via Undergraduate Linear Algebra». En: *arXiv preprint arXiv:2406.17045* (2024).

[16] David Poole. *Linear algebra: A modern introduction*. 2015.

[17] Nahae Kühn. *Cubical Homology and Applications*. 2023.

Anexos

Anexo A

Demostraciones

A.1. Demostración con otra notación de Lema 2.1

Demostración.

$$\begin{aligned}
\partial_{k-1}\circ\partial_k(Q) =& \partial_{k-1}\left(\sum_i (-1)^i\left([a_1,b_1]\times\cdots\times[a_i,a_i]\times\cdots\times[a_m,b_m]-[a_1,b_1]\times\cdots\times[b_i,b_i]\times\cdots\times[a_m,b_m]\right)\right)= \\
&=\sum_{i<j}(-1)^{i+j+1}[a_1,b_1]\times\cdots\times[a_i,a_i]\times\cdots\times[a_j,a_j]\times\cdots\times[a_m,b_m]+ \\
&+\sum_{i>j}(-1)^{i+j}[a_1,b_1]\times\cdots\times[a_j,a_j]\times\cdots\times[a_i,a_i]\times\cdots\times[a_m,b_m]+ \\
&+\sum_{i<j}(-1)^{i+j+1}[a_1,b_1]\times\cdots\times[b_i,b_i]\times\cdots\times[b_j,b_j]\times\cdots\times[a_m,b_m]+ \\
&+\sum_{i>j}(-1)^{i+j}[a_1,b_1]\times\cdots\times[b_j,b_j]\times\cdots\times[b_i,b_i]\times\cdots\times[a_m,b_m]+ \\
&+\sum_{i<j}(-1)^{i+j}[a_1,b_1]\times\cdots\times[a_i,a_i]\times\cdots\times[b_j,b_j]\times\cdots\times[a_m,b_m]+ \\
&+\sum_{i>j}(-1)^{i+j+1}[a_1,b_1]\times\cdots\times[a_j,a_j]\times\cdots\times[b_i,b_i]\times\cdots\times[a_m,b_m]+ \\
&+\sum_{i<j}(-1)^{i+j}[a_1,b_1]\times\cdots\times[b_i,b_i]\times\cdots\times[a_j,a_j]\times\cdots\times[a_m,b_m]+ \\
&+\sum_{i>j}(-1)^{i+j+1}[a_1,b_1]\times\cdots\times[b_j,b_j]\times\cdots\times[a_i,a_i]\times\cdots\times[a_m,b_m]=0
\end{aligned}$$

□

A.2. Demostración completa de Lema 2.4

Demostración. Por el método de generación del complejo en Algoritmo 1, un intervalo puede ser no degenerado de varias formas. Sea q_k el punto equivalente a un intervalo de profundidad i. Existen dos posibilidades:

1. Que proceda de la expansión de un intervalo degenerado. Un intervalo degenerado se expande a uno no degenerado si la función expansión no lo modifica. En este caso, tendremos que $q_k \in S_{i-1}$ y claramente $q_k \notin S_i$. Además, $q_k - \text{salto}_{i-1}, q_k + \text{salto}_{i-1} \in S_i$. Por Definición 2.10 se sigue el resultado.

2. Que proceda de la expansión de un intervalo no degenerado. Un intervalo no degenerado no puede, por nuestro algoritmo, convertirse de ningún modo en uno degenerado. Como todos los intervalos no degenerados proceden necesariamente de la expansión de uno degenerado (aunque sea del punto central inicial), tenemos que $q_k \in S_j$ para algún $j < i$. Como

$$q_k - \text{salto}_{i-1} = q_k - \text{salto}_j + \text{salto}_{j+1} + \cdots + \text{salto}_{i-1} \text{ y}$$

$$q_k + \text{salto}_{i-1} = q_k + \text{salto}_{j-1} - \text{salto}_j - \text{salto}_{j+1} - \cdots - \text{salto}_{i-1},$$

tenemos que $q_k \pm \text{salto}_{i-1} \in S_i$.

Además,

$$q_k + \text{salto}_{i-1} - (q_k - \text{salto}_{i-1}) = 2 \cdot \text{salto}_{i-1},$$

es decir, la mínima distancia entre puntos en S_i, por lo que $q_k - \text{salto}_{i-1}, q_k + \text{salto}_{i-1} \in S_i$ son los dos puntos en S_i más cercanos a q_k y el resultado está probado.

□

A.3. Caso restante de la demostración de Lema 2.6

- $f^j(Q) = R$ de dimensión menor que $i-1$.

  Supondremos, sin pérdida de generalidad, que R tiene dimensión $i-2$.

  En primer lugar, $f_i^j(Q) = 0$. Tenemos además que

$$f^j(Q) = R \Rightarrow Q = R + (x_1, \cdots, x_n) \text{ con } x_l = \begin{cases} 0 & \text{si } l \in \text{NoDeg}_{j-1}(R) \cup \{k_1, k_2\} \text{ y} \\ \pm\text{salto}_{j-1} & \text{si } l \in \text{Deg}_{j-1}(R) \setminus \{k_1, k_2\}, \end{cases}$$

  para $k_1, k_2 \in \text{Deg}_{j-1}(R)$. Por Lema 2.4, si T es una cara Q:

$$T = Q \pm salto_{j-1}^l = R + (x_1, \cdots, x_n) \pm \text{salto}_{j-1}^l \text{ con } l \in \text{NoDeg}_j(Q) = \text{NoDeg}_{j-1}(R) \cup \{k_1, k_2\}.$$

  - Si $l \in \text{NoDeg}_{j-1}(R)$, T provendría de una cara de R de dimensión $i-3$, de forma similar al primer caso.
  - Si $l = k_1$, tendríamos que $T = R + (y_1, \cdots, y_n)$, con

$$y_l = \begin{cases} 0 & \text{si } l \in \text{NoDeg}_{j-1}(R) \cup \{k_2\} \text{ y} \\ \pm\text{salto}_{j-1} & \text{si } l \in \text{Deg}_{j-1}(R) \setminus \{k_2\}, \end{cases}$$

    por lo que $T \in \mathfrak{e}_{j-1}(R)$ y, como $\dim_{j-1}(R) = i-2$, $f_{i-1}^j(T) = 0$.
  - Si $l = k_2$, es completamente análogo al caso anterior.

  En definitiva,

$$\partial_i^{j-1} \circ f_i^j(Q) = f_{i-1}^j \circ \partial_i^j(Q) = 0.$$

Anexo B

Códigos

B.1. Generadores de nubes de puntos

```python
# File: utils.py

# Pre: cloud is in [0,1]x[0,1]
def to_tikz(cloud : [float]) -> str:
    ret = ""
    ret += "\\begin{tikzpicture}\n"
    if len(cloud[0]) == 2:
        ret += "    \\draw[-] (-0.1,-0.1) -- (1.1,-0.1) node[right] {$x$};\n"
        ret += "    \\draw[-] (-0.1,-0.1) -- (-0.1,1.1) node[above] {$y$};\n"
        ret += "    \\draw[-] (-0.1,1.1) -- (1.1,1.1);\n"
        ret += "    \\draw[-] (1.1,-0.1) -- (1.1,1.1);\n"

        for p in cloud:
            ret += f"    \\fill[blue] ({p[0]},{p[1]}) circle[radius=0.005];\n"

    elif len(cloud[0]) == 3:
        for p in cloud:
            ret += f"    \\fill[blue] ({p[0]},{p[1]},{p[2]}) circle[radius=0.005];\n"

    ret += "\\end{tikzpicture}"

    return ret
```

```python
# File: square_sampler.py
import numpy as np
from utils import to_tikz

points = []

num_points = 1000
center = 0.5
radius = 0.24
min, max = center - radius, center + radius

for _ in range(num_points):
    side = np.random.randint(0, 4)
    pos = np.random.uniform(min,max)
    if side == 0:
        points.append([min, pos])
    if side == 1:
        points.append([max, pos])
    if side == 2:
        points.append([pos, min])
    if side == 3:
        points.append([pos, max])

with open('square.txt', 'w') as f:
    for point in points:
        f.write(f"{point[0]:.8f} {point[1]:.8f}\n")

with open('square.tikz', 'w') as f:
```

```
    f.write(to_tikz(points))
```

```python
# File: circumference_sampler.py
import numpy as np
from utils import to_tikz

num_points = 1000
center = 0.5
radius = 0.24
points = []

for _ in range(num_points):
    angle = np.pi * np.random.uniform(0, 2)
    points.append([center - radius*np.cos(angle),
                   center - radius*np.sin(angle)])

with open('circunference.txt', 'w') as f:
    for point in points:
        f.write(f"{point[0]:.8f} {point[1]:.8f}\n")

with open('circunference.tikz', 'w') as f:
    f.write(to_tikz(points))
```

```python
# File: 2circumference_sampler.py
import numpy as np
from utils import to_tikz

num_points = 1000
radius = 0.09
points = []

for _ in range(num_points):
    angle = np.pi * np.random.uniform(0, 2)
    # First circumference
    points.append([0.25 - radius*np.cos(angle),
                   0.25 - radius*np.sin(angle)])
    # Second circumference
    angle = np.pi * np.random.uniform(0, 2)
    points.append([0.75 - radius*np.cos(angle),
                   0.75 - radius*np.sin(angle)])

with open('2circunference.txt', 'w') as f:
    for point in points:
        f.write(f"{point[0]:.8f} {point[1]:.8f}\n")

with open('2circunference.tikz', 'w') as f:
    f.write(to_tikz(points))
```

```python
# File: cube_sampler.py
import numpy as np
from utils import to_tikz

num_points = 1000

center = 0.5
radius = 0.201
min, max = center - radius, center + radius

points = []

for _ in range(num_points):
    face = np.random.randint(0, 6)
    u, v = np.random.uniform(min, max, size=2)

    if face == 0:
        point = [min, u, v]
    elif face == 1:
        point = [max, u, v]
    elif face == 2:
        point = [u, min, v]
    elif face == 3:
```

```
        point = [u, max, v]
    elif face == 4:
        point = [u, v, min]
    elif face == 5:
        point = [u, v, max]

    points.append(point)

with open('cube.txt', 'w') as f:
    for point in points:
        f.write(f"{point[0]:.8f} {point[1]:.8f} {point[2]:.8f}\n")

with open('cube.tikz', 'w') as f:
    f.write(to_tikz(points))
```

B.2. Reducciones de matrices

```
#pragma once
/*
 * file: algorithms/reductions.h
 */

#include "../smatrix.h"

namespace cubitos {

/* Algorithms based on
 * https://www.jeremykun.com/2013/04/10/computing-homology/
 */

template <size_t _N, bool _EnableComplementary>
void rowReduce(SMatrix<_N>& A, SMatrix<_N>& B, SMatrix<_N>& P, SMatrix<_N>& P_inv) {
    auto field = SMatrix<_N>::field_;

    size_t numRows = A.n_, numCols = A.m_;

    /* Row echelonizes A into P A */
    for (size_t i = 0, j= 0; i < numRows && j < numCols;) {
        size_t nonzeroRow = i;
        for (; nonzeroRow < numRows && field.isZero(A.get(nonzeroRow, j));
             nonzeroRow++);
        if (nonzeroRow == numRows) {
            j++;
            continue;
        } else if (nonzeroRow != i) {
            A.rowSwap(i, nonzeroRow);
            P.rowSwap(i, nonzeroRow);
            // If we have set a complementary matrix
            if constexpr (_EnableComplementary) {
                B.colSwap(i, nonzeroRow);
            }
            P_inv.colSwap(i, nonzeroRow);
        }

        Element pivot = A.get(i, j);
        field.invin(pivot);
        field.negin(pivot);
        for (size_t otherRow = i + 1; otherRow < numRows; otherRow++) {
            Element c = A.get(otherRow, j);
            if (!field.isZero(c)) {
                field.mulin(c, pivot);
                A.rowCombine(otherRow, i, c);
                P.rowCombine(otherRow, i, c);
                field.negin(c);
                P_inv.colCombine(i, otherRow, c);
                // If we have set a complementary matrix
                if constexpr (_EnableComplementary) {
                    B.colCombine(i, otherRow, c);
                }
            }
```

```
        }

        i++; j++;
    }
}

template <size_t _N, bool _EnableComplementary>
void columnReduce(SMatrix<_N>& A, SMatrix<_N>& B, SMatrix<_N>& Q, SMatrix<_N>& Q_inv) {
    auto field = SMatrix<_N>::field_;
    size_t numRows = A.n_, numCols = A.m_;

    /* Column echelonizes A into A Q */
    for (size_t i = 0, j = 0; i < numRows && j < numCols;) {
        size_t nonzeroCol = j;
        for (; nonzeroCol < numCols && field.isZero(A.get(i, nonzeroCol));
             nonzeroCol++);

        if (nonzeroCol == numCols) {
            i++;
            continue;
        } else if (nonzeroCol != j) {
            A.colSwap(j, nonzeroCol);
            Q.colSwap(j, nonzeroCol);
            // If we have set a complementary matrix
            if constexpr (_EnableComplementary) {
                B.rowSwap(j, nonzeroCol);
            }
            Q_inv.rowSwap(j, nonzeroCol);
        }

        Element pivot = A.get(i, j);
        field.invin(pivot);
        field.negin(pivot);
        for (size_t otherCol = j + 1; otherCol < numCols; otherCol++) {
            Element c = A.get(i, otherCol);
            if (!field.isZero(c)) {
                field.mulin(c, pivot);
                A.colCombine(otherCol, j, c);
                Q.colCombine(otherCol, j, c);
                field.negin(c);
                // If we have set a complementary matrix
                if constexpr (_EnableComplementary) {
                    B.rowCombine(j, otherCol, c);
                }
                Q_inv.rowCombine(j, otherCol, c);
            }
        }

        i++; j++;
    }
}

template <size_t _N>
void columnReduce(SMatrix<_N>& A, SMatrix<_N>& B, SMatrix<_N>& Q, SMatrix<_N>& Q_inv) {
    columnReduce<_N, true>(A, B, Q, Q_inv);
}

template <size_t _N>
void columnReduce(SMatrix<_N>& A, SMatrix<_N>& Q, SMatrix<_N>& Q_inv) {
    static SMatrix<_N> dummy = SMatrix<_N>::zeroMatrix();
    Q = SMatrix<_N>::identity(A.m_);
    Q_inv = SMatrix<_N>::identity(A.m_);

    columnReduce<_N, false>(A, dummy, Q, Q_inv);
}

template <size_t _N>
void rowReduce(SMatrix<_N>& A, SMatrix<_N>& B, SMatrix<_N>& Q, SMatrix<_N>& Q_inv) {
    rowReduce<_N, true>(A, B, Q, Q_inv);
}

template <size_t _N>
void rowReduce(SMatrix<_N>& A, SMatrix<_N>& P, SMatrix<_N>& P_inv) {
```

```
127     static SMatrix<_N> dummy = SMatrix<_N>::zeroMatrix();
128     P = SMatrix<_N>::identity(A.n_);
129     P_inv = SMatrix<_N>::identity(A.n_);
130 
131     rowReduce<_N, false>(A, dummy, P, P_inv);
132 }
133 
134 template <size_t _N>
135 void simultaneousReduce(SMatrix<_N>& A, SMatrix<_N>& B, SMatrix<_N>& R,
136                         SMatrix<_N>& R_inv) {
137     assert(A.m_ == B.n_);
138 
139     R = SMatrix<_N>::identity(A.m_);
140     R_inv = SMatrix<_N>::identity(A.m_);
141 
142     columnReduce(A, B, R, R_inv);
143     rowReduce(B, A, R_inv, R);
144 }
145 
146 }  // namespace cubitos
```