Homología Persistente

Brayan Alejandro Romero Castro[1], Carlos Alberto Garzon Gutierrez[2], Eduardo Andrés Arrieta Contreras[3], and Jose Alfredo Urrea Moreno[4]

[1 - 4]Departamento de Matemáticas , Universidad Nacional de Colombia

Diciembre, 2022

Resumen

En el siguiente trabajo se presentan algunos ejemplos en donde se aplica la Homología Persistente, una rama de la Topología que ha resultado útil en los últimos años en el análisis de bases grandes de datos como se detalla en [3], y como se verá a lo largo del documento. Nuestro principal objetivo es describir como se puede aplicar la homología persistente por medio de Julia, en particular utilizando el paquete Eirene que da bastantes herramientas para este tipo de análisis topológico de datos (Ver [6]), Comenzaremos dando una introducción teórica de lo que es la homología persistente, luego pasaremos a dar un vistazo de un pequeño ejemplo en donde TDA (Topological Data Analisys), resulta mas efectivo que otros métodos convencionales para predecir posibles catástrofes en los precios de las acciones, luego seguiremos con la descripción del algoritmo que se utilizara, para terminar con los ejemplos y el código utilizado.

1. Homología

La homología persistente es una área de la topología aplicada relativamente nueva. Con apenas 30 años desde su creación ha obtenido gran popularidad en la última década dada la necesidad de manejar y entender grandes cantidades de datos, que se hacen cada vez mas inestables y costosas. La homología persistente presenta una solución a este problema [1].

La teoría matemática presentada en este texto se puede dividir en tres grandes secciones que son la topología, el álgebra lineal y optimización combinatoria. Estos van a ser los ejes centrales para explicar de manera general la teoría de homología y su computabilidad, que permiten a la homología persistente ser una teoría efectiva en la solución del problema de manejo de grandes bases de datos.

Empecemos hablando de los datos. Estos no vienen de una teoría abstracta, por el contrario, son recolectados de un problema real. La primera aplicación de la homología persistente se remonta a un problema de visualización en imágenes médicas, las cuales son procesadas por un computador como un arreglo reticular, donde en cada posición se almacena un valor de intensidad de luz. Estas matrices de números pueden ser considerablemente costosas si se busca una imagen con muchos detalles.

En este trabajo no nos centraremos en la visualización de imágenes, pues existe otra importante clase de datos, los cuales son las nubes de puntos. Estos son subconjuntos finitos de puntos de algún espacio Euclídeo, cada punto de la nube representa una serie de características de un individuo o de una región particular en una población o espacio. El análisis topológico de datos tiene como principio fundamental que los datos tienen forma, que esa forma tiene propiedades geométricas y que dichas propiedades estudiadas algebraicamente otorgan información local y global sobre los datos. A su vez, los primeros grupos de homología capturan fenómenos topológicos tales como la conexidad, los ciclos y los vacíos. En general, la homología es un tratamiento formal del concepto hueco n-dimensional.

Precisando sobre la homología, decimos que esta es una asignación de la clase de espacios topológicos a la clase de grupos abelianos, que se preserva bajo morfismos (funciones que preservan estructuras), es decir si al espacio X le asociamos $H_n(X)$ y a Y le asociamos $H_n(Y)$ su $n-$ésimo grupos de homología, y si además suponemos que existe una función continua de X a Y, entonces existe un homomorfismo de grupos entre $H_n(X)$ y $H_n(X)$. Junto con otros axiomas de compatibilidad entre estas clases de estructuras, diremos que esta asociación es un functor entre la categoría de espacios topológicos y grupos abelianos. Una propiedad inmediata de esta functorialidad es que para espacios homeomorfos tendremos grupos de homología isomorfos. De está manera los grupos de homología funcionan como invariantes topológicos y nos permiten verificar cuando dos espacios son distintos. Por ejemplo, se puede mostrar con ayuda de este invariante que $\mathbb{R}^n \neq \mathbb{R}^m$ siempre que $n \neq m$. Aunque esta afirmación parezca evidente, no resultar ser nada sencillo de demostrar. Sucede lo mismo cuando queremos demostrar que los siguientes espacios no son homeomorfos

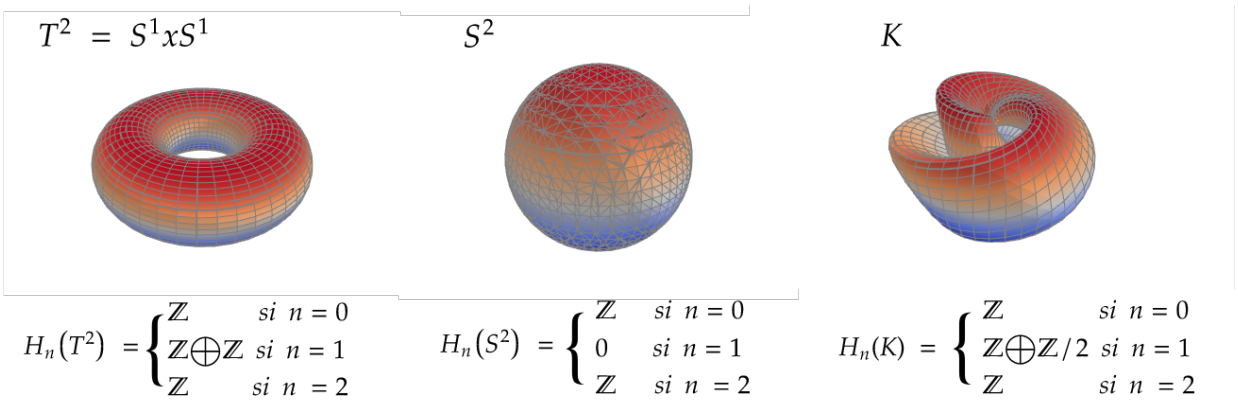


Figura 1: Tres superficies suaves que no son isomorfas entre sí con sus respectivos grupos de homología que permite diferenciarlos.

Este invariante no es perfecto, ya que pueden existir espacios distintos con los mismos grupos de homología. Sin embargo, es lo suficientemente fuerte para diferenciar una gran cantidad de espacios, y a su vez relativamente fácil de calcular. Para una introducción completa en homología ver [2]

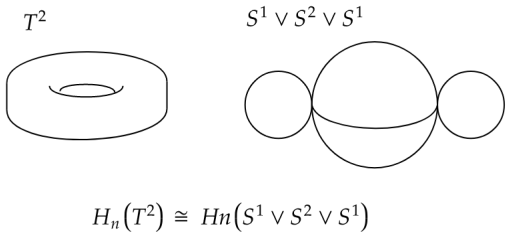


Figura 2: El toro y una esfera con dos círculos pegados tienen los mismos grupos de homología. El invariante no es lo suficientemente fuerte para diferenciar estos dos espacios.

Antes de construir este functor, tenemos que hablar sobre complejos simpliciales, ya que es la manera clásica de representar una colección discreta como una variedad topológica y nos va a dar una mejor idea de

por qué toda está maquinaria abstracta nos permite atacar problemas reales. Un complejo simplicial es una colección de bloques bien pegados, donde cada bloque es lo que llamaremos un simplejo. Un simplejo σ de dimensión n es la envoltura convexa (Convex Hull) de $n+1$ puntos afines linealmente independientes. Un punto es un 0-simplejo, un segmento de recta es un 1- simplejo, un triángulo es un 3-simplejo y un tetraedro es un 4-simplejo, y así en más dimensiones. Observe que un $n-$simplejo vive naturalmente en el espacio afín $\mathbb{R}^{n+1}$, si el espacio no es Euclideo entonces el simplejo es cualquier función de σ a el espacio. Para el n-simplejo σ, consideramos la envoltura convexa de k puntos en σ con $k < n$. Este k-simplejo es lo que llamaremos una cara de σ. Ahora, un complejo simplicial Σ es un conjunto de simpliciales pegados de tal manera que cada cara de un simplejo en Σ es una cara de Σ, y además la intersección no vacía de simplejos en Σ es una cara de ambos. Esto se puede entender mejor con la siguiente imagen.

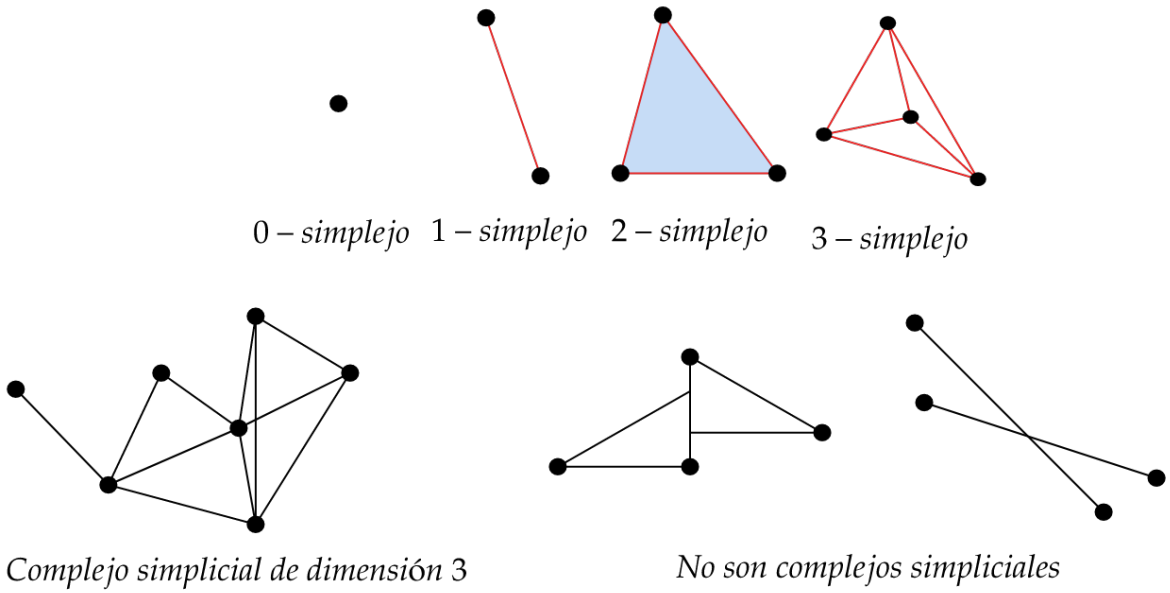


Figura 3: Tenemos en la parte de arriba los primos simpliciales de dimensión 0,1,2 y 3. Abajo, vemos como se pueden pegar estos simplejos para construir espacios mas complejos y también vemos lo que definitivamente no es un complejo.

Dado un complejo simplicial Σ, es posible definir una cadena de complejos asociado de Σ, que se denota como $C_*(\Sigma) = (C_k(\Sigma), \delta_k)_{k\in\mathbb{Z}}$, donde $C_k(\Sigma)$ es el grupo libre abeliano generado por los k-simplejos de Σ y $\delta_k : C_k(\Sigma) \to C_{k-1}(\Sigma)$ es un homomorfismo, que a un complejo de C_k lo envía en sus respectivas caras de C_{k-1} con unos ciertos signos asociados a cada cara, de tal manera que $\delta^2 = \delta_k \circ \delta_{k-1} = 0$. Esta última formula es equivalente a $Ker(\delta_k) \subseteq Im(\delta_{k+1})$. Denotamos como $Z_k(\Sigma) = ker(\delta_k)$ el grupo de k-ciclos, y $B_k(\Sigma) = Im(\delta_{k+1})$ el grupo de $k-$fronteras. Como estos son grupos libres abelianos, entonces definimos el k-ésimo grupo de homología de Σ como

$$H_n(\Sigma) = Z_k(\Sigma)/B_k(\Sigma)$$

Veamos porque se dice que los grupos de homología capturan la idea hueco en dimensión 1, es decir, detallemos lo que geométricamente representa $H_1(\Sigma)$. Nos vamos a centrar en lo que sucede con $C_1(\Sigma)$, que es el grupo abeliano libre generado por segmentos de rectas σ, con $\delta(\sigma) = x_0 - x_1$, donde x_0 y x_1 son los puntos inicial y final de σ. ¿Qué significa que para un complejo $\Sigma_0 \in C_1(\Sigma)$ su frontera sea cero, es decir $\delta(\Sigma_0) = 0$? Observemos el siguiente diagrama que muestra un caso

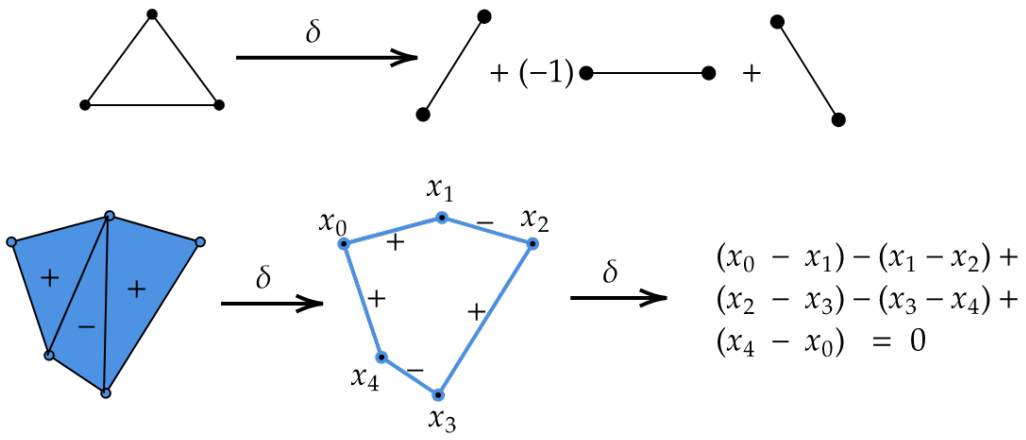


Figura 4: El operador frontera toma un complejo y aditivamente envía el complejo a la suma alternada de sus caras, todo esto con el propósito de que al efectuarse dos veces el operador sea 0.

Luego, Σ_0 es una curva cerrada en el espacio. Tomemos ahora alguien en la imagen, sea $\Sigma_0 = \delta_2(\Sigma_1)$. El complejo Σ_0 formado por líneas se puede construir por triángulos (simplejos de dimensión mayor) tales que su frontera es precisamente el complejo, ver (figura 3). Ahora, observe que al hacer el cociente de ciclos sobre fronteras estamos obteniendo un grupo abeliano (no necesariamente libre) que tiene como propiedad que dos simplejos están en la misma clase si y solo si existe un complejo simplicial que cubre la región comprendida y que al tomar su frontera obtenemos exactamente los dos simplejos, y en el caso de dimensión uno tenemos que los $1-$ciclos que no son frontera de nadie son precisamente los ciclos que rodean agujeros.

De manera general, la homología captura la existencia de estos agujeros, túneles, vacíos o discontinuidades del espacio y les asigna una clase por cada uno de estos fenómenos dependiendo de la dimensión que se este considerando. Una manera de formalizar y garantizar que efectivamente se está realizando esta caracterización es verificando los axiomas de Eilenberg-Steenrod, los cuales sin satisfechos por la homología simplicial. Estos axiomas caracterizan completamente el functor de homología, y toda teoría que satisfaga estos axiomas functoriales será naturalmente equivalente a la teoría de homología simplicial. Existen otras teorías de homología, como lo son la homología singular, la homología celular, la homología de cubicial y todas estas poseen distintos complejos de cadenas que nuevamente reconstruyen espacios a partir de bloques. La diferencia entre cada uno de estos planteamientos es que algunas son mas blandas en el sentido que permiten demostrar mas fácil ciertas propiedades categóricas y difíciles de calcular, mientras que otras son mas sencillas de calcular pero no permiten demostrar propiedades categóricas. Por ejemplo, la homología cubicial, los complejos son formados por cubos n-dimensionales y la homología celular construye los complejos a partir de esferas y al igual que la homología simplicial, estas teorías son computables.

1.1. Homología persistente

Dada una nube de puntos, queremos asociarle un espacio que preserve su forma y que tenga una topología más rica que la topología discreta Para lograrlo construiremos los complejos de Vietoris-Rips. La idea es considerar un $\epsilon > 0$ el cual será nuestra distancia umbral y consideramos las distancias entre cada par de puntos del espacio. Si dos puntos se encuentran a una distancia menor a ϵ entonces trazamos un segmento de recta que una a los dos puntos, o un 1-simplejo. Si tenemos tres puntos donde cada par está a una distancia menor al umbral entonces colocamos un triángulo con vértices los tres puntos, y así sucesivamente si tenemos n puntos donde cada par de puntos se encuentra a una distancia menor al umbral entonces colocamos un

n-simplejo con vértices los n puntos. De está manera hemos construido un espacio topológico asociado a la nube de puntos, que se asemeja en forma al espacio con posiblemente una topología más interesante. Claro está que aun no hemos solucionado nada, por el contrario ahora surgen nuevas preguntas y nuevos problemas

- ¿Cuál ϵ se debería tomar para el complejo de Vietoris-Rips?
- ¿Existe al menos un ϵ que verdaderamente me aproxime la nube de puntos a una variedad?
- ¿Si existiera el ϵ, este sería único?
- ¿El espacio topológico obtenido es bueno? ¿Qué significa que sea bueno?
- Dada una de estas topologías, ¿Es posible calcular computacionalmente su homología?
- ¿Cómo puedo interpretar está información topológica para solucionar problemas reales?

Varias de estás preguntas aun se encuentran sin una respuesta satisfactoria, pero varias de estás explican y motivan la razón de ser de la homología persistente. Centrémonos en la primer pregunta ¿Cuál ϵ se debería tomar?

Supongamos que existe una variedad topológica X que además es Riemmaniana. De manera general, nuestra variedad tiene muy buenas propiedades geométricas porque resulta ser una variedad orientable, Euclídea y diferenciable. Supongamos que nuestra colección de puntos efectivamente se asemeja a esta variedad de la siguiente manera: en vez de tomar distancias, vamos a considerar una bola alrededor de cada punto de tal manera que si n bolas se intersectan entonces pegamos un n-simplejo entre esos puntos, de forma análoga a la construcción del complejo de Vietoris-Rips. De hecho, estos complejos van a coincidir añadiendo una condiciones adicionales, y al resultado de esta adición se le conoce como complejos de Cech. Estos nos ofrecen el siguiente resultado

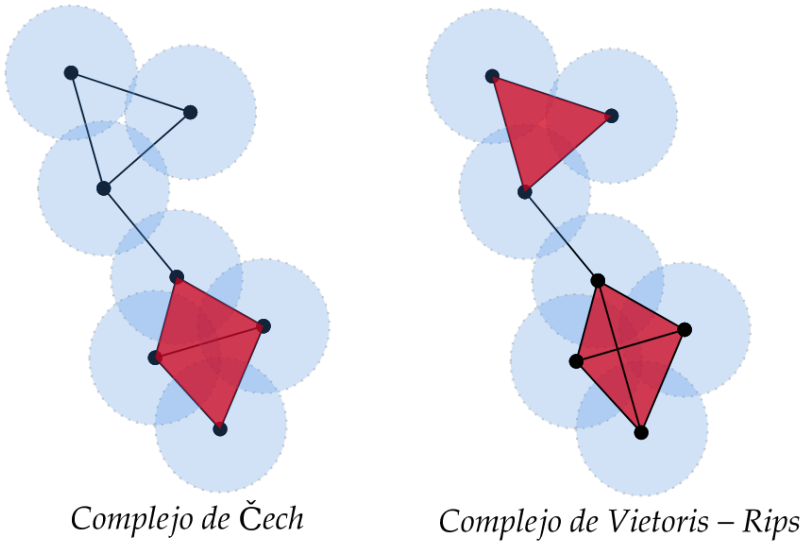


Figura 5: Se puede ver la diferencia entre las dos construcciones. Los complejos de Cech son considerablemente mas difíciles de computar, y la diferencia entre los dos no resulta ser determinante para los propósitos de este texto.

Teorema Sea X una variedad Riemanniana. Entonces existe un ϵ tal que para todo $e \leq \epsilon$, existe una nube de puntos finitos $D \subseteq X$ tal que el complejo de Cech asociado a la nube de puntos $C_e(D)$ es homotópico a X.

Decir que, el ser homotópicos significa que existe un funcional continuo que envía un espacio en el otro, tal que el funcional es invertible. La homotopía se comporta como una relación de equivalencia y es un concepto un poco más débil que el de isomorfismo. Este teorema muestra que normalmente los complejos de Cech, y bajo ciertas condiciones los complejos de Vietoris-Rips, son apropiados para estudiar nubes de puntos como variedades topológicas. Sin embargo, para nuestro infortunio, la demostración de este teorema es altamente no constructivo. A pesar de esto, como nuestra nube de datos es finita y de cierta manera estática, entonces el número de ϵ que da distintas topologías es finito. Por lo tanto, la homología persistente calcula la homología para un intervalo en vez de evaluar en un solo ϵ, y estudia como la homología cambia a lo largo de este intervalo. En particular estudia el comportamiento de los generadores de cada grupo de homología, observa como nuevos generadores nacen y como algunos desaparecen o mueren, y saca conclusiones de los que mas persisten en el intervalo. Para un tratamiento más profundo sobre el funcionamiento de la topología persistente leer [3].

1.2. Filtraciones

De forma general, la homología persistente describe los cambios en homología que le ocurre a un objeto, y estudia su evolución respecto a un parámetro. Formalmente, dado un complejo simplicial Σ, una filtración de Σ es una sucesión finita de subcomplejos $\Sigma^i := \{\Sigma^p : 0 \leq p \leq i\}$ de Σ tal que $= \Sigma^0 \subseteq \cdots \subseteq \Sigma^m = \Sigma$ para algún m número natural. Para $0 \leq p \leq q \leq m$, la $(p, q)-$persistencia del $k-$ésimo grupo de homología $H_k^{p,q}(\Sigma)$ de Σ consiste de los k-ciclos de $C_k(\Sigma^p)$ en $C_k(\Sigma^q)$ módulo el grupo de fronteras en Σ^q. Este grupo es la imagen del homomorfismo inducido del functor homología de la función inclusión $i_k^{p,q} : \Sigma_p \to \Sigma_q$. Note que la homología persistente es capaz de cuantificar qué tan importante es un ciclo en un espacio. Por ejemplo, un ciclo que muere para un cambio pequeño de ϵ se puede interpretar como ruido y que no representa una característica esencial del espacio.

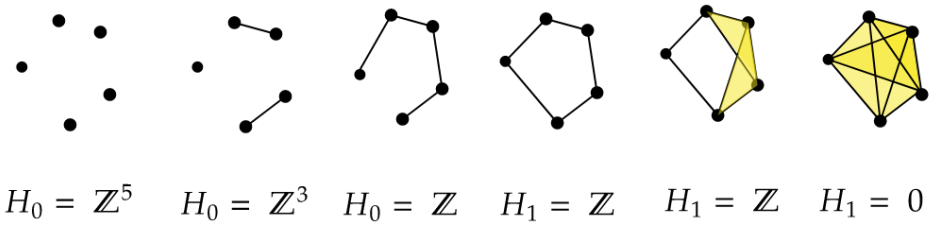


Figura 6: A medida que hacemos crecer la distancia umbral vamos obteniendo distintas topologías, se puede ver que la homología que mas persiste en dimensión uno es $\mathbb{Z}$, la homología de un circulo.

2. ¿Por qué aplicar TDA y no otro método?

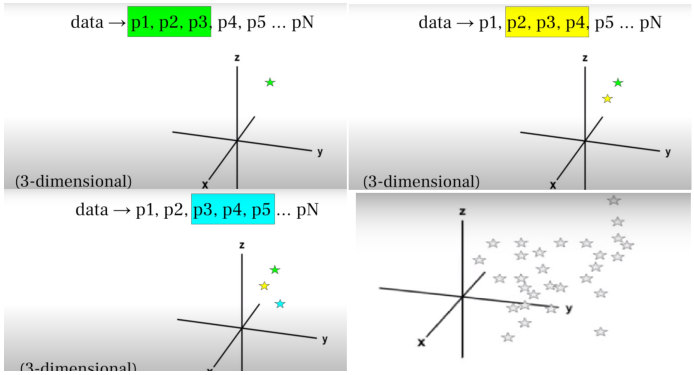
La extracción de información de conjuntos de datos de gran dimensión, incompletos y ruidosos suele ser un desafío. El análisis topológico de datos, o TDA por sus siglas en inglés, proporciona un marco general para analizar dichos datos de una manera que es insensible a la métrica particular elegida y proporciona

reducción de dimensionalidad y robustez al ruido. En adición a esto, el avance de nuevas tecnologías hace que el trato de la información sea mas complicado de manera "directa", i. e. viéndolas como lista gigantes de datos, mientras que el tratamiento con TDA es fácilmente adaptable a estos nuevos usos puesto que proporciona un manejo de la información mas implícito realizándolo a partir de su forma, junto con una robustez computacional que soporta su eficacia.

Existe más de una forma de clasificar las aplicaciones de TDA, quizás la forma más natural es por campo. Una lista muy incompleta de aplicaciones exitosas incluye la esqueletización de datos, estudio de formas, reconstrucción de gráficos, análisis de imágenes, material, análisis de progresión de la enfermedad, red de sensores, análisis de señales, red cósmica, red compleja, geometría fractal, evolución viral, propagación de contagios en redes, clasificación de bacterias mediante espectroscopia molecular, imágenes hiper espectrales en físico-química, teledetección, selección de características, y las primeras señales de advertencia de crisis financieras.

Para comprender mejor su importancia, y el esquema de su uso, veamos un ejemplo sencillo que parte del mundo financiero. Reuerde que en el contexto de las finanzas los datos suelen ser una serie unidimensional de tiempo, esto es, a cada hora se le asigna el precio de las acciones en ese instante de tiempo, tal como se puede observar en la gráfica.

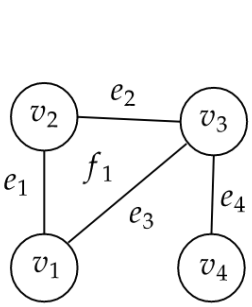
Con la intención de obtener la mayor cantidad de información de los datos queremos ïncrustarlos.[en] un espacio dimensional más alto. Para ello, lo que haremos es tomar triplas de precios de las acciones y colocarlas en $\mathbb{R}^3$ como se muestra a continuación.



El papel, y la importancia, del TDA en este caso es que al tener los datos en $\mathbb{R}^3$ podemos estudiarlos desde el punto de vista de su *forma*. Si realizamos el análisis en el conjunto de datos anterior y tomamos los primeros 5 puntos de izquierda a derecha, luego tomamos otro punto que esté más a la izquierda y dejar un par de puntos a la derecha, y así sucesivamente para movernos a lo largo de la nube de puntos, veremos la no invarianza en la forma de los datos nos alerta acerca de posible desastre financiero. Con lo anterior es posible entender como esa nueva visión acerca de los datos, obteniendo información geométrica de estos, permite una mayor eficacia en ciertas implementaciones.

3. Algoritmo

La información topológica de un espacio X obtenida por la homología persistente en una dimensión particular se puede visualizar con un gráfico de barras (barcode). Estos gráficos son una colección de intervalos, donde cada intervalo representa la clase de homología que existe en al menos un complejo de Vietoris-Rips generado por la nube de puntos. Los extremos izquierda y derecha de este intervalo representan el instante en que nace y muere una clase, respectivamente. De esta manera el eje horizontal representa la distancia umbral ϵ y el eje vertical representa un orden arbitrario de las clases capturadas. Observe que en $H_0(X)$ si $\epsilon \to 0$ entonces tendríamos tantas clases como puntos, ya que cada punto representa un $0-$simplejo aislado. y por otro lado $\epsilon \to \infty$ obtendríamos una sola clase que corresponde a un $n-$simplejo, donde n es el número de puntos de la nube de puntos. De igual manera sucede para los grupos $H_n(X)$ con $n > 0$. Los grupos $C_k(X)$ son $\mathbb{Z}-$módulos y así las transformaciones frontera δ se pueden ver como matrices. Más aún, es posible calcular los rangos de los grupos $H_n(X)$ y construir un nuevo invariante numérico denominado números de Betti. En la práctica, para poder calcular los grupos de homología, es necesario entender el operador frontera. Hallemos por ejemplo el operador frontera del siguiente complejo simplicial bidimensional.



Complejo simplicial

	v_1	v_2	v_3	v_4	e_1	e_2	e_3	e_4	f_1
v_1					1		1		
v_2					1	1			
v_3						1	1	1	
v_4								1	
e_1									1
e_2									1
e_3									1
e_4									
f_1									

Operador frontera

Figura 7: Aquí presentamos un complejo, con una cara, cuatro aristas y cuatro vértices. La homología aquí y se puede calcular estudiando los bloques de matrices superiores que aparecen en el operador frontera simplemente calculando la dimensión del kernel y la imagen para luego considerar el cociente.

Como se puede ver con solo 4 vértices, 4 aristas y una cara, el operador frontera asociado es una matriz 9×9. Por lo tanto, dar una expresión explícita de los grupos de homología no es lo más eficiente. Sin embargo, hallar los números de Betti de un complejo de Vietoris-Rips es considerablemente menos costoso en términos computacionales. De hecho, en los gráficos de barras descritos anteriormente se pueden encontrar los números de Betti trazando líneas verticales y contando el número de intersecciones con las barras verticales. En la implementación, después de la creación de los espacios filtrados de complejos simpliciales, se almacena la mayor cantidad de esta información en una matriz.

La idea principal del algoritmo consiste en usar la siguiente descomposición de matrices $DV = R$, donde V es una matriz invertible, triangular superior y de bloques diagonales (bloques que corresponden a grupos de $n-$simplejos), y R una matriz descrita de la siguiente forma:
Para A, se define $low_A(j) = max\{i : A[i,j] \neq 0\}$. De esta manera, $low_A(j)$, es el índice más abajo en la columna j que no es 0. Observe que esta matriz no está definida en columnas 0. Una matriz R decimos que está reducida si low_R es inyectiva en su dominio de definición. Si t_j es el tiempo para el cual la clase j entra en la filtración, y cada clase entra a un tiempo distinto, y las columnas de la matriz frontera están ordenadas

con estos tiempos, entonces para cada descomposición $DV = R$, el conjunto

$$PERS(X) = \{(t, t_{low_R(j)}) : low_R(j)\text{existe}\}$$

es el diagrama de persistencia de los complejos filtrados, donde j recorre todos las columnas no nulas de R. Estas parejas ordenadas definen un intervalo en la recta real. Así el gráfico de barras de X es

$$BARCODE(X) = \{[t_j, t_{low_{R(j)}}] \subseteq \mathbb{R} : low_R(j)\text{existe}\}$$

Para obtener estás factorizaciones tenemos el siguiente algoritmo denominado algoritmo columna, el cual fue extraído de [4].

```
def PHcol(D)
    R = D, V = I
    for i in 1:n
        while j<i and low_R(j}=low_R(i)
            c = R[low_R(i),i] - cR[low_R(j),j]
            R[_,i] = R[_,i]-cR[_,j]
            V[_,i] = V[_,i]-cR[_,j]
        end
    end
    S = [(low_R(i),i): R[_,i] != 0]
    return R,V,S
    #R es la matriz reducida, V es la matriz de transformación, y S la matriz de pivotes
end
```

Por ultimo tenemos un algoritmo muy similar pero que actúa sobre las filas y que también permite hallar la factorización descrita anteriormente, denominado algoritmo fila

```
def PHrow(D)
    R = D, V = I
    for i in n:1
        indices = [j: low_R(j)=i]
        p = indices[0]
        for j in indices
            c = R[low_R(i),i] - cR[low_R(j),j]
            R[_,i] = R[_,i]-cR[_,j]
            V[_,i] = V[_,i]-cR[_,j]
        end
    end
    S = [(low_R(i),i): R[_,i] != 0]
    return R,V,S
    #R es la matriz reducida, V es la matriz de transformación, y S la matriz de pivotes
end
```

Eirene usa múltiples herramientas de optimización combinatoria, teoría de Morse discreta y otras librerías como PERS la cual le permite reducir el tamaño del operador frontera en varios ordenes de magnitud. Existe hace uso de teorías del álgebra homológica y una reciente teoría de cohomología persistente para reducir el tamaño de los bloques diagonales de la matriz V. La ventaja de Eirene sobre otras librerías que construyen estos diagramas de barras es que por los general otros algoritmos no permiten guardar y recuperar la información en los primeros estados de la homología persistente, mientras que las optimizaciones efectuadas en Eirene, permiten reconstruir gran parte de la homología en estados antiguos y hacerlo de manera óptima. En resumen, Eirene calcula los complejos de Vietoris-Rips, captura los cambios en homología en las distintas filtraciones del espacio y almacena los números de Betti cuando se recorren los distintos complejos simpliciales. Como solo nos interesa la información del invariante de los números de Betti podemos darnos el lujo de perder información en el operador frontera, siempre que el operador mantenga el mismo rango,

para lograr eso se busca una factorización particular que se puede obtener gracias a los dos algoritmos descritos anteriormente. Finalmente, el último trabajo realizado por la librería Eirene consiste en optimizar el algoritmo. Para finalizar está parte, veamos como es el diagrama de barras en homología persistente para una colección pequeña de puntos, la siguiente imagen fue extraída del articulo [4]

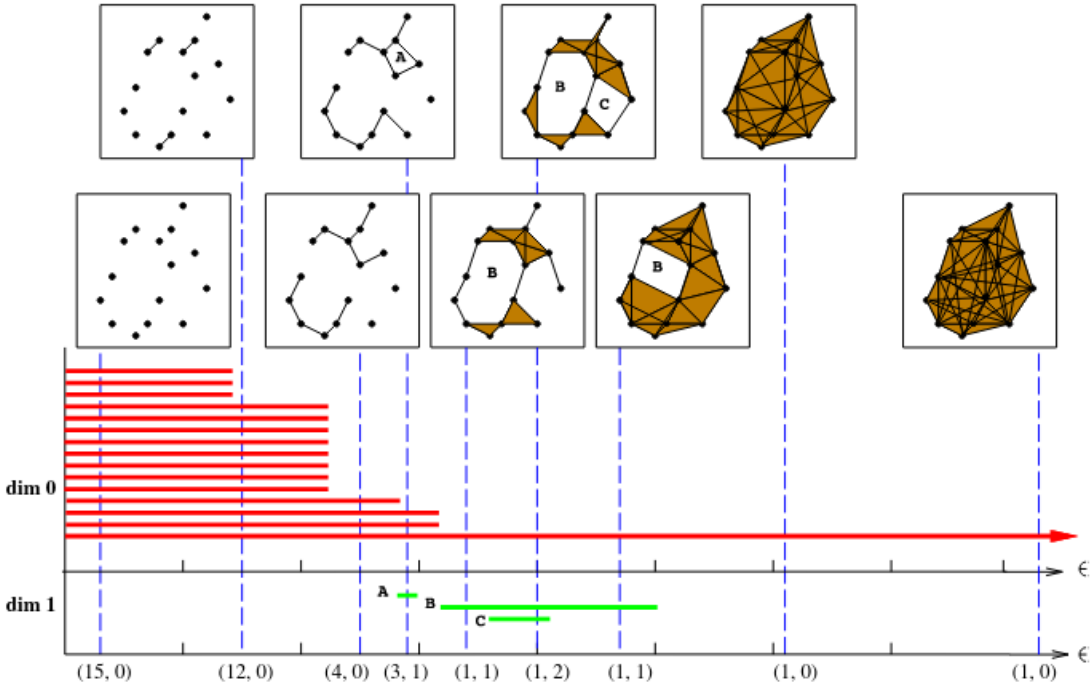


Figura 8: En este caso, se busca entender las componentes conexas y los huecos, y calcular los números de Betti de dimensión 1 y 2.

4. Ejemplos

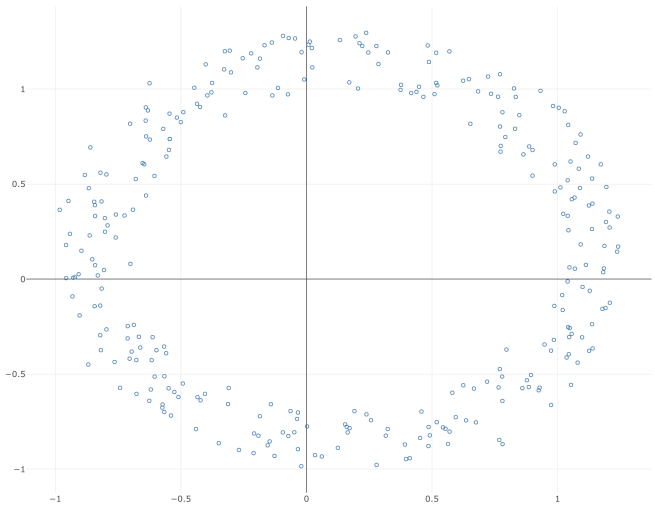
Analicemos un ejemplo de implementación donde sea posible evidenciar cada uno de los comandos que nos permiten hacer cálculos de persistencia homológica. Para este ejemplo se hace uso del paquete *DelimitedFiles* el cual permite leer una matriz desde el archivo de fuente donde cada línea (teniendo en cuenta su finalización) ofrece la información de una fila, con elementos separados por un delimitador dado. Este paquete permite leer archivos de texto o arrays de bytes. En el caso de nuestro ejemplo hacemos uso del método readdlm donde el delimitador se toma como \n. Dado que todos los datos son numéricos el resultado de esto será un array.

```
pointcloud = readdlm(filepath, ',')
```

Una vez tengamos la información de este array podemos graficar la información mediante la función propia de Eirene `ezplot_pjs`. Esta función se soporta en el paquete PlotlyJS.

```
ezplot_pjs(pointcloud)
```

El cálculo de la persistencia homológica del conjunto de datos se realiza mediante la función `eirene(point, model =` la cual tiene parámetros `point`, que recibe el conjunto de puntos, y `model =` la cual recibe el formato de datos que lleva el conjunto de puntos. Los diferentes formatos que recibe son `pc` que identifica a las nubes de puntos (point cloud), `vr` que identifica a los complejos de Vietoris-Rips, y `complex`. Todo lo anterior lo almacenamos en la variable C. Con esta información procedemos a graficar el barcode y el diagrama de persistencia de nuestros datos.

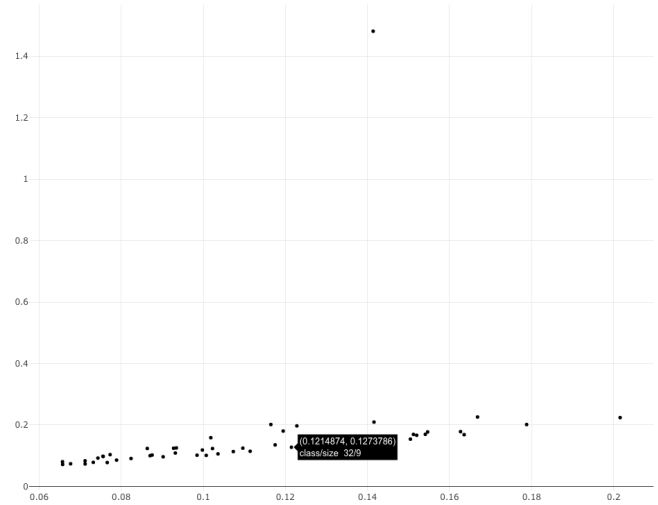



```
C = eirene(pointcloud, model = "pc")
plotbarcode_pjs(C, dim  =1)
plotpersistencediagram_pjs(C, dim = 1)
```

Existe una comando para calcular específicamente el barcode del conjunto de datos.

```
A = barcode(C, dim = k)
```

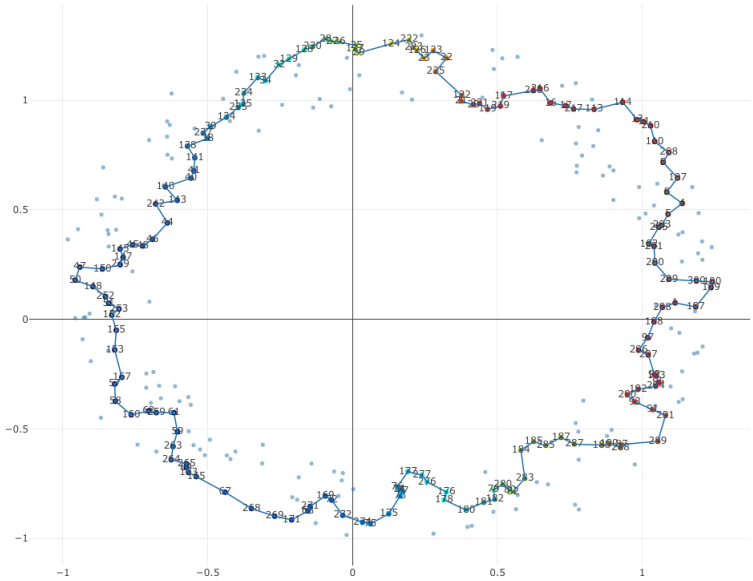
Esta función recibe como parámetros el cálculo de la persistencia homológica (que se realiza mediante la función `eirene()`), y la dimensión. Tal como se mencionó en secciones pasadas tiene como output un conjunto de líneas o un conjunto de puntos en el plano dependiendo de la dimensión de las clases de homología. La variable `A` será una matriz de nx2 donde n es el número de clases de homología.



Observemos que estas funciones nos permiten obtener más información que la ofrecida por un plot o un scatter. Si colocamos el mouse sobre cualquier de los puntos obtendremos la información acerca de los tiempos de vida y muerte, el número de la clase en cuestión y el número de células que soportan a dicho representante. Recordemos que, en términos generales, un complejo celular es un objeto construido "pegando"bloques de construcciones simples, llamados células. Lo importante es que cada célula tenga una dimensión. Un complejo simplicial es un complejo celular construido a partir de vértices (celdas de dimensión 0), aristas (células de

dimensión 1), triángulos (células de dimensión 2), tetraedros (células de dimensión 3), etc. Un complejo cúbico se construye a partir de vértices (celdas de dimensión 0), aristas (células de dimensión 1), cuadrados (células de dimensión 2), cubos (células de dimensión 3), etc. Las caras de una célula son las células de dimensión inferior que la "tocan". Por ejemplo, las caras de una arista en un grafo simple son los dos vértices que tocan al vértice. Las caras de un triángulo A en un complejo simplicial son las aristas y vértices que tocan A.

Si es necesario conocer más información acerca de un representante específico en un diagrama de persistencia de dimensión 1 basta con usar la función propia de Eirene `plotclassrep_pjs` tal y como sigue



Esta imagen nos permite ver en detalle la información de la clase 50, que nace en tiempo `t = 0.141` y muere en tiempo `t = 1.48`. El ciclo representativo de esta clase cuenta con 153 células.

Al igual que con los barcodes, existe una función que nos permite calcular explícitamente los representativos de las clases. Esto se realiza mediante el siguiente comando

```
S = classrep(C, class = p, dim = k)
```

Esta función recibe como parámetros el conjunto de datos `C`, el número de la clase `class =` y la dimensión `dim =` .

Así como con la función `barcode()` todo lo anterior se presenta como matrices, las cuales a su vez pueden ser graficadas mediante las funciones del paquete `Plot`. En caso de graficar usando este paquete se pierde la información específica que nos ofrecen las funciones propias del paquete Eirene.

Ahora haremos la misma implementación para un conjunto de datos más grande. Para ello, corremos el siguiente código, el cual lee y grafica un archivo predeterminado de Eirene que contiene los datos de un toro.

```
filepath = eirenefilepath("noisytorus")

pointcloud = readdlm(filepath,',')

ezplot_pjs(pointcloud)
```

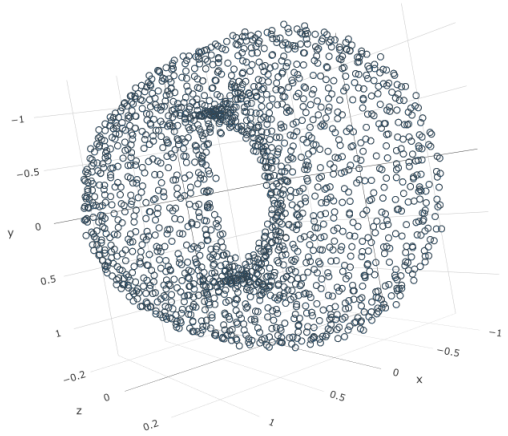


Figura 9: Nube de puntos de un toro

Usando el comando `size(pointcloud)`, podemos ver que este toro 1800 puntos, lo cual resulta ser un número muy grande. Por lo tanto, vamos a trabajar con un radio de corte:

```
C = eirene(pointcloud,maxdim=2,maxrad=0.3,model="pc")
```

Implementamos el siguiente código para obtener los mismos datos del ejemplo anterior:

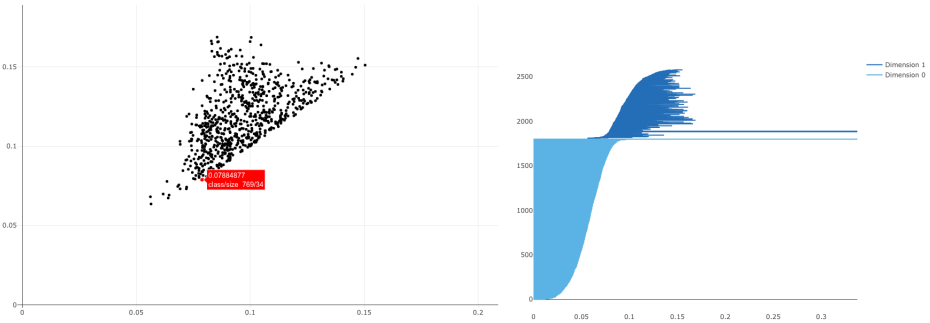
```
plotpersistencediagram_pjs(C,dim=1)

plotbarcode_pjs(C,dim=0:1)

plotclassrep_pjs(C,class=768,dim=1)

plotclassrep_pjs(C,class=769,dim=1)
```

Las siguientes gráficas muestran los resultados obtenidos:



Los puntos de la diagonal del barcode representan las clases que "nunca mueren".

4.1. Considereaciones acerca de la implementación de los ejemplos

La implementación de los anteriores ejemplos se encuentra en un notebook adjunto. No obstante, en el desarrollo de la implementación encontramos una posible serie de errores propios de la instalación de Jupyter y Anaconda, algo inherente a cada máquina donde se quiera realizar la implementación. Dichos errores conciernen al paquete WebIO, que es el enlace entre el cuaderno y la red; paquete que a su vez soporta al paquete PlotlyJS. Para remediar este error se debe instalar WebIO directamente desde Python o Anaconda y construirlo con el comando `build WebIO` desde la consola de Julia. De persistir el error es recomendable revisar la siguiente documentación, y discusiones en los foros de Julia que se encuentran en [7], [8]. Un *bug* que es posible se presente en el cuaderno de los ejemplos es que los outputs no muestren nada. Esto sucede por las razones previamente expuestas y es recomendable revisar la instalación de los paquetes y leer las referencias propuestas. Para evitar la falta de visualización hemos añadido todos los resultados del notebook en el presente documento.

Otro posible error que puede ocurrir durante la implementación del notebook concierne a los registros donde se almacenan los paquetes. Para resolver este error recomendamos visitar [9] y [10].

```
┌ Warning: could not download https://pkg.julialang.org/registries
│   exception = HTTP/1.1 301 SA internal redirect trigger (Failed to connect to sa.pkg.julialang.org port 443 after 3231
 ms: Connection refused) while requesting https://pkg.julialang.org/registries
└ @ Pkg.Registry C:\workdir\usr\share\julia\stdlib\v1.8\Pkg\src\Registry\Registry.jl:68
```

Es recomendable antes de implementar tener previamente instalados los paquetes Distances, JLD, Blink, PlotlyJS, Plotly, MultivariateStats, Colors, Dates, CSV; de preferencia instalarlos directamente desde la consola de Julia, habiendo previamente actualizado el Pkg.

Referencias

[1] Fugacci U., Scaramuccia S., Iuricich F., De Floriani L., *Persistent homology: a step-by-step introduction for newcomers*, 2016.

[2] Hatcher A., *Algebraic Topology*, 2001.

[3] Chazal F., Michel B., *An introduction to Topological Data Analysis: fundamental and practical aspects for data scientists*, 2021.

[4] Hylton A., Henselman-Petrusek G., Sang J., Short R., *Tuning the Performance of a Computational Persistent Homology Package*, Softw Pract Exp. 2019;49(5):885-905.

[5] Henselman, G., Ghrist, R. *Matroid Filtrations and Computational Persistent Homology*

[6] Eirene julia package. `https://github.com/Eetion/Eirene.jl`.

[7] `https://juliagizmos.github.io/WebIO.jl/v0.8/troubleshooting/not-detected/`

[8] `https://juliagizmos.github.io/WebIO.jl/latest/providers/ijulia/`

[9] `https://github.com/JuliaLang/Pkg.jl/issues/2742`

[10] `https://discourse.julialang.org/t/juliapro1-5-0-could-not-download-https-pkg-juliahub-com/registries/44811`