Persistent and Zigzag Homology
A Matrix Factorization Viewpoint

Gunnar Carlsson[1], Anjan Dwaraknath[2], and Bradley J. Nelson[2]

[1] *Department of Mathematics, Stanford University*
[2] *Institute for Computational and Mathematical Engineering, Stanford University*

Abstract

Over the past two decades, topological data analysis has emerged as a young field of applied mathematics with new applications and algorithmic developments appearing rapidly. Two fundamental computations in this field are persistent homology and zigzag homology. In this paper, we show how these computations in the most general case reduce to finding a canonical form of a matrix associated with a type-A quiver representation, which in turn can be computed using factorizations of associated matrices. We show how to use arbitrary induced maps on homology for computation, providing a framework that goes beyond the capabilities of existing software for topological data analysis. Furthermore, this framework offers multiple opportunities for parallelization which have not been previously explored.

1 Introduction

Persistent and zigzag homology both track how topological spaces change over a single parameter, extending the use of homology in algebraic topology to a variety of applied settings. Persistent homology is often motivated by tracking how sublevel sets of a Morse function map through inclusions, or through geometric constructions on point cloud data that connect points at varying distances [5], and is supported by a variety of software packages [1–3, 21, 31, 37]. Zigzag homology has been more limited in use and does not have widespread software support (notably a version is implemented in [31] for a special case), although there are several compelling applications for zigzag homology that have been proposed. Perhaps the simplest is in tracking homology through level sets of Morse functions [7]. Zigzag homology has also found use in reducing the cost of computing persistent homology for geometric complexes [11, 34]. Several inference applications have been proposed in [6] including tracking homology through several choices of density estimator; bootstrapping results from several samples; and testing stability of features in the witness complex computed using several landmark sets.

In this paper, we present an algorithm that ties together the computations of persistent and zigzag homology, and more generally computations to find the indecomposables of any finite type-A quiver representation. The computational framework enables the computation of arbitrary induced maps on homology. This capability is critical in both computations and applications performed "by hand", since in that context direct chain calculations are precluded by the fact that the chains either form an infinite dimensional vector space (in the case of singular homology) or one whose dimension is too large (general simplicial complexes). One important example of this process is the computation of the homology of a space X from a finite covering $\mathcal{U} = \{U_\alpha\}_{\alpha \in A}$. This method proceeds by computing the homology of all possible subsets $U_{\alpha_1} \cap U_{\alpha_2} \cap \cdots \cap U_{\alpha_k}$ (which can be carried out in parallel), and then using the maps induced by all possible inclusions among them to reassemble the homology of the original space. The second stage of the computation of the homology now involves much lower dimensional vector spaces (homology) rather than chains, and the first stage is efficient because it involves much smaller and simpler calculations assuming the cover is chosen in a useful way, and can be carried out in parallel.

This framework offers multiple opportunities for parallelization, resulting in computational advantages even for some situations where existing algorithms may be used. In contrast, existing software for persistent and zigzag homology primarily operates on chain complexes, mostly deals with maps induced by inclusion, and is not as amenable to parallelization. Once we obtain a quiver representation from induced maps on homology, our algorithm is based on computing a canonical form of a matrix, and uses extensive use of factorizations of submatrices which show that computing persistent and zigzag homology are intimately related. We hope this framework will expand the application of persistent and zigzag homology to new situations, and will open up the potential for further algorithmic improvements through the application of techniques from numerical linear algebra. We have released an open-source C++ implementation of our algorithm, available at https://github.com/bnels/BATS.

1.1 Linear Algebra and Topology

In the second half of the 20th century, developments in linear algebra split into multiple trajectories. One path focused on matrix factorizations, championed by Householder [22], and has become

fundamental to numerical computations in science and engineering. This flavor of linear algebra has had great impact, was called one of the top 10 algorithms of the 20th century by the Society for Industrial and Applied Mathematics [9] and has become a core part of applied and computational mathematics curricula. Another trajectory followed the development of homological algebra, driven by advancements in category theory and topology [15,16,27]. Homological algebra is comprised of a collection of techniques which have demonstrated great utility within topology, and do not appear explicitly in the matrix factorization literature. In this paper, we demonstrate how a number of problems arising in applied topology can be dealt with efficiently by converting them to problems more easily amenable to standard factorization techniques.

The birth of modern algebraic topology was a driving force behind the development of homological algebra and category theory. One important construction was a *functor*, called *homology*, from the category of topological spaces, Top, to the category of modules over a ring, $R\mathsf{Mod}$. For the purposes of this paper, we will assume that the ring is a field $\mathbb{F}$, which means that instead of modules over a ring we will be working with vector spaces in the category $\mathsf{Vect}_{\mathbb{F}}$. Concretely, homology associates each topological space $\mathcal{X}$ with vector spaces $H_k(\mathcal{X})$, $k = 0, 1, 2, \ldots$, and associates each map between topological spaces $f : \mathcal{X} \to \mathcal{Y}$, to a linear transformations $H_k(f) : H_k(\mathcal{X}) \to H_k(\mathcal{Y})$, $k = 0, 1, 2, \ldots$. The index k in H_k ranges over the dimension of topological features (not to be confused with the dimension of the vector space), for example connected components are associated with H_0, and loops are associated with H_1. Another way to write this is to say that the following diagram commutes:

$$\begin{array}{ccc} \mathcal{X} & \xrightarrow{\quad f \quad} & \mathcal{Y} \\ \downarrow & & \downarrow \\ H_k(\mathcal{X}) & \xrightarrow{H_k(f)} & H_k(\mathcal{Y}) \end{array}$$

The power of homology is that statements about topological spaces and maps have associated statements about vector spaces and linear transformations, which are sometimes more tractable (and computable).

There is an intermediate step in the construction of the homology functor which is very important for calculation, which involves the construction of chain complexes and chain maps in the category $\mathsf{Chain}_{\mathbb{F}}$. Chain complexes are sequences of (possibly infinite-dimensional) vector spaces C_k connected by boundary maps $\partial_k : C_k \to C_{k-1}$ with the property that $\partial_k \partial_{k+1} = 0$ (again k relates to topological dimension), and chain maps are linear transformations $F_k : C_k \to D_k$ that commute with the boundary maps. Chain complexes and maps can be explicitly obtained from combinatorial representations of spaces $\mathcal{X}$ and maps $f : \mathcal{X} \to \mathcal{Y}$, and the final step to compute homology is simply to compute quotient vector spaces $H_k = \ker \partial_k / \operatorname{img} \partial_{k+1}$.

Homology is a *homotopy invariant*, meaning that if $f, g : \mathcal{X} \to \mathcal{Y}$ are homotopic their induced maps $H_k(f)$ and $H_k(g)$ are identical for all k. This implies that if $\mathcal{X}$ and $\mathcal{Y}$ are deformations of each other, $H_k(\mathcal{X}) \cong H_k(\mathcal{Y})$ for all k. This property is explicitly related to the fact that H_k is a quotient space of C_k.

1.2 Topological Data Analysis

Topological data analysis (TDA) uses algebraic topology in order to produce topological descriptors of data. It can be applied to a single datum (for example a molecule in a database, or an image), or an entire set of sampled data viewed as a point cloud [5]. As a field, topological data analysis has historically been motivated by various filtrations, which track how a topological space changes as it grows. One example arises naturally when trying to describe the topology of a space from samples X. One reasonable approach would be to place balls of radius r at each sample, and use the resulting space $\mathcal{X}_r = \bigcup_{x_i \in \mathsf{X}} B_r(x_i)$ as a representative of the space the data was sampled from. If r can be appropriately determined, the space may indeed be topologically equivalent to the true space. However, if r is chosen to be too small there may be gaps or holes that do not really exist, or if r is chosen to be too large, real gaps or holes may be filled in. A solution is to consider all values of r, and to track how the space changes as r increases. This produces a *filtration* $\mathcal{X}_r$ known as the Čech filtration, where the subscript is understood to run over the index set for the radius $r \in [0, \infty)$. Passing to homology we have

$$\begin{array}{ccccccc} \mathcal{X}_0 & \longrightarrow & \cdots & \longrightarrow & \mathcal{X}_r & \longrightarrow & \cdots \\ \downarrow & & & & \downarrow & & \\ H_k(\mathcal{X}_0) & \longrightarrow & \cdots & \longrightarrow & H_k(\mathcal{X}_r) & \longrightarrow & \cdots \end{array}$$

Where the horizontal maps are inclusions on the top row, and the maps induced by inclusions on the bottom row. *Persistent homology* [14,38] tracks the survival of elements of $H_k(\mathcal{X}_r)$ through the induced maps. The *persistence barcode* is a multiset of pairs $\{(b_i, d_i)\}$ that encodes all the algebraic information in the above diagram – a birth b_i at parameter r if $H_k(\mathcal{X}_r)$ has an element that is not in the image of an inclusion, and a death d_i at the parameter at which that element enters the kernel of the inclusion (if this never happens, then we say $d_i = \infty$).

While persistent homology considers maps that go in a single direction, a related construction *zigzag homology* [6] considers maps that can alternate directions.

$$\begin{array}{ccccccccc} \mathcal{X}_0 & \longrightarrow & \mathcal{X}_1 & \longleftarrow & \mathcal{X}_2 & \longrightarrow & \cdots & \longleftarrow & \mathcal{X}_N \\ \downarrow & & \downarrow & & \downarrow & & & & \downarrow \\ H_k(\mathcal{X}_0) & \longrightarrow & H_k(\mathcal{X}_1) & \longleftarrow & H_k(\mathcal{X}_2) & \longrightarrow & \cdots & \longleftarrow & H_k(\mathcal{X}_N) \end{array}$$

One motivating example is tracking how a space changes over a series of intersections of open sets $\mathcal{X}_0 \leftarrow \mathcal{X}_0 \cap \mathcal{X}_1 \rightarrow \mathcal{X}_1 \leftarrow \mathcal{X}_1 \cap \mathcal{X}_2 \rightarrow \cdots$. Similarly to persistent homology, homology can be tracked through the maps, and there is a *zigzag barcode* consisting of a multiset of pairs $\{(b_i, d_i)\}$ where b_i is the parameter of the first appearance of a homology generator, and d_i is the last parameter for which that generator can be linked through the diagram.

So far, we have introduced persistent homology and zigzag homology in a way that may show some superficial resemblance, but it may not be clear that there is anything beyond that. In fact, there is a deep connection between the two constructions that is not immediately apparent, but that has deep theoretical and algorithmic implications, which is that both persistent homology and zigzag homology produce examples of type-A quiver representations [6].

1.3 Quiver Representations and Matrix Factorizations

First we must revisit our treatments of persistent and zigzag homology. Recall that maps between topological spaces $f : \mathcal{X} \to \mathcal{Y}$ induce maps on homology $F_* : H_*(\mathcal{X}) \to H_*(\mathcal{Y})$. Previously, we focused on the specific case where all maps were inclusions, which is the case in filtrations. Most generally, we will consider a diagram of spaces and maps

$$X_0 \overset{f^0}{\longleftrightarrow} X_1 \overset{f^1}{\longleftrightarrow} X_2 \overset{f^2}{\longleftrightarrow} \ldots \overset{f^{N-1}}{\longleftrightarrow} X_n$$

where the bi-directional arrows can be given either direction, and the superscripts on f denote an index (not a power or composition). For persistent homology, all arrows will point in a single direction, and for zigzag homology the direction will alternate, and we consider the most general case where any sequence of directions can be given. If we now look at induced maps on homology in dimension k we obtain the following diagram of vector spaces

$$H_k(\mathcal{X}_0) \overset{F_k^0}{\longleftrightarrow} H_k(\mathcal{X}_1) \overset{F_k^1}{\longleftrightarrow} H_k(\mathcal{X}_2) \overset{F_k^2}{\longleftrightarrow} \ldots \overset{F_k^{N-1}}{\longleftrightarrow} H_k(\mathcal{X}_n)$$

This diagram of vector spaces is an example of a quiver representation of type A_n (n vector spaces and $n-1$ linear transformations arranged in a line graph).

As we mentioned in Section 1.2, persistent homology and zigzag homology both track the images and kernels of maps throughout this diagram. An important practical and theoretical question is whether or not persistence and zigzag barcodes depend on choices of bases for vector spaces in the above diagram. A theorem due to Gabriel [17] assures that this is not the case, by showing that type-A quiver representations have a finite indecomposable representation (although the proof is not constructive). In fact, bars in the barcodes of topological data analysis are exactly the indecomposables of the associated quiver representation, called interval indecomposbales, and so are basis independent.

We will focus on classification of A-type quiver representations by putting an associated companion matrix into a canonical form. We consider a matrix A of the quiver, which has a block structure of the directed adjacency matrix of the underlying graph, and each non-zero block contains the map of vector spaces along each edge (induced maps on homology). The goal is to find a factorization of the matrix

$$A = B\Lambda B^{-1} \tag{1}$$

In which B is an invertible block diagonal matrix, and Λ has the same block structure as A, but blocks are put in a canonical form which is unique up to permutation. Once we have a factorization $A = B\Lambda B^{-1}$, it is easy to read off the persistence or zigzag barcode from the matrix Λ, which we will cover in Section 3.

This is a powerful conceptual viewpoint. First, it shows that the barcodes used and studied in persistent and zigzag homology are as fundamental as matrix decompositions such as the Jordan form. Second, it lays the groundwork for a new set of algorithms for topological data analysis.

1.4 Contributions and Related Work

Our main contributions are

1. A trivially parallelizable methodology to compute quiver representations from quiver diagrams of spaces.

2. Sequential and parallel algorithms to extract interval indecomposables (barcodes) from any finite type-A quiver representation over any field.

3. A computational framework that allows for the computation of induced maps on homology for arbitrary maps of chain complexes.

While we provide concrete algorithms for all steps, we also operate at a high level of abstraction, which will enable many optimizations to be applied in future work.

The study of persistent and zigzag homology as examples of type-A quiver representations was begun in [6], which is the theoretical basis for our algorithm. This has led a variety of interesting theoretical and applied work as surveyed in [33], but algorithmic implementations so far have not significantly leveraged this connection.

An algorithm for computing persistent homology was first described in [14] (for $\mathbb{F}_2$ coefficients) and was extended to general fields in [38], and algorithm for computing zigzag homology was first described in [7]. These approaches operate on inclusion maps between spaces, and computations work directly on chain complex boundary matrices. While both persistent and zigzag homology are known to be computable in matrix-multiplication time [6], sparsity considerations are typically much more important in practice, and in order to compute on large data sets, several approaches have been pursued. First, there have been efforts to speed up computation of persistence through high performance implementations that employ various optimizations [1–3, 21, 31, 37]. Second, there have been efforts to reduce the inherent size of computations using methods that preserve the homotopy type of a space while reducing the size of its combinatorial representation [4, 12, 30, 36]. Zigzag homology has received less attention than persistence, but similar efforts can be found in [28, 29]. The use of non-inclusion maps in persistent and zigzag homology has been somewhat limited in topological data analysis, although the case of simplicial maps has been investigated in [11, 23], based on a strategy that uses zigzag homology to compute a persistence barcode.

Our approach has several notable differences compared to existing computational approaches for persistent and zigzag homology. First, we consider a two step approach, where first induced maps on homology are computed to form a quiver representation, and then our algorithm extracts the barcode. In contrast, existing approaches work almost exclusively on the level of chain complexes, missing out on the abstraction and compression that induced maps on homology afford. We believe that the two approaches are complementary, and that many existing optimizations could be applied when computing homology and induced maps. Second, our approach works for general cell complexes, and general cell maps, whereas some existing approaches are focused on the simplicial (or cubical) complexes and simplicial maps. We also work with arbitrary field coefficients, when some existing approaches for induced maps are limited to $\mathbb{F}_2$ coefficients [11]. Third, our approach offers multiple opportunities for parallelization, whereas most existing approaches are sequential in nature. Computing induced maps on homology is trivially parallelizable, and our quiver algorithm also admits a parallelization scheme. Notably, [4] observes that a scheme to simplify complexes is trivially parallelizable for reasons similar to the trivial parallelization of induced maps, and [7] describes a parallelization scheme that has similarities to our quiver scheme, but operates on chain complexes and not induced maps on homology. Furthermore, our parallelization scheme is different and complementary to existing efforts to use spectral sequences to parallelize homology calculations [25]. Finally, we unify computations for persistent and zigzag homology to a degree that is not seen in existing algorithms, in the sense that modifications to handle arrows of different directions are trivial.

Our approach is closest in spirit to the original paper on zigzag homology [6], which started with induced maps on homology, and gave a constructive algebraic algorithm for the interval decomposables for A-type quivers. In this paper, we address the computational questions that are necessary for computing arbitrary induced maps on homology, and an explicit algorithm for computing interval indecomposables. In contrast to [6] and existing algorithms for zigzag homology [28, 29, 31], our algorithm does not explicitly use right filtrations, and instead use an approach that involves producing a matrix factorization.

As a companion to this work, we have released a new open-source package for the algorithms we describe. Inspired by the basic linear algebra subprograms (BLAS) [24], we call it the basic applied topology subprograms (BATS). The library includes high-level C++ templates for the algorithms described, as well as compatible matrix implementations which are templated over a choice of field. The code is publicly available at `https://github.com/bnels/BATS`.

1.5 Outline

The rest of the paper is organized as follows: We will review background necessary for computing homology and induced maps in Section 2, along with relevant background on computing persistent and zigzag homology. In Section 3 we will give a more formal treatment of quiver classification and its connection to topological data analysis. In Section 4 we will describe our sequential and parallel algorithms for computing indecomposables of A-type quiver representations.

2 Preliminaries

We will now introduce the algebraic and topological tools necessary for our algorithm. Specifically, we will need to compute homology of cell complexes, and compute induced maps on homology, which we will connect to existing methods for computing persistent homology. Finally, we will present an approach for using existing persistence algorithms for non-inclusion based maps using mapping cylinders.

While this section is reasonably self-contained for the purposes of setting up our algorithm and establishing notation, this material is fairly standard and can be found in a variety of locations. For a general reference for matrix computations, we defer to Golub and Van Loan's text [19]. For concepts in algebraic topology, we refer to Hatcher [20]. Early sources for computing persistent homology and zigzag homology are [14,38] and [6,7] respectively. Further background on computational topology can be found in [13] and for an overview of modern TDA software we recommend [32].

2.1 Algebra and Notation

Notation: Our algorithm uses matrix factorizations to organize computation, and so we shall use Householder notation for linear algebra [19,22]. Upper case Greek or Roman letters such as A or Λ will refer to matrices, lower case Roman letters such as a will refer to vectors, and lower case Greek letters such as λ will refer to scalars. This is not always consistent with notation found in algebraic topology or the TDA literature, which does not have conventions as strong as Householder notation, but we will attempt to use notation that is close to modern use. One special symbol, ∂, will always refer to a boundary matrix. Topological spaces will be denoted with calligraphic font as in $\mathcal{X}$.

We will prefer to work with matrix factorizations when possible. Due to the abundance of subscripts in other contexts, we will use square brackets instead of subscripts for indexing. Vectors will be assumed to be column vectors, and v^T will denote the corresponding row vector. e_i will refer to the vector with 1 in the i-th entry and 0 elsewhere, with dimension determined by context. When we need to access elements in vectors, we will use the notation $x[i]$ to denote the scalar value $e_i^T x$, or the i-th entry in x. When we need to access elements of matrices, we will use the notation $A[i,j]$ to denote the scalar $e_i^T A e_j$, or the entry in the i-th row and j-th column of A. When we want to indicate columns of matrices, we will use the notation $A[j]$ to denote the vector Ae_j, or the j-th column of A. As is standard in numerical linear algebra, indexes will begin with 1, meaning the valid range of indexes for a vector $x \in \mathbb{F}^n$ is $1, \ldots, n$. Asterisks indicate that a subscript or superscript runs over a range of values, determined by context. For example, F_* is often used to represent $F_k, k = 0, 1, \ldots$.

Computations will be done in vector spaces over a fixed field $\mathbb{F}$. Computations in topological data analysis are typically done over finite fields, for example $\mathbb{F} = \mathbb{F}_2 = \mathbb{Z}/2\mathbb{Z}$, or the rationals $\mathbb{Q}$, because homology requires exact computation of kernels and images. Floating point arithmetic is typically avoided due to numerical issues. One exception is Hodge theory, which uses the more familiar fields $\mathbb{R}$ or $\mathbb{C}$ – for a numerical and application focused introduction see [26].

Rank-nullity theorem: Let $A : V \to W$, where V, W are finite-dimensional vector spaces. Then $\operatorname{rank} A + \dim \ker A = \dim V$.

Change of Basis : Suppose that we have vector spaces V and W with bases $B^V = \{b_i^V\}$ and $B^W = \{b_i^W\}$ respectively and a linear transformation T represented by a matrix A in bases B^V and B^W. That is the coefficients $A[j] = Tb_j^V$. Now, suppose that we have new bases $C^V = \{c_i^V\}$ and $C^W = \{c_i^W\}$, where $c_i^* = U^* b_i^*$. Then if we wish to write T in terms of bases C^V and C^W, we can write a new matrix

$$\hat{A} = (U^W)^{-1} A U^V \tag{2}$$

The matrix U^V first maps coefficients in C^V to coefficients in B^V, then the linear transformation T is applied, mapping to coefficients in B^W, which are then mapped to coefficients in C^W via $(U^W)^{-1}$.

Quotient Vector Spaces: Homology is an example of a quotient vector space. Simply, if V is a vector space, and $W \subseteq V$ is a subspace, then the quotient space V/W has elements (cosets) $[v]$, where if $v \in [v]$, then $v + w \in [v]$ for any $w \in W$. The set V/W is also endowed with a vector space structure, so is called a quotient vector space. The quotient operation is a linear transformation $V \to V/W$, meaning

$$[\alpha x + \beta y] = \alpha[x] + \beta[y] \tag{3}$$

We will often represent elements $[v] \in V/W$ using a *representative* $v \in [v]$. The choice of representative is not unique, as $v + w$ for any $w \in W$ may also be used as a representative.

Four Fundamental Subspaces: Let $T : V \to W$. There are four fundamental subspaces associated with T: the kernel,

$$\ker T \subseteq V = \{v \in V \mid T(v) = 0\}$$

the image

$$\operatorname{img} T \subseteq W = \{w \in W \mid w = T(v), v \in V\}$$

the cokernel

$$\operatorname{coker} T = W / \operatorname{img} T$$

and the coimage

$$\operatorname{coimg} T = V / \ker T.$$

The cokernel and coimage both are quotient vector spaces. Note that Tv is identical for all $v \in [v] \in \operatorname{coimg} T$.

Triangular Matrices: Triangular matrices only contain non-zero elements on one side of a diagonal. Common triangular forms are lower and upper triangular matrices, where $L \in \mathbb{F}^{m\times n}$ is lower-triangular if $L_{i,j} = 0$ if $j > i$, and $U \in \mathbb{F}^{m\times n}$ is upper triangular if $U_{i,j} = 0$ if $j < i$. Triangular matrices are well extremely useful because solving linear systems using backward substitution [19] is as expensive as regular matrix multiplication. That is, solving $Tx = b$ for unknown x is as expensive as forming the matrix-vector product Tx. Backward substitution can be used successfully even when a triangular matrix is not invertible, either because it is not square or because it is rank deficient, as long as b is in the column space of T (i.e. the linear system is consistent).

The J Matrix:

Definition 2.1.1. *J is a square $n \times n$ matrix, such that*

$$J_{ij} = \begin{cases} 1, & \text{if } i = n - j - 1 \\ 0, & \text{otherwise} \end{cases}$$

In other words, it is the anti-diagonal permutation matrix. Specifically when multiplied on the left, it reverses the row order. Similarly it reverses the column order when multiplied on the right. It is its own inverse

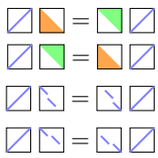
$$J^{-1} = J$$

A common operation is to conjugate with the J matrix, which reverses both row and column order and thus produces a reflection across the anti-diagonal. Note that this operation is distinct from taking the transpose of a matrix and cannot be expressed in terms of it.

$$JAJ^{-1} = JAJ = A'$$

The following are useful commutation relations between the J matrix and other matrix shapes

$$JL = UJ \tag{4}$$
$$JU = LJ \tag{5}$$
$$JE_L = \hat{E}_U J \tag{6}$$
$$JE_U = \hat{E}_L J \tag{7}$$



Pivot Matrices:

Definition 2.1.2. *A pivot matrix is a matrix in which every row and column has at most one non-zero element.*

In the context of matrix factorizations used in this paper, the non-zero element will always be 1 (the multiplicative identity of the field $\mathbb{F}$) by convention. The term pivot matrix refers to its use in recording pivots (last nonzeros of rows or columns) when computing matrix factorizations. Pivot matrices are similar to permutation matrices in the sense that they map a single basis element to a single basis element, but contain the possibility that some rows and columns can be entirely zero. Thus they are not generally invertible.

Lemma 2.1.3. *The class of pivot matrices is closed under multiplication*

Proof. For a pivot matrix Q, define $i(j)$ to be the index of the non-zero row of column j if column j has a pivot, and $i(j) = \infty$ otherwise. Additionally, define $i(\infty) = \infty$. Thus, we can write columns of Q as

$$Q[j] = e_{i(j)}$$

where $e_\infty = 0$.

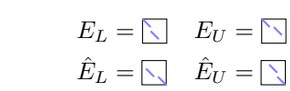
Let Q_1 and Q_2 be pivot matrices with compatible dimensions to form the product $A = Q_2Q_1$, then column j of A can be written as

$$Ae_j = Q_2(Q_1e_j) = Q_2e_{i_1(j)} = e_{i_2(i_1(j))}$$

So column $A[j]$ has at most one nonzero, with pivot $i_2(i_1(j))$. □

Echelon Pivot Matrices: Echelon pivot matrices are pivot matrices with added structure. There are 4 types we consider

$$E_L = \quad E_U = \tag{8}$$
$$\hat{E}_L = \quad \hat{E}_U = \tag{9}$$



E_L and $\hat{E}_U$ contain the pivots for variants of the column echelon form of a matrix, and E_U and $\hat{E}_L$ contain the pivots for variants of the row echelon form of a matrix. The L and U subscript indicates whether the matrix is lower or upper triangular.

Symbol	Meaning	Shape
A	Arbitrary matrix	
D	Diagonal matrix	
L	Lower-triangular	
U	Upper-triangular	
T	Any triangular form	
S	Schur Complement	
P	Permutations	
I	Identity	
J	Anti-diagonal permutation	
E	Echeleon-diagonal	
E_L	Echelon pivot lower	
$\hat{E}_L$	Echelon pivot lower	
E_U	Echelon pivot upper	
$\hat{E}_U$	Echelon pivot upper	
Q	Pivot matrix	

Figure 1: Notation for different matrices, along with pictorial symbols

Definition 2.1.4. *A matrix has the E_L shape if it is the sum of rank 1 matrices created from basis vectors*

$$E_L = \sum_{(i,j)\in S} e_i e_j^T$$

The set $S \subset \{1,\dots,m\} \times \{1,\dots,n\}$ contains the locations of the pivots. Since E_L is a pivot matrix, for every j, there must be a unique i, therefore the pairs can be written as $(i(j), j)$. The function $i(j)$ is defined on the subset of columns that have a pivot, and must satisfy the following properties

1. *$j_1 < j_2 \implies i(j_1) < i(j_2)$ on the domain of $i(j)$*
2. *For every j_1, j_2 s.t. $j_1 < j_2$ and $A[j_1] = 0 \implies A[j_2] = 0$*

The other echelon pivot matrices can be defined in terms of the E_L shape.

Definition 2.1.5. *A matrix is of shape E_U if its transpose is of shape E_L*

$$(E_U)^T = E_L$$

Definition 2.1.6. *A matrix is of shape $\hat{E}_L$ if its J-Conjugate is of shape E_L*

$$J\hat{E}_L J = E_L$$

Definition 2.1.7. *A matrix is of shape $\hat{E}_U$ if its J-Conjugate is of shape E_U*

$$J\hat{E}_U J = E_U$$

2.2 Cell Complexes

We will now turn to topological notions. First, we need to know how we can construct topological spaces from basic building blocks such as cells or simplices. It turns out that many topological spaces of practical interest can be represented cellular or simplicial complexes. See Hatcher [20] for a more complete discussion.

A *cell complex*, or CW complex $\mathcal{X}$ can be built inductively by starting with a discrete set of points (0-cells) $\mathcal{X}^0$ called the 0-skeleton, and inductively forming the k-skeleton $\mathcal{X}^k$ from $\mathcal{X}^{k-1}$ by adding open k-dimensional balls along their boundary to $\mathcal{X}^{k-1}$.

Cell complexes offer a general way to encode spaces, but in many applications the need to specify all boundary maps can be onerous. Often it is easier to use simplcial or cubical complexes, both of which are special cases of cell complexes, for which the boundary maps come for free. We will focus on simplicial complexes, which are commonly used for the purposes of triangulating spaces. A k-simplex is simply the convex hull of $k+1$ points in general position, denoted $s = (x_0, \dots, x_k)$. A k-simplex has $k+1$ faces which are $(k-1)$-simplices in its boundary, each which can be obtained by removing a single vertex $\partial(x_0, \dots, x_k) = \{(x_0, \dots, \hat{x}_i, \dots, x_k)\}$, where $\hat{x}_i$ indicates that the ith point has been removed. A *simplicial complex* $\mathcal{X}$ is a collection of simplices, where if a simplex $s \in \mathcal{X}$, its boundary is in $\mathcal{X}$: $\partial s \subset \mathcal{X}$.

We are also interested in maps between spaces. A map $f : \mathcal{X} \to \mathcal{Y}$ is *cellular* if $f(\mathcal{X}^k) \subset \mathcal{Y}^k$ for all k. For simplicial complexes, we say a map $f : \mathcal{X} \to \mathcal{Y}$ is simplicial if each simplex of $\mathcal{X}$ maps linearly to a simplex of $\mathcal{Y}$ (possibly of lower dimension) by mapping vertices to vertices and extending linearly. Explicitly, once a map on vertices $f^0 : \mathcal{X}^0 \to \mathcal{Y}^0$ has been specified, for higher order simplices we have

$$f^k : (x_0, \dots, x_k) = (f^0(x_0), \dots, f^0(x_k)) \tag{10}$$

2.3 Chain Complexes

A *chain complex* is a sequence of vector spaces $\{C_k\}$ $k = 0, 1, \ldots$ with *boundary* maps $\partial_k : C_k \to C_{k-1}$ with the property that $\partial_{k-1} \circ \partial_k = 0$. Throughout, we'll have $\partial_0 = 0$

$$0 \longleftarrow C_0 \xleftarrow{\partial_1} C_1 \xleftarrow{\partial_2} \cdots \xleftarrow{\partial_k} C_k \xleftarrow{\partial_{k+1}} \cdots$$

We'll use C_* to refer to the set of vector spaces as well as the maps in the chain complex. When considering more than one chain complex, for clarity we may use ∂^C to denote the boundaries in C_*. Elements of C_k are referred to as k-chains, elements of $\ker \partial_k$ are referred to as cycles, and elements of $\operatorname{img} \partial_{k+1}$ are referred to as boundaries.

A *chain map* between chain complexes C_* and D_* is a set of maps $\{F_k : C_k \to D_k\}$, $k = 0, 1, \ldots$ with the property $F_k \circ \partial^C_{k+1} = \partial^D_{k+1} \circ F_{k+1}$, i.e. the following diagram commutes

$$\begin{array}{ccccccccc} 0 & \longleftarrow & C_0 & \xleftarrow{\partial^C_1} & C_1 & \xleftarrow{\partial^C_2} & \cdots & \xleftarrow{\partial^C_k} & C_k & \xleftarrow{\partial^C_{k+1}} & \cdots \\ & & \downarrow{\scriptstyle F_0} & & \downarrow{\scriptstyle F_1} & & & & \downarrow{\scriptstyle F_k} & & \\ 0 & \longleftarrow & D_0 & \xleftarrow{\partial^D_1} & D_1 & \xleftarrow{\partial^D_2} & \cdots & \xleftarrow{\partial^D_k} & D_k & \xleftarrow{\partial^D_{k+1}} & \cdots \end{array} \tag{11}$$

We'll use $F_* : C_* \to D_*$ to denote a chain map.

There is a functor from the category of cell (simplicial) complexes to the category of chain complexes over a field $\mathbb{F}$, meaning that every cell (simplicial) complex $\mathcal{X}$ has an associated chain complex $C_*(\mathcal{X})$, and every cellular map $f : \mathcal{X} \to \mathcal{Y}$ has an associated chain map $F_* : C_*(\mathcal{X}) \to C_*(\mathcal{Y})$. We will consider the complex of cellular chains, where $C_k(\mathcal{X})$ is constructed as the free vector space with basis given by the k-cells of $\mathcal{X}$ - in other words elements of $C_k(\mathcal{X})$ are formal $\mathbb{F}$-linear combinations of k-cells in $\mathcal{X}$. We will not distinguish between a cell $s \in \mathcal{X}$ and the basis vector generated by s in $C_*(\mathcal{X})$ unless it is not clear from context.

Boundary matrices ∂_k are obtained by examining how faces are attached to the oriented boundary of cells. For general cell complexes this is obtained from the specified attaching maps, but for simplicial complexes the formula is combinatorial

$$\partial_k(x_0, \ldots, x_k) = \sum_{i=0}^{k} (-1)^i (x_0, \ldots, \hat{x}_i, \ldots, x_k)$$

For example, an edge (x_0, x_1) has boundary $(x_1) - (x_0)$.

Given a map $f : \mathcal{X} \to \mathcal{Y}$, one can also compute chain maps $F_* : C_*(\mathcal{X}) \to C_*(\mathcal{Y})$. Again, for cell complexes this may take some work to specify, but for simplicial maps, one need only worry about how the vertices map (since higher order simplices are extended linearly). For instance, we have

$$F_0 : (x) \to (f(x))$$

and for higher order simplices

$$F_k : (x_0, \ldots, x_k) \to (\operatorname{sgn} P)(f(x_0), \ldots, f(x_k))$$

If multiple vertices of x map to the same vertex in Y, then the simplex $(f(x_0), \ldots, f(x_k))$ is *degenerate*, so is not in the chain basis, and the coefficient for $(x_0, \ldots, x_k)$ in the chain map is zero. This follows from considering the simplicial complex $\mathcal{Y}$ as a simplicial set [18].

2.4 Homology

Given a chain complex C_*, the *homology* vector space in dimension k is defined as the quotient vector space $H_k(C_*) = \ker \partial_k / \operatorname{img} \partial_{k+1}$. Because $\partial_k \circ \partial_{k+1} = 0$, we know that $\operatorname{img} \partial_{k+1} \subseteq \ker \partial_k$, so the quotient vector space is defined.

A chain map $F_* : C_* \to D_*$ produces an induced map in homology $\tilde{F}_k : H_k(C_*) \to H_k(D_*)$ [20]. We'll make an argument using representatives. Let $x_k \in [x_k] \in H_k(C_*)$, and $x'_k \in \operatorname{img} \partial^C_{k+1}$, meaning there is some $y \in C_{k+1}$ so that $x'_k = \partial^C_{k+1}(y)$, and that $[x_k + x'_k] = [x_k]$. Then

$$\begin{aligned} F_k(x_k + x'_k) &= F_k x_k + F_k \partial^C_{k+1} y \\ &= F_k x_k + \partial^D_{k+1} F_{k+1} y \end{aligned}$$

First, note that because x_k is a homology representative, $x_k \in \ker \partial^C_k$, and from the commutation property of chain maps Equation (11), $F_k x_k$ must be in $\ker^D_k$, so is also a representative for some homology class in $H_k(D_k)$. Next, since $\partial^D_k \circ \partial^D_{k+1} = 0$, $F_{k+1} y \in \ker \partial^D_k$, he homology class of $F_k x_k$ does not depend on the representative of $[x_k]$ chosen. Thus

$$\tilde{F}_k[x_k] = [F_k x_k]. \tag{12}$$

When homology is computed on a chain complexes associated with a topological space $\mathcal{X}$, certain topological information can be extracted from the vector spaces $H_k(\mathcal{X})$. The dimension of $H_0(\mathcal{X})$

is equal to the number of connected components in $\mathcal{X}$, the dimension of $H_1(\mathcal{X})$ counts the number of non-contractible loops, and $H_k(\mathcal{X})$ generally counts the number of k-dimensional voids in $\mathcal{X}$. Representatives of vectors in H_k are also known as *generators*, and consist of linear combinations of cells in a subcomplex of $\mathcal{X}$, and since H_k is a quotient vector space, representatives are not unique. Because there are generally many choices of basis for $H_k(\mathcal{X})$, as well as many choices of representative, representatives need not be particularly interpretable, and may generally appear to be quite complex.

Induced maps on homology are often more revealing, and information about the kernel and image of a map can be used to understand what features in $\mathcal{X}$ are collapsed by a map to another space $\mathcal{Y}$. The actual matrix representation is dependent on the bases chosen for homology.

2.5 Computing Homology

Given matrices ∂_k and ∂_{k+1}, with the property $\partial_k \partial_{k+1} = 0$. We seek to compute the quotient space $\ker \partial_k / \operatorname{img} \partial_{k+1}$. This will require finding a basis for $\ker \partial_k$ (the k-cycles), and finding a sub-basis which is not in $\operatorname{img} \partial_{k+1}$.

Definition 2.5.1. *A* homology revealing *basis for C_k is a pair $(B_k, \mathcal{I}_k)$, where B_k is a basis for C_k, and $\mathcal{I}_k$ is an index set such that $\{b_i \in B_k\}_{i \in \mathcal{I}_k} \subseteq B_k$ generates a basis for $H_k(C_*)$. Explicitly, a basis for H_k is*

$$\{[b_i] \mid b_i \in B_k, i \in \mathcal{I}_k\}$$

In practice, a homology revealing basis for C_k is computed by first finding a basis for cycles $\ker \partial_k^C$, and then finding a sub-basis for cycles which are not boundaries $\operatorname{img} \partial_{k+1}^C$. A homology-revealing basis is certainly not unique choice - there may be other choices of $\mathcal{I}_k$ that would also give representatives that generate a basis for homology, or we can always modify the representatives by adding arbitrary boundaries from $\operatorname{img} \partial_{k+1}$. Once we have chosen a basis B_k and a set $\mathcal{I}_k$, we will say a homology representative $x \in [x]$ is the preferred representative of $[x]$ if x is written as linear combination of cycles exclusively in the set $\mathcal{I}_k$.

Proposition 2.5.2. *Given a homology-revealing basis $(B_k, \mathcal{I}_k)$ for C_k, every homology class $[x] \in H_k(C_*)$ has a unique preferred representative.*

Proof. Existence comes by definition, since $(B_k, \mathcal{I}_k)$ generates a basis.

Now, suppose that a homology class has two preferred representatives $\sum_{i \in \mathcal{I}_k} \alpha_i b_i$ and $\sum_{i \in \mathcal{I}_k} \beta_i b_i$. Then using Equation (3), we have

$$\Big[\sum_{i \in \mathcal{I}_k} \alpha_i b_i\Big] = \Big[\sum_{i \in \mathcal{I}_k} \beta_i b_i\Big]$$
$$\sum_{i \in \mathcal{I}_k} \alpha_i [b_i] = \sum_{i \in \mathcal{I}_k} \beta_i [b_i]$$

because $\{[b_i]\}$ is the generated basis for homology, we must have $\alpha_i = \beta_i$ for all $i \in \mathcal{I}_k$. □

The advantage of working with homology revealing bases explicitly is that we can reason about vectors in $H_*(C_*)$ in the generated basis using preferred representatives in the chain complex.

2.5.1 The Reduction Algorithm

In this section we'll review a common approach for finding a homology-revealing basis known as the reduction algorithm [38], which has also has useful properties for computing persistent homology of filtrations. It involves putting the boundary matrices ∂_k in a reduced form $\partial_k U_k = R_k$, or written as a factorization $\partial_k = R_k U_k^{-1}$, and extracting homology information from U_k and R_k. We define the *pivot* of a column $\operatorname{piv} v$ to be the largest index i so that $v[i]$ that has a non-zero value.

$$\operatorname{piv} v = \max\{i \mid v[i] \neq 0\}$$

If there are non non-zero values in v, we say $\operatorname{piv} v = 0$.

Algorithm 1 [38] Reduction Algorithm with formation of basis U.

Input: $m \times n$ matrix A
Result: Factorization $AU = R$.
$U = I_n$
$R = A$
for $j = 1, \ldots, n$ **do**
  while $\operatorname{piv}(R[j]) > 0$ and $j' < j$ exists so that $i = \operatorname{piv}(R[j]) = \operatorname{piv}(R[j'])$ **do**
    $\alpha = R[i, j]/R[i, j']$
    Update $R[j] = R[j] - \alpha R[j']$
    Update $U[j] = U[j] - \alpha U[j']$
  end while
end for
return U, R

In practice, pivots can be remembered using a data structure that permits fast look-up, and A can be modified in-place to form R. We will call the matrix R the *reduced matrix*. The rest of this section parallels the analysis found in [38], but we provide some additional care to explain why U is a homology-revealing basis in the context of applying the reduction algorithm to chain boundary matrices.

Proposition 2.5.3. *Algorithm 1 produces a valid factorization $AU = R$, where U is upper-triangular, and no two columns of R share the same non-zero pivot.*

Proof. At the beginning of the algorithm, we have the trivial identity $AU_0 = R_0$, where $U_0 = I$. We will count iterations over the columns using j, and iterations of the while-loop using ℓ.

Suppose that at step $\ell - 1$, we have maintained the invariant $AU_{\ell-1} = R_{\ell-1}$. Each iteration of the while loop performs a column operation that is equivalent to multiplying the matrices $R_{\ell-1}$ and $U_{\ell-1}$ on the right by a matrix $U'_\ell = I - e_{j'}\alpha e_j$. Then we have $AU_{\ell-1}U'_\ell = R_{\ell-1}U'_\ell$. Writing $R_\ell = R_{\ell-1}U'_\ell$ and $U_\ell = U_{\ell-1}U'_\ell$, we see that we have $AU_\ell = R_\ell$. Note that these updates are done in-place in the matrices R and U. By induction, the invariant $AU_\ell = R_\ell$ is maintained throughout the algorithm.

Note that $U_0 = I$ is upper-triangular. Suppose that $U_{\ell-1}$ is upper-triangular. Note that $U'_\ell = I - e_{j'}\alpha e_j$ is upper-triangular as well, and because the class of upper triangular matrices is closed under multiplication, $U_\ell = U_{\ell-1}U'_\ell$ will also be upper triangular. By induction, U_ℓ is upper-triangular throughout the algorithm.

Now, we will show that we maintain the invariant that columns $j' \leq j$ in R have unique non-zero pivots. We consider an arbitrary column j in R. Note that the j-th column of R is only modified during the j-th iteration of the for-loop. Note that at each iteration of the while loop, if j shares a pivot with column $j' < j$, then that pivot of j is eliminated, and the pivot of column j decreases (because adding $R[j']$ can not introduce non-zeros after the pivot). This process continues until there are no columns $j' < j$ that share a pivot with j, or the pivot becomes zero. Thus, j shares no pivots with columns $j' < j$. Because this holds for all columns j, the final R can not have two columns with the same pivot (otherwise this principle would be violated for the column with larger index).

When the loops terminate, we take U to be the final state U_ℓ, and R to be the final state R_ℓ. Thus, $AU = R$, U is upper-triangular, and R has unique non-zero pivots.

□

Proposition 2.5.4. *The worst case run-time of Algorithm 1 is $O(m^2 n)$.*

Proof. For a single column, the updates take worst case $O(m)$ time each (to loop over every entry), and we can iterate through at most m pivots, for at most $O(m^2)$ time for each column. For all n columns, the total time is $O(m^2 n)$. □

Note that due to sparsity of the boundary matrices, the worst case runtime is generally pessimistic.

As a consequence of Proposition 2.5.3, the set of non-zero columns in R is linearly independent because they have distinct pivots. Finally, note that $\operatorname{img} R = \operatorname{img} A$ because U is just a change of basis in the columns.

Now, we'll explain how to extract information about homology assuming we have reduced boundary matrices for each dimension k as $\partial_k U_k = R_k$. First, note that we can extract a basis for cycles from U_k by examining which columns of R_k are zero. Specifically, if $R_k e_j = 0$, then $R_k e_j = \partial_k U_k e_j = \partial_k U_k[j] = 0$, so $U_k[j] \in \ker \partial_k$. Because U_k is unit upper-triangular its columns are linearly independent, so the collection of cycles found in this way forms a basis for $\ker \partial_k$.

Now that we have a basis for cycles, we want to find a basis for homology by finding cycles that are not in $\operatorname{img} \partial_{k+1}$.

Lemma 2.5.5. *If $\operatorname{piv} R_{k+1}[j] = i > 0$, then $U_k[i]$ is a cycle.*

Proof. We know $\partial_k R_{k+1}[j] = \partial_k \partial_{k+1} U_{k+1}[j] = 0$. Thus,

$$\partial_k \left[R_{k+1}[i,j]e_i + \sum_{\ell<i} R_{k+1}[\ell,j]e_\ell \right] = 0.$$

This means that

$$\partial_k e_i = \sum_{\ell<i} (R_{k+1}[\ell,j]/R_{k+1}[i,j])\partial_k e_\ell,$$

so $\partial_k e_i = \partial_k[i]$ can be written as a linear combination of columns $\partial_k e_\ell = \partial_k[\ell], \ell < i$. This means that $R_k[i] = 0$, so $U_k[i]$ is a cycle. □

Proposition 2.5.6. *The set of cycles in U_k whose column index do not appear as a pivot in R_{k+1} form a basis for $H_k = \ker \partial_k / \operatorname{img} \partial_{k+1}$.*

Proof. We'll first show that if $U_k[i]$ is a cycle, and i does not appear as a pivot in R_{k+1}, that $U_k[i] \notin \operatorname{img} R_{k+1} = \operatorname{img} \partial_{k+1}$. Suppose $U_k[i] \in \operatorname{img} R_{k+1}$. Because $U_k[i,i] = 1$, there must be some linear combination of columns of R that produces a 1 in the i-th entry.

$$1 = \sum_j \alpha_j R_{k+1}[i,j]$$

Let j' be the index column with largest pivot and $\alpha_{j'} \neq 0$. We know that $\operatorname{piv} R_{k+1}[j'] > i$ because i does not appear as a pivot in R_{k+1}, and because $U_k[i,i] = 1]$, some column with pivot greater than i must be used. Next, note that because only one column in R_{k+1} can have pivot $i' = \operatorname{piv} R_{k+1}[j']$, we must have $U_k[i',i] = \alpha_{j'} R_{k+1}[i',j']$. However, since $i' > i$ and U_k is upper-triangular, we must have $U_k[i',i] = 0$, introducing a contradiction. Thus, $U_k[i]$ is not in $\operatorname{img} R_{k+1}$.

Next, note that the set of such cycles is linearly independent since they are distinct columns in an upper-triangular matrix. Furthermore, by Lemma Lemma 2.5.5, each non-zero column in R_{k+1} is matched with a cycle in U_k.

Finally, we can count dimensions. Note that $\dim H_k = \dim \ker \partial_k - \dim \operatorname{img} \partial_{k+1}$, and $\operatorname{img} \partial_{k+1} = \operatorname{img} R_{k+1}$. The number of cycles that do not appear as pivots in R_{k+1} is exactly $\dim \ker \partial_k - \dim \operatorname{img} R_{k+1}$, and since these cycles are linearly independent and represent non-trivial homology classes, they must form a basis for H_k. □

What we have shown is that the columns of U_k form a homology-revealing basis for the chain complex, where the subset of the basis that gives the homology basis is determined by looking at columns of R_k that have zero pivot, and which do not appear as a pivot in R_{k+1}. This produces an algorithm to get the indices $\mathcal{I}_k$ for U_k that give the homology basis.

Algorithm 2 Extraction of homology-revealing bases from reduced boundary matrices.

```
Input: Matrices U_k, R_k  k = 0, 1, ... from reduction algorithm.
Result: Index sets I_k for homology bases.
for k = 0, 1, ... do
    I_k ← {}
    n_k ← dim C_k
    for j = 1, ..., n_k do
        if piv(R_k[j]) = 0 and j is not a pivot in R_{k+1} then
            I_k ← I_k ∪ {j}
        end if
    end for
end for
return {I_k}
```

Proposition 2.5.7. *Algorithm 2 takes $O(n_k)$ time for dimension k, where n_k is the number of columns in R_k.*

Proof. We assume that it takes $O(1)$ time to find the pivot of a column in R_k (e.g. this can be stored in an array while performing the reduction algorithm, or simply computed if R_k is in a sparse format such as CSC or list of lists). We also assume that we can check if a column of R_{k+1} has a given pivot in $O(1)$ time, for instance this can be stored using an array or dictionary when running the reduction algorithm.

With these assumptions, it takes $O(1)$ time to check whether each column of R_k will be used to represent homology, for a total time of $O(n_k)$ in dimension k. □

2.5.2 Induced Maps

A final ingredient we need is the ability to compute induced maps on homology. Assume we have a chain map $F_* : C_* \to D_*$, and that we've found homology-revealing bases represented by $(U_*^C, \mathcal{I}_*^C)$ and $(U_*^D, \mathcal{I}_*^D)$ for C_* and D_*. Note that we wish to use preferred representatives to obtain the induced map $\tilde{F}_k : H_k(C_*) \to H_k(D_*)$ in terms of the generated bases by using linearity of the quotient Equation (3)

$$\tilde{F}_k[x] = [\sum_{i \in \mathcal{I}_k^D} \alpha u_i] = \sum_{i \in \mathcal{I}_k^D} \alpha_i [u_i]. \tag{13}$$

We'll consider how a homology basis vector $[x] \in H_k(C_*)$, represented by a preferred representative $x = U_k^C[i], i \in \mathcal{I}_k^C$ passes through the chain map F_k. The image in the basis U_k^D is given by Equation (2) as $y = (U_k^D)^{-1} F_k x$. We know from Equation (12) that y is a representative for the induced map on homology of $\tilde{F}_k[x]$, but it may not be the preferred representative determined for the homology revealing basis $(U_k^D, \mathcal{I}_k^D)$. Because $H_k(D) = \ker \partial_k^D / \operatorname{img} \partial_{k+1}^D$, we can arbitrarily add boundaries (elements of $\operatorname{img} \partial_{k+1}^D$) to y without changing the homology class. Furthermore, by Proposition 2.5.2, we know there exists a unique linear combination $\sum_{i \in \mathcal{I}_k^D} \alpha_i e_i$ that is in the same homology class as y in $H_k(D)$, which can be obtained by adding an element of $\operatorname{img} \partial_{k+1}^D$. If we have

used the reduction algorithm to obtain $\partial_k^D = R_k^D (U_k^D)^{-1}$, we can write ∂_k^D in the bases U_{k-1}^D, U_k^D as $(U_{k-1}^D)^{-1} \partial_k U_k^d = (U_{k-1}^D)^{-1} R_k$. Combining these observations, the linear system

$$(U_k^D)^{-1} R_{k+1}^D [\bar{\mathcal{I}_k^D}, :] z = y[\bar{\mathcal{I}_k^D}] \tag{14}$$

is consistent and can be solved for z, and then $y_2 = y - (U_k^D)^{-1} R_{k+1}^D z$ will be in the same homology class as y, but will only have non-zero coefficients in the preferred basis $\mathcal{I}_k^D$. We then can obtain the induced map on homology by reading off the coefficients $y_2[\mathcal{I}_k^D]$.

When we use the reduction algorithm to obtain R_k^D and U_k^D, we can write down an explicit algorithm to obtain y_2 via the solution of Equation (14). First note that R_{k+1}^D can be made upper triangular via a column permutation, and that the application of the upper triangular $(U_k^D)^{-1}$ will not affect the pivots. Thus, we can perform a variant of backward-substitution for upper-triangular matrices.

Algorithm 3 Computation of induced map on homology.

Input: Homology representative $x = U_k^C[i]$ in $H_k(C_*)$, U_k^D, R_{k+1}^D, from reduction algorithm applied to ∂_*^D, with index set $\mathcal{I}_k^D$. Chain map F_k in original basis.
Result: Induced map on homology, $\tilde{F}_k[x]$
$y \leftarrow (U_k^D)^{-1} F_k x$
$n \leftarrow \dim D_k$
$\hat{\partial}_{k+1}^D \leftarrow (U_k^D)^{-1} R_{k+1}$
for $j = n, n-1, \ldots, 1$ **do**
  if $y[j] \neq 0$ and j is a pivot of column i of $\hat{\partial}_{k+1}^D$ **then**
    $\alpha \leftarrow y[j] / \hat{\partial}_{k+1}^D[j, i]$
    $y \leftarrow y - \alpha \hat{\partial}_{k+1}^D[i]$
  end if
end for
return $y[\mathcal{I}_k^D]$

Proposition 2.5.8. *The homology class of y in Algorithm 3 is invariant.*

Proof. Only boundaries are added to y, as columns of $\hat{\partial}_{k+1}^D$, so the homology class is invariant. □

Proposition 2.5.9. *In Algorithm 3, at the end of the for-loop, y will be the preferred representative for its homology class with respect to the basis $(U_k^D, \mathcal{I}_k^D)$.*

Proof. This means that y will only have non-zeros in indices that are in the index set $\mathcal{I}_k^D$.

First, note that $\mathcal{I}_k^D$ was constructed to be the indices of cycles in D_k that did not appear in pivots of R_k, and $\hat{\partial}_k^D$ has the same pivots as R_k. Second, note that because x is a cycle, and F_k is a chain map, y must be a linear combination of cycles in the basis given by U_k^D. Finally, the for-loop removes non-zero coefficients for any cycle that has a non-zero pivot in $\hat{\partial}_k^D$. Thus, y can only have non-zero coefficients for the cycles indexed by $\mathcal{I}_k^D$. □

As a result, the coefficients returned by Algorithm 3 will be the coefficients for the induced map on homology as seen in Equation (13). We can construct a full matrix representing $\tilde{F}_k$ in the bases generated by the homology revealing bases by applying this procedure for every preferred basis element in U_k^C given by the index set $\mathcal{I}_k^C$.

Proposition 2.5.10. *Let n_k^C and n_k^D denote the dimension of C_k, D_k respectively, and β_k^C, β_k^D denote the respective homology dimensions. Then Algorithm 3 runs in $O((n_k^D)^2 n_{k+1}^D + (n_k^C n_k^D + (n_k^D)^2)\beta_k^C)$ time.*

Proof. We'll assume that the matrix $\hat{\partial}_{k+1}^D$ is formed once for all represenatives x, for a cost of $O((n_k^D)^2 n_{k+1}^D)$.

The application of the map F_k takes $O(n_k^C n_k^D)$ time, and the application of $(U_k^D)^{-1}$ takes $O((n_k^D)^2))$ time. The for-loop makes at most n_k^D iterations, and each update takes at most n_k^D time for an additional cost of $O((n_k^D)^2))$ time per representative. We then multiply the cost per representative by β_k^C representatives. □

Example 2.5.11. *Let $X = \{(x_0), (x_1), (x_0, x_1)\}$ and $Y = \{(y_0), (y_1), (y_0, y_1)\}$ be simplicial complexes, and let $f : X \to Y$ be the simplicial map that sends $(x_0) \mapsto (y_1)$, $(x_1) \mapsto (y_0)$. The chain boundaries and chain map can be expressed in the cell basis as*

$$\partial_0^X = \partial_0^Y = \begin{bmatrix} 0 & 0 \end{bmatrix}, \qquad \partial_1^X = \partial_1^Y = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \qquad F_0 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \qquad F_1 = \begin{bmatrix} 1 \end{bmatrix}$$

The reduction algorithm, Algorithm 1, will find the boundaries are already in reduced form, meaning $U_0^X = U_0^Y = I$, and we find $H_0(X) = H_0(Y) = \mathbb{F}$, with x_0 and y_0 selected as homology representatives.

Note that

$$F_0 x_0 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = y_1$$

is not a preferred representative for homology of $H_0(Y)$, so we need to use Algorithm 3 to find the induced map. Note that $\tilde{\partial}_1^Y = \partial_1^Y$, which has a non-zero pivot in index 2. Thus, the algorithm will eliminate the non-zero entry for y_1, and introduce a non-zero entry for y_0:

$$\begin{bmatrix} 0 \\ 1 \end{bmatrix} - \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = y_0$$

which is a preferred representative of the homology class. Finally, we just read off the coefficient, to obtain $\tilde{F}_0 = \begin{bmatrix} 1 \end{bmatrix}$.

In summary, the calculation confirms that the map f sends the single connected component of X to the single connected component of Y.

2.6 Persistent Homology of Filtrations

We will first consider persistent homology of filtrations before moving to the more general setting. This special case of persistent homology has been the focus of a lot of attention both for applications and algorithmic improvements in topological data analysis and we will focus on the special structure that aids in computation.

Recall that a filtration is a nested sequence of spaces

$$\mathcal{X}_0 \subseteq \mathcal{X}_1 \subseteq \cdots \subseteq \mathcal{X}_n \tag{15}$$

The persistent homology of the filtration studies how homology changes through the sequence of spaces.

$$H_k(\mathcal{X}_0) \to H_k(\mathcal{X}_1) \to \cdots \to H_k(\mathcal{X}_n) \tag{16}$$

Filtrations are not constrained to have integer parameters, for example the Rips and Čech filtrations both take real valued parameters. Our focus is on filtrations of finite cell complexes, in which case the filtration can be re-parameterized to take integer parameters with a distinct value for each real parameter that adds cells to the complex. For simplicity, we will consider filtrations where every cell is added one at a time. In general, multiple cells may appear at the same filtration value, in which case we can choose any arbitrary ordering that only ensures that a cell's boundary is present before a cell is added [35].

We can analyze what occurs at each step in this discrete filtration. The following result is standard in the literature [13].

Proposition 2.6.1. *Adding a k-dimensional cell x to a cell complex $\mathcal{X}$ either* creates *homology in dimension k or* destroys *homology in dimension $k-1$.*

Proof. The addition of a k-dimensional cell x appends a single column to ∂_k. Note that the only two subspaces that this can affect are $\ker \partial_k \subseteq C_k$ and $\operatorname{img} \partial_k \subseteq C_{k-1}$. There are two possibilities for reduction:

1. $\partial_k x$ is already in $\operatorname{img} \partial_k$, meaning that $\partial_k x = \partial_k y$ for some other chain $y \in C_k$. In this case, the chain $x - y$ will have boundary 0, so the dimension of $\ker \partial_k$ is incremented by 1. However, the dimension of $\operatorname{img} \partial_{k+1}$ does not change, so the dimension of $H_k = \dim \ker \partial_k - \dim \operatorname{img} \partial_{k+1}$ is also extended by 1. Because $\operatorname{img} \partial_k$ does not change, H_{k-1} does not change.

2. $\partial_k x$ is not already in $\operatorname{img} \partial_k$. In this case, the dimension of $\operatorname{img} \partial_k$ increases by 1, but because ∂_{k-1} is the same, so is $\ker \partial_k$. Thus the dimension of H_{k-1} will be reduced by 1. Because x is not in $\ker \partial_k$, H_k will be unaffected. However, a new pivot will be added to I_k, so H_{k-1} will be reduced by one dimension.

□

Definition 2.6.2. *The* birth *of a homology class is the index in the filtration that the homology class first appears. The* death *of a homology class is the index in the filtration that the class disappears. Each homology class in the filtration has an associated* birth-death pair (b, d)*, where $d = \infty$ if the class is present at the end of the filtration.*

Definition 2.6.3. *The* persistence barcode *of a filtered complex $\mathcal{X}_*$ is a multiset of birth-death pairs (b_i, d_i), one for each homology class that appears at some point in the filtration.*

2.7 Reduction Revisited

The reduction algorithm, Algorithm 1, can be used un-modified to compute persistent homology. All that is necessary is a little more nuance in its interpretation. Recall that the chain complex $C_k(\mathcal{X})$ has as a basis all k-cells in $\mathcal{X}$. Henceforth, we will assume that this basis is ordered by the order of appearance of cells in the filtration, and that the boundary matrices ∂_k are given in this

basis (i.e. the first column corresponds to the first k cell, and the first row corresponds to the first $k-1$ cell in the filtration).

The key observation is that because the reduction of column j only looks to the left for pivots, the reduction for the first j columns for ∂_k^j and $\partial_k^{j_2}$, $j < j_2$ will proceed in exactly the same manner. This means that instead of computing a homology revealing basis for each filtration parameter and examining the maps on homology induced by inclusion we can simply run the reduction algorithm once for the final cell complex $\mathcal{X}_n$, and add an additional layer of analysis to the discussion in Section 2.5.1.

Suppose we have performed the reduction algorithm $\partial_k U_k = R_k$, $k = 0, 1, \ldots$ for $\mathcal{X}_n$. Recall that if a column j of R_k is zero, then the jth column of U_k gives a cycle in $C_k(\mathcal{X}_n)$, and if j appears as a pivot in R_{k+1}, then that cycle is a boundary, so the index set $\mathcal{I}_k$ for the homology revealing basis is found by identifying the zero columns of R_k that do not appear as pivot indices in R_{k+1}, following Algorithm 2. However, if the reduction had been performed when column j was first added, no corresponding pivot column in R_{k+1} would yet be added, so the column would generate homology (causing a birth at the filtration parameter for column j). Finally, at some filtration parameter a column would be added to ∂_{k+1} producing a pivot in row j, killing that homology class. If no such column ever appears in ∂_{k+1}, the homology class will be present in the final complex $\mathcal{X}_n$, and the corresponding death will be at ∞.

The reason why the induced maps never need to be explicitly computed can be seen in the following proposition

Proposition 2.7.1. *Let $(U_k^i, \mathcal{I}_k^i)$ and $(U_k^j, \mathcal{I}_k^j)$ be homology revealing bases for a filtration $X_i \subseteq X_j$ computed using the ordered cell basis and the reduction algorithm. Then the induced map on homology $H_k(X_i) \to H_k(X_j)$ from inclusion either (a) sends a basis element $[x] \in H_k(X_i)$ to $[0]$, or (b) sends a basis element $[x] \in H_k(X_i)$ to exactly one other basis element $[x'] \in H_k(X_j)$. Furthermore, in case (b), the chain map sends the preferred representative of $[x]$ to the preferred representative of $[x']$.*

Proof. Let x be a preferred representative for $[x] \in H_k(\mathcal{X}_i)$. Note that because the reduction algorithm will produce the same result for the first n_i columns of ∂_k, the inclusion map $C_k(\mathcal{X}_i) \to C_k(\mathcal{X}_j)$ has the form

$$F_k = \begin{bmatrix} I \\ 0 \end{bmatrix}$$

where I denotes an identity and $(U_k^j)^{-1} F_k U_k^i = F_k$ takes the exact same form. We know that because x is a preferred representative, that it is a column of U_k^i, and the block identity in the chain map will map x to the corresponding column in U_k^j. Either that column is in $\mathcal{I}_k^j$, in which case it is a preferred representative for a basis element of $H_k(\mathcal{X}_j)$, or there must be a pivot with the corresponding index in R_{k+1}^j indicating that the column is a boundary, so is in $[0]$. □

Corollary 2.7.2. *The induced map on homology $\tilde{F}_k : H_k(X_i) \to H_k(X_j)$ is in E_U form.*

Proof. This follows from using the ordering inherited on the basis for homology from the ordering on the basis for the cell complex. □

2.8 Zigzag Homology

The introduction of zigzag homology started with arbitrary maps between spaces [6], and provided an algebraic algorithm to extract the zigzag barcode. The key construction is a *right filtration*, which decomposes each vector space in a quiver representation in terms of a filtration of subspaces coming from subspaces to the left in the quiver ordering.

Definition 2.8.1. *[6] A* right filtration *of a vector space V_n in a quiver of type A takes the form*

$$\mathcal{R}(V_n) = (R_0, R_1, \ldots, R_n)$$

where R_i are subspaces of V_n satisfying

$$0 = R_0 \leq R_1 \leq \cdots \leq R_n = V_n$$

right filtrations are defined recursively: $\mathcal{R}(V_1) = (0, V_1)$. Now, suppose we have defined $\mathcal{R}(V_n) = (R_0, \ldots, R_n)$. Then,

- if $V_n \xrightarrow{F_n} V_{n+1}$, $\mathcal{R}(V_{n+1}) = (F_n(R_0), \ldots, F_n(R_n), V_{n+1})$.
- if $V_n \xleftarrow{F_n} V_{n+1}$ then $\mathcal{R}(V_{n+1}) = (F_n^{-1}(0), F_n^{-1}(R_0), \ldots, F_n^{-1}(R_n))$

Where $F_n^{-1}(R_i)$ is the *inverse image* of R_i, *not* a matrix inverse. Right filtrations were useful as a proof technique in the construction of interval indecomposables, and have been adapted in implementations of zigzag homology to the chain complex setting [7].

2.9 From Inclusion to General Maps

Much of this section has focused on the use of persistent homology and zigzag homology in the specific case where all maps between spaces are inclusions. As we saw in the introduction, the more general case that uses arbitrary maps is well-defined, even if it isn't popularly used. We'll briefly cover how one may re-purpose existing tools to compute the more general case before we see how quiver algorithms solve it more gracefully.

The key construction we'll used is based on the mapping cylinder between two spaces [20]. Given a map $f : \mathcal{X} \to \mathcal{Y}$, the *mapping cylinder* is defined as the quotient space

$$\operatorname{Cyl} f = \mathcal{X} \times [0,1] \sqcup \mathcal{Y} / [(x,1) \sim f(x)]$$

where $\sqcup$ is the disjoint union. That is, $\operatorname{Cyl} f$ is obtained by taking the cylinder $\mathcal{X} \times [0,1]$ and attaching to a copy of $\mathcal{Y}$ by gluing $\mathcal{X} \times \{1\}$ to the image of f. The space is homotopic to $\mathcal{Y}$ (there is a retraction of $\mathcal{X}$ onto the image of f along the interval) so the space itself will have the same homology as $\mathcal{Y}$. However, if we add a filtration where $(\mathcal{X}, 0)$ appears at time 0, and the rest of the space appears at time 1, we see that homology in $\mathcal{X}$ that is killed by f dies at time 1 in the filtration, and homology in $\mathcal{X}$ in the image of f survives, while homology in $\mathcal{Y}$ that is in the cokernel of f is born at time 1.

There are several potential difficulties that may be encountered when attempting to use mapping cylinders in existing TDA packages. In the case where the map $f : \mathcal{X} \to \mathcal{Y}$ is simplicial, the cylinder has the structure of a simplicial set, which has desirable combinatorial properties. However, simplicial sets are not generally supported in TDA packages, and so one must fall back to either using general cell complexes, or triangulating the space to form a simplicial complex. In the more general case, the mapping cylinder of a cellular map is a cell complex. This can be encoded explicitly if desired and passed to any persistent homology package that can take cell complexes as input.

General cell complexes require one to know the boundary of each cell. We can use the product rule $\partial(a \times (0,1)) = -\partial a \times (0,1) + a \times \partial(0,1)$ to compute the boundary of cells in the cylinder. Given this information, one can either store the space as a general cell complex, or just form the boundary directly to pass to a solver. For a mapping cylinder $\operatorname{Cyl} f : \mathcal{X} \to \mathcal{Y}$, the boundary in dimension k is

$$\partial_k^{\operatorname{Cyl} f} = \begin{bmatrix} \partial_k^{\mathcal{X}} & & -I \\ & \partial_k^{\mathcal{Y}} & F_{k-1} \\ & & -\partial_{k-1}^{\mathcal{X}} \end{bmatrix} \tag{17}$$

One can check explicitly that $\partial^2 = 0$. The utility of the mapping cylinder is that the map on homology induced by inclusion $\mathcal{X} \hookrightarrow \operatorname{Cyl} f$ is equivalent to the map induced by $f : \mathcal{X} \to \mathcal{Y}$. Details can be found in appendix B. Thus, if we consider a filtration on $\operatorname{Cyl} f$, at which $\mathcal{X} \times \{0\}$ appears at parameter t_0, and the rest of the cylinder appears at parameter $t_1 > t_0$, in persistent homology the number of bars that survive from t_0 to t_1 will be equal to the rank of $\tilde{F}_k : H_k(\mathcal{X}) \to H_k(\mathcal{Y})$.

2.9.1 Persistent Homology

More generally, if we have a sequence of maps

$$X_0 \xrightarrow{f_0} \mathcal{X}_1 \xrightarrow{f_1} \dots \xrightarrow{f_{N-1}} \mathcal{X}_N$$

we can construct the *mapping telescope* [20]

$$\operatorname{Tel} f_i = \left[(\mathcal{X}_N \times \{N\}) \sqcup \bigsqcup_{i=0}^{N-1} (\mathcal{X}_i \times [i, i+1]) \right] / [(x, i+1) \sim (f_i(x), i+1)]$$

And we can use a filtration that adds each subsequent cylinder at times $t = 1, 2, \dots N$.

For the mapping telescope of cellular maps, the construction of the cell complex for the mapping cylinder can be extended in a straightforward way

$$\partial_k^{\operatorname{Tel} f_i} = \begin{bmatrix} \partial_k^{\mathcal{X}_0} & & -I & & \\ & \partial_k^{\mathcal{X}_1} & F_{0,k-1} & & -I \\ & & -\partial_{k-1}^{\mathcal{X}_0} & & \\ & & & \partial_k^{\mathcal{X}_2} & F_{1,k-1} \\ & & & & -\partial_{k-1}^{\mathcal{X}_1} \\ & & & & & \ddots \end{bmatrix}$$

Using a filtration that adds each cylinder at parameter $i+1$, we can compute persistent homology of the sequence of maps $\{f_i\}$ using Algorithm 1.

This construction can also be adapted for use with inclusion-based algorithms for zigzag homology in a straightforward way.

3 Quiver Classification and Barcodes

The application of quiver representations to persistent and zigzag homology has been of interest ever since it was used in the context of zigzag barcodes. A fairly complete survey of existing results and applications to topological data analysis can be found in [33]. In this section, we will introduce the necessary background to understand our algorithm.

3.1 Quiver Representations

A quiver representation is a directed graph $\mathcal{Q}(V, E)$ where every vertex $v_i \in V$ has an associated vector space V_i over a common field $\mathbb{F}$, and each directed edge $(v_i, v_j) \in E$ has an associated $\mathbb{F}$-linear transformation $A_{i,j} : V_i \to V_j$. Two quivers $\mathcal{Q}_1(V^1, E^1), \mathcal{Q}_2(V^2, E^2)$ are said to be isomorphic if the underlying graphs are isomorphic and there are change of bases B_i for each V_i^2 so that $A^1_{i,j} = (B_i)^{-1} A^2_{i,j} B_j$. Quiver representation theory is concerned with classifying quivers up to isomorphism, a problem that originated with classification of Lie Algebras [10].

The ability to classify an arbitrary quiver relies entirely on the underlying undirected graph, and not on the dimensions of the vector spaces or ranks of maps. Both persistent and zigzag homology are quivers of type A_n, for which the underlying graph is a line graph on n vertices, where n can be any finite positive integer.

$$\cdot \text{———} \cdot \text{———} \cdot \text{———} \cdots \text{———} \cdot$$

A theorem due to Gabriel shows that the collection of underlying graphs of quivers that have a finite number of indecomposable representations are known as the Dynkin diagrams, which include A_n, as well as several other classes of graph [17]. We will refer to type A_n quivers where all arrows point in the same direction *persistence-type* quivers, and type A_n quivers where arrows alternate direction *zigzag-type* quivers.

3.2 From Topology to Quiver Representations

Quiver representations arise naturally from diagrams of topological spaces through the homology functor. In Section 2.4, we saw that homology (with coefficients in a field $\mathbb{F}$) is a functor that associates topological vector spaces X with vector spaces $H_k(X)$, and maps $f : X \to Y$ to linear transformations $\tilde{F}_k : H_k(X) \to H_k(Y)$. This means that the homology functor turns diagrams of topological spaces into diagrams of vector spaces (quivers)

Example 3.2.1. *The homology functor creates the following transformation of diagrams*

$$\begin{array}{ccc} X & \xrightarrow{f} & Y \\ \downarrow{\scriptstyle g} & \overset{j}{\underset{i}{\rightleftarrows}} & \\ Z & & \end{array} \quad \Rightarrow \quad \begin{array}{ccc} H_k(X) & \xrightarrow{\tilde{F}_k} & H_k(Y) \\ \downarrow{\scriptstyle \tilde{G}_k} & \overset{\tilde{J}_k}{\underset{\tilde{I}_k}{\rightleftarrows}} & \\ J_k(Z) & & \end{array}$$

where $\tilde{F}_k = H_k(f)$, $\tilde{G}_k = H_k(g)$, $\tilde{J}_k = H_k(j)$, *and* $\tilde{I}_k = H_k(i)$

Note that neither the diagram of topological spaces nor the diagram of vector spaces is required to commute.

Example 3.2.2. *Persistent homology studies diagrams of spaces*

$$X_0 \xrightarrow{f_0} X_1 \xrightarrow{f_1} \cdots$$

which produces a quiver

$$H_k(X_0) \xrightarrow{(F_0)_k} H_k(X_1) \xrightarrow{(F_1)_k} \cdots$$

which corresponds to a Dynkin diagram of type A.

Similarly, zigzag homology studies diagrams of type A.

The advantage of using quivers to study persistent and zigzag homology instead of using an approach such as the one introduced in Section 2.9 is that the application of the homology functor is *trivially parallelizable*. In fact, this can be applied to general diagrams of spaces.

When it comes to representing diagrams of topological spaces or vector spaces, we will think of using the same data structure: a multidigraph

Definition 3.2.3. *A* multidigraph (V, E) is a collection V of vertices (nodes), and a multiset E of edges in $V \times V$.

A multidigraph is a more general version of a *graph*, in that edges are directed, and multiple edges can share the same source and target. We will only consider multidigraphs consisting of a finite number of nodes and edges.

A diagram in a category is then just a multigraph with an additional datum for each node and edge.

Example 3.2.4. *A diagram in* Top *is a multidigraph where every node* $i \in V$ *has an associated space* X_i*, and every edge* $(i,j) \in E$ *has an associated map* $f_{i,j} : X_i \to X_j$*.*

Example 3.2.5. *A diagram in* $\mathsf{Vect}_{\mathbb{F}}$ *is a multidigraph where every node* $i \in V$ *has an associated vector space* V_i*, and every edge* $(i,j) \in E$ *has an associated linear transofrmation* $T_{i,j} : V_i \to V_j$*. In other words, a quiver.*

The homology functor turns a diagram in Top to a diagram in $\mathsf{Vect}_{\mathbb{F}}$ which has the same underlying multidigraph structure, meaning the vertex and edge sets are the same, but the associated data is different.

Algorithm 4 Obtain a quiver representation from a diagram of spaces

1: **Input:** Multidigraph (V,E) with associated spaces X_i and maps $f_{i,j}$
2: **Result:** Multidigraph (V,E) with associated vector spaces $H_k(X_i)$ and maps $\tilde{F}_{i,j} = H_k(f_{i,j})$
3: **for** $i \in V$ **do**
4: Obtain chain complex $C_*(X_i)$
5: Obtain homology revealing bases $(U_*^i, \mathcal{I}_*^i)$ as well as reduced boundaries R_*^i using Algorithm 1.
6: **end for**
7: **for** $(i,j) \in E$ **do**
8: obtain $(\tilde{F}_{i,j})_* = H_*(f_{i,j})$ using Algorithm 3.
9: **end for**

The important observation is that both for loops of Algorithm 4 have *completely independent iterations*. That is, computing homology of X_i can be done completely independently of computing homology for X_j. Similarly, computing the induced maps $\tilde{F}_{i,j}$ only requires the pre-computed information for the source and target of the associated edge, and is independent of the computation on any other edge. This means the algorithm is *trivially parallelizable*, meaning that each for-loop iteration can be computed in parallel given enough processors. Furthermore, this is true for any diagram of spaces, meaning it will apply not only to the persistent and zigzag homology diagrams that we will study in this paper, but also other situations such as multiparameter persistence [8].

3.3 Quivers of type A_n

We will now focus on classification of type A_n quivers, which appear for both persistent and zigzag homology. The indecomposable representations of these quivers are known as interval indecomposables [6, 17, 33], and have the form

$$I[b,d] = \cdots \longrightarrow 0 \longrightarrow \mathbb{F} \longrightarrow \cdots \longrightarrow \mathbb{F} \longrightarrow 0 \longrightarrow \cdots$$

where b denotes the first index at which a copy of $\mathbb{F}$ appears, and d denotes the final index where $\mathbb{F}$ appears, all vector spaces with index $i \in [b,d]$ also have a copy of $\mathbb{F}$, with identity maps along all edges connecting two copies of $\mathbb{F}$ and zero maps along all other edges. In other words, any quiver of type A_n is isomorphic to the direct sum of these indecomposables

$$Q \cong \bigoplus_i I[b_i, d_i]$$

As a convention, we will use the lexicographical (total) order on $\mathbb{Z}^2$ when ordering interval indecomposables, using the parameters b, d. When the quiver is produced from induced maps on homology, the multiset $\{(b_i, d_i)\}$ is the barcode.

Definition 3.3.1. *The* companion matrix *of a quiver representation* $\mathcal{Q}$ *is the block matrix which has non-zero blocks in the non-zero entries of the adjacency matrix of the underlying directed graph, where blocks are filled by the linear transformations along the corresponding edges. By necessity, the size of the* i*-th block must be the dimension of* V_i *in the quiver.*

These matrices act on the vector space $V = \bigoplus_i V_i$ by sending vectors to their images in each linear transformation in Q. While general quivers may have multiple arrows between vector spaces, companion matrices can only represent quivers which have a underlying graph with at most one linear transformation on each edge – this will not limit our study of type A_n quivers which satisfy this property.

For example a persistence quiver $P_4 = \cdot \longleftarrow \cdot \longleftarrow \cdot \longleftarrow \cdot$ will have a companion matrix of the form

$$\begin{bmatrix} 0 & A_1 & & \\ & 0 & A_2 & \\ & & 0 & A_4 \\ & & & 0 \end{bmatrix}$$

whereas a zigzag quiver $Z_4 = \cdot \longrightarrow \cdot \longleftarrow \cdot \longrightarrow \cdot$ will have a companion matrix of the form

$$\begin{bmatrix} 0 & & & \\ A_1 & 0 & A_2 & \\ & & 0 & \\ & & A_3 & 0 \end{bmatrix}$$

Quiver isomorphism classes are maintained by conjugation of the companion matrix by block-diagonal change of bases matrices. For example two persistence quivers of type $P_3 = \cdot \longleftarrow \cdot \longleftarrow \cdot$ with maps denoted by A and B respectively are isomorphic if there exists an invertible matrix $M = M_1 \oplus M_2 \oplus M_3$ acting on V such that

$$\begin{bmatrix} 0 & A_1 & \\ & 0 & A_2 \\ & & 0 \end{bmatrix} = \begin{bmatrix} M_1 & & \\ & M_2 & \\ & & M_3 \end{bmatrix} \begin{bmatrix} 0 & B_1 & \\ & 0 & B_2 \\ & & 0 \end{bmatrix} \begin{bmatrix} M_1^{-1} & & \\ & M_2^{-1} & \\ & & M_3^{-1} \end{bmatrix}$$

An *indecomposable factorization* of the companion matrix A is a factorization $A = BTB^{-1}$, where B is an invertible (change of basis) matrix, and $T = \bigoplus_i I[b_i, d_i]$ is the matrix of indecomposables. We will allow for the indecomposable block to appear in any order but for exposition we will use the lexicographic partial order on $\mathbb{Z}^2$ to order the pairs (b_i, d_i). For example, in the case where P_4 and Z_4 both have indecomposable matrices $T = I[1,1] \oplus I[1,4] \oplus [2,3]$, the corresponding matrices are

$$T_{P_4} = \begin{bmatrix} 0 & & & & & & \\ & 0 & 1 & 0 & 0 & & \\ & 0 & 0 & 1 & 0 & & \\ & 0 & 0 & 0 & 1 & & \\ & 0 & 0 & 0 & 0 & & \\ & & & & & 0 & 1 \\ & & & & & 0 & 0 \end{bmatrix} \qquad T_{Z_4} = \begin{bmatrix} 0 & & & & & & \\ & 0 & 0 & 0 & 0 & & \\ & 1 & 0 & 1 & 0 & & \\ & 0 & 0 & 0 & 0 & & \\ & 0 & 0 & 1 & 0 & & \\ & & & & & 0 & 1 \\ & & & & & 0 & 0 \end{bmatrix} \tag{18}$$

where the information for the indecomposable $I[1,4]$ is colored in red. Note the indecomposable blocks all appear as adjacency matrices of sub-graphs of the underlying directed graph of the quiver. This means that even thought the indecomposables are written with the same notation, the indecomposable matrices are not identical due to the different directions of arrows. The advantage of the indecomposable factorization is that it is easy to determine the lengths the indecomposables, but information about which vector spaces participate is obscured.

A *barcode factorization* of the companion matrix A is a factorization $A = B\Lambda B^{-1}$, where B is a *block-diagonal* invertible matrix (representing a quiver isomorphism), where the block sizes are compatible with dimensions of the quiver, and

$$\Lambda = PTP^T$$

is the barcode matrix, where P is a permutation that preserves the block structure of A. Alternatively, we'll say Λ is the *barcode form* of the quiver companion matrix A, or simply the barcode form of the quiver representation. Continuing the previous example, in the case where P_4 and Z_4 both have barcode matrices $\Lambda \cong I[1,1] \oplus I[1,4] \oplus [2,3]$, the corresponding matrix representations are

$$\Lambda_{P_4} = \begin{bmatrix} & 0 & 0 & & & \\ & 1 & 0 & & & \\ & & & 1 & 0 & \\ & & & 0 & 1 & \\ & & & & & 1 \\ & & & & & 0 \end{bmatrix} \qquad \Lambda_{Z_4} = \begin{bmatrix} & & & & & \\ 0 & 1 & & 1 & 0 & \\ 0 & 0 & & 0 & 1 & \\ & & & & & \\ & & & & & \\ & & & 1 & 0 & \end{bmatrix} \tag{19}$$

The information for the indecomposable $I[1,4]$ is colored in red. Note that because the underlying graphs are different the matrices Λ are not equal even though the interval decomposition is superficially the same. In both quivers, the ranks of the vector spaces are $2, 2, 2, 1$. The advantage of the barcode factorization is that B clearly represents a quiver isomorphism due to its block structure.

Extracting the intervals $I[a,b]$ from a barcode factorization requires tracing the image of maps through the quiver, and the advantage compared to the indecomposable decomposition is that the starting point of an interval is clear. We extract the intervals from Λ by sweeping through the blocks left-to-right

Algorithm 5 Barcode Extraction

Input: Barcode matrix Λ, ranks of vector spaces V_i and directions of arrows in quiver.
Result: Barcode $\mathcal{B}$
for $i = 1, \ldots, n$ **do**
  for $j = 1, \ldots, \operatorname{rank} V_i$ **do**
    if $V_{i-1} \to V_i$ **then**
      if Row j in block i contains a non-zero in column j' of block $i-1$ **then**
        Extend the bar at index j' of block $i-1$ to have index j in block i.
      else
        Begin a bar with index j in block i
      end if
    else if $V_{i-1} \leftarrow V_i$ **then**
      if Column j in block i contains a non-zero in row j' of block $i-1$ **then**
        Extend the bar at index j' of block $i-1$ to have index j in block i.
      else
        Begin a bar with index j in block i
      end if
    else
      $(i = 1)$
      Begin bar with index j in block 1
    end if
  end for
end for
return $\mathcal{B}$

If we keep track of the indices used in each extension of a bar, we have the information necessary to form the permutation P so that $P\Lambda P^T = T$.

Proposition 3.3.2. *A companion matrix is in barcode form if and only if its blocks are pivot matrices.*

Proof. If all blocks of a companion matrix are pivot matrices, then every iteration of the for-loop in Section 3.3 will find at most one index that can be used to extend a bar. The set of bars found by the algorithm gives the indecomposables, so the matrix is in barcode form.

If the matrix is in barcode form, we'll consider the representation of the map $A_i : V \to W$, where either $V = V_i, W = V_{i+1}$ or $V = V_{i+1}, W = V_i$. Because the matrix is in barcode form, every basis vector $v \in V$ maps to either exactly one basis vector of W (continuing a bar), in which case the corresponding column of A_i has exactly one non-zero, or maps to zero (the bar ends at V), in which case the corresponding column of A_i is zero. Every basis vector $w \in W$ is either in the image of a basis vector of V, in which case the corresponding row of A_i has exactly one non-zero, or is the start of a new bar, in which case the row of A_i is zero. Thus A_i is a pivot matrix because it has at most one non-zero in each row and column. □

3.4 The Graded Module Structure of Persistent Homology

For persistence quivers, there is a relationship between the indecomposable factorization and the Jordan normal form of a matrix, observed in equation Equation (18). Because interval indecomposables have the structure of a sub-graph of the directed graph underlying the quiver, in the case of persistence quivers they will all have the form of a Jordan zero block. That is the sub-matrix associated with $I[a, b]$ is equal to $J_{[1+b-a]}(0)$, where

$$J_i(\lambda) = \begin{bmatrix} \lambda & 1 & & \\ & \lambda & \ddots & \\ & & \ddots & 1 \\ & & & \lambda \end{bmatrix}$$

is an $i \times i$ matrix. This immediately shows that the characteristic polynomial of A, $\det(tI - A) = t^N$, where $N = \sum \dim V_i$ in the quiver, and that A is nilpotent with index corresponding to the length of the longest interval $1 + b - a$.

Classification of persistent homology modules was established in [38] by showing that the persistent homology of inclusions has a $\mathbb{F}[T]$ module structure, where T acts by incrementing the filtration parameter. This first showed that the persistence barcode of a filtration is unique and computable. Explicitly, because $\mathbb{F}[T]$ modules are principal ideal domains, there is a unique description of the persistence module as

$$\bigoplus_{i \in \mathcal{I}} T^{b_i}\mathbb{F}[T]/T^{d_i} \oplus \bigoplus_{i \in \mathcal{J}} T^{b_i}\mathbb{F}[T]$$

where $\mathcal{I}$ is the set of finite bars in a persistence diagram, and $\mathcal{J}$ is the set of infinite bars. b_i and d_i correspond to the birth and death parameters in the discrete-time filtration.

In order to see the equivalence of the quiver representation and $\mathbb{F}[T]$ module viewpoints, we will consider infinite extensions of persistence quivers, where after a finite N all maps are isomorphisms

$$V_0 \xrightarrow{A_0} V_1 \xrightarrow{A_1} \cdots \xrightarrow{A_{N-1}} V_N \xrightarrow{I} V_{N+1} \xrightarrow{I} \cdots$$

In this case, any interval that is present at parameter N will extend infinitely to the right. Persistence quivers of this form appear naturally when considering homology of a filtered space

$$X_0 \subseteq X_1 \subseteq X_2 \subseteq \cdots \subseteq X_N = X_{N+1} = \cdots = X$$

The interval indecomposables will appear as transposed Jordan zero blocks, which are equivalent to the ordinary Jordan blocks via conjugation by the anti-diagonal matrix J

$$J^T \begin{bmatrix} 0 & & & \\ 1 & 0 & & \\ & \ddots & \ddots & \\ & & 1 & 0 \end{bmatrix} J = J_i(0) \tag{20}$$

The infinite extension of the quiver requires infinite indecomposables $I[b, \inf)$, represented by blocks of the form

$$\begin{bmatrix} 0 & & \\ 1 & 0 & \\ & \ddots & \ddots \end{bmatrix}$$

If we take $V = \bigoplus_i V_i$, and A to be the companion matrix of the quiver, there is $\mathbb{F}[A]$ module structure on V. The block structure of A shows that this is a graded module, graded by the index i of V_i, and the barcode factorization $A \cong \Lambda = \bigoplus_{i\in\mathcal{I}} I[b_i, d_i] \bigoplus_{j\in\mathcal{J}} I[b_j, \inf)$ shows gives a way to extract a basis for the module: $\{v_i\}_{i\in\mathcal{I}} \cup \{v_j\}_{j\in\mathcal{J}}$, where v_i is the vector with grade b_i that generates the subspace of the associated Jordan block. In the case of finite bars, these generate a torsion sub-module that disappears at grade d_i, so the sub-module is isomorphic to $T^{b_i}\mathbb{F}[T]/T^{d_i}$. In the case of infinite bars, the sub-module is free and so is isomorphic to $T^{b_i}\mathbb{F}[T]$.

We note that in practice, infinite extensions of quivers are not necessary for computation. We can simply take any bars that are present in the last vector space and extend them infinitely.

The module structure of persistent homology offers some insight into why the reduction algorithm works. The reason why only columns to the left can be used to eliminate pivots is that the grade of a basis element can not be altered under a change of basis. In other words, to maintain a $\mathbb{F}[T]$ module isormorphism it is valid to add elements with lower grades to elements with higher grades, but not *vice versa*.

3.5 Generic Quiver Computations

There is a correspondence between diagrams encoding quivers and block matrices encoding their companion matrices, and certain operations are easier to express using one notation or the other. In this section we establish some lemmas that apply to any quiver.

Lemma 3.5.1. *A change of basis (quiver isomorphism) at a single node via an invertible matrix M can be represented as*

Proof. This follows immediately from a change of basis on the central vector space in the diagram via Equation (2). □

If the quiver is representable by a companion matrix, this diagramatically encodes the quiver isomorphism

$$\begin{bmatrix} & B_0 & \dots & B_m \\ A_0 & & & \\ \vdots & & & \\ A_n & & & \end{bmatrix}$$
$$= \begin{bmatrix} M & & & \\ & I & & \\ & & \ddots & \\ & & & I \end{bmatrix} \begin{bmatrix} & M^{-1}B_0 & \dots & M^{-1}B_m \\ A_0M & & & \\ \vdots & & & \\ A_nM & & & \end{bmatrix} \begin{bmatrix} M^{-1} & & & \\ & I & & \\ & & \ddots & \\ & & & I \end{bmatrix}$$

We see that this only affects linear transformations that have the center node as a source or target. For any vector spaces that do not have an arrow to or from the center vector space are multiplied by an identity on both left and right and are unaffected.

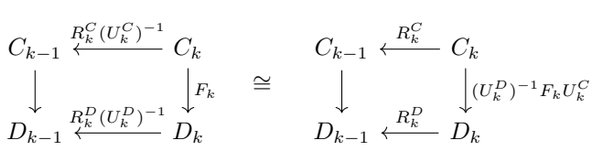
Lemma 3.5.1 implies the following two corollaries which set forth the rules for our *matrix passing algorithms*.

Corollary 3.5.2. *Passing an invertible matrix M through a target yields*

Corollary 3.5.3. *Passing an invertible matrix M through a source yields*

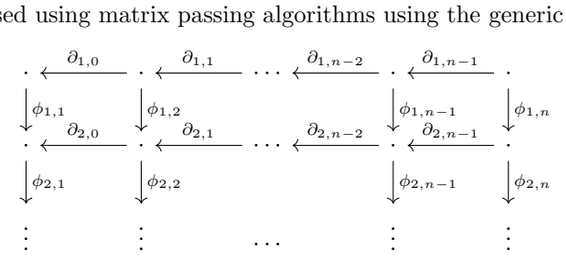
Notice that we draw arrows from right to left in the above diagrams. This is simply because the matrix M is closest to the central node. If we write arrows right to left we would have the correct, but less natural looking example

One can use this to easily verify the change of basis formula for induced maps on homology Equation (12) by tracking the extraction of the cycle-revelealing bases in a chain map $F_* : C_* \to D_*$, using factorizations $\partial_k = R_k U_k^{-1}$ and looking at the relevant sub-quiver:

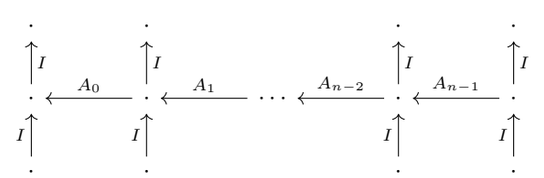


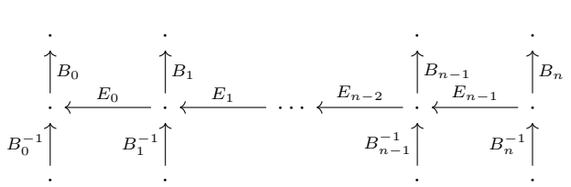
Generally, quivers on induced maps on homology can be derived from diagrams of chain maps.

The generic quiver computations are useful in representing a wide variety of computations. For example we can cast the problem of computing persistence homology from chain group information as a factorization of the following grid quiver. Since computing the barcodes entails a change of basis, it can be expressed using matrix passing algorithms using the generic quiver computation.



As another example, here is the standard type A quiver equipped with some extra identity maps. When the factorization algorithm is applied to the type A sub-quiver, the identity maps will be modified and will record the basis change matrices required for the factorization. In this way we see that these quiver diagrams and associated quiver computations are a useful tool for expressing these algorithms.





4 Algorithms for Canonical Forms of Type-A Quivers

In this section, we will describe our algorithm for computing the canonical form of a type-A quiver (Equation (1)). As we will see, the structure of the algorithm depends on the direction of the maps in the quiver, but there are core components which are the same.

There are two basic linear algebra operations we need as primitives for the algorithm.

1. Triangular factorizations
2. Shape commutation results with E-type matrices

We will first discuss them. Next we will consider the algorithm when all the the arrows point in the same direction. We will then show how this algorithm can be extended to the case of alternating arrow directions. Finally we will specify the full general algorithm.

4.1 Triangular Factorizations

In this section, we will discuss computing decompositions of a matrix A into triangular matrices with row or column pivoting. Specifically we will start with the LEUP decomposition. Variants include PLEU, UELP and PUEL form, which can be derived using the LEUP decomposition of either transposed or J-Conjugated versions of A. These factorizations are all variants of the standard LU decomposition [19], but we will need these specific forms for our quiver algorithm.

Given a matrix A, we will describe an algorithm that will maintain a block invariant at each step i

$$A = \begin{bmatrix} L_{11} & \\ L_{21} & I \end{bmatrix} \begin{bmatrix} E_{11} & \\ & \tilde{A} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ & I \end{bmatrix} P \tag{21}$$

Algorithm 6 LEUP factorization

Input: $m \times n$ matrix A
Result: Factorization $A = LEUP$
$L = I_m$
$U = I_n$
$P = I_n$
$i = 1, j = 1$
while $i <= m$ & $j <= n$ **do**
  if row i has a non-zero in column $j' \geq j$ **then**
    Swap columns j, j' in A
    Take Schur complement with respect to i, j entry
    $i = i + 1$
    $j = j + 1$
  else
    $i = i + 1$
  end if
end while

Proof of Correctness: We will show that the invariant Equation (21) is maintained at each step of the algorithm. Note that the loop increments i each iteration, so we use i as our index. For rows of A, as well as rows and columns of L, we will use the block index set $1 = \{0, \ldots, i-1\}$, and the block index set $2 = \{i+1, \ldots, m-1\}$, and for columns of A, as well as rows and columns of U we will use the block index set $1 = \{0, \ldots, j-1\}$ and block index set $2 = \{j+1, \ldots, n-1\}$.

Assume the invariant is maintained at the beginning of iteration i. We can break up the matrices into

$$A = \begin{bmatrix} L_{11} & & \\ L_{i1} & 1 & \\ L_{21} & & I \end{bmatrix} \begin{bmatrix} E_{11} & & \\ & A_{ij} & A_{i2} \\ & A_{2j} & A_{22} \end{bmatrix} \begin{bmatrix} U_{11} & U_{1j} & U_{22} \\ & 1 & \\ & & I \end{bmatrix} P$$

In the case that there are no non-zero entries in A_{ij} or A_{i2}, we simply increment i (in the else clause of the while loop), and the invariant is trivially maintained.

In the case there is a non-zero entry in A_{ij} or A_{i2}, assume we have already permuted it to the A_{ij} position, and used the relation Equation (34) to move the permutation to the right. We can write the interior matrix as

$$\begin{bmatrix} E_{11} & & \\ & A_{ij} & A_{i2} \\ & A_{2j} & A_{22} \end{bmatrix} = \begin{bmatrix} I & & \\ & I & \\ & A_{2j}A_{11}^{-1} & I \end{bmatrix} \begin{bmatrix} E_{11} & & \\ & A_{ij} & \\ & & S \end{bmatrix} \begin{bmatrix} I & & \\ & I & A_{11}^{-1}A_{12} \\ & & I \end{bmatrix}$$

where S is the Schur complement $S = A_{22} - A_{2j}A_{ij}^{-1}A_{i2}$. We can then pass off the left matrix to L and the right matrix to U, and now group A_{ij} with the echelon block E_{11}.

Note that because we may increment i without incrementing j, that the matrix E will be of type E_L.

4.1.1 Other triangular factorizations

$$A = LE_LUP \qquad (22)$$

$$A = PLE_UU \qquad (23)$$

$$A = U\hat{E}_ULP \qquad (24)$$

$$A = PU\hat{E}_LL \qquad (25)$$

All of the above factorizations can be shown to be equivalent to the LE_LUP factorization, using transposes and conjugation with J matrices.

For example consider the case,

$$A = (A^T)^T = (LE_LUP)^T = P^TU^T(E_L)^TL^T = \tilde{P}\tilde{L}E_U\tilde{U}$$

Thus the PLE_UU factorization is equivalent to the LE_LUP factorization of the transpose.

Similarly the $U\hat{E}_ULP$ can be expressed using J conjugation,

$$A = JJAJJ = J(JAJ)J = J\hat{A}J$$

Now we can replace $\hat{A}$ with its LE_LUP factorization, and apply the commutation relations of J.

$$J\hat{A}J = JLE_LUPJ = \hat{U}JE_LUPJ = \hat{U}\hat{E}_UJUPJ = \hat{U}\hat{E}_U\hat{L}JPJ = \hat{U}\hat{E}_U\hat{L}\hat{P}$$

Thus we have found the $U\hat{E}_ULP$ factorization of A.

If we apply the PLE_UU factorization for $\hat{A}$ instead, we get,

$$J\hat{A}J = JPLE_UUJ = \hat{P}JLE_UUJ = \hat{P}\hat{U}JE_UUJ = \hat{P}\hat{U}\hat{E}_LJUJ = \hat{P}\hat{U}\hat{E}_L\hat{L}$$

This gives us the $PU\hat{E}_LL$ factorization for the matrix A.

4.2 Shape Commutation relations

Now, we'll consider shape commutation relations of echelon matrices with triangular matrices. We will first show the following commutation relationship, and derive others from this.

Proposition 4.2.1. *Given an echelon pivot matrix E_L and lower triangular matrix L, we can rewrite the product E_LL in the following way*

$$E_LL = \tilde{L}E_L$$

Where $\tilde{L}$ matrix is also a lower triangular matrix.

Proof. In terms of matrix shapes this commutation looks like the following

Note that this is just the commutation of the shapes, In general L and $\tilde{L}$ are not the same matrices, or even the same dimensions. We will first characterize left and right multiplication by the E_L matrix.

Consider the product of E_L with an arbitrary matrix A

$$(E_LA)_{kl} = \sum_p (E_L)_{kp}A_{pl}$$

From the definition of the shape of E_L, we know that its entries are 1 only when $p = j(k)$, thus we get

$$(E_LA)_{kl} = \sum_p (E_L)_{kp}A_{pl} = (E_L)_{k,j(k)}A_{j(k),l} = A_{j(k),l}$$

Here we use the convention that if a particular k, l index pair is not assigned any value, it is by default 0. Similarly, if we are attempting to apply the function $j(\cdot)$ on an index not in its domain, the appropriate matrix element is 0. Multiplication on the right follows similarly

$$(AE_L)_{kl} = \sum_p A_{kp}(E_L)_{pl} = A_{k,i(l)}(E_L)_{i(l),l} = A_{k,i(l)}$$

We will now show that $\tilde{L}$, constructed in the following way will satisfy the proposition.

$$\tilde{L}_{kl} = \begin{cases} L_{j(k),j(l)}, & \text{if } k \in Domain(j(\cdot)) \text{ and } l \in Domain(j(\cdot)) \\ 1, & \text{if } k = l \text{ and } k \notin Domain(j(\cdot)) \\ 0, & \text{otherwise} \end{cases}$$

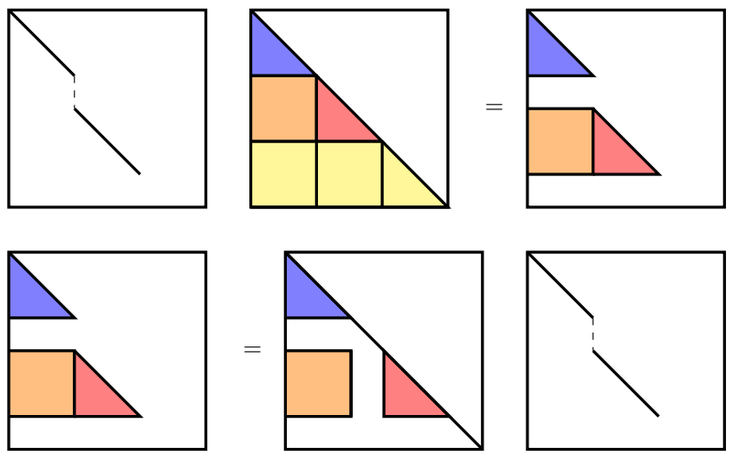


Figure 2: Pictoral representation of the shape commutation relationship in Proposition 4.2.1

Note that the above construction is not unique, we have opted to set the diagonal of otherwise zero columns to 1, this makes the matrix invertible, which will be useful later on. Next we will show that this construction is indeed lower triangular.

Suppose $k < l$, then we have from the definition of the shape of E_L that $j(k) < j(l)$ which implies that $L_{j(k),j(l)} = 0$, as L is a lower triangular matrix. This shows that for $k < l$, $\tilde{L}_{kl} = 0$, hence $\tilde{L}$ is lower triangular. Now we will compute $E_L L$ and $\tilde{L} E_L$ and show that they are equal.

$$(E_L L)_{kl} = L_{j(k),l}$$

$$(\tilde{L} E_L)_{kl} = \tilde{L}_{k,i(l)} = L_{j(k),j(i(l))} = L_{j(k),l}$$

Since $i(\cdot)$ is the inverse function of $j(\cdot)$, $i(l)$ will always be in the domain of $j(\cdot)$, thus

$$E_L L = \tilde{L} E_L$$

□

Proposition 4.2.2. *The following other shape commutation relations also hold*

$$L\hat{E}_L = \hat{E}_L \tilde{L} \qquad (26)$$

$$U E_U = E_U \tilde{U} \qquad (27)$$

$$\hat{E}_U U = \tilde{U} \hat{E}_U \qquad (28)$$

Proof. Taking transpose on the commutation result for E_L

$$(E_L L)^T = (\tilde{L} E_L)^T$$

$$L^T E_L^T = E_L^T \tilde{L}^T$$

Rewriting to denote the shapes we get,

$$U E_U = E_U \tilde{U}$$

Taking the J-Conjugate of the E_L commutation result we have,

$$J(E_L L)J = J(\tilde{L} E_L)J$$

$$J E_L L J = \hat{E}_U J L J = \hat{E}_U U$$

$$J \tilde{L} E_L J = \tilde{U} J E_L J = \tilde{U} \hat{E}_U$$

We get the commutation result for $\hat{E}_U$

$$\hat{E}_U U = \tilde{U} \hat{E}_U$$

Taking the transpose of the above result, we get,

$$(\hat{E}_U U)^T = (\tilde{U} \hat{E}_U)^T$$

$$U^T \hat{E}_U^T = \hat{E}_U^T \tilde{U}^T$$

Rewriting to denote shapes,

$$L\hat{E}_L = \hat{E}_L \tilde{L}$$

□

4.3 Algorithm for persistence-type quiver

At a high level, the algorithm will put each matrix in pivot matrix form. This is accomplished in two passes - we will work from left to right on the first pass, and then right to left on the second pass. Note that we could equally work in the opposite order (see fig. 4). We will use diagrams to notate the steps of the algorithm, keeping in mind that we can keep track of the invertible basis transformation for each of the steps. A pictoral description is contained in appendix C. We will first apply the LE_LUP factorization to A_0

$$\begin{array}{ccccccccc} \cdot & \xleftarrow{A_0} & \cdot & \xleftarrow{A_1} & \cdots & \xleftarrow{A_{n-1}} & \cdot \\ \cdot & \xleftarrow{L_0E_0U_0P_0} & \cdot & \xleftarrow{A_1} & \cdots & \xleftarrow{A_{n-1}} & \cdot \end{array}$$

We can now use matrix passing to move both U_0 and P_0 matrices, as they are both invertible. We then multiply the matrices to get $\tilde{A}_1 = U_0P_0A_1$

$$\begin{array}{ccccccccc} \cdot & \xleftarrow{L_0E_0} & \cdot & \xleftarrow{U_0P_0A_1} & \cdots & \xleftarrow{A_{n-1}} & \cdot \\ \cdot & \xleftarrow{L_0E_0} & \cdot & \xleftarrow{\tilde{A}_1} & \cdots & \xleftarrow{A_{n-1}} & \cdot \end{array}$$

We can now apply this procedure to $\tilde{A}_1$ and then iterate through the rest of the maps in the quiver representation

$$\begin{array}{ccccccccc} \cdot & \xleftarrow{L_0E_0} & \cdot & \xleftarrow{L_1E_1U_1P_1} & \cdots & \xleftarrow{A_{n-1}} & \cdot \\ & & \vdots & & \vdots & & \\ \cdot & \xleftarrow{L_0E_0} & \cdot & \xleftarrow{L_1E_1} & \cdots & \xleftarrow{L_{n-1}E_{n-1}U_{n-1}P_{n-1}} & \cdot \end{array}$$

Applying matrix passing on the final edge, we can remove the factor $U_{n-1}P_{n-1}$

$$\begin{array}{ccccccccc} \cdot & \xleftarrow{L_0E_0} & \cdot & \xleftarrow{L_1E_1} & \cdots & \xleftarrow{L_{n-1}E_{n-1}U_{n-1}P_{n-1}} & \cdot \\ \cdot & \xleftarrow{L_0E_0} & \cdot & \xleftarrow{L_1E_1} & \cdots & \xleftarrow{L_{n-1}E_{n-1}} & \cdot \end{array}$$

Next we can initiate the leftward pass by moving the lower triangular matrices leftward using the shape commutation relations at each step. We do so, as follows,

$$\begin{array}{ccccccccc} \cdot & \xleftarrow{L_0E_0} & \cdots & \xleftarrow{L_{n-2}E_{n-2}} & \cdot & \xleftarrow{L_{n-1}E_{n-1}} & \cdot \\ \cdot & \xleftarrow{L_0E_0} & \cdots & \xleftarrow{L_{n-2}E_{n-2}L_{n-1}} & \cdot & \xleftarrow{E_{n-1}} & \cdot \\ \cdot & \xleftarrow{L_0E_0} & \cdots & \xleftarrow{\tilde{L}_{n-2}E_{n-2}} & \cdot & \xleftarrow{E_{n-1}} & \cdot \end{array}$$

Here we have used the following shape commutation relation

$$L_{n-2}E_{n-2}L_{n-1} = \tilde{L}_{n-2}E_{n-2}$$

Applying iteratively the the rest of the edges in a right-to-left sweep, we obtain

$$\begin{array}{ccccccccc} \cdot & \xleftarrow{L_0E_0} & \cdots & \xleftarrow{\tilde{L}_{n-2}E_{n-2}} & \cdot & \xleftarrow{E_{n-1}} & \cdot \\ & & \vdots & & \vdots & & \\ \cdot & \xleftarrow{\tilde{L}_0E_0} & \cdots & \xleftarrow{E_{n-2}} & \cdot & \xleftarrow{E_{n-1}} & \cdot \end{array}$$

Finally, we can remove the last factor $\tilde{L}_0$, by matrix passing

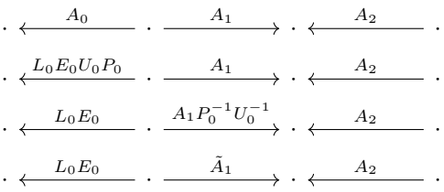
$$\begin{array}{ccccccccc} \cdot & \xleftarrow{\tilde{L}_0E_0} & \cdots & \xleftarrow{E_{n-2}} & \cdot & \xleftarrow{E_{n-1}} & \cdot \\ \cdot & \xleftarrow{E_0} & \cdots & \xleftarrow{E_{n-2}} & \cdot & \xleftarrow{E_{n-1}} & \cdot \end{array}$$

We have reduced all the matrices to pivot matrix form.

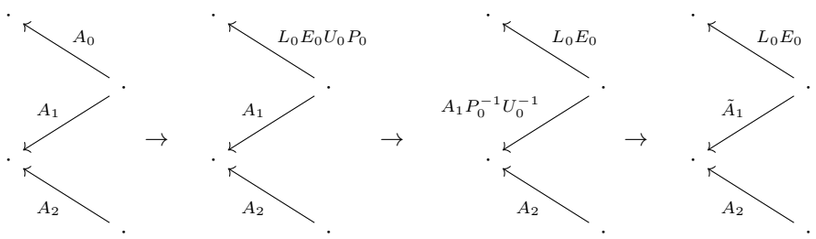
4.4 Alternating arrow directions

In the general A-type Quiver diagram, the arrows can be in either direction. So far we have seen an algorithm that works when all the arrows are in the same direction.

To illustrate how the algorithm would work with arbitrary arrow directions, consider the following zigzag quiver.



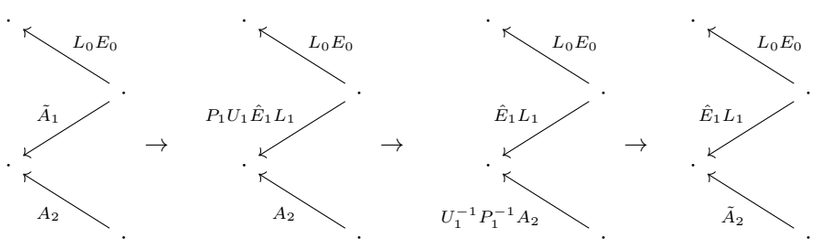
The above transformation might be easier to see if the diagram was in the following form,



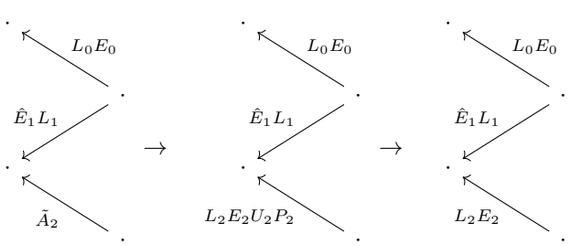
In the next step, as the arrow is reversed we cannot use the LE_LUP factorization. This would result in matrices that cannot be commuted during the second sweep. To handle this case, we use the $PU\hat{E}_LL$ factorization instead.

$$\tilde{A}_1 = P_1U_1\hat{E}_1L_1$$

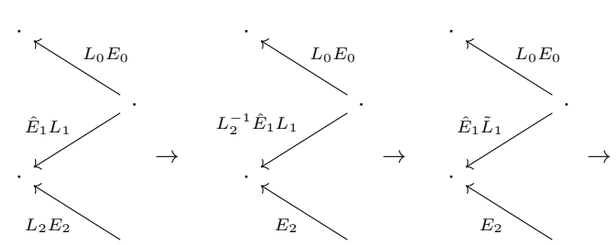
Applying matrix passing to move the factored matrices to the next edge, we get

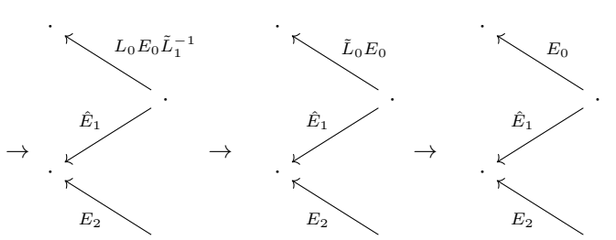


For the last edge, we can again apply the LE_LUP factorization,



Now we can perform the reverse sweep as follows,





During the reverse sweep we used the following commutation relations

$$L_2^{-1}\hat{E}_1 L_1 = \hat{E}_1\tilde{L}_1$$

$$L_0 E_0 \tilde{L}_1^{-1} = \tilde{L}_0 E_0$$

Thus we have reduced all matrices to pivot matrices, the barcodes can be directly read of from them.

Here we had to use a different factorization depending on the arrow direction, we can now use this to generalize the algorithm to a type-A quiver with arbitrary arrow directions.

4.5 General Sequential Quiver Algorithm

For arbitrary directions of the arrows, as long as we use the correct factorization for each of the arrow directions, we can use the shape commutation relations in the reverse sweep to reduce all the matrices to echelon pivot form.

Algorithm 7 Obtain Barcode factorization of type A quiver: Rightward-initial

Input: Type A quiver.
Result: Barcode form of quiver.
for forward pass **do**
  if $\leftarrow$ **then**
    Apply LE_LUP Factorization
    Matrix pass UP factors
  else
    Apply $PU\hat{E}_L L$ Factorization
    Matrix pass PU factors
  end if
end for
Matrix pass L factor on the last edge
for backward pass **do**
  if $\leftarrow$ **then**
    Commute $L_1E_LL_2 = \tilde{L}_1E_L$
  else
    Commute $L_1\hat{E}_LL_2 = \hat{E}_L\tilde{L}_2$
  end if
end for
Matrix pass the remaining L factor on the first edge.

We can also initiate the first sweep from the right to the left, to get a leftward initial algorithm. All the examples shown above initiate the first sweep from the left. For a general initial sweep direction and arrow direction, The tables fig. 3 and fig. 4 specify the factorization and commutation relation to use.

	Rightward Initial	Leftward Initial
$\leftarrow$	LE_LUP	PLE_UU
$\rightarrow$	$PU\hat{E}_LL$	$U\hat{E}_ULP$

Figure 3: The factorization to use for the first sweep.

	Rightward Initial	Leftward Initial
$\leftarrow$	$U_1E_UU_2 = E_U\tilde{U}_2$	$L_1E_LL_2 = \tilde{L}_1E_L$
$\rightarrow$	$U_1\hat{E}_UU_2 = \tilde{U}_1\hat{E}_U$	$L_1\hat{E}_LL_2 = \hat{E}_L\tilde{L}_2$

Figure 4: The commutation to use for the second sweep.

We note that the commutation relations established in Section 4.2 do not change the nonzero structure of the E matrices. Thus, it is possible to extract the barcode without performing the backward pass of Algorithm 7 if one does not care to form the change of basis.

4.6 Parallel Quiver Algorithm

We can also parallelize the algorithm for computing the barcode factorization of a quiver. The protocol of matrix factorizations and matrix passing will be different.

4.6.1 LQU Factorization

The LQU factorization is different from the triangular factorizations introduced before. It does not perform any pivoting and therefore there is no auxiliary permutation matrix produced. Instead, we

sacrifice structure in the pivot matrix and obtain a general pivot matrix Q as opposed to echelon pivot matrices.

Algorithm 8 LQU factorization

Input: $m \times n$ matrix A
Result: Factorization $A = LQU$
$L = I_m$
$Q = A$
$U = I_n$
$j = 1$
while $j <= n$ **do**
  if column j has a non-zero in a non-pivot row, let the first such row be i **then**
    Zero out all elements in non-pivot rows in column j below i
    Record row operations in L
    Mark i as pivot row
    $j = j + 1$
  else
    $j = j + 1$
  end if
end while
$i = 1$
while $i <= m$ **do**
  if row i is a pivot-row with pivot at j **then**
    Zero out all elements after j
    Record column operations in U
    $i = i + 1$
  else
    $i = i + 1$
  end if
end while

Proof of Correctness: In order to see that the above algorithm is correct, we first note that both the row operations and column operations are triangular, i.e. row i is used to eliminate rows at positions greater than i and column j is used to eliminate columns at positions greater than j. Thus the recorded L and U matrices are indeed lower and upper triangular respectively.

Now it is left to prove that the resultant matrix Q, has pivot structure. If we prove that the only non-zeros at the end of the algorithm are the pivots then we are done, as pivots are chosen such that no two of them share a row or column. Let us prove this by contradiction, suppose there is a non-zero that was not eliminated at the end of the algorithm. It has to be either in a non-pivot row or its column position is before a pivot in a pivot row, otherwise it would have been eliminated by the column operations. Such an element cannot exist as it should have been eliminated by row operations by a pivot above it in its column. The pivot cannot be below as it would imply that we did not pick the first non-zero in a non-pivot row when choosing the pivot for this column.

4.6.2 E Matrix transformations

In this section we will see how we can factorize any pivot matrix Q into a permutation and an echelon pivot matrix

Proposition 4.6.1. *Given any pivot matrix Q, we can rewrite it as the following*

$$Q = E_L P \tag{29}$$
$$Q = P E_U \tag{30}$$
$$Q = \hat{E}_U P \tag{31}$$
$$Q = P \hat{E}_L \tag{32}$$
$$\tag{33}$$

where P is an appropriate permutation matrix.

Proof. We apply the LE_LUP, PLE_UU, $U\hat{E}_ULP$ and $PU\hat{E}_LL$ factorizations to Q and note that the triangular matrices are just the identity matrices. This is because the triangular matrices are produced to eliminate one entry with another entry in the same row or column, but such a situation cannot occur in a pivot matrix Q, so only permutation operations are performed in the factorization, resulting in a permutation matrix and an echelon pivot matrix. □

4.6.3 Divide and conquer

In this section, we will show how we can divide a type-A quiver into two parts and perform the computation in parallel for each of the parts. The results can then be combined to give the full barcode factorization of the entire quiver. Consider a quiver Q_γ with general arrow directions,

$$\cdot \xleftarrow{A_0} \cdot \xleftarrow{A_1} \cdots \xleftarrow{A_{n-1}} \cdot$$

We will divide it into two parts at position m, to give us two sub-quivers Q_α and Q_β

$$\cdot \xleftarrow{A_0} \cdots \xleftarrow{A_{m-1}} \cdot \xleftarrow{A_m} \cdots \xleftarrow{A_{n-1}} \cdot$$

Next we will introduce three auxiliary edges containing identity maps to aide us in the computations by acting as a place holder for matrices.

$$\cdot \xleftarrow{I} \cdot \xleftarrow{A_0} \cdots \xleftarrow{A_{m-1}} \cdot \xleftarrow{I} \cdot \xleftarrow{A_m} \cdots \xleftarrow{A_{n-1}} \cdot \xleftarrow{I} \cdot$$

We can now perform two versions of the sequential algorithm in parallel. For quiver Q_α, we will use the rightward-initial algorithm (Algorithm 7). Notice that the terminal matrices are collected in the auxiliary edges, they will be important when we merge the results of the two sub-quivers.

$$\cdot \xleftarrow{L_\alpha} \cdot \xleftarrow{E_0} \cdots \xleftarrow{E_{m-1}} \cdot \xleftarrow{U_\alpha P_\alpha} \cdot \xleftarrow{A_m} \cdots \xleftarrow{A_{n-1}} \cdot \xleftarrow{I} \cdot$$

For the quiver Q_β, we will use the leftward-initial sequential algorithm. This allows us to collect both the permutation matrices in the center auxiliary edge.

$$\cdot \xleftarrow{L_\alpha} \cdot \xleftarrow{E_0} \cdots \xleftarrow{E_{m-1}} \cdot \xleftarrow{U_\alpha P_\alpha P_\beta L_\beta} \cdot \xleftarrow{E_m} \cdots \xleftarrow{E_{n-1}} \cdot \xleftarrow{U_\beta} \cdot$$

We can now multiply out the matrices in the center auxiliary edge and perform an LQU factorization using Algorithm 8

$$U_\alpha P_\alpha P_\beta L_\beta = C_\gamma$$

Since all the matrices $U_\alpha, P_\alpha, P_\beta, L_\beta$ are invertible, the pivot matrix Q_γ, will be full rank, and thus turn out to be a permutation matrix P_γ

$$C_\gamma = L_\gamma Q_\gamma U_\gamma = L_\gamma P_\gamma U_\gamma$$

$$\cdot \xleftarrow{L_\alpha} \cdot \xleftarrow{E_0} \cdots \xleftarrow{E_{m-1}} \cdot \xleftarrow{L_\gamma P_\gamma U_\gamma} \cdot \xleftarrow{E_m} \cdots \xleftarrow{E_{n-1}} \cdot \xleftarrow{U_\beta} \cdot$$

The matrices E_0 to E_{m-1} were produced by the rightward-intial algorithm, so they are of shape E_L or $\hat{E}_L$ depending on the arrow directions. They can be used to commute the L_γ factor all the way to the left. This process is very similar to the second sweep of the rightward-initial algorithm. Similarly, the matrices E_m to E_{n-1} are of shape E_U or $\hat{E}_U$. These matrices can be used to commute the U_γ factor towards the right in a manner similar to the second sweep of the leftward-initial algorithm.

$$\cdot \xleftarrow{L_\alpha \hat{L}_\gamma} \cdot \xleftarrow{E_0} \cdots \xleftarrow{E_{m-1}} \cdot \xleftarrow{P_\gamma} \cdot \xleftarrow{E_m} \cdots \xleftarrow{E_{n-1}} \cdot \xleftarrow{\hat{U}_\gamma U_\beta} \cdot$$

These propagated factors can now be multiplied and we get one lower triangular factor $\tilde{L}_\gamma$ on the left auxiliary edge and one upper triangular factor $\tilde{U}_\gamma$ on the right auxiliary edge leaving a permuation matrix P_γ in the center auxiliary edge. At this stage we can think of the result as the "LQU" factorization of the quiver Q_γ

$$\cdot \xleftarrow{\tilde{L}_\gamma} \cdot \xleftarrow{E_0} \cdots \xleftarrow{E_{m-1}} \cdot \xleftarrow{P_\gamma} \cdot \xleftarrow{E_m} \cdots \xleftarrow{E_{n-1}} \cdot \xleftarrow{\tilde{U}_\gamma} \cdot$$

At this stage, we technically have a valid barcode factorization and if this is the entire quiver we could stop here, but since we want to apply this algorithm recursively, we want to convert this into the result of either a leftward-intial or rightward-initial algorithm. This can be done by propagating the permutation matrix P_γ either leftwards or rightwards using the E matrix transformations discussed in Section 4.6.2. If we wish to obtain the result of the rightward-initial algorithm, then we propagate right. Note this transforms the E_U and $\hat{E}_U$ matrices in Q_β to E_L and $\hat{E}_L$. We will denote the transformed matrices by $\tilde{E}_m$ to $\tilde{E}_{n-1}$.

$$\cdot \xleftarrow{\tilde{L}_\gamma} \cdot \xleftarrow{E_0} \cdots \xleftarrow{E_{m-1}} \cdot \xleftarrow{I} \cdot \xleftarrow{\tilde{E}_m} \cdots \xleftarrow{\tilde{E}_{n-1}} \cdot \xleftarrow{\tilde{P}_\gamma \tilde{U}_\gamma} \cdot$$

We can now drop the auxiliary identity map in the center, to obtain the rightward-initial merge.

$$\cdot \xleftarrow{\tilde{L}_\gamma} \cdot \xleftarrow{E_0} \cdots \xleftarrow{E_{m-1}} \cdot \xleftarrow{\tilde{E}_m} \cdots \xleftarrow{\tilde{E}_{n-1}} \cdot \xleftarrow{\tilde{P}_\gamma \tilde{U}_\gamma} \cdot$$

Alternatively, we can also propagate the permutation matrix leftwards, to obtain the result of the leftward intial algorithm. This would transform the E_L and $\hat{E}_L$ matrices into E_U and $\hat{E}_U$ in Q_α

$$\cdot \xleftarrow{\tilde{L}_\gamma \tilde{P}_\gamma} \cdot \xleftarrow{\tilde{E}_0} \cdots \xleftarrow{\tilde{E}_{m-1}} \cdot \xleftarrow{I} \cdot \xleftarrow{E_m} \cdots \xleftarrow{E_{n-1}} \cdot \xleftarrow{\tilde{U}_\gamma} \cdot$$

$$\cdot \xleftarrow{\tilde{L}_\gamma \tilde{P}_\gamma} \cdot \xleftarrow{\tilde{E}_0} \cdots \xleftarrow{\tilde{E}_{m-1}} \cdot \xleftarrow{E_m} \cdots \xleftarrow{E_{n-1}} \cdot \xleftarrow{\tilde{U}_\gamma} \cdot$$

Note that the results are not exactly equal to the result you would obtain from applying either a rightward-intial or leftward-intial algorithm on the entire quiver. While the echelon matrices are of the right shape, the terminal matrices $\tilde{P}_\gamma \tilde{U}_\gamma$ or $\tilde{L}_\gamma \tilde{P}_\gamma$ appear in reversed order. However this does not matter if this is used to merge barcode factorizations at a higher level. At a higher level of the recursion, we would have the following product in the center auxiliary edge,

$$\cdot \xleftarrow{L_\alpha} \cdot \xleftrightarrow{E_0} \cdots \xleftrightarrow{E_{m-1}} \cdot \xleftarrow{P_\alpha U_\alpha L_\beta P_\beta} \cdot \xleftrightarrow{E_m} \cdots \xleftrightarrow{E_{n-1}} \cdot \xleftarrow{U_\beta} \cdot$$

They can still be multiplied out and the LQU factorization can be performed without affecting the rest of the algorithm.

$$P_\alpha U_\alpha L_\beta P_\beta = C_\gamma$$
$$C_\gamma = L_\gamma P_\gamma U_\gamma$$

Thus we have seen how a divide an conquer algorithm can be used to compute the barcode factorization of a quiver in two parts. We can now apply this recursively until the size of the quiver is small enough that it is more efficient to apply one of the sequential algorithms.

4.7 Correctness and Uniqueness of the Barcode Factorization

We have presented an algorithm to produce a barcode matrix Λ from the companion matrix A of a finite dimensional quiver representaiton of type A_n. In this section we'll show the algorithm produces a quiver isomorphism $A \cong \Lambda$, and that Λ uniquely defines the quiver isomorphism class.

Proposition 4.7.1. *Every finite dimensional quiver representation of type A_n has a barcode form.*

Proof. This follows immediately from the existence of the LEUP and PLEU factorizations at each step, and the commutations established. □

Proposition 4.7.2. *The barcode factorization $A = B\Lambda B^{-1}$ is a quiver isomorphism.*

Proof. We have shown that the factorizations at each step exist, and by Lemma 3.5.1 each matrix passing step is a quiver isomorphism. □

The following theorem recasts the known fact that a barcode determines the isomorphism class of a zigzag or persistence module in terms of the barcode form.

Theorem 4.7.3. *The barcode form Λ of a quiver representation of type A_n uniquely determines its isomorphism class.*

Proof. If two quivers represented by companion matrices A, A' have the same barcode factorization, then $\Lambda = B^{-1}AB = (B')^{-1}A'B'$, so $A' = B'B^{-1}AB(B')^{-1}$. Thus, the quivers are isomorphic via the isomorphsim represented by $B'B^{-1}$.

Now, suppose that two quivers represented by A, A' are isomorphic, that is, $A' = MAM^{-1}$ for some quiver isomorphism matrix M. In general, the barcode factorization algorithm will play out very differently on the two companion matrices, with different L, E, U, and P factors at each step, and it isn't obvious that the intervals recovered by Section 3.3 will be the same.

Comment that this is already known, but we include for clarity. Let $A = B\Lambda B^{-1}$ and $A' = B'\Lambda'(B')^{-1}$ be barcode factorizations. Then

$$\Lambda = B^{-1}M^{-1}B' \ \Lambda' \ (B')^{-1}MB$$

Let $v \in V_b$ be the basis element for an indecomposable $I[b,d]$ in A, and $v' = Mv$. Note that v' may be a mix of basis elements for indecomposables of A'. We can now trace v through the quiver.

$$\begin{array}{ccccc} V_i & \xleftrightarrow{A_i} & V_{i+1} & \xleftrightarrow{A_{i+1}} & \cdots \\ \downarrow{\scriptstyle M_i} & & \downarrow{\scriptstyle M_{i+1}} & & \\ V_i' & \xleftrightarrow{A_i'} & V_{i+1}' & \xleftrightarrow{A_i'} & \cdots \end{array}$$

There are two cases: either $V_i \to V_{i+1}$, in which case either both the image $A_i v$ and the image $A_i' v'$ are nonzero, or both images must be nonzero (otherwise the diagram does not commute and M is not a quiver isomorphism). In the other case, $V_{i+1} \to V_i$, in which case either both v and v' are images of elements in V_{i+1}^*, or both are not images. Thus, since we can propagate v from V_b to V_d, its image v' must propagate from V_b' to V_d', and if we consider v' in the indecomposable basis, there must be an element for the indecomposbable $I[b,d]$. We choose to associate v' with this element. Now, suppose that w is also a basis element for an indecomposable $I[b,d]$ in A, and is associated with the same basis element for $I[b,d]$ in A'. Then $M(v-w)$ would have a zero coefficient for an element of $I[a,b]$, and could not be propagated the full length of the bar in A', so we must have $v = w$ for the diagram to commute. Thus, there the indecomposables of A map injectively to the indecomposables of A'.

We can then use an identical argument to see that the inverse map M^{-1} maps the indecomposbales of A' inectively to indecomposables of A. Thus, there is a bijection between the indecomposables, and the barcode forms must be identical (up to permutation). □

Theorem 4.7.3 is equivalent to the statement that the indecomposables of A_n-type quivers are interval indecomposables. Note that the uniqueness of the Jordan normal form for companion matrices is a special case of Theorem 4.7.3 that applies to persistence-type quivers, following Section 3.4.

A Factorizations, Commutations, and Algorithms

This section contains details on matrix factorizations, some commutation relations, and algorithms for computing them.

A.1 Commutation Relations

First, we will consider commutations of permutations with certain block triangular matrices.

$$\begin{bmatrix} I & \\ & P \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ & I \end{bmatrix} = \begin{bmatrix} U_{11} & U_{12}P^T \\ & I \end{bmatrix} \begin{bmatrix} I & \\ & P \end{bmatrix} \tag{34}$$

Proof. It is easily verified that both products produce the matrix

$$\begin{bmatrix} U_{11} & U_{12} \\ & P \end{bmatrix}$$

□

Computation of the commutation is straightforward. All that needs to be done is apply the column permutation P^T to the U_{12} block. A similar relation can be shown for block lower triangular matrices.

A.2 Leftward-initial barcode factorization algorithm

This is the explciit specification of the sequential algorithm for computing the barcode factorization of a quiver. The factorizations and shape commutation results used are different compared to the rightward-initial algorithm.

Algorithm 9 Obtain Barcode factorization of type A quiver: Leftward-initial

Input: Type A quiver.
Result: Barcode form of quiver.
for forward pass **do**
  if $\leftarrow$ **then**
    Apply PLE_UU Factorization
    Matrix pass PL factors
  else
    Apply $U\hat{E}_ULP$ Factorization
    Matrix pass LP factors
  end if
end for
Matrix pass U factor on the last edge
for backward pass **do**
  if $\leftarrow$ **then**
    Commute $U_1E_UU_2 = E_U\tilde{U}_2$
  else
    Commute $U_1\hat{E}_UU_2 = \tilde{U}_1\hat{E}_U$
  end if
end for
Matrix pass the remaining U factor on the first edge.

B The Mapping Cylinder

The mapping cylinder can be viewed as the pushforward of the map $f : \mathcal{X} \to \mathcal{Y}$ and the inclusion of $\mathcal{X} \to \mathcal{X} \times I$.

$$\begin{array}{ccc} \mathcal{X} & \xrightarrow{f} & \mathcal{Y} \\ \downarrow\simeq & \searrow & \downarrow\simeq \\ \mathcal{X} \times I & \longrightarrow & \operatorname{Cyl} f \end{array}$$

We'd like to show that the induced maps on homology do indeed commute. Because $\mathcal{Y}$ is a deformation retract of $\operatorname{Cyl} f$, this would imply that the induced map on homology from the inclusion $\mathcal{X} \hookrightarrow \operatorname{Cyl} f$ is the same as the induced map on homology of $f : \mathcal{X} \to \mathcal{Y}$.

Definition B.0.1. *We say two chain maps $F_*, G_* : C_* \to D_*$ are* homotopic *if there exists a* chain homotopy *consisting of maps $M_k : C_k \to D_{k+1}$ so that*

$$\partial_k^D M_k + M_{k-1}\partial_k^C = F_k - G_k$$

A standard result in homological algebra is that if F_* and G_* are homotopic, then the induced maps on homology $\tilde{F}_*, \tilde{G}_* : H_*(C) \to H_*(D)$ are isomorphic [20].

The inclusion $G_* : C_*(X) \to C_*(\mathrm{Cyl}\, f)$ traversing the left and bottom arrows of the diagram is given by $G(x) = x \times \{0\}$, and the map through Y $G_* : C_*(\mathcal{X}) \to C_*(\mathrm{Cyl}\, f)$ traversing the top and right arrows of the diagram is given by $F(x) = f(x) \times \{1\}$, where if $f(x) = \sum_i \alpha_i y_i$, then $f(x) \times \{1\} = \sum_i \alpha_i (y_i \times \{1\})$.

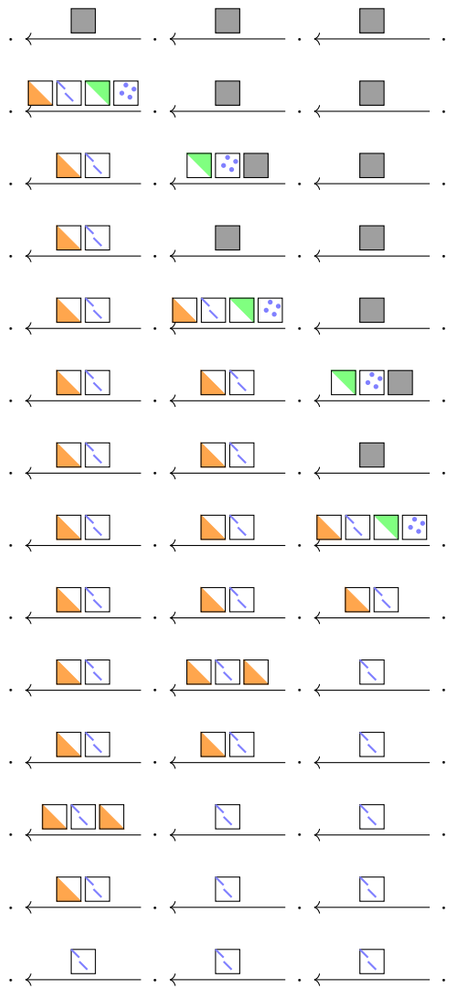
We now define maps $M_k : C_k(\mathcal{X}) \to C_{k+1}(\mathrm{Cyl}\, f)$ mapping x to the image of $x \times [0,1]$ in the cylinder. Explicitly, the basis element $x \in C_k(\mathcal{X})$ is mapped to the basis element $x \times [0,1]/f \in C_{k+1}(\mathrm{Cyl}\, f)$ with boundary $-(\partial x) \times [0,1]/f + f(x) \times \{1\} - x \times \{0\}$, appearing in the left block of Equation (17). Now, we simply verify

$$
\begin{aligned}
&\partial_k^{\mathrm{Cyl}\, f} M_k(x) + M_{k-1} \partial_k^X(x) \\
&= -(\partial x) \times [0,1]/f + f(x) \times \{1\} - x \times \{0\} + M_{k-1} \partial_k^X(x) \\
&= -(\partial x) \times [0,1] + f(x) \times \{1\} - x \times \{0\} + (\partial x) \times [0,1]/f \\
&= f(x) \times \{1\} - x \times \{0\} \\
&= F(x) - G(x)
\end{aligned}
$$

so M_* is a chain homotopy. Thus the induced maps on homology are isomorphic: $\tilde{F}_* \simeq \tilde{G}_*$.

C Diagrams

In this section, we include a pictoral representation of the operations described in Section 4.3. The diagram only depicts the shapes of the matrices at each step so it is clear what operations are involved in each step.



Acknowledgements

GC and BN are supported by Altor Equity Partners AB through Unbox AI (unboxai.org). BN was also supported by the US Department of Energy, Contract DE-AC02-76SF00515. AD was supported by Burt and Deedee McMurtry Fellowship.

Conflict of interest

The authors declare that they have no conflict of interest.

References

[1] Gudhi: Geometric understanding in higer dimensions. https://gudhi.inria.fr/.

[2] Bauer, U. Ripser: efficient computation of vietoris-rips persistence barcodes, Aug. 2019. Preprint.

[3] Bauer, U., Kerber, M., Reininghaus, J., and Wagner, H. Phat persistent homology algorithms toolbox. *Journal of Symbolic Computation 78* (2017), 76–90.

[4] Boissonnat, J.-D., Pritam, S., and Pareek, D. Strong collapse for persistence. *arXiv:1809.10945 [cs]* (2018).

[5] Carlsson, G. Topology and data. *Bulletin of the American Mathematical Society 46*, 2 (2009), 255–308.

[6] Carlsson, G., and de Silva, V. Zigzag persistence. *Foundations of Computational Mathematics 10*, 4 (2010), 367–405.

[7] Carlsson, G., de Silva, V., and Morozov, D. Zigzag persistent homology and real-valued functions. ACM Press, p. 247.

[8] Carlsson, G., and Zomorodian, A. The theory of multidimensional persistence. *Discrete & Computational Geometry 42*, 1 (2009), 71–93.

[9] Cipra, B. A. The best of the 20th century: Editors name top 10 algorithms. *SIAM News 33*, 4 (2000).

[10] Derksen, H., and Weyman, J. Quiver representations. *Notices of the American Mathematical Society 52*, 2 (2005).

[11] Dey, T. K., Fan, F., and Wang, Y. Computing topological persistence for simplicial maps. In *Proceedings of the Thirtieth Annual Symposium on Computational Geometry* (2014), SOCG'14, ACM, pp. 345:345–345:354. event-place: Kyoto, Japan.

[12] Dotko, P., and Wagner, H. Simplification of complexes of persistent homology computations. *Homology, Homotopy and Applications 16*, 1 (2014), 49–63.

[13] Edelsbrunner, H., and Harer, J. *Computational Topology - an Introduction.* American Mathematical Society, 2010.

[14] Edelsbrunner, H., Letscher, D., and Zomorodian, A. Topological persistence and simplification. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on* (2000), IEEE, pp. 454–463.

[15] Eilenberg, S., and MacLane, S. General theory of natural equivalences. *Transactions of the American Mathematical Society 58* (1945), 231–294.

[16] Eilenberg, S., and Steenrod, N. E. *Foundations of Algebraic Topology.* Princeton Mathematical Series. 1952.

[17] Gabriel, P. Unzerlegbare darstellungen. i. *Manuscripta Math. 6* (1972), 71–103.

[18] Gelfand, S., and Maninin, Y. *Methods of Homological Algebra*, 2 ed. Springer, 2003.

[19] Golub, G. H., and Van Loan, C. F. *Matrix Computations*, 4th ed. The Johns Hopkins University Press, Baltimore, 2013.

[20] Hatcher, A. *Algebraic Topology.* Cambridge University Press, 2001.

[21] Henselman, G., and Ghrist, R. Matroid Filtrations and Computational Persistent Homology. *ArXiv e-prints* (June 2016).

[22] Householder, A. S. *The Theory of Matrices in Numerical Analysis.* Blaisdell Pusblishing Company, 1964.

[23] Kerber, M., and Schreiber, H. Barcodes of towers and a streaming algorithm for persistent homology. *Discrete & Computational Geometry 61*, 4 (2019), 852–879.

[24] Lawson, C. L., Hanson, R. J., Kincaid, D. R., and Krogh, F. T. Basic linear algebra subprograms for fortran usage. *ACM Trans. Math. Softw. 5*, 3 (1979), 308–323.

[25] Lewis, R., and Morozov, D. Parallel computation of persistent homology using the blowup complex. ACM Press, pp. 323–331.

[26] Lim, L.-H. Hodge laplacians on graphs.

[27] MacLane, S. *Homology*. Springer-Verlag, 1963.

[28] Maria, C., and Oudot, S. Y. Zigzag persistence via reflections and transpositions. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, Proceedings. Society for Industrial and Applied Mathematics, 2014, pp. 181–199.

[29] Maria, C., and Schreiber, H. Discrete morse theory for computing zigzag persistence. In *Algorithms and Data Structures* (2019), Z. Friggstad, J.-R. Sack, and M. R. Salavatipour, Eds., Lecture Notes in Computer Science, Springer International Publishing, pp. 538–552.

[30] Mischaikow, K., and Nanda, V. Morse theory for filtrations and efficient computation of persistent homology. *Discrete & Computational Geometry 50*, 2 (2013), 330–353.

[31] Morozov, D. Dionysus2. https://www.mrzv.org/software/dionysus2/.

[32] Otter, N., Porter, M. A., Tillmann, U., Grindrod, P., and Harrington, H. A. A roadmap for the computation of persistent homology. *EPJ Data Science 6*, 1 (2017).

[33] Oudot, S. Y. *Persistence Theory: From Quiver Representations to Data Analysis*, vol. 209 of *Mathematical Surveys and Monographs*. American Mathematical Society, 2015.

[34] Oudot, S. Y., and Sheehy, D. R. Zigzag zoology: Rips zigzags for homology inference. *Foundations of Computational Mathematics 15*, 5 (2015), 1151–1186.

[35] Skraba, P., Thoppe, G., and Yogeshwaran, D. Randomly weighted $d-$ complexes: Minimal spanning acycles and persistence diagrams. *arXiv preprint arXiv:1701.00239* (2017).

[36] Wilkerson, A. C., Chintakunta, H., and Krim, H. Computing persistent features in big data: A distributed dimension reduction approach. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2014), pp. 11–15. ISSN: 1520-6149, 2379-190X.

[37] Zhang, S., Xiao, M., Guo, C., Geng, L., Wang, H., and Zhang, X. HYPHA: a framework based on separation of parallelisms to accelerate persistent homology matrix reduction. In *Proceedings of the ACM International Conference on Supercomputing - ICS '19* (2019), ACM Press, pp. 69–81.

[38] Zomorodian, A., and Carlsson, G. Computing persistent homology. *Discrete & Computational Geometry 33*, 2 (2005), 249–274.