Discrete Hodge Theory on Graphs: A Tutorial

Hodge theory enables a graph decomposition into the gradient, solenoidal, and harmonic components that are useful for solving certain ranking and approximation problems.

Hodge theory provides a unifying view of the various line, surface, and volume integrals that appear in physics and engineering applications. Hodge theory on graphs develops discrete versions of the differential forms found in the continuous theory and enables a graph decomposition into gradient, solenoidal, and harmonic components. Interpreted via similarity to gradient, curl, and Laplacian operators on vector fields, these components are useful for solving certain problems that arise naturally in a graph context. This tutorial develops the rudiments of discrete Hodge theory and provides several example applications.

A Motivating Example
In academic life, there arise from time to time certain amenities—such as sabbaticals, research grants, and teaching load reductions. Because candidates inevitably outnumber prizes, a lollipop committee is appointed to rank the applicants. In its first meeting, the committee finds a large stack of dossiers, each proclaiming a candidate’s merits, each written in a different style, and each emphasizing different accomplishments. Committee members, alone or in subgroups, examine the evidence and construct rankings. The process consumes several weeks and, of course, produces conflicting judgments. Reconciliation meetings then sort out the conflicts in an ad hoc fashion. The process is reasonably satisfactory and certainly sanctified by tradition. Given variations in human judgment, conflicts are expected, and the faculty committee serves to extract a group consensus from the individual voices.

Improvised human judgment, informed by a professional appreciation of the context, suffices to rank short lists, say, of length two or three. We typically assemble longer rankings by repeatedly positioning a new entry in an already ranked sublist, a process that frequently involves reassessments of neighboring candidates. An algorithm to assemble a global ranking from a collection of independent short subrankings is clearly needed.

Specifically, we propose that the committee generate a random but representative collection of three-candidate triads. On receiving a triad allotment, a committee member treats each triad as an independent problem. That is, considering a particular triad, a judge need not consider nor even remember judgments that he, she, or any other member has pronounced on other triads. He or she simply rates the triad candidates as top, middle, and bottom. Triad overlaps are expected, and therefore judgment conflicts persist. However, an algorithm can reconcile these conflicts into a global ranking that best approximates the triad judgments.

The reconciliation algorithm is, of course, our major topic here, and the following sections clarify...
the sense in which the computed global ranking best approximates the trial rankings. Before considering these details, however, we construct some artificial data that illustrate a typical committee’s judgment conflicts. We envision a 20-candidate field, where candidate 1 is the strongest, candidate 2 is next, and so forth, with candidate 20 being the weakest. We then generate typical committee trial data that somewhat confuses the true ranking and evaluate the algorithm on how well it approximates the correct ranking from this data.

On trial \((i < j < k)\), we simulate a judgment by independently sampling normal distributions \(N(i, \sigma_i)\), \(N(j, \sigma_j)\), and \(N(k, \sigma_k)\), obtaining values \(X_i, X_j,\) and \(X_k\). We note the permutation \((u, v, w)\) of \((i, j, k)\) such that \(X_u < X_v < X_w\) and declare the local ranking: \(u\) is strongest, \(v\) is in the middle, and \(w\) is weakest. If \(u < v < w\), this assignment discovers the true relative strengths of \((i, j, k)\). In this case, \(u = i, v = j, w = k,\) and the samples are sufficiently close to their means that the relative order \(X_i < X_j < X_k\) preserves the order \(i < j < k\). The variance parameter, \(\sigma^2\), increases or decreases the probability that samples constitute a different order than the normal centers.

The probability of an erroneous judgment on pair \(i < j\) is \(P(X_i \geq X_j)\). Noting that \(X_i - X_j\) is normally distributed with mean \((i-j)\) and variance \(2\sigma^2\), we obtain

\[
P(X_i \geq X_j) = P(X_i - X_j \geq 0)
\]

\[
= P \left( \frac{X_i - X_j - (i-j)}{\sigma \sqrt{2}} \geq \frac{-(i-j)}{\sigma \sqrt{2}} \right)
\]

\[
= 1 - \Phi \left( \frac{j-i}{\sigma \sqrt{2}} \right),
\]

where \(\Phi\) is the cumulative distribution function of the standard normal \(N(0, 1)\). Table 1 exhibits error probabilities for various strength differences and \(\sigma\) values. Clearly, candidates with large strength differences can nevertheless be misjudged with significant probability. For a given strength difference, misjudgments are more probable with larger \(\sigma\), which corresponds to greater overlap of the distributions and therefore simulates a less competent committee.

We propose ranking candidate triads rather than pairs. In this case, a misjudgment occurs when \(i < j < k\) holds but \(X_i < X_j < X_k\) fails. Because either \(X_i \geq X_j\) or \(X_j \geq X_k\) implies \(X_i < X_j < X_k\) fails, we conclude that the probability of a misjudgment is at least max \(P(X_i \geq X_j), P(X_j \geq X_k)\). Therefore if we substitute max \(\max \{j-i, k-j\}\) for \(j-i\) in Table 1, the corresponding line gives a lower bound for the misjudgment probability of triad \((i < j < k)\).

Returning to our 20-candidate context, we maintain that a random 10 percent of the \(\binom{20}{3}\) triads, provided they cover the candidate field, can give a good consensus ranking. The committee tasks are then to choose a random selection of 114 triads, ensuring that some chain of triad entries connects each candidate pair; rank the triads independently; and algorithmically obtain the global ranking most consistent with the independent judgments. Table 2 reports the algorithm’s performance on simulated data with parameter \(\sigma = 4\). While a perfect ranking would rank candidates 1 through 20 in increasing order, the average algorithm misplacement of 1.10 positions. No candidate is misplaced more than four positions, and eight of the 20 are placed exactly. The list’s top quarter contains all candidates who should be in that quarter; other quarters also exhibit near-perfect selection. In general, separated list segments are in perfect order—that is, for example, each candidate in the highest-ranking quarter is accurately judged of higher merit than any candidate in the third quarter.

The following sections elaborate on the theory behind Hodge decomposition of graphs and illustrate its applicability, returning to the lollipop problem in the last section. The tutorial exposition here assumes no familiarity with continuous Hodge theory. A deeper mathematical treatment appears elsewhere, as do similar approaches with different derivations and application areas.

**Graphs as Simplicial Complexes**

Traditionally, an undirected graph is a pair \((V, E)\), where \(V\) is a vertex set and \(E\) is an edge set. In this article, we deal only with finite graphs and fix \(|V| = n > 0\) throughout the discussion. Common practice denotes the vertices as \(v_1, v_2, \ldots, v_n\) and an edge between \(v_i\) and \(v_j\) as \(e_{ij}\). However, we wish to consider higher-dimensional structures and therefore need a more general notation. Consequently, we denote the vertex set by \((1), (2), \ldots, (n)\) and use the term 0-dimensional simplex for a vertex. In general, a \(k\)-dimensional simplex \((k\text{-simplex})\) of a graph is a fully connected subgraph on \(k + 1\) of the vertices.

A vertex is a 0-simplex, an edge is a 1-simplex, a triangle is a 2-simplex, and so forth. The graphs in Figure 1 all exhibit the same simplex collections. Specifically, each contains eight 0-simplices, the eight vertices. Each contains 13
of the possible \( \binom{8}{3} \) 1-simplices, the edges, with names (12), (18), (23), and so forth. We find five 2-simplices, the triangles (236), (345), (346), (356), and (456). However, it’s possible to have additional 2-simplices that aren’t boundary components of any triangle.

Clearly, each boundary component of a k-simplex constitutes a \((k - 1)\)-simplex. The four boundary faces of the 3-simplex (3456) are the 2-simplices (345), (346), (356), and (456). However, it’s possible to have additional 2-simplices that aren’t boundary components of a 3-simplex. In Figure 1a, for example, triangle (236) isn’t the boundary of any 3-simplex. Similarly, triangle boundaries constitute 1-simplices (edges), and there could exist additional edges that aren’t boundary components of any triangle.

When we specify a graph as \( G = (V, E) \), we emphasize only the lowest dimensional simplices. This specification determines the higher-dimensional components, so a definition need not explicitly reference them. However, our purpose here is better served by specifying \( G = (\Sigma_0, \Sigma_1, \ldots, \Sigma_k) \), where \( \Sigma_0 = V \), the collection of vertices, \( \Sigma_1 = E \), the collection of edges, \( \Sigma_2 \) is the collection of triangles, and so forth. In general, \( \Sigma_k \) contains fully connected vertex subsets of size \( k + 1 \). For any given graph on \( n \) vertices, there exists an integer \( K < n \) for which fully connected subgraphs over \( K + 1 \) vertices exists, but no larger vertex set participates in a fully connected subgraph. The collection \( S = \bigcup_{k=0}^{n} \Sigma_k \) is called a simplicial complex.

We used ascending vertex order to name the \( k \)-simplices in Figure 1. However, triangle (326) contains the same vertices and is therefore the same triangle as (236). In our computations, a given simplex may appear under any of its aliases, and we’ll be concerned with whether the alias \((i_1i_2\ldots i_k)\) is an even or odd permutation of the corresponding increasing sequence. We adopt the notation \( i_1\ldots i_k \) for an index sequence that might or might not be in ascending order, whereas \( i_1\ldots i_k \) means \( 1 \leq i_1 < \ldots < i_k \leq n \). To specify that \( j_1\ldots j_k \) is the increasing permutation of \( i_1\ldots i_k \), we use the abbreviation \( j_1\ldots j_k \sim i_1\ldots i_k \). The canonical name of a \( k \)-simplex is the increasing sequence \( j_1\ldots j_{k+1} \) of its vertices. When context suffices, we drop parentheses, as, for example, in a functional expression where \( f(i_1\ldots i_k) \) replaces \( f(j_1\ldots j_k) \).

An oriented \( k \)-simplex is one that has one of two possible orientations: positive or negative. We associate the positive orientation with the sequence \( i_1\ldots i_{k+1} \), and with any even index permutation of that canonical name. We associate the negative orientation with odd permutations. These orientations will play a role in the mappings from simplices to real numbers that we now introduce.

With \( G = (\Sigma_0, \Sigma_1, \ldots, \Sigma_k) \), let \( \langle \Sigma_k \rangle \) denote the real vector space spanned by \( \Sigma_k \). Thus,

\[
\langle \Sigma_k \rangle = \left\{ \sum_{i_1,j_1=1, i_1
ot= j_1}^{n} a_{i_1,j_1} \frac{(i_1\ldots j_{k+1})}{(j_1\ldots j_{k+1})} : a_{i_1,j_1} \in \mathbb{R} \right\} \quad (1)
\]

We’ll refer to the elements of \( \Sigma_k \) each indexed in ascending order, as the canonical basis for \( \langle \Sigma_k \rangle \). In Figure 1, \( \langle \Sigma_k \rangle \) contains vectors of the form \( a_{126}(236) + a_{145}(345) + a_{146}(346) + a_{156}(356) + a_{456}(456) \). The vector space \( \langle \Sigma_k \rangle \) has five dimensions, corresponding to the graph’s five triangles.
A vector \( f \in \Sigma_k \) defines a function \( f: \Sigma_k \to \mathbb{R} \) via \( f(i_1, \ldots, i_k) \), the coefficient of \( (i_1, \ldots, i_k) \). In \( \Sigma_k \), the vector representation is called a \((k + 1)\)-form and is traditionally written with wedge-products. As a 3-form, the example from \( \Sigma_2 \) is \( a_{236}^i dx^i \wedge dx^j \wedge dx^k + a_{146}^i dx^i \wedge dx^j \wedge dx^k + \ldots \). Such \( k \)-forms derive from the differential geometry context of continuous Hodge theory. The functional representation is called a cochain, a term deriving from cohomology studies. As a cochain \( f \), a term deriving from functional representation as an increasing sequence. Clearly \( f \) is the traditional dot-product of the coefficient vectors after adjusting each simplex to its representation as an increasing sequence. Clearly \( \langle \omega, \eta \rangle = \langle \eta, \omega \rangle \).

Returning now to the simplicial complex \( G = (\Sigma_0, \Sigma_1, \ldots, \Sigma_k) \), we define mappings from \( \Sigma_k \) to \( \Sigma_{k+1} \) and vice versa. For this purpose, two further notations are convenient: \( i_1, \ldots, i_k \) denotes \( i_1, i_2, \ldots, i_k \) with index \( j \) inserted in the proper position and \( i_1, \ldots, \hat{i}_j, \ldots, i_k \) with element \( i_j \) removed. For \( 0 \leq k < K \), define \( \delta_k: \Sigma_k \to \Sigma_{k+1} \) as the linear extension of

\[
\delta_k(i_1, \ldots, i_{k+1}) = \sum_{(j_1, \ldots, j_{k+1}) \in \Sigma_{k+1}} (j_1, \ldots, j_{k+1}) \rightarrow \sum_{(j_1, \ldots, j_{k+1}) \in \Sigma_{k+1}} \epsilon_{j_1 \ldots j_{k+1}} (i_1, \ldots, j_{k+1}). \tag{2}
\]

As an example of this computation, consider the vector \( \omega = a(36) + b(34) \in \Sigma_1 \) in Figure 1. We find

\[
\delta_1(\omega) = a\delta_1(36) + b\delta_1(34) = a(a(236) - 34.0(236)) + b(b(234) - 34.0(234)) = a(236) + b(346) - (a + b)346 < \Sigma_2 > \tag{3}
\]

For \( j_1 \ldots j_{k+1} \sim i_1 \ldots i_{k+1} \), linearity enables the following computation. Hence, we need not adjust the vector \( i_1, \ldots, i_{k+1} \) to an increasing sequence to apply the first construction in Equation 2:

\[
\delta_k(i_1, \ldots, i_{k+1}) = \sum_{(j_1, \ldots, j_{k+1}) \in \Sigma_{k+1}} \epsilon_{j_1 \ldots j_{k+1}} (i_1, \ldots, j_{k+1}) \rightarrow \sum_{(j_1, \ldots, j_{k+1}) \in \Sigma_{k+1}} \epsilon_{j_1 \ldots j_{k+1}} (i_1, \ldots, j_{k+1}).
\]

Figure 1. Widget flow among cities: (a) gradient, (b) nongradient, and (c) solenoidal.
Equation 2 obtains δ_k of a particular k-simplex as an algebraic combination of (k + 1)-simplices. By linearity, Equation 2 suffices to compute δ_k(ω) for any ω ∈ Σ_k. A computational formula for the functional representation is also useful; it gives the value assigned by δ_k(ω) to a particular (k + 1) simplex as a combination of the values assigned by ω to k-simplices. From Equation 2, a (k + 1)-simplex \( i_1...i_{k+2} \) appears as one of the summands in each

\[
 δ_k(i_1...i_{k+1}) \text{ expansion, in which case it appears with the sign } (-1)^{r+1};
\]

\[
 [δ_k(ω)](i_1...i_{k+1}) = \sum_{r=0}^{k} (-1)^r ω(i_1...i_r...i_{k+1}...).
\]  

Consider again ω = a(36) + b(34) ∈ Σ_k in Figure 1. As a function, \( ω : Σ_k \to \mathbb{R} \) is

\[
 ω(i_1i_2) = \begin{cases} 
 a; & (i_1i_2) = (36) \\
 b; & (i_1i_2) = (34) \\
 0; & \text{otherwise.}
\end{cases}
\]

Using Equation 4, we check that δ_k(ω)(36) = ω(36) - ω(34) = b - a. For 0 ≤ k < K, define δ_k : Σ_k → Σ_k via the inner product: \( δ(ω) \) is the linear map such that

\[
 <ω, δ_k(ω) \rangle = δ_k(ω), \eta >
\]

for all ω ∈ Σ_k. The discussion below shows how this expression unambiguously defines δ_k.

A linear map is determined by its action on each of the individual simplices in Σ_k+1. So, assuming

\[
 δ_k^*(i_1...i_{k+1}) = \sum_{(i_1...i_{k+1}) ∈ Σ_{k+1}} a_{i_1...i_{k+1}} (i_1...i_{k+1}),
\]

a formula for δ_k must calculate the unknown coefficients \( a_{i_1...i_{k+1}} \). For a fixed \( (i_1...i_{k+1}) \), Equation 6 gives

\[
 a_{i_1...i_{k+1}} = \langle (i_1...i_{k+1}), δ_k^*(i_1...i_{k+1}) \rangle = \langle δ_k(i_1...i_{k+1}), (i_1...i_{k+1}) \rangle = \sum_{(j_1...j_k) ∈ Σ_k} \langle (j_1...j_k), (i_1...i_{k+1}) \rangle.
\]

The last inner product is one if \( p = i_q \) and \( (j_1...j_k) = (i_1...i_{k+1}) \); otherwise it’s zero. Therefore,

\[
 a_{i_1...i_{k+1}} = \begin{cases} 
 \frac{\langle (j_1...j_k), (i_1...i_{k+1}) \rangle}{\langle (j_1...j_k), (j_1...j_k) \rangle}; & \text{if } p = i_q \text{ and } (j_1...j_k) = (i_1...i_{k+1}) \\
 0, & \text{otherwise.}
\end{cases}
\]

That is, \( δ_k^*(i_1...i_{k+1}) \) involves only the boundary elements of \( (i_1...i_{k+1}) \):

\[
 δ_k^*(i_1...i_{k+1}) = \sum_{r=0}^{k} (-1)^r (i_1...i_{k+1}).
\]

For example, consider \( η = a(346) + b(356) \) ∈ Σ_k in Figure 1:

\[
 δ_k^*(i_1i_2)(η) = aδ_k^*(i_1i_2)(346) + bδ_k^*(i_1i_2)(356)
\]

\[
 = a(346) - (36) + b(356) - (36) + (35) = a(34) + b(35) - (a + b)(36) + a(46) + b(56) \in Σ_k.
\]

As we did with δ_k, we can deduce a functional form for δ_k*. From Equation 7, we see that a term \( (i_1...i_{k+1}) \) occurs in the δ_k* expansion of each \( (i_1...i_{k+1}) ∈ Σ_{k+1} \). Consequently, for \( η ∈ Σ_{k+1} \),

\[
 [δ_k^*(η)](i_1...i_{k+1}) = \sum_{(p_1...p_k) ∈ Σ_k} η(p_1...p_k).
\]

In the preceding example, \( η \) assumes the functional form

\[
 [δ_k^*(η)](i_1i_2)(346) = \begin{cases} 
 a; & (i_1i_2) = (346) \\
 b; & (i_1i_2) = (356) \\
 0; & \text{otherwise.}
\end{cases}
\]

Next, we check that Equation 9 assigns \(-a - b\) to \( (36) \), in agreement with Equation 8:

\[
 [δ_k^*(η)](36) = η(326) + η(536) + η(436)
\]

\[
 = η(326) - η(356) - η(436) = -b - a.
\]

The Hodge Decomposition

Let \( G = (Σ_0, Σ_1, ..., Σ_k) \) as in the previous section. For \( 0 < k < K \), the Hodge decomposition exhibits \( Σ_k \) as a direct sum of three orthogonal subspaces: \( Σ_k = G_k ⊕ H_k ⊕ S_k \). \( G_k \) will be the image of \( Σ_k \) under \( δ_k \), \( S_k \) will be the image of \( Σ_k \) under \( δ_k^* \), and \( H_k \) will contain those elements of \( Σ_k \) that are orthogonal to both \( G_k \) and \( S_k \). Figure 2 visualizes the mappings among the three components of each \( Σ_k \). The three segmented vertical lines suggest the three orthogonal components of each space. The disks represent the zero-vector. There is, of course, just one zero-vector, so all disks in a given vertical line represent the unique zero vector; the multiple copies facilitate a cleaner graphic. Let’s delve into the details of these decompositions.
Lemma 1. For any \( \omega \in <\Sigma_k> \), \( \delta_{k+1}^{*}(\delta_k(\omega)) = 0 \). For any \( \eta \in <\Sigma_{k+1}> \), \( \delta_{k+1}(\delta_k(\eta)) = 0 \).

Proof. By linearity, we need only establish the claim on basis vectors. For \( \omega = (i_1 \ldots i_{k+1}) \in \Sigma_k \),
\[
\delta_{k+1}(\delta_k(\omega)) = \delta_{k+1}(\delta_k(\sum_{p \leq p} (q p i_1 \ldots i_{k+1})))
\]
\[
= \sum_{(p_1 \ldots p_{k+1}) \in \Sigma_{k+1}} \delta_{k+1}(q p i_1 \ldots i_{k+1})
\]
\[
= \sum_{(p_1 \ldots p_{k+1}) \in \Sigma_{k+1}} (q p i_1 \ldots i_{k+1})
\]
\[
= \sum_{(p_1 \ldots p_{k+1}) \in \Sigma_{k+1}} (q p i_1 \ldots i_{k+1})
\]
\[
\Delta_{k+1} (q p i_1 \ldots i_{k+1}) = 0.
\]

For the second claim, we compute, for an arbitrary \( \omega \in <\Sigma_{k+1}> \), \( \eta \in <\Sigma_{k+1}> \),
\[
\langle \omega, \delta_{k+1}(\delta_k(\eta)) \rangle = \langle \delta_{k+1}(\omega), \delta_k^{*}(\eta) \rangle = \langle \delta_k(\delta_{k+1}(\eta)), \omega \rangle = 0.
\]

Hence, \( \delta_{k+1}^{*}(\delta_k^*(\eta)) = 0 \).

Now, in each \( <\Sigma_k> \), define \( S_k \) for \( 0 \leq k < K \) and \( G_k \) for \( 0 \leq k \leq K \) as follows:

\[
S_k = \text{image}(\delta_k^*)
\]
\[
G_k = \text{image}(\delta_{k+1})
\]
\[
= \{ \omega \in <\Sigma_k> : \exists \eta \in <\Sigma_{k+1}> \text{ with } \delta_k(\eta) = \omega \}
\]
\[
= \{ \omega \in <\Sigma_k> : \exists \eta \in <\Sigma_{k+1}> \text{ with } \delta_{k+1}(\eta) = \omega \}
\]
\[
(10)
\]

Lemma 2. \( G_k \) and \( S_k \) are orthogonal subspaces of \( <\Sigma_k> \).

Proof. Let \( \omega \in G_k \), \( \eta \in S_k \). We have
\[
\omega = \delta_{k+1}(\omega'), \text{ for some } \omega' \in <\Sigma_{k+1}>
\]
\[
\eta = \delta_k(\eta'), \text{ for some } \eta' \in <\Sigma_{k+1}>
\]
\[
\langle \omega, \eta \rangle = \langle \delta_{k+1}(\omega'), \delta_k(\eta') \rangle = \langle \delta_k(\delta_{k+1}(\eta')), \omega \rangle = 0.
\]

From Lemma 2, \( G_k \cap S_k = \{0\} \). Defining \( H_k = (G_k \otimes S_k)^{\perp} \), we obtain \( <\Sigma_k> = G_k \oplus H_k \oplus S_k \).

Lemma 3. \( G_k^\perp = \text{kernel}(\delta_{k+1}) \) and \( S_k^\perp = \text{kernel}(\delta_k) \).

Figure 2. Hodge decompositions. Here we visualize the mappings among the three components of each \( <\Sigma_k> \).

Proof. Let \( \omega \in G_k^\perp \) and suppose \( \eta \) is an arbitrary vector in \( <\Sigma_{k+1}> \).
Then
\[
\langle \eta, \delta_{k+1}(\omega) \rangle = \langle \delta_{k+1}(\eta), \omega \rangle = 0
\]
by Lemma 2 because \( \delta_{k+1}(\eta) \in G_k \).

We conclude \( \delta_k^*(\omega) = 0 \), \( \omega \in \text{kernel}(\delta_k^*) \), and \( G_k^\perp \subseteq \text{kernel}(\delta_k^*) \). Conversely, if \( \omega \in \text{kernel}(\delta_k^*) \) and \( \eta \in G_k = \text{image}(\delta_{k+1}) \), we have \( \eta = \delta_{k+1}(\eta') \) for some \( \eta' \in <\Sigma_{k+1}> \) and
\[
\langle \eta, \omega \rangle = \langle \delta_{k+1}(\eta'), \omega \rangle = \langle \eta', \delta_{k+1}(\omega) \rangle = 0.
\]
This forces \( \omega \in G_k^\perp \) and consequently \( \text{kernel}(\delta_{k+1}) \subseteq G_k^\perp \).

The second claim follows in a similar fashion:
\[
\omega \in \text{kernel}(\delta_k), \eta \in S_k
\]
\[
\Rightarrow \langle \omega, \eta \rangle = \langle \omega, \delta_k(\eta') \rangle = \langle \delta_k(\omega), \eta' \rangle = 0 \Rightarrow \omega \in S_k^\perp
\]
\[
\omega \in S_k^\perp, \eta \in <\Sigma_{k+1}>
\]
\[
\Rightarrow \delta_k(\eta) \in \text{image}(\delta_k^*) = S_k
\]
\[
\Rightarrow \langle \delta_k(\omega), \eta \rangle = \langle \omega, \delta_k^*(\eta) \rangle = 0
\]
\[
\Rightarrow \langle \delta_k(\omega), \eta \rangle = 0 \Rightarrow \omega \in \text{kernel}(\delta_k).
\]

Lemma 4. For \( 0 \leq k < K \), \( S_k = \delta_k^*(G_{k+1}) \). For \( 0 \leq k \leq K \), \( G_k = \delta_{k+1}^{-1}(S_{k+1}) \). Also, \( \delta_{k+1}^{-1} \mid_{G_k} \) is a bijection from \( G_k \) to \( S_{k+1} \), and \( \delta_k \mid_{S_k} \) is a bijection from \( S_k \) to \( G_{k+1} \).

Proof. Using Lemma 3 in conjunction with Equation 10, we have
\[
S_k = \delta_k^*(<\Sigma_{k+1}>) = \delta_k^*(G_{k+1} \oplus S_{k+1})
\]
\[
= \delta_k^*(G_{k+1} \oplus \text{image}(\delta_k^*)) = \delta_k^*(G_{k+1})
\]
\[
G_k = \delta_{k+1}^{-1}(<\Sigma_{k+1}>) = \delta_{k+1}^{-1}(S_{k+1} \oplus S_{k+1})
\]
\[
= \delta_{k+1}^{-1}(S_{k+1} \oplus \text{kernel}(\delta_{k+1})) = \delta_{k+1}^{-1}(S_{k+1}).
\]
Shifting indices, we obtain \( \delta^*_k(G_k) = S_{k-1} \). That is, \( \delta^*_k \) maps \( G_k \) onto \( S_{k-1} \). If \( \omega, \omega' \in G_k \) and \( \delta^*_k(\omega) = \delta^*_k(\omega') \), then
\[
(\omega - \omega') \in \text{kernel}(\delta^*_{k-1}) = G^+_k
\]
\[
(\omega - \omega') \in G_k \cap G^+_k
\]
\[
(\omega - \omega') = 0,
\]
which establishes \( \delta^*_{k-1} \) as a one-to-one mapping. A similar proof shows that \( \delta^*_1 |_{S_1} \) is also bijective.

**Theorem 5 (Hodge decomposition).** For \( 0 < k < K_1 \),
\[
<\Sigma>G_k = \delta_{k-1}(\Sigma_{k-1}) \oplus \text{kernel}(\delta^*_k) \oplus \delta^*_k(<\Sigma_{k-1}>).
\]
Every \( \omega \in <\Sigma>G_k \) admits a unique decomposition \( \omega = \omega_k + \omega_h + \omega_k \).

**Proof.** Our definitions established
\[
<\Sigma>G = G_k \oplus H_k \oplus S_k,
\]
where \( G_k = \delta_{k-1}(\Sigma_{k-1}) \), \( H_k = \text{kernel}(\delta^*_{k-1}) \), and \( S_k = \delta^*_k(<\Sigma>-1) \).

It remains to show that \( \text{kernel}(\delta^*_{k-1}) \cap \text{kernel}(\delta^*_k) = \text{kernel}(\delta^*_{k-1}) \cap \text{kernel}(\delta^*_k) \).

The unique decomposition of \( \omega \in <\Sigma>G_k \) follows from the orthogonal subspace decomposition of \( <\Sigma>G_k \).

We now have the relationships implied by Figure 2. In particular, each \( G_k \) is isomorphic to the preceding \( S_k \). It follows that \( \text{dim}(G_k) = \text{dim}(S_{k-1}) \).

**Applications in Finding Best Estimates**
In this section, we emphasize graphs as elements of \( <\Sigma>G \)—that is, a given set of edge weights directly establishes an element of \( <\Sigma>G \). Figure 1a shows a directed graph that depicts widget flow among cities. Traffic flows in both directions, and the assigned edge weight reflects the net flow in the direction of the arrow. Negative weights then specify a net flow against the arrow. This graph is the \( <\Sigma>G \)-element \( \omega \), for which we compute \( \delta^*_0(\omega) \) via the methods of the previous section:
\[
\omega = -(12) + 4(18) - 2(23) + 8(26) + 3(34)
\]
\[
+ 6(35) + 10(36) + 7(38) + 3(45) + 7(46)
\]
\[
+ 4(56) - (67) - 2(78)
\]
(11)

\[
\delta^*_0(\omega) = -[(2) - (1)] + 4[(8) - (1)] - 2[(3) - (2)]
\]
\[
+ 8[(6) - (2)] + 3[(4) - (3)] + 6[(5) - (3)]
\]
\[
+ 10[(6) - (3)] + 7[(8) - (3)] + 3[(5) - (4)]
\]
\[
+ 7[(6) - (4)] + 4[(6) - (5)]
\]
\[
- [(7) - (6)] - 2[(8) - (7)]
\]
\[
- 3(1) - 7(2) - 28(3) - 7(4) + 5(5)
\]
\[
+ 30(6) + 7 + 9(8).
\]
(12)

In vector calculus, a vector field’s divergence is the net outgoing flux at a point. We compute this value by surrounding the point with a small ball and integrating the outgoing normal field component over the ball’s surface. The divergence is the limit of this integral divided by the ball volume, as the ball radius shrinks to zero; it gives a measure of the diverging or converging field strength at a point. By analogy, we consider our directed graph as a discrete vector field, defined at each vertex and having components directed to or from its neighbors with magnitudes given by the edge weights. At each vertex, we define the divergence as the excess of outgoing over incoming weights. That is, if \( \omega \) is the weight associated with edge \((i,j)\),
\[
\text{divergence}(i) = \sum_{j} w_{ij} - \sum_{j} w_{ji}.
\]
(13)

Returning to our example, we note that \( \delta^*_0(\omega) \) assigns to each vertex the negative of its divergence, and the short proof below extends this property in the general case.

**Lemma 6.** Let \( \omega \in <\Sigma>G \) represent a weighted directed graph. Then, for each vertex \( (i) \),
\[
[\delta^*_0(\omega)](i) = -\text{divergence}(i).
\]
**Proof.** Suppose the graph has \( n \) vertices and \( \omega = \sum_{ij} w_{ij} (i,j) \). From Equation 9,
\[
[\delta^*_0(\omega)](i) = \sum_{j} \omega(j) - \sum_{j} \omega(j)
\]
\[
- \sum_{j} \omega(i,j) = -\text{divergence}(i).
\]
For \( \omega \) given by Equation 11 and representing the graph of Figure 1a, we note from Equation 12 that vertex 3 has the largest divergence, a net outflow of 28, and is therefore a net widget producer. Similarly, vertex 6 has the smallest divergence, exhibits a net inflow of 30, and is a net widget consumer. Divergence then provides one measure of vertex importance. In this example, we can rank the vertices from vertex 6 with a divergence of -30 up through
vertex 3 with a divergence of 28, affording higher ranks to vertices with greater net production.

Continuing with the same \( \omega \), consider the \( \eta \in \Sigma_0 \) below, together with the edge assignments of \( \delta_0(\eta) \):

\[
\eta = -2(1) - 3(2) - 5(3) - 2(4) + 5 + 6(6) + 4(7) + 2(8) \]

\[
\delta_0(\eta)(12) = \eta(2) - \eta(1) = -1 = \omega(12) \\
\vdots \\
\delta_0(\eta)(78) = \eta(8) - \eta(7) = -2 = \omega(78).
\]

We have \( \delta_0(\eta) = \omega \), that is, \( \omega \in \text{image}(\delta_0) \), and therefore its Hodge decomposition is \( \omega = \omega_h \omega_t = \omega_t = 0 \). Also, \( \eta \in \Sigma_0 \) is a function that assigns values to vertices, and \( \delta_0(\eta) \) assigns to edge \((i,j)\) the incremental change in that function in passing from vertex \((i)\) to \((j)\). By analogy with a similar interpretation in the vector calculus, we refer to \( \delta_0 \) as the gradient operator. By extension, the graph \( \delta_0(\eta) \) is a gradient graph.

In a gradient graph, the edge weight sums along any paths connecting two given vertices are identical. That is, if \( p = ij_1 \ldots ijq \) and \( q = iq_1 \ldots ijq \) are two such paths in \( \omega = \sum_{(u,v)} a_{uv} \), \((i,j) = \delta_0(\eta)\), we have

\[
a_{pi} + \left( \sum_{k=1} a_{i_k j_k} \right) + a_{jq} = \eta(j) - \eta(p) + \sum_{k=2} \eta(i_k) - \eta(i_{k-1}) + \eta(q) - \eta(i) \\
= \eta(q) - \eta(p) \\
a_{pj} + \left( \sum_{k=1} a_{i_k j_k} \right) + a_{jq} = \eta(j) - \eta(p) + \sum_{k=2} \eta(j_k) - \eta(j_{k-1}) + \eta(q) - \eta(j) \\
= \eta(j) - \eta(p).
\]

An undirected cycle in a directed graph is a closed path that might move with or against the edge directions. When moving against the edge direction, we negate the edge weight. In this sense, the edge weights around any undirected cycle sum to zero for a gradient graph, as Figure 1a verifies.

**Lemma 7.** \( G \) is a gradient graph if and only if the edge weight sum along any undirected cycle is 0.

**Proof.** The discussion above establishes that a gradient graph produces zero sums around any undirected cycle. For the converse, we construct an \( \eta \in \Sigma_0 \) such that \( \omega = \delta_0(\eta) \). For each connected component of the undirected graph, choose an arbitrary starting vertex, say, \((i)\), and let \( \eta(i) = 0 \). Proceeding with or against edges, follow paths to other component vertices, summing edge weights, negated if necessary. On reaching a new vertex, say, \((j)\), let \( \eta(j) \) be the value of the sum at that point. Because the sum around any cycle is zero, if we encounter a vertex more than once, we assign it the same value each time. An edge weight in \( \omega \) is then the increment in \( \eta \) from its lower to higher vertex. That is, \( \omega = \delta_0(\eta) \) is a gradient graph.

Actually, the gradient graph \( \omega \) in Figure 1a was constructed as \( \delta_0(\eta) \), for the \( \eta \) in Equation 14. However, because the construction of Lemma 7 involves arbitrary choices, a given gradient graph possesses many generators in \( \Sigma_0 \). In general, for gradient graph \( \omega \in \Sigma_0 \), let \( \delta_0^{-1}(\omega) = \{ \eta \in \Sigma_0 : \delta_0(\eta) = \omega \} \). On a given component of the underlying undirected graph, the difference between any two elements of \( \delta_0^{-1}(\omega) \) will be a constant assignment to the vertex components. Consequently, on such a graph, all elements of \( \delta_0^{-1}(\omega) \) provide the same vertex ranking within each component. In particular, any connected gradient graph \( \omega \) admits a unique vertex ranking, obtained by arranging the vertex coefficients of any \( \eta \in \delta_0^{-1}(\omega) \) in descending order.

For any \( \eta \in \delta_0^{-1}(\omega) \), we can view the vertex assignments of \(-\eta\) as providing a pressure, or increments, to produce the edge flows of \( \omega \). If the graph represents an electrical network with unit resistance on each edge, then voltages maintained at the values given by \(-\eta\) create the currents given by \( \omega \). Similar observations apply when the \(-\eta\) assignments are interpreted as hydrostatic pressures or constant temperature heat sources/sinks. Among physicists, voltages, pressures, temperatures, and the like are known as potential functions, implying that such functions induce certain voltages, pressure, or temperature gradients. By analogy, we refer to an element \( \eta \in \delta_0^{-1}(\omega) \) as a potential function over the vertices that induces gradient \( \omega \).

The gradient graph in Figure 1a admits the comparisons in Table 3, where the potential \( \eta \in \delta_0^{-1}(\omega) \)
is that of Equation 14. Divergence ranks vertex 7 higher than vertex 8, while potential rank reverses this assessment. Divergence deals only with vertex flow, while potential rank takes the strength of neighboring nodes into consideration.

In Figure 3, we create a graph in which the divergence rank flips the potential rank of nodes 1 and 2 by specifying the lower divergence node as exporting widgets into a prodigious widget production. Two neighboring nodes into consideration.

A general graph isn’t necessarily a gradient graph, but it will have a gradient component via its Hodge decomposition. The gradient component represents the best least squares match of a gradient graph to the given graph. That is, if \( \omega = \omega_h + \omega_g + \omega_r \) is the Hodge decomposition of \( \omega \in \Sigma_G \), then
\[
\| \omega - \omega^r \| = \min_{\omega \in \Sigma_G} \| \omega - \omega' \|, \tag{15}
\]
where \( \|x\| = \langle x, x \rangle^{\frac{1}{2}} \) is the square root of the inner product defined earlier. This observation follows from an expansion of \( \| \omega - \omega' \| \). That is, for \( \omega' \in G_1 \),
\[
\| \omega - \omega' \|^2 = \langle \omega' \rangle - \omega - \omega', \omega - \omega' \rangle = \langle \omega - \omega', \omega - \omega' \rangle = \langle \omega - \omega', \omega - \omega' \rangle = \langle \omega - \omega', \omega - \omega' \rangle + \langle \omega - \omega', \omega - \omega' \rangle = \| \omega - \omega' \|^2 + \| \omega - \omega' \|^2,
\]
because \( \omega_h - \omega' \in G_1 \) and \( \omega_g + \omega_r \in G_1 \). The last expression clearly assumes its minimum value \( \| \omega_h + \omega_r \| \) when \( \omega' = \omega_h \). So, \( \omega_h \) is the best least squares approximation to \( \omega \) among the gradient graphs, and the magnitude of the residual \( \omega_h + \omega_r \) measures how well \( \omega_h \) approximates \( \omega \).

We now analyze nongradient graphs with nontrivial Hodge decompositions. Figure 1b is a modified version of Figure 1a, in which edge weights around undirected cycles no longer sum to zero, and therefore we don’t have a pure gradient graph. Referring to this graph as \( \omega \in \Sigma_G \), let \( \omega = \omega_k + \omega_g + \omega_r \) be its Hodge decomposition. To compute \( \omega_h \), it suffices to know an \( \alpha \in \delta^{-1}_G(\omega_h) \). Noting that \( \omega_k + \omega_r \in G_1 = \ker(\delta_0^\ast) \), we have
\[
\delta_0^\ast(\omega) = \delta_0^\ast(\omega_k) + \delta_0^\ast(\omega_r + \omega_k) = \delta_0^\ast(\omega_k) = \delta_0^\ast(\delta_0(\omega)). \tag{16}
\]

\( \langle \Sigma_G \rangle \) is an \( n \)-dimensional real vector space, whose basis vectors are the 0-simplices \((1,2,\ldots,n)\). Over this basis, the matrix form of \( \delta_0^\ast(\omega) : \langle \Sigma_G \rangle \to \langle \Sigma_0 \rangle \) is particularly simple. The \( i \)-th column describes
\[
\delta_0^\ast(\delta_0(\omega))(i) = \delta_0^\ast(\sum (i) - (j)) = \sum_{j\neq i;\delta_0(\omega)} n_i(i) - \sum_{j\neq i;\delta_0(\omega)} (j),
\]
where \( n_i \) is number of neighbors of \( i \). Column \( i \) of the matrix then has \( n_i \) in the diagonal position and a minus one in each row \( j \) such that an edge connects \( i \) and \( j \). Being the (negated) divergence of a gradient, the map \( \delta_0^\ast \delta_0 \) is called the graph Laplacian and is analogous to the Laplacian of vector calculus. Here, we denote the Laplacian by \( \Delta_0 \). Substituting in Equation 16, we obtain a suitable \( \alpha \in \delta^{-1}_G(\omega_h) \) as a solution of
\[
\Delta_0(\omega) = \delta_0^\ast(\omega) = -\text{divergence}(\omega). \tag{17}
\]

Assuming \( \alpha = \sum_{i=1}^n a_i(i) \), we have the following linear system for \( \omega \) of Figure 1b:
\[
\begin{bmatrix}
2 & -1 & -1 & a_1 \\
-1 & 3 & -1 & -1 & a_2 \\
-1 & 5 & -1 & -1 & -1 & a_3 & -27.9 \\
-1 & -1 & 3 & -1 & a_4 & 6.9 \\
-1 & -1 & -1 & 5 & -1 & a_5 & 4.9 \\
-1 & -1 & -1 & 3 & a_6 & 29.9 \\
1 & -2 & -1 & a_7 & 1.0 \\
-1 & -1 & -1 & 3 & a_8 & 9.3
\end{bmatrix}
\]
Table 4. Hodge decomposition of Figure 1b.

<table>
<thead>
<tr>
<th>Edge</th>
<th>(\omega)</th>
<th>(\omega g)</th>
<th>(\omega h)</th>
<th>(\omega \eta)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(12)</td>
<td>-1.0000</td>
<td>-0.9759</td>
<td>0.0000</td>
<td>-0.0241</td>
</tr>
<tr>
<td>(18)</td>
<td>4.2000</td>
<td>4.1759</td>
<td>0.0000</td>
<td>0.0241</td>
</tr>
<tr>
<td>(23)</td>
<td>-2.0000</td>
<td>-1.9167</td>
<td>-0.0800</td>
<td>-0.0033</td>
</tr>
<tr>
<td>(26)</td>
<td>8.1000</td>
<td>8.0407</td>
<td>0.0800</td>
<td>-0.0207</td>
</tr>
<tr>
<td>(34)</td>
<td>3.1000</td>
<td>3.0037</td>
<td>0.1050</td>
<td>-0.0087</td>
</tr>
<tr>
<td>(35)</td>
<td>5.9000</td>
<td>5.9537</td>
<td>-0.0450</td>
<td>-0.0087</td>
</tr>
<tr>
<td>(36)</td>
<td>9.8000</td>
<td>9.9574</td>
<td>-0.1400</td>
<td>-0.0174</td>
</tr>
<tr>
<td>(38)</td>
<td>7.1000</td>
<td>7.0685</td>
<td>0.0000</td>
<td>0.0315</td>
</tr>
<tr>
<td>(45)</td>
<td>3.1000</td>
<td>2.9500</td>
<td>0.1500</td>
<td>0.0000</td>
</tr>
<tr>
<td>(46)</td>
<td>6.9000</td>
<td>6.9537</td>
<td>-0.0450</td>
<td>-0.0087</td>
</tr>
<tr>
<td>(56)</td>
<td>4.1000</td>
<td>4.0037</td>
<td>0.1050</td>
<td>-0.0087</td>
</tr>
<tr>
<td>(67)</td>
<td>-1.0000</td>
<td>-0.9444</td>
<td>0.0000</td>
<td>-0.0556</td>
</tr>
<tr>
<td>(78)</td>
<td>-2.0000</td>
<td>-1.9444</td>
<td>0.0000</td>
<td>-0.0556</td>
</tr>
</tbody>
</table>

From the solution \(\alpha\), we obtain \(\omega = \delta_0(\alpha)\), calculated as \(\omega = (\omega_h + \omega_g)\). Table 4 displays the Hodge decomposition, \(\omega = \omega_h + (\omega_g + \omega)\), while Table 5 shows the solution \(\alpha\) and the divergence of \(\omega_h\).

At this point, we've shown only how to extract the gradient component \(\omega_g\). Subsequent discussion will take up a similar extraction of component \(\omega_h\), which will then enable the full decomposition shown in the table. Because \(\omega_g, \omega_h \in G_i = \text{kernel}(\delta_0)\), we expect \(\delta_0^*(\omega) = \delta_0^*(\omega_g) + \delta_0^*(\omega_h + \omega) = \delta_0^*(\omega_h)\), as confirmed by Table 5.

We observe small vertex residuals: \(\omega_h + \omega_g = \omega - \omega_h\). For widget flows, we might dismiss these residuals as traffic flow measurement errors. This same graph could arise from an electrical network, with the edge flows representing currents through unit resistances. In this case, \(-\alpha\) gives a best least squares estimate, up to an additive constant, of the node voltages. The actual currents forced by these voltages are given by the \(\omega_h\) edge weights, and the small residuals \(\omega_h + \omega_g\) give a measure of accuracy in the approximation.

A similar analysis extracts the component \(\omega_h \in S_1\). We know that \(\omega = \delta_1(\gamma)\) for some \(\gamma \in \Sigma_2\). Consequently, since both \(\omega_h\) and \(\omega_h^*\) reside in \(S_1 = \text{kernel}(\delta_1)\), we have

\[
\delta_1(\omega) = \delta_1(\omega_h + \omega_g) + \delta_1(\omega_h^*) = \delta_1(\omega_h^*) \quad (19)
\]

The term \(\delta_1(\omega)\) is known as the graph \textit{curl}. This term also derives from vector calculus, where the \textit{curl} measures a field’s rotational tendency at a given point. Unlike the scalar divergence, the \textit{curl} is a vector quantity. Computationally, we orient a small disk perpendicular to the direction in which we want to measure the \textit{curl}. We then integrate the tangential field component around the disk periphery, divide by the disk area, and take the limiting value as the disk radius shrinks to zero.

We can verify that \(\delta_1\) performs an analogous measurement, although the graph’s discrete nature leaves only triangles to play the role of “small disks.” For a triangle \((i_1i_2i_3)\), we have

\[
[\delta_1(\omega)(i_1i_2i_3)] = \omega(i_1i_2i_3) + \omega(i_1i_3i_2) - \omega(i_1i_3i_1),
\]

which is the edge sum around the triangle, starting at \(i_1\) and proceeding toward \(i_2\). That is, \textit{curl}(\omega) assigns a rotational value to each triangle, equal to the edge-sum sum on its boundary. Of course, if \(\omega\) is a pure gradient graph, this cyclical sum will be zero, conforming to the vector calculus identity: \textit{curl} \textit{grad} = 0.

Equation 19 now reads \(\delta_1(\delta_1^*(\gamma)) = \text{curl}(\omega_h)\). Using this equation to solve for \(\gamma\), we obtain \(\omega = \delta_1^*(\gamma)\). We envision a matrix form for \(\delta_1^* = \Sigma_2 \rightarrow \Sigma_2\), in which the elements \((i_1i_2i_3)\) of \(\Sigma_2\) label the rows and columns in increasing lexicographic order. In the graph of Figure 1b, for example, the row/column labels are \((236), (345), (346), (356),\) and \((456)\).

Column \((i_1i_2i_3)\) then represents

\[
\delta_1(\delta_1^*(\gamma)) = \delta_1((\overline{i_1i_2i_3}) - (\overline{i_1i_2} + (\overline{i_1i_3} = \delta_1((\overline{i_1i_2i_3}) = \sum_{(j_1j_2j_3) \in \Sigma_2} (j_1i_2i_3) - \sum_{(j_1j_2j_3) \in \Sigma_2} (j_1i_1i_3) + \sum_{(j_1j_2j_3) \in \Sigma_2} (j_1i_1i_3).
\]


we have \( \operatorname{curl}(s) = \omega \). Using (123) = (23/10/13)

\[
\begin{array}{c|c|c|c}
\text{Vertex} & \alpha & \operatorname{div}(\omega) & \operatorname{div}(\omega_g) \\
\hline
(1) & 0.0000 & 3.2 & 3.2 \\
(2) & -0.9759 & 7.1 & 7.1 \\
(3) & -2.8926 & 27.9 & 27.9 \\
(4) & 0.1111 & 6.9 & 6.9 \\
(5) & 3.0611 & -4.9 & -4.9 \\
(6) & 7.0648 & -29.9 & -29.9 \\
(7) & 6.1204 & -1.0 & -1.0 \\
(8) & 4.1759 & -9.3 & -9.3 \\
\hline
\text{Triangle} & \gamma & \operatorname{curl}(\omega) & \operatorname{curl}(\omega_g) \\
\hline
(236) & -0.0800 & -0.3000 & -0.3000 \\
(345) & -1.1309 & 0.3000 & 0.3000 \\
(346) & 1.2359 & 0.2000 & 0.2000 \\
(356) & -1.1795 & 0.2000 & 0.2000 \\
(456) & 1.2809 & 0.3000 & 0.3000 \\
\end{array}
\]

where

\[
\begin{align*}
J_1 &= \{j : j \notin \{i_1, i_2, i_3\} \text{ and } (j \bar{j} \bar{i}_1) \in \Sigma_2\} \\
J_2 &= \{j : j \notin \{i_1, i_2, i_3\} \text{ and } (j \bar{i}_1 \bar{j}) \in \Sigma_2\} \\
J_3 &= \{j : j \notin \{i_1, i_2, i_3\} \text{ and } (j \bar{j} \bar{i}_3) \in \Sigma_2\}.
\end{align*}
\]

We conclude that column \( (i_1 i_2 i_3) \) contains a three on the diagonal. Moreover, for \( (j_1 j_2) \neq (i_1 i_2 i_3) \),

\[
\begin{aligned}
\text{row} (j_1 j_2 j_3) \text{ contains} & \\
\epsilon_{j_1 j_2 j_3} & \begin{cases} 
\exists j \text{ with } j_1 j_2 j_3 \sim j_1 j_2, & \epsilon_{j_1 j_2 j_3} \\
\exists j \text{ with } j_1 j_2 j_3 \sim j_1 j_3, & -\epsilon_{j_1 j_2 j_3} \\
\exists j \text{ with } j_1 j_2 j_3 \sim j_2 j_3, & \epsilon_{j_1 j_2 j_3} \\
0, & \text{otherwise.}
\end{cases}
\end{aligned}
\]

This formula is notationally cumbersome, but simple in practice. An off-diagonal matrix entry is zero unless the row and column labels differ in exactly one index. If so, replace the nonmatching index in the row label by the nonmatching index in the column label, or vice versa, and permute the resulting label to canonical form. The matrix entry is ±1 as the permutation is even or odd. For example, suppose we’re computing the matrix entry in row (236), column (346). We compute either of the sequences:

(236) replace 2 with 4 ⇒ (436) permute ⇒ (346)
(346) replace 4 with 2 ⇒ (326) permute ⇒ (236),

each requiring an odd number of transposition in the last step. Accordingly, the matrix entry is (–1).

Assuming \( \gamma \) takes the form \( \sum_{\omega(i,j)\in \Sigma_2} c_{ij} \omega(i,j) \).

Equation (19) assumes the following form for the graph of Figure 1b:

\[
\begin{bmatrix}
3 & -1 & -1 & c_{236} & -0.3 \\
3 & 1 & -1 & 1 & c_{436} & 0.3 \\
-1 & 1 & 3 & 1 & c_{146} & 0.2 \\
-1 & -1 & 1 & 3 & c_{56} & 0.2 \\
1 & -1 & 1 & 3 & c_{65} & 0.3
\end{bmatrix}
\]

From the solution \( \gamma \) we obtain \( \omega_i = \delta_i(\gamma) \). Using \( \omega_k \) from our earlier partial decomposition, we complete the full Hodge decomposition with final component \( \omega_k = \omega - \omega_x - \omega_h \). Table 4 shows the decomposition, while Table 5 gives the solutions for \( \alpha \) and \( \gamma \) of Equations 18 and 20, respectively.

Table 5. Solutions for \( \alpha \) and \( \gamma \) of Equations 18 and 20, respectively.

<table>
<thead>
<tr>
<th>Vertex</th>
<th>( \alpha )</th>
<th>( \operatorname{div}(\omega) )</th>
<th>( \operatorname{div}(\omega_g) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>0.0000</td>
<td>3.2</td>
<td>3.2</td>
</tr>
<tr>
<td>(2)</td>
<td>-0.9759</td>
<td>7.1</td>
<td>7.1</td>
</tr>
<tr>
<td>(3)</td>
<td>-2.8926</td>
<td>27.9</td>
<td>27.9</td>
</tr>
<tr>
<td>(4)</td>
<td>0.1111</td>
<td>6.9</td>
<td>6.9</td>
</tr>
<tr>
<td>(5)</td>
<td>3.0611</td>
<td>-4.9</td>
<td>-4.9</td>
</tr>
<tr>
<td>(6)</td>
<td>7.0648</td>
<td>-29.9</td>
<td>-29.9</td>
</tr>
<tr>
<td>(7)</td>
<td>6.1204</td>
<td>-1.0</td>
<td>-1.0</td>
</tr>
<tr>
<td>(8)</td>
<td>4.1759</td>
<td>-9.3</td>
<td>-9.3</td>
</tr>
<tr>
<td>Triangle</td>
<td>( \gamma )</td>
<td>( \operatorname{curl}(\omega) )</td>
<td>( \operatorname{curl}(\omega_g) )</td>
</tr>
<tr>
<td>(236)</td>
<td>-0.0800</td>
<td>-0.3000</td>
<td>-0.3000</td>
</tr>
<tr>
<td>(345)</td>
<td>-1.1309</td>
<td>0.3000</td>
<td>0.3000</td>
</tr>
<tr>
<td>(346)</td>
<td>1.2359</td>
<td>0.2000</td>
<td>0.2000</td>
</tr>
<tr>
<td>(356)</td>
<td>-1.1795</td>
<td>0.2000</td>
<td>0.2000</td>
</tr>
<tr>
<td>(456)</td>
<td>1.2809</td>
<td>0.3000</td>
<td>0.3000</td>
</tr>
</tbody>
</table>

The Hodge gradient component captures the graph divergence at each node. At each vertex in our widget examples, positive divergence indicates excess production over consumption; negative divergence indicates excess consumption. However, if we add traffic that simply moves widgets around closed paths, these additional flows don’t change any node divergence and consequently have no net impact on production or consumption. The residuals \( \omega_x + \omega_h \) represent such cyclical traffic. Small residuals might suggest traffic flow measurement errors, whereas large residuals could signify unnecessary network congestion, in the sense that widget demands don’t justify the measured circulation.

The smallest closed paths are triangles, and we use the term solenoidal flow to designate circulatory flow around them. The Hodge component \( \omega_h \) measures, on each edge, the traffic portion due to solenoidal circulation around triangles. As with the edges, we adopt an orientation for triangles. If the flow follows the vertices along an even permutation of the canonical order \( i_1 i_2 i_3 \), then the flow is positive. If it follows an odd permutation of the vertices, we negate its value. That is, the terms 16.3(245) = −16.3(425) both specify a flow of magnitude 16.3 proceeding around the triangle (245).

The solenoidal component can’t account for all divergence-free circulation because some circulation can occur around nontriangular paths. Path (12678) of Figure 1b provides an example. The final Hodge component \( \omega_h \) provides edge flows associated with such circulations. After computing \( \omega_x \) and \( \omega_h \), we can obtain \( \omega_k \) by subtracting the two known components from the given graph.
edge weights. Because \( \omega \in \text{kernel}(\delta^0) \cap \text{kernel}(\delta^1) \), we have

\[
(\delta^0 \delta^0 + \delta^1 \delta^1) (\omega) = 0.
\]

For \( k > 0 \), the map \( \delta^k : A_{k+1}^+ + \delta^k : <\Sigma_2> \rightarrow <\Sigma_2> \) is called the \textit{generalized Laplacian} and is denoted by \( \Delta_k \). Hence, \( \Delta_k (\omega) = 0 \), and because such solutions are termed \textit{harmonic} in the continuous theory, we call \( \omega \) the harmonic component.

We can imitate the observations in Equation 15 to show that

\[
\| \omega - \omega' \| = \min_{\omega \in \Sigma} \| \omega - \omega' \|
\]

which means that \( \omega \) gives the best least squares approximation to a given graph \( \omega' \) within the class of solenoidal graphs. For example, let \( \omega \) represent the graph of Figure 1c, where a quick check reveals that the divergence is zero at each node. Consequently, this graph’s decomposition should have no gradient component. Moreover, weights associated with edges that don’t participate in triangles are zero. We then expect that the flow derives from triangular circulations alone, forcing the harmonic component to be zero. That is, the graph will be a pure solenoid \( \omega = \omega = \delta^k (\gamma) \) for some \( \gamma \in <\Sigma_2> \). Indeed, using Equation 9, we find

\[
\gamma = 4(326) + 2(345) + (346) + 3(356) + 4(456)
\]

yields \( \delta^k (\gamma) = \omega \).

Recall that a given gradient graph can arise from several potentials in \( \Sigma_0 \), with the various contenders differing by constants on the graph’s connected components. A similar multiplicity occurs with solutions \( \gamma \) of Equation 19: \( \delta^k (\delta^k (\gamma)) = \text{curl} (\omega) \). In the example at hand, we can verify that for any constant \( c \), \( \gamma = c(345) - c(346) + c(356) - c(456) \) yields \( \delta^k (\gamma) = 0 \). Consequently, for any solution \( \gamma \) of \( \delta^k (\delta^k (\gamma)) = \text{curl} (\omega) \), \( \gamma + \gamma \) is also a solution. Multiple solutions present no difficulty, because they all map to the same unique solenoidal component of the decomposition.

**Application to Global Ranking**

We return now to the lollipop problem that opened this tutorial. Let’s develop a random triad selection by iterating through all three-vertex subsets and choosing the corresponding triad with some probability \( p \). We reject any sampling in which the triad edges don’t generate a connected graph on the candidate vertices. We used \( p = 0.1 \) to generate the lollipop data earlier. In that example, we computed triad judgments using overlapping Gaussians, centered at known ranking positions. In general, however, ground truth isn’t known, and we accept elementary triad judgments from a panel of experts. In either case, we construct a graph \( \omega \) from the local judgments, in which a positive edge weight from vertex \( i \) to vertex \( j \) indicates a “preference” flow from weaker candidate \( i \) to stronger candidate \( j \).

In the absence of conflicts, this preference flow forms a gradient graph because the sum around any cycle is necessarily zero. In that case, the preference graph \( \omega \) is equal to its Hodge gradient component \( a \). The gradient component in turn derives from a potential \( \alpha \) over the vertices, and \( [\delta^k (\alpha)] (ij) = \alpha (j) - \alpha (i) \) recovers the overall preference flow from \( i \) to \( j \). The potential \( \alpha \) then serves to globally rank the vertices.

Of course, we can hardly expect our conflicting local triad rankings to form a pure gradient graph. Such a miracle requires not only that each candidate pair be judged in the correct direction, but also that the magnitude of the assigned preferences be consistent across the local judgments. Nevertheless, we conjecture that a common appreciation of merit among the rankers will produce a graph in which the gradient component approximates the group judgment. That is, if \( \sigma \) is small in the lollipop simulation, or if the panel is competent in the real-world counterpart, preference conflicts should be minor.

To this end, we convert our triad judgments into edge weights \( a_{ij} \), such that \( a_{ij} > 0 \) when candidate \( j \) is perceived superior to candidate \( i \). Specifically, because we’re ranking triads, we record the assessment \( \text{merit}(k) > \text{merit}(i) \) as \( a_{ij} = 1 \), \( a_{ik} = 1 \), \( a_{ik} = 2 \). Clearly, this triangle in isolation is a pure gradient graph. At this point, however, a complication occurs. Because we choose triads randomly, the same edge, say, \( (ij) \), can occur in many trials and may therefore receive different weights. Since we’re attempting to discover the group’s judgment of candidate \( i \) versus candidate \( j \), it’s reasonable to accumulate the assigned weights. Opposing judgments then tend to reduce the preference flow, while reinforcing judgments increase it. Other methods might be appropriate, if, for example, we know that some rankers are better qualified to assess the relative merit of particular candidates. Using simple summation produces the result noted in the introduction. In brief, the algorithm proceeds as follows:

1. Establish a graph \( \omega \) by adding the edge weights assigned to the local triangles.
2. Solve the Laplacian equation: \( \Delta_k (\alpha) = \delta^k (\omega) \).
3. Rank the vertices via \( \alpha \). That is, \( \alpha (j) > \alpha (i) \) implies \( j \) is ranked higher than \( i \).
We could use edges or tetrahedra instead of triangles, and in our artificial data experiments, we can vary the $\sigma$ parameter to simulate more or fewer conflicts among the rankers. The curves in Figure 4 illustrate how algorithm performance varies with $\sigma$, measured as the average number of positions a candidate is displaced from ground truth. We expect this metric to rise monotonically with increasing $\sigma$. However, because the simulations choose random triads, it can happen that a triad group chosen with a larger $\sigma$ can nevertheless outperform a group chosen with a smaller $\sigma$. To extract the trend, we smooth the data by averaging many simulations at each $\sigma$ value. The graphs then report an expected average displacement. All exhibit greater displacements with increased $\sigma$, corresponding to greater error associated with less-competent judges. It’s notable that this displacement remains relatively small, even for large $\sigma$.

In both panels of Figure 4, the upper curve corresponds to local judgments pronounced independently on randomly chosen pairs. The middle curve corresponds to triads and the lowest curve to quadruples. Performance improves markedly in passing from pairs to triads and also exhibits more stability with increasing $\sigma$. Further improvement appears when quadruples are chosen for the independent local judgments.

In Figure 4a, pairs were chosen with probability $p = 0.16$, while triads and quadruples were chosen with $p = 0.04$. Recall that the random selection is constrained to cover all candidates in the sense that any two candidates are connected by a chain of local comparisons. For pairs, this constraint is difficult to realize with smaller acceptance probabilities. For triads or quadruples, it’s easily satisfied with $p = 0.04$. In Figure 4b, pairs, triads, and quadruples were all chosen with $p = 0.2$. As expected, all curves improve with larger random selections, but this improvement isn’t without cost. For a given $p$, a random selection of triads is much larger than a corresponding selection of pairs. A quadruple selection is yet larger.

Hodge decomposition and ranking software, in the form of executable Java files, are available from www.cs.wwu.edu/johnson/hodge/hodgeSoftware.htm as executable Java jar-files.

The ranking software provides an extension of the triad-judgment reconciliation described here. Specifically, it accommodates a heterogeneous collection of local judgments (pair, triads, quadruples, and so forth) on the input file. The site also provides access to the Java source code for both decomposition and ranking. To compute solutions to the various matrix algebra equations that appear in our earlier discussion, the code adapts the minresQLP linear solver available from the Stanford Systems Optimization Laboratory (www.stanford.edu/group/SOL/download.html).

References

James L. Johnson is a professor of computer science at Western Washington University. His research interests include probability applications, graph theory, and algorithms. Johnson has a PhD in mathematics from the University of Minnesota. He’s a member of the Society for Industrial and Applied Mathematics (SIAM). Contact him at James.Johnson@wwu.edu.

Tom Goldring is a mathematician at the US National Security Agency. His research interests focus on graph theory. Goldring has a PhD in mathematics from Yeshiva University. Contact him at Tom.Goldring@gmail.com.

Selected articles and columns from IEEE Computer Society publications are also available for free at http://ComputingNow.computer.org.

KEEP YOUR COPY OF IEEE SOFTWARE FOR YOURSELF!

Give subscriptions to your colleagues or as graduation or promotion gifts—way better than a tie!

IEEE Software is the authority on translating software theory into practice.

www.computer.org/software/subscribe