PyDEC: Software and Algorithms for Discretization of Exterior Calculus

NATHAN BELL, Google Inc.
ANIL N. HIRANI, University of Illinois at Urbana-Champaign

3

This article describes the algorithms, features, and implementation of PyDEC, a Python library for computations related to the discretization of exterior calculus. PyDEC facilitates inquiry into both physical problems on manifolds as well as purely topological problems on abstract complexes. We describe efficient algorithms for constructing the operators and objects that arise in discrete exterior calculus, lowest-order finite element exterior calculus, and in related topological problems. Our algorithms are formulated in terms of high-level matrix operations which extend to arbitrary dimension. As a result, our implementations map well to the facilities of numerical libraries such as NumPy and SciPy. The availability of such libraries makes Python suitable for prototyping numerical methods. We demonstrate how PyDEC is used to solve physical and topological problems through several concise examples.

Categories and Subject Descriptors: G.4 [**Mathematical Software**]: ; G.1.8 [**Numerical Analysis**]: Partial Differential Equations—*Finite element methods, finite volume methods*; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Geometric algorithms, languages, and systems*

General Terms: Algorithms, Theory, Experimentation, Design

Additional Key Words and Phrases: Discrete exterior calculus, finite element exterior calculus, Whitney form, simplicial complex, cubical complex, Vietoris-Rips complex, computational topology, boundary operator, coboundary operator, chain, cochain

ACM Reference Format:
Bell, N. and Hirani, A. N. 2012. PyDEC: Software and algorithms for discretization of exterior calculus. ACM Trans. Math. Softw. 39, 1, Article 3 (November 2012), 41 pages.
DOI = 10.1145/2382585.2382588 http://doi.acm.org/10.1145/2382585.2382588

1. INTRODUCTION

Geometry and topology play an increasing role in the modern language used to describe physical problems [Abraham et al. 1988; Frankel 2004]. A large part of this language is *exterior calculus* which generalizes vector calculus to smooth manifolds of arbitrary dimensions. The main objects are differential forms (which are antisymmetric tensor fields), general tensor fields, and vector fields defined on a manifold. In addition to physical applications, differential forms are also used in cohomology theory in topology [Bott and Tu 1982].

Once the domain of interest is discretized, it may not be smooth and so the objects and operators of exterior calculus have to be reinterpreted in this context. For example, a surface in $\mathbb{R}^3$ may be discretized as a two-dimensional simplicial complex

---

This work was funded in part by NSF grant DMS-0645604.
Authors' addresses: N. Bell, Google, Inc.; A. N. Hirani (corresponding author), Department of Computer Science, University of Illinois at Urbana-Champaign, IL; email: hirani@cs.illinois.edu.

DOI 10.1145/2382585.2382588 http://doi.acm.org/10.1145/2382585.2382588

embedded in $\mathbb{R}^3$, that is, as a triangle mesh. Even when the domain is a simple domain in space, such as an open subset of the plane or space, the discretization is usually in the form of some mesh. The various objects of the problem then become defined in a piecewise varying fashion over such a mesh and so a discrete calculus is required there as well. After discretizing the domains, objects, and operators, one can compute numerical solutions of partial differential equations (PDEs), and compute some topological invariants using the same discretizations. Both these classes of applications are considered here.

There have been several recent developments in the discretization of exterior calculus and in the clarification of the role of algebraic topology in computations. These go by various names, such as covolume methods [Nicolaides and Trapp 2006], support operator methods [Shashkov 1996], mimetic discretization [Bochev and Hyman 2006; Hyman and Shashkov 1997], discrete exterior calculus (DEC) [Desbrun et al. 2005; Gillette and Bajaj 2010; Hirani 2003], compatible discretization, finite element exterior calculus [Arnold et al. 2006, 2010; Holst and Stern 2012], edge and face elements or Whitney forms [Bossavit 1988; Hiptmair 2002], and so on. PyDEC provides an implementation of discrete exterior calculus and lowest-order finite element exterior calculus using Whitney forms.

Within pure mathematics itself, ideas for discretizing exterior calculus have a long history. For example, the de Rham map that is commonly used for discretizing differential forms goes back at least to de Rham [1955]. The reverse operation of interpolating discrete differential forms via the Whitney map appears in Whitney [1957]. A combinatorial (discrete) Hodge decomposition theorem was proved in Dodziuk [1976] and the idea of a combinatorial Hodge decomposition dates to Eckmann [1945]. More recent work on discretization of Hodge star and wedge product is in Wilson [2007, 2008]. Discretizations on other types of complexes have been developed as well [Gradinaru and Hiptmair 1999; Sen 2003].

1.1. Main Contributions

In this article we describe the algorithms and design of PyDEC, a Python software library implementing various complexes and operators for discretization of exterior calculus, and the algorithms and data structures for those. In PyDEC all the discrete operators are implemented as sparse matrices and we often reduce algorithms to a sequence of standard high-level operations, such as sparse matrix-matrix multiplication [Bank and Douglas 1993], as opposed to more specialized techniques and ad hoc data structures. Since these high-level operations are ultimately carried out by efficient, natively compiled routines (e.g., C or Fortran implementations) the need for further algorithmic optimization is generally unnecessary.

As is commonly done, in PyDEC we implement discrete differential forms as real-valued cochains which will be defined in Section 2. PyDEC has been used in a thesis [Bell 2008], in classes taught at University of Illinois, in experimental parts of some computational topology papers [Dey et al. 2010; Dunfield and Hirani 2011; Hirani et al. 2011a], in Darcy flow [Hirani et al. 2011c], and in least-squares ranking on graphs [Hirani et al. 2011b]. The PyDEC source code and examples are publicly available [Bell and Hirani 2008]. We summarize here our contributions grouped into four areas.

Basic objects and methods. (1) Data structures for simplicial complexes of dimension n embedded in $\mathbb{R}^N$, $N \geq n$; abstract simplicial complexes; Vietoris-Rips complexes for points in any dimension; and regular cube complexes of dimension n embedded in $\mathbb{R}^n$; (2) cochain objects for the preceding complexes; (3) discrete exterior derivative as a coboundary operator, implemented as a method for cochains on various complexes.

Finite element exterior calculus. (1) Fast algorithm to construct sparse mass matrices for Whitney forms by eliminating repeated computations and (2) assembly of stiffness matrices for Whitney forms from mass matrices by using products of boundary and stiffness matrices. Note that only the lowest-order ($\mathcal{P}_1^-$) elements of finite element exterior calculus are implemented in PyDEC.

Discrete exterior calculus. (1) Diagonal sparse matrix discrete Hodge star for well-centered (circumcenters inside simplices) and Delaunay simplicial complexes (with an additional boundary condition); (2) circumcenter calculation for k-simplex in an n-dimensional simplicial complex embedded in $\mathbb{R}^N$ using a linear system in barycentric coordinates; and (3) volume calculations for primal simplices and circumcentric dual cells.

Examples. (1) Resonant cavity curl-curl problem; (2) flow in porous medium modeled as Darcy flow, that is, Poisson's equation in first-order (mixed) form; (3) cohomology basis calculation for a simplicial mesh, using harmonic cochain computation using Hodge decomposition; (4) finding sensor network coverage holes by modeling an abstract, idealized sensor network as a Rips complex; and (5) least-squares ranking on graphs using Hodge decomposition of partial pairwise comparison data.

2. OVERVIEW OF PYDEC

One common type of discrete domain used in scientific computing is triangle or tetrahedral mesh. These and their higher-dimensional analogs are implemented as n-dimensional simplicial complexes embedded in $\mathbb{R}^N$, $N \geq n$. Simplicial complexes are useful even without an embedding and even when they don't represent a manifold, for example, in topology and ranking problems. Such abstract simplicial complexes without any embedding for vertices are also implemented in PyDEC. The other complexes implemented are regular cube complexes and Rips complexes. Regular cubical meshes are useful since it is easy to construct domains even in high dimensions whereas simplicial meshing is hard enough in 3 dimensions and rarely done in 4 or larger dimensions. Rips complexes are useful in applications such as topological calculations of sensor network coverage analysis [de Silva and Ghrist 2007]. The representations used for these four types of complexes are described in Sections 3–6. A complex that is a manifold (i.e., locally Euclidean) will be referred to as *mesh*.

The definitions here are given for simplicial complexes and generalize to the other types of complexes implemented in PyDEC. In PyDEC we only consider integer-valued chains and real-valued cochains. Also, we are only interested in finite complexes, that is, ones with a finite number of cells. Let K be a finite simplicial complex and denote its underlying space by $|K|$. Give $|K|$ the subspace topology as a subspace of $\mathbb{R}^N$ (a set U in $|K|$ is open iff $U \cap |K|$ is open in $\mathbb{R}^N$). For a finite complex this is the same as the standard way of defining topology for $|K|$ [Munkres 1984, pages 8–9] and $|K|$ is a closed subspace of $\mathbb{R}^N$.

An oriented simplex with vertices $v_0, \ldots, v_p$ will be written as $[v_0, \ldots, v_p]$ and given names like σ_i^p with the superscript denoting the dimension and subscript denoting its place in some ordering of p-simplices. Sometimes the dimensional superscript and/or the indexing subscript will be dropped. The orientation of a simplex is one of two equivalence classes of vertex orderings. Two orderings are equivalent if one is an even permutation of the other. For example, $[v_0, v_1, v_2]$ and $[v_1, v_2, v_0]$ denote the same oriented triangle while $[v_0, v_2, v_1]$ is the oppositely oriented one.

A *p-chain of K* is a function c from oriented p-simplices of K to the set of integers $\mathbb{Z}$, such that $c(-\sigma) = -c(\sigma)$ where $-\sigma$ is the simplex σ oriented in the opposite way. Two chains are added by adding their values. Thus p-chains are formal linear combinations (with integer coefficients) of oriented p-dimensional simplices. The space of p-chains is denoted $C_p(K)$ and it is a free abelian group. See Munkres [1984, page 21]. Free abelian groups have a basis and one does not need to impose a vector space structure. For example, a basis for $C_p(K)$ is the set of integer-valued functions that are 1 on a p-simplex and 0 on the rest, with one such basis element corresponding to each p-simplex. These are called *elementary chains* and the one corresponding to a p-simplex σ^p will also be referred to as σ^p. The existence of this basis and the addition and negation of chains is the only aspect that is important for this article. The intuitive way to think of chains is that they play a role similar to that played by the domains of integration in the smooth theory. The negative sign allows one to talk about orientation reversal and the integer coefficient allows one to say how many times integration is to be done on that domain.

Sometimes we will need to refer to a *dual mesh* which will in general be a cell complex obtained from a subdivision of the given complex K. We'll refer to the dual complex as $\star K$. For a discrete Hodge star diagonal matrix of DEC, the dual mesh is the one obtained from circumcentric subdivision of a well-centered or Delaunay simplicial complex and such a Hodge star is described in Section 10.

Homomorphisms from the p-chain group $C_p(K)$ to $\mathbb{R}$ are called *p-cochains of K* and denoted $C^p(K;\mathbb{R})$. This set is an abelian group and also a vector space over $\mathbb{R}$. Similarly the dual p-cochains are denoted $C^p(\star K;\mathbb{R})$ or $D^p(\star K;\mathbb{R})$. The discretization map from space of smooth p-forms to p-cochains is called the *de Rham* map $\mathrm{R} : \Omega^p(K) \to C^p(K;\mathbb{R})$ or $\mathrm{R} : \Omega^p(K) \to C^p(\star K;\mathbb{R})$. See de Rham [1955] and Dodziuk [1976]. For a smooth p-form α, the de Rham map is defined as $\mathrm{R} : \alpha \mapsto (c \mapsto \int_c \alpha)$ for any chain $c \in C_p(K)$. We will denote the evaluation of the cochain $\mathrm{R}(\alpha)$ on a chain c as $\langle \mathrm{R}(\alpha), c\rangle$. A basis for $C^p(K;\mathbb{R})$ is the set of *elementary cochains*. The elementary cochain $(\sigma^p)^*$ is the one that takes value 1 on elementary chain σ^p and 0 on the other elementary chains. Thus the vector space dimension of $C^p(K;\mathbb{R})$ is the number of p-simplices in K. We'll denote this number by N_p. Thus N_0 will be the number of vertices, N_1 the number of edges, N_2 the number of triangles, and so on.

Like most of the numerical analysis literature mentioned in Section 1 we assume that the smooth forms are either defined in the embedding space of the simplicial complex, or on the complex itself, or can be obtained by pullback from the manifold that the complex approximates. In contrast, most mathematics literature quoted including Whitney [1957] and Dodziuk [1976] use simplicial complex defined on the smooth manifold as a "curvilinear" triangulation. In the applied literature, the complex approximates the manifold. Many finite element papers deal with open subsets of the plane or $\mathbb{R}^3$ so they are working with triangulations of a manifold with piecewise smooth boundaries. Surface finite element methods have been studied outside of exterior calculus [Demlow and Dziuk 2007]. A variational crimes methodology is used for finite element exterior calculus on simplicial approximations of manifolds in Holst and Stern [2012]. In the computer graphics literature, piecewise-linear triangle mesh surfaces embedded in $\mathbb{R}^3$ are common and convergence questions for operators on such surfaces have been studied [Hildebrandt et al. 2006]. In light of all of these, PyDEC's framework of using simplicial or other approximations of manifolds is appropriate.

Operators such as the discrete exterior derivative (d) and Hodge star ($*$) can be implemented as sparse matrices. At each dimension, the exterior derivative can be easily determined by the incidence structure of the given simplicial mesh. For DEC the Hodge star is a diagonal matrix whose entries are determined by the ratios of primal and dual volumes. Care is needed for dual volume calculation when the mesh is not well-centered. For finite element exterior calculus we implement Whitney forms. The corresponding Hodge star is the mass matrix which is sparse but not diagonal. One of the stiffness matrices can be obtained from it by combining it with the exterior derivative.

Once the matrices implementing the basic operators have been determined, they can be composed together to obtain other operators such as the codifferential (δ) and Laplace-deRham (Δ). While this composition could be performed manually, that is, the appropriate set of matrices combined to form the desired operation, it is prone to error. In PyDEC this composition is handled automatically. For example, the function d(.), which implements the exterior derivative, looks at the dimension of its argument to determine the appropriate matrix to apply. The same method can be applied to the codifferential function δ(.), which then makes their composition δ(d(.)) work automatically. This automation eliminates a common source of error and makes explicit which operators are being used throughout the program.

PyDEC is intended to be fast, flexible, and robust. As an interpreted language, Python by itself is not well-suited for high-performance computing. However, combined with numerical libraries such as NumPy and SciPy one can achieve fast execution with only a small amount of overhead. The NumPy array structure, used extensively throughout SciPy and PyDEC, provides an economical way of storing N-dimensional arrays (comparable to C or Fortran) and exposes a C API for interfacing Python with other, potentially lower-level libraries [van der Walt et al. 2011]. In this way, Python can be used to efficiently "glue" different highly optimized libraries together with greater ease than a purely C, C++, or Fortran implementation would permit [Oliphant 2007]. Indeed, PyDEC also makes extensive use of the `sparse` module in SciPy which relies on natively compiled C++ routines for all performance-sensitive operations, such as sparse matrix-vector and matrix-matrix multiplication. PyDEC is therefore scalable to large datasets and capable of solving problems with millions of elements [Bell 2008].

Even large-scale, high-performance libraries such as Trilinos provide Python bindings showing that Python is useful beyond the prototyping stage. We also make extensive use of Python's built-in unit testing framework to ensure PyDEC's robustness. For each nontrivial component of PyDEC, a number of examples with known results are used to check for consistency.

2.1. Previous Work

Discrete differential forms now appear in several finite element packages such as FEMSTER [Castillo et al. 2005], DOLFIN [Logg and Wells 2010], and deal.II [Bangerth et al. 2007]. These libraries support arbitrary order conforming finite element spaces in two and three dimensions. In contrast, for finite elements PyDEC supports simplicial and cubical meshes of arbitrary dimension, albeit with lowest-order elements. In addition, PyDEC also supports the operators of discrete exterior calculus and complexes needed in topology. We note that Exterior [Logg and Mardal 2008], an experimental library within the FEniCS [Logg et al. 2012] project, realizes the framework developed by Arnold et al. [2009] which generalizes to arbitrary order and dimension. Exterior uses symbolic methods and supports integration of forms on the standard simplex. PyDEC supports mass and stiffness matrices on simplicial and cubical complexes. The discovery of lower-dimensional faces in a complex and the computation of all the boundary matrices is also implemented in PyDEC.

The other domain where PyDEC is useful is in computational topology. There are several packages in this domain as well, and again PyDEC has a different set of features and aims from these. In Kaczynski et al. [2004] efficient techniques are

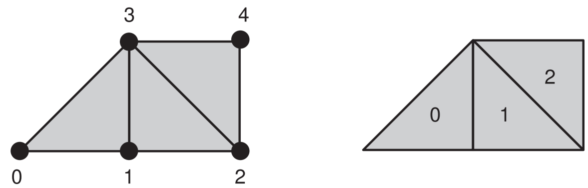


Fig. 1. Simplicial mesh with enumerated vertices and simplices.

developed for finding meaningful topological structures in cubical complexes, such as digital images. In addition to simplicial and cubical manifolds, PyDEC also provides support for abstract simplicial complexes such as the Rips complex of a point set. The Applied and Computational Topology group at Stanford University has been the source for several packages for computational topology. These include various versions of PLEX such as JPlex and javaPlex which are designed for persistent homology calculations. Another package from the group is Dionysus, a C++ library that implements persistent homology and cohomology [Edelsbrunner et al. 2002; Zomorodian and Carlsson 2005] and other interesting topological and geometric algorithms. In contrast, we view the role of PyDEC in computational topology as providing a tool to specify and represent different types of complexes, compute their boundary matrices, and compute cohomology representatives with or without geometric information.

3. SIMPLICIAL COMPLEX REPRESENTATION

Before detailing the algorithms used to implement discretizations of exterior calculus, we discuss the representation of various complexes, starting in this section with simplicial complexes. Consider the triangle mesh shown in Figure 1 with vertices and faces enumerated as shown. This example mesh is represented by arrays

$$\mathbb{V} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 2 & 0 \\ 1 & 1 \\ 2 & 1 \end{bmatrix}, \qquad \mathbb{S}_2 = \begin{bmatrix} 0 & 1 & 3 \\ 1 & 2 & 3 \\ 2 & 4 & 3 \end{bmatrix},$$

where the subscript 2 denotes the dimension of the simplices. The i-th row of $\mathbb{V}$ contains the spatial coordinates of the i-th vertex. Likewise the i-th row of simplex array $\mathbb{S}_2$ contains the indicies of the vertices that form the i-th triangle. The indices of each simplex in $\mathbb{S}_2$ in this example are ordered in a manner that implies a counter-clockwise orientation for each. For an n-dimensional discrete manifold, or mesh, arrays $\mathbb{V}$ and $\mathbb{S}_n$ suffice to describe the computational domain.

In addition to $\mathbb{V}$ and $\mathbb{S}_n$, an n-dimensional simplicial complex is comprised by its p-dimensional faces, $\mathbb{S}_p$, $0 \leq p < n$. In the case of Figure 1, these are

$$\mathbb{S}_0 = \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}, \qquad \mathbb{S}_1 = \begin{bmatrix} 0 & 1 \\ 0 & 3 \\ 1 & 2 \\ 1 & 3 \\ 2 & 3 \\ 2 & 4 \\ 3 & 4 \end{bmatrix},$$

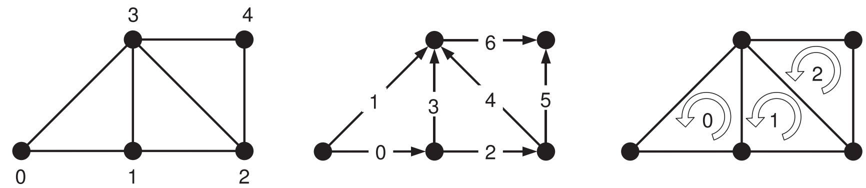


Fig. 2. Simplicial complex with oriented edges and triangles.

which correspond to the vertices (0-simplices) and oriented edges (1-simplices) of the complex. A graphical representation of this simplicial complex is shown in Figure 2. Since the orientation of the lower ($< n$)-dimensional faces is arbitrary, we use the convention that face indices will be in sorted order. Furthermore, we require the rows of $\mathbb{S}$ to be sorted in lexicographical order. As pointed out in Sections 7 and 9, these conventions facilitate efficient construction of differential operators and stiffness matrices.

4. REGULAR CUBE COMPLEX REPRESENTATION

PyDEC provides a regular cube complex of dimension n embedded in $\mathbb{R}^n$ for any n. As mentioned earlier, in dimension higher than 3, constructing simplicial manifold complexes is hard. In fact, even construction of good tetrahedral meshes is still an active area in computational geometry. This is one reason for using regular cube complexes in high dimensions. Moreover, for some applications, like topological image analysis or analysis of voxel data, the regular cube complex is a very convenient framework [Kaczynski et al. 2004].

A regular cube complex can be easily specified by an n-dimensional array of binary values (bitmap) and a regular n-dimensional cube is placed where the bit is on. For example, the cube complex shown in Figure 3 can be created by specifying the bitmap array

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

A bitmap suffices to describe the top-level cubes, but a cube array (like simplex array) is used during construction of differential operators and for computing faces. In this article we describe the construction of exterior derivative, Hodge star, and Whitney forms on simplicial complexes. For cube complexes we describe only the construction of exterior derivative and lower-dimensional faces. However, the other operators and objects are also implemented in PyDEC for such complexes. For example, Whitney-like elements for hexahedral grids are described in Bochev and Robinson [2002] and are implemented in PyDEC.

Converting a bitmap representation of a mesh into a cube array representation is straightforward. For example, the cube array representation of the mesh in Figure 3 is

$$\mathbb{C}_2 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix}.$$

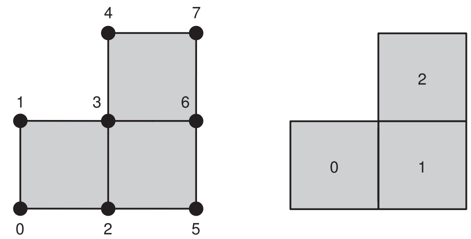


Fig. 3. Regular cube mesh with enumerated vertices and faces.

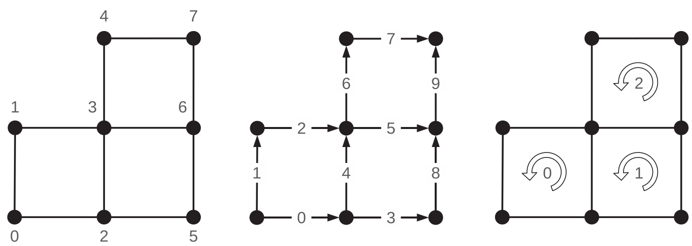


Fig. 4. Regular cube complex with oriented edges and faces.

As with the simplex arrays, the rows of $\mathbb{C}_2$ correspond to individual two-dimensional cubes in the mesh. The two leftmost columns of $\mathbb{C}_2$ encode the origins of each two-dimensional cube, namely $(0, 0)$, $(1, 0)$, and $(1, 1)$. The remaining two columns encode the coordinate directions spanned by the cube. Since $\mathbb{C}_2$ represents the top-level elements, all cubes span both the x (coordinate 0) and y (coordinate 1) dimensions. In general, the first n columns of $\mathbb{C}_k$ encode the origin or corner of a cube while the remaining k columns identify the coordinate directions swept out by the cube. We note that the cube array representation is similar to the cubical notation used by Sen [2003].

The edges of the mesh in Figure 3 are represented by the cube array

$$\mathbb{C}_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 0 \\ 2 & 0 & 1 \\ 2 & 1 & 1 \end{bmatrix}$$

where again the first two columns encode the origin of each edge and the last column indicates whether the edge points in the x or y direction. For example, the row $[0, 0, 0]$ corresponds to edge 0 in Figure 4 which begins at $(0, 0)$ and extends one unit in the x direction. Similarly the row $[2, 1, 1]$ encodes an edge starting at $(2, 1)$ extending one

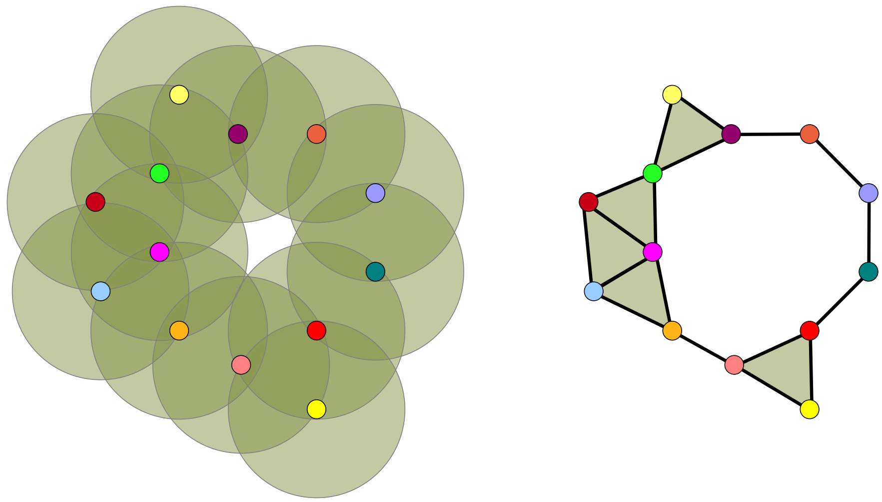


Fig. 5. Broadcast radii and Rips complex for a sensor network.

unit in the y direction. Since zero-dimensional cubes (points) have no spatial extent their cube array representation

$$\mathbb{C}_0 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 2 & 0 \\ 2 & 1 \\ 2 & 2 \end{bmatrix}$$

contains only their coordinate locations.

The cube array provides a convenient representation for regular cube complexes. While a bitmap representation of the top-level cubes is generally more compact, the cube array representation generalizes naturally to lower-dimensional faces and is straightforward to manipulate.

5. RIPS COMPLEX REPRESENTATION

The *Rips complex*, or Vietoris-Rips complex of a point set, is defined by forming a simplex for every subset of vertices with diameter less than or equal to a given distance r. For example, if pair of vertices (v_i, v_j) are no more than distance r apart, then the Rips complex contains an edge (1-simplex) between the vertices. In general, a set of $p \geq 2$ vertices forms a $(p-1)$-simplex when all pairs of vertices in the set are separated by at most r.

In recent work, certain sensor network coverage problems have been shown to reduce to finding topological properties of the network's Rips complex, at least for an abstract model of sensor networks [de Silva and Ghrist 2007]. Such coordinate-free methods rely only on pairwise communication between nodes and do not require the

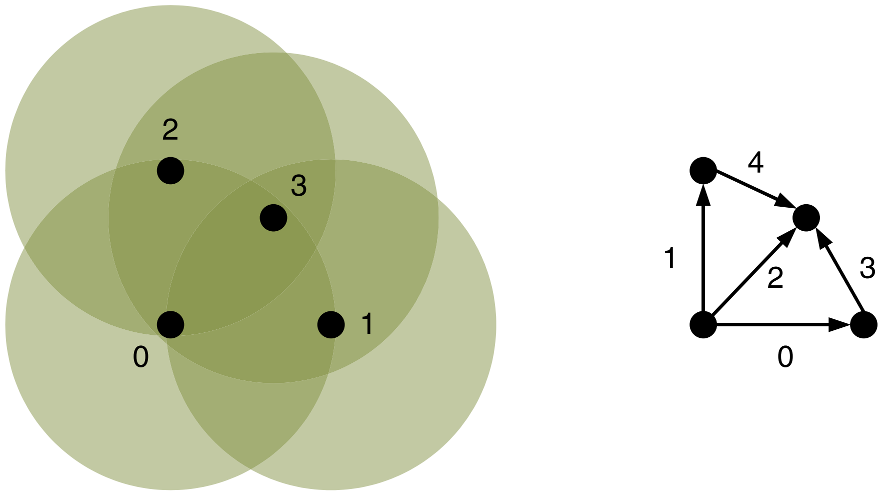


Fig. 6. Five directed edges form the 1-skeleton of the Rips complex.

use of positioning devices. These traits are especially important in the context of ad hoc wireless networks with limited per-node resources. Figure 5 depicts a planar sensor network and its associated Rips complex.

In this section we describe an efficient method for computing the Rips complex for a set of points. Although we consider only the case of points embedded in Euclidean space, our methodology applies to more general metric spaces. Indeed, only the construction of the 1-skeleton of the Rips complex requires metric information. The higher-dimensional simplices are constructed directly from the 1-skeleton.

We compute the 1-skeleton of the Rips complex with a kD-tree data structure. Specifically, for each vertex v_i we compute the set of neighboring vertices $\{v_j : \|v_j - v_i\| \leq r\}$. The hierarchical structure of the kD-tree allows such queries to be computed efficiently.

The 1-skeleton of the Rips complex is stored in an array $\mathbb{S}_1$, using the convention discussed in Section 3. Additionally, the (oriented) edges of the 1-skeleton are used to define $\mathbb{E}$, a directed graph stored in a sparse matrix format. Specifically, $\mathbb{E}(i, j) = 1$ if $[i, j]$ is an edge of the Rips complex, and zero otherwise. For the Rips complex depicted in Figure 6,

$$\mathbb{S}_1 = \begin{bmatrix} 0 & 1 \\ 0 & 2 \\ 0 & 3 \\ 1 & 3 \\ 2 & 3 \end{bmatrix} \qquad \mathbb{E} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

are the corresponding simplex array and directed graph, respectively.

The arrays of higher-dimensional simplices $\mathbb{S}_2, \mathbb{S}_3, \ldots$ can be computed as follows. Let $\mathbb{F}_p$ denote the (sparse) matrix whose rows are identified with the p-simplices as specified by $\mathbb{S}_p$. Each row of $\mathbb{F}_p$ encodes the vertices which form the corresponding simplex. Specifically, $\mathbb{F}_p(i, j)$ takes the value 1 if the i-th simplex contains vertex j and zero otherwise. For the example shown in Figure 6,

$$\mathbb{F}_1 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

encodes the edges stored in $\mathbb{S}_1$. Once $\mathbb{F}_p$ is constructed we compute the sparse matrix-matrix product $\mathbb{F}_p\,\mathbb{E}$. For our example the result is

$$\mathbb{F}_1\,\mathbb{E} = \begin{bmatrix} 0 & 1 & 1 & 2 \\ 0 & 1 & 1 & 2 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Like $\mathbb{F}_p$, the product $\mathbb{F}_p\,\mathbb{E}$ is a matrix that relates the p-simplices to the vertices: the matrix entry (i, j) of $\mathbb{F}_p\,\mathbb{E}$ counts the number of directed edges that exist from the vertices of simplex i to vertex j. When the value of (i, j) entry of $\mathbb{F}_p\,\mathbb{E}$ is equal to $p + 1$, we form a p-simplex of the Rips complex by concatenating simplex i with vertex j. In the example, matrix entries $(0, 3)$ and $(1, 3)$ of $\mathbb{F}_1\,\mathbb{E}$ are equal to 2 which implies that the 2-skeleton of the Rips complex contains two simplices, formed by appending vertex 3 to the 1-simplices $[0, 1]$ and $[0, 2]$, or

$$\mathbb{S}_2 = \begin{bmatrix} 0 & 1 & 3 \\ 0 & 2 & 3 \end{bmatrix}$$

in array format. This process may be applied recursively to develop higher-dimensional simplices $\mathbb{S}_3, \mathbb{S}_4, \ldots$ as required by the application. Thus our algorithm computes simplices of the Rips complex with a handful of sparse and dense matrix operations.

6. ABSTRACT SIMPLICIAL COMPLEX REPRESENTATION

In Section 5 we saw an example of a simplicial complex which was not a manifold complex (Figure 5). Rips complexes described in Section 5 demonstrate one way to construct such complexes in PyDEC, starting from locations of vertices. There are other applications, for example, in topology, where we would like to create a simplicial complex that is not necessarily a manifold. In addition we would like to do this without requiring that the location of vertices be given. For example, in topology, surfaces are often represented as a polygon with certain sides identified. One way to describe such an object is as an *abstract simplicial complex* [Munkres 1984, Section 3]. This is a collection of finite nonempty sets such that if a set is in the collection, so are all the nonempty subsets of it. Figure 7 shows two examples of abstract simplicial complexes created in PyDEC.

In PyDEC, abstract simplicial complexes are created by specifying a list of arrays. Each array contains simplices of a single dimension, specified as an array of vertex numbers. Lower-dimensional faces of a simplex need not be specified explicitly. For

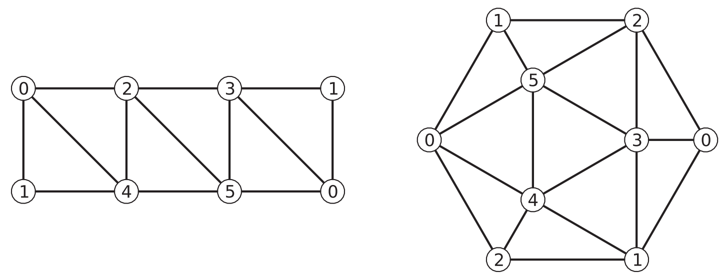


Fig. 7. Examples of abstract simplicial complexes. The one of the left represents the triangulation of a Möbius strip and the one on the right that of a projective plane.

example, the Möbius strip triangulation shown in Figure 7 can be created by giving the array

$$\begin{bmatrix} 0 & 1 & 3 \\ 0 & 3 & 5 \\ 3 & 2 & 5 \\ 5 & 2 & 4 \\ 2 & 0 & 4 \\ 0 & 1 & 4 \end{bmatrix}$$

as input to PyDEC. Abstract simplicial complexes need not be a triangulation of a manifold. For example, one consisting of 2 triangles with an extra edge attached and a stand-alone vertex may be created using a list consisting of the arrays

$$\begin{bmatrix} 5 \end{bmatrix} \qquad \begin{bmatrix} 1 & 4 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \end{bmatrix}$$

as input.

The boundary matrices of a simplicial complex encode the connectivity information and can be computed from a purely combinatorial description of a simplicial complex. The locations of the vertices are not required. Thus the abstract simplicial complex structure is all that is required to compute these matrices as will be described in the next section.

7. DISCRETE EXTERIOR DERIVATIVE

Given a manifold M, the exterior derivative $\mathrm{d} : \Omega^p(M) \to \Omega^{p+1}$, that acts on differential p-forms, generalizes the derivative operator of calculus. When combined with metric-dependent operators Hodge star, sharp, and flat appropriately, the vector calculus operators div, grad, and curl can be generated from d. But d itself is a metric-independent operator whose definition does not require any Riemannian metric on the manifold. See Abraham et al. [1988] for details. The discrete exterior derivative (which we will also denote as d) in PyDEC is defined as is usual in the literature, as the coboundary operator of algebraic topology [Munkres 1984]. Thus

$$\langle \mathrm{d}_p\, a, c\rangle = \langle a, \partial_{p+1}\, c\rangle$$

for arbitrary p-cochain a and $(p + 1)$-chain c. Recall that the boundary operator on cochains, $\partial_p : C_p(K) \to C_{p-1}(K)$ is defined by extension of its definition on an oriented simplex. The boundary operator on a p-simplex $\sigma^p = [v_0, \ldots, v_p]$ is given in terms of its $(p-1)$-dimensional faces ($p+1$ in number) as

$$\partial_p \sigma^p = \sum_{i=0}^{p} (-1)^i \big[v_0, \ldots, \widehat{v_i}, \ldots, v_p \big] , \tag{7.1}$$

where $\widehat{v_i}$ means that v_i is omitted. Therefore, given an n-dimensional simplicial complex represented by $\mathbb{S}_0, \ldots, \mathbb{S}_n$, for the discrete exterior derivative, it suffices to compute $\partial_0, \ldots, \partial_n$. As is usual in algebraic topology, in PyDEC we compute matrix representations of these in the elementary chain basis. Boundary matrices are useful in finite elements since their transposes are the coboundary operators. They are also useful in computational topology since homology and cohomology groups are the quotient groups of kernel and image of boundary matrices [Munkres 1984].

For the complex pictured in Figure 2 the boundary operators are

$$\partial_0 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \qquad \partial_1 = \begin{bmatrix} -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & -1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix},$$

$$\partial_2 = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & -1 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \\ 0 & 0 & -1 \end{bmatrix}.$$

In the following we describe an algorithm that takes as input $\mathbb{S}_n$ and computes both $\mathbb{S}_{n-1}$ and ∂_{n-1}. This procedure is applied recursively to produce all faces of the complex and their boundary operator at that dimension.

The first step of the algorithm converts a simplex array $\mathbb{S}_n$ into a canonical format. In the canonical format each simplex (row of $\mathbb{S}_n$) is replaced by the simplex with sorted indices and the relative parity of the original ordering to the sorted order. For instance, the simplex $(1, 3, 2)$ becomes $((1, 2, 3), -1)$ since an odd number of transpositions (namely one) are needed to transform $(1, 3, 2)$ into $(1, 2, 3)$. Similarly, the canonical format for simplex $(2, 1, 4, 3)$ is $((1, 2, 3, 4), +1)$ since an even number of transpositions are required to sort the simplex indices. Since the complex dimension n is typically small (i.e., < 10), a simple sorting algorithm such as insertion sort is employed at this stage. We denote the aforementioned process `canonical_format` $(\mathbb{S}_n) \rightarrow \mathbb{S}_n^+$ where the rightmost column of $\mathbb{S}_n^+$ contains the simplex parity. Applying `canonical_format` to $\mathbb{S}_2$ in our example yields

$$\mathbb{S}_2 = \begin{bmatrix} 0 & 1 & 3 \\ 1 & 2 & 3 \\ 2 & 4 & 3 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 1 & 3 & 1 \\ 1 & 2 & 3 & 1 \\ 2 & 3 & 4 & -1 \end{bmatrix} = \mathbb{S}_2^+.$$

Once a simplex array $\mathbb{S}_n$ has been transformed into canonical format, the $(n-1)$-dimensional faces $\mathbb{S}_{n-1}$ and boundary operator ∂_n are readily obtained. We denote this process

$$\texttt{boundary\_faces}(\mathbb{S}_n^+) \to \mathbb{S}_{n-1}, \partial_n \ .$$

In order to establish the correspondence between the n-dimensional simplices and their faces, we first enumerate the simplices by adding another column to $\mathbb{S}_n^+$ to form $\mathbb{S}_n^{++}$. For example,

$$\mathbb{S}_2^+ = \begin{bmatrix} 0 & 1 & 3 & 1 \\ 1 & 2 & 3 & 1 \\ 2 & 3 & 4 & -1 \end{bmatrix} \to \mathbb{S}_2^{++} = \begin{bmatrix} 0 & 1 & 3 & 1 & 0 \\ 1 & 2 & 3 & 1 & 1 \\ 2 & 3 & 4 & -1 & 2 \end{bmatrix}.$$

The formula (7.1) is applied to $\mathbb{S}_n^{++}$ in a columnwise fashion by excluding the i-th column of simplex indices, multiplying the parity column by $(-1)^i$, and carrying the last column over unchanged. For example,

$$\mathbb{S}_2^{++} = \begin{bmatrix} 0 & 1 & 3 & 1 & 0 \\ 1 & 2 & 3 & 1 & 1 \\ 2 & 3 & 4 & -1 & 2 \end{bmatrix} \to \begin{bmatrix} 1 & 3 & 1 & 0 \\ 2 & 3 & 1 & 1 \\ 3 & 4 & -1 & 2 \\ 0 & 3 & -1 & 0 \\ 1 & 3 & -1 & 1 \\ 2 & 4 & 1 & 2 \\ 0 & 1 & 1 & 0 \\ 1 & 2 & 1 & 1 \\ 2 & 3 & -1 & 2 \end{bmatrix}.$$

The resultant array is then sorted by the first n columns in lexicographical order, allowing the unique faces to then be extracted.

$$\begin{bmatrix} 1 & 3 & 1 & 0 \\ 2 & 3 & 1 & 1 \\ 3 & 4 & -1 & 2 \\ 0 & 3 & -1 & 0 \\ 1 & 3 & -1 & 1 \\ 2 & 4 & 1 & 2 \\ 0 & 1 & 1 & 0 \\ 1 & 2 & 1 & 1 \\ 2 & 3 & -1 & 2 \end{bmatrix} \to \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 3 & -1 & 0 \\ 1 & 2 & 1 & 1 \\ 1 & 3 & 1 & 0 \\ 1 & 3 & -1 & 1 \\ 2 & 3 & 1 & 1 \\ 2 & 3 & -1 & 2 \\ 2 & 4 & 1 & 2 \\ 3 & 4 & -1 & 2 \end{bmatrix} \to \begin{bmatrix} 0 & 1 \\ 0 & 3 \\ 1 & 2 \\ 1 & 3 \\ 2 & 3 \\ 2 & 4 \\ 3 & 4 \end{bmatrix}$$

Furthermore, a Compressed Sparse Row (CSR) [Saad 2003] sparse matrix representation of ∂_n as

$$\partial_n = (\texttt{ptr}, \texttt{indices}, \texttt{data})$$

is obtained from the sorted matrix. For ∂_2 these are

$$\texttt{ptr} = \begin{bmatrix} 0 & 1 & 2 & 3 & 5 & 7 & 8 & 9 \end{bmatrix},$$
$$\texttt{indices} = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 2 & 2 & 2 \end{bmatrix},$$
$$\texttt{data} = \begin{bmatrix} 1 & -1 & 1 & 1 & -1 & 1 & -1 & 1 & -1 \end{bmatrix},$$

where `indices` and `data` correspond to the fourth and third rows of the sorted matrix.

This process is then applied to $\mathbb{S}_{n-1}$ and so on down the dimension. Since the lower-dimensional simplex array rows are already sorted, those arrays are already in canonical format. Thus a single algorithm generates the lower-dimensional faces as well as the boundary matrices at all the dimensions. The boundary matrices, and hence the coboundary operators, are generated in a convenient sparse matrix format.

7.1. Generalization to Abstract Complexes

The boundary operators and faces of an abstract simplicial complex are computed with a straightforward extension of the `boundary_faces` algorithm. Recall from Section 6 that an abstract simplicial complex is specified by a *list* of simplex arrays of different dimensions, where the lower-dimensional simplex arrays represent simplices that are not a face of any higher-dimensional simplex. Generalizing the previous scheme to the case of abstract simplicial complexes is accomplished by: (1) augmenting the set of computed faces with the user-specified simplices and (2) modifying the computed boundary operator accordingly.

Consider the abstract simplicial complex represented by the simplex arrays

$$\mathbb{S}_0 = [5] \qquad \mathbb{S}_1 = [1, 4] \qquad \mathbb{S}_2 = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \end{bmatrix}$$

which consists of two triangles, an edge, and an isolated vertex. Applying `boundary_faces` to $\mathbb{S}_2$ produces an array of face edges and corresponding boundary operator

$$\texttt{boundary\_faces}(\mathbb{S}_2) \rightarrow \mathbb{S}_1, \partial_2 = \begin{bmatrix} 0 & 1 \\ 0 & 2 \\ 1 & 2 \\ 1 & 3 \\ 2 & 3 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 1 & 1 \\ 0 & -1 \\ 0 & 1 \end{bmatrix},$$

which includes all but the user-specified edge $[1, 4]$. User-specified simplices are then incorporated into the simplex array in a three-stage process: (1) user-specified simplices are concatenated to the computed face array; (2) the rows of the combined simplex array are sorted lexicographically; (3) redundant simplices (if any) are removed from the sorted array. Upon completion, the augmented simplex array contains the union of the face simplices and the user-specified simplices. Continuing the example, the edge $[1, 4]$ is incorporated into $\mathbb{S}_1$ as follows.

$$\begin{bmatrix} 0 & 1 \\ 0 & 2 \\ 1 & 2 \\ 1 & 3 \\ 2 & 3 \end{bmatrix}, \begin{bmatrix} 1 & 4 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 1 \\ 0 & 2 \\ 1 & 2 \\ 1 & 3 \\ 2 & 3 \\ 1 & 4 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 1 \\ 0 & 2 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \\ 2 & 3 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 1 \\ 0 & 2 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \\ 2 & 3 \end{bmatrix} = \mathbb{S}_1$$

In the final stage of the procedure, the computed boundary operator (∂_2 in the example) is updated to reflect the newly incorporated simplices. Since the new simplices do not lie in the boundary of any higher-dimensional simplex, we may simply add empty rows into the sparse matrix representation of the boundary operator for each newly added simplex. Therefore, the boundary operator update procedure amounts to a simple remapping of row indices. In the example, the addition of the edge $[1, 4]$ into the fifth row of the simplex array requires the addition of an empty row into the boundary operator at the corresponding position,

$$\begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 1 & 1 \\ 0 & -1 \\ 0 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 1 & 1 \\ 0 & -1 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} = \partial_2 \, .$$

The Rips complex of Section 5 does have the location information for the vertices. However, ignoring those, such a complex is an abstract simplicial complex. Thus the boundary matrices for a Rips complex can be computed as described earlier. In practice some efficiency can be obtained by ordering the computation differently, so that the matrices are built as the complex is being built from the edge skeleton of the Rips complex. That is how it is implemented in PyDEC.

7.2. Boundary Operators and Faces for Cubical Complexes

The algorithm used to compute the faces and boundary operator of a given cube array ($\mathbb{C}_p \rightarrow \mathbb{C}_{p-1}, \partial_p$) is closely related to the procedure discussed in Section 7 for simplex arrays. Consider a general p-cube in n dimensions, denoted by the pair $\big[(c_0, \ldots, c_{n-1})(d_0, \ldots, d_{p-1})\big]$ where $(c_0, \ldots, c_{n-1})$ are the coordinates of the cube's origin and $(d_0, \ldots, d_{p-1})$ are the directions which the p-cube spans. Note that the values $[c_0, \ldots, c_{n-1}, d_0, \ldots, d_{p-1}]$ correspond exactly to a row of the cube array representation introduced in Section 4. Using this notation, the boundary of a p-cube is given by the expression

$$\partial_p\big[(c_0, \ldots, c_{n-1})(d_0, \ldots, d_{p-1})\big] = \sum_{i=0}^{p-1} (-1)^i \Big(\big[(c_0, \ldots, c_{d_i} + 1, \ldots, c_{n-1})(d_0, \ldots, \widehat{d_i}, \ldots, d_{p-1})\big] - \big[(c_0, \ldots, c_{d_i} + 0, \ldots, c_{n-1})(d_0, \ldots, \widehat{d_i}, \ldots, d_{p-1})\big] \Big) , \quad (7.2)$$

where $\widehat{d_i}$ denotes the omission of the i-th spanning direction and c_{d_i} is the corresponding coordinate. For example, the boundary of a square centered at the location $(10, 20)$ is

$$\partial_2\big[(10, 20)(0, 1)\big] = \big[(11, 20)(1)\big] - \big[(10, 20)(1)\big] - \big[(10, 21)(0)\big] + \big[(10, 20)(0)\big]. \quad (7.3)$$

The canonical format for a p-cube is the one where the spanning directions are specified in ascending order. For instance, the 2-cube $\big[(10, 20)(0, 1)\big]$ is in the canonical format because $d_0 < d_1$. As with simplices, each cube has a unique canonical format through which duplicates are easily identified. Since the top-level cube array $\mathbb{C}_n$ is generated from a bitmap it is already in the canonical format and no reordering of indices or parity tracking is necessary.

Applying Eq. (7.2) to a p-cube array with N members generates a collection $2N$ oriented faces. In the mesh illustrated in Figure 3 the three squares in $\mathbb{C}_2$ are initially expanded into

$$\mathbb{C}_2 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 0 & 1 & -1 & 0 \\ 1 & 0 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 & 2 \\ 1 & 0 & 1 & 1 & 0 \\ 2 & 0 & 1 & 1 & 1 \\ 2 & 1 & 1 & 1 & 2 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 2 \\ 0 & 1 & 0 & -1 & 0 \\ 1 & 1 & 0 & -1 & 1 \\ 1 & 2 & 0 & -1 & 2 \end{bmatrix} = \mathbb{C}_1^+,$$

where the fourth column of $\mathbb{C}_1^+$ encodes the orientation of the face and the fifth column records the 2-cube to which each face belongs. Sorting the rows of $\mathbb{C}_1^+$ in lexicographical order

$$\begin{bmatrix} 0 & 0 & 1 & -1 & 0 \\ 1 & 0 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 & 2 \\ 1 & 0 & 1 & 1 & 0 \\ 2 & 0 & 1 & 1 & 1 \\ 2 & 1 & 1 & 1 & 2 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 2 \\ 0 & 1 & 0 & -1 & 0 \\ 1 & 1 & 0 & -1 & 1 \\ 1 & 2 & 0 & -1 & 2 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 1 & 0 & -1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & -1 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 2 \\ 1 & 1 & 0 & -1 & 1 \\ 1 & 1 & 1 & -1 & 2 \\ 1 & 2 & 0 & -1 & 2 \\ 2 & 0 & 1 & 1 & 1 \\ 2 & 1 & 1 & 1 & 2 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 0 \\ 2 & 0 & 1 \\ 2 & 1 & 1 \end{bmatrix} = \mathbb{C}_1$$

allows the unique faces to be extracted. Lastly, a sparse matrix representation of the boundary operator is obtained from the sorted cube array in the same manner as for simplices.

8. REVIEW OF WHITNEY MAP AND WHITNEY FORMS

In this section we review and collect some material, most of which is well-known in DEC and finite element exterior calculus research communities. It is included here partly to fix notation. In this section we also give the monomials-based definition of inner product of differential forms. This is not the way inner product of forms is usually defined in most textbooks, Morita [2001] being one exception we know of. The monomial form leads to an efficient algorithm for computation of stiffness and mass matrices for Whitney forms given in Section 9.

The basic function spaces that are useful with exterior calculus are the space of square integrable p-forms on a manifold and Sobolev spaces derived from that. Let M be a *Riemannian manifold*, a manifold on which a smoothly varying inner product is defined on the tangent space at each point. Let g be its *metric*, a smooth tensor field that defines the inner product on the tangent space at each point on M.

For differential forms on such a manifold M, the space of square integrable forms is denoted $L^2\Omega^p(M)$. One can then define the spaces $H\Omega^p(M)$ which generalize the spaces $H(\text{div})$ and $H(\text{curl})$ used in mixed finite element methods [Arnold et al. 2010]. To define $L^2\Omega^p(M)$ one has to define an inner product on the space of forms which is our starting point for this section. All these function spaces have been discussed in Arnold et al. [2006, 2010]. The definitions and properties of Whitney map and Whitney forms is in Dodziuk [1976], the geometric analysis background is in Jost [2005], and the basic definition of inner products on forms is in Abraham et al. [1988, page 411].

8.1. Inner Product of Forms

To define the spaces $L^2\Omega^p(M)$ and $H\Omega^p(M)$ more precisely, we recall the definitions related to inner products of forms. We will need the exterior calculus operators wedge product and Hodge star which we recall first. For a manifold M the *wedge product* $\wedge : \Omega^p(M) \times \Omega^q(M) \to \Omega^{p+q}(M)$ is an operator for building $(p+q)$-forms from p-forms and q-forms. It is defined as the skew symmetrization of the tensor product of the two forms involved. For a Riemannian manifold of dimension n, the Hodge star operator $* : \Omega^p(M) \to \Omega^{n-p}(M)$ is an isomorphism between the spaces of p and $(n-p)$-forms. For more details, see Abraham et al. [1988, page 394] for wedge products and Abraham et al. [1988, page 411] for Hodge star.

Definition 8.1. Given two smooth p-forms $\alpha, \beta \in \Omega^p(M)$ on a Riemannian manifold M, their *pointwise inner product* at point $x \in M$ is defined by

$$\langle \alpha(x), \beta(x) \rangle \, \mu = \alpha(x) \wedge * \beta(x) \,, \tag{8.1}$$

where $\mu = *1$ is the volume form associated with the metric induced by the inner product on M.

The pointwise inner products of forms can be defined in another way, which will be more useful to us in computations. The second definition given shortly in Definition 8.2 is equivalent to the one given before in Definition 8.1. The operator $\sharp$ (the *sharp* operator) used next is an isomorphism between 1-forms and vector fields and is defined by $g(\alpha^\sharp, X) = \alpha(X)$ for given 1-form α and all vector fields X. See Abraham et al. [1988] for details.

Definition 8.2. Let $\alpha_1, \dots, \alpha_p$ and $\beta_1, \dots, \beta_p$ be 1-forms on a Riemannian manifold M. By analogy with polynomials we'll call p-forms of the type $\alpha_1 \wedge \cdots \wedge \alpha_p$ and $\beta_1 \wedge \cdots \wedge \beta_p$ *monomial p-forms*. Define the following operator at a point $x \in M$:

$$\left\langle \alpha_1 \wedge \cdots \wedge \alpha_p, \beta_1 \wedge \cdots \wedge \beta_p \right\rangle := \det\left[g\left(\alpha_i^\sharp, \beta_j^\sharp\right)\right] , \tag{8.2}$$

where $\left[g\left(\alpha_i^\sharp, \beta_j^\sharp\right)\right]$ is the matrix obtained by taking $1 \le i, j \le p$. In the preceding equation: all the 1-forms are evaluated at the point $x \in M$. Extend the operation in (8.2) bilinearly pointwise to the space of all p-forms. It can be shown that this defines a *pointwise inner product* of p-forms equivalent to the one defined in (8.1). Note that if $\alpha_i = \beta_i$ for all i, the expression on the right in (8.2) is the Gram determinant.

Remark 8.3. Note that unlike (8.1) the definition in (8.2) does not involve wedge product and Hodge star explicitly. This is an advantage of the latter form since a discrete wedge product is not available in PyDEC. The RHS of (8.2) does involve the sharp operator, but as we will see in the next section, this is easy to interpret for the purpose of discretization in this context.

Definition 8.4. The pointwise innerproduct in (8.1), or equivalently in (8.2), induces an L^2 *inner product on* M as

$$(\alpha, \beta)_{L^2} = \int_M \langle \alpha(x), \beta(x) \rangle \, \mu \,. \tag{8.3}$$

The space of p-forms obtained by completion of $\Omega^p(M)$ under this inner product is the Hilbert space of *square integrable p-forms* $L^2\Omega^p(M)$. The other useful space mentioned at the beginning of this section is the Sobolev space $H\Omega^p(M) := \{\alpha \in L^2\Omega^p(M) \mid \mathrm{d}\alpha \in L^2\Omega^{p+1}(M)\}$.

8.2. Whitney Map and Whitney Forms

Let K be an n-dimensional manifold simplicial complex embedded in $\mathbb{R}^N$ and $|K|$ the underlying space. The metric on the interiors of the top dimensional simplices of K will be the one induced from the embedding Euclidean space $\mathbb{R}^N$. As is usual in finite element methods, finite dimensional subspaces of the function spaces described in the previous paragraph are used in the numerical solution of PDEs. The finite dimensional spaces can be obtained by "embedding" the space of cochains into these spaces by using an interpolation. For example, to embed $C^p(K;\mathbb{R})$ into $L^2\Omega^p\big(|K|\big)$ one can use the Whitney map $\mathrm{W} : C^p(K;\mathbb{R}) \to L^2\Omega^p\big(|K|\big)$, which will be reviewed in this subsection. The image $\mathrm{W}\big(C^p(K;\mathbb{R})\big)$ is a linear vector subspace of $L^2\Omega^p\big(|K|\big)$ and is the space of *Whitney p-forms* [Bossavit 1988; Dodziuk 1976; Whitney 1957] and is denoted $\mathcal{P}_1^-\Omega^p(|K|)$ in Arnold et al. [2009, 2010]. (We use Ω^p instead of Λ^p used in Arnold et al. [2010].) The embedding of cochains is analogous to how scalar values at discrete sample points would be interpolated to get a piecewise affine function. In the scalar case also, the space of such functions is a vector subspace of square integrable functions. In fact, $\mathrm{W}\big(C^0(K;\mathbb{R})\big)$, the space of Whitney 0-forms, is the space of continuous piecewise affine functions on $|K|$. The Whitney map for $p > 0$ is actually built from barycentric coordinates that are the building blocks of piecewise linear interpolation. Thus the embedding of $C^p(K;\mathbb{R})$ into $L^2\Omega^p\big(|K|\big)$ for $p > 0$ can be considered to be a generalization of the embedding of $C^0(K;\mathbb{R})$ into $L^2\Omega^0\big(|K|\big)$. Thus Whitney forms enable only low-order methods. However, arbitrary degree polynomial spaces suitable for use in finite element exterior calculus have been discovered [Arnold et al. 2006, 2009]. These, however, are not yet a part of PyDEC.

The space of Whitney p-forms is the space of piecewise smooth differential p-forms obtained by applying the Whitney map to p-cochains. It can be thought of as a method for interpolating values given on p-simplices of a simplicial complex. For example, inside a tetrahedron Whitney forms allow the interpolation of numbers on edges or faces to a smooth 1-form or 2-form, respectively. As mentioned before, for 0-cochains, that is, scalar functions sampled at vertices, the interpolation is the one obtained using the standard scalar piecewise affine basis functions on each simplex, that is the barycentric coordinates corresponding to each vertex of the simplex. We recall the definition of barycentric coordinates followed by the definition of the Whitney map.

Definition 8.5. Let $\sigma^p = [v_0, \dots, v_p]$ be a p-simplex embedded in $\mathbb{R}^N$. The affine functions $\mu_i : \mathbb{R}^N \to \mathbb{R}$, $i = 0, \dots, p$, which when restricted to σ^p take the value 1 on vertex v_i and 0 on the others, are called the *barycentric coordinates* in σ^p.

Definition 8.6. Let σ^p be an oriented p-simplex $[v_{i_0}, \ldots, v_{i_p}]$ in an n-dimensional manifold complex K, and $\left(\sigma^p\right)^*$ the corresponding elementary p-cochain. We define

$$\mathrm{W}\left(\left(\sigma^p\right)^*\right) := p! \sum_{k=0}^{p} (-1)^k \mu_{i_k}\, d\mu_{i_0} \wedge \cdots \wedge \widehat{d\mu_{i_k}} \wedge \cdots \wedge d\mu_{i_p}\,, \tag{8.4}$$

where μ_{i_k} is the barycentric coordinate function with respect to vertex v_{i_k} and the notation $\widehat{d\mu_{i_k}}$ indicates that the term $d\mu_{i_k}$ is omitted from the wedge product. The *Whitney map* $\mathrm{W} : C^p(K;\mathbb{R}) \to L^2\Omega^p\left(|K|\right)$ is the preceding map W extended to all of $C^p(K;\mathbb{R})$ by requiring that W be a linear map. $\mathrm{W}((\sigma^p)^*)$ is called the *Whitney form* corresponding to σ^p, and for a general cochain c, $\mathrm{W}(c)$ is called the Whitney form corresponding to c.

For example, the Whitney form corresponding to the edge $[v_0, v_1]$ is $\mathrm{W}([v_0, v_1]^*) = \mu_0\, \mathrm{d}\,\mu_1 - \mu_1\, \mathrm{d}\,\mu_0$, and the Whitney form corresponding to the triangle $[v_1, v_2, v_3]$ in a tetrahedron $[v_0, v_1, v_2, v_3]$ is

$$\mathrm{W}([v_1, v_2, v_3]^*) = 2\,(\mu_1\, \mathrm{d}\,\mu_2 \wedge \mathrm{d}\,\mu_3 - \mu_2\, \mathrm{d}\,\mu_1 \wedge \mathrm{d}\,\mu_3 + \mu_3\, \mathrm{d}\,\mu_1 \wedge \mathrm{d}\,\mu_2)\,.$$

Remark 8.7. If we were using local coordinate charts on a manifold then at any point a p-form would be a linear combination of monomials. Note from (8.4) that the Whitney form $\mathrm{W}(\sigma^*)$ is a sum of monomials with coefficients. Thus Whitney forms allow us to treat forms at a point as a linear combination of monomials even though we are not using local coordinate charts.

We emphasize again that this section was a review of known material. We have tried to present this material in a manner that makes it easier to explain the examples of Section 11 and the construction of mass matrix for Whitney forms described in the next section.

9. WHITNEY INNER PRODUCT OF COCHAINS

Given a manifold simplicial complex K, an inner product between two p-cochains a and b can be defined by first embedding these cochains into $L^2\Omega^p\left(|K|\right)$ using Whitney map and then taking the L^2 inner product of the resulting Whitney forms [Dodziuk 1976].

Definition 9.1. Given two p-cochains $a, b \in C^p(K;\mathbb{R})$, their *Whitney inner product* is defined by

$$(a, b) := (\mathrm{W}\,a, \mathrm{W}\,b)_{L^2} = \int_{|K|} \langle \mathrm{W}\,a, \mathrm{W}\,b\rangle \mu\,, \tag{9.1}$$

using the L^2 inner product on forms given in (8.3). The matrix for Whitney inner product of p-forms in the elementary p-cochain basis will be denoted M_p. That is, M_p is a square matrix of order N_p (the number of p-simplices in K) such that the entry in row i and column j is $M_p(i, j) = \left((\sigma_i^p)^*, (\sigma_j^p)^*\right)$, where $(\sigma_i^p)^*$ and $(\sigma_j^p)^*$ are the elementary p-cochains corresponding to the p-simplices σ_i^p and σ_j^p with index number i and j, respectively.

The integral in (9.1) is the sum of integrals over each top dimensional simplex in K. Inside each such simplex the inner product of smooth forms applies since the Whitney form in each simplex is smooth all the way up to the boundary of the simplex. The interior of each top dimensional simplex is given an inner product that is induced from the standard inner product of the embedding space $\mathbb{R}^N$.

Remark 9.2. Given cochains $a, b \in C^p(K; \mathbb{R})$ we will refer to their representations in the elementary cochain basis also as a and b. Then the matrix representation of the Whitney inner product of a and b is $(a, b) = a^T M_p b$.

The inner product of cochains defined in this way is a key concept that connects exterior calculus to finite element methods and different choices of the inner product lead to different discretizations of exterior calculus. This is because the inner product matrix M_p is the mass matrix of finite element methods based on Whitney forms. The details of the efficient computation of M_p for any p and n will be given in Sections 9.2 and 9.3.

Recall that for a Riemannian manifold M, if $\delta_{p+1} : \Omega^{p+1}(M) \to \Omega^p(M)$ is the codifferential, then the Laplace-deRham operator on p-forms is $\Delta_p := \mathrm{d}_{p-1}\, \delta_p + \delta_{p+1}\, \mathrm{d}_p$. For a boundaryless M the codifferential δ_{p+1} is the adjoint of the exterior derivative d_p. In case M has a boundary, we have instead that

$$(\mathrm{d}_p\, \alpha, \beta) = (\alpha, \delta_{p+1}\, \beta) + \int_{\partial M} \alpha \wedge * \beta\,. \tag{9.2}$$

See Abraham et al. [1988, Exercise 7.5E] for a derivation of the preceding. Now consider Poisson's equation $\Delta_p\, u = f$ on p-forms defined on a p-dimensional simplicial manifold complex K. For simplicity, we'll consider the weak form of this using smooth forms rather than Sobolev spaces of forms. See Arnold et al. [2006, 2010] for a proper functional analytic treatment. We will also assume that the correct boundary conditions are satisfied, so that the boundary term in (9.2) is 0. In our simple treatment, the weak form of the Poisson's equation is to find a $u \in \Omega^p(|K|)$ such that $(\delta_p\, u, \delta_p\, v)_{L^2} + (\mathrm{d}_p\, u, \mathrm{d}_p\, v)_{L^2} = (f, v)_{L^2}$. Thus it is clear that a Galerkin formulation using Whitney forms $\mathcal{P}_1^- \Omega^p$ will require the computation of a term like $(\mathrm{d}_p\, \mathrm{W}\, a, \mathrm{d}_p\, \mathrm{W}\, b)_{L^2}$ for cochains a and b. By the commuting property of Whitney forms $\mathrm{d}_p\, W = W\, \mathrm{d}_p$ (where the second d_p is the coboundary operator on cochains) we have that the aforesaid inner product is equal to $(\mathrm{W}\, \mathrm{d}_p\, a, \mathrm{W}\, \mathrm{d}_p\, b)_{L^2}$. (See Dodziuk [1976] for a proof of the commuting property.) Now, by definition of the Whitney inner product of cochains in (9.1) this is equal to $(\mathrm{d}_p\, a, \mathrm{d}_p\, b)$ in the inner product on $(p + 1)$-cochains. The matrix form of this inner product can be obtained from the mass matrix M_{p+1} as $\mathrm{d}_p^T\, M_{p+1}\, \mathrm{d}_p$. This is what we mean when we say that the stiffness matrix can be computed easily from the mass matrix. The term on the right in the weak form will use the mass matrix M_p. Since codifferential of Whitney forms is 0, the first term in the weak form has to be handled in another way, as described in Arnold et al. [2010].

Remark 9.3. Exploiting the aforementioned commutativity of Whitney forms to compute the stiffness matrix represents a significant simplification to our software implementation. While computing the stiffness matrix directly from the definition is possible, it is a complex operation that requires considerable programmer effort, especially if the performance of the implementation is important. In contrast, our formulation requires no additional effort and has the performance of the underlying sparse matrix-matrix multiplication implementation, an optimized and natively compiled routine. All of the complex indexing, considerations of relative orientation, mappings between faces and indices, etc., is reduced to a simple linear algebra expression. The lower-dimensional faces are oriented lexicographically and the orientation information required in stiffness matrix assembly is implicit in the boundary matrices.

9.1. Computing Barycentric Differentials

Given that the Whitney form $\mathrm{W}(\sigma^*)$ in (8.4) is built using wedges of differentials of barycentric coordinates, it is clear that the algorithm for computing an inner product of Whitney forms involves computation of the gradients or differentials of the barycentric coordinates. The following lemma shows how these are computed using simple linear algebra operations.

LEMMA 9.4. *Let $\sigma^p = [v_0, \dots, v_p]$ be a p-simplex embedded in $\mathbb{R}^N$, $p \le N$ where the vertices $v_i \in \mathbb{R}^N$ are given in some basis for $\mathbb{R}^N$. Let $X \in \mathbb{R}^{N\times p}$ be a matrix whose jth column consists of the components of $\mathrm{d}\,\mu_j$ in the dual basis, for $j = 1, \dots, p$. Let $V_0 \in \mathbb{R}^{N\times p}$ be a matrix whose j-th column is $v_j - v_0$, for $j = 1, \dots, p$. Then $X^T = \left(V_0^T V_0\right)^{-1} V_0^T = V_0^+$, the pseudoinverse of V_0.*

PROOF. Let $\zeta = [\mu_1, \dots, \mu_p]^T$ be the vector of barycentric coordinates (other than μ_0) with respect to σ^p, for a point $x = [x_1, \dots, x_N]^T \in \mathbb{R}^N$. Then by definition of barycentric coordinates and simplices, $V_0\zeta = x - v_0$ is the linear least-squares system for the barycentric coordinates. Thus $\zeta = V_0^+(x - v_0)$ which implies that $\mathrm{d}\,\zeta = X^T = V_0^+$. □

Remark 9.5. The use of normal equations in the solution of the least-squares problem in the preceding proposition suffers the well-known condition squaring problem. This is only likely to be a problem if the simplices are nearly degenerate. In that case one can just use an orthogonalization method to compute a QR factorization and use that to solve the least-squares problem. Notice that in typical physical problems V_0 will typically be 2×2, 3×2 or 3×3 matrix so any of these methods are easy to implement.

Once the components for $\mathrm{d}\,\mu_i$ have been obtained for $i = 1, \dots, p$, the components of $\mathrm{d}\,\mu_0$ can be obtained by noting that $\mathrm{d}\,\mu_0 + \cdots + \mathrm{d}\,\mu_p = 0$ which follows from the fact that the barycentric coordinates sum to 1. Also note that the components of the gradients $\nabla\mu_i$ will be the same as those of $\mathrm{d}\,\mu_i$ if the standard metric of Euclidean space is used for the embedding space $\mathbb{R}^N$ which is the case in all of PyDEC.

9.2. Whitney Inner Product Matrix

We will now use the inner product of forms in (8.2) and the cochains inner product defined in (9.1) to give a formula for the computation of M_p, the Whitney inner product matrix for p-cochains. We will also refer to this as the Whitney mass matrix.

NOTATION 9.6. *Given simplices σ and τ the notation $\sigma \succeq \tau$ or $\tau \preceq \sigma$ means τ is a face of σ. Note that this means τ can be equal to σ since any simplex is its own face. For proper inclusion we use $\tau \prec \sigma$ or $\sigma \succ \tau$ to indicate that τ is a proper face of σ. The use of this notation simplifies the expression of summations over various classes of simplices in a complex. For example, given two fixed p-simplices σ_i^p and σ_j^p*

$$\sum_{\substack{\sigma^n \\ \sigma^n \succeq \sigma_i^p, \sigma_j^p}}$$

is read as "sum over all n-simplices σ^n which have σ_i^p and σ_j^p as faces". Another notation used in the proof of the proposition that follows is the star *of a simplex σ, written* $\mathrm{St}(\sigma)$ *(not to be confused with the dual star $\star\sigma$). This star* $\mathrm{St}(\sigma)$ *is the union of the interiors of all simplices of the complex that have σ as a face. That includes σ also. The closure* *of this open set is called the* closed star *and written* $\overline{\mathrm{St}}\,\sigma$. *This is the union of simplices that contain* σ.

PROPOSITION 9.7. *Let* $\sigma_i^p = [v_{i_0}, \ldots, v_{i_p}]$ *and* $\sigma_j^p = [v_{j_0}, \ldots, v_{j_p}]$ *be oriented p-simplices in an n-dimensional manifold simplicial complex K, with* $0 \le p \le n$. *Then the row i, column j entry of the Whitney inner product matrix* M_p *is given by*

$$M_p(i, j) = (p!)^2 \sum_{\substack{\sigma^n \\ \sigma^n \succeq \sigma_i^p, \sigma_j^p}} \sum_{k,l=0}^{p} (-1)^{k+l} c_{kl} \int_{\sigma^n} \mu_{i_k} \mu_{j_l}\, \mu\,,$$

where $c_{kl} = 1$ *for* $p = 0$, *and for* $p > 0$

$$c_{kl} = \det \begin{bmatrix} \langle \mathrm{d}\,\mu_{i_0}, \mathrm{d}\,\mu_{j_0}\rangle & \ldots & \widehat{\langle \mathrm{d}\,\mu_{i_0}, \mathrm{d}\,\mu_{j_l}\rangle} & \ldots & \langle \mathrm{d}\,\mu_{i_0}, \mathrm{d}\,\mu_{j_p}\rangle \\ \vdots & & \vdots & & \vdots \\ \widehat{\langle \mathrm{d}\,\mu_{i_k}, \mathrm{d}\,\mu_{j_0}\rangle} & \ldots & \widehat{\langle \mathrm{d}\,\mu_{i_k}, \mathrm{d}\,\mu_{j_l}\rangle} & \ldots & \widehat{\langle \mathrm{d}\,\mu_{i_k}, \mathrm{d}\,\mu_{j_p}\rangle} \\ \vdots & & \vdots & & \vdots \\ \langle \mathrm{d}\,\mu_{i_p}, \mathrm{d}\,\mu_{j_0}\rangle & \ldots & \widehat{\langle \mathrm{d}\,\mu_{i_p}, \mathrm{d}\,\mu_{j_l}\rangle} & \ldots & \langle \mathrm{d}\,\mu_{i_p}, \mathrm{d}\,\mu_{j_p}\rangle \end{bmatrix},$$

the hats indicating the deleted terms. Here μ *is the volume form corresponding to the standard inner product in* $\mathbb{R}^N$ *and* μ_{i_k} *and* μ_{j_l} *are the barycentric coordinates corresponding to vertices* i_k *and* j_l.

PROOF. See Appendix. □

For the $p = n$ case a simpler formulation is given in Proposition 9.9. The previous proposition shows that computation of the Whitney inner product matrix involves computations of inner products of differentials of barycentric coordinates. Since the only metric implemented in PyDEC is the standard one inherited from the embedding space $\mathbb{R}^N$,

$$\langle \mathrm{d}\,\mu_i, \mathrm{d}\,\mu_j\rangle = g((\mathrm{d}\,\mu_i)^\sharp, (\mathrm{d}\,\mu_j)^\sharp) = \nabla\mu_i \cdot \nabla\mu_j\,.$$

Example 9.8. Consider the simplicial complex corresponding to a tetrahedron σ^3 embedded in $\mathbb{R}^3$ for which we want to compute M_2, the Whitney inner product matrix for 2-cochains. Here $N = n = 3$ and $p = 2$ and M_2 is of order $N_2 = 4$, the number of triangles in the complex. Label the vertices as $0, 1, 2, 3$. Then by PyDEC's lexicographic numbering scheme, the edges numbered 0 to 5 are $[0, 1]$, $[0, 2]$, $[0, 3]$, $[1, 2]$, $[1, 3]$, $[2, 3]$ and the triangles numbered 0 to 3 are $[0, 1, 2]$, $[0, 1, 3]$, $[0, 2, 3]$, and $[1, 2, 3]$. We will describe the computation of the row 0, column 3 entry and the row 0, column 1 entry of M_2. The $(0, 3)$ entry corresponds to the inner product of cochains $[v_0, v_1, v_2]^*$ and $[v_1, v_2, v_3]^*$ since, in the lexicographic ordering and naming convention of PyDEC these are σ_0^2 and σ_3^2 respectively. Thus we are computing

$$\left((\sigma_0^2)^*, (\sigma_3^2)^*\right) = \left(\mathrm{W}(\sigma_0^2)^*, \mathrm{W}(\sigma_3^2)^*\right)_{L^2}.$$

The corresponding Whitney forms are

$$\begin{aligned} \mathrm{W}(\sigma_0^2)^* &= 2!\left(\mu_0\ \mathrm{d}\,\mu_1 \wedge \mathrm{d}\,\mu_2 - \mu_1\,\mathrm{d}\,\mu_0 \wedge \mathrm{d}\,\mu_2 + \mu_2\,\mathrm{d}\,\mu_0 \wedge \mathrm{d}\,\mu_1\right) \\ \mathrm{W}(\sigma_3^2)^* &= 2!\left(\mu_1\ \mathrm{d}\,\mu_2 \wedge \mathrm{d}\,\mu_3 - \mu_2\,\mathrm{d}\,\mu_1 \wedge \mathrm{d}\,\mu_3 + \mu_3\,\mathrm{d}\,\mu_1 \wedge \mathrm{d}\,\mu_2\right). \end{aligned}$$

Then $\big(\mathrm{W}(\sigma_0^2)^*, \mathrm{W}(\sigma_3^2)^*\big)_{L^2}/(2!)^2$ is

$$\int_{\sigma^3} \mu_0\mu_1 \big\langle \mathrm{d}\,\mu_1 \wedge \mathrm{d}\,\mu_2, \mathrm{d}\,\mu_2 \wedge \mathrm{d}\,\mu_3 \big\rangle\, \mu - \int_{\sigma^3} \mu_0\mu_2 \big\langle \mathrm{d}\,\mu_1 \wedge \mathrm{d}\,\mu_2, \mathrm{d}\,\mu_1 \wedge \mathrm{d}\,\mu_3 \big\rangle\, \mu + \ldots\,, \tag{9.3}$$

where μ is just $\mathrm{d}x \wedge \mathrm{d}y \wedge \mathrm{d}z$, the standard volume form in $\mathbb{R}^3$. Each term like $\big\langle \mathrm{d}\,\mu_1 \wedge \mathrm{d}\,\mu_2, \mathrm{d}\,\mu_2 \wedge \mathrm{d}\,\mu_3 \big\rangle$ is

$$\det \begin{bmatrix} \langle \mathrm{d}\,\mu_1, \mathrm{d}\,\mu_2 \rangle & \langle \mathrm{d}\,\mu_1, \mathrm{d}\,\mu_3 \rangle \\ \langle \mathrm{d}\,\mu_2, \mathrm{d}\,\mu_2 \rangle & \langle \mathrm{d}\,\mu_2, \mathrm{d}\,\mu_3 \rangle \end{bmatrix},$$

which in the notation of Proposition 9.7 is

$$\det \begin{bmatrix} \langle \mathrm{d}\,\widehat{\mu_0, \mathrm{d}\,\mu_1} \rangle & \langle \mathrm{d}\,\widehat{\mu_0, \mathrm{d}\,\mu_2} \rangle & \langle \mathrm{d}\,\widehat{\mu_0, \mathrm{d}\,\mu_3} \rangle \\ \langle \mathrm{d}\,\widehat{\mu_1, \mathrm{d}\,\mu_1} \rangle & \langle \mathrm{d}\,\mu_1, \mathrm{d}\,\mu_2 \rangle & \langle \mathrm{d}\,\mu_1, \mathrm{d}\,\mu_3 \rangle \\ \langle \mathrm{d}\,\widehat{\mu_2, \mathrm{d}\,\mu_1} \rangle & \langle \mathrm{d}\,\mu_2, \mathrm{d}\,\mu_2 \rangle & \langle \mathrm{d}\,\mu_2, \mathrm{d}\,\mu_3 \rangle \end{bmatrix}.$$

Using a shorthand notation for matrices like the ones just given, the 2×2 matrices whose determinants need to be computed for calculating the $(0, 3)$ entry of M_2 are given next.

$$\begin{bmatrix} \widehat{01} & \widehat{02} & \widehat{03} \\ \widehat{11} & 12 & 13 \\ \widehat{21} & 22 & 23 \end{bmatrix} \begin{bmatrix} \widehat{02} & \widehat{01} & \widehat{03} \\ \widehat{12} & 11 & 13 \\ \widehat{22} & 21 & 23 \end{bmatrix} \begin{bmatrix} \widehat{03} & \widehat{01} & \widehat{02} \\ \widehat{13} & 11 & 12 \\ \widehat{23} & 21 & 22 \end{bmatrix}$$

$$\begin{bmatrix} \widehat{11} & \widehat{12} & \widehat{13} \\ \widehat{01} & 02 & 03 \\ \widehat{21} & 22 & 23 \end{bmatrix} \begin{bmatrix} \widehat{12} & \widehat{11} & \widehat{13} \\ \widehat{02} & 01 & 03 \\ \widehat{22} & 21 & 23 \end{bmatrix} \begin{bmatrix} \widehat{13} & \widehat{11} & \widehat{12} \\ \widehat{03} & 01 & 02 \\ \widehat{23} & 21 & 22 \end{bmatrix}$$

$$\begin{bmatrix} \widehat{21} & \widehat{22} & \widehat{23} \\ \widehat{01} & 02 & 03 \\ \widehat{11} & 12 & 13 \end{bmatrix} \begin{bmatrix} \widehat{22} & \widehat{21} & \widehat{23} \\ \widehat{02} & 01 & 03 \\ \widehat{12} & 11 & 13 \end{bmatrix} \begin{bmatrix} \widehat{23} & \widehat{21} & \widehat{22} \\ \widehat{03} & 01 & 02 \\ \widehat{13} & 11 & 12 \end{bmatrix}$$

Removing the deleted rows and columns the preceding matrices are next below as the actual 2×2 matrices.

$$\begin{gathered} \begin{bmatrix} 12 & 13 \\ 22 & 23 \end{bmatrix} \begin{bmatrix} 11 & 13 \\ 21 & 23 \end{bmatrix} \begin{bmatrix} 11 & 12 \\ 21 & 22 \end{bmatrix} \\ \begin{bmatrix} 02 & 03 \\ 22 & 23 \end{bmatrix} \begin{bmatrix} 01 & 03 \\ 21 & 23 \end{bmatrix} \begin{bmatrix} 01 & 02 \\ 21 & 22 \end{bmatrix} \\ \begin{bmatrix} 02 & 03 \\ 12 & 13 \end{bmatrix} \begin{bmatrix} 01 & 03 \\ 11 & 13 \end{bmatrix} \begin{bmatrix} 01 & 02 \\ 11 & 12 \end{bmatrix} \end{gathered} \tag{9.4}$$

The 2×2 matrices whose determinants are needed in computing $\left((\sigma_0^2)^*, (\sigma_1^2)^*\right)$, that is, entry $(0, 1)$ of M_2 are given next.

$$\begin{bmatrix} 11 & 13 \\ 21 & 23 \end{bmatrix} \begin{bmatrix} 10 & 13 \\ 20 & 23 \end{bmatrix} \begin{bmatrix} 10 & 11 \\ 20 & 21 \end{bmatrix}$$
$$\begin{bmatrix} 01 & 03 \\ 21 & 23 \end{bmatrix} \begin{bmatrix} 00 & 03 \\ 20 & 23 \end{bmatrix} \begin{bmatrix} 00 & 01 \\ 20 & 21 \end{bmatrix} \tag{9.5}$$
$$\begin{bmatrix} 01 & 03 \\ 11 & 13 \end{bmatrix} \begin{bmatrix} 00 & 03 \\ 10 & 13 \end{bmatrix} \begin{bmatrix} 00 & 01 \\ 10 & 11 \end{bmatrix}$$

Recall that each number in these matrices is shorthand for an inner product of two barycentric differentials. For example, the entry 12 stands for $\langle \mathrm{d}\,\mu_1, \mathrm{d}\,\mu_2 \rangle = g((\mathrm{d}\,\mu_1)^\sharp, (\mathrm{d}\,\mu_2)^\sharp) = \nabla\mu_1 \cdot \nabla\mu_2$.

PROPOSITION 9.9. *For $p = n$, M_p is a diagonal matrix with $M_p(i, i) = 1/|\sigma_i^n|$, where $|\sigma_i^n|$ is the volume of the simplex.*

PROOF. For any n-simplex σ^n, the Whitney form $\mathrm{W}(\sigma^n)^*$ is 0 on other n-simplices and so M_p is diagonal. Furthermore, it is a constant coefficient volume form on σ^n with $\int_{\sigma^n} \mathrm{W}(\sigma^n)^* = 1$. See Dodziuk [1976] for proofs of these properties. Thus it must be that $\mathrm{W}(\sigma^n)^* = \mu/|\sigma^n|$ where μ is the volume form on the simplex. Thus

$$\left\langle \mathrm{W}(\sigma^n)^*, \mathrm{W}(\sigma^n)^* \right\rangle \mu = \mathrm{W}(\sigma^n)^* \wedge * \,\mathrm{W}(\sigma^n)^* = \frac{\mu}{|\sigma^n|^2}\,.$$

Thus $\left(\mathrm{W}(\sigma^n)^*, \mathrm{W}(\sigma^n)^*\right)_{L^2}$ is $\int_{\sigma^n} \mu/|\sigma^n|^2$ which is $1/|\sigma^n|$. □

9.3. Algorithm for Whitney Inner Product Matrix

We motivate our algorithm for Whitney mass matrix computation by making some observations about Example 9.8. The first and obvious observation is that matrix M_p is symmetric, being an inner product matrix. Thus only the diagonal entries and those above (or below) the diagonal need be computed. A more interesting efficiency comes from the structure of the entries of the matrix collections, such as ones shown in (9.4) and (9.5). Note that many entries repeat in the shorthand collection of matrices in (9.4) and (9.5). For example the entry 12 appears 4 times by itself in the matrix collection (9.4). Moreover, due to the symmetry of inner product, the entry 12 corresponds to the same result as the entry 21 and 21 appears 4 times as well. That entry also appears 4 times in the collection (9.5). Thus it is clear that a saving in computational time can be achieved by doing such calculations only once. That is, $\langle \mathrm{d}\,\mu_1, \mathrm{d}\,\mu_2 \rangle = \langle \mathrm{d}\,\mu_2, \mathrm{d}\,\mu_1 \rangle$ need only be computed once for the tetrahedron.

The determinants of all the matrices in a collection such as (9.4) are needed to plug into an expression like (9.3) to obtain a single entry (in this case row 0, column 3) of the Whitney inner product matrix for p-cochains ($p = 2$ in this case), whose size (4×4 in this case) depends on the number of p-simplices in the simplicial complex. Thus reusing repeated inner products of barycentric differentials can add up to a substantial saving in computational expense when all the unique entries of M_p are computed. These savings are quantified later in this subsection.

Another useful point to note in the example calculation is that the collection (9.5) of matrices can be obtained from the collection (9.4) by keeping the first digit in each entry same and making the substitutions $1 \to 0$; $2 \to 1$; and $3 \to 3$ in the second digit. The first digits in the two collections are the same because both correspond to the triangle σ_0^2. The preceding substitution works for the second digit because $\sigma_3^2 = [v_1, v_2, v_3]$ and $\sigma_1^2 = [v_0, v_1, v_3]$. This suggests the use of a template simplex for creating a template collection of matrices whose determinants are needed. The actual instances of the collections can then be obtained by using the vertex numbers in a given simplex. This is another idea that is used in the algorithm implemented in PyDEC. The algorithm takes as input a manifold simplicial n-complex K, embedded in $\mathbb{R}^N$ and $0 \le p \le n$. The output is M_p, an $N_p \times N_p$ matrix representation of inner product on $C^p(K;\mathbb{R})$ using elementary cochain basis. If a naive algorithm that does not take into account the duplications in determinant calculations were to be used, the number of operations required in the mass matrix calculation are

$$N_n \times \frac{\binom{n+1}{p+1}^2 + \binom{n+1}{p+1}}{2} \times \binom{n}{p}^2 \times Np^2 \times (O(p!) \text{ or } O(p^3)) .$$

The last term is written as $O(p!)$ or $O(p^3)$ because a determinant can be computed using the formula for determinant or by LU factorization. For low values of p (i.e., $\le$ about 5) the formula will likely be better.

According to the preceding formula, for example, for $n = 3, p = 2$, the number of determinants required in a naive implementation of mass matrix calculation would be

$$\frac{\binom{4}{3}^2 + \binom{4}{3}}{2} \times \binom{3}{2}^2 = 10 \times 9 = 90 .$$

But there are only 21 unique determinants needed for $n = 3, p = 2$. Our algorithm computes the unique determinants first and the operation count is

$$N_n \times \frac{\binom{n+1}{p}^2 + \binom{n+1}{p}}{2} \times Np^2 \times (O(p!) \text{ or } O(p^3)) .$$

Figure 8 shows a comparison of determinant counts for our algorithm compared with a naive algorithm that does the duplicate work that our PyDEC algorithm avoids. Note that for any n, the most advantage is gained intermediate values of p. The savings that the PyDEC implementation provides over a naive algorithm are several orders of magnitude, especially for moderately large n and higher. For $p = n$ case, in PyDEC we use the shortcut described in Proposition 9.9.

10. METRIC-DEPENDENT OPERATORS

We now describe the PyDEC implementations of some metric-dependent exterior calculus operators. The simplicial complex K is now supposed to be an approximation of a Riemannian n-manifold M. The metric implemented in PyDEC is the one induced from an embedding space $\mathbb{R}^N$. The main metric-dependent operator is the Hodge star that enables the discretization of codifferential and Laplace-deRham operators. The sharp and flat, which are isomorphisms between 1-forms and vector fields, are not implemented.

For the DEC Hodge star, the implementation is using the circumcentric dual as in Hirani [2003] and Desbrun et al. [2005] and the other operators are then simply defined in terms of the exterior derivative and the Hodge star. For PyDEC's

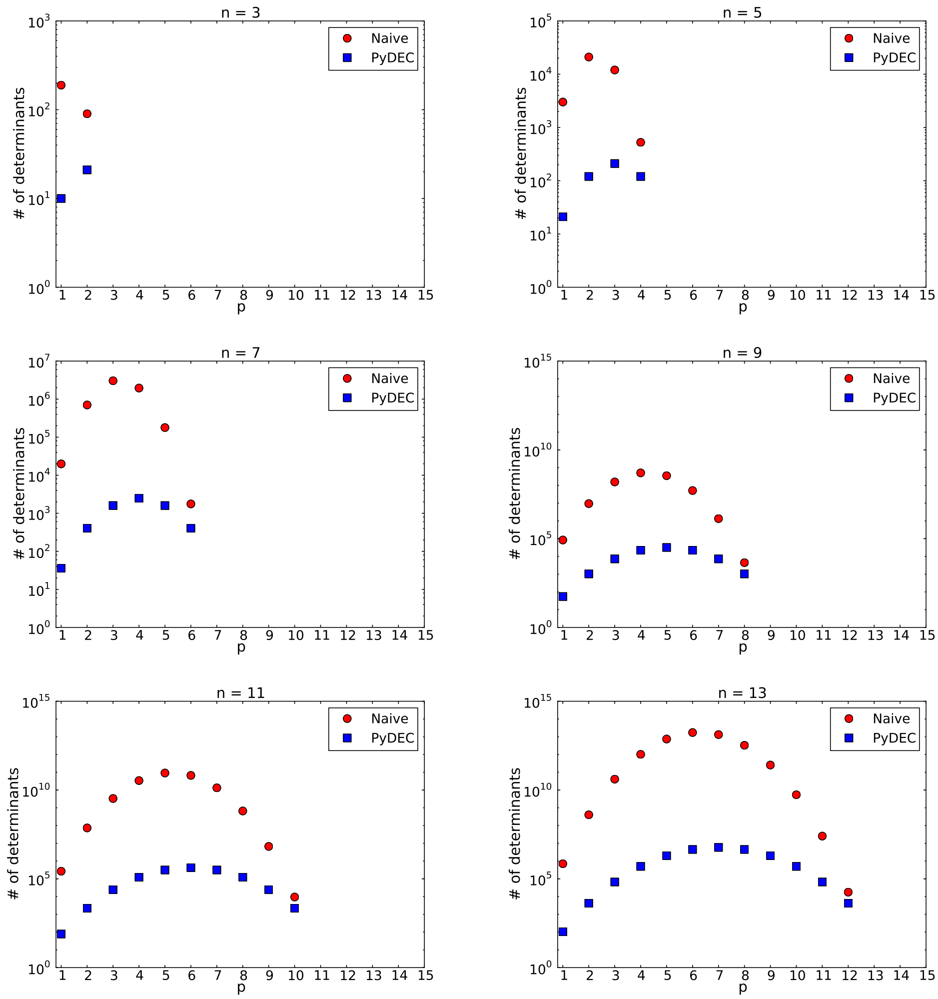


Fig. 8. Comparison between PyDEC and a naive algorithm for computing Whitney mass matrix. The figure shows the number of determinant computations needed by the two algorithms, for various values of p, the Whitney form dimension, and n, the simplicial complex dimension. The embedding dimension is not relevant in these calculations. For $p = n$ case we use the shortcut described in Proposition 9.9 so that case is not shown.

implementation of low-order finite element exterior calculus, we define the Hodge star to be the Whitney mass matrix described in Section 9. The other operators are defined by analogy with DEC even though the dual mesh concept is not part of finite element exterior calculus. Extensive experimental justification for this approach can be seen in its effectiveness in numerical experiments in Bell [2008] and in Hirani et al. [2011a].

For the definitions in this section we will need two cochain complexes of real-valued cochains. One will be on the simplicial complex K and for brevity we'll call this space of p-cochains $C^p(K)$ instead of $C^p(K;\mathbb{R})$. The other cochain complex is on the circumcentric dual cell complex $\star K$ and we'll denote the $(n-p)$-dimensional cochains as $D^{n-p}(\star K)$. At each dimension, these will be connected by discrete Hodge star operators to be defined shortly. Since the exterior derivative is the coboundary operator, the matrix representation for the exterior derivative on the dual mesh is the boundary operator. The matrix form for the DEC Hodge star on p-cochains will be denoted $*_p : C^p(K) \to D^{n-p}(\star K)$. One box of the primal and dual complexes is shown next.

$$\begin{array}{ccc} C^p(K) & \xrightarrow{\mathrm{d}_p} & C^{p+1}(K) \\ \downarrow{\scriptstyle *_p} & & \downarrow{\scriptstyle *_{p+1}} \\ D^{n-p}(\star K) & \xleftarrow{\mathrm{d}_p^T} & D^{n-p-1}(\star K) \end{array}$$

As described in Hirani [2003] and Desbrun et al. [2005] and other references, the DEC Hodge star is defined by

$$\frac{\langle *_p\, \sigma_i^*, \star\, \sigma_j \rangle}{|\star\, \sigma_j|} = \frac{\langle \sigma_i^*, \sigma_j \rangle}{|\sigma_j|},$$

for p-simplices σ_i and σ_j. Here σ^* is the elementary cochain corresponding to σ and $\langle \sigma^*, \tau \rangle$ stands for the evaluation of the cochain σ^* on the elementary chain τ. Thus the matrix representation of the DEC Hodge star $*_p$ is as a diagonal matrix with $*_p(i, i) = |\star\, \sigma_i| / |\sigma_i|$. In Hirani [2003] and Desbrun et al. [2005] this was defined for well-centered meshes. Recent work has shown that this definition extends to Delaunay meshes with an additional condition for boundary simplices. See Hirani et al. [2012] for details. Here we describe the codimension 1 case. For the codimension 1 Hodge star one computes the length of $\star\,\sigma$ taking into account signs. Consider a codimension 1 simplex σ shared by simplices L and R. For the portion of $\star\,\sigma$ corresponding to L, the sign is positive if the circumcenter and remaining vertex of L are on the same side of σ. Similarly for R. If σ is a codimension 1 face of top dimensional τ and is on domain boundary then the circumcenter of τ and vertex opposite to σ should be on the same side.

The smooth Hodge star on p-forms satisfies $** = (-1)^{p(n-p)}$. In the discrete setting $**$ is written as $*_p^{-1} *_p$ or $*_p *_p^{-1}$ and this is defined to be $(-1)^{p(n-p)} I$ where I is the identity matrix.

In the smooth theory, the codifferential $\delta_{p+1} : \Omega^{p+1}(M) \to \Omega^p(M)$ is defined as $\delta_{p+1} = (-1)^{np+1} * \mathrm{d} *$ and so we define the discrete codifferential $\delta_{p+1} : C^p(K) \to C^{n-p}(K)$ as $\delta_{p+1} := (-1)^{np+1} *_p^{-1} \mathrm{d}_p^T *_{p+1}$. For finite element exterior calculus implemented in PyDEC, we take this to be the definition, without reference to a dual mesh. If we now take $*_p$ to be the Whitney mass matrix then d_p and δ_{p+1} are adjoints (up to sign) with respect to the Whitney inner product on cochains as shown in Hirani et al. [2011a]. We will call the use of Whitney mass matrix as $*_p$ to be a *Whitney Hodge star* matrix.

In the discrete setting the Laplace-deRham operator is implemented in the weak form. For $0 < p < n$ the discrete definition is $\Delta_p := \mathrm{d}_p *_{p+1} \mathrm{d}_p + (-1)^{(p-1)(n-p+1)} *_p \mathrm{d}_{p-1} *_{p-1}^{-1} \mathrm{d}_{p-1}^T *_p$, with the appropriate term dropped for the $p = 0$ and $p = n$ cases. The preceding expression involves inverses of the Hodge star, which is easy to compute for a DEC Hodge star since that is a diagonal matrix. For a Whitney Hodge star, see Bell [2008] and Hirani et al. [2011a] for various approaches to avoiding explicitly forming the inverse Whitney mass matrix in computations.

10.1. Circumcenter Calculation

Circumcentric duality is used in DEC. To compute the DEC Hodge star, a basic computational step is the computation of the circumcenter of a simplex. We give here a linear system for computing the circumcenter using barycentric coordinates.

The circumcenter of a simplex is the unique point that is equidistant from all vertices of that simplex. In the case that a simplex (or face) is not of the same dimension as the embedding (e.g., a triangle embedded in $\mathbb{R}^4$), we choose the point that lies in the affine space spanned by the vertices of the simplex. In either case we can write the circumcenter in terms of barycentric coordinates of the simplex.

Let σ^p be the p-simplex defined by the points $\{v_0, v_1, \ldots v_p\}$ in $\mathbb{R}^N$. Let R denote the circumradius and c the circumcenter of simplex, which can be written in barycentric coordinates as $c = \sum_j b_j v_j$, where b_j is the barycentric coordinate for the circumcenter corresponding to v_j. For each vertex i we have

$$\left\| v_i - \sum_{j=0}^{p} b_j v_j \right\|^2 - R^2 = 0\,,$$

which can be rewritten as

$$v_i \cdot v_i - 2v_i \cdot \left(\sum_{j=0}^{p} b_j v_j \right) + \left\| \sum_{j=0}^{p} b_j v_j \right\|^2 - R^2 = 0\,.$$

Here the norm and the dot product are the standard ones on $\mathbb{R}^N$. Rearranging the preceding yields

$$2v_i \cdot \left(\sum_{j=0}^{p} b_j v_j \right) - \left(\left\| \sum_{j=0}^{p} b_j v_j \right\|^2 - R^2 \right) = v_i \cdot v_i\,.$$

The second term on the left-hand side is some scalar that is unknown, but is the same for every equation. So we can replace it by the unknown Q and write

$$2v_i \cdot \left(\sum_{j=0}^{p} b_j v_j \right) + Q = v_i \cdot v_i\,.$$

With the additional constraint that barycentric coordinates sum to one, we have a linear system with $p + 2$ unknowns ($b_0 \ldots b_p$ and Q) and $p + 2$ equations with the following matrix form.

$$\begin{pmatrix} 2v_0 \cdot v_0 & 2v_0 \cdot v_1 & \ldots & 2v_0 \cdot v_p & 1 \\ 2v_1 \cdot v_0 & 2v_1 \cdot v_1 & \ldots & 2v_1 \cdot v_p & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 2v_p \cdot v_0 & 2v_p \cdot v_1 & \ldots & 2v_p \cdot v_p & 1 \\ 1 & 1 & \ldots & 1 & 0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_p \\ Q \end{pmatrix} = \begin{pmatrix} v_0 \cdot v_0 \\ v_1 \cdot v_1 \\ \vdots \\ v_p \cdot v_p \\ 1 \end{pmatrix}$$

The solution to this yields the barycentric coordinates from which the circumcenter c can be located. Another quantity required for DEC Hodge star is the unsigned volume of a simplex. This can be computed by the well-known formula $\sqrt{\det V^T V}/p!$, where V is the p by N matrix with rows formed by the vectors $\{v_1 - v_0, v_2 - v_0, \ldots v_p - v_0\}$.

11. EXAMPLES

In the domains for which PyDEC is intended, it is often possible to easily translate the mathematical formulation of a problem into a working program. To make this point, and to demonstrate a variety of applications of PyDEC, we give 5 examples from different fields. The first example (Section 11.1) is a resonant cavity eigenvalue problem in which Whitney forms work nicely while the nodal piecewise linear Lagrange vector finite element $\mathcal{P}_1^2$ fails when directly applied. The second is Darcy flow (Section 11.2), which is an idealization of the steady flow of a fluid in a porous medium. We solve it here using DEC. The third problem (Section 11.3) is computation of a basis for the cohomology group of a mesh with several holes. This is achieved in our code here by Hodge decomposition of cochains, again using DEC. The next example (Section 11.4) is an idealization of the sensor network coverage problem. Some randomly located idealized sensors in the plane are connected into a Rips complex based on their mutual distances. Then a harmonic cochain computation reveals the possibility of holes in coverage. The last example (Section 11.5) involves the ranking of alternatives by a least-squares computation on a graph.

None of these problems is original and they all have been treated in the literature by a variety of techniques. We emphasize that we are including these just to demonstrate the capabilities of PyDEC. We have included the relevant parts of the Python code in this article. The full working programs are available with the PyDEC package [Bell and Hirani 2008].

11.1. Resonant Cavity Curl-Curl Problem

An electromagnetic resonant cavity is an idealized box made of a perfect conductor and containing no enclosed charges in which Maxwell's equations reduce to an eigenvalue problem. Several authors have popularized this example as one of many striking examples that motivate finite element exterior calculus. See, for instance, Arnold et al. [2010]. The use of $\mathcal{P}_1^2$ finite element space, that is, piecewise linear, Lagrange finite elements with 2 components, yields a corrupted spectrum. On the other hand, the use of $\mathcal{P}_1^-$ elements, that is, Whitney 1-forms yields the qualitatively correct spectrum. For detailed analysis and background see Boffi et al. [1999] and Arnold et al. [2010].

Let $M \subset \mathbb{R}^2$ be a square domain with side length π. We first give the equation in vector calculus notation and then in the corresponding exterior calculus notation. In the former, the resonant cavity problem is to find vector fields E and eigenvalues $\lambda \in \mathbb{R}$ such that

$$\mathbf{curl}\,\mathrm{curl}\, E = \lambda E \quad \text{on } M \quad \text{and} \quad E_\parallel = 0 \text{ on } \partial M\,,$$

where $E_\parallel$ is the tangential component of E on the boundary. Here $\mathbf{curl}\,\phi = (\partial\phi/\partial y, -\partial\phi/\partial x)$ and $\mathrm{curl}\, v = \partial v_2/\partial x - \partial v_1/\partial y$ for scalar function ϕ and vector field $v = (v_1, v_2)$. Note that for $\lambda \neq 0$ this equation is equivalent to the pair of equations $\Delta E = \lambda E$ and $\mathrm{div}\, E = 0$. This is because the vector Laplacian $\Delta = \mathbf{curl} \circ \mathrm{curl} - \mathrm{grad} \circ \mathrm{div}$ and $\mathrm{div} \circ \mathrm{curl} = 0$.

Now we give the equation in exterior calculus notation so the transition to PyDEC will be easier. Let $u \in \Omega^1(M)$ be the unknown electric field 1-form and $i : \partial M \hookrightarrow M$ the inclusion map. Then the preceding vector calculus equation is equivalent to

$$\begin{aligned} \delta_2\, \mathrm{d}_1\, u &= \lambda\, u \quad && \text{in } M \\ i^* u &= 0 && \text{on } \partial M\,. \end{aligned}$$

The pullback i^*u by inclusion map means restriction of u to the boundary, that is, allowing only vectors tangential to the boundary as arguments to u. As usual, we will seek u not in $\Omega^1(M)$ but in $H\Omega^1(M)$ subject to boundary conditions. Define the vector space $V = \{v \mid v \in H\Omega^1(M), i^*v = 0 \text{ on } \partial M\}$.

To express the PDE in weak form, we seek a (u, λ) in $V \times \mathbb{R}$ such that $(\delta_2\, \mathrm{d}_1\, u, v)_{L^2} = \lambda(u, v)_{L^2}$ for all $v \in V$. By the properties of the codifferential, the expression on the left is equal to $(\mathrm{d}_1\, u, \mathrm{d}_1\, v)_{L^2} - \int_{\partial M} u \wedge * \mathrm{d}_1\, v$. But the boundary term is 0 because u is in V.

Thus the weak form is to find a $(u, \lambda) \in V \times \mathbb{R}$ such that $(\mathrm{d}_1 u, \mathrm{d}_1 v)_{L^2} = \lambda(u, v)_{L^2}$ for all $v \in V$.

Taking the Galerkin approach of looking for a solution in a finite dimensional subspace of V here we pick the space of Whitney 1-forms, that is, $\mathcal{P}_1^-\Omega^1$ as the finite dimensional subspace. We define these over a triangulation of M which we will call K. The Whitney map $\mathrm{W} : C^1(K;\mathbb{R}) \to L^2\Omega^1(|K|)$ is an injection with its image $\mathcal{P}_1^-(K)$. Thus an equivalent formulation is over cochains. Using the same names for the variables, we seek a $(u, \lambda) \in C^1(K;\mathbb{R}) \times \mathbb{R}$ such that $(\mathrm{d}_1 \mathrm{W}\, u, \mathrm{d}_1 \mathrm{W}\, v)_{L^2} = \lambda(\mathrm{W}\, u, \mathrm{W}\, v)_{L^2}$ for all 1-cochains $v \in C^1(K;\mathbb{R})$. Since the Whitney map commutes with the exterior derivative and coboundary operator, and using the definition of cochain inner product, the preceding is same as $(\mathrm{d}_1 u, \mathrm{d}_1 v) = \lambda(u, v)$ where now the inner product is over cochains and d_1 is the coboundary operator. In matrix notation, using $*_1$ and $*_2$ to stand for the Whitney mass matrices M_1 and M_2, the generalized eigenvalue problem is to find $(u, \lambda) \in C^1(K;\mathbb{R}) \times \mathbb{R}$ such that

$$\mathrm{d}_1^T *_2 \mathrm{d}_1 u = \lambda *_1 u\,.$$

We now translate this equation into PyDEC code. Once the appropriate modules have been imported, a simplicial complex object `sc` is created after reading in the mesh files. Now the main task is to find matrix representations for the stiffness matrix. $\mathrm{d}_1^T *_2 \mathrm{d}_1$ and the mass matrix $*_1$. This is accomplished by the following two lines, where `K` is the stiffness matrix.

```
K = sc[1].d.T * whitney_innerproduct (sc,2) * sc[1].d
M = whitney_innerproduct (sc,1)
```

The boundary conditions can be imposed by simply removing the edges that lie on the boundary. The indices of such edges are easily determined and stored in the list `non_boundary_indices` which is used next to impose the boundary conditions.

```
K = K[non_boundary_indices,:][:,non_boundary_indices]
M = M[non_boundary_indices,:][:,non_boundary_indices]
```

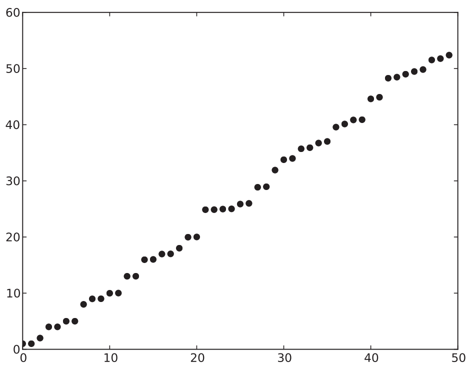
Now all that remains is to solve the eigenvalue problem. To simplify the code and because the matrix size is small, we use the dense eigenvalue solver `scipy.linalg.eig`.

```
eigenvalues, eigenvectors = eig(K.todense(), M.todense())
```

Some of the resulting eigenvalues are displayed in the left part of Figure 9. The 1-cochain u which is the eigenvector corresponding to one of these eigenvalues is shown as a vector field in the right part of Figure 9. The visualization as a vector field is achieved by interpolating the 1-cochain u using the Whitney map and then sampling the vector field $(\mathrm{W}\, u)^\sharp$ at the barycenter. This is achieved by the PyDEC command

```
bases, arrows = simplex_quivers (sc,all_values)
```

where `all_values` contains both the known and the computed values of the 1-cochain. There is no sharp operator in PyDEC. But since PyDEC only implements the Riemannian metric from the embedding space of simplices the transformation from 1-form to vector field just involves using the components of the Whitney 1-form as the vector field components.



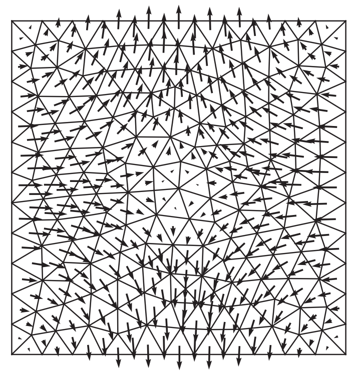


Fig. 9. The first 50 nonzero eigenvalues for the resonant cavity problem of Section 11.1 and the eigenvector corresponding to one of these eigenvalues. The eigenvector is a 1-cochain which is visualized as a vector field by first interpolating it using a Whitney map. See Section 11.1 for details.

11.2. Darcy Flow or Poisson's in Mixed Form

We give here a brief description of the equations of Darcy flow and their PyDEC implementation. For more details see Hirani et al. [2011c]. The resonant cavity example in Section 11.1 was implemented using finite element exterior calculus. For variety we use a DEC implementation for Darcy flow.

Darcy flow is a simple model of steady state flow of an incompressible fluid in a porous medium. It models the statement that flow is from high to low pressure. For a fixed pressure gradient, the velocity is proportional to the permeability κ of the medium and inversely proportional to the viscosity μ of the fluid. Let the domain be M, a polygonal planar domain. Assuming that there are no sources of fluid in M and there is no other force acting on the fluid, the equations of Darcy flow are

$$v + \frac{\kappa}{\mu}\nabla p = 0 \quad \text{and} \quad \operatorname{div} v = 0 \quad \text{in } M \quad \text{with} \quad v \cdot \hat{n} = \psi \quad \text{on } \partial M, \tag{11.1}$$

where $\kappa > 0$ is the coefficient of permeability of the medium $\mu > 0$ is the coefficient of (dynamic) viscosity of the fluid, $\psi : \partial M \to \mathbb{R}$ is the prescribed normal component of the velocity across the boundary, and $\hat{n}$ is the unit outward normal vector to ∂M. For consistency $\int_{\partial M} \psi \, d\Gamma = 0$, where $d\Gamma$ is the measure on ∂M. Since $\operatorname{div} \circ \operatorname{grad} = \Delta$, the preceding simplified Darcy flow equations are equivalent to Laplace's equation.

Let K be a simplicial complex that triangulates M. Instead of velocity and pressure, we will use flux and pressure as the primary unknowns. The flux through the edges is $f = * v^\flat$ and thus it will be a primal 1-cochain. Although PyDEC does not implement a flat operator, this is not an issue here because we never solve for v, and make f itself one of the unknowns. This implies that the pressure p will be a dual 0-cochain since $*\,\mathrm{d}\,p$ has to be of the same type as f. The choice to put flux on primal edges and pressures on circumcenters can be reversed, as shown in a dual formulation in Gillette and Bajaj [2011]. In exterior calculus notation, the PDE in (11.1) is $-(\mu/k) * f + \mathrm{d}\, p = 0$ and $\mathrm{d}\, f = 0$, which, when discretized, translates to the matrix equation

$$\begin{bmatrix} -(\mu/k) *_1 & \mathrm{d}_1^T \\ \mathrm{d}_1 & 0 \end{bmatrix} \begin{bmatrix} f \\ p \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

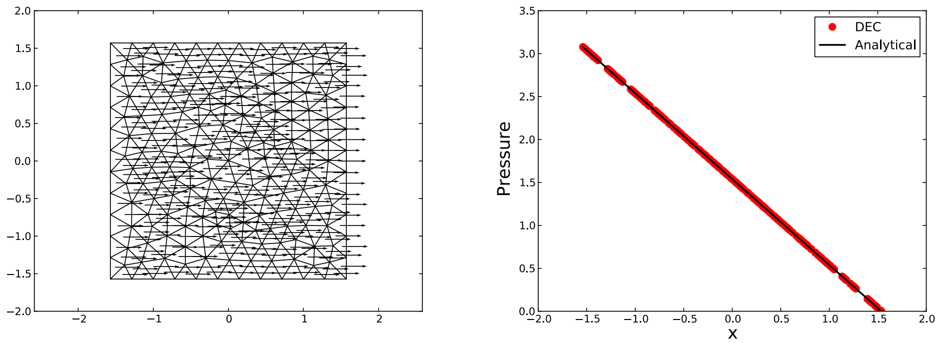


Fig. 10. Darcy flow using discrete exterior calculus. The boundary condition is that fluid is coming in from left and leaving from right with velocity 1. The velocity inside should be constant and pressure should be linear. The flux and pressure are computed in a mixed formulation. The flux is taken to be a primal 1-cochain associated with primal edges, and the pressure is a dual 0-cochain on dual vertices, which are circumcenters of the triangles. The velocity is obtained by Whitney interpolation of the flux, which is sampled at the barycenters. See Section 11.2 for more details.

In PyDEC, the construction of this matrix is straightforward. Once a simplicial complex sc has been constructed, the following 3 lines construct the matrix in the previous system.

```
d1 = sc[1].d; star1 = sc[1].star
A = bmat([[(-mu/k)*sc[1].star, sc[1].d.T],
          [sc[1].d, None]], format='csr')
```

After computing the boundary condition in terms of flux through the boundary edges, the linear system is adjusted for the known values and then solved for the fluxes and pressures. Figure 10 shows the solution for the case of constant horizontal velocity and linear pressure gradient.

11.3. Cohomology Basis Using Hodge Decomposition

The Hodge Decomposition Theorem [Abraham et al. 1988, page 539] states that for a compact boundaryless smooth manifold M, for any p-form $\omega \in \Omega^p(M)$, there exists an $\alpha \in \Omega^{p-1}(M)$, $\beta \in \Omega^{p+1}(M)$, and a harmonic form $h \in \Omega^p(M)$ such that $\omega = d\alpha + \delta\beta + h$. Here harmonic means that $\Delta h = 0$, where Δ is the Laplace-deRham operator $\mathrm{d}\,\delta + \delta\,\mathrm{d}$. Moreover $\mathrm{d}\,\alpha$, $\delta\beta$ and h are mutually L^2-orthogonal, which makes them uniquely determined. In case of a manifold with boundary, the decomposition is similar, with some additional boundary conditions. See Abraham et al. [1988] for details.

The Hodge-deRham theorem [Abraham et al. 1988] relates the analytical concept of harmonic forms with the topological concept of cohomology. For any topological space, the cohomology groups or vector spaces of various dimension capture essential topological information about the space [Munkres 1984]. For the preceding manifold M, the p-dimensional cohomology group with real coefficients, which is a finite dimensional space, is denoted $H^p(M;\mathbb{R})$ or just $H^p(M)$. For example, for a torus, H^1 has dimension 2. For a square with 4 holes used in this example, which does have boundaries, H^1 has dimension 4. The elements of $H^p(M)$ are equivalence classes of closed forms (those whose d is 0). Two closed forms are equivalent if their difference is exact (that is, is d of some form). While the representatives of 1-homology spaces can be visualized as loops around holes, handles, and tunnels, those of 1-cohomology should be visualized as fields. If the space of harmonic forms is denoted $\mathcal{H}^p(M)$, then the Hodge-deRham theorem says that it is isomorphic, as a vector space, to the p-th cohomology space $H^p(M)$ in the case of a closed manifold. See Jost [2005] for details. Again, the case of M with boundary requires some adjustments in the definitions, as given in Abraham et al. [1988].

For finite dimensional spaces, Hodge decomposition follows from very elementary linear algebra. If U, V and W are finite dimensional inner product vector spaces and $A : U \to V$ and $B : V \to W$ are linear maps such that $B \circ A = 0$ then middle vector space V splits into 3 orthogonal components, which are $\operatorname{im} A$, $\operatorname{im} B^T$, and $\ker A^T \cap \ker B$. In this example, we find a basis for H^1 for a square. This is done by finding a basis of harmonic 1-cochains. Thus given a 1-cochain ω, its discrete Hodge decomposition exists and is $\omega = \mathrm{d}_0\,\alpha + \delta_2\,\beta + h$. In this example, the cochains α and β are obtained by solving the linear systems $\delta_1\,\mathrm{d}_0\,\alpha = \delta_1\,\omega$ and $\mathrm{d}_1\,\delta_2\,\beta = \mathrm{d}_1\,\omega$. The harmonic component can then be computed by subtraction.

In the example code, the main function is the one that computes the Hodge decomposition of a given cochain `omega`. First empty cochains for `alpha` and `beta` are created.

```
sc = omega.complex
p = omega.k
alpha = sc.get_cochain(p - 1)
beta = sc.get_cochain(p + 1)
```

Now the solution for `alpha` and `beta` closely follows the preceding equations for α and β.

```
A = delta(d(sc.get_cochain_basis(p - 1))).v
b = delta(omega).v
alpha.v = cg( A, b, tol=1e-8 )[0]

A = d(delta(sc.get_cochain_basis(p + 1))).v
b = d(omega).v
beta.v = cg( A, b, tol=1e-8 )[0]
```

Even though the matrices `A` given before are singular, the solutions exist, and since conjugate gradient is used, the presence of the nontrivial kernels does not pose any problems [Bochev and Lehoucq 2005].

The harmonic 1-forms shown in Figure 11 are obtained by decomposing random 1-forms and retaining their harmonic components. Since the initial basis has no particular spatial structure, an ad hoc orthogonalization procedure is then applied. For each basis vector, the algorithm identifies the component with the maximum magnitude and applies Householder transforms to force the other vectors to zero at that same component.

11.4. Sensor Network Coverage

As discussed in Section 5, sensor network coverage gaps can be identified with coordinate-free methods based on topological properties of the Rips complex. This is for an idealized abstraction of a sensor network. The following example constructs a `rips_complex` object from a set of 300 points randomly distributed over the unit square, as illustrated in top left in Figure 12. Recall that the Rips complex is constructed

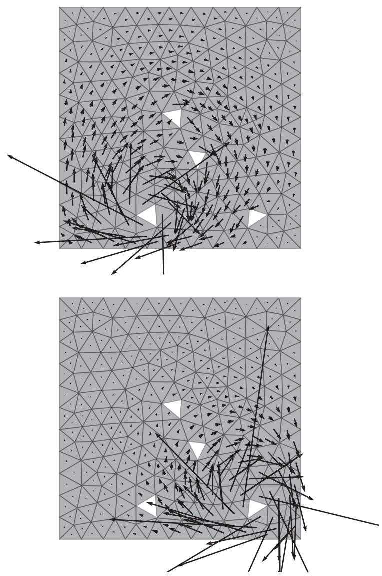


Fig. 11. Four harmonic cochains form a basis for the first cohomology space H^1 for a mesh with four holes. Each cochain is visualized as a vector field by interpolating it from the edge values using Whitney interpolation. See Section 11.3 for details.

by adding an edge between each pair of points within a given radius. Top right of Figure 12 illustrates the edges of the Rips complex produced by a cut-off radius of 0.15. The triangles of the Rips complex, illustrated in bottom left of Figure 12, represent triplets of vertices that form a clique in the edge graph of the Rips complex. The Rips complex is created by the following two lines of code.

```
pts = read_array('300 pts.mtx')  # 300 random points in 2D
rc = rips_complex( pts , 0.15 )
```

The sensor network is tested for coverage holes by inspecting the kernel of the matrix $\Delta_1 = \partial_1^T \partial_1 + \partial_2 \partial_2^T$ [de Silva and Ghrist 2007]. Specifically, null-vectors of Δ_1, which are called harmonic 1-cochains (by analogy with the definition of harmonic cochains used in the previous subsection), reveal the presence of holes in the sensor network. In this example we explore the kernel of Δ_1 by generating a random 1-cochain `x` and extracting its harmonic part using a discrete Hodge decomposition as outlined in the previous subsection. If the harmonic component of `x` is (numerically) zero then we may conclude with high confidence that Δ_1 is nonsingular and that no holes are present. However, in this case the Hodge decomposition of `x` produces a nonzero harmonic component `h`. Indeed, plotting `h` on the edges of the Rips complex localizes the coverage hole, as the bottom right of Figure 12 demonstrates.

To set up the linear systems, the boundary matrices are obtained from the Rips complex `rc` created earlier.

```
cmplx = rc.chain_complex () # boundary operators [ b0 , b1 , b2 ]
b1 = cmplx[1].astype( float ) # edge boundary operator
b2 = cmplx[2].astype( float ) # face boundary operator
```

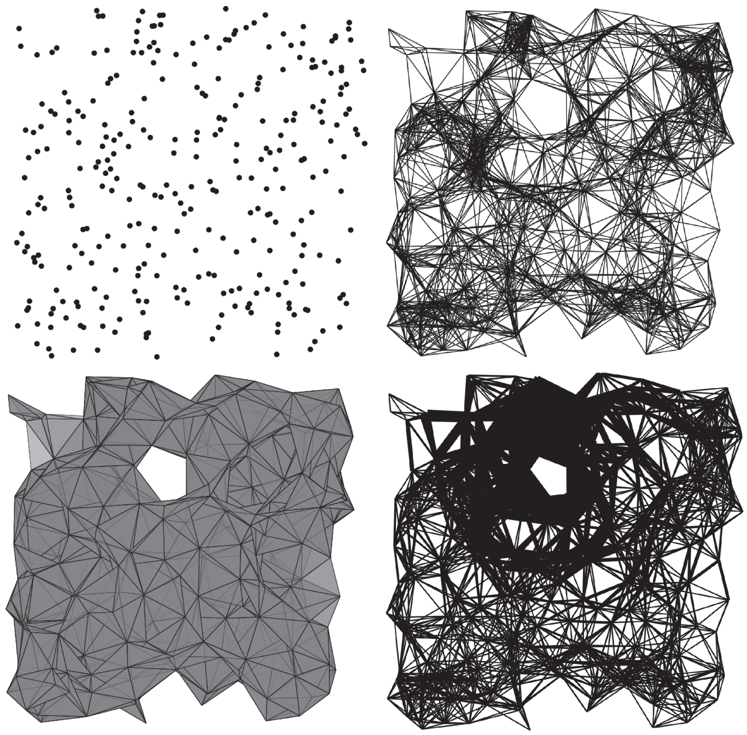


Fig. 12. Hodge decomposition for finding coverage holes in an idealized sensor network. Top left shows a sample sensor network with 300 randomly distributed points. Pairs of points within a fixed distance of one another are connected by an edge and this is shown on top right. Triangles are added to the Rips complex when three points form a clique (complete graph). These are shown in bottom left. The existence of a harmonic 1-cochain indicates a potential hole in the sensor network coverage. In the bottom right figure, edge thickness reflects the magnitude of the harmonic cochain on each edge. See Section 11.4 for more details.

Then the random cochain is created and the Hodge decomposition computed, to find the harmonic cochain which is then normalized.

```
x = rand(b1.shape[1]) # random 1-chain
# Decompose x using discrete Hodge decomposition
alpha = cg( b1 * b1.T, b1 * x, tol=1e-8)[0]
beta = cg( b2.T * b2, b2.T * x, tol=1e-8)[0]
h = x - (b1.T * alpha) - (b2 * beta) # harmonic component of x
h /= abs(h).max() # normalize h
```

11.5. Least-Squares Ranking on Graphs

This is a formulation for ranking alternatives based on pairwise data. Given is a collection of alternatives or objects that have to be ranked, by computing a ranking score that sorts them. The ranking scores are to be computed starting from some pairwise comparisons. Some examples of objects to be ranked are basketball teams, movies, and candidates for a job. Typically, the given data will not have pairwise comparisons for all the possible pairs. There is no geometry in this application, hence no exterior calculus is involved. PyDEC, however, still proves useful because the full version of this example [Hirani et al. 2011b] uses an abstract simplicial 2-complex. Thus PyDEC is useful in forming the complex and for determining its boundary matrix. In this simplified example, only an abstract simplicial 1-complex is needed.

Form a simple graph G, with the objects to be ranked being the vertices and with an edge between any two which have pairwise comparison data given. If there are n objects, possibly only a sparse subset out of all possible $O(n^2)$ pairs may have comparison data associated with them. Here we'll only require that the graph be connected. This condition can be dropped with the consequence that the rankings of separate components become independent of each other. The comparison values are real numbers.

Since G is a simple graph, by orienting the edges arbitrarily it becomes an oriented 1-dimensional abstract simplicial complex. The vector ω of pairwise comparison values is a 1-cochain since if A is preferred over B by, say, 4 points, then B is preferred over A by -4 points.

The ranking scores α which are to be computed on vertices form a 0-cochain. For any edge $e = (u, v)$ from vertex u to vertex v, the difference of vertex values $\alpha(v) - \alpha(u)$ should match $\omega(e)$ as much as possible, for example, in a least-squares sense. This idea is from Leake [1976] who proposed it as a method for ranking football teams. By including the 3-cliques as triangles, G becomes a 2-dimensional simplicial complex. This was used in Jiang [2011] to extend this ranking idea. In Jiang [2011] the computation of the scores α is interpreted as one part of the Hodge decomposition of ω. See Section 11.3 for a basic discussion of Hodge decomposition where it is used for computing harmonic cochains on a mesh. Here we will just compute the ranking score α. This is done by solving the least-squares problem $\partial_1^T\ \alpha \simeq \omega$.

The graph in this example is used for ranking basketball teams, using real data for a small subset of American Men's college basketball games from 2010–2011 season. Each team is a node in the graph and has been given a number as a name. An edge between two teams indicates that one of more games have been played between them. The score difference from these games becomes the input 1-cochain ω, with one value on each edge. If multiple games were played by a pair the score differences were added to create this data. The data is stored as a matrix in which the first two columns are the teams and the third column is the value of the 1-cochain on that edge.

Once this data is loaded from file, an abstract simplicial complex is created from the first two columns which form the edges of the graph. The loading and complex creation is done by the following few lines of code.

```
data = loadtxt('data.txt').astype(int)
edges = data[:,:2]

# Create abstract simplicial complex from edges
asc = abstract_simplicial_complex([edges])
```

In PyDEC, the simplices that are given as input to construct a complex are preserved as is. Lower-dimensional simplices that are derived from them are stored and oriented in sorted order. Thus in the preceding data, the edge between node 8 and 1 will be oriented from 8 to 1. The previous example data may mean, for example, that team labeled 1 lost to team labeled 8 by 9 points.

The 1-cochain ω is now extracted from the data array and the boundary matrix needed is obtained from the complex and the least-squares problem solved. All this is accomplished in the following lines.

```
omega = data[:,-1] # pairwise comparisons
B1 = asc.chain_complex()[1] # boundary matrix
alpha = lsqr(B1.T, omega)[0] # solve least squares problem
```

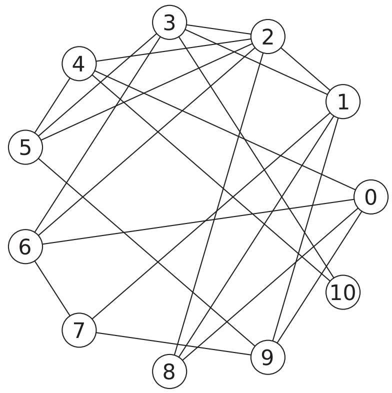


Fig. 13. A typical graph of a subset of basketball games. Each node is a team, labelled by a number. An edge represents one or more games played by the two teams connected by it. The actual graph used in the example in Section 11.5 is a complete graph.

The resulting alpha values computed are given next.

Team number	0	1	2	3	4	5	6	7	8	9	10
α value	14.5	2.0	0.0	7.2	5.9	9.0	2.3	23.6	11.0	23.8	21.7

The team with $\alpha = 0$ is the worst team according to these rankings and the one with the largest α value (23.8 here) is the best team. Note that the score difference from the game between 8 and 1 happens to be exactly the difference in α values between them. This won't always be true. See, for example, teams 3 and 9. Such discrepancies come from having a residual in the least-squares solution, and that is a direct result of the presence of cycles in the graph. In fact it may even happen that team A beats B, which beats C, which in turn beats A. Thus no assignment of α values will resolve this inconsistency. This is where a second least-squares problem and hence a Hodge decomposition plays a role. The second problem is not considered in this example. See Hirani et al. [2011b] for details on the second least-squares problem and the role of Hodge decomposition and harmonic cochains in the ranking context.

12. CONCLUSIONS

PyDEC is intended to be a tool for solving elliptic PDEs formulated in terms of differential forms and for exploring computational topology problems. It has been used for numerical experiments for Darcy flow [Hirani et al. 2011c], computation of harmonic cochains on two- and three-dimensional meshes [Hirani et al. 2011a], and least-squares ranking on graphs [Hirani et al. 2011b]. It has also proved valuable in computational topology work [Dey et al. 2010; Dunfield and Hirani 2011], for creating complexes and computing boundary matrices. The design goals for PyDEC have been efficiency and ability to express mathematical formulations easily. Section 11 which described some examples should give an idea of how close the PyDEC code is to the mathematical formulation of the problems considered. Many packages exist and are being created for numerical PDE solutions using differential forms. There are also many excellent computational topology packages. PyDEC can handle a large variety of complexes and provides implementations of discrete exterior calculus and lowest-order finite element exterior calculus using Whitney forms. These qualities make it a convenient tool to explore the interrelationships between topology, geometry, and numerical PDEs.

APPENDIX

PROOF OF PROPOSITION 9.7. By definition $M_p(i,j) = \int_{|K|} \langle \mathrm{W}(\sigma_i^p)^*, \mathrm{W}(\sigma_j^p)^* \rangle \mu$. The integrand is nonzero only in $\overline{\mathrm{St}}(\sigma_i^p) \cap \overline{\mathrm{St}}(\sigma_i^p)$ since the Whitney form corresponding to a simplex is zero outside the star of that simplex [Dodziuk 1976]. Thus

$$M_p(i,j) = \sum_{\substack{\sigma_n \\ \sigma^n \succeq \sigma_i^p, \sigma_j^p}} \int_{\sigma_n} \langle \mathrm{W}(\sigma_i^p)^*|_{\sigma^n}, \mathrm{W}(\sigma_j^p)^*|_{\sigma^n} \rangle \mu \,.$$

By the definition of the Whitney map, $M_p(i,j)$ is

$$(p!)^2 \sum_{\substack{\sigma^n \\ \sigma^n \succeq \sigma_i^p, \sigma_j^p}} \sum_{k,l=0}^{p} (-1)^{k+l} \int_{\sigma^n} \langle \mathrm{d}\,\mu_{i_0} \wedge \ldots \widehat{\mathrm{d}\,\mu_{i_k}} \cdots \wedge \mathrm{d}\,\mu_{i_p}, \mathrm{d}\,\mu_{j_0} \wedge \ldots \widehat{\mathrm{d}\,\mu_{j_l}} \cdots \wedge \mathrm{d}\,\mu_{j_p} \rangle \mu_{i_k} \mu_{j_l}\, \mu \,.$$

But differentials of barycentric coordinates are constant in σ^n. Thus the term

$$\langle \mathrm{d}\,\mu_{i_0} \wedge \ldots \widehat{\mathrm{d}\,\mu_{i_k}} \cdots \wedge \mathrm{d}\,\mu_{i_p}, \mathrm{d}\,\mu_{j_0} \wedge \ldots \widehat{\mathrm{d}\,\mu_{j_l}} \cdots \wedge \mathrm{d}\,\mu_{j_p} \rangle$$

comes out of the integral. Recalling Definition 8.2 of inner product of forms,

$$\langle \mathrm{d}\,\mu_{i_0} \wedge \ldots \widehat{\mathrm{d}\,\mu_{i_k}} \cdots \wedge \mathrm{d}\,\mu_{i_p}, \mathrm{d}\,\mu_{j_0} \wedge \ldots \widehat{\mathrm{d}\,\mu_{j_l}} \cdots \wedge \mathrm{d}\,\mu_{j_p} \rangle$$

is given by

$$\det \begin{bmatrix} \langle \mathrm{d}\,\mu_{i_0}, \mathrm{d}\,\mu_{j_0} \rangle & \ldots & \langle \widehat{\mathrm{d}\,\mu_{i_0}, \mathrm{d}\,\mu_{j_l}} \rangle & \ldots & \langle \mathrm{d}\,\mu_{i_0}, \mathrm{d}\,\mu_{j_p} \rangle \\ \vdots & & \vdots & & \vdots \\ \langle \widehat{\mathrm{d}\,\mu_{i_k}, \mathrm{d}\,\mu_{j_0}} \rangle & \ldots & \langle \widehat{\mathrm{d}\,\mu_{i_k}, \mathrm{d}\,\mu_{j_l}} \rangle & \ldots & \langle \widehat{\mathrm{d}\,\mu_{i_k}, \mathrm{d}\,\mu_{j_l}} \rangle \\ \vdots & & \vdots & & \vdots \\ \langle \mathrm{d}\,\mu_{i_p}, \mathrm{d}\,\mu_{j_0} \rangle & \ldots & \langle \widehat{\mathrm{d}\,\mu_{i_p}, \mathrm{d}\,\mu_{j_l}} \rangle & \ldots & \langle \mathrm{d}\,\mu_{i_p}, \mathrm{d}\,\mu_{j_p} \rangle \end{bmatrix},$$

which completes the proof. □

REFERENCES

ABRAHAM, R., MARSDEN, J. E., AND RATIU, T. 1988. *Manifolds, Tensor Analysis, and Applications* 2nd Ed. Springer.

ARNOLD, D. N., FALK, R. S., AND WINTHER, R. 2006. Finite element exterior calculus, homological techniques, and applications. In *Acta Numerica*, A. Iserles Ed., vol. 15. Cambridge University Press, 1–155.

ARNOLD, D. N., FALK, R. S., AND WINTHER, R. 2009. Geometric decompositions and local bases for spaces of finite element differential forms. In *Comput. Meth. Appl. Mech. Engrg. 198*, 21–26, 1660–1672.

ARNOLD, D. N., FALK, R. S., AND WINTHER, R. 2010. Finite element exterior calculus: From Hodge theory to numerical stability. *Bull. Amer. Math. Soc. 47*, 2, 281–354.

BANGERTH, W., HARTMANN, R., AND KANSCHAT, G. 2007. deal.II – a general purpose object oriented finite element library. *ACM Trans. Math. Softw. 33*, 4, 24/1–24/27.

BANK, R. AND DOUGLAS, C. 1993. Sparse matrix multiplication package (SMMP). *Adv. in Comput. Math. 1*, 1, 127–137.

BELL, N. AND HIRANI, A. N. 2008. PyDEC: A python library for discrete exterior calculus. Google Code.

BELL, W. N. 2008. Algebraic multigrid for discrete differential forms. Ph.D. thesis, University of Illinois at Urbana-Champaign, Urbana, Illinois.

BOCHEV, P. AND LEHOUCQ, R. B. 2005. On the finite element solution of the pure Neumann problem. *SIAM Rev. 47*, 1, 50–66.

BOCHEV, P. B. AND HYMAN, J. M. 2006. Principles of mimetic discretizations of differential operators. In *Compatible Spatial Discretizations*, D. N. Arnold, P. B. Bochev, R. B. Lehoucq, R. A. Nicolaides, and M. Shashkov Eds., The IMA Volumes in Mathematics and Its Applications Series, vol. 142. Springer, 89–119.

BOCHEV, P. B. AND ROBINSON, A. C. 2002. Matching algorithms with physics: Exact sequences of finite element spaces. In *Collected Lectures on Preservation of Stability Under Discretization*, D. Estep and S. Tavener Eds., SIAM, 145–166.

BOFFI, D., FERNANDES, P., GASTALDI, L., AND PERUGIA, I. 1999. Computational models of electromagnetic resonators: Analysis of edge element approximation. *SIAM J. Numer. Anal. 36*, 4, 1264–1290.

BOSSAVIT, A. 1988. Whitney forms: A class of finite elements for three-dimensional computations in electromagnetism. *IEE Proc. 135, Part A*, 8, 493–500.

BOTT, R. AND TU, L. W. 1982. *Differential Forms in Algebraic Topology*. Springer.

CASTILLO, P., RIEBEN, R., AND WHITE, D. 2005. FEMSTER: An object oriented class library of high-order discrete differential forms. *ACM Trans. Math. Softw. 31*, 4, 425–457.

DE RHAM, G., 1955. *Variétés Différentiables. Formes, Courants, Formes Harmoniques*. Actualités Sci. Ind., no. 1222 = Publ. Inst. Math. Univ. Nancago III. Hermann et Cie, Paris, Paris.

DE SILVA, V. AND GHRIST, R. 2007. Homological sensor networks. *Not. Amer. Math. Soc. 54*, 1, 10–17.

DEMLOW, A. AND DZIUK, G. 2007. An adaptive finite element method for the Laplace–Beltrami operator on implicitly defined surfaces. *SIAM J. Numer. Anal. 45*, 1, 421–442.

DESBRUN, M., HIRANI, A. N., LEOK, M., AND MARSDEN, J. E. 2005. Discrete exterior calculus. E-print arXiv:math/0508341[math.DG].

DEY, T. K., HIRANI, A. N., AND KRISHNAMOORTHY, B. 2010. Optimal homologous cycles, total unimodularity, and linear programming. In *Proceedings of the 42nd ACM Symposium on Theory of Computing*. ACM, New York, 221–230.

DODZIUK, J. 1976. Finite-Difference approach to the Hodge theory of harmonic forms. *Amer. J. Math. 98*, 1, 79–104.

DUNFIELD, N. M. AND HIRANI, A. N. 2011. The least spanning area of a knot and the optimal bounding chain problem. In *Proceedings of the 27th Annual ACM Symposium on Computational Geometry (SoCG'11)*. ACM, New York, 135–144.

ECKMANN, B. 1945. Harmonische functionen and randwertaufgaben in ainem komplex. *Comment. Math. Helv. 17*, 240–255.

EDELSBRUNNER, H., LETSCHER, D., AND ZOMORODIAN, A. 2002. Topological persistence and simplification. *Discr. Comput. Geom. 28*, 4, 511–533.

FRANKEL, T. 2004. *The Geometry of Physics* 2nd Ed. Cambridge University Press.

GILLETTE, A. AND BAJAJ, C. 2010. A generalization for stable mixed finite elements. In *Proceedings of the 14th ACM Symposium on Solid and Physical Modeling*. ACM, New York, 41–50.

GILLETTE, A. AND BAJAJ, C. 2011. Dual formulations of mixed finite elements methods with applications. In *Comput.-Aided Des. 43* 10, 1213–1221.

GRADINARU, V. AND HIPTMAIR, R. 1999. Whitney elements on pyramids. *Electron. Trans. Numer. Anal. 8*, 154–168.

HILDEBRANDT, K., POLTHIER, K., AND WARDETZKY, M. 2006. On the convergence of metric and geometric properties of polyhedral surfaces. *Geom. Dedicata 123*, 89–112.

HIPTMAIR, R. 2002. Finite elements in computational electromagnetism. In *Acta Numerica*, A. Iserles Ed., vol. 11. Cambridge University Press, 237–339.

HIRANI, A. N. 2003. Discrete exterior calculus. Ph.D. thesis, California Institute of Technology, Pasadena, CA.

HIRANI, A. N., KALYANARAMAN, K., WANG, H., AND WATTS, S. 2011a. Cohomologous harmonic cochains. arXiv:1012.2835.

HIRANI, A. N., KALYANARAMAN, K., AND WATTS, S. 2011b. Least squares ranking on graphs. E-print arXiv:1011.1716[cs.NA].

HIRANI, A. N., NAKSHATRALA, K. B., AND CHAUDHRY, J. H. 2011c. Numerical method for darcy flow derived using discrete exterior calculus. arXiv:0810.3434[math.NA].

HIRANI, A. N., KALYANARAMAN, K., AND VANDERZEE, E. B. 2012. Delaunay Hodge star. *Comput.-Aided Des. 45*, 2, 540–544.

HOLST, M. AND STERN, A. 2012. Geometric variational crimes: Hilbert complexes, finite element exterior calculus, and problems on hypersurfaces. *Found. Comput. Math. 12*, 3, 263–293.

HYMAN, J. M. AND SHASHKOV, M. 1997. Natural discretizations for the divergence, gradient, and curl on logically rectangular grids. *Comput. Math. Appl. 33*, 4, 81–104.

JIANG, X., LIM, L., YAO, Y., AND YE, Y. 2011. Statistical ranking and combinatorial hodge theory. *Math. Program. 127*, 203–244.

JOST, J. 2005. *Riemannian Geometry and Geometric Analysis* 4th Ed. Universitext. Springer.

KACZYNSKI, T., MISCHAIKOW, K., AND MROZEK, M. 2004. *Computational Homology*. Applied Mathematical Sciences Series, vol. 157. Springer.

LEAKE, R. J. 1976. A method of ranking teams: With an application to college football. In *Management Science in Sports*, R. E. Machol and S. P. Ladany Eds., TIMS Studies in the Management Sciences Series, vol. 4. North-Holland, 27–46.

LOGG, A. AND MARDAL, K.-A. 2008. A symbolic engine for finite element exterior calculus. European Finite Element Fair.

LOGG, A. AND WELLS, G. N. 2010. DOLFIN: Automated finite element computing. *ACM Trans. Math. Softw. 37*, 20:1–20:28.

LOGG, A., MARDAL, K.-A., WELLS, G. N., ET AL. 2012. *Automated Solution of Differential Equations by the Finite Element Method*. Springer.

MORITA, S. 2001. *Geometry of Differential Forms*. Translations of Mathematical Monographs Series, vol. 201. Amsterdam Mathematical Society, Providence, RI.

MUNKRES, J. R. 1984. *Elements of Algebraic Topology*. Addison–Wesley.

NICOLAIDES, R. A. AND TRAPP, K. A. 2006. Covolume discretization of differential forms. In *Compatible Spatial Discretizations*, D. N. Arnold, P. B. Bochev, R. B. Lehoucq, R. A. Nicolaides, and M. Shashkov Eds., The IMA Volumes in Mathematics and its Applications Series, vol. 142. Springer, 161–171.

OLIPHANT, T. E. 2007. Python for scientific computing. *Comput. Sci. & Engin. 9*, 3, 10–20.

SAAD, Y. 2003. *Iterative Methods for Sparse Linear Systems* 2nd Ed. Society for Industrial and Applied Mathematics, Philadelphia, PA.

SEN, S. 2003. A cubic Whitney and further developments in geometric discretisation. arXiv:hep-th/0307166.

SHASHKOV, M. 1996. *Conservative Finite-Difference Methods on General Grids*. CRC Press, Boca Raton, FL.

VAN DER WALT, S., COLBERT, S., AND VAROQUAUX, G. 2011. The NumPy array: A structure for efficient numerical computation. *Comput. Sci. Engin. 13*, 2, 22–30.

WHITNEY, H. 1957. *Geometric Integration Theory*. Princeton University Press, Princeton, NJ.

WILSON, S. 2007. Cochain algebra on manifolds and convergence under refinement *Topol. Appl. 154*, 9, 1898–1920.

WILSON, S. 2008. Conformal cochains. *Trans. Amer. Math. Soc. 360*, 5247–5264.

ZOMORODIAN, A. AND CARLSSON, G. 2005. Computing persistent homology. *Discr. Comput. Geom. 33*, 2, 249–274.

Received March 2011; revised February 2012; accepted February 2012