Computational topology: Lecture 6

Alberto Paoluzzi

March 21, 2019

Topology computing with chains

Example A.2 (Chains)

Example A.2 (Chains)

Unoriented chains take coefficients from $\mathbb{Z}/2\mathbb{Z} = \mathbb{Z}_2 = \{0, 1\}$.

Example A.2 (Chains)

Unoriented chains take coefficients from $\mathbb{Z}/2\mathbb{Z} = \mathbb{Z}_2 = \{0, 1\}$.
$c \in C_0$ is given by $c = 1\nu_1 + 1\nu_2 + 1\nu_3 + 1\nu_5$. Hence, the coefficients associated to all other cells are zero.

Example A.2 (Chains)

Unoriented chains take coefficients from $\mathbb{Z}/2\mathbb{Z} = \mathbb{Z}_2 = \{0, 1\}$.
$c \in C_0$ is given by $c = 1\nu_1 + 1\nu_2 + 1\nu_3 + 1\nu_5$. Hence, the coefficients associated to all other cells are zero.
$[1, 1, 1, 0, 1, 0]^t$ is the coordinate vector of c with respect to the (ordered) basis $(u_1, u_2, \ldots, u_6) = (1\nu_1, 1\nu_2, \ldots, 1\nu_6)$.
Analogously for the 1-chain $d \in C_1$ and the 2-chain $e \in C_2$, written as $d = \eta_2 + \eta_3 + \eta_5$ and $e = \gamma_1 + \gamma_3$, with coordinate vectors $[0, 1, 1, 0, 1, 0, 0, 0]^t$ and $[1, 0, 1]^t$, respectively.

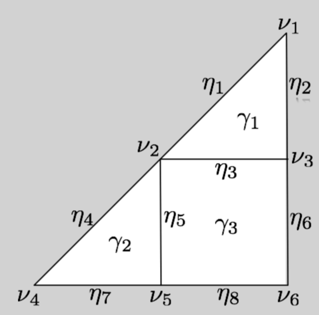


Figure 1: Cellular 2-complex

Example A.3 (Orientation).

Oriented version of the cellular complex
$\Lambda = \Lambda_0 \cup \Lambda_1 \cup \Lambda_2$

Example A.3 (Orientation).

Oriented version of the cellular complex
$\Lambda = \Lambda_0 \cup \Lambda_1 \cup \Lambda_2$
1-cells are oriented from vertex with lesser index to vertex with greater index, and all 2-cells are counterclockwise oriented

Example A.3 (Orientation).

Oriented version of the cellular complex
$\Lambda = \Lambda_0 \cup \Lambda_1 \cup \Lambda_2$
1-cells are oriented from vertex with lesser index to vertex with greater index, and all 2-cells are counterclockwise oriented
The orientation of each cell may be fixed arbitrarily, since it can always be reversed by the associated coefficient, now taken from $\{-1, 0, +1\}$

Example A.3 (Orientation).

Oriented version of the cellular complex
$\Lambda = \Lambda_0 \cup \Lambda_1 \cup \Lambda_2$
1-cells are oriented from vertex with lesser index to vertex with greater index, and all 2-cells are counterclockwise oriented
The orientation of each cell may be fixed arbitrarily, since it can always be reversed by the associated coefficient, now taken from $\{-1, 0, +1\}$
The oriented 1-chain w first vertex ν_1 and last vertex ν_5 is given as $d' = \eta_2 - \eta_3 + \eta_5$, with coordinate vector $[0, 1, -1, 0, 1, 0, 0, 0]^t$

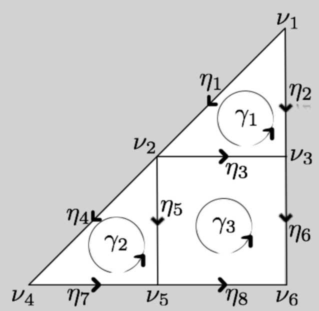


Figure 2: Oriented 2-complex

Example A.4 (Dual cochains)

0-cells are given arbitrary numbers, e.g., the values of an arbitrary scalar field;

Example A.4 (Dual cochains)

0-cells are given arbitrary numbers, e.g., the values of an arbitrary scalar field;
the values for 1-cells and 2-cells are computed from these using the coboundary relations

Example A.4 (Dual cochains)

0-cells are given arbitrary numbers, e.g., the values of an arbitrary scalar field;
the values for 1-cells and 2-cells are computed from these using the coboundary relations
$\boxed{\delta_0 = \partial_1^\top}$ and $\boxed{\delta_1 = \partial_2^\top}$
They are discrete gradients and curl values associated to 1-cells and 2-cells, respectively.

Example A.4 (Dual cochains)

0-cells are given arbitrary numbers, e.g., the values of an arbitrary scalar field;
the values for 1-cells and 2-cells are computed from these using the coboundary relations
$\boxed{\delta_0 = \partial_1^\top}$ and $\boxed{\delta_1 = \partial_2^\top}$
They are discrete gradients and curl values associated to 1-cells and 2-cells, respectively.
In discrete geometric calculus, cochains are functions from chains to reals.

Example A.4 (Dual cochains)

0-cells are given arbitrary numbers, e.g., the values of an arbitrary scalar field;
the values for 1-cells and 2-cells are computed from these using the coboundary relations
$\boxed{\delta_0 = \partial_1^\top}$ and $\boxed{\delta_1 = \partial_2^\top}$
They are discrete gradients and curl values associated to 1-cells and 2-cells, respectively.
In discrete geometric calculus, cochains are functions from chains to reals.
Colored numbers on 1- and 2-cells are exactly the evaluation $\phi^k(u_k) = \langle \phi^k, u_k \rangle$ of the dual elementary cochain on each elementary chain

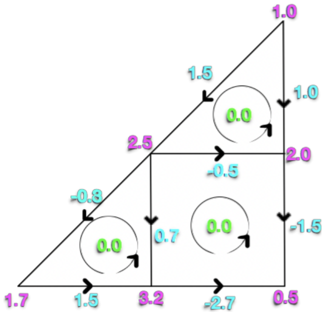


Figure 3: Elementary cochains

Definition (Boundary)

Boundary operators are maps $C_p \to C_{p-1}$, with $1 \leq p \leq d$.

Hence for a 2-complex we have two operators, denoted as

$$\partial_2 : C_2 \to C_1 \quad \text{and} \quad \partial_1 : C_1 \to C_0$$

As linear maps between linear spaces, may be represented by matrices of coefficients $[\partial_2]$ and $[\partial_1]$ from the underlying field $\mathbb{F}$.

Definition (Boundary)

Boundary operators are maps $C_p \to C_{p-1}$, with $1 \leq p \leq d$.

Hence for a 2-complex we have two operators, denoted as

$$\partial_2 : C_2 \to C_1 \quad \text{and} \quad \partial_1 : C_1 \to C_0$$

As linear maps between linear spaces, may be represented by matrices of coefficients $[\partial_2]$ and $[\partial_1]$ from the underlying field $\mathbb{F}$.

For the unsigned and the signed case (see previous slides) we have:

$$[\partial_2] = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \text{ and } \quad [\partial_2'] = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 0 & 0 \\ 1 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{1}$$

Example (Boundary)

Analogously, for the unsigned ∂_1 and the signed ∂_1' operators we have:

$$[\partial_1] = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}, \quad [\partial_1'] = \begin{pmatrix} -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & -1 & -1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}.$$

Example A.5 (Boundary)

As a check, let us compute:

1. the 0-boundary of the coordinate representations of the unsigned 1-chain $[d] = [0, 1, 1, 0, 1, 0, 0, 0]^t$ and

$$\partial_1 d = [\partial_1][d] \ mod \ 2 = [1, 0, 0, 0, 1, 0]^t = \nu_1 + \nu_5 \in C_0,$$

where the matrix product is computed *mod* 2, and

$$\partial_1' d' = [\partial_1'][d'] = [-1, 0, 0, 0, 1, 0]^t = \nu_5 - \nu_1 \in C_0'.$$

Example A.5 (Boundary)

As a check, let us compute:

1. the 0-boundary of the coordinate representations of the unsigned 1-chain $[d] = [0, 1, 1, 0, 1, 0, 0, 0]^t$ and
2. the signed 1-chain $[d'] = [0, 1, -1, 0, 1, 0, 0, 0]^t$

$$\partial_1 d = [\partial_1][d] \; mod \; 2 = [1, 0, 0, 0, 1, 0]^t = \nu_1 + \nu_5 \in C_0,$$

where the matrix product is computed *mod* 2, and

$$\partial_1' d' = [\partial_1'][d'] = [-1, 0, 0, 0, 1, 0]^t = \nu_5 - \nu_1 \in C_0'.$$

Example A.5 (Boundary)

```
julia> B_1 = [
        1  1  0  0  0  0  0  0;
        1  0  1  1  1  0  0  0;
        0  1  1  0  0  1  0  0;
        0  0  0  1  0  0  1  0;
        0  0  0  0  1  0  1  1;
        0  0  0  0  0  1  0  1]
6×8 Array{Int64,2}:
...

julia> d = [0,1,1,0,1,0,0,0];

julia> B_1 * d
6-element Array{Int64,1}:
1 2 2 0 1 0

julia> B_1 * d .% 2
6-element Array{Int64,1}:
1 0 0 0 1 0
```

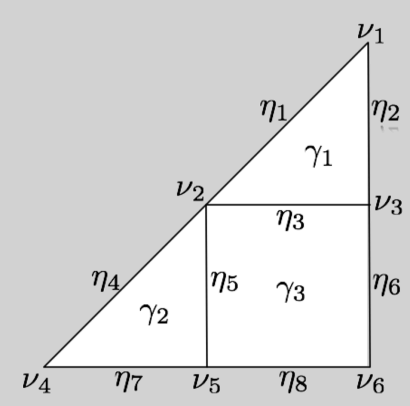


Figure 4: Cellular 2-complex

Example A.5 (Boundary)

```
julia> B_1 = [
       -1 -1  0  0  0  0  0  0;
        1  0 -1 -1 -1  0  0  0;
        0  1  1  0  0 -1  0  0;
        0  0  0  1  0  0 -1  0;
        0  0  0  0  1  0  1 -1;
        0  0  0  0  0  1  0  1]
6×8 Array{Int64,2}:
...

julia> d = [0,1,-1,0,1,0,0,0]
8-element Array{Int64,1}:
0 1 -1 0 1 0 0 0

julia> B_1 * d
6-element Array{Int64,1}:
-1 0 0 0 1 0
```

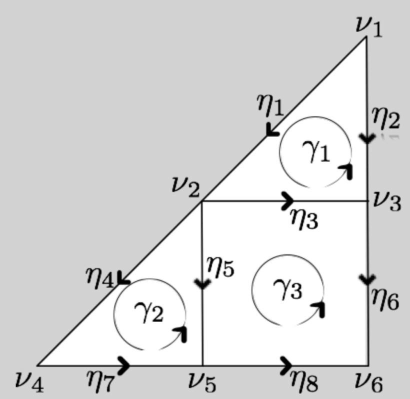


Figure 5: Cellular 2-complex

Example A.6 (Cell with a hole).

```
V = [[0.,0.],[3.,3.],[1.,2.],[2.,1.],[3.,0.],[1.,1.],[0.,3.],[2.,2.]]
FV = [[1,2,3,4,5,6,7,8],[3,4,6,8]]
EV = [[1,5],[1,7],[2,5],[2,7],[3,6],[3,8],[4,6],[4,8]]
```

```
M_1 = [
1   0   0   0   1   0   0   0;
1   0   0   0   0   0   1   0;
0   1   0   0   1   0   0   0;
0   1   0   0   0   0   1   0;
0   0   1   0   0   1   0   0;
0   0   1   0   0   0   0   1;
0   0   0   1   0   1   0   0;
0   0   0   1   0   0   0   1]

M_2 = [
1   1   1   1   1   1   1   1;
0   0   1   1   0   1   0   1]
```

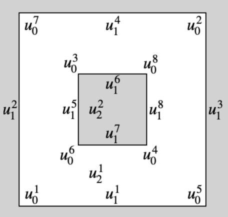


Figure 6: Cellular 2-complex

```
convert(
Array{Int64,2},(M_1 * M_2')
8×2 Array{Int64,2}:
 1  0
 1  0
 1  0
 1  0
 1  1
 1  1
 1  1
 1  1

julia> (B_2 * [1,1]) .% 2
8-element Array{Int64,1}:
1 1 1 1 0 0 0 0
```

Boundary and coboundary

Coboundary operator

The coboundary operator $\delta^p : C^p \to C^{p+1}$ acts on p-cochains as the dual of the boundary operator ∂_{p+1} on $(p+1)$-chains. For all $\phi^p \in C^p$ and $c_{p+1} \in C_{p+1}$:

$$\langle \delta^p \phi^p, c_{p+1} \rangle = \langle \phi^p, \partial_{p+1} c_{p+1} \rangle.$$

Coboundary operator

The coboundary operator $\delta^p : C^p \to C^{p+1}$ acts on p-cochains as the dual of the boundary operator ∂_{p+1} on $(p+1)$-chains. For all $\phi^p \in C^p$ and $c_{p+1} \in C_{p+1}$:

$$\langle \delta^p \phi^p, c_{p+1} \rangle = \langle \phi^p, \partial_{p+1} c_{p+1} \rangle.$$

Since chain-cochain duality means integration, this defining property is the combinatorial archetype of Stokes' theorem.

See also, $(\delta^1 \circ \delta^0)(\gamma_1) = 0$.

Coboundary operator

The coboundary operator $\delta^p : C^p \to C^{p+1}$ acts on p-cochains as the dual of the boundary operator ∂_{p+1} on $(p+1)$-chains. For all $\phi^p \in C^p$ and $c_{p+1} \in C_{p+1}$:

$$\langle \delta^p \phi^p, c_{p+1} \rangle = \langle \phi^p, \partial_{p+1} c_{p+1} \rangle.$$

Since chain-cochain duality means integration, this defining property is the combinatorial archetype of Stokes' theorem.

See also, $(\delta^1 \circ \delta^0)(\gamma_1) = 0$.

This property, i.e., $\delta \circ \delta = 0$, is the discrete archetype of the fact that the curl of gradient is zero.

Coboundary operator

The coboundary operator $\delta^p : C^p \to C^{p+1}$ acts on p-cochains as the dual of the boundary operator ∂_{p+1} on $(p+1)$-chains. For all $\phi^p \in C^p$ and $c_{p+1} \in C_{p+1}$:

$$\langle \delta^p \phi^p, c_{p+1} \rangle = \langle \phi^p, \partial_{p+1} c_{p+1} \rangle.$$

Since chain-cochain duality means integration, this defining property is the combinatorial archetype of Stokes' theorem.

See also, $(\delta^1 \circ \delta^0)(\gamma_1) = 0$.

This property, i.e., $\delta \circ \delta = 0$, is the discrete archetype of the fact that the curl of gradient is zero.

Note that a scalar field, in the discrete version, becomes a real valued 0-cochain to be valued on 0-chains, i.e., on 0-cells.

Coboundary operator

The coboundary operator $\delta^p : C^p \to C^{p+1}$ acts on p-cochains as the dual of the boundary operator ∂_{p+1} on $(p+1)$-chains. For all $\phi^p \in C^p$ and $c_{p+1} \in C_{p+1}$:

$$\langle \delta^p \phi^p, c_{p+1} \rangle = \langle \phi^p, \partial_{p+1} c_{p+1} \rangle.$$

Since chain-cochain duality means integration, this defining property is the combinatorial archetype of Stokes' theorem.

See also, $(\delta^1 \circ \delta^0)(\gamma_1) = 0$.

This property, i.e., $\delta \circ \delta = 0$, is the discrete archetype of the fact that the curl of gradient is zero.

Note that a scalar field, in the discrete version, becomes a real valued 0-cochain to be valued on 0-chains, i.e., on 0-cells.

Since we use dual bases, matrices representing dual operators are the transpose of each other: for all $p = 0, \dots, d-1$:

$$[\delta^p]^t = [\partial_{p+1}]$$

Interval trees

From Wikipedia

In computer science, an interval tree is a tree data structure to hold intervals.

It allows to efficiently find all intervals that overlap with any given interval or point.

It is often used for windowing queries, for instance, to find all roads on a computerized map inside a rectangular viewport, or to find all visible elements inside a three-dimensional scene.

A similar data structure is the segment tree.

The data structure

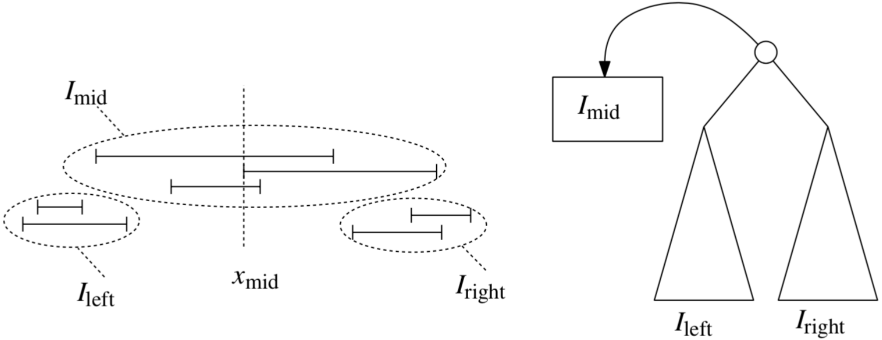
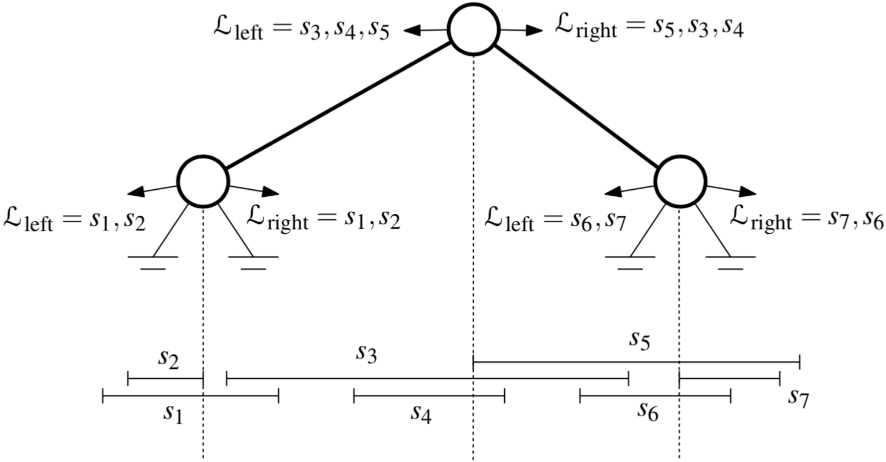


Figure 7: Classification of segments with respect to x_{mid}

Example

Dotted vertical segments indicate the values x_{mid} for each node



Description of the data structure

- If $I = \emptyset$ then the interval tree is a leaf.

Description of the data structure

- If $I = \emptyset$ then the interval tree is a leaf.
- Otherwise, let x_{mid} be the median of endpoints of the intervals. Let

Description of the data structure

- If $I = \emptyset$ then the interval tree is a leaf.
- Otherwise, let x_{mid} be the median of endpoints of the intervals. Let
  - $I_{left} = \{[x_j : x'_j] \in I : x'_j < x_{mid}\}$,

Description of the data structure

- If $I = \emptyset$ then the interval tree is a leaf.
- Otherwise, let x_{mid} be the median of endpoints of the intervals. Let
  - $I_{left} = \{[x_j : x'_j] \in I : x'_j < x_{mid}\}$,
  - $I_{mid} = \{[x_j : x'_j] \in I : x_j \leq x_{mid} \leq x'_j\}$,

Description of the data structure

- If $I = \emptyset$ then the interval tree is a leaf.
- Otherwise, let x_{mid} be the median of endpoints of the intervals. Let
  - $I_{left} = \{[x_j : x_j'] \in I : x_j' < x_{mid}\}$,
  - $I_{mid} = \{[x_j : x_j'] \in I : x_j \leq x_{mid} \leq x_j'\}$,
  - $I_{right} = \{[x_j : x_j'] \in I : x_{mid} < x_j\}$.

Algorithm

Algorithm CONSTRUCTINTERVALTREE(I)

Input. A set I of intervals on the real line.

Output. The root of an interval tree for I.

1. **if** $I = \emptyset$
2. **then return** an empty leaf
3. **else** Create a node ν. Compute x_{mid}, the median of the set of interval endpoints, and store x_{mid} with ν.
4. Compute I_{mid} and construct two sorted lists for I_{mid}: a list $\mathcal{L}_{\text{left}}(\nu)$ sorted on left endpoint and a list $\mathcal{L}_{\text{right}}(\nu)$ sorted on right endpoint. Store these two lists at ν.
5. $lc(\nu) \leftarrow$ CONSTRUCTINTERVALTREE(I_{left})
6. $rc(\nu) \leftarrow$ CONSTRUCTINTERVALTREE(I_{right})
7. **return** ν

Figure 9: from: de Berg, Otfried Cheong, van Kreveld, Overmars: *Computational Geometry, Algorithms and Applications*, Third Edition, Springer.

Julia Package

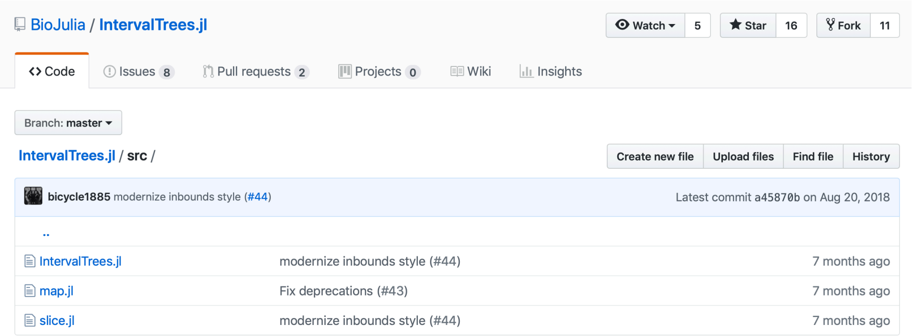


Figure 10: BioJulia/IntervalTrees.jls