ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФГАОУ ВО НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа «Прикладная математика и информатика»

Отчет о программном проекте

на тему: ПО для топологического анализа данных

Выполнил:
Студент группы БПМИ225 Кор С.А.Корняков
Подпись И.О.Фамилия

29.04.2025
Дата

Принял:
Руководитель проекта Качан Олег Николаевич
Имя, Отчество, Фамилия

-
Должность, ученое звание

Лаборатория ИИ Сбербанка
Место работы (Компания или подразделение НИУ ВШЭ)

Дата проверки ________ 2025
Оценка (по 10-ти бальной шкале) Подпись

Москва 2025

Содержание

Аннотация

Цель работы заключается в разработке библиотеки для топологического анализа данных, объединяющей современные алгоритмы построения симплициальных комплексов и вычисления персистентных гомологий. Основные задачи включают:

- Реализацию оптимизированных алгоритмов (`Twist`, `DoubleTwist`) для построения диаграмм устойчивости
- Вычисление гармонических представителей через ядро матрицы Ходжа-Лапласа
- Создание модульной архитектуры с поддержкой исследовательских модификаций
- Интеграцию современных структур данных (`BitTreeColumn`, оптимизированные деревья симплексов)

Библиотека предоставляет два режима работы: высокопроизводительный для бенчмаркинга и расширяемый для научных экспериментов. Эксперименты на реальных данных подтвердили преимущество новых алгоритмов над классическими подходами.

Ключевые слова: топологический анализ, симплициальный комплекс, Вьеторис-Рипс, DoubleTwist, диаграмма устойчивости, параллельное вычисление, C++, матрица Ходжа-Лапласа, гармонические представители

1 Введение

Современные задачи анализа данных в машинном обучении, вычислительной биологии и нейронауках требуют обработки высокоразмерных пространств. Традиционные библиотеки (GUDHI, PHAT), основанные на алгоритмах 2000-х годов, становятся неэффективны для работы с большими данными. На замену им приходят новые теории и алгоритмы, которые могут на порядок превосходить прежние алгоритмы по скорости.

Задача проекта заключается в создании библиотеки, которая реализует и комбинирует алгоритмы, собранные с множества новейших статей, для ускоренного и параллельного вычисления симплициальных комплексов, диаграмм устойчивости, ядра матрицы Ходжа-Лапласа и гармонических представителей. Библиотека имеет два направления:

1. Оптимизированная под конкретные алгоритмы для наибыстрейшего вычисления. Это позволит сравнить данную имплементацию с другими библиотеками на скорость
2. Модульная, с максимальным количеством шаблонов и абстракций. Данная версия необходима в научных и исследовательских целях для экспериментов над гипотезами. Она позволит быстро и удобно реализовывать нужные для исследований алгоритмы и объекты

В поставленные задачи входило:

Изучение теории

Изучена теория симплициальных комплексов, гомологических групп, диаграмм устойчивости, матрицы Ходжа-Лапласа на графах, гармонических цепей и циклов. Прочитаны реализованные в библиотеках алгоритмы для вычисления и построения этих объектов, а также исследованы новейшие алгоритмы, которые пока не имеют общедоступных реализаций.

Реализация необходимых объектов

Имплементированы на языке программирования C++ различные топологические объекты, нужные для вычисления гармонический циклов:

- Облако точек
- Симплекс, фильтрованный симплекс
- Дерево симплексов для быстрого нахождения их позиций в отфильтрованном комплексе [3]
    1. Обычная реализация с провязыванием соседей с одинаковыми номерами вершин на одной высоте
    2. Оптимизированные версии, чтобы быстро работать с полным 2-скелетом
- Фильтрованный симплициальный комплекс

- Разреженная матрица позиций, представляющая граничную или кограничную матрицу фильтрованного комплекса
- Стоблец позиций для быстрого сложения во время редуцирования граничных/кограничных матриц
    1. В виде обычного сета позиций
    2. В виде дерева битов [2]
- Диаграмма устойчивости

Имплементация алгоритмов

Написаны следующие алгоритмы на полученных объектах:

- Фильтрация облака точек для создания фильтрованного симплициального комплекса
    1. Создание полного 2-скелета комплекса Вьеториса-Рипса
    2. Создание ограниченного по размерности и величине функции фильтрации комплекса Вьеториса-Рипса [12]
- Создание граничной, кограничной и ориентированной граничной матриц фильтрованного симплициального комплекса.
- Редуцирование граничной матрицы фильтрованного симплициального комплекса:
    1. Наивная реализация
    2. Параллельное редуцирование [4]
    3. Twist [5]
    4. DoubleTwist [10]
- Вычисление матрицы Ходжа-Лапласа [8]
- Вычисление ядра разреженной матрицы
- Вычленение гармонического цикла из ядер матрицы Ходжа-Лапласа для различных позиций фильтрации

Абстрагирование алгоритмов и объектов

Выделены модели объектов и алгоритмов, которые с помощью CRTP позволяет по-разному их реализовывать. Это позволило написать различные реализации одних и тех же действий, и с лёгкостью написать для них тесты, просто подменяя шаблонные параметры.

Данная абстрация позволила не разделять репозиторий на две ветки, а содержать все имплементации в одной и по запросу их использовать.

Тестирование библиотеки

Тестирование проводилось с помощью фреймворка Catch2 [9]

Написаны юнит-тесты, стресс-тесты и тесты на корректность для всех объектов и алгоритмов. Используя синтетические и реальные данные, доказана корректность всех объектов и алгоритмов.

Для различных реализаций одних и тех же действий написаны бенчмарки, сравнивающие эти реализации, а также бенчмарки на масштабируемость.

Проведены сравнительные тесты с другими библиотеками и реализациями.

Изложение результатов

В отчёте изложена теория, необходимая для понимания рассматриваемых алгоритмов и структур данных, а также описана архитектура и дизайн библиотеки. Написана сопроводительная документация для пользователей библиотеки.

2 Требования к проекту

2.1 Функциональные требования

Библиотека позволяет

1. Считать облако точек из файлов разных форматов
2. Построить фильтрованный симплициальный комплекс из облака точек
3. Вычислить граничную/когграничную матрицы для фильтрованного комплекса
4. Найти устойчивые пары фильтрованного комплекса
5. Найти гармонические циклы фильтрованного комплекса
6. Подменять реализацию объектов/алгоритмов

2.1.1 Модели

Архитектура построена на принципах Generic Programming с использованием:

- CRTP (Curiously Recurring Template Pattern) для статического полиморфизма
- Концепций `C++20` для валидации типов
- Policy-based design для замены компонентов

Большинство функций, методов и классов принимают не конкретные реализации, а модели объектов/алгоритмов, что делает код универсальным и открывает возможность добавлять эксперементальный функционал без трудностей.

Перечислим основные модели (а также их реализации, если их несколько):

- Модель облака точек
- Модель фильтрации облака точек. Реализации:
  - `NewVR` - фильтрация Вьеториса-Рипса с настраиваемым ограничением по радиусу и размерности симплексов [12]
  - `FullVR` - фильтрация Вьеториса-Рипса, которая строит полный 2-скелет
- Модель дерева симплексов. Реализации:
  - `SimplexTree` - классическое дерево симплексов [3]
  - `FullTree` - оптимизированное для полного 2-скелета дерево
- Модель фильтрованного симплициального комплекса
- Модель матрицы инцидентности
- Модель кучи, которая имитирует исключающее "или"над позициями фильтрации. Реализации:
  - `SetHeap` - позиции хранятся в сете
  - `BitTreeHeap` - позиции хранятся в куче над вектором блоков [2]
- Модель вычисления устойчивых пар из фильтрованного комплекса. Реализации:
  - `Twist` — стандартная редукция граничной матрицы [5]
  - `DoubleTwist` — комбинированная редукция кограничной и граничной матриц [10]
- Модель вычисления гармонических циклов из фильтрованного комплекса и его устойчивых пар

2.2 Нефункциональные требования

2.2.1 Окружение программы

1. Программа написана на языке `C++`, используется версия языка `C++20`
2. Для сборки и тестирования кода используется кросс-платформенная система сборки `CMake`
3. Программа поддерживает множество компиляторов, но основной - `Clang`
4. Проект использует систему поддержки версий `git` вместе с `github`
5. Требуется не менее 32GiB RAM для корректной работы с большим объёмом данных
6. Для форматирования используется `clang-format`
7. Используется `clang-tidy` для унификации именований и статической диагностики программы

2.2.2 Используемые библиотеки

1. Библиотеки `Eigen` [7] и `Spectra` [11] для линейной алгебры над разреженными матрицами
2. Библиотека `oneTBB` [6] для параллелизации вычислений
3. Фреймворк `Catch2` [9] для тестирования программы

2.2.3 Политика обработки ошибок

1. Исключения включены, но не бросаются самой библиотекой
2. Вставлены `assert`-ы для выявления ошибок имплементации методов и алгоритмов
3. Используется `std::optional` для обработки несуществования выходных данных алгоритмов и методов

3 Теория и описание алгоритмов

Для ознакомления с основными понятиями см. раздел A

Опишем объекты и алгоритмы по порядку:

3.1 Фильтрация облака точек

Опустим вырожденные случаи, а также ограничения на индексы, которые можно вывести из контекста.

Определение 1. Пусть задано конечное множество вершин V. Симплициальным комплексом K называется подмножество 2^V, такое что:

1. Если $\sigma \in K$ и $\tau \subseteq \sigma$, то $\tau \in K$

Тогда:

- Каждый элемент $\sigma \in K$ называется симплексом
- Симплекс $\sigma \in K$, состоящий $k+1$ вершин, называется k-симплекс и его размерность равна $\dim \sigma := k$
- Размерностью комплекса $\dim K$ называется максимальная размерность его симплексов
- i-скелетом комплекса K называется $K^{(i)} := \{\sigma \in K \colon \dim \sigma \leq i\}$.

  Пример 2. 1-скелет имеет структуру обычного графа с вершинами и рёбрами
- Подкомплексом симплициального комплекса K называется комплекс $K' \colon K' \subseteq K$. В частности, все i-скелеты комплекса являются его подкомплексами

Определение 3. Фильтрацией симплициального комплекса K называется упорядоченное семейство симплициальных комплексов $\{K_f\}_{f\in A\subset\mathbb{R}}$, параметризованнное вещественным числом, такое что

$$f_1 \leq f_2 \implies K_{f_1} \subseteq K_{f_2}$$

Определение 4. Пусть задано конечное метрическое пространство (M, ρ). Для заданного радиуса $r \in [0, \infty)$, комплекс Вьеториса-Рипса определяется как:

$$\mathrm{VR}(M, r) = \{S \subseteq M : \forall v, u \in S \;\; \rho(v, u) \leq r\}$$

Фильтрация Вьеториса-Рипса - это коллекция комплексов $\mathrm{VR}(M, r)$:

$$\mathrm{VR} = \{\mathrm{VR}(M, r)\}_{r\in[0,\infty)}$$

Часто в топологическом анализе ограничивают $r \leq r_{max}$ и $|S| \leq d_{max}+1$, чтобы работать только с k-скелетами комплексов.

Реализация фильтрации `FullVR` использует $r_{max} = \infty$ и $d_{max} = 2$, то есть просто строит полный 2-скелет. Но достаточно часто оказывается, что комплексы с радиусом больше некоторого порога не создают полезные гармонические циклы, а просто засоряют данные и замедляют вычисления. Поэтому был имплементирован новейший алгоритм `New-VR`, который позволяет эффективно строить такие фильтрации.

3.1.1 New-VR

Данный алгоритм использует несколько топологических наблюдений для ускорения построения фильтрации.

Лемма 5. *Пусть σ_1, σ_2 - два k-симплекса, и у них есть общая $(k-1)$-грань ρ. Пусть $v_i := \sigma_i \setminus \rho$. Если симплекс $\tau := \{v_0, v_1\} \in K$, то $\sigma_1 \cup \sigma_2$ является $(k+1)$-симплексом*

Определение 6. Пусть K - симплициальный комплекс, и множество его вершин линейно-упорядочено. Тогда для каждой размерности k, мы введём лексикографический порядок для k-симплексов по порядку их вершин, т.е. $\sigma_1 < \sigma_2 \Leftrightarrow$ упорядоченный список вершин σ_1 лексикографически меньше упорядоченного списка вершин σ_2.

Аналогично введём порядок на парах различных k-симплексов:

$$(\sigma_1 < \sigma_2) < (\tau_1 < \tau_2) \Leftrightarrow \begin{cases} \sigma_1 < \tau_1, \text{ or} \\ \sigma_1 = \tau_1 \text{ and } \sigma_2 < \tau_2 \end{cases}$$

Лемма 7. *Порядок выше задаёт линейно упорядоченное множество на парах различных k-симплексов.*

Теорема 8. *Каждому $(k+1)$-симплексу соответствует ровно одна минимальная пара его k-граней и наоборот.*

Данный факт позволяет при построении комплекса избегать дублирования симплексов и эффективно проверять только одно ребро для каждой пары $(\sigma_1 < \sigma_2)$.

Описание алгоритма создания фильтрации Вьеториса-Рипса из облака точек:

1. Постройка графа смежности

Algorithm 1: BuildAdjacencyGraph

Input: Облако точек $P = [p_1, \ldots, p_n]$, радиус r_{max}
Output: Граф смежности G
$G \leftarrow \varnothing$ `// Инициализация пустого графа`
for $u \in 1..n$ **do**
  for $v \in u + 1..n$ **do**
    if $\rho(p_u, p_v) \leq r_{max}$ **then**
      $G[u] \leftarrow G[u] \cup \{v\}$;
    end
  end
end

2. Рекурсивное добавление кограней:

Algorithm 2: NewAddCofaces

```
Input: Текущий симплекс τ, верхние соседи N, текущий радиус r_curr, фильтрация F, граф G
Output: Обновлённая фильтрация F
F ← F ∪ {τ, r_curr} // Добавление симплекса τ в комплексы радиуса ≥ r_curr
if |τ| > d_max then
    return
end
for v ∈ N do
    σ ← τ ∪ {v}
    r_new ← max(r_curr, max_{u∈τ} ρ(p_u, p_v))
    M ← N ∩ G[v] // Пересечение верхних соседей
    NewAddCofaces(σ, M, r_new, F, G)
end
```

3. Сам алгоритм можно выразить как

Algorithm 3: NewVR

```
Input: Облако точек P = [p_1, ..., p_n], радиус r_max, размерность d_max
Output: Фильтрация F
G ← BuildAdjacencyGraph(P, r_max)
F ← ∅
for u ∈ 1..n do
    τ ← {u}
    N ← {v ∈ G[u]: v > u}
    NewAddCofaces(τ, N, 0, F, G)
end
```

3.2 Диаграмма устойчивости

Далее предполагаем, что вершины линейно-упорядочены.

3.2.1 Гомологическая группа

Определение 9. Пусть K - симплициальный комплекс, а Σ_k - множество его k-симплексов. Тогда k-цепной группой $C_k(K)$ называется свободная абелева группа с базисом $B_k = \{[\sigma] \colon \sigma \in \Sigma_k\}$:

$$C_k(K) := \left\{ \sum_{i=1}^{n} a_i \sigma_i \right\}$$

k-цепью называется элемент k-цепной группы. Обычно a_i берут из поля $\mathbb{Z}_2$.

Определение 10. Граничным оператором $\delta_k \colon C_k(K) \to C_{k-1}(K)$ называется оператор, который отображает цепь $c = \sum a_i \sigma_i$ в:

$$\delta_k(c) = \sum a_i \delta_k'(\sigma_i)$$

где δ_k' отображает симплекс $\sigma := [v_1, \ldots, v_k]$ в

$$\delta_k(\sigma) := \sum_{i=1}^{k} (-1)^{k-1} [v_1, \ldots, \hat{v}_i, \ldots, v_k]$$

где $\hat{v}_i$ означает удаление вершины v_i.

Пример 11. Цепь $c := [v_1, v_2, v_3] - [v_2, v_3, v_5]$ отобразится в

$$\delta_3(c) = [v_2, v_3] - [v_1, v_3] + [v_1, v_2] - ([v_3, v_5] - [v_2, v_5] + [v_2, v_3]) =$$

$$= -[v_1, v_3] + [v_1, v_2] - [v_3, v_5] + [v_2, v_5]$$

С помощью данного оператора можно дать определения топологическим дыркам:

Определение 12. Пусть $C_k(K)$ - k-цепная группа симплициального комплекса K. Тогда k-цепь $z \in C_k(K)$, для которой выполнено

$$\delta_k(z) = 0$$

называется k-циклом. Можно заметить, что множество всех k-циклов образуют ядро граничного оператора:

$$Z_k(K) := \operatorname{Ker} \delta_k$$

Пример 13. Цепь из рёбер $c := [v_1, v_2] + [v_2, v_3] - [v_1, v_3]$ образует цикл, если провести ориентированные рёбра между этими вершинами:

$$\delta_1(c) = [v_2] - [v_1] + [v_3] - [v_2] - [v_3] + [v_1] = 0$$

Определение 14. Пусть $C_k(K)$ - k-цепная группа симплициального комплекса K. Тогда k-границей называют k-цепь, которая является границей некоторой $(k+1)$-цепи, обозначим их:

$$B_k(K) := \operatorname{Im} \delta_{k+1}$$

Лемма 15. *$B_k(K) \subseteq Z_k(K)$, так как $\delta_k \circ \delta_{k+1} = 0$.*

Определение 16. Группа гомологий $H_k(K)$ называется факторгруппа

$$H_k(K) := Z_k(K)/B_k(K)$$

Элементы $H_k(K)$ называются классами гомологий. Два k-цикла называются гомологичными, если они принадлежат одному классу гомологий.

Пример 17. Классы гомологий можно представить как объединения всех способов описать дырку циклами. На рисунке 1 дырку из 4 вершин описывают 3 разными циклами z_1, z_2 и z_3.

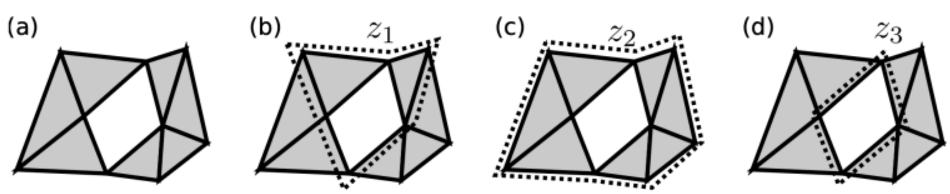


Рис. 1: Homology classes

3.2.2 Устойчивая гомология

Фильтрацию F комплекса K можно представить как вложенную последовательность подкомплексов:

$$\varnothing = K_0 \subset K_1 \subset \cdots \subset K_m = K$$

Заметим, что эти вложенные комплексы могут возрастать скачками, то есть добавлять сразу множество симплексов. Чтобы это исправить, надо ввести порядок на этих симплексах:

Определение 18. Пусть F - фильтрация симплициального комплекса K. Введём на симплексах этого комплекса линейный порядок по:

1. Значению фильтрации
2. Размеру симплекса
3. Лексикографическому порядку вершин в симплексах

Назовём позицией симплекса его номер в этом линейно-упорядоченном множестве. Заметим, что теперь можно представить фильтрацию комплекса как постепенное добавление симплексов в комплекс:

$$\varnothing = \{\} \subset \{\sigma_1\} \subset \{\sigma_1, \sigma_2\} \subset \cdots \subset \{\sigma_1, \dots, \sigma_n\} = K$$

Для удобства обозначений, пусть $K_i := \{\sigma_1, \dots, \sigma_i\}$.

Во время добавления нового симплекса σ_i размерности k к комплексу K_{i-1} может произойти только одна из следующих вещей:

1. σ_i создаст новый гомологический класс

2. σ_i разрушит существующий гомологический класс

Назовём симплекс позитивным или негативным соответственно. Заметим, что каждому негативному симплексу соответствует ровно один уникальный позитивный симплекс.

Определение 19. Пара симплексов (σ_i, σ_j), где $i < j$, называется устойчивой, если добавление симплекса σ_j в фильтрацию разрушило гомологический класс, который создало добавление симплекса σ_i.

Замечание 20. Если у позитивного симплекса не оказалось пары, его называют существенным, но такие симплексы нам не интересны.

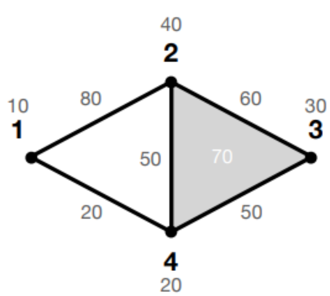
3.2.3 Граничные матрицы

Определение 21. Гранью симплекса σ называется любой $\tau \in K : \tau \subseteq \sigma$. Назовём грань с размерностью $\dim \tau = \dim \sigma - 1$ правильной гранью (`facet`). Границей симплекса называется множество его правильных граней.

Определение 22. Если τ - грань симплекса σ, то для τ симплекс σ является кограню. Аналогично определяется правильная когрань (`cofacet`) и кограница симплекса.

Определение 23. Граничной матрицей фильтрации F с n симплексами называется бинарная квадратная матрица M размера $n \times n$, где $M_{i,j} = 1$ для каждой пары (σ_i, σ_j), если σ_i является правильной гранью σ_j, и $M_{i,j} = 0$ иначе.

Пример 24. На рисунке 2 подобие граничной матрицы для фильтрованного симплициального комплекса



B =

	10	20	20	30	40	50	50	60	70	80
	1	**4**	**14**	**3**	**2**	**24**	**34**	**23**	**234**	**12**
0										
1			1							1
4			1			1	1			
14										
3							1	1		
2						1		1		1
24									1	
34									1	
23									1	
12										

Filtration function provides order on simplices, therefore on columns and rows of the filtration matrix. Ties are broken, first by simplex dimension, second by lexicographic order given by order on vertices.

Рис. 2: Boundary matrix

Из граничной матрицы мы можем получить позиции устойчивых пар с помощью её редуцирования. Рассмотрим различные способы её редуцирования.

Определение 25. Пусть M_j - j столбец граничной матрицы M. Назовём нижней позицией в этом слобце:

$$\operatorname{low}(M_j) := \max\{i \in 1, \dots, n \colon M_{i,j} = 1\}$$

Наивный способ редуцирования предполагает метод Гаусса на колонках граничной матрицы, т.е.

Algorithm 4: Standard Reduction

Input: Граничная матрица M с n колонками
Output: Редуцированная граничная матрица M
$L \leftarrow [0, \dots, 0]$ размера n `// в` $L[i]$ `мы храним индекс столбца` j`, для которого` $\operatorname{low}(M_j) = i$
for $j \in 1..n$ **do**
    while $M_j \neq 0$ *and* $L[\operatorname{low}(M_j)] \neq 0$ **do**
        $M_j \leftarrow M_j + M_{L[\operatorname{low}(M_j)]}$ `// выполняется в поле` $\mathbb{Z}_2$
    end
    if $M_j \neq 0$ **then**
        $L[\operatorname{low}(M_j)] \leftarrow j$
    end
end
return M

Лемма 26. *Пусть M - редуцированная граничная матрица. Тогда каждый ненулевой столбец в ней указывает на индексы симплексов в устойчивых парах, т.е.*

$$\{(i, j) \colon i = \operatorname{low}(M_j) \text{ and } M_j \neq 0\} \Leftrightarrow (\sigma_i, \sigma_j) \text{ является устойчивой парой}$$

Пример 27. Пример редуцирования фильтрованного 1-скелета можно увидеть на рисунке 3. Получаем следующие устойчивые пары: [(σ_2,σ_4), (σ_1, σ_4), (σ_3, σ_6)].

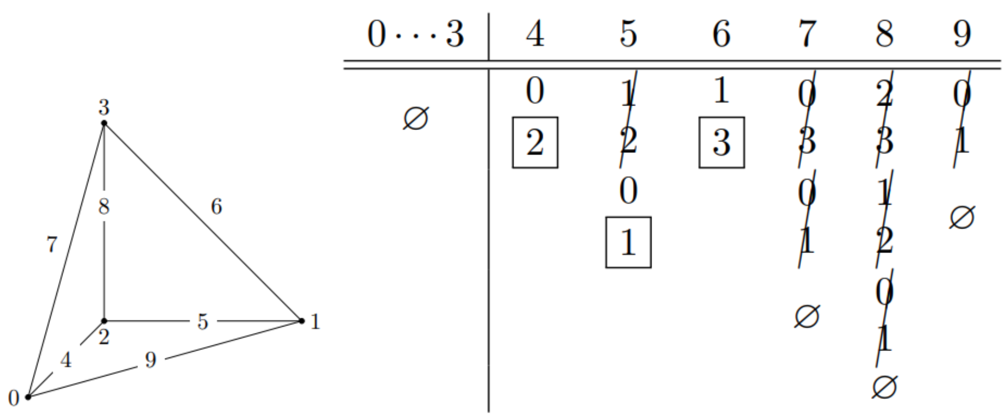


Рис. 3: Standard reduction

3.2.4 Twist

Лемма 28. *Пусть M - редуцированная граничная матрица фильтрации симплексов $[\sigma_1, \dots, \sigma_n]$. Тогда каждая колонка, которая соответствует позитивному симплексу в этой фильтрации нулевая.*

Значит, если сначала средуцировать колонки, отвечающие за $(p+1)$-симплексы, то их нижние позиции будут указывать на позиции столбцов p-симплексов, которые можно занулить без длительного редуцирования:

Algorithm 5: Twist Reduction

```
Input: Граничная матрица M с n колонками, максимальная размерность симплекса в фильтрации d
Output: Редуцированная граничная матрица M
L ← [0, ..., 0] размера n // в L[i] мы храним индекс столбца j, для которого low(M_j) = i
C ← структура данных для быстрого получения low(C)
for δ ∈ d..1 do
    for j ∈ 1..n do
        if dim σ_j ≠ δ then
            continue
        end
        C ← M_j
        while C ≠ 0 and L[low(C)] ≠ 0 do
            C ← C + M_{L[low(C)]} // выполняется в поле Z_2
        end
        if C ≠ 0 then
            L[low(M_j)] ← j
            M_{low(C)} ← 0 // killing
        end
        M_j ← C
    end
end
return M
```

В алгоритме также используетс структура данных для быстрого получения максимальной позиции в колонке. Для этой задачи подойдёт `BitTree`.

3.2.5 BitTree

Дерево представлено в виде кучи из массива 64 битных блоков. Уровни дерева:

- Родительские узлы: блоки, где бит i указывает на наличие дочерних элементов в поддереве i.
- Листья: блоки, хранящие битовые маски позиций (уровень 0).

Опишем два главных алгоритма:

Algorithm 6: Поиск максимальной позиции

```
Input: Поля BitTree
Output: Позиция старшего установленного бита
if data_[0] = 0 then
    // Дерево пусто - из корня нет рёбер
    return UnknownPos
end
node ← 0
index ← 0
while true do
    block ← data[node]
    index ← RightmostBit(block)
    next_node ← (node × 64) + index + 1
    if next_node ≥ |data| then
        Break
    end
    node ← next_node
end
pos ← (node − offset) × 64 + index
return pos
```

Поиск RightmostBit $(block)$ выполняется с помощью битхака [1]

$$\text{RightmostBit}\,(block) := 63 - \text{DeBruijn}_{64}\left[(block \mathbin{\&} -block) \cdot C \gg 58\right]$$

где $C = 0x07EDD5E59A4E28C2$ — константа для 64-битной де Брёйновой последовательности.

Algorithm 7: Обновление бита для позиции (Xor)

Input: Позиция pos
$block\_idx \leftarrow \left\lfloor \frac{pos}{64} \right\rfloor$
$bit\_idx \leftarrow 63 - (pos \mod 64)$
$mask \leftarrow 1 \ll bit\_idx$
$\texttt{data}[block\_idx + \text{offset}] \leftarrow \texttt{data}[block\_idx + \text{offset}] \oplus mask$
while $block\_idx > 0$ **do**
  $parent\_block \leftarrow \left\lfloor \frac{block\_idx - 1}{64} \right\rfloor$
  $parent\_bit \leftarrow (block\_idx - 1) \mod 64$
  $parent\_mask \leftarrow 1 \ll (63 - parent\_bit)$
  $\texttt{data}[parent\_block + \text{offset}] \leftarrow \texttt{data}[parent\_block + \text{offset}] \oplus parent\_mask$
  $block\_idx \leftarrow parent\_block$
end

3.2.6 DoubleTwist

Заметим, что если мы исследуем полные k-скелеты, то симплексов больших размерностей будет намного больше, чем малых, поэтому техника `Twist` неэффективна в таком случае. На помощь нам приходят когомологии.

Введём нотацию $i^\star = n - 1 - i$, где n - общее количество симплексов.

Определение 29. Кограничной матрицей фильтрации F с n симплексами называется бинарная квадратная матрица M размера $n \times n$, где $M_{i^\star, j^\star} = 1$ для каждой пары $(\sigma_{i^\star}, \sigma_{j^\star})$, если $\sigma_{i^\star}$ является правильной awгранью $\sigma_{j^\star}$, и $M_{i^\star, j^\star} = 0$ иначе.

Лемма 30. *Если $j^\star$ является максимальной позицией к столбце $i^\star$ в редуцированной кограничной матрице, то $(\sigma_{i^\star}, \sigma_{j^\star})$ - устойчивая пара.*

Заметим, что это отличается от случая граничной матрицы, где если бы j была максимальной позицией в колонке i, то ей соответствовала бы устойчивая пара (σ_j, σ_i).

Лемма 31. *Аналогично граничному случаю, если $j^\star$ максимальная позиция непустого столбца $i^\star$, то можно занулить столбец $j^\star$.*

Так как симплекс столбца $j^\star$ имеет большую размерность, чем симплекс столбца $i^\star$, то мы можем занулять столбцы симплексов больших размерностей, редуцируя сначала симплексы малых размерностей.

Получив редуцированную кограничную матрицу, нам надо отфильтровать столбцы граничной матрицы, чтобы получить все устойчивые пары. Фильтрация происходит следующим образом:

Algorithm 8: Saving technique

Input: Редуцированная кограничная матрица M размера n
Output: Позиции отфильтрованных столбцов
$S \leftarrow \varnothing$
for $j^\star \in 1..n$ **do**
  if $M_{j^\star} \neq 0$ **then**
    $S \leftarrow S \cup \{\text{Low}(M_{j^\star})\}$
  end
end
return S

Затем мы конструируем граничную матрицу, где ненулевыми столбцами могут быть только сохранённые позиции. Далее редуцируем её как обычно.

Лемма 32. *Редуцированная граничная матрица, полученная с помощью техники `DoubleTwist` совпадает с матрицей, редуцированной с помощью обычных методов (метод Гаусса, `Twist`).*

Опишем весь алгоритм целиком:

Algorithm 9: DoubleTwist Reduction

Input: Фильтрация F размера n

Output: Редуцированная граничная матрица M

```
M ← CoboundaryMatrix(F)
reduce M // например, с помощью Twist, но с возрастающим обходом размерностей
S ← SavedPoses(M)
delete M from memory
B ← ∅
foreach i ∈ S do
  | B[i] ← Boundary(σ_i)
end
reduce B
return B
```

Остался вопрос, как быстро посчитать кограничную матрицу.

Можно заметить, что кограничная матрица является анти-транспонированной граничной матрицей. Но постройка и анти-транспонирование граничной матрицы намного медленнее, чем построение её с нуля.

Здесь появляется проблема: чтобы построить кограничную матрицу, надо быстро получать позиции правильных кограней симплексов в фильтрации.

Одним из решений будет генерировать всевозможные правильные构 когрaни симплекса, но тогда появляется другая проблема: как проверить, что полученная кограль присутствует в фильтрации? Данная проблема не возникала с гранями, так как в симплициальном комплексе присутствуют все подсимплексы всех симплексов.

3.2.7 SimplexTree

Для решения этой проблемы была использована структура данных под названием `SimplexTree`. Это древовидная структура для представления симплициальных комплексов. Каждый узел соответствует вершине симплекса, а путь от корня к узлу представляет собой симплекс. Есть два уровня:

- Корневой, где располагаются симплексы, представляющие собой вершины. Обычно представлен в виде массива указателей на следующий слой.
- Не корневой, представляет собой отдельные узлы дерева

В узлах дерева дополнительно хранится позиция симплекса, который представляет данный узел, в фильтрации, для быстрого и удобного получения позиции по номерам вершин симплекса.

Также, дерево поддерживает для каждой глубины дерева словарь списков, используемый для ускорения операций с ко-гранями. В списках хранятся все узлы дерева с заданной вершиной на данной глубине. Структура позволяет быстро находить все узлы определённого уровня, содержащие заданную вершину.

Опишем основные методы класса:

Algorithm 10: Добавление симплекса

Input: Отсортированный симплекс $S = \{v_1, v_2, \ldots, v_k\}$, его позиция в фильтрации pos
Output: Обновленное дерево
$current \leftarrow null$ $depth \leftarrow 1$
foreach $v_i \in S$ **do**
  if $current = null$ **then**
    $next \leftarrow root[v_i]$
  else
    $next \leftarrow current.next[v_i]$
  end
  if *next не существует* **then**
    $next \leftarrow$ Новый узел с родителем $current$ и вершиной v_i
    Связать узел в списке уровня глубины $depth$ и вершиной v_i
  end
  $current \leftarrow next$
  $depth \leftarrow depth + 1$
end
if $current! = null$ **then**
  $current.pos \leftarrow pos$
end

Algorithm 11: Получение позиций кограней

Input: Отсортированный симплекс $S = \{v_1, v_2, \ldots, v_k\}$
Output: Множество позиций кограней
$cofacets \leftarrow \emptyset$
if $k = 0$ **then**
  return все корневые узлы
end
$base \leftarrow \mathrm{Find}(S)$
if $base = null$ **then**
  return $cofacets$
end
foreach $child \in base.next$ **do**
  $cofacets \leftarrow cofacets \cup child.pos$
end
$list \leftarrow$ Список нод на уровне $k + 1$ для вершины v_k
foreach $node \in list$ **do**
  if *node содержит base как подсимплекс с 1 пропуском* **then**
    $cofacets \leftarrow cofacets \cup node.pos$
  end
end
return $cofacets$

3.2.8 Промежуточные итоги

Подводя итог, с помощью редуцированная граничной матрицы мы получили позиции пар устойчивых симплексов, которые все вместе составляют диаграмму устойчивости. В разной литературе под диаграммой устойчивости подразумевают пары значения фильтрации симплексов, а не их позиции в фильтрации, но

- Можно легко перевести их в пары значений фильтрации по позициям в фильтрованном симплициальном комплексе
- Нам нужны именно позиции рождения и смерти для следующего шага

3.3 Гармонические представители

3.3.1 Постановка задачи

Устойчивая гомология примененяется к анализу мозговых сетей для определения формы мозговых сетей при различных пороговых значениях. Для измерения различий в сетях был предложен новый метод, а именно, использование гармонических дыр, которые выделяют подструктуры сетей мозга. Результаты показали, что эффективность кластеризации по гармоническим дырам выше, чем у сетевых расстояний, основанных только на глобальном изменении топологии.

Возникает задача быстрого вычисления всех гармонических дыр, полезных для выделения подструктур мозга. Но сначала надо ввести их определение.

3.3.2 Матрица Ходжа-Лапласа

Определение 33. Пусть граничный оператор δ_k отобразил цепь из k-симплекса $[v_1, \ldots, v_n]$ в сумму его правильных граней:

$$\delta_k(\{\sigma_k\}) = \sum_{j=1}^{k} (-1)^{j-1}[v_1, \ldots, \hat{v_j}, \ldots, v_k]$$

Тогда полученные грани называются положительно/отрицательно ориентированными относительно симплекса σ_k.

Определение 34. Ориентированной граничной матрицей фильтрации F с n симплексами называется квадратная матрица M размера $n \times n$, где $M_{i,j} = \pm 1$ для каждой пары $(\sigma_i,\, \sigma_j)$, если σ_i является правильной гранью σ_j, и $M_{i,j} = 0$ иначе. Знак зависит от ориентации симплекса σ_i относительно симплекса σ_j.

Введём обозначения для подматриц ориентированной матрицы M:

- M_i - подматрица M, содержащая только столбцы, соответствующие i-симплексам фильтрации
- M^k - подматрица M, содержащая столбцы $[1, \ldots, k] \subseteq [1, \ldots, n]$
- M_i^k - комбинация верхних ограничений

Определение 35. Комбинаторный Лапласиан Ходжа $L_i \colon C_i(K) \to C_i(K)$ определяется как:

$$L_i := M_i^T M_i + M_{i+1} M_{i+1}^T$$

Элементы $\ker L_i$ называют гармоническими циклами, а само ядро - гармоническим пространством H_i.

Лемма 36. $\operatorname{rk} H_i = \operatorname{rk} H_i\tilde(K)$

Таким образом, группу гомологий (обозначим её $H_i\tilde(K)$) можно заменить гармоническим пространством.

3.3.3 Гармонические дырки

Определение 37. Аналогично группе гомологий, в гармоническом пространстве H_i присутствуют дырки, назовём их гармоническими дырками.

Пусть у нас есть фильтрация 2-скелета симплициального комплекса с p вершинами, q рёбрами и r треугольниками. Тогда $L_1 \in \mathbb{Z}^{q \times q}$ и

$$H_1 = \{x \in \mathbb{R}^{q \times 1} \colon L_1 x = 0\}$$

Собственный вектор L_1 с нулевым собственным значением $x \in \mathbb{R}^{q \times 1}$ является представителем гармонической дырки. Абсолютное значение каждого элемента вектора x представляет вес соответствующего ребра, то есть по появлению ребра в фильтрации.

Так как x и $-x$ оба имеют нулевое собственное значение, то они представляют одну и ту же гармоническую дырку, и $||x|| = 1$.

Пример 38. Пусть мы получили собственный вектор x, представляющий некую гармоническую дырку. Отобразим его веса на рёбра как их толщину на рисунке 4.

Явно можно наблюдать, что этот собственный вектор описывает дырку на вершинах 2, 3, 4, 6, 5. Остальные рёбра также имеют некий вес, но он мал по сравнению с весом рёбер описываемой дырки.

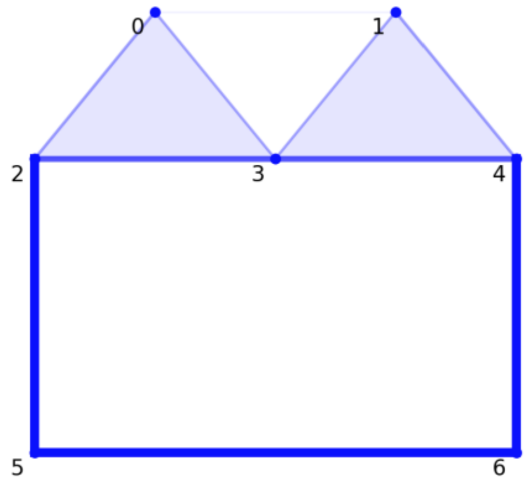


Рис. 4: Harmonic hole

Но чтобы отследить все дырки, которые появлялись и исчезали во время фильтрации, пришлось бы вычислять ядро

$$L_1^k = (M_1^k)^T M_1^k + M_2^k (M_2^k)^T$$

для каждого момента фильтрации, от 1 до n, а это очень дорого из-за спектрального разложения. Поэтому, берут не все моменты фильтрации, а только моменты появления и исчезновения дырок, а это как раз и есть моменты из диаграммы устойчивости.

3.3.4 Сепарация гармонических циклов

Пусть мы рассматриваем пару из (b_i, d_i) - моменты рождения и смерти очередной дырки. При нахождении ядра матрицы Ходжа-Лапласа мы получаем множество из гармонических циклов. Для выделения представителя гармонической дырки рассмотрим следующий метод

1. Пусть $i_X = b_i$, $i_Y = d_i - 1$, $i_Z = d_i$. i_Y - это последний момент перед смертью дырки. Вычисляем гармоническое пространство для каждого из этих моментов фильтрации:

$$H_X = [x_1, \dots, x_l] \in \mathbb{R}^{q \times l}, \;\; H_Y = [y_1, \dots, y_m] \in \mathbb{R}^{q \times m}, \;\; H_Z = [z_1, \dots, z_n] \in \mathbb{R}^{q \times n}$$

2. Искомый представитель где-то в H_X и H_Y, но не в H_Z

   Лемма 39. *Если $y \in H_Y$ линейно-зависим со столбцами H_Z, то минимальное сингулярное значение матрицы $[H_Z, y]$ будет близко к 0.*

   Так как существует $y \in H_Y$, линейно-независимый с H_Z, то мы выбираем y как:

$$y = \underset{y \in H_Y}{\operatorname{argmax}} \{\sigma_{min} \text{ of } [H_Z, y]\}$$

   Назовём его старшим представителем.

3. Выберем младшего представителя как:

$$x = \underset{x \in H_X}{\operatorname{argmin}} \{\sigma_{min} \text{ of } [x, y]\} = \underset{x \in H_X}{\operatorname{argmin}} \left\{ \sqrt{1 - |x^T y|} \right\} = \underset{x \in H_X}{\operatorname{argmax}} \{|x^T y|\}$$

Получаем для каждой гармонической дырки два представителя.

3.3.5 Использование результатов

Полученные на рёбрах веса можно агрегировать на вершины, складывая веса инцидентных рёбер или применяя другую функцию.

Далее, полученных представителей используют для подсчёта сетевых расстояний или дифференциации гармонический дырок. Также, к полученным данным применяются методы машинного обучения для дальнейшей идентификации и кластеризации гармонических дыр, но это выходит за рамки данной работы.

4 Дизайн решения

4.1 Структурная диаграмма классов

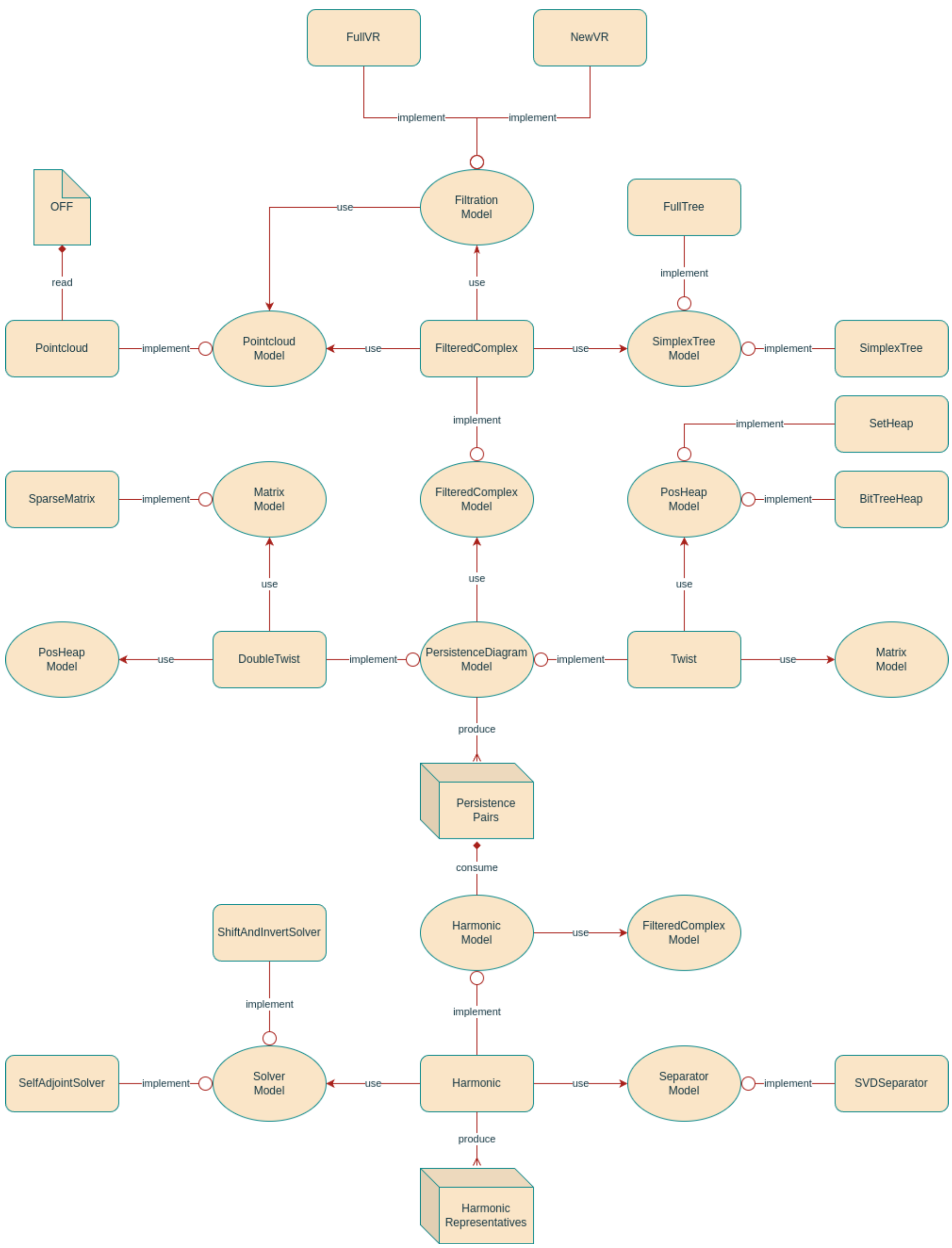


Рис. 5: Class Diagram

Описание потока данных

1. data.off → Pointcloud: Файл с набором точек считывается в облако точек
2. Pointcloud → Filtration: Облако точек преобразуется в фильтрацию (например, полный комплекс Вьеториса-Рипса)
3. Filtration → FilteredComplex: Симплексы сортируются и сохраняются в структуру данных
4. FilteredComplex → PersistenceDiagram: Строится и редуцируется граничная матрица. По ней получаются устойчивые пары
5. PersistenceDiagram → Harmonic: Пары передаются для анализа гармонических циклов
6. Harmonic + Solver, Separator → HarmonicRepresentatives: Для каждой пары вычисляется её старший и младший представители

4.2 Модели и их реализации

- **Pointcloud** — структура для хранения произвольно облака точек. Реализация позволяет считывать данные из файлов
- **Filtration** — класс для построения фильтрации из облака точек. Генерирует симплексы и их значения фильтрации (например, полный 2-скелет комплекса Вьеториса-Рипса в `FullVR`)
- **SimplexTree** — структура для хранения симплексов и их позиций. Реализации: `FullTree` (для полного 2-скелета) и `SimplexTree` (для произвольного комплекса)
- **FilteredComplex** — базовый класс для представления фильтрованного комплекса. Хранит симплексы с их позициями и предоставляет доступ к смежным элементам (граням и кограням). Используется для построения топологических структур
- **Matrix** — абстракция для работы с разреженными матрицами (например, граничными операторами). Реализации: `SparseMatrix` на основе хеш-таблицы
- **PosHeap** — структура данных для эффективного управления позициями в алгоритмах редукции. Реализации: `BitTreeHeap` (битовая куча) и `SetHeap` (на основе множества)
- **PersistenceDiagram** — вычисляет устойчивые пары (рождение и смерть) на основе фильтрованного комплекса. Реализован в классах `Twist` и `DoubleTwist` через редукцию граничных матриц
- **Solver** — решает задачу нахождения ядра матрицы. Реализации: `SelfAdjointSolver` (спектральное разложение для эрмитовых матриц) и `ShiftSolver` (использует метод Shift and Invert для разложения разреженных матриц)
- **Separator** — решает задачу выделения линейно-независимого вектора из множеств векторов. Реализация `SVDSeparator` использует SVD разложение матриц для этих целей
- **Harmonic** — вычисляет гармонические циклы, используя устойчивые пары. Интегрирует `Solver` и `Separator` для вычислений ядра и выделения линейно-независимых векторов соответсвтенно

4.3 Ключевые алгоритмы

- **Фильтрация**

  Фильтрация Вьеториса-Рипса из облака n точек

  - **FullVR**: Сложность: $O(n^3)$; Память: $O(n^3)$
  - **NewVR**: Пусть допустимые рёбра будут определятся по случайному графу Эрдёша-Реньи ($G(n,p)$, $p \in [0,1]$). Тогда сложность будет равна $O(np^{(d_{max}-1)})$, тогда как в реализации GUDHI сложность при тех же вводных $O(np)$, где n - количество вершин

- **Дерево симплексов**

- **SimplexTree**:
  1. **Add**, **Has**, **GetPos**: Сложность $O(k \log m)$, где k — размер симплекса, m — среднее количество потомков узла
  2. **GetFacetsPos**: Сложность $O(k^2 \log m)$
  3. **GetCofacetsPos (Sparse)**: Сложность $O(q)$, где q - количество узлов на уровне $k+1$
  4. **GetCofacetsPos (Dense)**: Сложность $O(nk)$
- **FullTree**:
  1. **Add**, **Has**, **GetPos**: Сложность $O(1)$
  2. **GetFacetsPos**: Сложность $O(1)$
  3. **GetCofacetsPos**: Сложность $O(n)$, где n - количество вершин

- **Вычисление диаграммы устойчивости**:
  - **Twist**: Сложность $O(n+m)$, где n - количество симплексов, а m - количество ненулевых элементов в матрице
  - **DoubleTwist**: Сложность $O(n+m)$, но константа значительно меньше

- **Вычисление гармонических циклов**:
  - **Фильтрация устойчивых пар**: $O(m \log m)$, где m - количество пар
  - **Построенние ориентированной граничной матрицы**: $O(n)$
  - **Вычисление Лапласиана**: $O(nnz)$, где nnz - количество ненулевых элементов в матрицах B_0 и B_1
  - **Поиск ядра лапласиана**: $O(kn^2)$, где k - число итераций в методе Shift-and-Invert, n - размер матрицы
  - **SVD-разделение**: $O(m^3)$, где m - число столбцов в матрице `basis`
  - **Обновление циклов**: $O(k(e+v))$, где k - число циклов, e - рёбер, v - вершин

  Доминирующие шаги - решение спектральной задачи $O(kn^2)$ и SVD-разделение $O(m^3)$. Худший случай: $O(n^3)$.

4.4 Тестирование

Все классы и функции покрыты юнит и стресс тестами с помощью фреймворка Catch2 [9]. Проведены сравнительные бенчмарки для нахождения наилучших комбинаций из алгоритмов для поиска гармонических циклов.

Добавлены сравнительные бенчмарки с различными имплементациями других библиотек, а также построены их графики и добавлены в репозитории проекта.

A Глоссарий

- Симплициальный комплекс/комплекс - Подмножество 2^V, замкнутое относительно взятия подмножеств. Состоит из симплексов.
- k-симплекс - Симплекс, содержащий $k+1$ вершину. Размерность $\dim \sigma = k$
- i-скелет - Подкомплекс комплекса K содержащий все симплексы размерности $\leq i$
- Полный 2-скелет - Это комплекс, содержащий все возможные рёбра и треугольники (1- и 2-симплексы) между вершинами
- Фильтрация/фильтрованный комплекс/фильтрованный симплициальный комплекс - Упорядоченное семейство вложенных симплициальных комплексов $\{K_f\}_{f \in \mathbb{R}}$
- Граничный оператор - Линейный оператор $\delta_k \colon C_k(K) \to C_{k-1}(K)$, сопоставляющий симплексу сумму его граней с учётом ориентации
- Группа гомологий - Факторгруппа $H_k(K) = Z_k(K)/B_k(K)$, где Z_k - циклы, B_k - границы
- Когомологии — Двойственные к гомологиям структуры, где цепи рассматриваются как функции на симплексах
- Устойчивая пара - Пара симплексов $(\sigma_i,\, \sigma_j)$, где добавление σ_j уничтожает гомологический класс, созданный σ_i
- Гармонический цикл - Элемент ядра комбинаторного оператора Ходжа-Лапласа $L_i = \delta_i^* \delta_i + \delta_{i+1} \delta_{i+1}^*$

Список литературы

[1] Sean Eron Anderson. «Bit Twiddling Hacks». В: (2005). (дата обр. 29.04.2025). URL: http://graphics.stanford.edu/~seander/bithacks.html.

[2] Ulrich Bauer и др. «Phat–persistent homology algorithms toolbox». В: *Journal of symbolic computation* 78 (2017), с. 83—84.

[3] Jean-Daniel Boissonnat и Clément Maria. «The simplex tree: An efficient data structure for general simplicial complexes». В: *Algorithmica* 70 (2014), с. 406—427.

[4] Erik von Brömssen. «Computing persistent homology in parallel with a functional language». В: (2021).

[5] Chao Chen и Michael Kerber. «Persistent homology computation with a twist». В: *Proceedings 27th European workshop on computational geometry*. Т. 11. 2011, с. 197—200.

[6] Unified Acceleration Foundation. *oneTBB*. Вер. 2022.1.0. 2025. URL: https://github.com/uxlfoundation/oneTBB.

[7] Gaël Guennebaud, Benoît Jacob и др. *Eigen v3*. Вер. 3.4.0. 2010. URL: http://eigen.tuxfamily.org.

[8] Hyekyoung Lee и др. «Harmonic holes as the submodules of brain network and network dissimilarity». В: *Computational Topology in Image Context: 7th International Workshop, CTIC 2019, Málaga, Spain, January 24-25, 2019, Proceedings 7*. Springer. 2019, с. 110—122.

[9] Catch Org. *Catch2*. Вер. 3.5.4. 2025. URL: https://github.com/catchorg/Catch2.

[10] Tuyen Pham и Hubert Wagner. «Computing Representatives of Persistent Homology Generators with a Double Twist». В: *arXiv preprint arXiv:2403.04100* (2024).

[11] Yixuan Qiu. *Spectra*. Вер. 1.1.0. 2025. URL: https://github.com/yixuan/spectra.

[12] Antonio Rieser. *A New Construction of the Vietoris-Rips Complex*. 2024. arXiv: 2301.07191 [math.CO]. URL: https://arxiv.org/abs/2301.07191.