Code details

This document provides an overview of main functions and their outputs. Those interested in contributing to the project may find this document useful.

1 The persistent extension method

1.1 Implementation

- The persistent extension method is implemented in 6 different functions:
  - `run_extension_VR_to_VR()`
  - `run_extension_VR_to_VR_bar()`
  - `run_extension_VR_to_W()`
  - `run_extension_VR_to_W_bar()`
  - `run_extension_W_to_VR()`
  - `run_extension_W_to_VR_bar()`
- The functions implement a variation of the cycle-to-bars extension (Algorithm 5 from paper) or a bar-to-bars extension (Algorithm 3 from paper). The ones implementing bar-to-bars extension has the string `_bar` at the end of its function name.
- All implementations are component-wise extensions with $\mathbb{F}_2$ coefficients.
- Due to computation speed issues, the functions do not return the full collection of cycle extensions and bar extensions as described in the paper. Instead, the functions return the baseline and offset cycle extensions and bar extensions. The output of the above functions can then be processed to return the full collection of cycle extensions $E(\tau, Y^\bullet)$ and bar extensions $S(\tau, Y^\bullet)$.
- The algorithm below summarizes the bar-to-bars extension method variation implemented by our functions. The cycle-to-bars extension functions implement a similar variation.

Algorithm 1: Bar-to-bars extension method variation

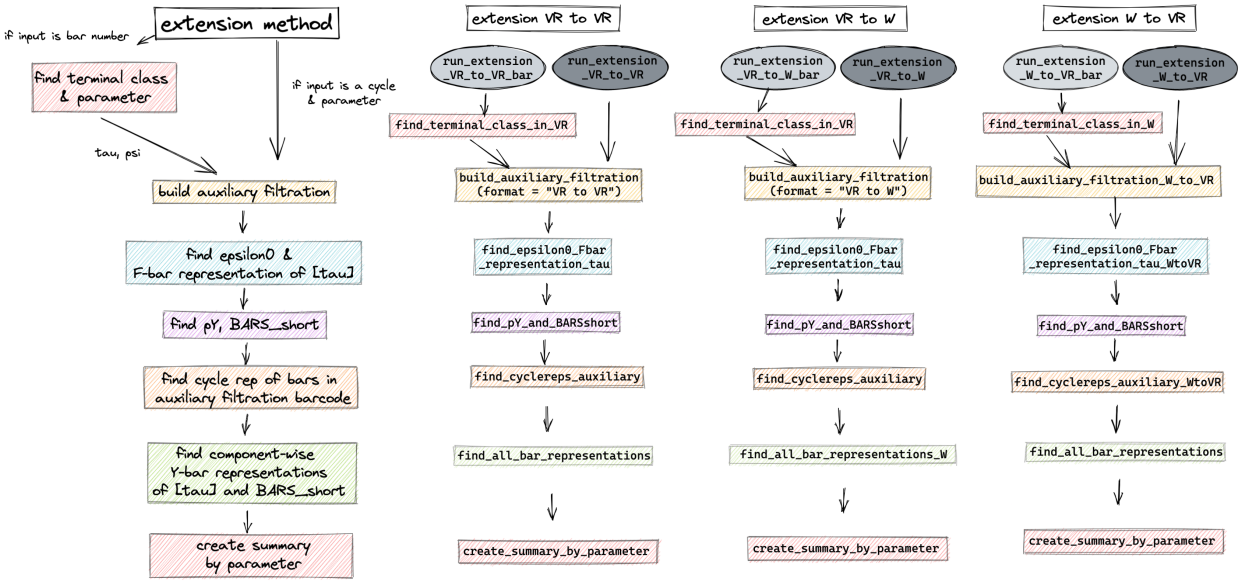
Input:

- filtered simplicial complexes $Z^\bullet$, $Y^\bullet$ on vertex set P,
- a bar $\tau \in \mathrm{BC}_k(Z^\bullet)$

Steps:

1. Fix interval decompositions $\mathcal{B}$ of $P\widetilde{H}_k(Z^\bullet)$ and $\mathcal{D}$ of $P\widetilde{H}_k(Y^\bullet)$.
2. Find parameter $\delta(\tau) - 1$ and cycle $[\tau_*^{\mathcal{B}}]$.
3. Let p_Y be the collection of parameters from Algorithm 1 of paper run with inputs $\delta(\tau) - 1$ and $[\tau_*^{\mathcal{B}}]$.
4. Let $\mathrm{BARS}_\tau^{\mathcal{F}} = \{\rho_1, \ldots, \rho_m\}$ be the set of bars in the $\mathcal{F}$-bar representation.
5. For each $\ell \in p_Y$:
   (a) Find the baseline cycle extension $\mathfrak{E}_\ell^{\text{baseline}} = \Upsilon_\ell(\sum_{i=1}^m [\rho_i^{\mathcal{F},\ell}])$
   (b) Find the offset cycle extension $\mathfrak{E}_\ell^{\text{offset}} = \{\Upsilon_\ell[\rho^{\mathcal{F},\ell}] \mid \rho \in \mathrm{BARS}_{\text{short}}^\ell\}$
   (c) Find the baseline bar extension $B^\ell = \{S_{[t]}^{\mathcal{D}} \mid [t] \in \mathfrak{E}_\ell^{\text{baseline}}\}$.
   (d) Find the offset bar extension $O^\ell = \{S_{[t]}^{\mathcal{D}} \mid [t] \in \mathfrak{E}_\ell^{\text{offset}}\}$.
6. Return $\mathfrak{E}_\ell^{\text{baseline}}$, $\mathfrak{E}_\ell^{\text{offset}}$, B^ℓ, and O^ℓ for $\ell \in p_Y$.

- The following flow chart summarizes the key functions implemented in each step. The left-most column describes the general flow. The remaining columns show the explicit functions called to execute the specific task. Depending on the filtration types, the component functions require slight modifications.



- If you choose to take a closer look at the code, you will notice that there are many functions and variables that keeps track of the various ways in which we describe a simplex (and therefore, a chain). Here are some things to keep in mind:
  - A simplex can either be expressed using some index (the "i"-th simplex) or it can be expressed using its 0-simplices ($s = [v_0, v_1]$). We'll refer to the former as the "index notation" and the latter as the

"simplex notation". Any functions and variables with the t erm `simplex2index` or `index2simplex` will convert one notation to another.

- The vertex ordering used in various functions may differ from the vertex ordering provided by the input. That is, let's say we provide a distance matrix D. We would intuitively think that the 1st row of D corresponds to vertex v_1, 2nd row of D corresponds to vertex v_2, and so on. However, the vertex ordering used in the computations may differ from the order of the rows of D. This is because Eirene permutes the vertices during its computations. We often use specific permutations to go back and forth between Eirene's vertex ordering and the default vertex ordering.
- When building the Witness complex from a cross-distance matrix $D_{P,Q}$, it's possible to provide the maximum parameter at which to build the Witness complex. In such cases, one may end up using only a sub-collection of the vertices provided by the rows of $D_{P,Q}$. In such cases we end up with two distinct ordering of the vertices: The ordering provided by the distance matrix $D_{P,Q}$ and the ordering among the sub-collection of vertices used to build the Witness complex. The former is called the `default_vertex` and the latter is called the `Wepsilon_vertex` in the code.

1.2 Finding the full collection of cycle extensions and bar extensions

- As noted in the previous section, the 6 extension method function returns the **component-wise** cycle and bar extensions. We provide several functions that help the user to find the full collection of cycle extensions ($E(\tau, Y^\bullet)$ from paper) and bar extensions ($S(\tau, Y^\bullet)$ from paper) .
- We first discuss methods for finding the collection of cycle extensions $E(\tau, Y^\bullet)$ and the bar extensions under a **fixed interval decomposition** of $P\widetilde{H}_k(Y^\bullet)$. We'll then discuss methods for finding the full bar extensions under any interval decomposition of $P\widetilde{H}_k(Y^\bullet)$.
- In each section, we present multiple functions that achieve the same goal. The appropriate use of the functions depend on the size of the data – In particular, the number of bars in $\mathrm{BC}_k(Z^\psi \cap Y^\bullet)$ and $\mathrm{BC}_k(Y^\bullet)$.

1.2.1 Finding all cycle extensions and bar extensions under fixed target interval decomposition

- Use the function `return_extension_results_at_parameter()` for an interactive exploration of baseline and offset bar extensions. For each parameter, user can select which offset bar extensions to include in their final view.
- To find all cycle extensions and bar extensions at a specific parameter, run the function `find_CE_BE_at_param()`.
- To find all cycle extensions and bar extensions at all parameters, run `find_CE_BE()`.
  - Let `CE`, `BE` be the outputs of the function.
  - Given a parameter `param`, and some index `i`, `CE[param][i]` is the i-th cycle extension at given parameter
  - Given a parameter `param`, and some index `i`, `BE[param][i]` is the i-th bar extension at given parameter

1.2.2 Finding all alternative bar extensions

- We now discuss methods for finding all alternative bar extensions under possible interval decompositions of $P\widetilde{H}_k(Y^\bullet)$.
- To find the full collection of alternative bar extensions $S(\tau, Y^\bullet) = \{S_{[y]}^{\mathcal{D}\circ L^{-1}} | \ell \in p_Y, [y] \in \mathfrak{E}_\ell, L \in L_Y\}$, run the function `find_alt_BE()`. Note that this is appropriate for data with small barcodes sizes.
- To find the alternative bar extensions at **specific parameters**, run the function `find_alt_BE_at_param()`. Given a parameter ℓ, this method finds $S(\tau, Y^\bullet; \ell) = \{S_{[y]}^{\mathcal{D}\circ L^{-1}} | [y] \in \mathfrak{E}_\ell, L \in L_Y\}$.
- To find alternative bar extensions of a specific bar extension, use the function `find_alternative_bar_extension()`. Given a parameter ℓ and a cycle extension $[y] \in \mathfrak{E}_\ell$, this method returns $\{S_{[y]}^{\mathcal{D}\circ L^{-1}} | L \in L_Y\}$.

1.3 Understanding the output of the functions

The output of the 6 extension method functions is a dictionary with the following key-value pairs.

- `comparison` : Indicates the types of the filtrations under comparison. Values can be "VR to VR" or "VR to W" or "W to VR".
- `C_Z`: Eirene's dictionary output on the "from" filtration $Z^\bullet$.
  - If `comparison` is "VR to VR" then `C_Z` refers to the "from" filtration in which a cycle or bar was selected.
  - If `comparison` is "VR to W", then the dictionary output will have key `C_VR` to refer to the "from" filtration.
  - If `comparison` is "W to VR", then the dictionary output will have key `C_W` to refer to the "from" filtration.
- `C_Y`: Eirene's dictionary output on the "target" filtration $Y^\bullet$.
  - If `comparison` is "VR to VR" then `C_Y` refers to the "target" filtration in which a cycle or bar was selected.
  - If `comparison` is "VR to W", then the dictionary output will have key `C_W` to refer to the "target" filtration.
  - If `comparison` is "W to VR", then the dictionary output will have key `C_VR` to refer to the "target" filtration.
- `dim`: Dimension of interest.
- `selected_cycle`: Input cycle.
  - If the output is obtained from a bar-to-bars extension, then `extension["selected_cycle"]` is a simply a copy of the input cycle.
  - If the output is obtained from a cycle-to-bars extension, then `extension["selected_cycle"]` is the cycle representative of the input bar.
  - In the code, this cycle is often referred to as `tau`.
- `C_auxiliary_filtration`: Eirene's dictionary output on filtration $Z^\psi \cap Y^\bullet$
- `aux_filt_cycle_rep`: Cycle representatives of bars in `C_auxiliary_filtration`
- `p_Y`: Collection of parameters p_Y
- `epsilon_0`: Minimum parameter value in p_Y
- `nontrivial_pY`: A sub-collection of parameters in p_Y.

  $$\{\varepsilon \in p_Y \mid \text{ there exists a cycle in } Z^\psi \cap Y^\bullet \text{ that is born at } \varepsilon \text{ withx a non-trivial cycle extension}\}$$

  To find all cycle extensions and bar extensions, it suffices to consider only the parameters in `nontrivial_pY` (instead of `p_Y`, which is quite large).
- `nontrivial_pY_dict`: A dictionary of parameter index and values.
- `Ybar_rep_tau`: Given the `selected_cycle` $[\tau]$, let $\{\rho_1, \ldots, \rho_m\}$ be the $\mathcal{F}$-bar representations of $[\tau]$ (Algorithm 1 step (2)(a)). Let $[y] = \Upsilon_{\varepsilon_0}([\rho_1^{\mathcal{F},\varepsilon_0}] + \cdots + [\rho_m^{\mathcal{F},\varepsilon_0}]) \in \mathfrak{E}_{\varepsilon_0}$. `Ybar_rep_tau` is the $\mathcal{D}$-bar representation of $[y]$ ($S^{\mathcal{D}}_{[y]}$).
- `Ybar_rep_short_epsilon0`: A dictionary of the short bars $\rho \in \mathrm{BARS}^{\varepsilon_0}_{\mathsf{short}}$ and their $\mathcal{D}$-bar representations. Given a short bar `rho` representing $\rho \in \mathrm{BARS}^{\varepsilon_0}_{\mathsf{short}}$, let $[y] = \Upsilon_{\varepsilon_0}([\rho_1^{\mathcal{F},\varepsilon_0}])$.

  Then, `extension["Ybar_rep_short_epsilon0"][rho]` is the $\mathcal{D}$-bar representation $S^{\mathcal{D}}_{[y]}$.

- `Ybar_rep_short`: Dictionary of the short bars $\rho \in \text{BARS}_{\text{short}} \setminus \text{BARS}_{\text{short}}^{\varepsilon_0}$ and their $\mathcal{D}$-bar representations.
- `cycle_extensions`: Dictionary summarizing cycle extensions of given input at various parameters. Given a parameter `l` (or ℓ) in `extension["nontrivial_pY"]`,
  - `extension["cycle_extensions"][l]["baseline"]` is the baseline cycle extension at parameter `l`. (See section 4.4 of paper). Given the `selected_cycle` $[\tau]$, let $\{\rho_1, \ldots, \rho_m\}$ be the $\mathcal{F}$-bar representations of $[\tau]$. Let $[y] = \Upsilon_\ell([\rho_1^{\mathcal{F},\ell}] + \cdots + [\rho_m^{\mathcal{F},\ell}]) \in \mathfrak{E}_\ell$. `extension["cycle_extensions"][l]["baseline"]` returns $[y]$ if $[y]$ is nontrivial at parameter l. Otherwise returns empty array.
  - `extension["cycle_extensions"][l]["offset"]` is a dictionary whose keys are $\text{BARS}_{\text{short}}^{l}$ (Algorithm 1, step 3(b)(i)) and whose values are the corresponding offset cycle extensions.
    - Let `j` be a bar in $\text{BARS}_{\text{short}}^{\ell}$. (That is, $\rho_j \in \text{BARS}_{\text{short}}^{\ell}$)
    - `extension["cycle_extensions"][l]["offset"][j]` is $\Upsilon_\ell[\mathcal{F}^\ell(\vec{e}_{\rho_j}^{\,\ell})]$.
    - See section 4.4 of paper for notations.
- `bar_extensions`: Dictionary summarizing bar extensions of given input at various parameters. Given a parameter `l` (or ℓ) in `extension["nontrivial_pY"]`,
  - `extension["bar_extensions"][l]["baseline"]` is the baseline bar extension at parameter `l`. (See section 4.4 of paper). Given the `selected_cycle` $[\tau]$, let $\{\rho_1, \ldots, \rho_m\}$ be the $\mathcal{F}$-bar representations of $[\tau]$. Let $[y] = \Upsilon_\ell([\rho_1^{\mathcal{F},\ell}] + \cdots + [\rho_m^{\mathcal{F},\ell}]) \in \mathfrak{E}_\ell$. `extension["bar_extensions"][l]["baseline"]` returns the $\mathcal{D}$-bar representation of $[y]$: $S_{[y]}^{\mathcal{D}}$.
  - `extension["bar_extensions"][l]["offset"]` is a dictionary whose keys are $\text{BARS}_{\text{short}}^{\ell}$ (Algorithm 1, step 3(b)(i)) and whose values are the corresponding offset bar extensions.
    - Let `j` be a bar in $\text{BARS}_{\text{short}}^{\ell}$. (That is, $\rho_j \in \text{BARS}_{\text{short}}^{\ell}$)
    - Recall the cycle extension: `extension["cycle_extensions"][l]["offset"][j]` is $\Upsilon_\ell[\mathcal{F}^\ell(\vec{e}_{\rho_j}^{\,\ell})]$. We'll call this cycle $[y]$.
    - `extension["bar_extensions"][l]["offset"][j]` is the offset bar extension for bar `j`.
    - That is, it returns the $\mathcal{D}$-bar representation of $[y]$: $S_{[y]}^{\mathcal{D}}$.
  - Example:
    - `extension["bar_extensions"][0.123]["baseline"]` = [1,2,3] : At parameter 0.123, the baseline bar extension is [1,2,3].
    - `extension["bar_extensions"][0.123]["offset"][5]` = [4,5]: At parameter 0.123, the offset bar extension corresponding to bar 5 of $\text{BAR}_{\text{short}}$ is [4,5]

2 The analogous bars method

2.1 Implementation

- As discussed in the paper, there are two types of analogous bars method: the similarity-centric analogous bars method and the feature-centric analogous bars method
- **similarity-centric analogous bars**
  - Given two point clouds `P` and `Q`, user selects a bar of interest in `barcode(W(P,Q))`. (The Witness barcode with `P` as landmark and `Q` as witness)
  - The function returns the output of implementing bar-to-bars extension to `barcode(VR(P))` and `barcode(VR(Q))`.
- **feature-centric analogous bars**
  - There is no one function that implements this method because it requires user interaction.

- One should first run the bar-to-cycle extension method from `VR(P)` to `W(P,Q)`. We run the function `run_extension_VR_to_W_bar()`.
- User then selects a particular cycle extension of interest, say `cycle_W_PQ`.
-
- One should then run a cycle-to-bar extension method from `W(Q,P)` to `VR(Q)` by calling the function `run_extension_W_to_VR()`.

2.2 Understanding the output

Both the similarity-centric analogous bars and feature-centric analogous bars method return the dictionary outputs that result from running the appropriate extension methods. We thus direct the reader to sections 1.2 and 1.3 of this documentation.

3 To-do list for future versions

- Generalize the persistent extension and analogous bars method from dimension 1 to higher dimensions. In particular, for the feature-centric analogous bars method, we'll have to implement Dowker's Theorem in higher dimensions.
- Address speed and memory issues:
  - Creating the Witness complex and running the extension method can take a long time. Is there a way to speed up this process? Maybe via distributed computation?
  - When creating the Witness complex, we create a dictionary to keep track of all simplices and their indices. Is there a way to avoid creating dictionaries while keeping track of the simplices and their indices?
  - Current implementation for exploring alternative bar extensions runs into memory issues if the auxiliary barcode or the target barcode has too many bars. Can we resolve this issue?
- Implement "lazy" extension. Before running any of the persistent extension or analogous bars method, assume that the user selects bars of "significant" length as bars that can participate in the bar extension. The resulting bar extensions must be a subset of the pre-selected bars. Implement the lazy extension and analogous bars method.