RIVET Documentation

Release 1.0

The RIVET Developers

Oct 09, 2018

Contents:

CHAPTER 1

About

RIVET is a tool for topological data analysis, and more specifically, for the visualization and analysis of two-parameter persistent homology. RIVET was initially designed with interactive visualization foremost in mind, and this continues to be a central focus. But RIVET also provides functionality for two-parameter persistence computation that should be useful for other purposes.

Many of the mathematical and algorithmic ideas behind RIVET are explained in detail in the paper Interactive Visualization of 2-D Persistence Modules. Additional papers describing other aspects of RIVET are in preparation.

RIVET is written in C++, and depends on the Qt, Boost, and MessagePack libraries. In addition, RIVET now incorporates some modified code from PHAT.

A python API for RIVET called pyrivet is available in a separate repository. pyrivet has separate documentation, and is not discussed elsewhere in this document.

1.1 Overview

The basic idea of 2-parameter persistent homology is simple: Given a data set (for example, a point cloud in \(\mathbb R^n\)), one constructs a 2-parameter family of simplicial complexes called a *bifiltration* whose topological structure captures some geometric structure of interest about the data. For example, the bifiltration may encode information about the presence of clusters, holes, or tendrils in the data. Applying simplicial homology to the bifiltration gives a diagram of vector spaces called a *bipersistence module*, which algebraically encodes information about the topological structure of the bifiltration. In contrast to the 1-parameter case, bipersistence modules can have very delicate algebraic structure. As result, there is (in a sense that can be made precise) no good definition of a barcode for bipersistence modules.

Nevertheless, one can define invariants of bipersistence modules that capture aspects of their structure relevant for data analysis. RIVET computes and visualizes three such kinds of invariants, the *Hilbert function*, the *bigraded Betti numbers*, and the *fibered barcode.*

The fibered barcode is a parameterized family of barcodes obtained by restricting a bipersistence module to various lines in parameter space. A key feature of RIVET is that it computes a data structure called the *augmented arrangement*, on which fast queries of these barcodes can be performed. These queries are used by RIVET's GUI to provide an interactive visualization of the fibered barcode.

RIVET also computes *minimal presentations* of bipersistence modules. These are specifications of the full algebraic structure of a persistence module which are as small as possible, in a certain sense. Those who wish to study invariants of 2-parameter persistent homology not computed by RIVET may find it useful to take the minimal presentations output by RIVET as a starting point. In practice, when working with real data, these presentations are often surprisingly small.

1.2 Contributors

RIVET project was founded by Michael Lesnick and Matthew Wright, who designed and developed a first version of RIVET in 2013-2015. Matthew was the sole author of the code during this time. Since 2016, several others have made valuable contributions to RIVET, some of which will be incorporated into later releases.

Here is a list of contributors, with a brief, incomplete summary of contributions:

- Madkour Abdel-Rahman : Error handling
- Bryn Keller : Parallel-friendly code organization, command line interactivity, APIs, software design leadership.
- Michael Lesnick : Design, performance optimizations, computation of minimal presentations
- Phil Nadolny : Error handling, code for constructing path through dual graph
- Matthew Wright : Design, primary developer
- Simon Segert : Major improvements to the GUI, extensions of RIVET
- David Turner : Parallel minimization of a presentation
- Alex Yu : Extensions of RIVET
- Roy Zhao : Multicritical bifiltrations and Degree-Rips bifiltrations, performance optimizations

The RIVET development team can be reached by email at info@rivet.online.

1.3 Contributing

We welcome your contribution! Code, documentation, unit tests, interesting sample data files are all welcome!

Before submitting your branch for review, please run the following from the top level RIVET folder you cloned from Github:

```
clang-format -i **/*.cpp **/*.h
```

This will format the source code using the project's established source code standards (these are captured in the `.clang-format` file in the project root directory).

1.4 Issues

A full list of bugs and todos can be found on the Github issue tracker. Please feel free to add any bugs/issues you discover, if not already listed.

1.5 Acknowledgments

The RIVET project is supported in part by the National Science Foundation under grant DMS-1606967. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Additional support has been been provided by the Institute for Mathematics and its Applications, Columbia University, Princeton Neuroscience Institute, St. Olaf College, and the NIH (grant U54-CA193313-01).

Special thanks to Jon Cohen at Princeton for his support of the RIVET project. Thanks also to Ulrich Bauer for many enlightening conversations about computation of 1-parameter persistent homology, which have influenced our thinking about 2-parameter persistence.

1.6 License

1.7 Citation

For convenient citation of RIVET in your own work, we provide the following BibTex entry:

```
@software{rivet,
        author = {{The RIVET Developers}},
        title = {RIVET},
        url = {http://rivet.online},
        version = {1.0},
        year = {2018}
}
```

1.8 Documentation Todos

Formatting todos:

- A lot of displayed math is not displaying properly on the .pdf provided by read the docs.
- The caption for the figure on the "Computation Pipeline" page does not display properly in the downloaded .html. MW: It seems that the downloaded HTML is using a slightly different style sheet than the online version. However, nearly everyone will use the online version rather than downloading a zip archive containing all of the HTML and supporting files.

Content Todos:

- The descussion of the Persistence Diagram Window has now been updated to reflect the recent improvements to the visualization. However, some details about how the persistence diagram is drawn appear in the appendix of the arXiv paper, but not in the documentation. For example, the way we handle normalization when "fit to window" is selected is only discussed in the appendix. I suggest that all details be given in the documentation and removed from the paper.
- I suggest to not print out xi_0, xi_1, and xi_2 when —Betti is called.
- The example could use some polish. More examples are desirable.

Minor Todos:

- There is a formatting problem in the “cases” environment used in the definition of a free module.
- It’s a small thing, but the .png of the the file input dialog looks a little off center.
- The FAQ section has been removed from the documentation, but the .rst file is still in the repository in case we want to reintroduce this.

CHAPTER 2

Installation

RIVET is available at https://github.com/rivetTDA/rivet.

2.1 Requirements

Before starting to build RIVET, you will need to have the following installed:

- A C++ compiler (g++ or clang are what we use)
- CMake
- Qt 5
- Boost (version 1.58 or newer)

Below we give step-by-step instructions for installing these required dependencies and building RIVET on Ubuntu and Mac OS X. Building RIVET on Windows is not yet supported, but it is possible to build RIVET using the Bash shell on Windows 10.

2.2 Building On Ubuntu

2.2.1 Installing Dependencies

On Ubuntu, installation of dependencies should be relatively simple:

```
sudo apt-get update
sudo apt-get install cmake qt5-default qt5-qmake qtbase5-dev-tools libboost-all-dev
```

2.2.2 Building RIVET

After cloning to `$RIVET_DIR`:

```
cd $RIVET_DIR
mkdir build
cd build
cmake ..
make
cd ..
qmake
make
```

You may see compiler warnings during either of the `make` executions. These can safely be ignored.

After this, you will have two executables built: the viewer (rivet_GUI), and the computation engine (rivet_console).

It is then necessary to move or symlink the console into the same folder where the viewer was built. On Ubuntu and most other systems:

```
ln -s build/rivet_console
```

In the future, all these steps will be automated so that a single cmake build will create both executables, and put everything in the right place.

2.3 Building On Mac OS X

2.3.1 Installing Dependencies

First, ensure you have the XCode Command Line Tools installed by running:

```
# only needed if you've never run it before, (running it again doesn't hurt anything)
# installs XCode Command Line Tools
xcode-select --install
```

If a popup window appears, click the "Install" button, and accept the license agreement.

Next, install XCode from the App Store, if you've not done so already. You will also need accept the license agreement for XCode, which you can do from the command line by running:

```
sudo xcodebuild -license
```

To install the remaining packages, we recommend using the package manager [Homebrew](http://brew.sh/). To install Homebrew:

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
↪master/install)"
```

Now install cmake, qt5, and boost:

```
brew install cmake qt5 boost
```

2.3.2 Building RIVET

Please note that part of the build requires the use of `qmake`, and you might very well have a version of `qmake` in your `PATH` that is not for Qt 5, but for an older version. In the steps below, we assume `$QT_BASE` is the installation folder for Qt 5.

As of the time of writing, `brew` installs `qmake` in a version-specific folder under `/usr/local/Cellar/qt5/[my_version_#]/bin`, and does not add it to your `PATH`. You can find the folder where qt5 is installed using this command:

```
brew info qt5 | grep Cellar | cut -d' ' -f1
```

In fact, let's store that in a variable so we can use it below:

```
export QT_BASE=`brew info qt5 | grep Cellar | cut -d' ' -f1`
```

In order to ensure that `qmake` can find where Boost is installed, add the following lines to the bottom of the file RIVET.pro, changing the paths in the last three lines, if necessary, to match the location and version of your copy of Boost.

```
CONFIG += c++11
QMAKE_CFLAGS += -std=c++11 -stdlib=libc++ -mmacosx-version-min=10.9
QMAKE_CXXFLAGS += -std=c++11 -stdlib=libc++ -mmacosx-version-min=10.9

LIBS += -L"/usr/local/Cellar/boost/1.63.0/lib"
INCLUDEPATH += "/usr/local/Cellar/boost/1.63.0/include"

LIBS += -L"/usr/local/Cellar/boost/1.63.0/lib" -lboost_random
```

After cloning to `$RIVET_DIR`:

```
cd $RIVET_DIR
mkdir build
cd build
cmake ..
make
cd ..
$QT_BASE/bin/qmake
make
```

You may see compiler warnings during either of the `make` executions. These can safely be ignored.

After this, you will have two executables built: the viewer (rivet_GUI.app), and the computation engine (rivet_console).

It is then necessary to move or symlink the console into the same folder where the viewer was built:

```
cd rivet_GUI.app/Contents/MacOS
ln -s ../../../build/rivet_console
```

In the future, all these steps will be automated so that a single `cmake` build will create both executables, and put everything in the right place.

2.3.3 Troubleshooting

Our experience has been that if Homebrew is installed before XCode, then running qmake during the build process returns an error:

```
Project ERROR: Could not resolve SDK Path for 'macosx'
```

To solve the problem, try running:

```
sudo xcode-select --switch /Applications/Xcode.app/Contents/Developer
```

2.4 Building in the Bash Shell on Windows 10

First, ensure that you have the Windows 10 Creators Update. Then activate the Windows 10 Bash Shell. This will provide a Bash shell with Ubuntu 16.04 inside of Windows 10.

Open the Bash shell and install dependencies. Use the following command to install cmake, a compiler, and Qt5:

```
sudo apt-get update
sudo apt-get install cmake build-essential qt5-default qt5-qmake qtbase5-dev-tools␣
↪libboost-all-dev
```

In order to use the RIVET viewer, you must install an X server such as Xming.

It is probably also necessary to set an environment variable, as follows:

```
export DISPLAY=:0
```

This environment variable will be reset when you close the Bash shell. To avoid having to run the line above when you reopen the shell, add this line to the end of the file `~/.bashrc`.

You are now ready to build RIVET. Follow the instructions in the section *Building On Ubuntu*.

CHAPTER 3

Mathematical Preliminaries

To prepare for a detailed explaination of what RIVET can do and how it is used, we review some basic mathematical notions and establish some terminology.

To start, we define a partial order on \(\mathbb R^2\) by taking \(a \leq b\) if and only if \(a_1 \leq b_1\) and \(a_2 \leq b_2\).

3.1 Bifiltrations

A *bifiltration* \(F\) is a collection of finite simplicial complexes indexed by \(\mathbb R^2\) such that \(F_a\subset F_b\) whenever \(a\leq b\). In the computational setting, the bifiltrations \(F\) we encounter are always *essentially finite*. This finiteness condition can be specified succinctly in the language of category theory: \(F\) is essentially finite if \(F\) is a left Kan extension of a diagram indexed by a finite grid in \(\mathbb R^2\). (See the RIVET paper for a more elementary definition.) Such \(F\) can be specified by a single simplicial complex \(S\) (the colimit of \(F\)) together with a collection of incomparable points \(\mathrm{births}(\sigma)\subset\mathbb R^2\) for each simplex \(\sigma\in S\), specifying the bigrades at which \(\sigma\) is born. If \(\mathrm{births}(\sigma)\) contains one element for each \(\sigma\in S\), then we say \(F\) is *one-critical*. Otherwise, we say \(F\) is *multi-critical*.

We next introduce two contructions of bifiltrations from data.

3.2 Function-Rips Bifiltraiton

For \(P\) a finite metric space and \(r\geq 0\), let \(N(P)_r\) denote the \(r\)-neighborhood graph of \(P\), i.e., the vertex set of \(N(P)_r\) is \(P\), and edge \([i,j]\in N(P)_r\) if and only if \(d(i,j)\leq r\). If \(r<0\), we define \(N(P):=\emptyset.\) We define the *Vietoris-Rips complex* \(R(P)_r\) to be the clique complex on \(N(P)_r\), i.e. the largest simplical complex with 1-skeleton \(N(P)_r\).

Given a finite metric space \(P\) and any function \(\gamma:P\to \mathbb R\), we define the *function-Rips* bifiltration \(FR(\gamma)\) as follows: \[FR(\gamma)_{a,b}:=R(\gamma^{-1}(-\infty,a])_b.\] \(FR(\gamma)\) is always 1-critical.

\(\gamma\) is often chosen to be a density estimate on \(P\). Another common choice is to take \(\gamma\) to be a coeccentricity function on \(P\), e.g., \(\gamma(x):= \sum_{y\in P} d(x,y)\).

3.3 Degree-Rips Bifiltration

For \(d\in \mathbb R\), let \(N(P)_{d,r}\) be the subgraph of \(N(P)_r\) obtained by removing all vertices of degree less than \(d\). We define the *degree-Rips bifiltration* \(DR(P)\) by taking \(DR(P)_{d,r}:=N(P)_{d,r}.\) (Note that this is in fact a bifiltration indexed by \(\mathbb R^{\mathrm{op}}\times \mathbb R\), where \(\mathbb R^{\mathrm{op}}\) denotes the opposite poset of \(\mathbb R\); that is, \(DR(P)_{a,b}\subset DR(P)_{a',b'}\) whenever \(a\geq a'\) and \(b\leq b'\).). If \(P\) has more than one point, then \(DR(P)\) is multi-critical.

3.4 Bipersistence Modules

Let us fix a field \(K\). A *bipersistence module* (also called a 2-D persistence module or 2-parameter persistence module in the literature) \(M\) is a diagram of \(K\)-vector spaces indexed by \(\mathbb R^2\). That is, \(M\) is a collection of vector spaces \(\{M_a\}_{a\in \mathbb{R^2}}\), together with a collection of linear maps \[\{M_{a,b}:M_a\to M_b\}_{a\leq b}\] such that \(M_{a,a}=\mathrm{Id}_{M_a}\) and \(M_{b,c}\circ M_{a,b}=M_{a,c}\) for all \(a \leq b\leq c\).

A *morphism* \(f:M\to N\) of bipersistence modules is a collection of maps \[\{f_a:M_a\to N_a\}_{a\in \mathbb R^2}\]such that \[f_b\circ M_{a,b}= N_{a,b} \circ f_a\] for all \(a\leq b\in \mathbb R^2\). This definition of morphism gives the bipersistence modules the structure of an abelian category; thanks in part to this, many usual constructions for modules from abstract algebra have analogues for bipersistence modules. In particular, direct sums and quotients are well defined.

3.5 Free Persistence Modules

For \(c \in \mathbb R^2\), define the bipersistence module \(\mathcal I^c\) by \[\mathcal I^c_a= \begin{cases} K &\mathrm{if }\ a\geq c,\ 0 & \mathrm{otherwise.} \end{cases} \qquad \mathcal I^c_{a,b}= \begin{cases} \mathrm{Id}_K &\mathrm{if }\ a\geq c,\ 0 & \mathrm{otherwise.} \end{cases} \] Note that the support of \(\mathcal I^a\) is the closed upper quadrant in \(\mathbb R^2\) with lower left corner at \(a\).

A *free bipersistence module* is one isomorphic to \[\oplus_{c\in \mathcal B}\ \mathcal I^c\] for some multiset \(\mathcal B\) of points in \(\mathbb R^2\). There is a natural definition of basis for free modules, generalizing the definition of bases for vector spaces in linear algebra. In close analogy with linear algebra, a morphism \(f:M\to N\) of finitely generated free modules can be represented by a matrix, with respect to a choice of ordered bases for \(M\) and \(N\). Thus, to encode the isomorphism type of \(f\), it enough to store a matrix, together with a bigrade label for each row and each column of the matrix; the labels specify \(M\) and \(N\) up to isomorphism.

3.6 Presentations

A *presentation* of a bipersistence module \(M\) is a map \(f:F\to G\) such that \(M\cong G/\mathrm{im}\ f\). We say \(M\) is finitely presented if \(F\) and \(G\) can be chosen to be finitely generated. If \(M\) is finitely presented then, up to isomorphism, there exists a presentation \(f:F\to G\) such that both \(F\) and \(G\) are minimial, i.e., for any other presentation \(f':F'\to G'\), \(F\) is a summand of \(F'\) and \(G\) is a summand of \(G'\). We call such a presentation *minimal*. Minimal presentations are unique up to isomorphism, but importantly, their matrix representations are non-unique.

3.7 FIReps (Short Chain Complexes of Free Modules)

We define a *FIRep* to be chain complex of free bipersistence modules of length 3. Explicitly, then, an firep is a sequence of free bipersistence modules \[C_2 \xrightarrow{f} C_1 \xrightarrow{g} C_0. \] such that \(g\circ f=0\). Associated to an firep is a unique homology module \(\ker g/\mathrm{im}\ f\). A presentation of a bipersistence module can be thought of as a special case of an FIRep, where the last module is trivial.

3.8 Homology of a Bifiltration

Applying \(i^{\mathrm{th}}\) simplicial homology with coefficients in \(K\) to each simplicial complex and each inclusion map in a bifiltration \(F\) yields a bipersistence module \(H_i(F)\). If \(F\) is essentially finite, then \(H_i(F)\) is finitely presented.

\(H_i(F)\) is in fact the \(i^{\mathrm{th}}\) homology module of a chain complex \(C(F)\) of bipersistence modules whose value at each point in \(a\in \mathbb R^2\) is the simplical chain complex of \(F_a\). If \(F\) is one-critical, each module of \(C(F)\) is free. In general, \(C(F)\) needn't be free, but given the portion of \(C(F)\) at indexes \(i-1,\) \(i\), and \(i+1\), it is easy to construct an FIRep whose homology is \(H_i(F)\); this is an observation of Chacholski et al.

3.9 Invariants of a Bipersistence Module

As mentioned above, RIVET computes and visualizes three simple invariants of a bipersistence module \(M\):

- The *fibered barcode*, i.e., the function sending each affine line \(L\subset \mathbb R^2\) with non-negative slope to the barcode \(\mathcal B(M^L)\), where \(M^L\) denotes the restriction of \(M\) along \(L\).
- The *Hilbert function*, i.e., the function \(\mathbb R^2\to \mathbb N\) which sends \(a\) to \(\dim M_a\).
- The *bigraded Betti numbers* \(\xi_i^M\). These are functions \(\mathbb{R}^2 \to \mathbb{N}\) that, respectively, count the number of births, deaths, and "relations amongst deaths" at each bigrade. Formally, given \(r \in \mathbb{R}^2\) and a minimal free resolution $$0 \to F^2\to F^1\to F^0$$ for \(M\), \(\xi_i^M(r)\) is the number of elements at bigrade \(r\) in a basis for \(F^i\).

3.10 Coarsening a Persistence Module

Given a finitely presented bipersistence module \(M\), we can *coarsen* \(M\) to obtain an algebraically simpler module carrying approximately the same persistence information as \(M\). As we will describe it here, the coarsening operation depends on a choice of finite grid \(G\subset\mathbb R^2\), such that \(G\) contains some element ordered after all points in the support of the Betti numbers of \(M\). The coarsened module, denoted \(M^G\), is defined by taking \(M^G_a:= M_g\), where \(g\in G\) is the minimum grid element such that \(a\leq g\). The internal maps of \(M^G\) are induced by those of \(M\) in the obvious way.

CHAPTER 4

Computation Pipeline

The following figure illustrates RIVET's pipeline for working with the 2-parameter persistent homology of data.

We now explain this pipeline:

RIVET can accept as input a data set, a bifiltration, or an firep. RIVET always works with a single homology degree at a time; when giving data or a bifiltration as input, specifies the degree of homology to consider.

RIVET accepts data in the form of either a point cloud in \(\mathbb R^n\), or a finite metric space (represented as distance matrix). Optionally, a function on each point can also be given. If a function is given, RIVET computes a function-Rips bifiltration. If no function is given, it computes the degree-Rips bifiltration.

Given a bifiltration \(F\), RIVET constructs an FIrep for \(H_j(F)\) in the specified degree \(j\). If \(F\) is multi-critical, RIVET uses the trick of Chacholski et al. LINK to obtain the FIrep.

Given an FIrep, RIVET computes a minimal presentation of its homology module. This computation also yields the Hilbert function of the module with almost no extra work. The 0th and 1st bigraded Betti numbers of a bipersistence module \(M\) can be read directly off of the minimal presentation. Given these and the Hilbert function, a simple formula yields the 2nd bigraded Betti numbers as well.

RIVET uses the minimal presentation of \(M\) to compute the *augmented arrangement* of \(M\). This is a line arrangement in the right half plane, together with a barcode at each face of the line arrangement. The augmented arrangement is used to perform fast queries of the fibered barcode of \(M\).

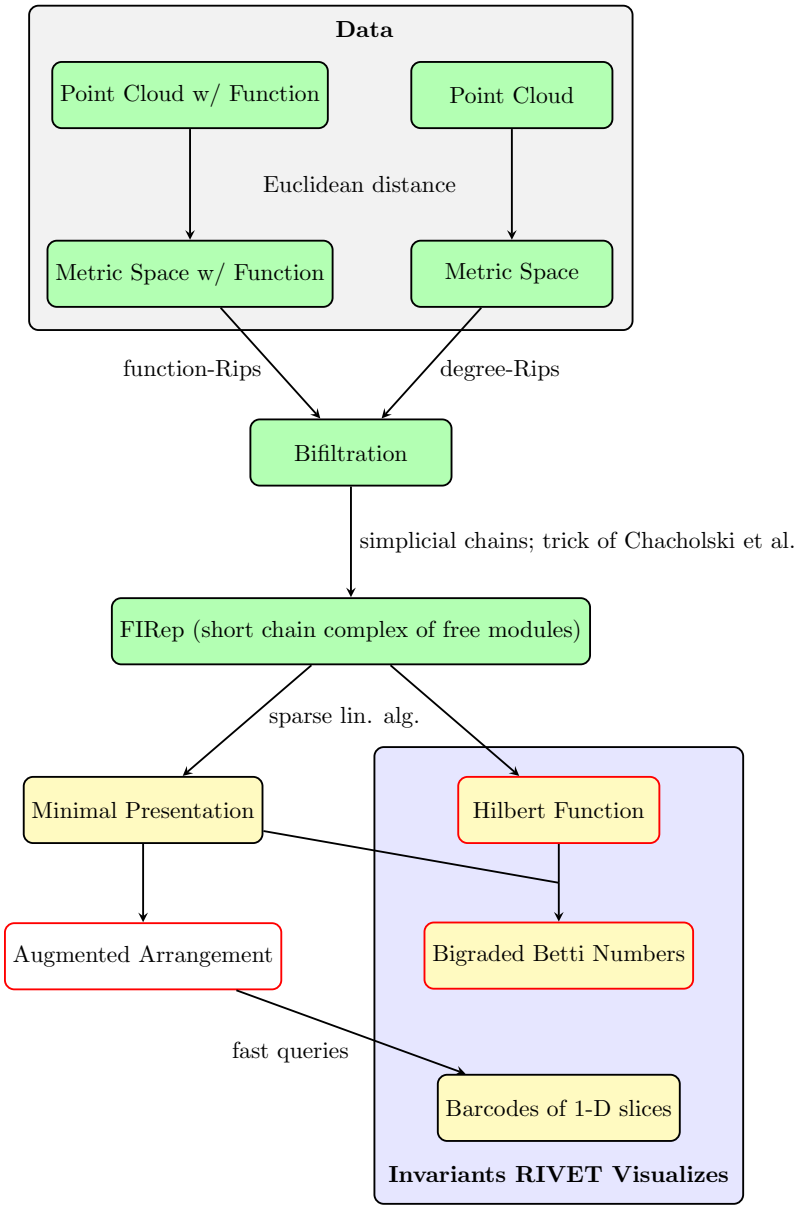


Fig. 1: The RIVET pipeline. Green items can be input directly to RIVET via a file. Yellow items can be printed to the console. Items with red boundary can be saved in a *module invariants file*, which serves as input to RIVET's visualization.

CHAPTER 5

Running RIVET

The RIVET software consists of two separate but closely related executables: **rivet_console**, a command-line program, and **rivet_GUI**, a GUI application. **rivet_console** is the computational engine of RIVET; it implements the computation pipeline described in the previous section. **rivet_GUI** is responsible for RIVET's visualizations.

5.1 rivet_console

rivet_console has three main functions:

- Given an *input data file* in one of the formats described in the *Input Data* section of this documentation, **rivet_console** can compute a file called the *module invariants (MI) file*. The MI file stores the Hilbert function, bigraded Betti numbers, and augmented arrangment of a persistent homology module of the input data. The MI file is used by the RIVET visualization, and also for the following:
- Given an MI file of a bipersistence module \(M\) and a second file, the *line file*, specifying a list of lines, **rivet_console** prints the barcodes of the 1-D slices of each line to the console. The computations are performed using fast queries of the augmented arrangment of \(M\).
- Given a *raw data* file as input, **rivet_console** can print The Hilbert function and Bigraded Betti numbers of a persistent homology module of the input data. It can also print a minimal presentation of the module.

In what follows we explain in more detail how to use **rivet_console**. The syntax for running **rivet_console** is also described in the executable's help information, which can be accessed via the command:

```
rivet_console (-h | --help)
```

The help file also describes some additional technical functionality of **rivet_console** that we will not discuss here.

5.1.1 Computation of a Module Invariants File

Here the basic syntax for computing a module invariants file:

```
rivet_console <input> <output> [-H <dimension>] [-x <xbins>] [-y <ybins>]
```

- <input> is an input data file;
- <output> is the name of the module invariants file to be computed.
- [-H <dimension>] is the dimension of homology to compute [default: 0, ignored if the input file is of firep type]
- [-x <xbins>] and [-y <ybins>] specifies the dimension of a grid used for coarsening. The grid spacing is taken to be uniform in each dimension. A value of 0 means no coarsening is done at all in that coordinate direction. This is the default. However, to control the size of the augmented arrangment, most computations of a MIF should use some coarsening of the module.

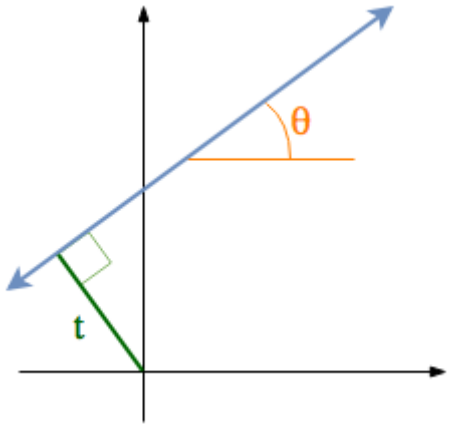
Other (technical) command line options for computation of a MI file are given in the **rivet_console** help.

5.1.2 Computing Barcodes of 1-D Slices

Here is the basic syntax for computing the barcodes of 1-D slices of a bipersistence module, given an MI file as input:

```
rivet_console <module_invariants_file> --barcodes <line_file>
```

<line_file> is a file specifying a list of affine lines in \(\mathbb R^2\) with non-negative slope. Each line is specified by its *angle* and *offset* parameters. The following diagram shows these parameters for a particular line, with *angle* denoted \(\theta\) and *offset* denoted \(t\).



As the diagram indicates, \(\theta\) is the angle between the line and the horizontal axis in degrees (0 to 90). The offset parameter \(t\) is the *signed* distance from the line to the origin, which is positive if the line passes above/left of the origin and negative otherwise. This choice of parameters makes it possible to specify any line of nonnegative slope, including vertical lines.

The following gives a sample line file:

```
#A line that starts with a # character will be ignored, as will blank lines
23 -0.22
67 1.88
10 0.92
#100 0.92   <-- will error if uncommented, 100 > 90
```

For each line specified in <line_file>, **rivet_console** will print barcode information as a single line of text, beginning by repeating the query parameters. For example, output corresponding to the sample line file above might be:

```
23 -0.22: 88.1838 inf x1, 88.1838 91.2549 x5, 88.1838 89.7194 x12
67 0.88: 23.3613 inf x1
10 0.92: 11.9947 inf x1, 11.9947 19.9461 x2, 11.9947 16.4909 x1, 11.9947 13.0357 x4
```

The barcodes are given with respect to an isometric parameterization of the query line that takes zero to be the intersection of the query line with the nonnegative portions of the coordinate axes; there is a unique such intersection point except if the query line is one of the coordinate axes, in which case we take zero to be origin.

Furthermore, barcodes are returned as multisets of intervals. For example, in the sample output above, `88.1838 inf x1` indicates a single interval \([88.1838, \infty)\).

5.1.3 Printing a Minimal Presentation

The basic syntax for computing a minimal presentation of a bipersistence module is the following:

```
rivet_console <input_file> --minpres [-H <dimension>] [-x <xbins>] [-y <ybins>]
```

- <input> is an input data file;
- The options for choosing homology dimensions and coarsening parameters behave exactly as for the computation of the MI file.

The following example shows the output format for the minimal presentation:

```
x-grades
3
7/2
4

y-grades
0
1
2

MINIMAL PRESENTATION:
Number of rows:2
Row bigrades:
| (1,0) (0,1) |
Number of columns:3
Column bigrades:
| (1,1) (2,1) (1,2) |
0 1
1
0
```

The first few lines give lists of possible x- and y-grades of generators and relations in the presentation. (NOTE: With the current code, these lists may not be minimal; we plan to change this soon.)

The next lines specify the bigrades of the generators and relations, via indices for the lists of x- and y-grades. Lists are indexed from 0. Thus, in this example, the row bigrades specified are (7/2,0) and (3,1).

The final three lines specify columns of the matrix in sparse format. Rows are indexed from 0. Hence, the matrix specified is:

```
1 0 1
1 1 0
```

5.1.4 Printing Hilbert Function and Bigraded Betti Numbers

Here is the basic syntax for computing both the Hilbert function and bigraded Betti numbers of a bipersistence module:

```
rivet_console <input_file> --betti [-H <dimension>] [-x <xbins>] [-y <ybins>]
```

As above,

- <input> is an input data file;
- The options for choosing homology dimensions and coarsening parameters behave exactly as for the computation of the MI File.

NOTE: Currently, one cannot print the Hilbert function and bigraded Betti numbers of a module separately. Nor can one print the minimal presentation, Betti numbers, and Hilbert Function together. This will change soon.

The following shows the output format for the Hilbert function and bigraded Betti numbers, for the minimal presentation in the example above:

```
x-grades
3
7/2
4

y-grades
0
1
2

Dimensions > 0:

(0, 1, 1)
(0, 2, 1)

(1, 0, 1)
(1, 1, 1)
(1, 1, 1)

(2, 0, 1)


Betti numbers:
xi_0:
(1, 0, 1)
(0, 1, 1)
xi_1:
(1, 1, 1)
(1, 2, 1)
(2, 1, 1)
xi_2:
(2, 2, 1)
```

The first few lines give lists of possible x- and y-grades of non-zero Betti numbers. This defines a finite grid \(G\in \mathbb R^2\).

The next few lines specify the points in \(G\) where the Hilbert function is non-zero, together with the value of the Hilbert function at each point. For each such point, a triple (x-index, y-index, value) is printed. (Note that this information in fact determines the Hilbert function at all points in \(\mathbb R^2\).)

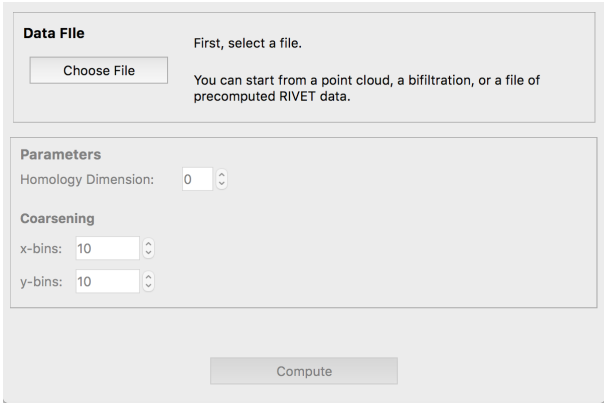
The remaining lines specify the points where the Betti numbers are non-zero, along with the value of the Betti number

at that point. (0th, 1st, and 2nd Betti numbers are handled separately.) Again, for each such point, a triple (x-index, y-index, value) is printed.

5.2 rivet_GUI

The visualizations performed by **rivet_GUI** require an MI file as input. This can be computed by an explicit call to **rivet_console** and then opened in **rivet_GUI**. Alternatively, **rivet_GUI** can call **rivet_console** directly to compute the MI file.

When the user runs **rivet_GUI**, a window opens which allows the user to select a file. This file can be either an input data file in one of the input formats described in the *Input Data* section of this documentation, or a MI file.



If an input data file is selected, then (unless the file is of type firep) the user must choose the homology degree: RIVET currently handles one homology degree at a time. The x-bins and y-bins parameters for the call to **rivet_console** must also be seleced. After the user clicks the compute button, the MI file is computed via a call to **rivet_console** and the visualization is started. (Note that once the Hilbert Function and Betti numbers are shown in the visualization, it may take a significant amount of additional time to prepare the interactive visualization of the barcodes of 1-D slices.) Using the file menu in the GUI, the user may save an MI file.

If an MI file is selected in the file dialogue window, the data in the file is loaded immediately into the RIVET visualization, and the visualization begins.

The RIVET visualization itself is explained in the section "*The RIVET Visualization*".

CHAPTER 6

Input Data

As explained in the section "*Running RIVET*" above, RIVET requires an *input data file*. This file can specify input of the following types:

- Point Cloud or finite metric space, with or without a real-valued function.
- Bifiltration
- FIRep (i.e., short chain complex of free modules).

(These are exactly the objects in green boxes in the figure of the section "*Computation Pipeline*" in this documentation.)

We now specify the formats of the input data file for each of these types of input.

NOTE: RIVET ignores lines that begin with the symbol #; such lines may be used for comments. Blank lines are also ignored.

6.1 Point Cloud with a Function

This format specifies a set of points \(X\) in Euclidean \(n\)-space, a function \(\gamma:X\to \mathbb R\), and a maximum scale parameter \(d\). Given this, RIVET builds the function-Rips bifltration \(R(\gamma)\), including only simplices with diameter at most \(d\).

The file has the following format:

1. The first (non-empty, uncommented) line contains the word "points" and no other characters.
2. The second line specifies the dimension \(n\) of Euclidean space in which the point cloud is embedded.
3. The third line specifies the maximum distance \(d\) of edges constructed in the Vietoris-Rips complex. This must be a positive number (integer or decimal).
4. The fourth line gives the label for the axis along which the values of \(\gamma\) appear.
5. The remaining lines of the file specify the points, one point per line. Each line must specify the coordinates of a point (\(n \) decimal numbers specified by white space), followed by the value of \(\gamma\) on the point.

Here is an example with three points in \(\mathbb R^2\):

```
points
2
3.2
birth time
0 0 3
1.1 2 0.5
-2 3 4
```

Putting the characters `[-]` at the beginning of the line before the label tells RIVET to take the filtration direction on vertices to be descending rather than ascending, as in the following example:

```
points
2
3.2
[-] birth time
0 0 3
1.1 2 0.5
-2 3 4
```

This is useful, e.g., when taking \(\gamma\) to be a density function.

6.2 Finite Metric Space with Function

This format is similar to the one just described, except one specifies the entries of a symmetric distance matrix rather than the coordinates of points in \(\mathbb R^n\). As above, RIVET constructs a function-Rips bifiltration from the input. The given distances are not required to satisfy the triangle inequality.

The file has the following format:

1. The first (non-empty, uncommented) line contains the word the word "metric" and no other printed characters.
2. The second line gives the label for the function \(\gamma\).
3. The third line specifies \(\gamma\). This line consists of a list of \(n\) decimal numbers, separated by white space.
4. The fourth line gives the label for the "distance" axis.
5. The fifth line specifies the maximum distance \(d\) of edges constructed in the Vietoris-Rips complex. This must be a positive number (integer or decimal).
6. The remaining line(s) of the file specify the distances between pairs of points. These distances appear as \(\frac{n(n-1)}{2}\) numbers (integer or decimal), separated by white space or line breaks. Let the points be denoted \(p_1, p_2, \ldots, p_n\). The first \(n-1\) numbers are the distance from \(p_1\) to \(p_2, \ldots, p_n\). The next \(n-2\) numbers give the distances from \(p_2\) to \(p_3, \ldots, p_n\), and so on. The last number gives the distance from \(p_{n-1}\) to \(p_n\).

Here is an example, for a metric space of cardinality 3:

```
metric
birth time
1 1.1 -2
geodesic distance
2.5
2 3.2
1.25
```

As above, we can reverse the filtration direction on vertices by placing `[-]` at the beginning of the appropriate label.

6.3 Point Cloud / Finite Metric Space without Function

Given either a point cloud in Euclidean space or a finite metric space with no function on vertices specified, RIVET constructs the degree-Rips bifiltration.

A point cloud with no function is specified as in the following example:

```
points
2
3.2
no function
0 0
1.1 2
-2 3
```

Given the input specification for a point cloud with a function, this variant should be self-explanatory.

A finite metric space with no function is specified as in the following example:

```
metric
no function
3
Rips scale
2.5
2 3.2
1.25
```

As above, this format is mostly self-explanatory, given the input specification for a metric space with a function. However, the 3 appearing on the third line requires explanation: This is the number of points in the finite metric space. (This input convention is redundant: the number in the third line is always one greater than the number of entries on sixth line. The reason for this choice of convention is that it made it simpler to write the code to parse this input, given what we already had.)

6.4 Bifiltration

RIVET can accept as input any essentially finite bifiltration. (Multicritical bifiltrations are allowed.)

Let \(v_1, v_2, \ldots, v_n\) denote the vertices (0-simplices) of the bifiltration. Specifying the bifiltration requires specifying each simplex (given as a subset of \(v_1, v_2, \ldots, v_n\)) and its birth indices. Simplices are specified, one simplex per line, in the bifiltration input file.

The user must ensure that the input file specifies a valid bifiltration, in the sense that a simplex is never born before its faces; RIVET does not error-check this.

A file in the bifiltration format must have the following format:

1. The first (non-empty, uncommented) line contains the word "bifiltration" and no other printed characters.
2. The second line gives a label for the first filtration parameter.
3. The third line gives a label for the second filtration parameter.
4. The remaining lines of the file each specify a simplex and its bigrades of appearance. A line specifying a \(j\)-simplex with \(n\) grades of appearance must have \(j+1\) non-negative integers (separated by white space), followed by a semicolon, followed by \(2n\) numbers (which may be integers or decimals. The semicolon must

be surrounded by spaces. The first \(j+1\) integers give the vertices of the simplex. The remaining numbers specify the bigrades at which the simplex appears.

A sample multicritical bifiltration file appears below. This consists of: the boundary of a triangle born at \((0,0)\); the interior of the triangle born at both \((1,0)\) and \((0,1)\); two edges that complete the boundary of a second triangle adjacent to the first, born at \((1,1)\):

```
bifiltration
time of appearance
network distance
0 ; 0 0
1 ; 0 0
2 ; 0 0
3 ; 0 0
0 1 ; 0 0
0 2 ; 0 0
1 2 ; 0 0
0 1 2 ; 0 1 1 0
1 3 ; 1 1
2 3 ; 1 1
```

The minimal grades of appearance of a given simplex may be given in arbitrary order. For example, it is also valid to take the seventh of the above input file to be:

```
0 1 2 ; 1 0 0 1
```

Moreover, the code can handle non-minimial bigrades of appearance; it simply removes them. (However, in the current code, non-minimal bigrades of appearance may change the coarsening behavior, as the \(x\)- and \(y\)-grades of such bigrades are currently not ignored when performing coarsening.)

One can also take the filtration direction for either of the axes to be decreasing, by placing `[-]` in front of an axis label. For instance, the following variant of the last example replaces the y-coordinate of each bigrade with its negative, and takes the filtration direction for the \(y\)-coordinate to be descending:

```
bifiltration
time of appearance
[-] network distance
0 ; 0 0
1 ; 0 0
2 ; 0 0
3 ; 0 0
0 1 ; 0 0
0 2 ; 0 0
1 2 ; 0 0
0 1 2 ; 0 -1 1 0
1 3 ; 1 -1
2 3 ; 1 -1
```

6.5 FIRep (Algebraic Input)

An FIRep \[C_2 \xrightarrow{f} C_1 \xrightarrow{g} C_0. \] is specified as follows:

1. The first (non-empty, uncommented) line says “firep”.
2. The second line is the \(x\)-label.
3. The third line is the \(y\)-label.

4. The fourth line is of the form `t s r`, where `t`, `s`, and `r` are, repsectively, the number of generators in bases for \(C_2\), \(C_1\), and \(C_0\).
5. Each of the next `t` lines specifies the bigrade of appearance of a basis element for \(C_2\), together with the corresponding column of the matrix representing \(f\): the format for such a line is: `x y ; b1 b2 b3`, where the `bi` are the row indices of nonzero column entries. (Recall that we work with \(\mathbb{Z}/2\mathbb{Z}\) coefficients.)
6. Each of the next `s` lines specifies the bigrade of appearance of a basis element for \(C_1\), together with the corresponding column of the matrix representing \(g\).

An example FIRep input is shown below:

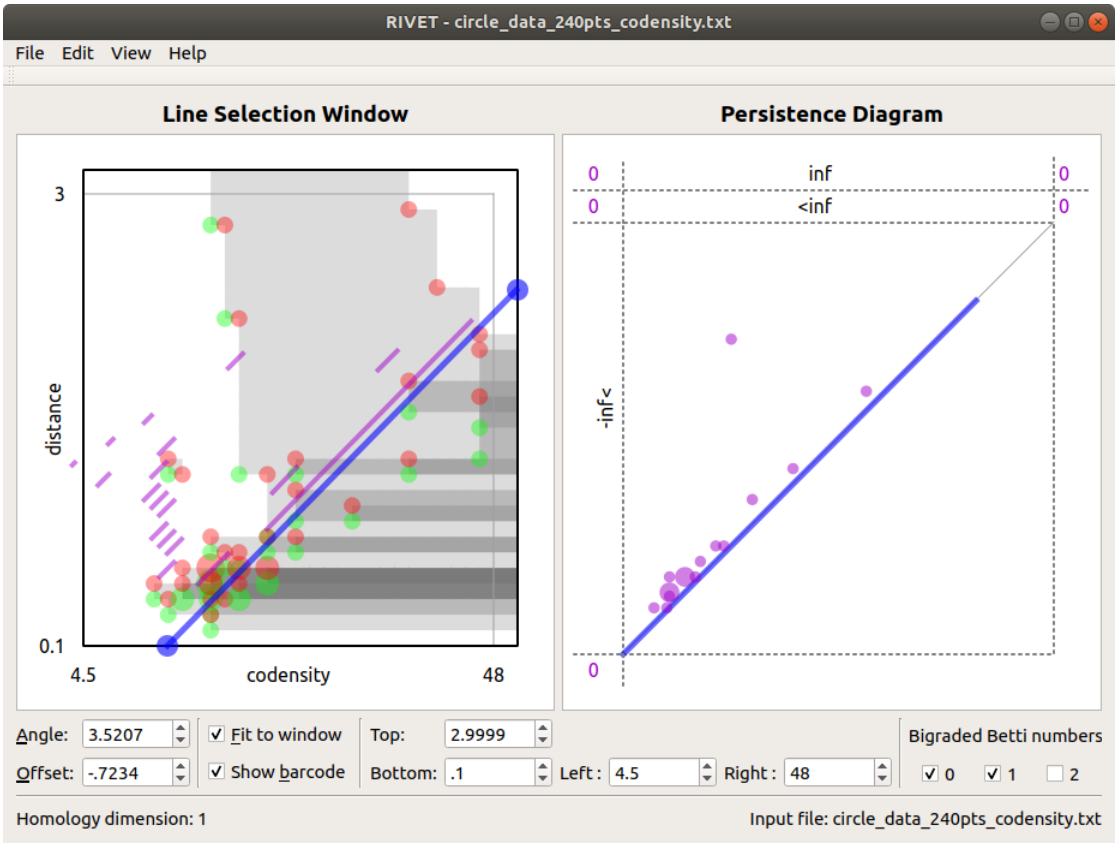
```
firep
parameter 1
parameter 2
2 3 3
1 0 ; 1 1 1
0 1 ; 1 1 1
0 0 ; 1 2
0 0 ; 0 2
0 0 ; 0 1
```

This example has a natural geometric interpretation. The boundary of a triangle is born at \((0,0)\), and the triangle is filled in at both \((1,0)\) and \((0,1)\). The input gives the portion of the resulting chain complex required to compute the 1st persistent homology module.

CHAPTER 7

The RIVET Visualization

The RIVET interface contains two main windows, the *Line Selection Window* and the *Persistence Diagram Window*, shown in the screenshot below.



7.1 Line Selection Window

By default, the *Line Selection Window* plots a rectangle in \(\mathbb{R}^2\) containing the union of the supports of bigraded Betti number functions \(\xi_i^M\), \(i\in \{0,1,2\}\). (If the input to RIVET is an firep and the Betti numbers are not supported on a horizontal or vertical line, this will be the smallest such rectangle. If the input is a point cloud, metric space, or bifiltration, and the birth indices of all simplices in the bifiltration do not lie on a single line, the rectangle will be the smallest one containing the birth indices of all simplices.)

Points in the supports of \(\xi_0^M\), \(\xi_1^M\), and \(\xi_2^M\) are marked with green, red, and yellow dots, respectively (though these colors are customizable via the Edit menu on Linux or the Preferences menu on Mac). The area of each dot is proportional to the corresponding function value. The dots are translucent, so that, for example, overlaid red and green dots appear brown on their intersection. This allows the user to read the values of the Betti numbers at points which are in the support of more than one of the functions. Furthermore, hovering the mouse over a dot produces a popup box that gives the values of the bigraded Betti numbers at that point.

The greyscale shading at a point \(a\) in this rectangle represents \(\dim M_a\): \(a\) is unshaded when \(\dim M_a=0\), and larger \(\dim M_a\) corresponds to darker shading. Hovering the mouse over \(a\) brings up a popup box which gives the precise value of \(\dim M_a\).

A key feature of the RIVET visualization is the ability to interactively select the line \(L\) via the mouse and have the barcode \(\mathcal B(M^L)\) update in real time. The Line Selection Window contains a blue line \(L\) of non-negative slope, with endpoints on the boundary of the displayed region of \(\mathbb{R}^2\). RIVET displays a barcode for \(M^L\) in the line selection window, provided the "show barcode" box is checked below. The intervals in the barcode for \(M^L\) are displayed in purple, perpendicularly offset from the line \(L\).

The user can click and drag the blue line with the mouse to change the choice of line \(L\). Clicking and dragging an endpoint of the line moves that endpoint while keeping the other fixed. One endpoint is locked to the top and right sides of the displayed rectangle; the other endpoint is locked to the bottom and left sides. Clicking and dragging the interior of the line (away from its endpoints) moves the line as follows:

- Left-clicking moves the line in the direction perpendicular to its slope, keeping the slope constant.
- Right-clicking changes the slope of the line, while keeping the bottom/left endpoint fixed.

As the line moves, both the barcode in the Line Selection Window and its persistence diagram representation in the Persistence Diagram Window are updated in real time. The *Angle* and *Offset* controls below the Line Selection Window can also be used to select the line.

The coordinate bounds of the viewable rectangle may be adjusted using the *Top*, *Bottom*, *Left*, and *Right* control boxes at the bottom of the RIVET window. The window can be reset to the default by clicking on "Restore Default Window," under the "View" menu. Clicking on "Betti number window" under the "View" menu sets the window to the smallest rectangle containing all non-zero Betti numbers.

7.2 Persistence Diagram Window

The Persistence Diagram Window (at right in the screenshot above) displays a persistence diagram representation of the barcode for \(M^L\).

The bounds for the square viewable region (surrounded by dashed lines) in this window are chosen automatically. They depend on the bounds of the viewable region in the slice diagram window, but not on \(L\).

Let the square \([0,B]\times[0,B]\) be the viewable region. It may be that the barcode contains some intervals \([\alpha,\beta)\) with either \(\alpha\) or \(\beta\) not contained in \([0,B]\). To represent such intervals on the screen, RIVET displays some information at the top and left of the persistence diagram which is not found in typical persistence diagrams.

Above the square region of persistence diagram are two narrow horizontal strips, separated by a dashed horizontal line. The upper strip is labeled *inf*, and the lower strip is labeled \(\lt\)*inf*. RIVET plots a point in the upper strip for each interval \([\alpha, \infty)\) in the barcode with \(0\leq \alpha \le B\). RIVET plots a point in the lower strip for each interval \([\alpha, \beta)\) in the barcode with \(0\leq \alpha \le B\lt \beta \lt \infty)\).

To the left of the square region of persistence diagram is a vertical strip labeled - *inf* \(\lt\). RIVET plots a point in this strip for each interval \([\alpha, \beta)\) in the barcode with \(\alpha \lt 0\leq \beta \leq B)\).

Just to the right and to the left of each of the two upper horizontal strips is a number, separated from the strip by a dashed vertical line:

- To the upper right is the number of intervals \([\alpha, \infty)\) in the barcode with \(B \lt \alpha\).
- To the lower right is the the number of intervals \([\alpha, \beta)\) with \(B \lt \alpha\) and \(\beta \lt \infty\).
- To upper left is the number of intervals \([\alpha, \infty)\) with \(\alpha\lt 0\).
- To the lower left is the number of intervals \([\alpha, \beta)\) with \(\alpha< 0\) and $B\lt\beta \lt\infty)).

Finally, there is a number in the bottom left corner of the persistence diagram window. This is the number of intervals \([\alpha, \beta)\) with \(\alpha\lt \beta< 0\).

As with the bigraded Betti numbers in the Line Selection Window, the multiplicity of a point in the persistence diagram is indicated by the area of the corresponding dot. Additionally, hovering the mouse over a dot produces a popup that displays the multiplicity of the dot.

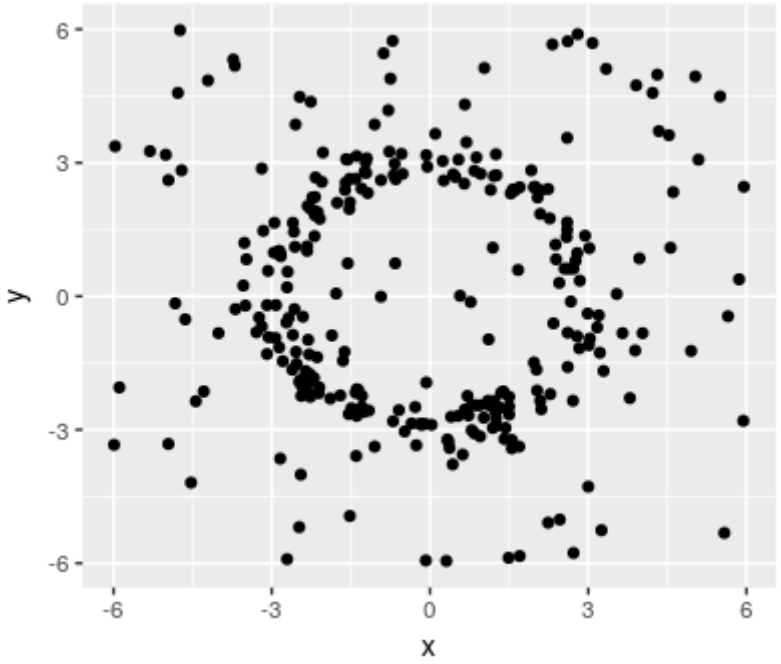
7.3 Customizing the Visualization

The look of the visualization can be customized by clicking on “Preferences” in the “RIVET” menu, and adjusting the settings there.

CHAPTER 8

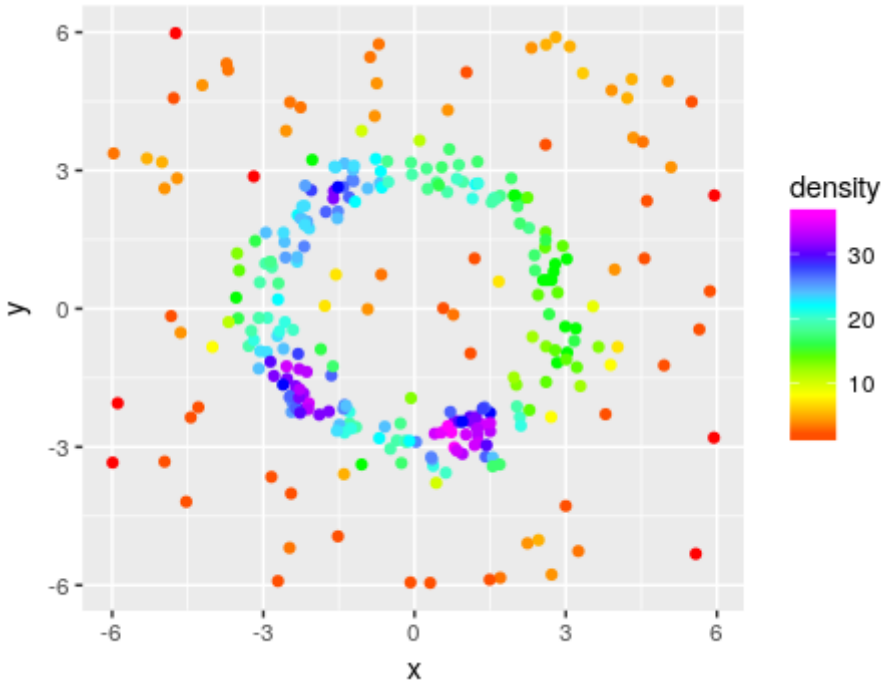
Example

This page demonstrates how to use RIVET to visualize the topological structure of a point cloud. We consider the point cloud in \(\mathbb{R}^2\) pictured below. We emphasize, however, that RIVET can handle point clouds in any dimension, as well as other data types.



Evidently, this point cloud contains a dense circle of points, as if many of the points were sampled from an annulus. However, we also observe outliers, both inside and outside of the annulus. Note that one-parameter persistence (e.g., using a Rips filtration) cannot easily detect the dense cycle, due to the presence of outliers.

We will associate to this data a function-Rips bifiltration, taking the function on the points to be a density estimate; here, we use a simple ball density estimator, which for each point \(p\) simply counts the number of points within a unit distance of \(p\). The following diagram colors the points by density.



8.1 Input File

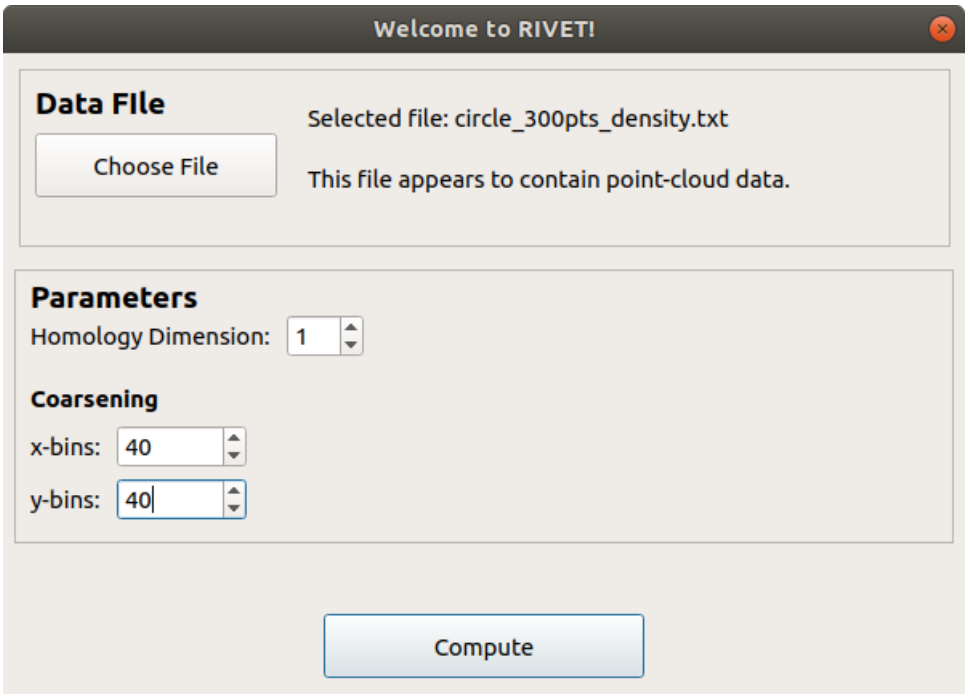
We next prepare an input file for RIVET. This is a text file written in the format described on the *Input Data* page. The first lines of the text file appear below (the complete file is in the RIVET repository):

```
points
2
4
[-] density
1.57    2.40    20
1.21    2.70    22
0.79    -2.44   32
...
```

The first four lines are parameters for RIVET: `points` tells RIVET that the file contains point-cloud data, `2` indicates that each point is specified by two coordinates, `4` specifies that the maximum length of edges to be constructed in the Rips filtration will be 4, and `density` is a text label for the horizontal axis in RIVET. The `[-]` preceding the axis label tells RIVET to filter by density in decreasing order, so that points with the largest density appear first in the bifiltration.

8.2 Computation from the GUI

We proceed using the the RIVET GUI. With RIVET installed, we run `rivet_GUI`. The input data dialog box appears. We select the data file and computation parameters, as shown below. Since we are interested in detecting a cycle, we select homology dimension 1. We must also set parameters for the coarseness of the computation, via the *bins* selectors. These tell RIVET to round the computed values for density and distance to a specified number of equally-spaced values, which speeds up the computation. We will choose 40 bins in each direction.

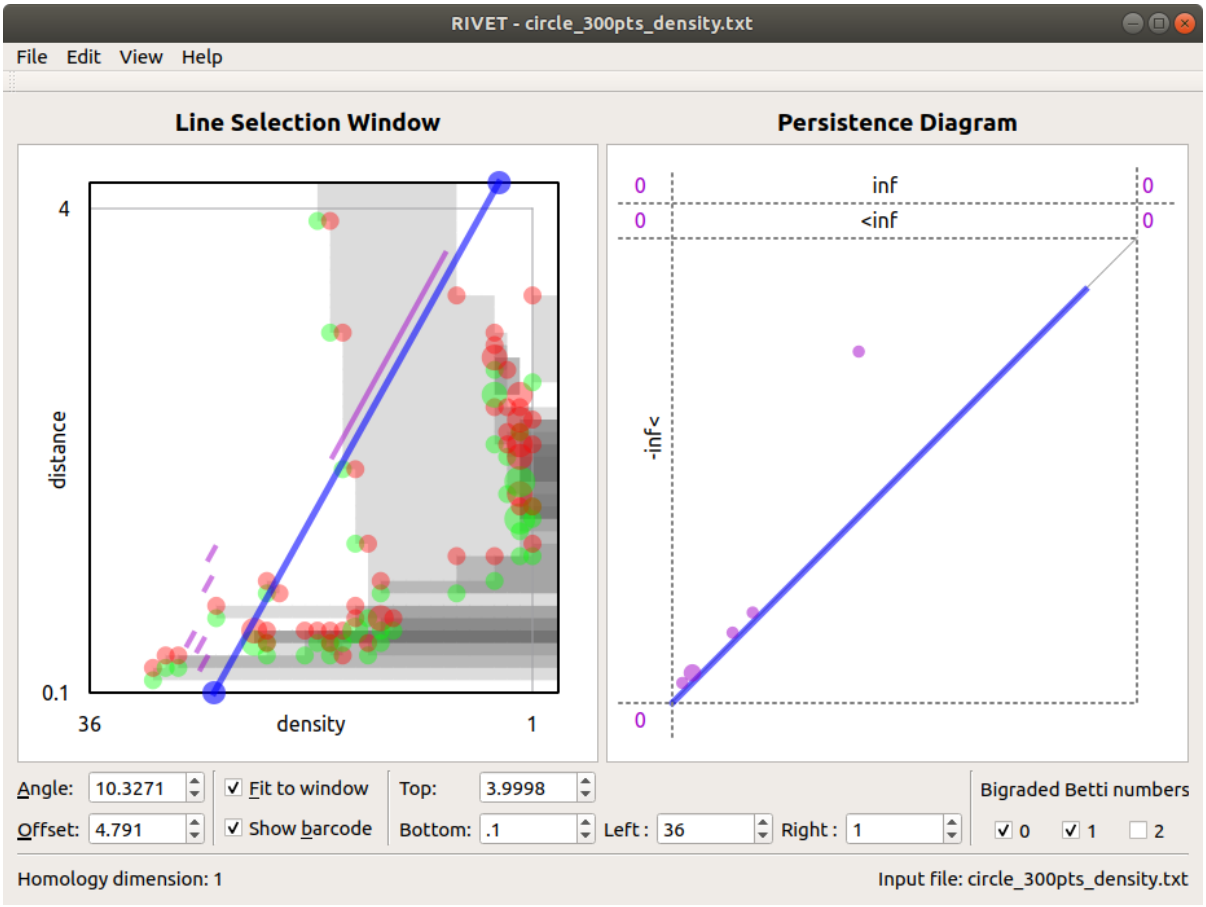


(Note that setting the bin selectors to *zero* will cause RIVET not to round the computed values, is more computationally expensive.)

We click **Compute**, and RIVET computes the augmented arrangement. This may take several minutes, depending on the computing power available.

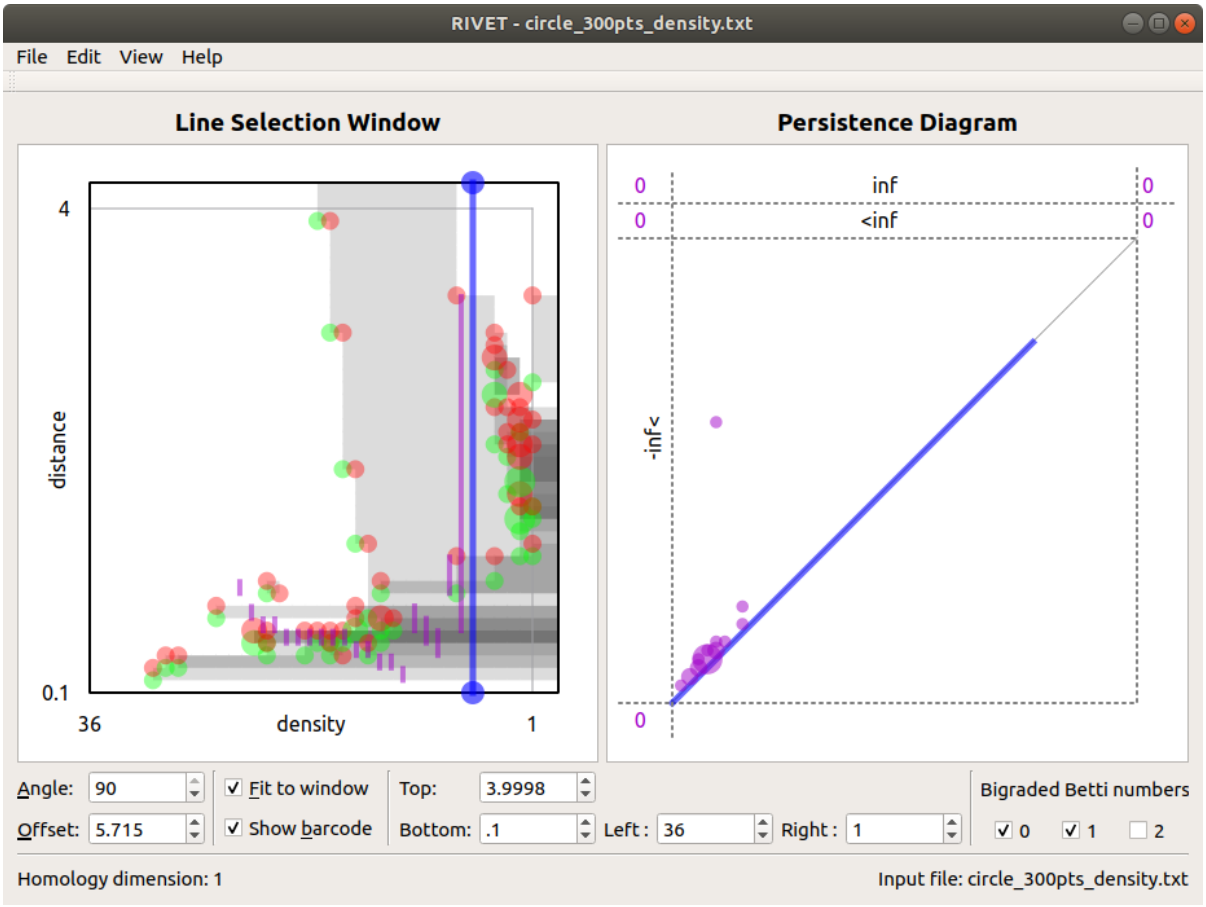
8.3 Visualization

When the Hilbert function and bigraded Betti numbers have been computed, visualizations of these appear in the *Line Selection Window* on the left side of the RIVET window (see the *The RIVET Visualization* page for more details). When the computation of the augmented arrangement is complete, a barcode appears in the *Line Selection Window* and a persistence diagram appears in the *Persistence Diagram* window in RIVET, as shown below. RIVET is now ready for interactive browsing of barcodes along linear slices through the bipersistence persistence module.



For this data, note that the barcode contains a single long bar when the selected line goes roughly from the lower-left corner to the upper-right corner of the Line Selection Window. This bar corresponds to the dense cycle of points in the point cloud.

Furthermore, note that selecting vertical lines effectively thresholds the points by density. That is, selecting a vertical line with density value \(d\) produces a barcode computed from a Rips filtration on only those points with density value greater than \(d\), as shown below. This effectively reduces the analysis to one-parameter persistence, using a density threshold. The RIVET GUI allows the user to slide the vertical line left and right, thereby displaying the barcodes for different choices of the density threshold.



8.4 Computation from the Console

Rather than using the RIVET GUI, one may use the RIVET console application to compute the augmented arrangement and even obtain barcodes. This is done using the command line, as described in *Running RIVET*. For example, the computation described above can be obtained from **rivet_console** using the following command, run from the root directory of the RIVET repository:

```
./rivet_GUI/data/Test_Point_Clouds/circle_300pts_density.txt circle_300_computed.mif -
↪H 1 -X 40 -y 40
```

This will produce a module invariants file `circle_300_computed.mif`, which may then be loaded into the RIVET GUI or queried for barcodes on a collection of user-chosen lines. Please see *Running RIVET* for more details.

CHAPTER 9

Indices and tables

- genindex
- search