

UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Fisica e Astronomia “Galileo Galilei”

Master Degree in Physics

Final Dissertation

Topological Data Analysis, Persistent Homology and Machine Learning for Gravitational Wave Detection

Thesis supervisor

Prof. Pierpaolo Mastrolia

Thesis co-supervisor

Dr. Manoj Kumar Mandal

Candidate

Filip Levi Rancan

Academic Year 2025/2026

Abstract

In this thesis we address the problem of gravitational wave (GW) detection, with a novel interdisciplinary approach combining general relativity, geometry and computer science.

The problem of gravitational-wave detection consists in identifying the presence of a gravitational-wave signal, typically extremely weak, within data dominated by instrumental and environmental noise. It is a challenging problem, since a genuine astrophysical contribution has to be distinguished from random fluctuations of the noise, maximising sensitivity while at the same time minimising false alarms.

Detecting gravitational waves is of crucial importance: they open an observational window on the Universe that is invisible to light, letting us study phenomena such as black-hole mergers and test general relativity in the strong-field regime. Next-generation detectors, such as the Einstein Telescope and Cosmic Explorer, are expected to reach sources far more distant and far more numerous than those observed today.

Specifically, in this work we employ topological data analysis (TDA), a collection of methods, drawn from a branch of mathematics called algebraic topology, for studying the *shape* of data. Its principal method is *persistent homology*, built on the idea that a set of measurements, however noisy, often carries a geometric structure — points grouping into clusters, closing up into loops, or surrounding an empty region — that a traditional analysis based on averages and correlations could miss. In particular, persistent homology is robust to noise and analyses the data from a multi-scale perspective. Because of its properties, topological data analysis has been developing over the last two decades, and is already used with success across the sciences, from the study of protein structures to cosmology.

Our motivating question is whether the shape of gravitational-wave data, measured topologically, carries enough information to detect gravitational-wave signals. To investigate this we also rely on machine learning (ML), the branch of artificial intelligence concerned with systems that learn patterns from data: a ML model adjusts its own internal parameters until it captures the patterns the data contain, aiming to perform well on data it has never seen.

In this work we develop a novel algorithmic approach for the detection of gravitational waves, combining elements of topological data analysis and machine learning. Therefore we first simulate the gravitational waveforms using Python software developed within the LIGO–Virgo–KAGRA collaboration, and embed those waveforms in simulated noise of the LIGO interferometers. Each resulting time series is then preprocessed with standard data-analysis techniques, so that the injected waveform emerges as clearly as possible from the surrounding noise.

With the dataset ready, we implement a computational pipeline based on topological data analysis. Each time series is first turned into a cloud of points in $\mathbb{R}^n$ by a suited algorithm, and the resulting point cloud shape depends on whether the time series contains a gravitational-wave signal or only noise.

We then apply persistent homology to each of these point clouds, tracking, across every scale at once, how the points gather into clusters, close up into loops, or surround an empty region, and reporting how long each such feature survives. A gravitational-wave signal produces a cloud that winds into a spiral, whose loops persistent homology detects as long-lived features; pure noise produces a shapeless cloud, whose loops are all short-lived. The result of persistent homology is a collection of numbers and images, which we call *topological summaries*, that describe the shape of the cloud each time series produced.

We then design a feature-selection stage to identify which of these topological summaries carry the most discriminating power, and use them to train a range of machine-learning classifiers. What each classifier learns is a statement about shape: that a gravitational-wave signal arranges the points of its cloud into clusters and loops in distinctive ways. Once trained, we show that the machine learning classifiers reliably detect the presence of a gravitational wave in the noisy time series.

The entire procedure — from waveform simulation to machine learning classification — is implemented as a single, reusable Python pipeline.

Contents

Abstract	1
Introduction	6
1 Persistent Homology	10
1.1 Geometric preliminaries	10
1.1.1 Three notions of sameness	10
1.1.2 Topological spaces and connected components	14
1.2 Simplicies and simplicial complexes	16
1.2.1 Abstract simplicies	16
1.2.2 Geometric simplicial complexes	18
1.3 Chains	20
1.3.1 Oriented simplicies	24
1.4 Holes: homology groups and Betti numbers	26
1.4.1 The zeroth Betti number	29
1.4.2 The Euler characteristic	30
1.5 Computing homology	33
1.6 Filtrations and persistent homology groups	35
1.6.1 Definition	35
1.6.2 Barcodes and persistence diagrams	38
1.6.3 Computing persistent homology	40
2 Topological Data Analysis	42
2.1 From point clouds to complexes	42
2.1.1 The homology inference problem	42
2.1.2 Rips complexes	43
2.1.3 Čech complexes	45
2.1.4 Nerve complexes	46
2.2 Stability	48
2.2.1 Continuous filtrations	48
2.2.2 Bottleneck distance and stability theorem	49
2.3 Takens embedding: from timeseries to point clouds	51
2.3.1 From time series to time delay embeddings	52
2.3.2 From time delay embeddings to persistence diagrams	56
2.3.3 Picking the embedding dimension and time delay	56
2.3.4 Running the search algorithm	57
2.3.5 Principal component analysis	58
2.4 Vectorization	60
2.4.1 Real-valued summaries	60
2.4.2 Persistence images	63
2.4.3 Betti curves and area under the Betti curve	64

3	Topological Data Analysis for Periodic and non-Periodic Signals	67
3.1	The toy model	67
3.2	From signal to point cloud	68
3.3	Computing persistent homology	70
3.4	Topological summaries	72
3.4.1	Scalar summaries	72
3.4.2	Betti curves and their area	73
3.4.3	Persistence images	74
3.5	Searching for the embedding, without forcing three dimensions	75
3.6	Edge collapses	78
3.6.1	Timing test	80
4	Gravitational Waves: Theory and Detection	83
4.1	Introduction and outline	83
4.2	General relativity, gravitational waves and compact binaries	84
4.2.1	General relativity	84
4.2.2	Gravitational waves in general relativity	85
4.2.3	General relativity at linear order and gravitational waves	87
4.2.4	The transverse-traceless gauge and the two polarizations	88
4.2.5	Inspiral of compact binaries and the chirp signal	88
4.3	The detector as a linear system and the noise spectral density	89
4.3.1	The Michelson interferometer	89
4.3.2	Input, output and the detector tensor	91
4.3.3	The noise spectral density	91
4.3.4	The physical origin of the noise	92
4.4	Matched filtering	93
4.4.1	The intuitive idea	93
4.4.2	The optimal linear filter	94
4.4.3	The optimal signal-to-noise ratio	95
4.4.4	Matched filtering in practice	96
4.5	Probability, statistics and detection	97
4.5.1	Frequentist and Bayesian frameworks	97
4.5.2	Detection statistic, threshold and false alarms	98
4.6	Application to coalescing compact binaries	100
4.6.1	The inspiral waveform and the chirp mass	100
4.6.2	The optimal SNR and the sight distance	101
4.6.3	Eliminating the extrinsic parameters	101
4.6.4	Template banks	102
4.7	The case for new search strategies	103
4.7.1	The need for an accurate waveform model	104
4.7.2	Computational cost and the template-bank bottleneck	105
4.7.3	Non-Gaussian, non-stationary noise and glitches	106
4.7.4	The case for a new detection strategy	106
5	Topological Data Analysis for Gravitational Waves	108
5.1	Preparing a phenomenologically realistic dataset	108
5.1.1	Generating and tapering the waveform	109
5.1.2	Generating coloured detector noise	110
5.1.3	Placing the signal at a random merger time	111
5.1.4	Rescaling the signal, not the noise, to a target SNR	112
5.1.5	Whitening and band-passing	113
5.1.6	Summary of the resulting dataset	114
5.2	Running the topological pipeline on a single sample	115
5.2.1	The three signals	115
5.2.2	From time series to point cloud	116

5.2.3	Persistent homology	118
5.2.4	Topological summaries	119
5.2.5	What this example does, and does not, establish	122
5.3	Extracting topological features from the whole dataset	123
5.3.1	One embedding for the whole dataset	123
5.3.2	Reducing the point clouds	124
5.3.3	Persistent homology of the whole collection	125
5.3.4	Computing the summaries	125
6	Machine Learning Detection of Gravitational Waves from Topological Features	128
6.1	Machine learning classifiers	128
6.1.1	Supervised binary classification	128
6.1.2	The classifiers	129
6.1.3	Validating the classifiers	136
6.2	Results	139
6.2.1	The dataset and the features	140
6.2.2	The classical classifiers	141
6.2.3	The convolutional networks (CNNs)	144
6.2.4	Results interpretation	146
6.2.5	Detection sensitivity as a function of signal strength	155
6.3	Comparing the cost against matched filtering	156
6.3.1	Measuring the difference	156
6.3.2	What the speed does not buy	158
	Conclusions	159
	Acknowledgments	163
A	Deferred proofs and constructions	164
A.1	The formal definition of a metric	164
A.2	Every isometry is a homeomorphism	164
A.3	The Euclidean topology	165
A.4	Continuous images of connected components	165
A.5	Homeomorphic spaces have the same number of connected components	165
A.6	Homotopy equivalent spaces have the same number of connected components	165
A.7	Geometric simplices and simplicial complexes	166
A.8	The topological realization of an abstract complex	167
A.9	The formal definition of a triangulation	167
A.10	Dimension of a quotient vector space	167
A.11	The boundary of a boundary is zero	168
A.12	Every boundary is a cycle	168
A.13	Induced orientation of a facet	168
A.14	Parity of 0-boundaries	168
A.15	Homologous cycles	169
A.16	Computing the Betti numbers of the square	169
A.17	The zeroth Betti number counts components	170
A.18	Euler–Poincaré formula	171
A.19	Rank, nullity, and the Betti-number formula	171
A.20	Extracting representing cycles from the Smith normal form	171
A.21	Simplicial maps	172
A.22	Functoriality of homology	172
A.23	The barcode, formally	173
A.24	Fundamental lemma of persistent homology	174
A.25	Computing persistent homology: matrix reduction, extraction, and representatives	174
A.26	Rips and Čech filtrations (continuous formulation)	178

A.27	Transitivity of homeomorphism	179
A.28	Invariants under homotopy equivalence	179
A.29	Why the nerve theorem applies to a collection of balls	179
A.30	Passing between discrete and continuous filtrations	180
A.31	The stability theorem, complete statement	180
A.32	Persistence images: formal construction	180
A.33	Shannon entropy: an intuitive discussion	181
A.34	The noise spectral density of a real signal is symmetric	182
A.35	Dimensions of the noise spectral density	182
A.36	The detection statistic is Gaussian in the absence of a signal	183
A.37	The detection statistic in the presence of a signal	183
A.38	The distribution of the SNR in energy	183
A.39	The mean detected energy SNR	184
A.40	From 10^5 templates to 10^{15} trials	184
A.41	Computational complexity of the matched-filter statistic	185
A.42	Spectral leakage from an abruptly truncated signal	185
A.43	Rescaling a waveform to a target optimal SNR	186
A.44	What <code>inverse_spectrum_truncation</code> does, and why	187
A.45	Two ways to average an AUC under cross-validation	187
A.46	Correlated features and unstable coefficients	188
B	Pattern functions and the angular response	190
C	Full code listing: the toy-model TDA pipeline	191
C.1	Imports	191
C.2	Configuration	191
C.3	Build the three timeseries	192
C.4	Takens embedding — automatic parameter search	193
C.5	Takens embedding — three point clouds per signal	194
C.6	Helper functions for the topological pipeline	195
C.7	Topological pipeline — Rips complex + all features, run once per branch	197
C.8	Run the pipeline for all three branches	200
D	Full code listing: the dataset-generation pipeline	201
D.1	Imports	201
D.2	Configuration	201
D.3	Dataset-level parameters	203
D.4	Sampling binary parameters	203
D.5	Tapering the waveform edges	204
D.6	Generating the waveform	204
D.7	Placing the signal in the buffer	204
D.8	Noise spectral density and coloured noise	205
D.9	Computing and rescaling the optimal SNR	205
D.10	Whitening and band-passing	206
D.11	Building one signal or noise sample	206
D.12	Plotting helpers	207
D.13	Generating the full dataset	208
D.14	Topological-feature test mode	209
D.15	Saving the dataset	210

Introduction

Einstein inferred the existence of gravitational waves (GW) in 1916, one year after he formulated general relativity [1, 2]. For four decades it was not clear whether these waves were physical or an artefact of the coordinate freedom of the theory, and the question was settled only at the Chapel Hill conference of 1957 [3]. The first indirect evidence arrived in the following decades, from the steady decay of the orbit of the binary pulsar discovered by Hulse and Taylor [4, 5]. The first direct detection came in 2015, when the LIGO interferometers observed the merger of two black holes [6, 7]. Detections have since then become routine and, as the detector network grows further, they are expected to arrive almost daily [8]. A third generation of instruments is already being designed — the Einstein Telescope [9] and Cosmic Explorer [10] on the ground, and the space-based LISA [11] — each of them substantially more sensitive than what is operating today.

What makes these observations remarkable is how little there is to observe. A passing gravitational wave changes the length of a detector’s arms by a fraction of their length so small that the resulting displacement is a small fraction of the size of a proton [12], and the resulting signal sits far beneath the instrument’s noise. This would be a hopeless situation, but one fact compensates for it. If we know in advance the shape of the waveform we are looking for, we can stop asking whether the data is loud and ask instead whether it has that shape. A real GW chirp matches the expected shape consistently, cycle after cycle; noise matches it only now and then, by chance. Adding up that agreement over the whole signal is enough to detect a GW: a signal can be quieter than the noise to be found [12]. Detecting a gravitational wave is therefore not only a problem of building a quieter instrument. It is a problem of what one does with the data afterwards. The method that realises this idea is *matched filtering*, and it is what the LIGO–Virgo–KAGRA interferometers actually run.

Matched filtering is a powerful tool, but it has some limitations. The waveform is assumed to be known exactly, yet some sources have no precise gravitational-wave template [13] and even binary coalescences are modelled only approximately [14]. The noise is assumed Gaussian and stationary, yet real detectors drift and carry short, loud transients, or *glitches*, which can imitate a real signal [12]. And since the source parameters are unknown in advance, the matched filtering must scan a bank of hundreds of thousands of waveform templates [12] which means that the method is computationally intensive.

These conditions motivate the search for complementary methods for gravitational-wave detection: in this thesis we investigate whether a topological description of the shape of the data carries usable information about the presence of a gravitational wave, and at what computational cost it can be obtained. To achieve this goal we need to introduce the field of *topological data analysis*.

Topological data analysis (TDA) is a collection of methods, drawn from a branch of mathematics called algebraic topology, for studying the *shape* of data. The principal method of topological data analysis is *persistent homology*. Its premise is that a set of measurements, however noisy they are, often carries a structure that is geometric rather than statistical: the points group into clusters, or close up into loops, or surround an empty region. Such features are qualitative, and they record something real about the process that produced the data, and a more traditional analysis built on averages and correlations passes over them [15]. Two further properties make this kind of description attractive for physical measurements. The first is that topological data analysis is robust to noise [12]. Counting clusters or holes is a coarser question than asking where each point lies, and topological data analysis tolerates imprecise data: displace every point slightly and a clearly formed cluster is still a cluster, a clearly enclosed hole still enclosed. The second property of topological data analysis is that it is not tied to a precise scale [16]: whether two points count as part of the same cluster, or a ring of points counts as enclosing a hole, depends on how close together we require points to be before we treat them as connected. Fixing that threshold once, in advance, means committing to a single answer; topological data analysis instead lets the threshold

grow from zero to infinity and tracks how the clusters and holes appear, merge and close up along the way. The field of topological data analysis has been developing over the last two decades [15] and this topological approach is already being used with success across the sciences, from the study of protein structure [17] and of brain connectivity [18] to materials science [19] and cosmology [20].

The topological description above produces, for a given stretch of GW data, a collection of topological features that capture the shape and structure of the data. Deciding from those topological features whether a gravitational wave is present is a further step, and one way to take it is *machine learning*. Machine learning (ML) is the branch of artificial intelligence concerned with systems that learn patterns from data [21]. In particular, the machine learning model adjusts its own internal parameters until it captures the patterns that data contains, aiming not to reproduce the examples it was shown but to perform well on data it has never seen. Machine learning divides into several families, and the one used in this thesis is *supervised learning*, which learns from labelled examples [21]: each training example is an input together with the correct answer for it, and the model is adjusted until its own answers agree with those provided. In particular in this work we deal with classification, which is the task of selecting an answer from a fixed set of alternatives, such as determining whether a stretch of data contains a signal [22]. Machine learning has already found use in gravitational-wave detection. It has been applied directly to the GW interferometers output and deep networks have already been shown to identify compact-binary signals in Advanced LIGO data, and to do so in real time [23].

In this thesis we combine topological data analysis and machine learning for gravitational-wave detection. Elaborating on the earlier, preliminary study proposed in [24], we use persistent homology to improve the discrimination GW signals from the noisy data. To achieve this goal, we need to build a dataset of gravitational-wave signals embedded in noise, which is then processed through the topological data analysis pipeline and fed to machine learning classifiers for training and evaluation.

Therefore the first step is to build a gravitational-wave dataset in the way that is standard in the field of gravitational-wave data analysis [25]. Our method ends in a machine-learning classifier, and training a classifier takes tens of thousands of labelled examples, whereas the confirmed catalogue of detections holds only a couple of hundred [26] — far too small to train a machine-learning model.

The first step is therefore to simulate the gravitational waveforms ourselves, using PyCBC [27], an open-source Python package for gravitational-wave data analysis built to detect and study the mergers of black holes and neutron stars. This lets us generate as many waveforms as we need and each one with its own values for the parameters of the coalescing binary.

The second step is to embed each simulated waveform in noise, itself simulated to reproduce the noise of the LIGO interferometers. What we are after here is a dataset varied enough to stand in for the range of what a real detector sees. The masses of the two bodies, whose merger produces the wave, are drawn at random for every waveform, so no source is ever generated twice. Each waveform is also embedded at a randomly chosen instant within its stretch of noise, and at a randomly chosen strength, so that the dataset holds both strong and weak gravitational waves. The dataset is then put through the standard preprocessing of gravitational-wave data analysis [25, 28]. Its purpose is to flatten the noise spectrum, so that no band of frequencies dominates artificially and the injected waveform can emerge as clearly as possible.

With the dataset ready, we can apply the topological methods described above. Each time series in the dataset — a stretch of interferometer noise, with or without a gravitational-wave signal buried in it — is turned into a cloud of points in $\mathbb{R}^n$ by an algorithm known as the *Takens embedding* [29]. The resulting point cloud has a shape, and that shape depends on what the time series was doing. A gravitational-wave signal, whose frequency and amplitude grow as the two bodies spiral together, produces a cloud that winds into a spiral. A stretch of pure noise produces a shapeless ball. The two clouds differ in a way that is topological rather than numerical, and it is that difference we set out to detect.

We then apply persistent homology to each cloud, which records, at every scale at once, how the points are arranged: where they gather into clusters, where they close up into loops, where they surround an empty region. Concretely, we do this by applying the *Rips filtration* to each point cloud. Around every point we draw a ball of radius r , and whenever the balls of two nearby points intersect, we connect the corresponding points. The radius starts at zero and is then increased: at the beginning every point

stands on its own, and as r grows the points progressively connect, forming the clusters, loops and cavities described above. For very large radius r all the points are connected in a single cluster: all the loops and cavities disappear. Persistent homology then detects these structures — by means of linear algebra, through the reduction of a matrix — and reports, for each topological feature (like a cluster, a loop, or a cavity), how long it survives as the radius grows.

A gravitational wave produces a pointcloud that winds into an elongated spiral: persistent homology detects the loops that the spiral encloses, and reports them as long-lived features. The cloud of a stretch of pure noise is instead a shapeless ball with little structure, so persistent homology reports that its clusters and loops are all short-lived features. The final result of persistent homology is a collection of numbers and images — we refer to them collectively as *topological summaries* — that summarise the shape of the cloud each time series produced.

Finally after studying several features, we identify which of them have more discriminating power, and use them to train the machine-learning classifiers. We consider a range of them, from logistic regression — a simple linear model — to convolutional networks, which requires images as input. What each of them learns, in the end, is a statement about shape: that a gravitational-wave signal arranges the points of its cloud in particular ways, gathering them into clusters and closing them into loops. Once trained, the machine-learning classifiers are shown time series they have never met before, and on these they detect the presence of a gravitational wave reliably.

Outline of the thesis

The thesis is structured as follows.

- In Chapter 1 we develop persistent homology from the beginning. We build the simplicial complexes out of which a shape is assembled, define what it means for two spaces to have the same shape, and construct homology as a way of counting holes by describing what surrounds them. We then let a complex grow, and follow the holes that appear and disappear as it does: this is persistence, and it is summarised by a barcode or a persistence diagram. We close with the algorithm that computes homology, run by hand on a small example.
- In Chapter 2 we turn that mathematics into a way of handling data. We show how a finite cloud of points can be turned into a complex, and when the complex we get faithfully reflects the shape the points were sampled from. We then prove the property on which everything else depends — that small perturbations of the points produce only small changes in the persistence diagram — and introduce the time-delay embedding, which is what turns a time series into a point cloud in the first place. Finally we reduce the persistence diagram to numbers a machine learning classifier can be given: scalar topological summaries, Betti curves, and persistence images.
- In Chapter 3 we run the whole construction, in Python, on a signal we control completely. Taking a cosine, a noise time series, and their sum, we follow each in turn from a raw time series to a topological summary, and check at every step that the computation does what the previous two chapters predict. We also measure what the computation costs, and how much of that cost can be removed, which becomes a practical concern once the same pipeline has to be run tens of thousands of times.
- In Chapter 4 we assemble the gravitational-wave physics the rest of the thesis needs. We recall the linearised theory, the two polarisations, and the chirp emitted by a coalescing binary; we then model the detector as a linear system, characterise its noise, and derive matched filtering as the optimal linear filter, together with the statistics used to decide whether a candidate is real. The chapter closes by drawing out the three assumptions on which that matched filtering construction rests, and showing where each of them fails — which is the motivation for everything that follows.
- In Chapter 5 we build the topological detection pipeline. We first construct a dataset intended to behave like real detector data, then run the pipeline of Chapter 3 on a single sample to see what a chirp does to the shape of the cloud, and finally on the whole dataset, extracting the topological features every classifier of the next chapter will use.

- In Chapter 6 we test the impact of those topological summaries. We train several machine learning classifiers on them, ask which features they rely on and why, measure how weak a signal this topological method can still find, and close by measuring what it costs against matched filtering as a function of the size of the template bank. We conclude by showing a strenght of this topological data analysis approach compared to match filtering: this topological method, once the machine learning classifiers are trained, gives a faster way to detect gravitational waves in stretches of data.

In the Conclusions we gather the main results and set out the directions in which this work could be continued.

Some material is collected in the appendices.

- Appendix A contains the proofs and constructions deferred from the main text, together with several derivations from the gravitational-wave and statistical chapters.
- Appendix B derives the pattern functions describing the angular response of an interferometer.
- Appendix C lists the code for the toy-model pipeline of Chapter 3.
- Appendix D lists the code that generates the gravitational-wave dataset.

Chapter 1

Persistent Homology

In recent decades, Topological Data Analysis (TDA) has established itself as an approach complementary to traditional statistical and machine learning methods, offering tools capable of capturing the shape and intrinsic structure of data, independently of the coordinate system or the specific metric used. Among the techniques developed in this field, persistent homology represents one of the most powerful and versatile tools, capable of identifying topological features — such as connected components, cycles, and voids — that persist across different scales of observation.

The underlying idea consists of building a family of topological spaces, parametrized by a scale threshold, and tracking the "birth" and "death" of topological features as this parameter varies. Features that survive over a wide range of scales are considered structurally significant, while short-lived ones are generally interpreted as noise.

This chapter will introduce the fundamental concepts necessary to understand persistent homology, starting from the basic notions of algebraic topology — simplicial complexes and simplicial homology — and the next chapter will proceed with the construction of filtered complexes (in particular the Vietoris-Rips and Čech complexes) and to the formal definition of persistence diagram, and barcode. These tools will constitute the theoretical foundation on which the analyses carried out in the following chapters are based.

1.1 Geometric preliminaries

1.1.1 Three notions of sameness

Before asking what shape a space has, we must decide when two spaces are to count as having the *same* shape. There is no single answer; there is a hierarchy of answers, each preserving less structure than the last. We take them in order of decreasing rigidity, assuming throughout that our spaces come equipped with a notion of distance between their points (a *metric* d).¹

Definition 1.1 (Isometry). Two spaces X, Y are **isometric** if Y is an exact rigid copy of X : every pair of distances in X reappears unchanged in Y . Precisely, a bijection $f: X \rightarrow Y$ exists such that, for every two points $x_1, x_2 \in X$, the distance between their images equals the distance between the points themselves,

$$d_Y(f(x_1), f(x_2)) = d_X(x_1, x_2).$$

Such an f is called an **isometry**, and X, Y are then **isometric**. Intuitively, Y is obtained from X only by moving it around — translating, rotating, reflecting — never by stretching, shrinking, or bending it.

Isometries of the plane are exactly the combinations of translations, rotations and reflections, as in Figure 1.1.

¹The formal definition of a metric is in Appendix A.1.

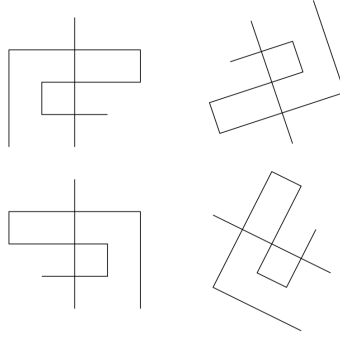


Figure 1.1: Four isometric planar sets [16]. Each is obtained from the others by a translation, a rotation or a reflection, so all pairwise distances agree. Image has been collected from [16]

Isometries are convenient, but too rigid for our purposes: stretching a circle into an ellipse destroys every distance, yet nothing we care about — it is still a single loop — has changed. We need a looser notion of sameness that only cares about that kind of qualitative shape.

Definition 1.2 (Homeomorphism). Two spaces X, Y are **homeomorphic**, written $X \cong Y$, if X can be continuously deformed into Y and back: a bijection $f: X \rightarrow Y$ exists such that both f and its inverse f^{-1} are continuous. Such an f is called a **homeomorphism**. Intuitively, f may bend, stretch, and shrink X , but it may never tear it apart or glue distinct points together.

It is easy to convince yourself that every isometry is a homeomorphism.² The converse fails, and that is precisely the point: homeomorphisms are far more flexible, and they preserve invariants of a space we shall meet later — dimension, the number of components, the number of holes — while discarding purely metric information such as diameter or area, which a homeomorphism is free to change.

Let’s define spheres, which will be useful in this chapter.

Example 1.1. For $n \in \mathbb{N}$, the n -**sphere** S^n is a space homeomorphic to the n -dimensional unit sphere in $\mathbb{R}^{n+1}$,

$$\{x \in \mathbb{R}^{n+1} : d_2(x, 0) = 1\} = \left\{ (x_1, \dots, x_{n+1}) \in \mathbb{R}^{n+1} \mid \sum_{i=1}^{n+1} x_i^2 = 1 \right\}, \quad (1.1)$$

i.e. any space obtained from that unit sphere by bending, stretching or shrinking it, without tearing or gluing.

In low dimensions, S^n recovers familiar shapes: S^0 consists of two points, S^1 is homeomorphic to a circle, and S^2 to the usual sphere, as Figure 1.2 shows.

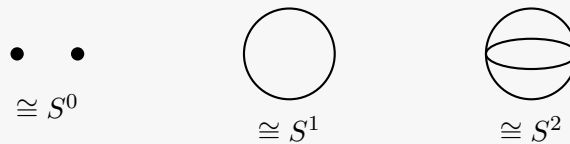


Figure 1.2: The spheres S^0 , S^1 and S^2 of Example 1.1: two points, a circle, and the usual sphere (drawn with an equator to suggest its two-dimensional surface).

The next example is the one to keep in mind whenever the word “homeomorphic” appears, because it makes visible what is and is not allowed.

Example 1.2 (What a homeomorphism may do). Consider the four filled planar regions of Figure 1.3: a banner-like shape, an irregular curved blob, a disc with a round hole punched out, and a square

²The verification is done in the appendix, see Appendix A.2.

with a square hole punched out. All four are homeomorphic to each other.

The reason is that a homeomorphism may bend, stretch and shrink, but it may not tear and it may not glue. Rounding the corners of the square, or straightening the curves of the blob, requires only bending; each of these deformations is invertible and continuous both ways. But no such deformation can close up a hole without gluing, or open one without tearing. *Each of the four regions has exactly one hole*, and that is what they have in common — each is homeomorphic to a standard annulus. None of them is homeomorphic to a filled disc, which do not have any hole.

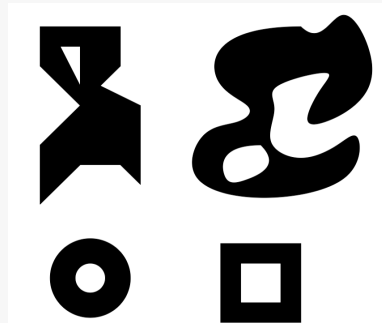


Figure 1.3: Four homeomorphic sets in the plane. They differ in every metric respect — area, diameter, smoothness of boundary — but each has exactly one hole, and that is all a homeomorphism preserves here. Image has been collected from [16]

Being homeomorphic is not merely a comparison between two spaces at a time: it sorts *all* spaces into families of “the same shape,” because homeomorphism is an equivalence relation. Reflexivity and symmetry follow at once from the definition³; transitivity takes a little more care: if $X \cong Y$ and $Y \cong Z$, then $X \cong Z$ [30]. The proof is a short composition-of-maps argument; see Appendix A.27.

Together, the three properties make “being homeomorphic” an equivalence relation, so we can sort all spaces into classes: class of circles, another class of intervals, another class of crosses, another class of 2-spheres; then come the tori, the Klein bottles, and so on (Figure 1.4). Within a class the members may look nothing alike, but the moral is: *to a topologist, two homeomorphic spaces are the same space*.

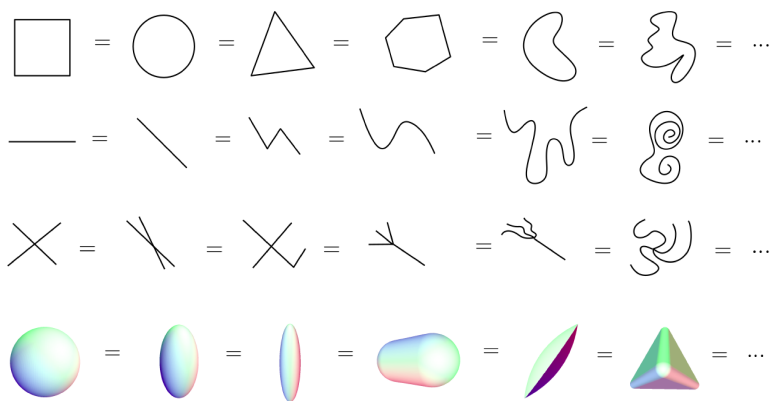


Figure 1.4: Classes of homeomorphism equivalence. Every space in a row is homeomorphic to every other space in that row; no space in one row is homeomorphic to a space in another. Image has been collected from [30].

Establishing whether two given spaces are homeomorphic can be a delicate matter [30]: a homeomorphism, if one exists, can be very hard to exhibit directly. Showing that 2 spaces are *not* homeomorphic is often easier, and the tool for that is the notion of an **invariant**: a quantity or property computed from a space that comes out the same for every space homeomorphic to it. If two spaces are computed to have *different* values of the same invariant, they cannot be homeomorphic — no homeomorphism could have changed that value.⁴

Homeomorphism is the focal equivalence of topology, but it is still finer than we actually need: many of the invariants we shall be able to compute do not change under a looser kind of continuous deformation,

³See Appendix A.27.

⁴The first invariant we shall meet, in just a few pages, is connectedness.

one that is allowed to squash a space down rather than only bend it. This last, coarsest notion of sameness is what we introduce now.

Definition 1.3 (Homotopy). Two continuous maps $f, g: X \rightarrow Y$ are **homotopic**, written $f \simeq g$, if f can be continuously morphed into g : there is a continuous map $F: X \times [0, 1] \rightarrow Y$, called a **homotopy**, which agrees with f at time 0 and with g at time 1,

$$F(\cdot, 0) = f, \quad F(\cdot, 1) = g.$$

$F(\cdot, t)$ can be pictured as a continuously changing snapshot: at each instant $t \in [0, 1]$ it is some map $X \rightarrow Y$, it starts as f , ends as g , and never jumps in between, see Example 1.3 to clarify this.

Example 1.3. Figure 1.5 shows a homotopy between two curves in the plane, drawn as five snapshots of the deforming map $F(\cdot, t)$ at $t = 0, 0.2, 0.5, 0.6, 1$.

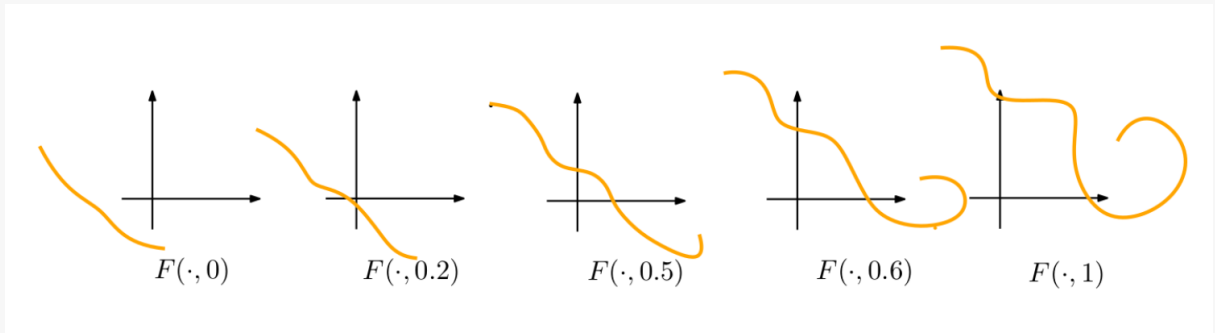


Figure 1.5: A homotopy between two curves in the plane, shown as five snapshots of $F(\cdot, t)$ as t runs from 0 to 1. Image has been collected from [30].

A single map can be homotoped to a constant map, in which case we call it **trivial** from a homotopy point of view.

Definition 1.4 (Homotopy equivalence). Two spaces X, Y are **homotopy equivalent**, written $X \simeq Y$, if each can be continuously deformed into the other and back to where it started. Precisely, there exist continuous maps $f: X \rightarrow Y$ and $g: Y \rightarrow X$ such that composing them in either order is homotopic to the identity map,

$$g \circ f \simeq \text{id}_X, \quad f \circ g \simeq \text{id}_Y.$$

The maps f, g are **homotopy equivalences**.

Example 1.4 (Homotopy equivalence). Figure 1.6 shows a circle X and an irregular wiggly loop Y , related by continuous maps f, g going back and forth between them. I can continuously deform the circle X into the wiggly loop Y by following f , and then deform it back to X by following g , so the circle X and the wiggly loop Y are homotopy equivalent.

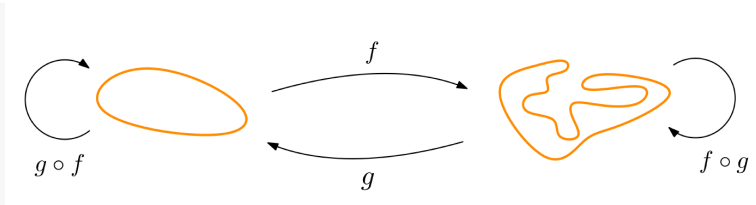


Figure 1.6: A homotopy equivalence between a circle and a wiggly loop. Image has been collected from [30].

A space homotopy equivalent to a single point is **contractible** — it can be continuously shrunk to a point.

Homeomorphic spaces are always homotopy equivalent, but the converse fails, and that failure is exactly what makes the notion useful.

Example 1.5 (Homotopy equivalent, not homeomorphic). Consider the interval $[0, 1]$ and the one-point space $\{0\}$:

- These two spaces are not *homeomorphic*, and the reason requires no topology at all: Definition 1.2 demands a *bijection*, and a bijection forces two sets to have the same number of elements, while $[0, 1]$ has infinitely many points and $\{0\}$ has exactly one. No such map exists, so the very first condition of the homeomorphism definition already fails.
- Yet the two spaces are *homotopy equivalent*: sliding every point of $[0, 1]$ continuously towards 0 is a homotopy from the identity map to the constant map at 0, so $[0, 1]$ is homotopy equivalent to $\{0\}$.

Homotopy equivalence does not preserve every topological property. Dimension, for instance, is lost: Example 1.5 already showed that the interval $[0, 1]$ and the one-point space $\{0\}$, spaces of different dimension, are homotopy equivalent all the same. But homotopy equivalence does preserve many of the invariants we can actually compute: the number of connected components, the number of holes, and so on. Counting those is exactly what the rest of this chapter is built to do.

1.1.2 Topological spaces and connected components

Our first invariant, the number of connected components, is built entirely out of “open sets,” so we start by defining the topological spaces [30].

Definition 1.5 (Topological space). A **topological space** is a pair $(X, \mathcal{T})$ where X is a set and $\mathcal{T}$ is a collection of subsets of X such that:

- $\emptyset \in \mathcal{T}$ and $X \in \mathcal{T}$;
- a union of open sets however many still belongs to $\mathcal{T}$.
- an intersection of finitely many open sets still belongs to $\mathcal{T}$.

Such a $\mathcal{T}$ is a **topology** on X , and its members are the **open sets** of the space.

Example 1.6. Take $X = \{0, 1\}$, a set with just two points. It carries exactly four topologies:

- $\mathcal{T}_1 = \{\emptyset, \{0, 1\}\}$,
- $\mathcal{T}_2 = \{\emptyset, \{0\}, \{0, 1\}\}$,
- $\mathcal{T}_3 = \{\emptyset, \{1\}, \{0, 1\}\}$,
- $\mathcal{T}_4 = \{\emptyset, \{0\}, \{1\}, \{0, 1\}\}$,

drawn in Figure 1.7. Each already contains $\emptyset$ and X , as Definition 1.5 demands, and each is small enough that every union and intersection of its members is checked in a moment. For instance, in $\mathcal{T}_2$:

- every union of members of $\mathcal{T}_2$ stays in $\mathcal{T}_2$, e.g. $\{0\} \cup \{0, 1\} = \{0, 1\} \in \mathcal{T}_2$;
- every intersection of members of $\mathcal{T}_2$ stays in $\mathcal{T}_2$, e.g. $\{0\} \cap \{0, 1\} = \{0\} \in \mathcal{T}_2$.

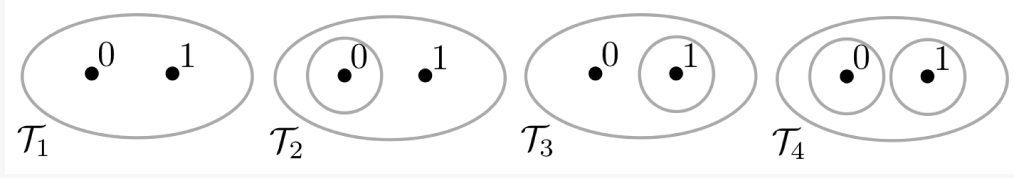


Figure 1.7: The four topologies on a two-point set. The outer oval is X itself and the inner circles are the other open sets. Image has been collected from [30].

On $\mathbb{R}^n$ we shall always use the *Euclidean* topology, the one where open sets are built out of open balls: the open ball of centre $x \in \mathbb{R}^n$ and radius $r > 0$, denoted $B(x, r)$, is defined as

$$B(x, r) = \{y \in \mathbb{R}^n : \|x - y\| < r\}.$$

⁵ A subset of $\mathbb{R}^n$ — a point cloud, an interval, a circle — inherits a topology of its own from this one, like we see in the following definition.

Definition 1.6 (Subspace topology). Let $(X, \mathcal{T})$ be a topological space and $Y \subset X$. Intuitively, Y borrows its open sets from X . Precisely, the **subspace topology on Y** is

$$\mathcal{T}_Y = \{O \cap Y : O \in \mathcal{T}\}.$$

Thanks to the subspace topology, any subset of $\mathbb{R}^n$ inherits a particular topology. This is the only topology we will consider on subsets of $\mathbb{R}^n$.

Definition 1.7 (Connected). A space is **connected** if it consists of a single piece: it is impossible to split it into two separate, non-touching open chunks. Precisely, a topological space $(X, \mathcal{T})$ is connected if, for every pair of disjoint open sets $O, O' \in \mathcal{T}$,

$$X = O \cup O' \implies O = \emptyset \text{ or } O' = \emptyset.$$

The example below shows this definition at work.

Example 1.7. Give $X = [0, 1] \cup [2, 3] \subset \mathbb{R}$ the subspace topology. It is not connected: the two pieces $[0, 1]$ and $[2, 3]$ are open in X , but they are disjoint, (and non-empty), so X is not connected.



Figure 1.8: The interval $[0, 1]$ (left), and the space $[0, 1] \cup [2, 3]$ (right), visibly falling apart into two separate pieces. Image has been collected from [30].

Intuitively, the **connected components** of a space are the separate pieces it falls into.⁶

⁵The formal definition of the Euclidean topology, built out of these balls, is in Appendix A.3.

⁶The formal definition of connected components is in Appendix A.5.

Example 1.8. Take $X = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\} \subset \mathbb{R}$ with the subspace topology. Every singleton $\{i\}$, $i \in X$, is open in X ; the singletons are non-empty, connected and disjoint. Hence X admits eleven connected components.

Connectedness as an invariant

Proposition 1.8. *Homeomorphic topological spaces have the same number of connected components.*

The proof can be found in Tinarrage [30].

Example 1.9. The interval $[0, 2\pi)$ and the circle $S^1 \subset \mathbb{R}^2$ are not homeomorphic. We argue by contradiction: suppose a homeomorphism $f: [0, 2\pi) \rightarrow S^1$ did exist, and see what goes wrong.

- Pick any point $x \neq 0$ in the interval, and remove it, together with its image $f(x)$ on the circle, see Figure 1.9. Restricting f this way still gives a homeomorphism, $[0, 2\pi) \setminus \{x\} \rightarrow S^1 \setminus \{f(x)\}$, since removing a point from both sides of a homeomorphism leaves a homeomorphism between what is left [30].
- But the two punctured spaces do not match, as Figure 1.9 shows: removing an interior point x splits the interval into two disconnected pieces, while removing a single point from the circle leaves it in one piece — a circle only breaks apart once two or more points are removed.
- So $[0, 2\pi) \setminus \{x\}$ has two connected components while $S^1 \setminus \{f(x)\}$ has only one, contradicting Proposition 1.8, which says a homeomorphism must preserve the number of connected components.

No homeomorphism f can exist, so the interval $[0, 2\pi)$ and the circle S^1 are not homeomorphic.

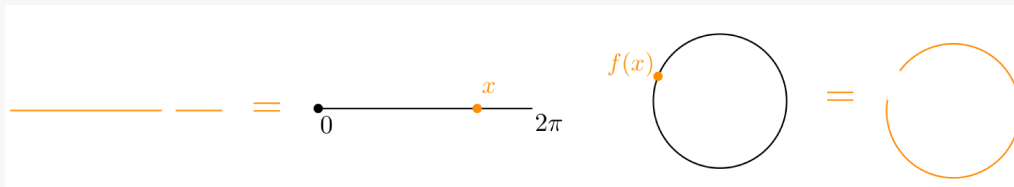


Figure 1.9: Removing an interior point disconnects the interval but not the circle. Since the number of connected components is preserved by homeomorphisms, the two spaces cannot be homeomorphic. Image has been collected from [30].

Remark 1.9. The number of connected components is therefore a topological invariant: homeomorphic spaces must agree on it because of Proposition 1.8.

1.2 Simplices and simplicial complexes

1.2.1 Abstract simplices

Definition 1.10 (Simplex). Given a finite vertex set V , any subset $\sigma \subseteq V$ of cardinality $k + 1$ is called a k -simplex.

A 0-simplex is a single vertex, a 1-simplex is an edge joining two vertices, and a 2-simplex is a triangular face spanned by three vertices; the same idea continues in higher dimensions, see Figure 1.10.



Figure 1.10: Low-dimensional simplices: a point, an edge, a triangle, and a tetrahedron. In the tetrahedron, the edges hidden behind the front face are dashed.

Definition 1.11 (Dimension of a simplex). The dimension of a simplex σ is $\dim(\sigma) = |\sigma| - 1$, where $|\sigma|$ is the number of vertices in σ . For example the simplex-tetrahedron in Figure 1.10 has four vertices, so its dimension is $4 - 1 = 3$ and we call it a 3-simplex.

The name “ k -simplex” is chosen so that the subscript matches the picture: a triangle has three vertices but is a 2-simplex, indeed the triangle spans a 2-dimensional space.

Example 1.10. The edge $\sigma = \{a, b\}$ has two vertices, so $\dim(\sigma) = 2 - 1 = 1$: it is a 1-simplex, matching the picture above. The triangle $\sigma = \{a, b, c\}$ has three vertices, so $\dim(\sigma) = 3 - 1 = 2$.

If $\tau \subseteq \sigma$ for two simplices τ, σ , we say:

- τ is a **face** of σ — every edge and vertex of a triangle is one of its faces;
- σ is a **coface** of τ — the triangle is a coface of each of them.

A single simplex only records one group of points. To describe a whole data set we need many simplices at once, glued together consistently: whenever a simplex is included, all of its faces must be included too, since it makes no sense to record a triangle without also recording its edges and vertices.

Definition 1.12 (Abstract simplicial complex). Let V be a finite vertex set. A collection K of simplices on V is an **abstract simplicial complex** if:

1. every point of V is itself a simplex of K , i.e. $\{v\} \in K$ for every $v \in V$ — so no vertex is left out of the complex;
2. K is closed under taking faces, i.e. if $\sigma \in K$ and $\tau \subseteq \sigma$ then $\tau \in K$ — exactly the requirement we motivated above: a complex cannot contain a simplex without also containing everything it is built from.

Example 1.11. Let $V = \{0, 1, 2\}$ and

$$K = \{\{0\}, \{1\}, \{2\}, \{0, 1\}, \{1, 2\}, \{0, 2\}\}.$$

This is an abstract simplicial complex: every vertex of V is included as a 0-simplex, and every edge’s two endpoints are included too, so both conditions of Definition 1.12 hold.

Example 1.12. Let $V = \{0, 1, 2\}$ and

$$K = \{\{0\}, \{1\}, \{2\}, \{0, 1\}, \{1, 2\}, \{0, 1, 2\}\}.$$

This is *not* an abstract simplicial complex: the triangle $\{0, 1, 2\}$ lies in K , but its face $\{0, 2\}$ does not, so condition 2 of Definition 1.12 fails.

Up to now an abstract complex is nothing but a list of finite sets. The next step is to give it a shape: a natural way to draw it in space.

1.2.2 Geometric simplicial complexes

So far a simplicial complex is a purely combinatorial object: a list of finite sets, with no topology of its own. But there is a natural way to draw one: place its vertices V in the plane, or in space, and for every simplex of K fill in the convex hull of its vertices. The next example spells out exactly what “filling in the convex hull” means.

Example 1.13. Take again the vertex set $V = \{0, 1, 2\}$ and the abstract simplicial complex K : $\{0, 1, 2\}$:

$$K = \{\{0\}, \{1\}, \{2\}, \{0, 1\}, \{1, 2\}, \{0, 2\}, \{0, 1, 2\}\}.$$

To draw the complex K , we place its three vertices at explicit coordinates in the plane, say

$$0 \mapsto (0, 0), \quad 1 \mapsto (2, 0), \quad 2 \mapsto (1, 1.73).$$

We then fill in the convex hull of each simplex’s vertices in turn:

- for the 1-simplex $\{0, 1\}$, we take its two vertices, the points $(0, 0)$ and $(2, 0)$, and fill in their convex hull — the straight segment joining them — giving an edge embedded in $\mathbb{R}^2$ running from $(0, 0)$ to $(2, 0)$;
- the 1-simplices $\{1, 2\}$ and $\{0, 2\}$ give the other two edges the same way;
- for the 2-simplex $\{0, 1, 2\}$, we take all three vertices and fill in their convex hull, which is the solid filled triangle they span, not merely its outline.

Figure 1.11 shows the result.

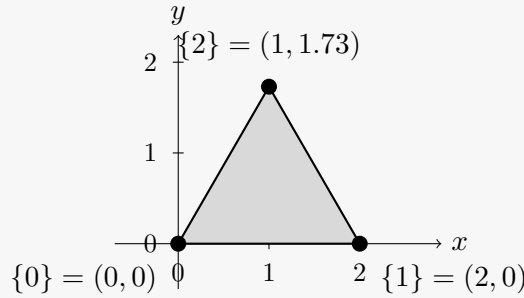


Figure 1.11: Drawing the complex of Example 1.13: each vertex is placed at its coordinates in the plane, each edge is the convex hull of its two endpoints, and the 2-simplex $\{0, 1, 2\}$ is the convex hull of all three vertices — the filled triangle, shaded, not just its boundary.

Definition 1.13 (Geometric simplicial complex). What we have just done is very natural: we passed from an abstract complex to a **geometric** one, and in a geometric complex the vertices carry real coordinates in $\mathbb{R}^n$. In other words, a geometric simplicial complex is exactly what an abstract complex looks like once embedded in $\mathbb{R}^n$: the abstract 1-simplex $\{1, 2\}$, say, which was just a pair of labels, becomes a genuine edge embedded in $\mathbb{R}^n$, running from the point labelled 1 to the point labelled 2.^a

^aThe formal definitions of a geometric simplex and a geometric simplicial complex are in Appendix A.7.

Passing from a geometric complex to its abstract counterpart costs nothing: replace each vertex, given by its coordinates, by a label, and read off the simplices, as in the Figure 1.12. The reverse direction is the harder one, since coordinates must now be invented, and invented in a way that meets the two conditions of Definition 1.12: every vertex present as a 0-simplex, and the whole collection closed under taking faces. When this succeeds, the resulting geometric complex is called a **geometric realization** of the abstract one. Realizations always exist; what is awkward is obtaining them in a space of low dimension. Two bounds are worth recording.



Figure 1.12: A geometric simplicial complex on labelled vertices a, b, c (left), redrawn with its coordinates dropped (right). Replacing the points by labels leaves the abstract complex $\{\{a, b, c\}, \{a, b\}, \{b, c\}, \{a, c\}, \{a\}, \{b\}, \{c\}\}$.

Theorem 1.14 ([16]). *Every abstract simplicial complex K with n vertices admits a geometric realization in $\mathbb{R}^{n-1}$.*

The proof can be found in [16].

A geometric simplicial complex is, by construction, a subset of $\mathbb{R}^n$ (Definition A.10), so like any other subset of $\mathbb{R}^n$ it inherits a topology of its own: the subspace topology of Definition 1.6.

Definition 1.15 (Triangulation). Intuitively, a **triangulation** of a topological space is a decomposition of the space into simplices, assembled into a simplicial complex whose geometric picture recovers the space (up to homeomorphism).^a

^aThe formal definition is in Appendix A.9.

Example 1.14. The simplicial complex K :

$$K = \{\{0\}, \{1\}, \{2\}, \{0, 1\}, \{1, 2\}, \{0, 2\}\}$$

is the triangulation of the circle S^1 . Drawn geometrically, K is a triangle (Figure 1.13), and a triangle can be continuously deformed into a circle — push each edge outward until it bulges into an arc — with a continuous inverse doing the opposite. This deformation is a homeomorphism, so the triangle and the circle are homeomorphic, and K is indeed a triangulation of the circle.

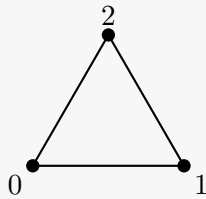


Figure 1.13: The complex K of Example 1.14, drawn geometrically: since K contains no 2-simplex, only its three edges are filled, giving the outline of a triangle rather than a solid one.

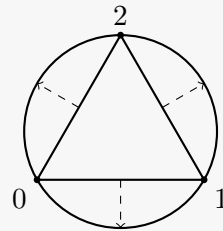


Figure 1.14: The triangulation K of Example 1.14, deformed continuously from a triangle into the circle it triangulates.

Example 1.15. The simplicial complex

$$K = \{\{0\}, \{1\}, \{2\}, \{3\}, \{0, 1\}, \{1, 2\}, \{2, 3\}, \{3, 0\}, \{0, 2\}, \{1, 3\}, \\ \{0, 1, 2\}, \{0, 1, 3\}, \{0, 2, 3\}, \{1, 2, 3\}\}$$

is a triangulation of the sphere: its four triangular faces are exactly the four faces of a tetrahedron, and Figure 1.15 shows why the tetrahedron itself is homeomorphic to a sphere — the same push-the-

faces-out deformation used for the circle in Figure 1.14, only now in three dimensions.

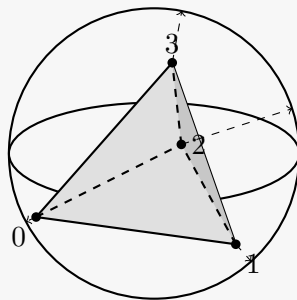


Figure 1.15: The tetrahedron of Example 1.15, seen from an oblique angle so that all four vertices are visible (vertex 2 tucked behind the front face, with its edges $\{0, 2\}$, $\{1, 2\}$ and $\{2, 3\}$ drawn dashed), inscribed in a sphere and pushed radially outward, one vertex at a time, until its four flat faces bulge into the round surface of the sphere — the three-dimensional analogue of Figure 1.14. This deformation is a homeomorphism, so the tetrahedron and the sphere S^2 are homeomorphic, and K is indeed a triangulation of the sphere.

Not every topological space can be triangulated. When one can, it admits not a single triangulation but a great many: the three complexes of Figure 1.16, for instance, all triangulate the circle, and one could go on refining indefinitely. Nothing distinguishes a preferred one, which is worth keeping in mind.

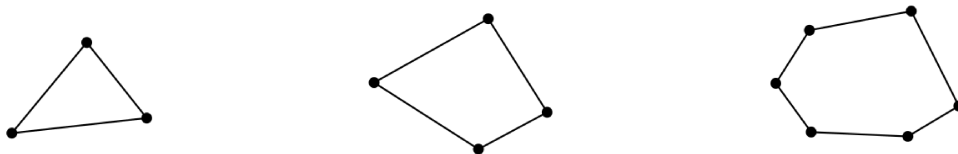


Figure 1.16: Three different triangulations of the circle. A space that can be triangulated at all can be triangulated in many ways. Image has been collected from [30].

1.3 Chains

So far a simplicial complex K only lets us say whether a given simplex belongs to it or not. To eventually define holes, we need before to define a way to sum simplices of the same dimension.

Definition 1.16 (Chain group). Let K be a simplicial complex. A **q -chain** c of the complex K is a formal sum of the q -simplices of K ,

$$c = \sum_{\sigma} \epsilon_{\sigma} \cdot \sigma, \quad \sigma \text{ ranging over the } q\text{-simplices of } K, \quad \epsilon_{\sigma} \in F = \mathbb{Z}/2\mathbb{Z}, q \geq 0,$$

where each coefficient ϵ_{σ} is only ever 0 or 1: a simplex is either part of the sum, or it is not.^a The set of all q -chains of K is written $C_q(K; F)$.

^aCoefficients are taken in the two-element field $F = \mathbb{Z}/2\mathbb{Z}$, where 0 and 1 are the only elements and $1 + 1 = 0$.

Example 1.16. Consider the simplicial complex

$$K = \{\{0\}, \{1\}, \{2\}, \{0, 1\}, \{0, 2\}\}.$$

The 0-chains of K are the linear combinations of its 0-simplices, i.e. of the three vertices $\{0\}, \{1\}, \{2\}$: since each vertex is independently either included in the sum or not, there are $2^3 = 8$ possible 0-chains,

$$C_0(K; F) = \{0, \{0\}, \{1\}, \{2\}, \{0\} + \{1\}, \{0\} + \{2\}, \{1\} + \{2\}, \{0\} + \{1\} + \{2\}\}.$$

Figure 1.17 draws each of these eight chains.

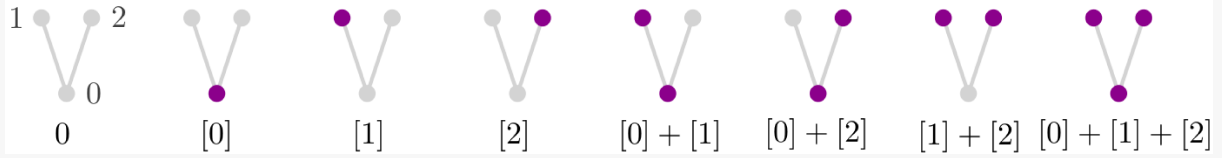


Figure 1.17: The eight 0-chains of Example 1.16, each drawn by marking the vertices it contains. One of the eight has nothing marked at all: this is the **null chain** 0, the chain in which every coefficient ϵ_σ is 0. It is not a peculiarity of this example — every chain group, at every dimension, contains this same null chain, not just $C_0(K; F)$. Image has been collected from [30].

Likewise, the 1-chains of K are the linear combinations of its 1-simplices, i.e. of the two edges $\{0, 1\}$ and $\{0, 2\}$: with two edges, each independently included or not, there are $2^2 = 4$ possible 1-chains,

$$C_1(K; F) = \{0, \{0, 1\}, \{0, 2\}, \{0, 1\} + \{0, 2\}\}.$$

Figure 1.18 draws each of these four chains.

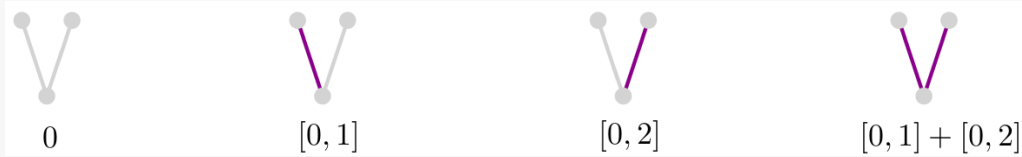


Figure 1.18: The four 1-chains of Example 1.16, each drawn by marking the edges it contains. Image has been collected from [30].

Our coefficients live in the field $F = \mathbb{Z}/2\mathbb{Z}$, this field has two elements (0 and 1) in which $1 + 1 = 0$. So adding any chain to itself cancels every one of its coefficients, one at a time, and the result is the null chain: every q -chain $\mathbf{c} \in C_q(K; F)$ satisfies $\mathbf{c} + \mathbf{c} = 0$.

The boundary operator.

Example 1.17. Consider the simplicial complex

$$K = \{\{0\}, \{1\}, \{2\}, \{3\}, \{0, 1\}, \{0, 2\}, \{1, 2\}, \{0, 3\}, \{2, 3\}, \{0, 1, 2\}\}.$$

The simplex $\{0, 1\}$ has the faces $\{0\}$ and $\{1\}$. Intuitively, the **boundary** of an edge would be its two endpoints, namely we expect the boundary of $\{0, 1\}$ to return us $\{0\} + \{1\}$:

$$\partial_1\{0, 1\} = \{0\} + \{1\}.$$

Figure 1.19 draws this computation.

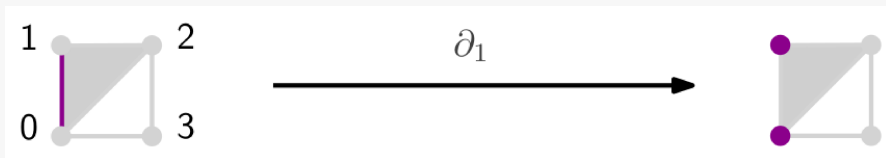


Figure 1.19: The boundary map ∂_1 applied to the edge $\{0, 1\}$ of Example 1.17, giving $\partial_1\{0, 1\} = \{0\} + \{1\}$. Image has been collected from [30].

Definition 1.17 (Boundary operator). Let $q \geq 1$ and let σ be a q -simplex. The **boundary** $\partial_q\sigma$ of σ

is the $(q-1)$ -chain obtained by deleting, one at a time, each single vertex of σ :

$$\partial_q \sigma = \sum_{\substack{\tau \subset \sigma \\ |\tau|=|\sigma|-1}} \tau.$$

There is nothing one dimension below a vertex, so by convention ∂_0 sends every 0-chain to the null chain 0^a .

^aAnd we set $C_{-1}(K; F) = \{0\}$

Example 1.18 (Continuing Example 1.17). So far we only know the boundary of a single simplex. To take the boundary of a whole chain, simply take the boundary of each simplex appearing in it. For instance, the boundary of the 1-chain $\{0, 1\} + \{1, 2\} + \{0, 2\}$ is found by taking the boundary of each of its three edges and adding the results:

$$\begin{aligned} \partial_1(\{0, 1\} + \{1, 2\} + \{0, 2\}) &= \partial_1\{0, 1\} + \partial_1\{1, 2\} + \partial_1\{0, 2\} \\ &= \{0\} + \{1\} + \{1\} + \{2\} + \{0\} + \{2\} \\ &= 0, \end{aligned}$$

since every vertex occurs twice and cancels, $\{0\} + \{0\} = \{1\} + \{1\} = \{2\} + \{2\} = 0$ in $C_0(K; F)$.

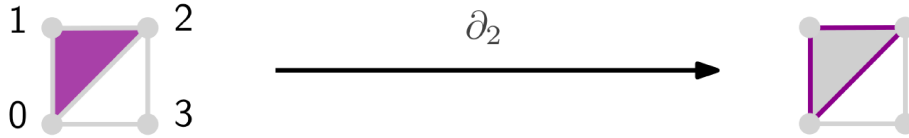


Figure 1.20: Taking the boundary of the filled triangle $\{0, 1, 2\}$ (in purple in the left figure): applying ∂_2 to it gives its three edges, $\{0, 1\} + \{1, 2\} + \{0, 2\}$ (in purple in the right figure). Image has been collected from [30].

Cycles. Some chains have no loose ends: applying the boundary operator to them returns zero 0. These chains are given a name.

Definition 1.18 (Cycles). Intuitively, a chain is a **cycle** if it has no loose ends: its boundary is 0. Precisely, the q -cycles of K are those q -chains sent to 0 by the boundary map,

$$Z_q(K; F) = \ker(\partial_q) = \{c \in C_q(K; F) : \partial_q c = 0\}.$$

Example 1.19 (Cycles). Consider again the complex K of Example 1.11, drawn in Figure 1.21, and its 1-chain $\{0, 1\} + \{1, 2\} + \{0, 2\}$ — the sum of all three edges of the triangle. Its boundary is

$$\begin{aligned} \partial_1(\{0, 1\} + \{1, 2\} + \{0, 2\}) &= \partial_1\{0, 1\} + \partial_1\{1, 2\} + \partial_1\{0, 2\} \\ &= (\{0\} + \{1\}) + (\{1\} + \{2\}) + (\{0\} + \{2\}) \\ &= (\{0\} + \{0\}) + (\{1\} + \{1\}) + (\{2\} + \{2\}) \\ &= 0, \end{aligned}$$

so $\{0, 1\} + \{1, 2\} + \{0, 2\}$ is a 1-cycle: every vertex cancels in the sum^a and no loose end is left over.

^aSince coefficients are such that $1 + 1 = 0$, as we discussed in 1.3

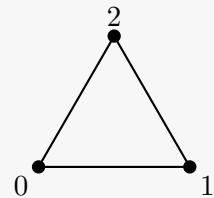


Figure 1.21: The complex K of Example 1.11, drawn again for Example 1.19: the 1-chain $\{0, 1\} + \{1, 2\} + \{0, 2\}$ runs all the way around the triangle, with no loose end.

Boundaries. Some chains arise naturally as the edge of a higher-dimensional complex: they are, quite literally, boundaries.

Definition 1.19 (Boundaries). Intuitively, a chain is a **boundary** if it is the boundary of some chain one dimension up. Precisely, the q -**boundaries** of K are the chains that arise as the boundary of some $(q + 1)$ -chain,

$$B_q(K; F) = \text{im}(\partial_{q+1}) = \{\partial_{q+1}\mathbf{c} : \mathbf{c} \in C_{q+1}(K; F)\}.$$

Example 1.20 (Boundaries). Let $K = \{\{0\}, \{1\}, \{2\}, \{0, 1\}, \{1, 2\}, \{0, 2\}, \{0, 1, 2\}\}$, the filled triangle on vertices 0, 1, 2 (Figure 1.22).

The chain $\{0, 1\} + \{1, 2\} + \{0, 2\}$ is also a 1-boundary of K : it is exactly the boundary of the filled triangle $\{0, 1, 2\}$,

$$\partial_2\{0, 1, 2\} = \{0, 1\} + \{1, 2\} + \{0, 2\}.$$

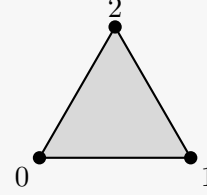


Figure 1.22: The filled triangle of Example 1.20. Its boundary, the sum of its three edges, is exactly the cycle already met in Example 1.19.

Here is a key property of the boundary operator: applying it twice in a row always gives zero.

Lemma 1.20. Take any chain and apply the boundary operator to it twice in a row: the result is always the null chain. Precisely, for any $q \geq 0$ and any q -chain $\mathbf{c} \in C_q(K; F)$,

$$\partial_{q-1} \circ \partial_q(\mathbf{c}) = 0.$$

A formal proof of this fact is given by [30]; a fully general version, in our own notation, is worked out in Appendix A.11. To build some intuition for why it is true, it helps to see it happen on a single example.

Example 1.21. Let us see why applying the boundary operator twice to the filled triangle $\{0, 1, 2\}$ gives zero (Figure 1.23). Applying ∂_2 once gives its three edges,

$$\partial_2\{0, 1, 2\} = \{0, 1\} + \{1, 2\} + \{0, 2\},$$

and applying ∂_1 to that — exactly the computation of Example 1.19 — gives

$$\partial_1(\{0, 1\} + \{1, 2\} + \{0, 2\}) = 0.$$

So $\partial_1 \circ \partial_2\{0, 1, 2\} = 0$: every vertex of the triangle is the endpoint of two of its edges, and is therefore picked up twice by the composite map — and two copies of the same vertex cancel, since $1 + 1 = 0$ in $F = \mathbb{Z}/2\mathbb{Z}$. You can see grafically this computation in 1.23.

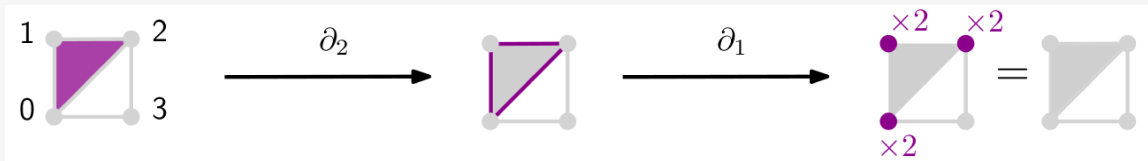


Figure 1.23: Applying $\partial_1 \circ \partial_2$ to the filled triangle $\{0, 1, 2\}$: every vertex of the triangle is hit twice by the composite map, and $2 = 0$ in $F = \mathbb{Z}/2\mathbb{Z}$, so the result is the zero chain. Image has been collected from [30].

Corollary 1.21. *Every boundary has no loose ends of its own, and is therefore itself a cycle: $B_q(K; F) \subset Z_q(K; F)$.*

This is exactly what happened in Example 1.21:

- the chain $\{0, 1\} + \{1, 2\} + \{0, 2\}$ is the boundary of the filled triangle, obtained by applying ∂_2 to it;
- that same chain $\{0, 1\} + \{1, 2\} + \{0, 2\}$ is also a cycle, since its own boundary ∂_1 came out to be 0;
- this is no coincidence: applying the boundary operator twice in a row always gives 0.

The proof is a direct consequence of Lemma 1.20, and is given in Appendix A.12.

Corollary 1.21 says more than just that boundaries and cycles overlap: at every dimension q , the boundaries sit inside the cycles, and the cycles in turn sit inside the full chain group,

$$B_q(K; F) \subset Z_q(K; F) \subset C_q(K; F).$$

The boundary maps ∂_q link these nested groups across dimensions to one another, and Figure 1.24 draws the whole picture at once: at each dimension q , a chain group with a cycle group nested inside it, and a boundary group nested inside that, all connected to their neighbouring dimensions by the boundary maps. This whole structure is called the **chain complex**.

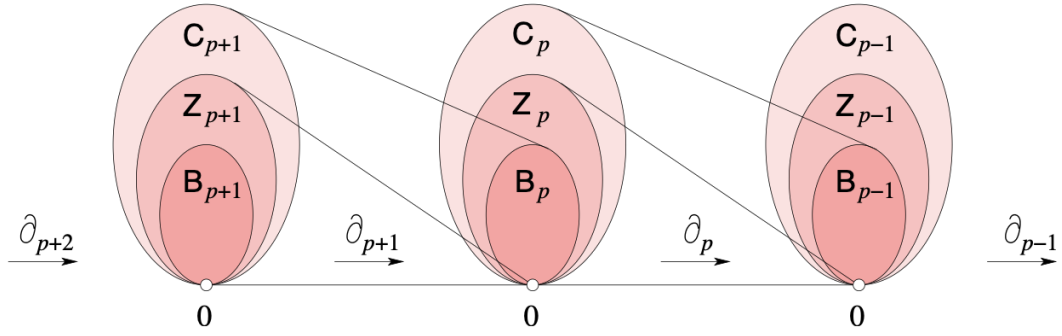


Figure 1.24: The chain complex: the chain, cycle and boundary groups at each dimension, connected across dimensions by the boundary homomorphisms ∂_q . Image has been collected from [31].

1.3.1 Oriented simplices

So far every chain has been built from unoriented simplices over $F = \mathbb{Z}/2\mathbb{Z}$, where a simplex and its own negative are literally the same thing, so no notion of sign was ever needed. Working over a general coefficient field F call for simplices to carry a sign first.

Example 1.22. Take the triangle $\{x, y, z\}$ of Figure 1.25. Walking around its three vertices in the order x, y, z traces it out counterclockwise; starting the walk at a different vertex but keeping the same direction, y, z, x or z, x, y , traces out exactly the same walk, so all three orderings are declared equal:

$$\langle x, y, z \rangle = \langle y, z, x \rangle = \langle z, x, y \rangle.$$

Swapping any two of the three vertices reverses the direction of the walk to clockwise, and this opposite walk is declared to be the negative of the first:

$$\langle y, x, z \rangle = \langle x, z, y \rangle = \langle z, y, x \rangle = -\langle x, y, z \rangle.$$

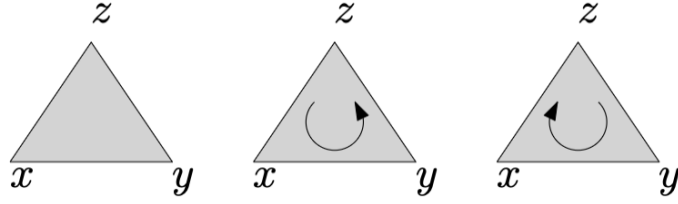


Figure 1.25: The triangle $\{x, y, z\}$ (left) and its two orientations (centre, right): cyclic relabelling leaves an orientation unchanged, while swapping any two vertices reverses it. Image has been collected from [16].

Definition 1.22 (Oriented simplex). Generalizing the example above: an **oriented simplex** is a simplex together with a choice of direction to walk around its vertices, where two orderings that walk the same way are considered the same oriented simplex, and the reverse walk is its negative. Writing an oriented simplex as an ordered tuple $\sigma = \langle v_0, v_1, \dots, v_k \rangle$, this means that reordering its vertices by a permutation π leaves it unchanged if π is even, and flips its sign if π is odd [32]:

$$\sigma = \text{sgn}(\pi) \langle v_{\pi(0)}, v_{\pi(1)}, \dots, v_{\pi(k)} \rangle.$$

A single vertex v , a 0-dimensional simplex, has nothing to walk around, but can still be oriented in two ways, written $\langle v \rangle$ and $-\langle v \rangle$.

Orienting σ does more than fix its own sign: the choice also fixes a sign on every one of its facets, which is exactly what lets the boundary operator carry signs of its own, rather than just recording presence or absence.⁷ Given it, the boundary of an oriented q -simplex is the signed sum of its facets, each carrying the sign it inherits from σ — the natural extension of Definition of boundary operator 1.17 to a general coefficient field, and the two agree over $F = \mathbb{Z}/2\mathbb{Z}$, where every sign equals 1.

Definition 1.23 (Boundary matrix). The boundary map ∂_q is represented, in the basis of oriented simplices, by a matrix M_q with entries in F :

- columns are indexed by the oriented q -simplices of K ;
- rows are indexed by the oriented $(q-1)$ -simplices of K ;
- the entry at position (i, j) is $+1$ or -1 according to the sign with which the i -th row's simplex appears in the boundary of the j -th column's simplex, and 0 if it does not appear at all.

This definition is easier to follow once seen in practice; Example 1.23 below works it out explicitly for a specific complex.

Example 1.23 (Boundary matrix). Consider the abstract simplicial complex

$$K = \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{a, b\}, \{b, c\}, \{a, c\}, \{b, d\}, \{c, d\}, \{a, b, c\}\}$$

of Figure 1.26.

⁷The precise rule for this induced orientation is worked out in Appendix A.13.

It consists of a filled triangle on a, b, c , an unfilled triangle on b, c, d sharing the edge $\{b, c\}$, and an isolated vertex e . To see where the matrix M_2 representing ∂_2 comes from, start from how ∂_2 acts on the single 2-simplex $\langle a, b, c \rangle$ of K :

$$\partial_2 \langle a, b, c \rangle = \mathbf{1} \cdot \langle a, b \rangle + \mathbf{1} \cdot \langle b, c \rangle + (-\mathbf{1}) \cdot \langle a, c \rangle.$$

These three coloured coefficients are exactly the entries of M_2 : one column, indexed by $\langle a, b, c \rangle$, and one row for every oriented 1-simplex of K , with the entry in each row equal to the coefficient it carries above, coloured to match, and 0 for every edge that does not appear in $\partial_2 \langle a, b, c \rangle$ at all:

$$\partial_2 = M_2 = \begin{bmatrix} & \langle a, b, c \rangle \\ \langle a, b \rangle & \mathbf{1} \\ \langle b, c \rangle & \mathbf{1} \\ \langle a, c \rangle & -\mathbf{1} \\ \langle b, d \rangle & 0 \\ \langle c, d \rangle & 0 \end{bmatrix}.$$

The matrix M_1 representing ∂_1 is built the same way, now reading off, column by column, how ∂_1 acts on each of the five edges of K in turn:

$$\partial_1 = M_1 = \begin{bmatrix} & \langle a, b \rangle & \langle b, c \rangle & \langle a, c \rangle & \langle b, d \rangle & \langle c, d \rangle \\ \langle a \rangle & -1 & 0 & -1 & 0 & 0 \\ \langle b \rangle & 1 & -1 & 0 & -1 & 0 \\ \langle c \rangle & 0 & 1 & 1 & 0 & -1 \\ \langle d \rangle & 0 & 0 & 0 & 1 & 1 \\ \langle e \rangle & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

There is no matrix M_0 to write down: ∂_0 is just the zero map into $\{0\}$ (Definition 1.17), so its matrix would have no rows at all.

Computing a boundary is then just matrix-vector multiplication: multiply M_q by the coordinate vector of a chain in that same basis. For instance, take the 1-chain $\mathbf{c} = \langle a, b \rangle + \langle b, c \rangle$, whose coordinate vector in the column order of M_1 is $(1, 1, 0, 0, 0)^\top$. Multiplying,

$$\partial_1 \mathbf{c} = \partial_1 (\langle a, b \rangle + \langle b, c \rangle) = M_1 \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} -1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = -\langle a \rangle + \langle c \rangle,$$

recovers exactly the boundary of the path from a to c through b : it starts at a (with a minus sign) and ends at c , with b cancelling out along the way.

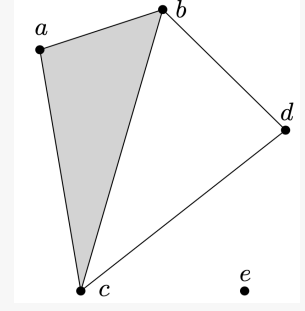


Figure 1.26: The abstract simplicial complex K of Example 1.23 [16]: the triangle a, b, c is filled, the triangle b, c, d is not, and e is an isolated vertex. Image has been collected from [16].

1.4 Holes: homology groups and Betti numbers

What we ultimately want to count is the number of holes of a complex. Before building the machinery, it is worth saying plainly what a hole is, and the complex of Example 1.23 already contains everything needed to see it.

Look at the unfilled triangle on b, c, d in Figure 1.27. Walking around it traces the chain $\langle b, c \rangle + \langle c, d \rangle - \langle b, d \rangle$, and this chain is a cycle: its boundary vanishes. Indeed, every vertex picked up along the walk is picked up twice with opposite signs, and so cancels:

$$\partial_1(\langle b, c \rangle + \langle c, d \rangle - \langle b, d \rangle) = (\cancel{\langle c \rangle} - \cancel{\langle b \rangle}) + (\cancel{\langle d \rangle} - \cancel{\langle c \rangle}) - (\cancel{\langle d \rangle} - \cancel{\langle b \rangle}) = 0.$$

A closed loop has no loose ends, so its boundary is zero — that is exactly what being a cycle means.

But being a cycle is not yet enough to be a hole. A hole, quite literally, is a loop with *nothing inside it*. The loop around a, b, c makes the difference visible. It is a cycle too, by the same computation:

$$\partial_1(\langle a, b \rangle + \langle b, c \rangle - \langle a, c \rangle) = (\cancel{\langle b \rangle} - \cancel{\langle a \rangle}) + (\cancel{\langle c \rangle} - \cancel{\langle b \rangle}) - (\cancel{\langle c \rangle} - \cancel{\langle a \rangle}) = 0.$$

Yet the region it encloses is not empty: the 2-simplex $\langle a, b, c \rangle$ sits inside it, filling it in. There is no hole there, and the formal signature of this is that the loop is precisely the boundary of that triangle,

$$\partial_2 \langle a, b, c \rangle = \langle a, b \rangle + \langle b, c \rangle - \langle a, c \rangle.$$

The loop around b, c, d , by contrast, is the boundary of nothing: K contains no 2-simplex on b, c, d to fill it.

This is the distinction we were after. A hole is a cycle that is *not* the boundary of anything one dimension up. To count holes, then, we should take all the cycles and discard those that are boundaries — and this is exactly a quotient of groups. The homology group is, in this precise sense, the group of holes.

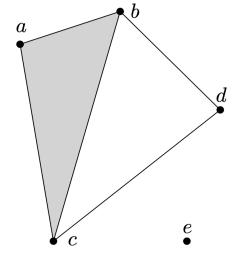


Figure 1.27: The complex of Example 1.23 once more: the loop around b, c, d is empty and counts as a hole, while the loop around a, b, c is filled by the triangle $\langle a, b, c \rangle$ and does not. Image has been collected from [16].

Definition 1.24 (Homology group). The q -th **homology group** of K is the quotient of the cycles by the boundaries^a,

$$H_q(K; F) = Z_q(K; F) / B_q(K; F) = \ker(\partial_q) / \text{im}(\partial_{q+1}).$$

Intuitively, two cycles that differ only by a boundary are considered to enclose “the same hole,” and homology is the construction that identifies them: it is what remains of the cycles once every boundary has been collapsed to nothing.

^aEvery boundary is a cycle (Corollary 1.21), namely $B_q(K; F)$ sits inside $Z_q(K; F)$ as a linear subspace. This is exactly the situation in which we can quotient one by the other.

Thanks to a general property of finite vector spaces — quotienting by a subspace simply subtracts dimensions — the dimension of this quotient is easy to get at:

$$\dim H_q(K; F) = \dim Z_q(K; F) - \dim B_q(K; F).$$

The proof of this relation is given in Appendix A.10.

Example 1.24. Consider the simplicial complex

$$K = \{\{0\}, \{1\}, \{2\}, \{3\}, \{0, 1\}, \{0, 2\}, \{1, 2\}, \{0, 3\}, \{2, 3\}, \{0, 1, 2\}\}$$

of Figure 1.28.

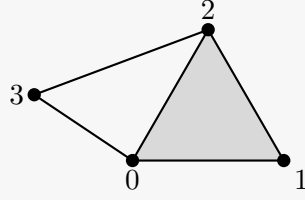


Figure 1.28: The complex K of Example 1.24.

The set of 1-cycles $Z_1(K; F)$ consists of the chains

$$0, \quad \{0, 1\} + \{1, 2\} + \{0, 2\}, \quad \{0, 2\} + \{2, 3\} + \{0, 3\}, \quad \{0, 1\} + \{1, 2\} + \{2, 3\} + \{0, 3\}.$$

Each of these does indeed have zero boundary, as required of a cycle — and this is no surprise once we look at Figure 1.28: apart from the null chain, these are exactly the closed loops traced out by K .

The only boundaries $B_1(K; F)$ are

$$\partial_2(0) = 0 \quad \text{and} \quad \partial_2(\{0, 1, 2\}) = \{0, 1\} + \{1, 2\} + \{0, 2\}.$$

This too is visible directly in the figure: $\{0, 1\} + \{1, 2\} + \{0, 2\}$ is exactly the boundary of the filled triangle $\{0, 1, 2\}$.

We now have everything needed for the homology group $H_1(K; F) = Z_1(K; F)/B_1(K; F)$. $Z_1(K; F)$ has $4 = 2^2$ elements and $B_1(K; F)$ has $2 = 2^1$ elements; a finite $\mathbb{Z}/2\mathbb{Z}$ -vector space with 2^d elements has dimension d [30], so $\dim Z_1(K; F) = 2$ and $\dim B_1(K; F) = 1$, giving

$$\dim H_1(K; F) = \dim Z_1(K; F) - \dim B_1(K; F) = 2 - 1 = 1.$$

$\dim H_1(K; F) = 1$ means that there is really only one hole in K : this hole is the loop $\{0, 2\} + \{2, 3\} + \{0, 3\}$ running around the outside of K , through vertex 3 in figure 1.28. The other loop we listed, $\{0, 1\} + \{1, 2\} + \{0, 2\}$, is not a hole at all — its interior is filled by the triangle $\{0, 1, 2\}$, as is plain from Figure 1.28.^a

^aOne might reasonably ask whether the fourth cycle, $\{0, 1\} + \{1, 2\} + \{2, 3\} + \{0, 3\}$, is a genuine hole of its own. It is not: this cycle turns out to be the “same” hole as $\{0, 2\} + \{2, 3\} + \{0, 3\}$. The two are what is called **homologous**, and the reason is worked out in Appendix A.15.

Definition 1.25 (Betti number). Intuitively, the q -th **Betti number** of K counts the number of independent q -dimensional holes of K : it is simply the dimension of the homology group we just built,

$$\beta_q(K) = \dim H_q(K; F) = \dim Z_q(K; F) - \dim B_q(K; F) = \dim \ker(\partial_q) - \dim \text{im}(\partial_{q+1}).$$

Example 1.25 (Betti numbers). Consider the simplicial complex

$$K = \{\{0\}, \{1\}, \{2\}, \{3\}, \{0, 1\}, \{1, 2\}, \{2, 3\}, \{0, 3\}\}$$

of Figure 1.29: a square outline on four vertices, with no filled interior.

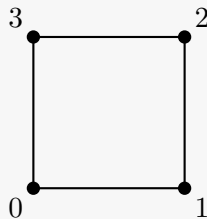


Figure 1.29: The simplicial complex K of Example 1.25: a square outline on four vertices, with no 2-simplex.

Computing a Betti number always follows the same three steps, at whatever dimension q we are interested in: find the cycles $Z_q(K; F)$, find the boundaries $B_q(K; F)$, and take the difference of their dimensions. Carrying this out at $q = 0, 1, 2$ — worked in full in Appendix A.16 — gives:

- $\beta_2(K) = 0$: K has no 2-simplex at all, so there is nothing to be a 2-cycle $Z_2(K; F) = \{0\}$ or a 2-boundary $B_2(K; F) = \{0\}$ in the first place, so^a

$$\beta_2(K) = \dim H_2(K; F) = \dim Z_2(K; F) - \dim B_2(K; F) = 0 - 0 = 0.$$

- $\beta_1(K) = 1$: the loop running all the way around the square traces the chain $\langle 0, 1 \rangle + \langle 1, 2 \rangle + \langle 2, 3 \rangle - \langle 0, 3 \rangle$, and this chain is a cycle, namely it has null boundary:

$$\partial_1(\langle 0, 1 \rangle + \langle 1, 2 \rangle + \langle 2, 3 \rangle - \langle 0, 3 \rangle) = (\cancel{\langle 1 \rangle} - \cancel{\langle 0 \rangle}) + (\cancel{\langle 2 \rangle} - \cancel{\langle 1 \rangle}) + (\cancel{\langle 3 \rangle} - \cancel{\langle 2 \rangle}) - (\cancel{\langle 3 \rangle} - \cancel{\langle 0 \rangle}) = 0.$$

Having no boundary is exactly what it means to be a cycle, so this loop is a 1-cycle; and since K has no 2-simplex to fill it in, it is not the boundary of anything — it survives as a genuine hole.

- $\beta_0(K) = 1$: K is a single connected piece, and indeed β_0 always counts the number of connected components of a complex, as we'll prove later. Actually one can compute this β_0 by hand as we do in A.16.

^a $B_2(K; F)$ and $Z_2(K; F)$ are finite $\mathbb{Z}/2\mathbb{Z}$ -vector space. If a $\mathbb{Z}/2\mathbb{Z}$ -vector space has 2^d elements its dimension is d [30].

Everything above defines homology for a simplicial complex K , not directly for a topological space X . But we already know a space can be triangulated in more than one way (Figure 1.16) — so before we can speak of “the homology of the topological space X ” we need to know that every triangulation gives back the same answer.

Theorem 1.26. *Let X be a topological space and K a triangulation of X (Definition 1.15). The homology of X and the homology of K agree at every dimension: $H_q(X; F) \cong H_q(K; F)$ for all $q \geq 0$.*

Corollary 1.27. *If K_1 and K_2 are two different triangulations of the same space X , their homology groups agree at every dimension too: $H_q(K_1; F) \cong H_q(K_2; F)$ for all $q \geq 0$. Homology does not depend on which triangulation we happened to choose.*

So we allow ourselves to speak of “the homology of X ” for a triangulable space X without naming a particular triangulation — Corollary 1.27 is exactly what let us do that.

1.4.1 The zeroth Betti number

Proposition 1.28 (The zeroth Betti number counts components). *The zeroth Betti number $\beta_0(K)$ equals the number of connected components of K .*

The proof is due to Botnan [33], and is given in Appendix A.17. Here is the intuition behind it. Every vertex is trivially a 0-cycle, since a vertex has no boundary of its own. And two vertices become equal in $H_0(K; F)$ exactly when a path of edges joins them, since the boundary of such a path is just its two endpoints. So each homology class collects exactly the vertices reachable from one another — that is, one class per connected component, and $\beta_0(K)$ counts them.

For instance, in the square complex K of Example 1.25, the vertices 0 and 2 are joined by the path $\langle 0, 1 \rangle + \langle 1, 2 \rangle$, whose boundary is

$$\partial_1(\langle 0, 1 \rangle + \langle 1, 2 \rangle) = (\cancel{\langle 1 \rangle} - \langle 0 \rangle) + (\langle 2 \rangle - \cancel{\langle 1 \rangle}) = \langle 2 \rangle - \langle 0 \rangle,$$

where the shared vertex 1 cancels and only the two endpoints survive: 0 and 2 differ by a boundary, so they land in the same class — as they should, being two corners of one connected square.

Our computations from here on are ultimately aimed at a rank β_q , the q -th Betti number, since that rank is precisely what counts the q -dimensional holes of K :

- β_0 equals the number of components of K , as we just saw — its 0-dimensional holes.
- β_1 is the number of holes in the everyday geometric sense, loops or 1-dimensional holes.⁸
- β_2 counts 3 dimensional cavities, or 3d voids — 2-dimensional holes.

These readings of the Betti numbers will be explored, and partly proved, throughout the rest of this chapter.

Example 1.26 (All three Betti numbers at once). Consider the complex K of Figure 1.30: a hollow tetrahedron shell on vertices a, b, c, d (all four triangular faces filled, but not the solid tetrahedron itself), together with an unfilled triangle b, c, e hanging off the edge $\{b, c\}$, and an isolated vertex f .

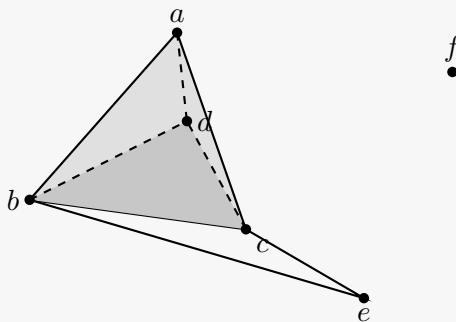


Figure 1.30: The complex K of Example 1.26, seen from the side: a hollow tetrahedron shell a, b, c, d with all four faces filled but no solid interior (vertex d tucked behind the front face a, b, c , its edges drawn dashed), an unfilled triangle b, c, e hanging off edge $\{b, c\}$, and an isolated vertex f .

In symbols,

$$K = \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}, \\ \{a, b\}, \{a, c\}, \{a, d\}, \{b, c\}, \{b, d\}, \{c, d\}, \{b, e\}, \{c, e\}, \\ \{a, b, c\}, \{a, b, d\}, \{a, c, d\}, \{b, c, d\}\}.$$

Reading off the Betti numbers:

- $\beta_2(K) = 1$: β_2 counts 2-holes, namely the 3-dimensional cavities enclosed by filled surfaces, and K has exactly one, sealed inside the four faces $\{a, b, c\}, \{a, b, d\}, \{a, c, d\}, \{b, c, d\}$ of the tetrahedron shell — so $\beta_2(K) = 1$.
- $\beta_1(K) = 1$: β_1 counts the ordinary loops, 1-holes with nothing filling them in. The four faces of the tetrahedron shell are all filled, so no loop supported on a, b, c, d alone survives as a hole. The triangle b, c, e , however, is not filled: the loop running around it is a genuine hole, and it is the only one — so $\beta_1(K) = 1$.
- $\beta_0(K) = 2$: β_0 counts the connected components. The vertices a, b, c, d, e are all joined together by edges, but the vertex f is isolated, giving two pieces in total — so $\beta_0(K) = 2$.

1.4.2 The Euler characteristic

The Betti numbers count holes dimension by dimension, but computing them means working out kernels and images of the boundary maps. There is a much cruder invariant of a simplicial complex, obtained by merely counting simplices, which turns out to be tied to the Betti numbers by a clean formula.

Definition 1.29 (Euler characteristic). Let K be a simplicial complex of dimension n . Its **Euler**

⁸Unlike β_0 , β_1 can in principle depend on the coefficient field F . For planar graphs, however, β_1 always agrees with the everyday count of holes.

characteristic $\chi(K)$ adds up the number of simplices of each dimension, with alternating sign: vertices count positively, edges negatively, triangles positively, and so on up to the top dimension n . Precisely, it is the integer

$$\chi(K) = \sum_{0 \leq i \leq n} (-1)^i \cdot (\text{number of simplices of dimension } i). \quad (1.2)$$

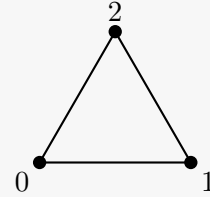
Example 1.27. The simplicial complex of Example 1.11,

$$K = \{\{0\}, \{1\}, \{2\}, \{0, 1\}, \{1, 2\}, \{0, 2\}\},$$

has Euler characteristic

$$\chi(K) = 3 - 3 = 0,$$

its three vertices contributing with a plus sign and its three edges with a minus.



Let $(X, \mathcal{T})$ be a topological space, like a circle, and K a triangulation of it, like the one of Example 1.27. It is tempting to define the Euler characteristic of this generic topological space X as the Euler characteristic of its triangulation K , in formulas $\chi(X) = \chi(K)$. But a space has many triangulations, as Figure 1.16 showed, and it is not yet clear if all these triangulations have the same Euler characteristic value.

Proposition 1.30. *The Euler characteristic of a topological space is the Euler characteristic of any triangulation of it.*

The proof is beyond what we shall develop here.

This makes the Euler characteristic χ well defined on topological spaces, not just on complexes. The next result says it is moreover an invariant, and a strong one: it survives homotopy equivalence.

Proposition 1.31 ([30]). *If X and Y are two homotopy equivalent topological spaces, then $\chi(X) = \chi(Y)$: they have the same Euler characteristic.*

Our intuition for the Euler characteristic usually comes from ordinary solids in $\mathbb{R}^3$, so it is worth spelling out that picture too. A **polyhedron** is a shape built from faces, which meet along edges, which in turn meet at vertices — in other words, nothing but a simplicial complex sitting in $\mathbb{R}^3$. For a subset $X \subset \mathbb{R}^3$ homeomorphic to a polyhedron P , the Euler characteristic can be computed as

$$\chi(X) \equiv V - E + F,$$

where V , E and F count the vertices, edges and faces of P : this is exactly Definition 1.29 specialised to a polyhedron, since its 0-simplices are the V vertices, its 1-simplices the E edges, and its 2-simplices the F faces, so the alternating sum reads $V - E + F$ term by term.

Example 1.28. The simplest polyhedron homeomorphic to the circle S^1 is a triangle, and a square works just as well:

- the triangle, already seen in Example 1.27, has $\chi(S^1) = \chi(\triangle) = 3 - 3 = 0$, its three vertices minus its three edges;
- the square has $\chi(\square) = 4 - 4 = 0$, its four vertices minus its four edges — the same value.

This is exactly what topological invariance demands, since turning a triangle into a square only splits

one vertex into two, joined by a new edge, a purely local change the Euler characteristic cannot see (Proposition 1.31).

Example 1.29 (The torus, from top to bottom). The 2-torus T^2 — the surface of a doughnut, drawn on the left of Figure 1.31 — is a little less obvious. On the right of the same figure sits a triangulation K of it (Definition 1.15), that is, a simplicial complex whose realization is the torus itself: it is the surface of a box with a square hole punched through, built from $V = 16$ vertices, $E = 32$ edges and $F = 16$ faces.

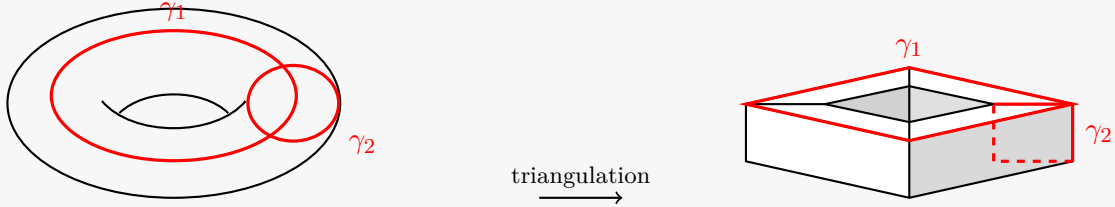


Figure 1.31: *Left*: the 2-torus T^2 , the surface of a doughnut. *Right*: a triangulation K of it — the surface of a box with a square hole punched through, built from 16 vertices, 32 edges and 16 faces. In red, the two independent loops counted by $\beta_1(T^2) = 2$: γ_1 runs the long way around the doughnut, and γ_2 passes through the hole and back around the tube. On the right the hidden half of γ_2 , running along the underside and up the inner wall, is dashed.

We may read the topology of the torus straight off this triangulation: by Theorem 1.26, the homology groups of a topological space and of any triangulation of it agree at every dimension, so $H_q(T^2; F) \cong H_q(K; F)$ for all q , and therefore the Betti numbers of the torus are exactly those of the complex on the right of Figure 1.31. Counting holes in the picture:

- $\beta_2(T^2) = 1$: β_2 counts the 3-dimensional cavities enclosed by filled surfaces, and the torus is a closed surface sealing off exactly one region of space, the inside of the doughnut — so $\beta_2(T^2) = 1$.
- $\beta_1(T^2) = 2$: β_1 counts the ordinary loops with nothing filling them in, and the torus has two independent ones, both drawn in red in Figure 1.31 — the loop γ_1 running the long way around the doughnut, and the loop γ_2 running through the hole and back around the tube — neither of which can be filled in or deformed into the other, so $\beta_1(T^2) = 2$.
- $\beta_0(T^2) = 1$: β_0 counts the connected components, and the torus is a single connected surface — so $\beta_0(T^2) = 1$.
- $\chi(T^2) = 0$: the Euler characteristic of a space is the Euler characteristic of any triangulation of it (Proposition 1.30), so we may count the simplices of K directly,

$$\chi(T^2) = \chi(K) = V - E + F = 16 - 32 + 16 = 0,$$

and the Euler–Poincaré formula, which we state next (Theorem 1.32).

The next result ties the Euler characteristic, defined by counting simplices, to the Betti numbers, that counts as seen before the number of independent holes.

Theorem 1.32 (Euler–Poincaré formula, [33]). *For a simplicial complex K , the Euler characteristic equals the alternating sum of the Betti numbers too:*

$$\chi(K) = \sum_{q=0}^{\infty} (-1)^q \dim C_q(K; F) = \sum_{q=0}^{\infty} (-1)^q \beta_q(K).$$

The argument is a short computation, not conceptually demanding, so we defer it to Appendix A.18.

1.5 Computing homology

Computing a homology group $H_q(K; F)$ straight from its Definition 1.24 gets unwieldy once a complex has any real size, since it means finding cycles Z_q and boundaries B_q by hand. Matrix reduction fixes this: it turns the whole computation into reading off the rank of the boundary maps M_q , and once those ranks are known the Betti numbers fall out immediately.

Throughout, fix an abstract simplicial complex K of dimension n and a coefficient field F , and write n_q for the number of q -dimensional simplices of K , for $0 \leq q \leq n$.

Proposition 1.33. *The dimension of the cycle group Z_q , and the Betti number β_q , can both be read off the rank of the boundary matrices M_q alone:*

1. $\dim \ker \partial_q = n_q - \text{rank } \partial_q$ — the cycles are everything the boundary map does not see, so their count is the total number of q -simplices minus how much of $C_q(K; F)$ the map actually moves;
2. $\beta_q = n_q - \text{rank } \partial_q - \text{rank } \partial_{q+1}$ — the Betti number is that same count of cycles, with the boundaries (measured by $\text{rank } \partial_{q+1}$) subtracted off.

The proof is given in Appendix A.19. What is left, then, is a reliable way to find the rank of a matrix — which row and column reduction give us directly.

Echelon forms. A row echelon form of a matrix is reached using the following operations, carried out in the field F :

- R1: exchange two rows;
- R2: multiply a row by a non-zero element of F ;
- R3: add a multiple of one row to a different row;
- C1: exchange two columns.

The target of the reduction is a matrix with r pivot entries, equal to 1, running down the diagonal, arbitrary entries to their right, and zero rows beneath:

$$\begin{pmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & & \vdots \\ a_{m,1} & \cdots & a_{m,n} \end{pmatrix} \longrightarrow \left(\begin{array}{ccc|ccc} 1 & & & * & \cdots & * \\ & \ddots & & \vdots & & \vdots \\ & & 1 & * & \cdots & * \\ \hline & & & 0 & \cdots & 0 \\ & & & \vdots & & \vdots \\ & & & 0 & \cdots & 0 \end{array} \right).$$

The rank of the matrix is exactly the count r of diagonal entries that survive non-trivial: every operation above is invertible, so it never changes the rank, only the matrix's appearance. We will often not bother with C1 and stop at the ordinary *row* echelon form Gaussian elimination already produces; swapping the roles of rows and columns gives the *column* echelon form the same way.

Example 1.30. Let us compute the homology of the simplicial complex K of Example 1.23 (Figure 1.26), whose boundary matrices $M_2(=\partial_2)$ and $M_1(=\partial_1)$ were already written down there.

Performing only row operations, we obtain

$$M'_2 = \begin{bmatrix} & \langle a, b, c \rangle \\ \langle a, b \rangle & 1 \\ \langle b, c \rangle & \\ \langle a, c \rangle & \\ \langle b, d \rangle & \\ \langle c, d \rangle & \end{bmatrix}, \quad M'_1 = \begin{bmatrix} & \langle a, b \rangle & \langle b, c \rangle & \langle a, c \rangle & \langle b, d \rangle & \langle c, d \rangle \\ \langle a \rangle & 1 & & 1 & & \\ \langle b \rangle & & 1 & 1 & 1 & \\ \langle c \rangle & & & & 1 & 1 \\ \langle d \rangle & & & & & \\ \langle e \rangle & & & & & \end{bmatrix}.$$

These are ordinary row echelon forms, obtained by Gaussian elimination, and the rank of each is just its number of pivots. So M_2 reduces to rank 1 and M_1 to rank 3, i.e. $\text{rank } \partial_2 = 1$ and $\text{rank } \partial_1 = 3$. Together with $n_2 = 1$, $n_1 = 5$, $n_0 = 5$, Proposition 1.33 gives:

- $\beta_2 = n_2 - \text{rank } \partial_2 = 1 - 1 = 0$: the complex encloses no “void”.
- $\beta_1 = n_1 - \text{rank } \partial_1 - \text{rank } \partial_2 = 5 - 3 - 1 = 1$: one loop-hole, the empty triangle on b, c, d .
- $\beta_0 = n_0 - \text{rank } \partial_1 = 5 - 3 = 2$: two components, the triangle-plus-triangle piece and the isolated vertex e .

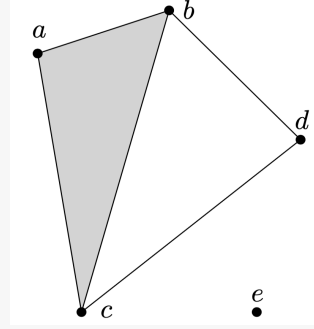


Figure 1.32: The complex K of Example 1.30 once more: the triangle a, b, c is filled, the triangle b, c, d is not, and e is an isolated vertex, matching $\beta_2 = 0$, $\beta_1 = 1$ and $\beta_0 = 2$ just found. Image has been collected from [16].

Smith normal form and representatives. Echelon form is enough for the Betti numbers themselves, but it says nothing about which actual cycles generate a homology group. Pinning those down requires for a stricter normal form, the **Smith normal form**: starting from the row echelon form, we clear away every last $*$ as well, using R1, R2, R3, C1, C2 and C3 (C2 and C3 being the column versions of R2 and R3):

$$\left(\begin{array}{ccc|ccc} 1 & & & * & \cdots & * \\ & \ddots & & \vdots & & \vdots \\ & & 1 & * & \cdots & * \\ \hline & & & 0 & \cdots & 0 \\ & & & \vdots & & \vdots \\ & & & 0 & \cdots & 0 \end{array} \right) \rightarrow \left(\begin{array}{ccc|ccc} 1 & & & 0 & \cdots & 0 \\ & \ddots & & \vdots & & \vdots \\ & & 1 & 0 & \cdots & 0 \\ \hline & & & 0 & \cdots & 0 \\ & & & \vdots & & \vdots \\ & & & 0 & \cdots & 0 \end{array} \right).$$

Every row and column operation mixes together the simplices that index the rows and columns — adding row $\langle b \rangle$ to row $\langle c \rangle$, say, means that row no longer stands for the single vertex $\langle c \rangle$, but for the chain $\langle c \rangle + \langle b \rangle$. So to recover representing cycles — the specific cycles that form a basis for a homology group — we track, alongside the reduction, which chain of simplices each row and column has become after all the operations applied so far; call this its *annotation*. Let us now see what this means in practice, with a concrete example.

Example 1.31. Let us find representing cycles for the homology of the same complex K used in Example 1.30, drawn once more in Figure 1.33:

$$K = \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{a, b\}, \{b, c\}, \{a, c\}, \{b, d\}, \{c, d\}, \{a, b, c\}\}.$$

There we already reduced its two boundary matrices $\partial_2 = M_2$ and $\partial_1 = M_1$ to row echelon form, which was enough to count the holes. Continuing with row *and* column operations carries them all the way to Smith normal form, where the only non-zero entries are a run of 1s down the diagonal. The Smith normal forms M_2'' and M_1'' are:

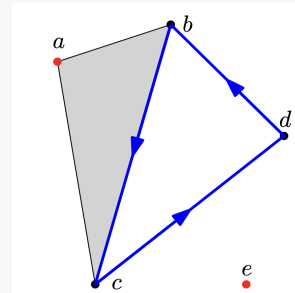


Figure 1.33

$$M_2'' = \begin{bmatrix} \langle a, b, c \rangle \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix},$$

$$M_1'' = \begin{bmatrix} \langle a, b \rangle & \langle b, c \rangle & \langle b, d \rangle - \langle b, c \rangle & \langle a, c \rangle - \langle b, c \rangle - \langle a, b \rangle & \langle c, d \rangle - \langle b, d \rangle + \langle b, c \rangle \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

The column headings are the annotations: the chains of simplices each column has turned into along the way. From these annotated Smith normal forms one can extract the representing cycles — the specific cycles forming a basis of each homology group.^a:

- *Dimension 0*: the two vertices $\langle a \rangle$ and $\langle e \rangle$, drawn in red, span $H_0(K; F)$. This is what we should expect: $\beta_0(K) = 2$ counts the two connected pieces of K , and picking one vertex out of each piece — a from the large component, e from the isolated one — names those two pieces. Any other vertex of the large component would have done just as well, since a path of edges joins it to a .
- *Dimension 1*: the loop $\langle c, d \rangle - \langle b, d \rangle + \langle b, c \rangle$, drawn in blue, spans $H_1(K; F)$. Again this matches the picture: $\beta_1(K) = 1$ counts the single loop-hole, and this cycle is exactly the walk around the empty triangle on b, c, d . The other loop in K , around a, b, c , is not chosen because it is filled by the triangle $\langle a, b, c \rangle$, and so counts as no hole at all.

^aThe precise algorithm, applied step by step to this very complex, is worked out in Appendix A.20.

1.6 Filtrations and persistent homology groups

Homology, as built so far, counts the holes of one fixed complex. Real data, though, rarely comes with a single privileged complex attached to it: more often we have a whole family of complexes, each containing the previous smaller complex. **Persistent homology** applies ordinary homology at every stage of such a growing complex at once, and keeps track of when each hole is born and when it dies. Once combined with a construction that builds a complex out of data in the first place, persistent homology carries geometric information, not just topological information, as we'll see.

1.6.1 Definition

We start from a geometric picture of persistent homology, before pinning down the formal definition behind it.

Example 1.32 (Geometric intuition). The picture to have in mind is a simplicial complex that keeps growing, and persistent homology study how its holes come and go along the way. Take four simplicial complexes $K_1 \leq K_2 \leq K_3 \leq K_4$ (Figure 1.34). We read off the corresponding zero-dimensional **barcode**: intuitively this barcode is a picture of horizontal arrows called **bars**, each recording the lifespan of one connected component, as follows:

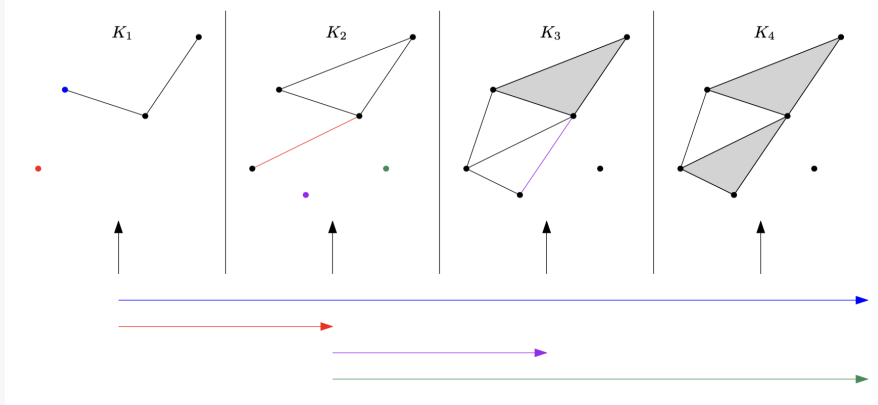


Figure 1.34: Four nested simplicial complexes $K_1 \leq K_2 \leq K_3 \leq K_4$, with the zero-dimensional barcode — one bar per connected component — shown beneath. Image has been collected from [16].

- K_1 : there are two components, visualised by two bars (blue and red) starting at this stage; the corresponding generators (points) are coloured with a matching color (blue and red).
- K_2 : two new isolated points appear, the violet and the green one, each starting its own bar; at the same time the two components of K_1 merge into one. We declare the disappearing component to be the red one, visualised by the red bar terminating just before K_2 .^a
- K_3 : the violet point, born in isolation at K_2 , is joined by two new edges to the surviving blue component. The two components merge into one, and we declare the violet bar to be the one that terminates at this stage. The green point, meanwhile, stays isolated and untouched.
- K_4 : no further change — both surviving blue and green bars run on to infinity — these 2 bars correspond to the 2 connected components of K_4 (the big complex and the isolated point).

The one-dimensional barcode of Figure 1.35 is read the same way, but now each bar records the lifespan of a loop-hole rather than a component:

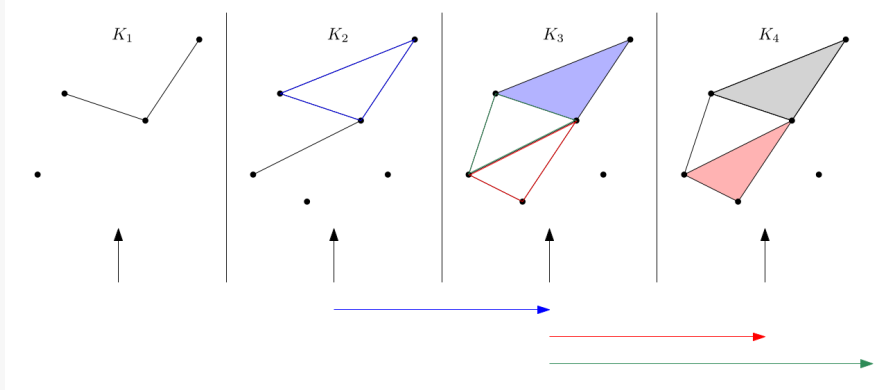


Figure 1.35: The same filtration as Figure 1.34, now with the one-dimensional barcode — one bar per hole — shown beneath. Image has been collected from [16].

- K_1 : no holes, so no bars begin.
- K_2 : a hole appears, giving a blue bar.
- K_3 : that hole becomes trivial (a triangle fills it in), so the blue bar terminates; at the same time two new holes appear, each starting its own bar (the red and the green bars).
- K_4 : the red hole becomes trivial and its bar terminates; the other green hole survives forever, its bar running on to infinity.

^aSince the two components of K_1 appear at the same time, we could equally have kept the red bar and terminated the blue one; the uncoloured barcode would be unchanged. Whenever components of different birth times merge, though, we follow the **elder rule**: the older component survives. This is not a matter of preference but a rule forced on us by the mathematical structure of persistence (in particular the interleaving and stability results developed later).

The rest of this section makes this geometric picture precise, and introduces the computational procedure needed to obtain barcodes like these.

Formal definition. The next step is to pin down, precisely, the object behind the growing-complex picture above.

Definition 1.34 (Filtration). Let K be a simplicial complex. A (discrete) **filtration** of K is a sequence of subcomplexes, each contained in the next, running from nothing up to all of K :

$$K_1 \leq K_2 \leq \dots \leq K_m = K.$$

Figure 1.34 and Figure 1.35 are an example of a filtration. What “persistent” homology tracks, precisely, is which of these homology elements — components, holes, and so on — stay non-trivial as the filtration grows. The inclusion of each K_s into the next K_t gives, for every pair $s \leq t$, a map $i_{s,t}: K_s \hookrightarrow K_t$,⁹ and stringing these together recovers the whole filtration:

$$K_1 \xrightarrow{i_{1,2}} K_2 \xrightarrow{i_{2,3}} \dots \xrightarrow{i_{m-1,m}} K_m = K. \quad (1.3)$$

Now fix a coefficient field F and a dimension q , and take the q -th homology group of every complex in the sequence (1.3). Each inclusion $i_{s,t}$ carries the holes of K_s over to holes of K_t , so it gives a map $(i_{s,t})_*$ between the corresponding homology groups, and we obtain a chain of homology groups mirroring the chain of complexes:¹⁰

$$H_q(K_1; F) \xrightarrow{(i_{1,2})_*} H_q(K_2; F) \xrightarrow{(i_{2,3})_*} \dots \xrightarrow{(i_{m-1,m})_*} H_q(K_m; F) = H_q(K; F). \quad (1.4)$$

Definition 1.35 (Persistent homology groups and persistent Betti numbers). Given a filtration $K_1 \leq K_2 \leq \dots \leq K_m = K$, a field F and $q \in \{0, 1, 2, \dots\}$, the q -dimensional **persistent homology** groups track which homology classes of an early stage K_s are still non-trivial at a later stage K_t : precisely, they are the images of the maps

$$(i_{s,t})_*: H_q(K_s; F) \rightarrow H_q(K_t; F) \quad (1.5)$$

for all $0 \leq s \leq t \leq m$. The dimension of such an image, its rank, counts how many independent classes survive from s to t , and is called a **persistent Betti number**,

$$\beta_{s,t}^q = \text{rank}(i_{s,t})_*. \quad (1.6)$$

Example 1.33 (Persistent Betti numbers). Take the filtration of Figure 1.36; whatever field F we

⁹Each $i_{s,t}$ is a simplicial map, in the sense of Definition A.18.

¹⁰That these maps behave as one expects — composing two inclusions and then passing to homology gives the same map as passing to homology first and then composing — is the *functoriality* of homology, Proposition A.21. It is what allows the maps below to be strung together into a single sequence.

work over, its zero- and one-dimensional persistent Betti numbers are tabulated below:

$s \backslash t$	1	2	3	4
$\beta_{s,t}^0 \rightarrow$	1	2	1	1
	2	/	3	2
	3	/	/	2
	4	/	/	/

$s \backslash t$	1	2	3	4
$\beta_{s,t}^1 \rightarrow$	1	0	0	0
	2	/	1	0
	3	/	/	2
	4	/	/	/

The diagonal entries are just the ordinary Betti numbers of the corresponding stage of the filtration, $\beta_{s,s}^q = \beta_q(K_s)$; the sub-diagonal entries are left blank, since $\beta_{s,t}^q$ is only defined for $s \leq t$.

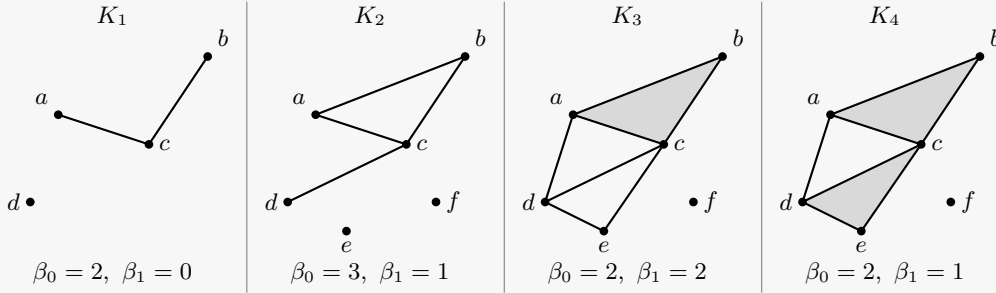


Figure 1.36: The filtration $K_1 \leq K_2 \leq K_3 \leq K_4$ of Example 1.33, with its vertices named and the ordinary Betti numbers of each stage recorded below it. Filled triangles are shaded. Passing from K_2 to K_3 the loop on a, b, c is filled in, while the vertex e is joined to the rest by two new edges; passing from K_3 to K_4 the triangle $\{c, d, e\}$ is filled in.

Concretely, here is what these numbers are telling us geometrically, reading off Figure 1.36:

- $\beta_{2,3}^0 = 2$: the complex K_2 falls into three pieces — the loop on a, b, c together with d , the lone vertex e , and the lone vertex f — but only two of them are still separate in K_3 , since the new edges $\{d, e\}$ and $\{e, c\}$ have attached e to the rest, while f remains on its own.
- $\beta_{3,4}^1 = 1$: the complex K_3 has two loops with nothing inside them, one around a, c, d and one around c, d, e . Only the first is still a hole in K_4 , because K_4 fills the second one in with the triangle $\{c, d, e\}$.
- $\beta_{2,3}^1 = 0$: the only loop of K_2 is the one around a, b, c , and K_3 fills it in with the triangle $\{a, b, c\}$, so none of the loops of K_2 is still a hole by K_3 .

1.6.2 Barcodes and persistence diagrams

We have been drawing barcodes since Figure 1.34, guided only by the geometric picture of components and holes appearing and merging. It remains to say what a barcode actually is. Throughout, fix a field F , a dimension $q \in \{0, 1, \dots\}$, and a filtration $K_1 \leq K_2 \leq \dots \leq K_m = K$.¹¹

Barcodes. The persistent Betti number $\beta_{s,t}$ counts the holes of K_s still alive at K_t , but that is a coarser statement than a barcode makes: among those holes there are some born earlier than K_s , and some that go on living past K_t . A bar $[s, t)$ is meant to be sharper — it stands for one hole born *precisely* at s and dying *precisely* on entering t . So what we need is a way to count, out of all the holes alive across an interval, only the ones whose birth and death fall exactly at its two ends.

Persistent Betti numbers are enough to do this. The number of holes born exactly at K_s and dying

¹¹We drop q from the notation $\beta_{s,t}^q$ and simply write $\beta_{s,t}$, since the dimension is fixed throughout.

exactly on entering K_t is $n_{s,t}$ ¹²:

$$n_{s,t} = (\beta_{s,t-1} - \beta_{s,t}) - (\beta_{s-1,t-1} - \beta_{s-1,t}). \quad (1.7)$$

Holes that never die get their own count: $n_{s,\infty} = \beta_{s,m} - \beta_{s-1,m}$ is the number born at K_s still alive at the very last stage K_m .

The **barcode** of the filtration in dimension q is then simply the collection of all these bars: one interval $[s, t)$ for each of the $n_{s,t}$ holes living from s to t , and one interval $[s, \infty)$ for each of the $n_{s,\infty}$ holes that never die. The same interval may occur several times, or not at all.¹³ The longer a bar, the longer the corresponding hole persists; a longer persistence is usually taken to mean higher importance, though there are settings — especially when the bars are numerous and specifically distributed — where the information sits in the short bars instead.

Example 1.34. Let us apply (1.7) to the filtration of Example 1.33, taking $s = 2$ and $t = 3$. Reading the $\beta_{s,t}^0$ table,

$$n_{2,3} = (\beta_{2,2}^0 - \beta_{2,3}^0) - (\beta_{1,2}^0 - \beta_{1,3}^0) = (3 - 2) - (1 - 1) = 1,$$

so exactly one component is born at K_2 and dies entering K_3 , giving one bar $[2, 3)$, the yellow bar of Figure 1.37. In the same way one finds $n_{1,2} = n_{1,\infty} = n_{2,\infty} = 1$ and $n_{1,3} = n_{1,4} = n_{2,4} = n_{3,4} = n_{3,\infty} = n_{4,\infty} = 0$ and using of them one can write down the entire barcode of Figure 1.37.

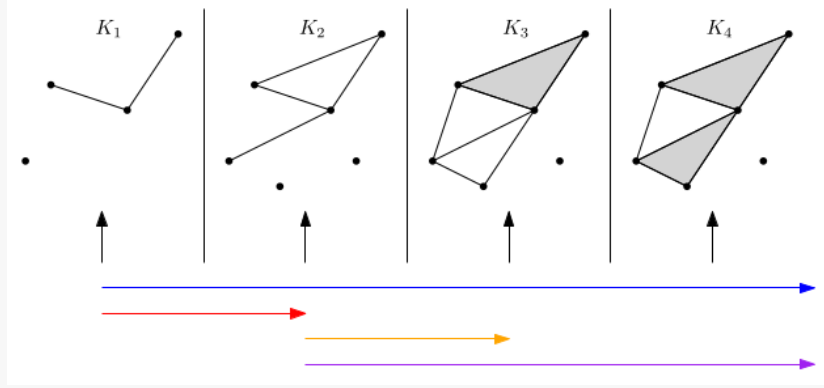


Figure 1.37: The filtration of Figure 1.36, with its zero-dimensional barcode below; the bar/point coloured orange is $[2, 3)$, whose multiplicity $n_{2,3} = 1$ was computed in Example 1.34. Image has been collected from [16].

Persistence diagrams. A barcode can also be drawn as a scatter of points in the plane, one per bar, which is often the more readable picture once there are many bars.

Definition 1.36 (Persistence diagram). Given a barcode, its **persistence diagram** plots each

¹²To motivate this formula, let us read it term by term:

1. **The first bracket counts the holes born at or before K_s that die entering K_t .** Both of its terms count holes that were already there at K_s : $\beta_{s,t-1}$ counts those still alive at the stage K_{t-1} , and $\beta_{s,t}$ those still alive one stage later, at K_t . Since no hole ever comes back, the second group sits inside the first, and subtracting leaves precisely the holes that were still there at K_{t-1} and are gone at K_t — which is exactly what dying on entering K_t means.
2. **The second bracket counts the same thing, but only for the holes born strictly before K_s .** It is the identical computation with s replaced by $s - 1$: $\beta_{s-1,t-1}$ and $\beta_{s-1,t}$ count the holes already there at K_{s-1} that survive to K_{t-1} and to K_t respectively, so their difference again counts those dying on entering K_t . And being already there at K_{s-1} is just another way of saying born strictly before K_s .
3. **Subtracting the second bracket from the first leaves the holes born exactly at K_s .** Every hole counted by the second bracket is also counted by the first: a hole born strictly before K_s was certainly born at or before K_s , and both brackets ask for the same death time. So the difference keeps exactly those holes that the first bracket counts and the second does not — the ones born at or before K_s but *not* before it, which is to say born exactly at K_s , and dying on entering K_t .

¹³The formal construction of the barcode — where $n_{s,t}$ comes from, in terms of the induced maps $(i_{s,t})_*$, and the resulting Definition A.22 — is given in Appendix A.23.

interval $[s, t)$ as the point $(s, t) \in \mathbb{R}^2$, weighted by its multiplicity $n_{s,t}$. An infinite bar $[s, \infty)$ cannot literally be plotted, so we represent it by a finite point $(s, m+1)$ instead, choosing $m+1$ simply because it sits safely above every finite death time $t \leq m$ that actually occurs.^a

^aAny y -coordinate strictly above every finite death time would do; $m+1$ is only a convenient choice, since every s, t already lies in $\{1, \dots, m\}$.

A persistence diagram encodes precisely the same information as its barcode: the persistence of a bar is measured by its length, while the persistence of the corresponding point is measured by its distance from the diagonal $\Delta = \{(x, x) \mid x \in \mathbb{R}\}$. Every point of a persistence diagram lies strictly above Δ , since a bar's birth always comes before its death.¹⁴ Diagrams are usually the preferred visualisation once the number of points grows large enough that a scatter reads more clearly than a stack of bars; with only a few points or bars, a barcode is often the more descriptive picture.

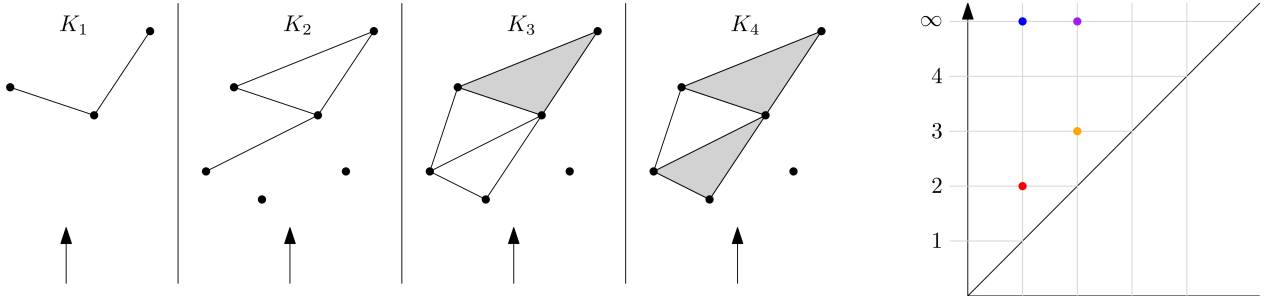


Figure 1.38: The zero-dimensional barcode of Figure 1.37, alongside the corresponding persistence diagram. Image has been collected from [16].

The fundamental lemma of persistent homology. The multiplicities $n_{s,t}$ were built out of the persistent Betti numbers $\beta_{s,t}$. The reverse expression also holds: knowing every $n_{s,t}$ is enough to recover every $\beta_{s,t}$, by simply counting how many bars pass over the interval $[s, t]$.

Lemma 1.37 (Fundamental lemma of persistent homology). *The persistent Betti number $\beta_{s,t}$ equals the total multiplicity of every bar containing the pair (s, t) , i.e. born at or before s and terminating strictly after t :*

$$\beta_{s,t} = \sum_{s' \leq s, t' > t} n_{s',t'}.$$

The proof is short and is given in Appendix A.24. Lemma 1.37 means the information encoded in a barcode, in a persistence diagram, and in the persistent Betti numbers, is exactly the same in all three cases: any one of them can be recovered from either of the others.

1.6.3 Computing persistent homology

Computing every persistent Betti number $\beta_{s,t}$ via (1.7) to get the multiplicities $n_{s,t}$ would be wasteful. There is instead an algorithm that extracts every $n_{s,t}$ directly, in two steps — a matrix reduction, then a direct read-off — without computing a single rank $\beta_{s,t}$ along the way.

Its starting point is that a filtration is nothing more than the list of the simplices of the complex K , each tagged with the stage at which it enters. We put them in that order and write down the boundary matrices of K (Definition 1.23) with rows and columns labelled by simplices in exactly that order, so that the position of an entry already records when the corresponding piece appeared. All the algorithm then does is apply linear row and column operations to those matrices, until each column either vanishes or is left with a single lowest surviving entry of its own.¹⁵

This reduction of the boundary matrices produces the whole barcode of the filtration, and also the persistence diagram, since the two are equivalent from the Lemma 1.37. This is what makes persistent

¹⁴A point (s, t) comes from a bar $[s, t)$, and $s < t$ (or $t = m+1 = \infty$) always, so the point sits strictly above the line $y = x$.

¹⁵The operations, their order, and the reasons they achieve this are given in Appendix A.25, together with a worked example.

homology usable in practice: the barcode of a filtration far too large to inspect by hand costs one reduction of its boundary matrices, and nothing more.

Chapter 2

Topological Data Analysis

The first step in a topological treatment is thus frequently a creation of simplicial complexes. This chapter presents several ways of building complexes out of a point cloud, a finite collection of points in a metric space — like a set of numerical data, or similar. Two important properties will be presented: whether the complex faithfully reflects the shape the points were sampled from, and how stable it is to perturbing the points (for example if the points are perturbed by noise).

2.1 From point clouds to complexes

2.1.1 The homology inference problem

On its own, a finite point cloud S (a finite collection of points, like a set of numerical data) has no interesting topology: it is just finitely many points, so $\beta_0(S) = |S|$, namely the number of connected components equals the number of points, and every other Betti number is zero¹. What makes S worth studying topologically is that we think of the point cloud S as a *sample* of some continuous shape M that we do not otherwise have access to — think of S as measurements taken from an underlying curve or surface M — and we would like to recover the homology of the continuous shape M from the finite point cloud S alone. This is the **homology inference problem**.

Definition 2.1 (Thickening). For a scale $r \geq 0$, take every point that lies within reach r of some point of the point cloud S : call this the **r -thickening** of S , written S^r . In symbols,

$$S^r = \{y \in \mathbb{R}^n \mid \exists x \in S, \|x - y\| \leq r\}.$$

Equivalently, and more usefully for pictures, S^r is nothing but the union of the closed balls of radius r centred at every point of S :

$$S^r = \bigcup_{x \in S} B(x, r).$$

¹Since we have zero loops $\beta_1 = 0$, zero cavities $\beta_2 = 0$, and so on

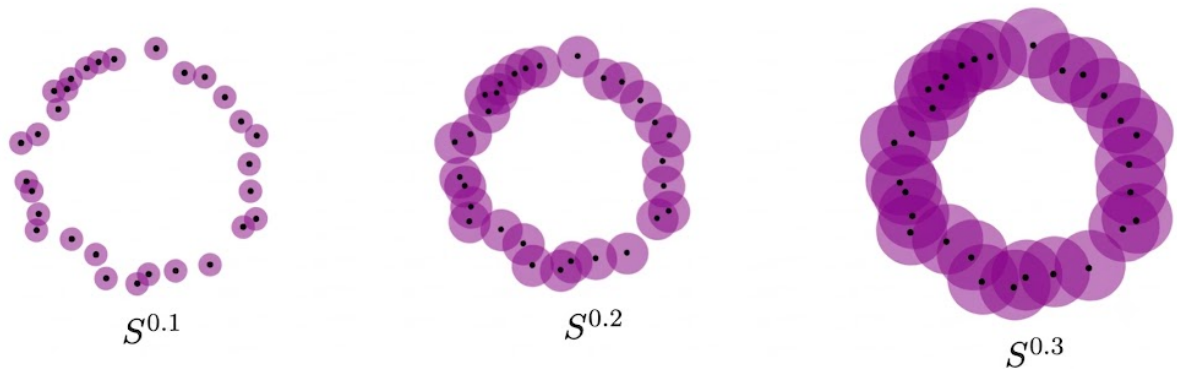


Figure 2.1: Three thickenings $S^{0.1} \subset S^{0.2} \subset S^{0.3}$ of a finite sample S of a circle (Tinarrage, 2021). The largest one shown already has the homotopy type of the circle itself. Image has been collected from [30].

Figure 2.1 shows why thickening helps: for a well-chosen radius r , the thickening S^r is homotopy equivalent to the continuous shape M , and homotopy equivalent spaces always have the same homology (Appendix A.28). Because of this recover the homology of the surface M then reduces to computing the homology of point cloud S^r , and we have this point cloud of data. One can indeed show that such an r exists, and even give explicit bounds for it.²

This leaves two questions:

- how to choose such a radius r ;
- how to compute the homology of the thickening S^r once it is chosen.

The first, unfortunately, has no practical answer, as [30] explains.³ Any single choice of the radius r is therefore a gamble, as Figure 2.1 already suggests:

- too small, and the thickening S^r is still too disconnected to see the shape of M ;
- too large, and separate features of M get glued together and disappear.

This is exactly the difficulty that persistent homology is built to sidestep: rather than committing to one radius r , it tracks the topology of the thickening S^r across *every* radius r at once (Section 1.6.1), and lets the features that persist across a wide range of scales distinguish themselves from those that do not.

2.1.2 Rips complexes

The most direct way to turn a scale r into a complex is to connect two points whenever they are within reach of one another.

Definition 2.2 (Rips complex). Let X be a metric space and let a sample $S \subset X$ be a finite subset. Choose a scale $r \geq 0$. The Rips complex^a $\text{Rips}(S, r)$ is an abstract simplicial complex defined by the following rules:

1. The vertex set is S .
2. A subset $\sigma \subseteq S$ is a simplex iff its points are all within distance r of one another, i.e. iff its diameter^b satisfies $\text{Diam}(\sigma) \leq r$.

^aRips complexes are sometimes also called Vietoris-Rips complexes.

²The precise statement needs two more quantities: the Hausdorff distance $d_H(S, M)$, measuring how closely the point cloud S samples the continuous shape M , and the reach of M , a measure of how “thin” or sharply curved M is allowed to be. Given these, one can name an explicit range of radii r for which S^r is guaranteed homotopy equivalent to M [30].

³The reason is that the bounds on r are stated in terms of two quantities describing the continuous shape M itself — the Hausdorff distance $d_H(S, M)$ and the reach of M , in the notation of the footnote above — and M is exactly the object we do not have direct access to; see [30] for a precise discussion.

^bDiameter of a finite subset $A \subset X$ of a metric space X is defined as $\text{Diam}(A) = \max_{x,y \in A} d(x,y)$. Diameter of X is defined as $\text{Diam}(X) = \sup_{x,y \in X} d(x,y)$.

Example 2.1. Figure 2.2 follows the same five points through three snapshots as we turn up the scale r , with a ball of radius $r/2$ drawn around each point to help read off which pairs are close enough:

- We start with r small: none of the balls touch yet, so nothing is close enough to connect, and $\text{Rips}(S, r)$ is just the five bare points.
- We turn r up: balls begin to overlap, and every time two balls touch we draw an edge between their centres — an edge appears exactly when two points are within distance r of each other.
- We turn r up further still: now three of the points have all grown close enough to be pairwise within distance r , so all three edges between them are already there. But Definition 2.2 does not stop at pairs — it asks whether in the whole *group* all the points are within distance r , and here it does, so the filled triangle on those three points also becomes a simplex of $\text{Rips}(S, r)$.

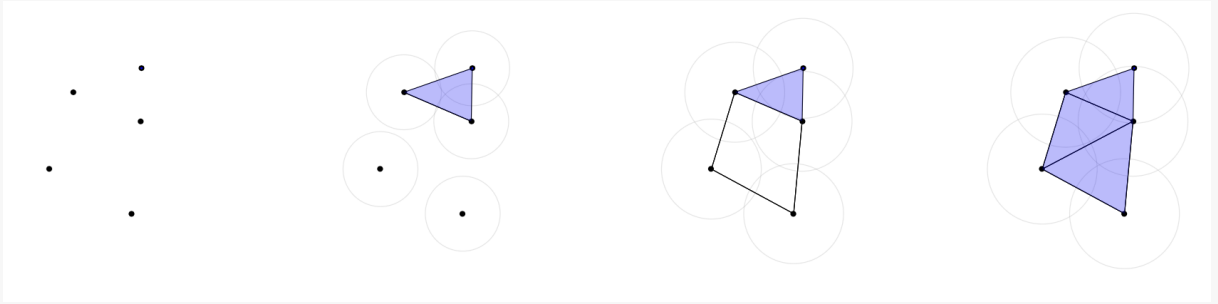


Figure 2.2: Five points in the plane and three corresponding Rips complexes $\text{Rips}(S, r)$. Visualisation is assisted by circles of radius $r/2$ around each point. For much larger scales the Rips complex is not planar and eventually becomes 4-dimensional (see the footnote below for details on this). Image has been collected from [16].

Once r is large enough that all five points are mutually within reach, every possible triangle, tetrahedron, and so on among them becomes a simplex of $\text{Rips}(S, r)$ at once — not just the one triangle pictured above, but a solid 4-dimensional block on all five points. Already just connecting every pair of the five points by a straight edge cannot be drawn in the plane without two edges crossing, which is why the complex is no longer planar; and fitting the full solid block in without any of its five corners collapsing together needs four dimensions, not two.

Remark 2.3. A few comments:

- $\text{Rips}(S, r)$ represents a combinatorial snapshot of S at scale r .
- It is easy to verify that Rips complexes are indeed abstract simplicial complexes in the sense of Definition 1.12: if σ is a simplex then so is each of its subsets, since shrinking a group of mutually close points to a smaller group leaves them mutually close too.

Remark 2.4. Some properties of the Rips complexes:

1. Rips complexes are often the preferred construction in TDA due to their simplicity in terms of the definition and computation.
2. If $r_1 \leq r_2$, then $\text{Rips}(S, r_1) \subseteq \text{Rips}(S, r_2)$: growing the scale from r_1 to a larger r_2 only ever admits new simplices, since any group of points that was already mutually within distance r_1 is still mutually within the larger distance r_2 , so nothing already present is ever lost.

Recall from Definition 1.34 that a filtration of a complex K is just a nested sequence $K_1 \leq K_2 \leq \dots \leq K_m = K$. The Rips complexes give one directly: by item 2 of Remark 2.4, growing r only ever adds simplices to $\text{Rips}(S, r)$, never removes them. Listing the scales $r_1 < r_2 < \dots < r_m$ at which $\text{Rips}(S, r)$ actually changes therefore gives a filtration

$$\text{Rips}(S, r_1) \leq \text{Rips}(S, r_2) \leq \dots \leq \text{Rips}(S, r_m)$$

called the **Rips filtration** of S — and the persistent homology machinery of Section 1.6.1 applies to it without any further work⁴

Example 2.2. The three complexes of Example 2.1 and Figure 2.2 are exactly the first three stages of a Rips filtration on those five points.

2.1.3 Čech complexes

A Rips complex connects points once they are pairwise close. Čech complexes actually do something more: a group of points spans a simplex only when their balls all genuinely share a point.

Definition 2.5 (Čech complex). Let X be a metric space and let a sample $S \subset X$ be a finite subset. Choose a scale $r \geq 0$. The Čech complex^a $\check{\text{Cech}}(S, r)$ is an abstract simplicial complex defined by the following rules:

1. The vertex set is S .
2. A subset $\sigma \subseteq S$ is a simplex iff $\bigcap_{x \in \sigma} B(x, r) \neq \emptyset$, namely iff their balls all genuinely share a point.

^aČech complexes are a special case of nerve complexes, a connection explained in Section 2.1.4.

Remark 2.6. $\check{\text{Cech}}(S, r)$ represents a combinatorial snapshot of S at scale r , exactly as $\text{Rips}(S, r)$ does, and it is easy to verify that Čech complexes are indeed abstract simplicial complexes in the sense of Definition 1.12⁵.

Rips and Čech complexes are close relatives, but not identical:

- If $r_1 \leq r_2$, then $\check{\text{Cech}}(S, r_1) \subseteq \check{\text{Cech}}(S, r_2)$, exactly as for Rips complexes.

Example 2.3. Figure 2.3 shows the smallest example telling the two constructions apart: three points, every pair of them less than a distance r apart. We build both the Rips complex and the Čech complex on these same three points.

- On the left, $\check{\text{Cech}}(S, r)$: the three balls of radius r around the three points do not all share a common point, so by Definition 2.5 the triangle is *not* included in $\check{\text{Cech}}(S, r)$ — only the three edges are.
- On the right, $\text{Rips}(S, r)$: since all three points are within distance r of one another, their diameter is at most r , so by Definition 2.2 the group of all three also spans a simplex — the triangle, or 2-simplex, is included in $\text{Rips}(S, r)$.

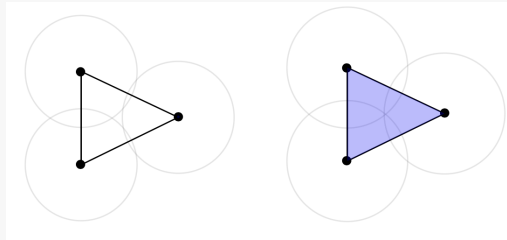


Figure 2.3: The same three points spanning a triangle in $\text{Rips}(S, r)$ (right) but not in $\check{\text{Cech}}(S, r)$ (left), since the three balls of radius r fail to share a common point. Image has been collected from [16].

⁴A fully formal definition of the Rips filtration is given in Appendix A.26.

⁵If σ is a simplex then so is each of its subsets.

2.1.4 Nerve complexes

Čech complexes are themselves a special case of more general construction, the **nerve**, which turns any collection of sets into a simplicial complex simply if all the sets overlap.

Definition 2.7 (Nerve). For $k \in \mathbb{N}$ let $\mathcal{U} = \{U_1, U_2, \dots, U_k\}$ be any collection of subsets of X . The nerve of $\mathcal{U}$ is the abstract simplicial complex $\mathcal{N}(\mathcal{U})$ defined by the following rules:

1. The vertex set is $\mathcal{U} = \{U_1, U_2, \dots, U_k\}$, consisting of k elements.
2. A subset $\sigma \subseteq \mathcal{U}$ is a simplex iff $\bigcap_{U \in \sigma} U \neq \emptyset$, namely iff all the subsets $\{U_1, U_2, \dots, U_k\}$ overlap.

Example 2.4 (Nerve complex construction). On the left we have five overlapping subsets, one per colour: red, blue, green, orange, and black. To build the nerve $\mathcal{N}(\mathcal{U})$ shown on the right, we go through the collection $\mathcal{U}$ and check, one group of sets at a time, whether that group shares a common point.

- We start with pairs. Red and blue subsets overlap. By Definition 2.7 that overlap alone is enough to put an edge (a 1-simplex) between the red and the blue vertex on the right.
- We then move to triples. Blue, orange, and black overlap all at once: there is a point lying in all three regions simultaneously (not just in each pair of them). That triple overlap is exactly the condition Definition 2.7 asks for a 2-simplex, so the complex on the right gains a filled triangle with the blue, orange, and black points as vertices.

The same check, repeated for every remaining group of sets, produces the full nerve on the right: an edge for each pair that overlaps, a filled triangle for each triple that overlaps, and so on for larger groups – which is why the nerve of $\mathcal{U}$ ends up looking like a combinatorial “shadow” of how the sets are arranged.

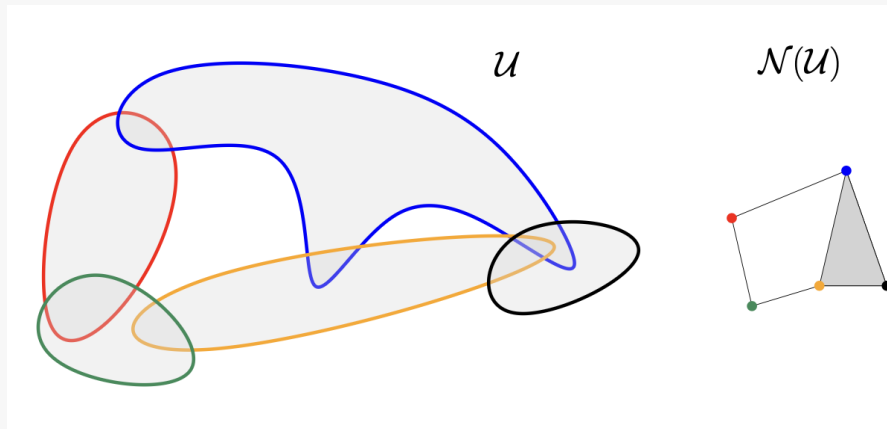


Figure 2.4: An example of a nerve. Image has been collected from [16].

Comparing the two definitions, the definition of the nerve and the definition of the Čech complex look strikingly similar — and indeed the Čech complex is nothing but the nerve $\mathcal{N}$ of a particular collection of sets, namely

$$\check{\text{Cech}}(S, r) = \mathcal{N}(\{B(s, r)\}_{s \in S}).$$

The reason is simple: take $\mathcal{U}$ to be the collection of r -balls, one centred at each point of S .

- Definition 2.5 puts a simplex on a group of points $\sigma \subseteq S$ exactly when their balls $\{B(x, r)\}_{x \in \sigma}$ all share a common point.
- Definition 2.7 puts a simplex of the nerve on a group of sets exactly when they all overlap.

For the collection of balls, these are the very same condition. So building $\check{\text{Cech}}(S, r)$ and building the nerve of the r -balls are just two names for the same construction; in symbols:

$$\check{\text{Cech}}(S, r) = \mathcal{N}(\{B(s, r)\}_{s \in S}). \quad (2.1)$$

Figure 2.4 shows a nerve of a generic collection of sets.

One of the main advantages of nerve complexes is that their homotopy type in some cases represents the union of the elements of $\mathcal{U}$. This is formalized by the nerve theorem, of which we state a special case.

Theorem 2.8 (Nerve theorem, [16]). *Let $n \in \mathbb{N}$ and assume a collection $\mathcal{U} = \{U_1, U_2, \dots, U_k\}$ consists of closed convex subsets of $\mathbb{R}^n$. Then $U_1 \cup U_2 \cup \dots \cup U_k \simeq \mathcal{N}(\mathcal{U})$.*

We omit the proof; a sketch is given in [16].

$$\check{\text{Cech}}(S, r) \stackrel{(2.1)}{=} \mathcal{N}(\{B(s, r)\}_{s \in S}) \simeq \bigcup_{s \in S} B(s, r).$$

In words: the Čech complex has exactly the same homotopy type as the union of the r -balls themselves, namely the Čech complex has the same homotopy type as the shape you get by literally drawing all those balls and looking at the blob they cover. We are allowed to write this thanks to the nerve theorem (Theorem 2.8)⁶.

Notice that the union $\bigcup_{s \in S} B(s, r)$ on the right is exactly the r -thickening S^r of Definition 2.1. So the Čech complex has the same homotopy type as the thickening — this is the whole reason for bothering with Čech complexes: although they are harder to compute than Rips complexes (Section 2.1.3), what they produce is precisely the shape of the thickening we wanted in the first place, as illustrated in Figure 2.5.

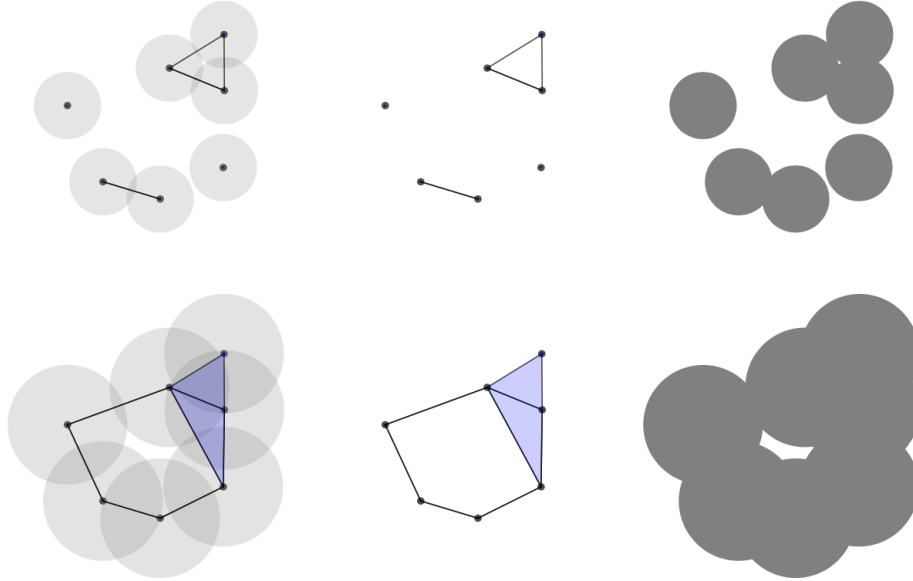


Figure 2.5: Two examples of Čech complexes: balls and the corresponding complex superimposed (left), complex only (centre), and the union of balls, homotopy equivalent to the complex by the nerve theorem (right). Image has been collected from [16].

This closes the loop opened in Section 2.1.1, where recovering the homology of an unknown shape M from a finite sample (point-cloud) S was reduced to two questions:

- *How to compute the homology of the thickening S^r , once r is chosen?* The nerve theorem answers this directly: $\check{\text{Cech}}(S, r)$ is homotopy equivalent to S^r , so computing the homology of the Čech complex *is* computing the homology of the thickening.

⁶For more details, see Appendix A.29.

- *How to choose r in the first place?* We still do not have to answer this: instead of picking one radius r , we let r grow and watch the whole Čech filtration $\{\check{\text{Cech}}(S, r)\}_{r \geq 0}$ (Appendix A.26) go by, exactly as persistent homology is built to do.

2.2 Stability

We now introduce continuous filtrations. These structures will be crucial in the formulation of the stability theorem, which states that persistent homology behaves continuously with respect to the underlying filtration. In practice this stability theorem is very important, as it guarantees that small perturbations of the input data lead to small changes to the persistence diagrams 1.36.

2.2.1 Continuous filtrations

Recall from Definition 1.34 that a discrete filtration of a simplicial complex K is a sequence of subcomplexes $K_1 \leq K_2 \leq \dots \leq K_m = K$, shown once more in Figure 2.6.

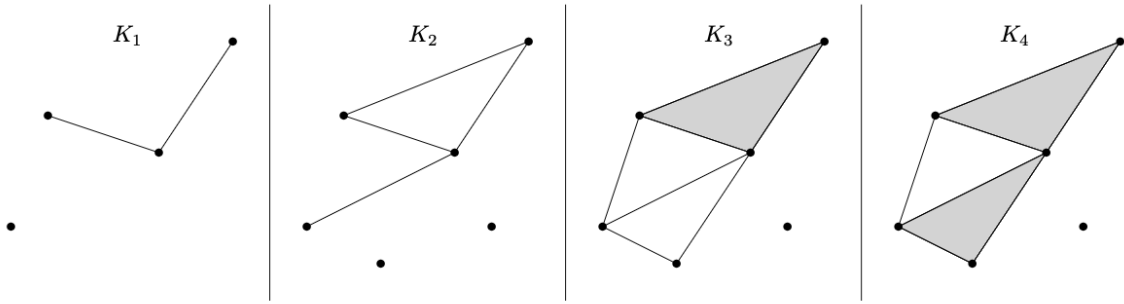


Figure 2.6: A discrete filtration, recalled from Figure 1.34. Image has been collected from [16].

There is an equivalent, more compact way of recording exactly the same information: instead of storing each subcomplex separately, one can annotate every simplex $\sigma \in K$ by the index $f(\sigma)$ at which σ first appears. Given such an annotation, each K_i is recovered as

$$K_i = \{\sigma \in K \mid f(\sigma) \leq i\}. \quad (2.2)$$

Example 2.5 (Annotation of a filtration). Figure 2.7 shows this annotation for the same filtration as Figure 2.6:

- At the first stage K_1 there are four vertices and two edges: all six of these simplices already belong to K_1 , so they all get annotation 1.
- The simplices that only newly appear at the second stage K_2 — present in K_2 but not yet in K_1 — get annotation 2 instead.
- Likewise, every later simplex is labelled by the first stage at which it shows up.

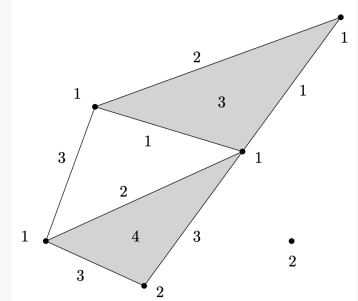


Figure 2.7: The annotation of simplices encoding the filtration of Figure 2.6. Image has been collected from [16].

This reformulation suggests two ways of expanding the scope of filtrations: allowing infinitely many subcomplexes, indexed continuously rather than by $1, \dots, m$; and defining a filtration directly from such an annotation function:

Just as a discrete filtration is read off from an annotation function, a continuous filtration can be built directly from annotation functions.

Definition 2.9 (Filtration function, sublevel filtration). Let K be a simplicial complex.

- A **filtration function** is an annotation f that labels every simplex $\sigma \in K$ by a non-negative real number $f(\sigma)$, the scale at which σ enters the filtration, subject to $\sigma \leq \tau \implies f(\sigma) \leq f(\tau)$: a simplex never appears before any of the pieces it is built from.
- The **sublevel filtration** associated to f is the continuous filtration consisting, at every scale $r \geq 0$, of the **sublevel complex** $K_r = \{\sigma \in K \mid f(\sigma) \leq r\} \leq K$: the simplices whose label does not exceed r .

There are two motivating reasons for introducing continuous filtrations:

- Most of the standard constructions of filtrations actually yield continuous filtrations: the Čech filtration, the Rips filtration, filtration by alpha complexes, sublevel filtrations, etc.
- The interleaving structure and the resulting stability theorem depend on the continuous choice of parameter.

The definition of persistent homology groups for continuous filtrations is the same as Definition 1.35, with the only difference being the continuous range of the indices $0 \leq s \leq t$. The matrix-reduction based computation of Section 1.6.3 is unhindered by the expansion to continuous filtrations.⁷ In a nutshell, we can compute the same barcodes using the same procedure for continuous or discrete filtrations.

It is moreover always possible to pass back and forth between a discrete and a continuous filtration without losing information; we spell out exactly how this works in Appendix A.30.⁸ Continuous filtrations conveniently model the geometric setup of the standard filtrations. On the other hand, discrete filtrations are a convenient finite description on which we may develop algorithmic approaches.

2.2.2 Bottleneck distance and stability theorem

One of the most useful facts about persistent homology is that it is *stable*: if we perturb the input — the point cloud, or the filtration function — only slightly, the resulting persistence diagram (Definition 1.36) also only changes slightly, rather than jumping around unpredictably. To make this precise we first need a way of measuring the distance between two persistence diagrams themselves, so that "how much the diagram changed" has an actual numerical meaning. This distance is called the **bottleneck distance**.

Look at the two persistence diagrams in Figure 2.8: the left one has points colored in blue, the right has points colored in red. We want to measure the distance between these two diagrams.



Figure 2.8: Two persistence diagrams we want to compare: blue points on the left, red points on the right. Images from [16].

⁷The computation depends only on the filtration function defined on the (finitely many) simplices, and does not require all (infinitely many) complexes of the filtration separately.

⁸From this point on, whenever we mention an unspecified filtration, or consider a transition from a finite to continuous filtration or vice versa, the underlying interplays we have in mind are the ones detailed in Appendix A.30.

Here is how to do it, step by step:

- **Match points across the two diagrams.** Pick a way of pairing up points of the blue diagram with points of the red diagram: this is called a **partial matching**, since not every point is required to find a partner in the other diagram. Figure 2.9 shows an example of a partial matching.

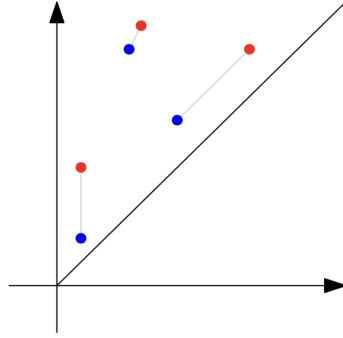


Figure 2.9: An example of a partial matching between the blue and red persistence diagrams. Image has been collected from [16].

- **Send leftover points to the diagonal.** Any point left unmatched is instead compared to its own projection onto the diagonal $\Delta = \{(z, z) \mid z \in \mathbb{R}\}$.⁹ This way every point, matched or not, ends up compared to something.
- **Measure how far apart each pair is.** Then we measure the distance between two points $v = (x_1, y_1)$ and $w = (x_2, y_2)$ in the plane using a distance function, though it is not very important to know the details of exactly which distance is used.¹⁰
- **Score the matching by its worst pair.** The cost of a given matching is the largest of all these pairwise distances: so a single badly-matched pair sets the cost of the whole matching, no matter how well every other pair is matched. Figure 2.10 shows several different matchings between the same two diagrams.
- **Pick the cheapest matching.** Trying every possible matching and keeping the one with the smallest cost gives the bottleneck distance between the two diagrams.

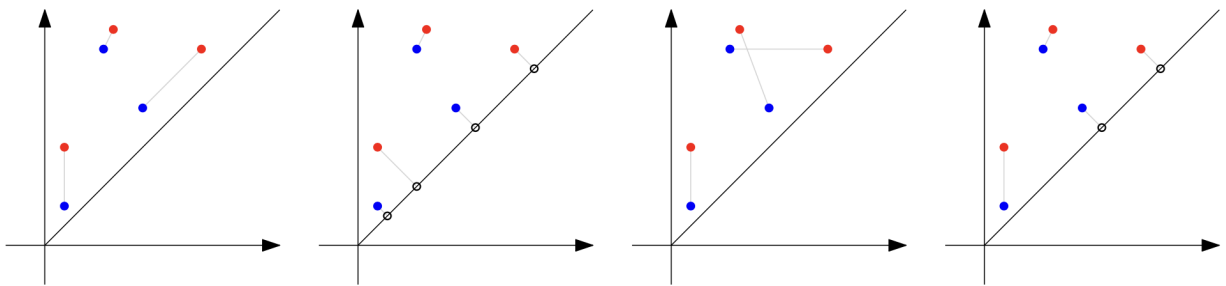


Figure 2.10: Four different partial matchings between the same blue and red persistence diagrams. Image has been collected from [16].

Definition 2.10 (Bottleneck distance). The bottleneck distance $d_B(\mathcal{A}, \mathcal{B})$ between two persistence diagrams $\mathcal{A}$ and $\mathcal{B}$ is the smallest cost among all partial matchings between them.

⁹The diagonal is the nearest point representing a feature of essentially zero persistence (since axis of the persistence diagram are birth and death coordinates, so a point on the diagonal is a point that dies immediately).

¹⁰The distance used is

$$d_\infty(v, w) = \max\{|x_1 - x_2|, |y_1 - y_2|\}, \quad (2.3)$$

so we want the larger of the horizontal and vertical gap between the two points.

Figure 2.11 makes the same idea visual: draw, around every matched point, the square witnessing its d_∞ distance¹¹ to its partner; the cost of a matching is then the size of its largest square, and the bottleneck distance is realized by whichever matching keeps that largest square as small as possible.

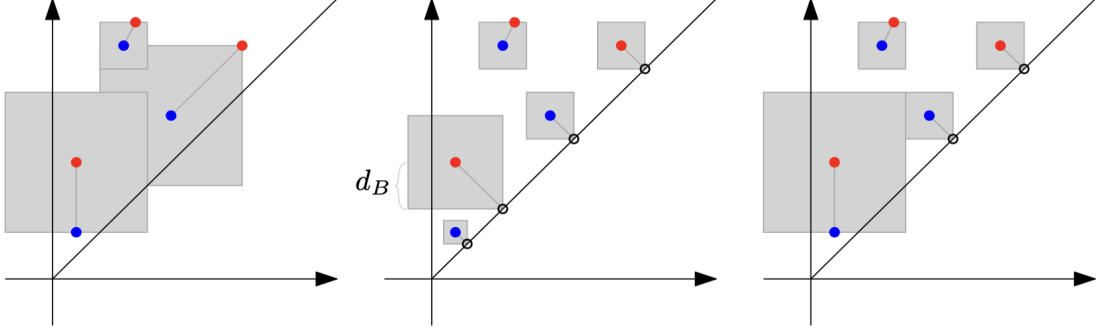


Figure 2.11: The matching distance visualised as the largest square among the matched pairs; the smallest such largest square, achieved by the middle matching, is the bottleneck distance d_B . Image has been collected from [16].

With a distance between persistence diagrams in hand, we can finally state the stability theorem: no matter which of our constructions produced the diagrams, if the two inputs are close to one another then the resulting diagrams are close too, within the bottleneck distance.

Theorem 2.11 (Stability theorem). *Assume persistence diagrams $\mathcal{A}$ and $\mathcal{B}$ represent the persistent homology of filtrations $\mathcal{V}$ and $\mathcal{W}$ built from two inputs that are ε -close to one another, in a sense depending on the construction used.^a Then the bottleneck distance between the two diagrams is at most ε : $d_B(\mathcal{A}, \mathcal{B}) \leq \varepsilon$.*

^aThere are in fact several such senses of closeness, one for each construction (sublevel filtrations, Rips filtrations, Čech filtrations); the complete statement, with all of them spelled out, is given in Appendix A.31.

The moral of the theorem is that small perturbations of the input lead to small changes in persistence diagrams.¹²

We have just seen the stability of persistence diagrams under noise, namely under small perturbations of the points themselves (Section 2.2.2). Persistent homology is stable also in another sense, it is stable with respect to the spaces: if we have two point clouds where the second is a small perturbation of the first, then the filtrations we build on them¹³ are “close”¹⁴. We’ll not discuss this form of stability in this thesis, in [16] one can find a precise discussion of this property of persistence homology.

2.3 Takens embedding: from timeseries to point clouds

In the previous sections we saw how to construct simplicial complexes out of a point cloud S , a finite collection of points in a metric space, using the Rips complex (Section 2.1.2) or the Čech complex (Section 2.1.3). We now want to explain the step that comes before this one: how can one construct this point cloud S in the first place? More precisely, how can a point cloud be built out of a **time series**, namely a sequence of measurements recorded one after another as time goes by — think of the strain read out by a gravitational-wave detector, or a temperature logged once an hour. This discussion is based on documentation of a famous python library for dealing with Topological Data Analysis: [34].

¹¹The distance used is

$$d_\infty(v, w) = \max\{|x_1 - x_2|, |y_1 - y_2|\}, \quad (2.4)$$

so we want the larger of the horizontal and vertical gap between the two points.

¹²On the other hand, homology representatives may be unstable, homology representative are defined in (Appendix A.25) and more precisely in [16].

¹³Rips and Čech.

¹⁴We still have to say precisely what “does not change much” means for a filtration; that requires a notion of distance between filtrations, which we do not develop here. A complete discussion is given in [16].

2.3.1 From time series to time delay embeddings

Given such a time series — a function $f(t)$ recording one measurement at every time t — the idea is to slide a window of fixed size along the signal, and turn every position of the window into a single point of a (possibly higher-dimensional) space. More formally this construction is called a **time delay embedding**, or **Takens embedding**, named after Floris Takens, who pioneered its use in the study of dynamical systems.

Example 2.6 (Building an embedding by hand). • Figure 2.12 shows a cosine signal on the left, sampled at a sequence of instants; take the first two consecutive points and read off their two values y_1 and y_2 . Plot the pair (y_1, y_2) as a single point in the plane $\mathbb{R}^2$: that point is shown on the right of the same figure.

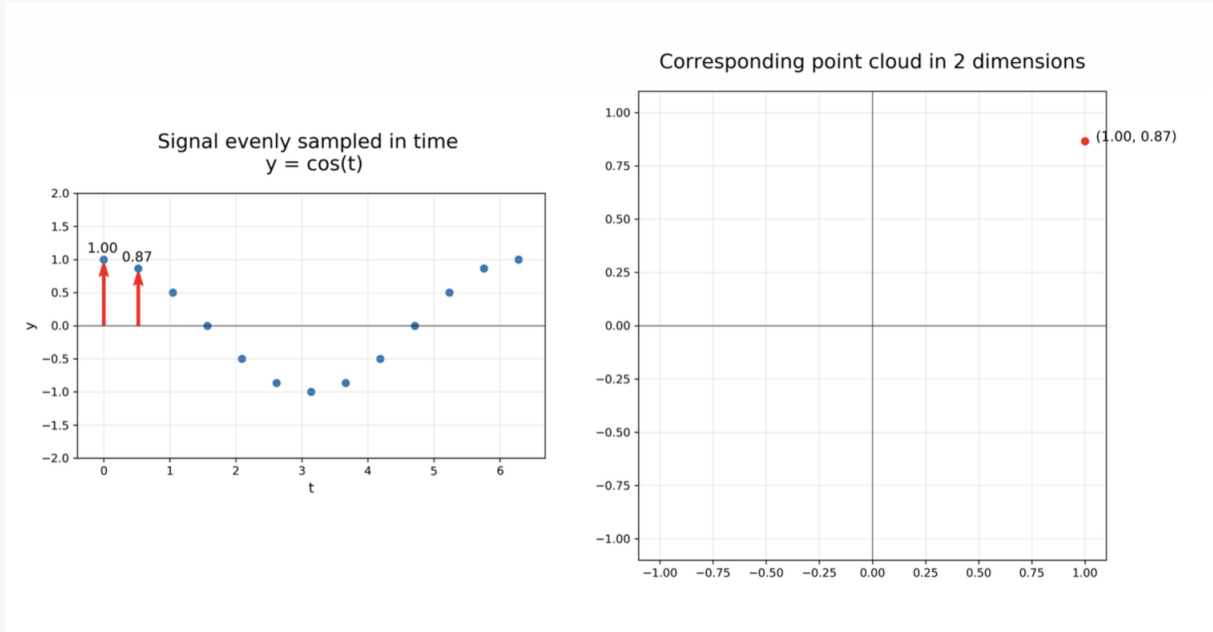


Figure 2.12: The first step of building a 2-dimensional embedding by hand (Example 2.6). Image has been collected from [29].

- Now move one step along the signal: Figure 2.13 takes the next pair of consecutive points, reads off their values y_2 and y_3 , and plots (y_2, y_3) as a second point in the plane, alongside the first.

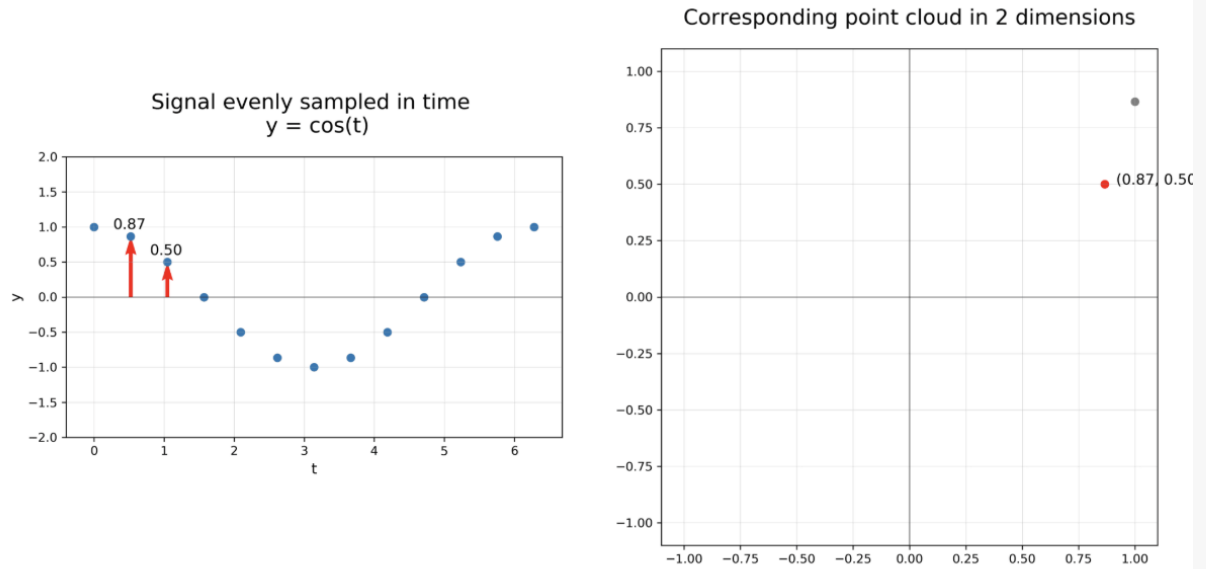


Figure 2.13: The second step: one point further along the signal (Example 2.6). Image has been collected from [29].

- Repeating this all the way along the signal — always pairing each point with the one right after it — produces one point in the plane for every position of the sliding window. Figure 2.14 shows the result once the whole signal has been swept through this way: the points trace out a complete ellipse.

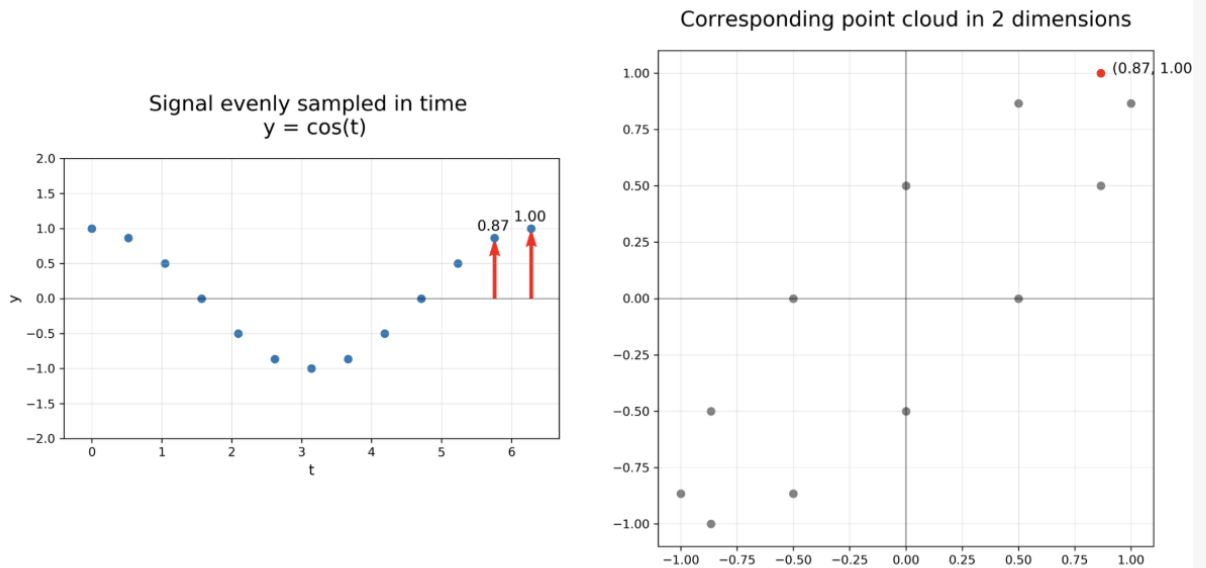


Figure 2.14: Repeating the process to the end of the signal: the cosine signal becomes a complete ellipse in the plane (Example 2.6). Image has been collected from [29].

Definition 2.12 (Time delay embedding). Let $f(t)$ be a time series.

- Choose an **embedding dimension** $d \in \mathbb{N}$
- Choose a **time delay** $\tau \in \mathbb{N}$: how far apart, in time, those measurements are spaced (in Example 2.6, τ was the gap between one sample and the next)^a.

- The **time delay embedding** of f with parameters (d, τ) is then the map

$$TD_{d,\tau}f: \mathbb{R} \rightarrow \mathbb{R}^d,$$

$$t \mapsto (f(t), f(t + \tau), f(t + 2\tau), \dots, f(t + (d - 1)\tau)).$$

Evaluated at a particular instant t_i , this vector $(f(t_i), f(t_i + \tau), f(t_i + 2\tau), \dots, f(t_i + (d - 1)\tau))$ is the i -th point of the point cloud we are constructing.

^aNotice that this time delay is a $\mathbb{N}$ number, for example a $\tau = 1$ means that we are collecting consecutive points

In words: at every instant t we record the d values that f takes at t and at $d - 1$ further instants spaced τ apart, and bundle them into one point of $\mathbb{R}^d$ — exactly the pairing-up procedure of Example 2.6, but now for a general embedding dimension d rather than just $d = 2$.

- **Time delay τ .** A big time delay τ makes us select points of the timeseries $f(t_i)$ that are far apart in time, as one can see directly from the formula of Definition 2.12: the entries $f(t_i), f(t_i + \tau), \dots$ are spaced τ apart. On the contrary, a small time delay τ makes us select points of the timeseries $f(t_i)$ that are close to each other; taking the smallest time delay possible makes us select points of the timeseries $f(t_i)$ that are simply consecutive.
- **Embedding dimension d .** This value decides the dimension d of the space where the point cloud is going to live, namely decides the $\mathbb{R}^d$. In Example 2.6 above we decided to plot the pairs (y_1, y_2) in a plane, so we decided to have the point cloud (the ellipse) live in 2-dimensional space, i.e. $\mathbb{R}^2$.¹⁵

The main idea we will explore in this section is that if a timeseries f has a non-trivial recurrent structure, then its point cloud will have non-trivial topology, for appropriate choices of the embedding dimension d and the time delay τ .

A periodic example.

Example 2.7 (A periodic signal). We have just seen how we passed from a cosine signal to an ellipse in a 2-dimensional plane (Example 2.6). Take now again a cosine function $f(t) = \cos(5t)$, which is a periodic signal, shown in Figure 2.15: it repeats itself exactly every $T = 2\pi/5$ in time.^a

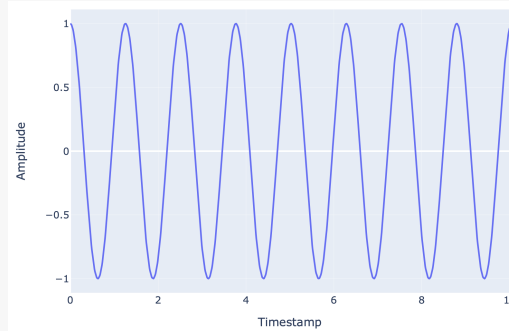


Figure 2.15: The periodic signal of Example 2.7.

We can decide to have as embedding dimension $d = 3$ instead of $d = 2$, and in this case we find that the ellipse — the embedded point cloud — lives in 3-dimensional space, as shown in Figure 2.16. So we can see that the periodicity of f is reflected in the ellipticity of the time delay embedding. It turns out that in general, periodic functions trace out ellipses in $\mathbb{R}^d$, whatever the embedding dimension d .

¹⁵There is also another parameter, the stride, that we define later; for the moment it is a value that downsamples the timeseries: with a high value of stride the timeseries appears less populated.

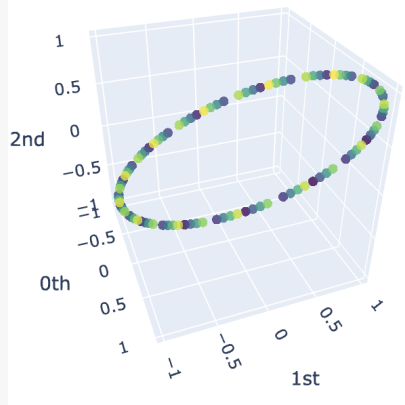


Figure 2.16: The point cloud obtained by embedding the periodic signal of Example 2.7: a single closed loop, since the signal is periodic. Points are coloured by their position along the signal, from earliest to latest, purely to make the direction of travel around the loop visible.

^aA function f is periodic with period $T > 0$ if $f(t + T) = f(t)$ for every time t .

A non-periodic example.

Example 2.8 (A non-periodic, recurrent signal). Not every signal worth embedding is periodic. Take $f(t) = \cos(t) + \cos(\pi t)$: since the ratio of the two frequencies is irrational, f is not a periodic function, yet the sum of the two oscillations still produces a recurrent, quasi-periodic pattern, shown in Figure 2.17.

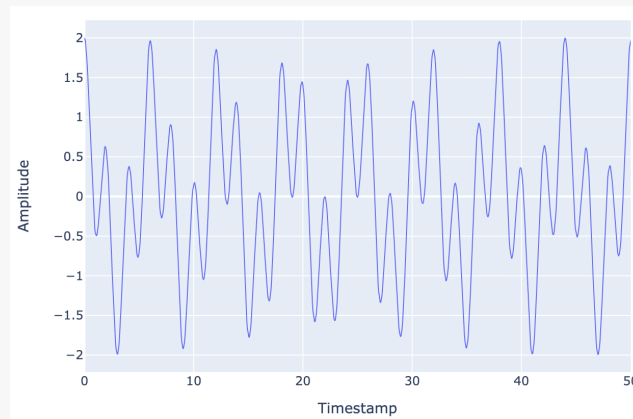


Figure 2.17: The non-periodic, recurrent signal of Example 2.8.

If we apply the same machinery as before, namely the time delay embedding, to this non-periodic time series, we get the point cloud of Figure 2.18. Notice that again we have chosen embedding dimension $d = 3$.^a Notice that we get a point cloud^b which looks like a ball with a spiral inside it. This may not be easy to see from the picture alone, but on the giotto-tda documentation site [34] one can rotate the point cloud interactively to look at it from every angle.

^aWe have also chosen a time delay $\tau = 16$, but for the moment this parameter is not really important.

^bWe should call this point cloud the Takens embedding of the non-periodic signal.

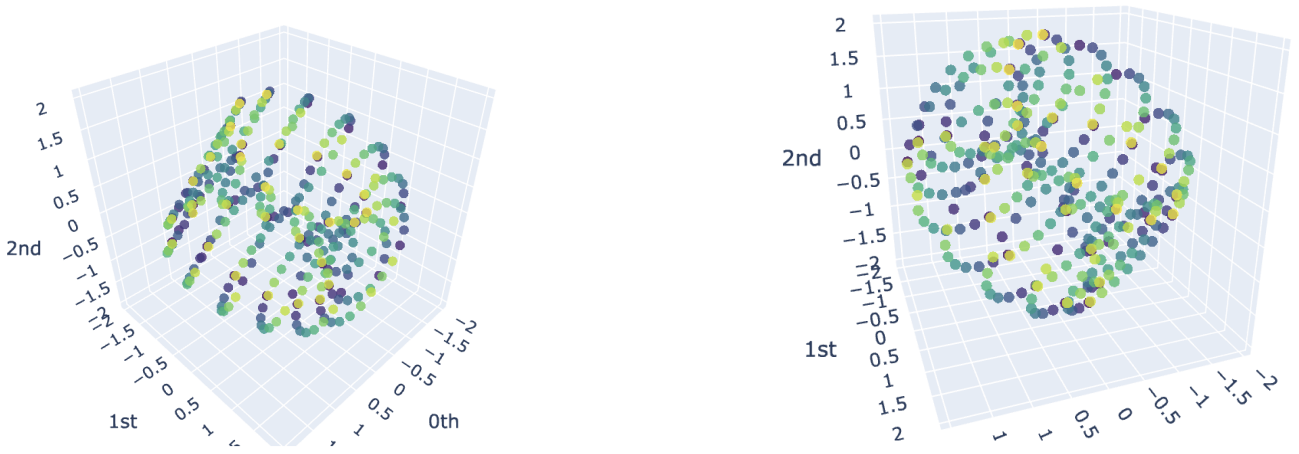


Figure 2.18: The point cloud obtained by embedding the non-periodic signal of Example 2.8, shown from two different angles.

2.3.2 From time delay embeddings to persistence diagrams

The two point clouds of the examples above look visibly different in shape; persistent homology lets us confirm this precisely, rather than just by eye. We compute the persistent homology of the Rips filtration (Section 2.1.2) built on each point cloud, in dimensions 0, 1, and 2 — namely connected components, loops, and voids — and obtain the two persistence diagrams of Figure 2.19: one for the periodic signal, one for the non-periodic signal.

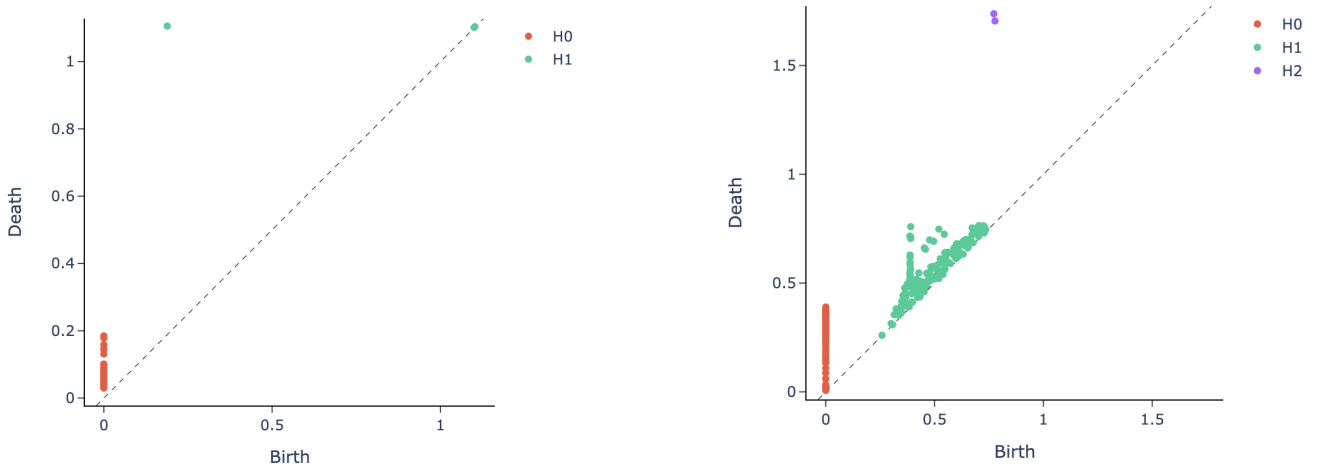


Figure 2.19: The persistence diagrams of the two point clouds of cosine example 2.7 (left) and non-periodic example 2.8 (right).

The two diagrams tell a clear story. In the periodic case (2.19 left), a single point (in green) stands out at high persistence, in dimension H_1 : a loop, matching the ellipse we already saw in Figure 2.16. In the non-periodic case (2.19 right), instead, it is dimension H_2 that stands out, with two violet points at high persistence: two voids, consistent with the more intricate, ball-with-a-spiral shape of Figure 2.18. These clear differences in topology are exactly what make the time delay embedding technique powerful for telling different time series apart.

2.3.3 Picking the embedding dimension and time delay

In the examples above, we manually chose values for the embedding dimension d and the time delay τ . It turns out, however, that there are two techniques that can determine these two parameters automatically instead: **mutual information** to determine the time delay τ , and **false nearest neighbours** to determine the embedding dimension d [34].

Mutual information. The idea behind mutual information is to ask, for a candidate delay τ : if I already know the value of the signal f at some instant, how much *new* information do I gain by also looking at its value τ later?

- To make this a number, we sort the values the signal f takes into a handful of bins, from smallest to largest.
- If the value at $t_i + \tau$ is highly predictable from the value at t_i — for instance because τ is tiny, so the signal has barely had time to change — then looking τ ahead teaches us almost nothing new, and the mutual information is high but uninformative.
- As τ grows, the value τ ahead becomes progressively less predictable, and the mutual information drops.
- The recipe is then to pick the *first* value of τ at which this mutual information stops dropping and reaches a first minimum: past that point, a bigger delay τ is not buying us any additional useful information, so this first minimum is taken as the optimal time delay.¹⁶

False nearest neighbours. The idea behind false nearest neighbours is different: it looks at whether the chosen embedding dimension d is already large enough to faithfully unfold the signal’s underlying structure.

- Take two points of the point cloud that end up close together at dimension d .
- If, once we add one more coordinate to reach dimension $d + 1$, those two points suddenly turn out to be far apart, then their original closeness at dimension d was an illusion, an artefact of not having enough room to tell them apart — a so-called *false* neighbour.
- The optimal embedding dimension is then the smallest d for which such false neighbours have essentially disappeared, meaning points that look close together genuinely stay close together as further coordinates are added.

We do not develop either method in full mathematical detail here; see [34] for both algorithms spelled out precisely.

2.3.4 Running the search algorithm

Applying these two automatic criteria to our two example signals means letting the search scan over a wide range of candidate values of embedding dimension d and time delay τ — up to some upper bound chosen generously enough that the true optimum is not missed:

- for the periodic signal of Example 2.7, this search settles on an embedding dimension $d = 6$ and a time delay $\tau = 29$;
- for the non-periodic signal of Example 2.8, it settles instead on $d = 6$ and $\tau = 14$.

Both results are noticeably different from the illustrative choices $d = 3$ used earlier in this section — a reminder that the values we picked by hand were chosen for ease of drawing, not because they were already optimal.

Since the resulting point cloud now lives in a 6-dimensional space, too many dimensions to plot directly, one convenient way to still visualise it is to first project it down to 3 dimensions — for instance via principal component analysis, that we’ll define in the next subsection technique — and only then visualise the point cloud as we did before.

¹⁶A formal discussion about the time delay is given in [35].

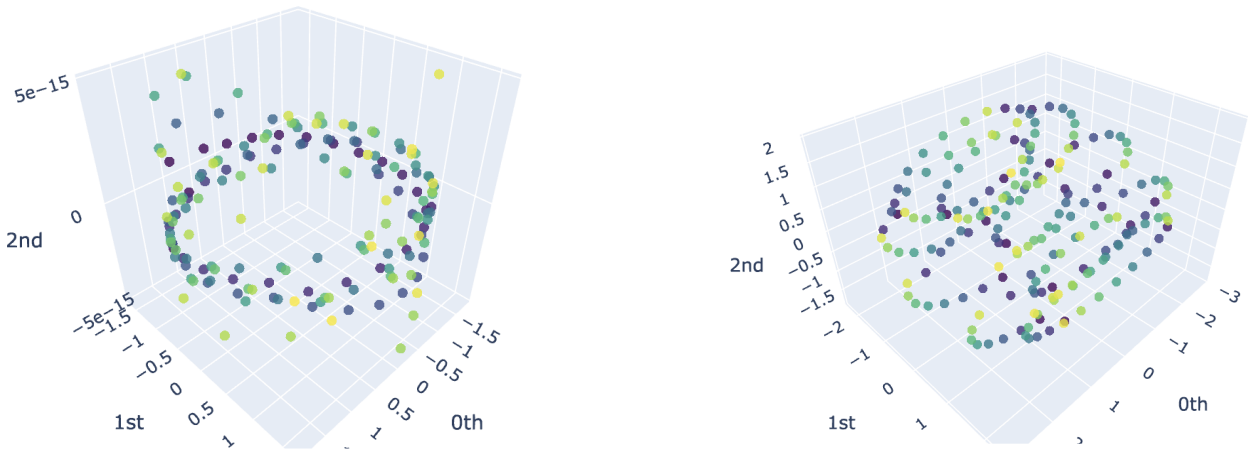


Figure 2.20: The point clouds obtained from the search-optimized embeddings of the periodic and non-periodic signals [34].

Once a point cloud has been built this way out of a time series, everything from Section 2.1 onward — like Rips complexes — applies to it exactly as it would to any other point cloud, with no further modification. So if we construct the Rips filtration (section 2.1.2) on each of these two new point clouds, we get the corresponding persistence diagram for each of them.

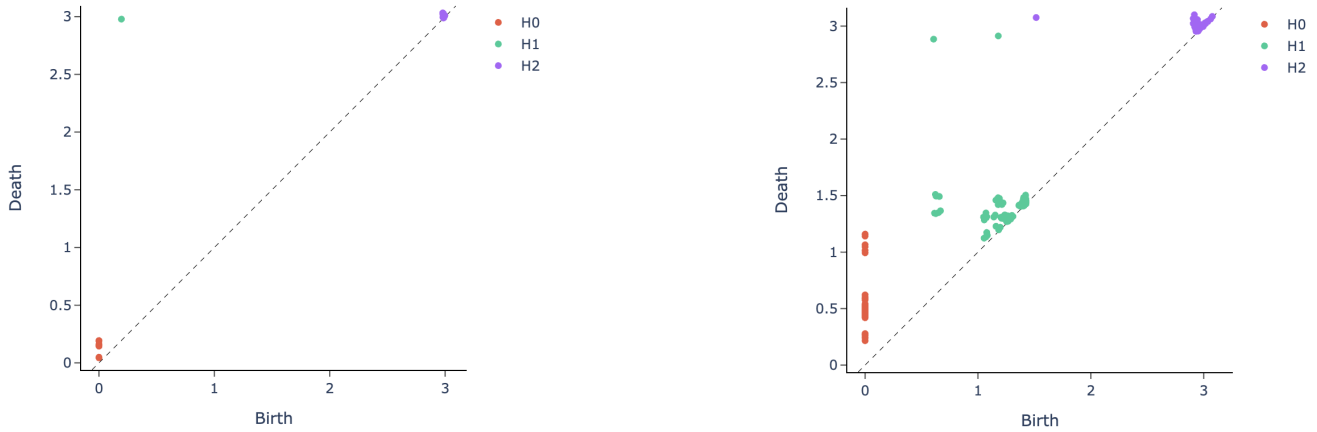


Figure 2.21: The persistence diagrams of the two point clouds of Figure 2.20 [34].

- In this case the persistence diagram for the periodic signal is essentially unchanged: as in Section 2.3.2, a single (green) point stands out at high persistence, this point represent a H_1 element: a loop.¹⁷
- The non-periodic signal, however, now reveals two H_1 (green) points (they correspond to two distinct loops) and one H_2 point (violet), which correspond to a 3-D cavity.¹⁸

These clear differences in topology are exactly what make the time delay embedding technique powerful for telling different time series apart.¹⁹

2.3.5 Principal component analysis

We used principal component analysis, or PCA, above to draw a 6-dimensional point cloud on paper; we now say briefly what it actually does.

¹⁷Looking more closely at this persistence diagram for the pure cosine signal, we also see the emergence of some violet H_2 points, three-dimensional cavities that were not present in the earlier diagram of Figure 2.19. These are a spurious creation of the principal component analysis projection step used to visualise the 6-dimensional embedding, not a genuine feature of the periodic signal's topology: as the diagram shows, they are extremely short-lived, since they lie very close to the diagonal.

¹⁸This is the homological signature of a hypertorus [29].

¹⁹The code for the worked examples of this section, including the choice of parameters, is available on GitHub [29].

Picture a cloud of points scattered in space — say, a blob stretched out in one direction more than in the others, like a squashed rugby ball.

- PCA looks for the direction along which the cloud is most spread out, and calls that its **first principal component**.
- It then looks for the next most spread-out direction, among those orthogonal to the first direction, and calls that the **second principal component**.
- It keeps going this way, each new direction chosen to be at right angles to all the previous ones and as spread-out as possible among what is left.

The result is a new set of directions, or axes, tailored to this particular cloud of points: the first few already capture most of how the cloud is spread out, while the last few capture very little.

Example 2.9 (PCA in action). Figure 2.22 shows PCA applied to a point cloud living in 3 dimensions.

1. On the left we see the point cloud in its original 3-dimensional space: notice that the points are distributed in an almost “flat ellipse” shape, spread out much more in two directions than in the third.
2. Thanks to the mathematics briefly described later, we can compute the 3 principal components of this point cloud, shown in the figure as 3 orthogonal arrows: the direction of the first principal component $PC1$ accounts for 65% of the variance of the dataset, the direction of the second principal component $PC2$ accounts for 30%, and the direction of the third principal component $PC3$ accounts for the remaining 5%.
3. We then decide to project this point cloud down to a 2-dimensional space, using PCA: namely, we project the point cloud onto the plane spanned by the two principal components $PC1$ and $PC2$. This projection is shown in the right panel of Figure 2.22, whose two axes are exactly $PC1$ and $PC2$. Since $PC1$ and $PC2$ together already account for $65\% + 30\% = 95\%$ of the variance, this projection keeps essentially all of the shape of the original point cloud, while dropping one whole dimension.
4. PCA is not always the optimal choice, though — something we will encounter again later in the gravitational-wave application of this thesis. PCA selects and projects onto the directions of *maximum* variance; but sometimes it is precisely the directions of *least* variance that carry the physically interesting information, and those are exactly the directions PCA discards first.

PRINCIPAL COMPONENT ANALYSIS (PCA): VISUALIZATION AND PROJECTION

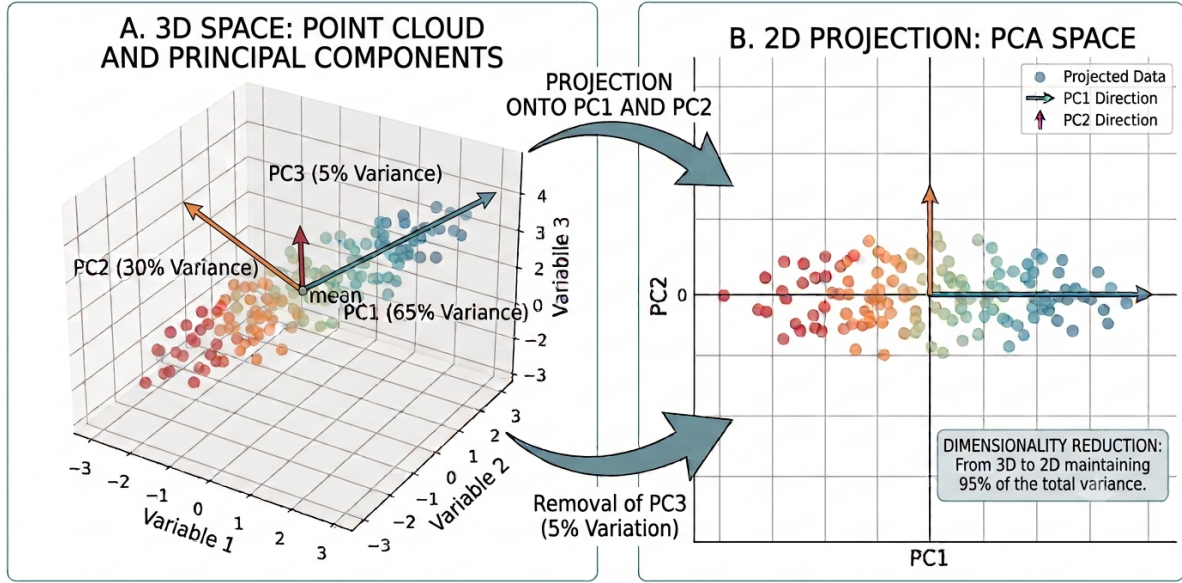


Figure 2.22: PCA applied to a point cloud in 3 dimensions (Example 2.9). Left: the original point cloud and its three principal components $PC1$, $PC2$, $PC3$, with the percentage of variance each one accounts for. Right: the point cloud projected onto the plane spanned by $PC1$ and $PC2$, retaining 95% of the variance.

Mathematically, these directions are not found by trial and error: they come out of the **covariance matrix** of the point cloud, a table that records how much each coordinate of the data varies, and how much pairs of coordinates vary together. The principal components turn out to be exactly the **eigenvectors** of this matrix, and the corresponding **eigenvalues** tell us how much of the cloud's total spread each direction accounts for: the eigenvector with the largest eigenvalue is the first principal component, the one with the second-largest eigenvalue is the second principal component, and so on.

This is what makes PCA useful for visualizing a high-dimensional point cloud such as ours: if we keep only the first two or three principal components, namely the two or three directions along which the cloud is most spread out, and ignore the rest, we lose comparatively little of the cloud's shape — and what remains can finally be drawn on a page or a screen. This is exactly what Section 2.3.4 did: reduce a 6-dimensional point cloud down to the 3 directions along which it varies the most, in order to plot it.

Naturally, keeping only the top few directions and discarding the rest does throw some information away.

This discussion of PCA has been deliberately qualitative, so as not to make the thesis too math-heavy; for the underlying mathematics, see [36].

2.4 Vectorization

The representations of persistent homology we have used so far — barcodes and persistence diagrams — are not good candidates for feeding into statistical analysis or machine learning: their structure (a variable-length collection of intervals or points) does not even admit a natural addition of two persistence diagrams [37]. The fix is to **vectorize** a persistence diagram, namely to summarize it by one or several numbers, or more generally by an element of a vector space, that can be fed directly into standard statistical or machine-learning pipelines. These vectorizations of persistence diagram are in general called *topological summaries*. We review a few such summaries below.

2.4.1 Real-valued summaries

Recall from Definition 1.36 that a persistence diagram D is a finite collection of points; in this section we write such a point as (b_i, d_i) , where:

- b_i is the **birth time** of the corresponding topological feature (the instant a d -dimensional hole appears in the filtration);

- d_i is its **death time** (the instant that same hole disappears);
- the **lifetime** of the hole is its birth time subtracted from its death time, $l_i = d_i - b_i$.

A simple summary of D just combines these lifetimes into a single number.

Definition 2.13 (Total persistence). For a persistence diagram $D = \{(b_i, d_i)\}_{i=1}^m$ with point lifetimes $\{l_i\}_{i=1}^m$, the **total persistence** $TotPers_k$ is $\sum_{i=1}^m l_i$, the sum of the lifetimes of all its points.

Notice that the total persistence is itself computed once per dimension: a total persistence in dimension $k = 0$, another in dimension $k = 1$, and so forth.

Example 2.10 (Total persistence). As a concrete instance of a persistence diagram, take D shown in Figure 2.23. Adding up the lifetimes of all the points in a given dimension gives the total persistence of D in that dimension. Looking at Figure 2.23:

- Total persistence in dimension $k = 0$: the red H_0 points are all squeezed into a dense little cluster right against the origin, so each individual point has a tiny lifetime. Adding up all of these tiny lifetimes, one per point of the cloud, still gives a moderate total persistence of 7.45, simply because there are so many of them (The number of points in dimension $k = 0$ is $N_0 = 991$ from Example 2.12).
- Total persistence in dimension $k = 1$: there is only a single H_1 point, sitting far above the diagonal, at birth ≈ 0.03 and death ≈ 2.03 . Since it is the only point, the total persistence in this dimension is simply its own lifetime, 2.00.

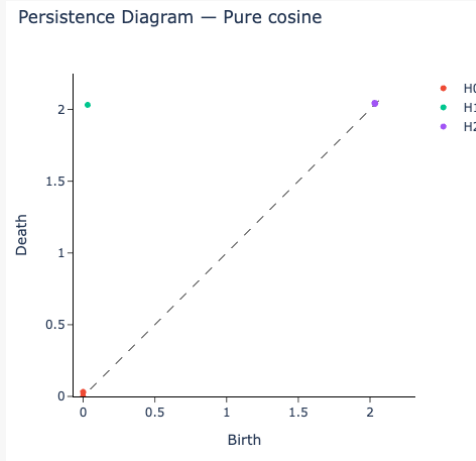


Figure 2.23: A persistence diagram D (Example 2.10).

Definition 2.14 (Maximum persistence). For a persistence diagram $D = \{(b_i, d_i)\}_{i=1}^m$ with point lifetimes $\{l_i\}_{i=1}^m$, the **maximum persistence** $MaxPers_k$ is $\max_i \{l_i\}$, the lifetime of its longest-lived point.

As with the total persistence, there is a maximum persistence $MaxPers_k$ for each homology dimension k .

Example 2.11 (Maximum persistence). The maximum persistence of the same diagram D (Figure 2.23) in a given dimension is simply the lifetime of its longest-lived point in that dimension.

- Maximum persistence in dimension $k = 0$: within that dense cluster of red H_0 points near the origin, one point sits (just barely) furthest from the diagonal — that single point's lifetime, 0.03, is the maximum persistence in this dimension, even though it is hard to pick out by eye among

so many overlapping points.

- Maximum persistence in dimension $k = 1$: the single H_1 point is the longest-lived one, so the maximum persistence coincides with the total persistence of Example 2.10, $MaxPers_1 \approx 2.00$.

A third, even simpler summary just counts how many points there are.

Definition 2.15 (Number of points). For a persistence diagram $D_k = \{(b_i, d_i)\}$ in homology dimension k , the **number of points** of D_k is $N_k = \#\{(b_i, d_i) \in D_k\}$.

In words, N_k just counts how many topological features of dimension k were born and died over the filtration. A large N_k means many topological features, so a rather complex filtration, while a small N_k means few topological features, so a cleaner one.

Example 2.12 (Number of points). Counting the points of D (Figure 2.23) in each dimension gives N_k .

- Number of points in dimension 0: the red cluster at the origin looks almost like a single dot, since so many points overlap there, but it is really 991 separate points — one per point of the original point cloud, each merging into a larger component almost immediately.
- Number of points in dimension 1: only 1 point stands out clearly from the diagonal, and it is the only one — $N_1 = 1$.

Total persistence, maximum persistence and the number of points are useful, but they throw away almost everything about how the lifetimes are distributed — two very different diagrams can easily have the same total, the same maximum, or the same number of points. A finer summary, still a single number, is given by the **persistent entropy**.

Definition 2.16 (Persistent entropy). For a persistence diagram $D = \{(b_i, d_i)\}_{i=1}^m$ with point lifetimes $\{l_i\}_{i=1}^m$ and total persistence $L = \sum_{i=1}^m l_i$, the **persistent entropy** of D is

$$E_k = - \sum_{i=1}^m \frac{l_i}{L} \log\left(\frac{l_i}{L}\right).$$

Again there is a persistent entropy E_k for each homology dimension k .

Notice that this E_k is exactly the Shannon entropy, which is a quantity that in general measures the average uncertainty, randomness, or surprise in a set of data or random outcomes.²⁰

Example 2.13. Figure 2.24 shows two persistence diagrams we'll illustrate in Section 3: the pure cosine's diagram on the left, and the pure noise's diagram on the right^a. Qualitatively, looking at their dimension-1 points (the loops):

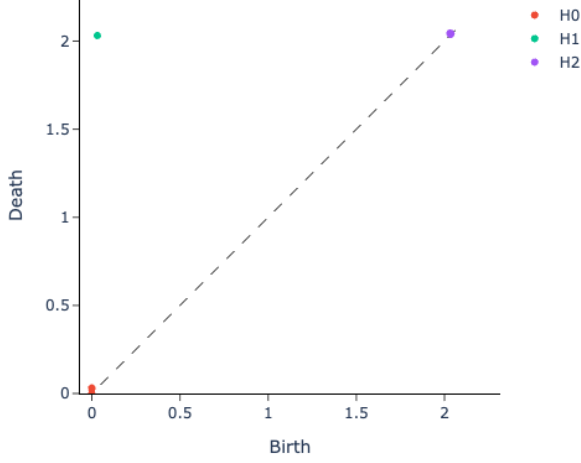
- The persistent entropy E_k is **high** when the point lifetimes are all roughly comparable to one another: no single point dominates the diagram, so the “distribution” of lifetimes is close to uniform, and (as for any Shannon entropy) a close-to-uniform distribution is exactly what maximizes entropy. This is the case for the diagram on the right (pure noise), whose persistent entropy in dimension $k = 1$ is $E_1 = 8.39$.
- The persistent entropy E_k is **low** when one or a few points are much longer-lived than all the rest: the diagram's persistence is concentrated in those few dominant points, and a concentrated distribution has low entropy. This is the case for the diagram on the left (pure cosine), whose persistent entropy in dimension $k = 1$ is $E_1 = 0.00$, since a single loop completely dominates

²⁰An intuitive discussion of Shannon entropy on its own, away from persistence diagrams, is given in Appendix A.33.

the diagram.

²¹Is not necessary at this point to really know precisely what is the pure noise's persistence diagram: what is important is that the 2 diagrams of figure 2.24 are different diagrams.

Persistence Diagram — Pure cosine



Persistence Diagram — Pure noise

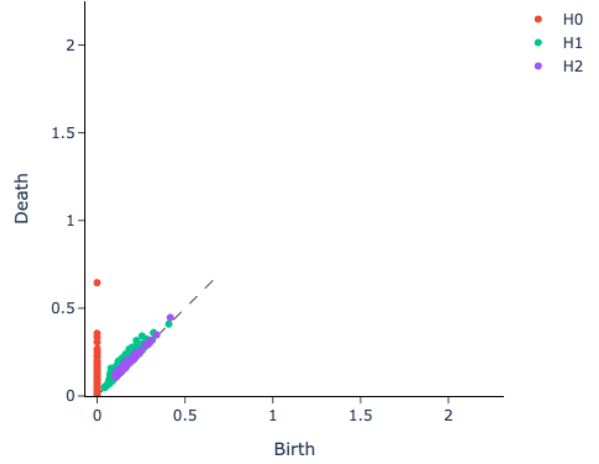


Figure 2.24: Two persistence diagrams with low (left, pure cosine) and high (right, pure noise) persistent entropy (Example 2.13).

Persistent entropy is a scale invariant summary function which is stable under small perturbations.²¹

2.4.2 Persistence images

A single real number, however cleverly chosen, is a very coarse summary of a persistence diagram: it necessarily discards a lot of information. **Persistence images** instead summarize a diagram by a whole vector, obtained by turning the diagram into an actual image.²²

Figure 2.25 shows the sequence of steps that turns a persistence diagram into a persistence image.

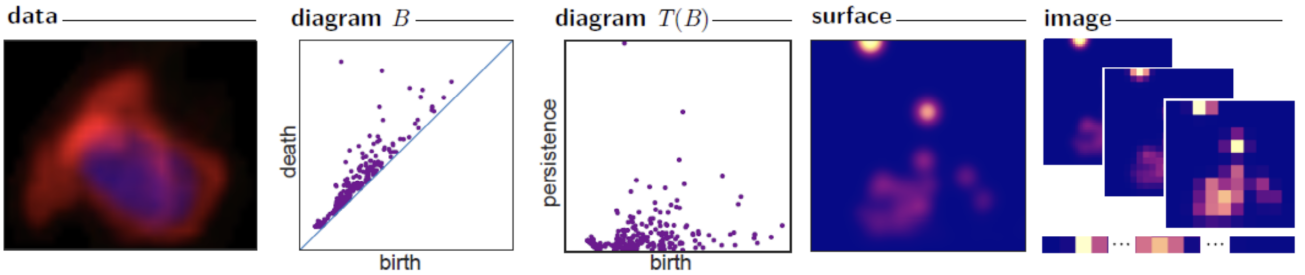


Figure 2.25: The pipeline from data to persistence image [37]. Notice there is a very persistent point in the persistence diagram (namely a long-lived topological feature, far away from the diagonal), this point is weighted up in the persistence surface and shows up as a bright pixel in the persistence image on the right.

The construction is taken from [37]; here is the same idea told more intuitively:

- First step: re-draw the diagram in **birth–persistence** coordinates instead of birth–death coordinates: replace each point (b, d) by $(b, d - b)$, namely by its birth time and its persistence (called

²¹This is because persistence diagrams are themselves robust under perturbations of the input, as we discussed in Section 2.2.2: since the persistent entropy E_k is computed directly from the point lifetimes of D , a small change to D (in the bottleneck distance) only changes those lifetimes slightly, and hence only changes E_k slightly too. A more complete discussion of persistent entropy can be found in [38].

²²More precisely, persistence images are a stable (with respect to the 1st-Wasserstein distance) and interpretable finite-dimensional vector representation of persistence diagrams; for a complete discussion see Adams et al. [39].

also lifetime) rather than its birth and death times. This has the effect of flattening the diagonal of the persistence diagram 1.36 onto the horizontal axis, so that persistence is now read directly as height above that axis.

- Second step: place a small smooth “bump” (a probability distribution) centred at every point of this new diagram, and weight each bump by how important we consider that point: typically, points close to the horizontal axis — namely short-lived, likely-noise features — are weighted down or ignored entirely, while long-lived points are weighted up.
- Third step: add up all these weighted bumps into a single smooth surface over the birth–persistence plane, called the **persistence surface**: it is high wherever many important (long-lived) points cluster together, and low elsewhere.
- Fourth step: chop this surface into a grid of small boxes, or pixels, and record how much of the surface sits above each pixel. The result is literally a small image, the **persistence image**, whose pixel intensities can be read off as a vector and fed directly into standard statistical or machine-learning tools.

The fully formal version of this construction, following [37], is given in Appendix A.32.

2.4.3 Betti curves and area under the Betti curve

Recall the thickenings S^t , $t \geq 0$, of a point cloud S — obtained by replacing every point of S with a small ball of radius t and taking the union of all these balls, so that as t grows, the balls swell and progressively merge together (Definition 2.1) — like the ones in Figure 2.26.

- In Figure 2.26 we build three thickenings of the same point cloud S .
- Notice that the scale t of these thickenings is increasing progressively from left to right.
- For the first thickening, the balls have not yet started to overlap, so the number of connected components equals the number of points, $\beta_0 = 30$; and since no overlap of balls forms a loop yet, there are no 1-dimensional holes, $\beta_1 = 0$.²³
- For the second thickening, all the balls overlap into a single connected component, $\beta_0 = 1$; but the overlap of balls still does not form a complete closed loop, so there are still no 1-dimensional holes, $\beta_1 = 0$.
- For the third thickening, we again have a single connected component, $\beta_0 = 1$; but this time the overlap of balls does form a complete loop, so there is one 1-dimensional, “circular” hole, $\beta_1 = 1$.

²³Formally, to compute the Betti numbers we need to compute the homology of the thickenings: we have already studied how to compute the homology of these thickenings, via the Čech and Rips complexes of Section 2.1.1, Section 2.1.2 and Section 2.1.3.

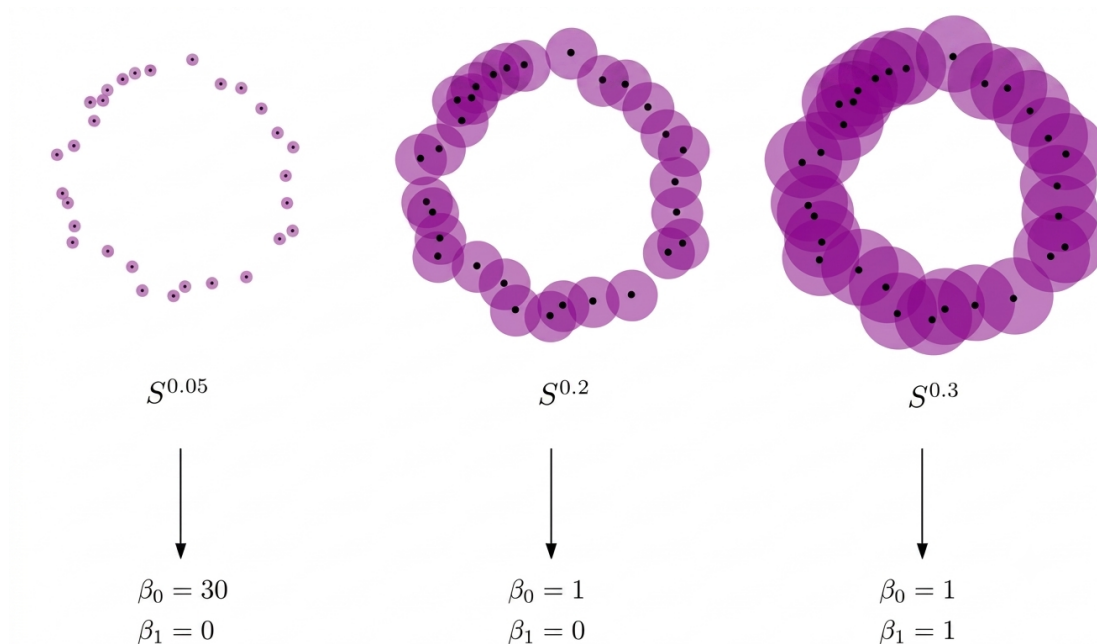


Figure 2.26: The Betti numbers β_0, β_1 of three thickenings of the same point cloud S , at three different scales. Image has been collected from [30].

In practice, rather than reading off the Betti numbers of just a handful of thickenings as in Figure 2.26, we would like to track betti numbers $\beta_i(S^t)$ as the scale t grows continuously. This is exactly what the **Betti curve** records.

Definition 2.17 (Betti curve). Given a generic filtration (Definition 1.34), such as a Rips filtration, the **Betti curve** is simply the function that tells us, as the scale parameter t grows:

- $\beta_0(t)$: the evolution of the number of connected components with the scale parameter t ;
- $\beta_1(t)$: the evolution of the number of loops with the scale parameter t ;
- $\beta_2(t)$: the evolution of the number of 3-dimensional cavities with the scale parameter t ;

and in general, the i -th **Betti curve** $\beta_i(t)$ is the evolution of the number of i -dimensional holes with the scale parameter t .

Notice that in a filtration the components can only merge, so we expect the Betti curve $\beta_0(t)$ (i.e. the number of connected components) to be a decreasing function.

Example 2.14 (Betti curves of a circular point cloud). Consider a different point cloud S from the one of Figure 2.26: this time, take points sampled almost perfectly along a circle, shown in Figure 2.27.

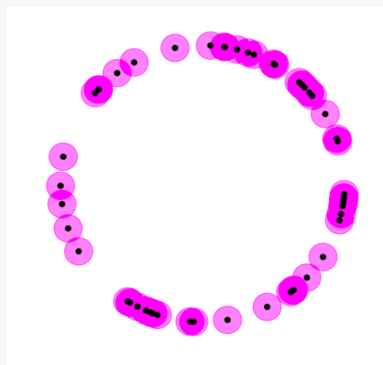


Figure 2.27: The circular point cloud S of Example 2.14. Image has been collected from [30].

As we thicken the point cloud S by growing the scale t , its balls start out disjoint and then this balls progressively merge. Computing the Betti curves of this thickening, shown in Figure 2.28, makes this evolution precise:

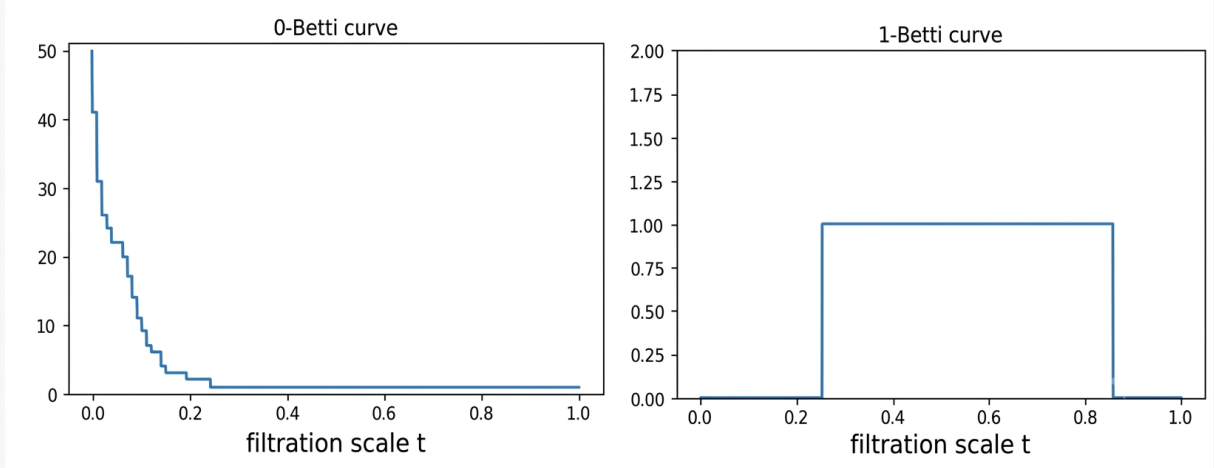


Figure 2.28: The Betti curves $\beta_0(t)$ (left) and $\beta_1(t)$ (right) of the thickenings of the circular point cloud of Figure 2.27 [30]. Image has been collected from [30].

- On the left, the 0-th Betti curve $\beta_0(t)$ tracks the number of connected components: as we already argued above, it is a decreasing function, starting at $\beta_0 = 50$ (one component per sample point, before any balls overlap) and dropping down to $\beta_0 = 1$ once every ball has merged into a single component.
- On the right, the 1-st Betti curve $\beta_1(t)$ tracks the number of loops: it stays at 0 until a single loop appears at $t \approx 0.25$, once the thickening has closed up into a ring around the circle's hole (as shown in the figure); it then stays at $\beta_1 = 1$ while that loop survives, and drops back to 0 once the hole itself fills in, at $t \approx 0.85$.

We can also compute the area under these (Betti) curves. Actually we do something more, we split the range of t into a handful of equal-width windows and, in each window, we compute a single number: the area underneath the curve in that precise window.

Definition 2.18 (Area under the Betti curve). Fix a Betti curve $\beta_i(t)$ (Definition 2.17) on an interval $[0, t_{\max}]$, and split this interval into n equal windows $[t_0, t_1], [t_1, t_2], \dots, [t_{n-1}, t_n]$, with $t_0 = 0$ and $t_n = t_{\max}$. The **area under the Betti curve** (or **partial integral**) over the k -th window is

$$A_i^{(k)} = \int_{t_{k-1}}^{t_k} \beta_i(t) dt,$$

namely the area under the graph of β_i restricted to that window — in words, how much that homological feature (connected components for $i = 0$, loops for $i = 1$, and so on) was present during that stretch of the filtration. Splitting β_i into n such areas turns one curve into a short list of n numbers, one per window.

For instance, splitting the 0-th Betti curve $\beta_0(t)$ of Figure 2.28 into two equal windows $[0, 0.5]$ and $[0.5, 1]$, and recalling that it reaches $\beta_0 = 1$ by $t \approx 0.25$:

- since $\beta_0(t)$ drops from 50 to 1 entirely within the first half of the range, the first window $[0, 0.5]$ already captures most of the curve's decline;
- the second window's area is instead close to the minimum possible, since $\beta_0(t) \approx 1$ there, essentially constant.

The two areas together give a compact, two-number summary of how quickly the point cloud merged into a single piece.

Chapter 3

Topological Data Analysis for Periodic and non-Periodic Signals

The previous chapters developed, one construction at a time, the machinery of persistent homology: simplicial complexes, filtrations, homology, Betti curves, and real-valued summaries such as persistent entropy. All of this was, so far, mathematics on paper. This chapter closes that gap: we take a simple, fully controlled signal — a cosine — and walk through, in Python, exactly how the constructions of the previous chapters are computed in practice, from a raw time series to a topological summary.

We choose a cosine because it is the simplest possible periodic signal: its shape, and the shape of the point cloud obtained from it, are known in advance, so at every step we can check that the code is doing what the theory predicts, before applying the same pipeline, in the next chapter, to a real gravitational-wave signal where the answer is not known ahead of time.

GUDHI and Giotto-tda are two of the most important open-source software libraries for Topological Data Analysis. They are closely related, both technically and historically: Giotto-tda integrates and builds on internal C++ components of GUDHI [40], alongside other high-performance backends such as Ripser and Hera [34].

3.1 The toy model

Before going through the code in detail, here is, in plain terms, what it is going to do:

- **Start from a cosine signal.** We build a short, time series which is a cosine wave,

$$x(t) = A \cos(2\pi f_0 t),$$

oscillating with a fixed amplitude A at a single fixed frequency f_0 — a clean, fully known signal, so that at every later step we can check the result against what we already expect.

- **Turn the signal into a point cloud.** A time series is just a list of numbers, with no shape a topological method could act on. We fix this with a Takens embedding (Section 2.3.1): a way of turning a single sequence of measurements into a cloud of points in space by grouping together several consecutive (suitably spaced) values of the signal, so that the shape of this cloud reflects how the signal behaves.
- **Build a Rips filtration on the point cloud.** Once we have a point cloud, we grow a ball of radius t around every point and connect points whose balls overlap, exactly as in Section 2.1.2 — letting t increase produces the filtration whose persistent homology we will compute.
- **Extract topological summaries.** From the resulting persistence diagram we compute the quantities introduced in the previous chapter (Section 2.4.1, Section 2.4.3): persistent entropy, Betti curves, and a few simple numerical statistics (such as the number of points in the diagram and their total persistence).

With this outline in mind, we now go through the code itself, step by step: in what follows we comment selected extracts of code, chosen and explained to show, in practice, how TDA can be applied to real

data, rather than reproducing the notebook line by line. The full code is reproduced in Appendix C, for anyone who wants to see it in its entirety.

3.2 From signal to point cloud

We work with three raw timeseries, each 4 seconds long and sampled at 2048 Hz:

- a short burst of a **pure cosine wave**: a clean, fully periodic signal;
- **pure white Gaussian noise** of the same length: a signal with no periodic structure at all;
- their **sum**, a noisy version of the periodic signal.

Building these three signals is straightforward, and we describe it here only in words rather than in code:

- the **cosine burst** is generated by evaluating a cosine of a fixed frequency and amplitude at every sample time. In total this cosine signal is of 4 seconds and is sampled at 2048 Hz.
- the **white noise** is generated by drawing one independent Gaussian random number, with a fixed standard deviation, for every sample time of the same 4-second window.
- the **sum** is simply the two signals above added together, sample by sample.

No code is shown for this step, since we can create these timeseries in many ways and this is not specific to topological data analysis.

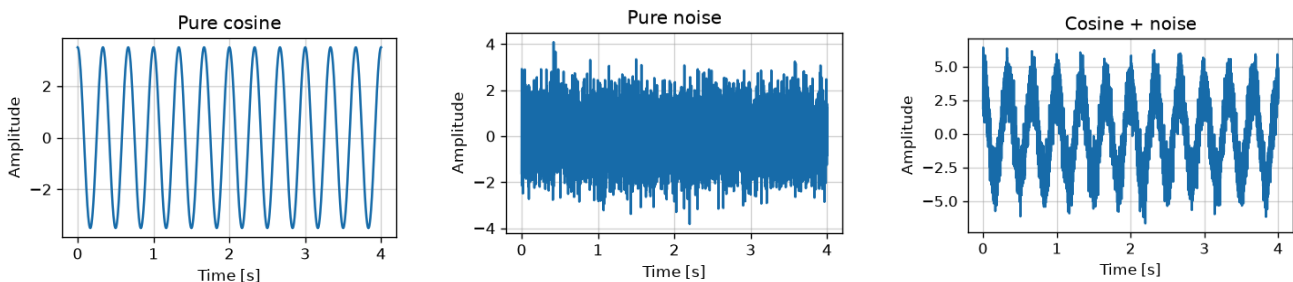


Figure 3.1: The three raw timeseries used throughout this chapter: pure cosine burst (left), pure white Gaussian noise (centre), and their sum (right).

None of these three signals is, by itself, a point cloud. We turn each of them into one with the Takens embedding introduced above. Two choices are involved:

- **the embedding dimension**: for now we fix it by hand at 3, so that the resulting point clouds can be plotted directly and inspected by eye;
- **the time delay** (how far apart, and how densely, the grouped samples are taken): these are instead taken from the automatic search of Section 3.5, discussed later in this chapter.

Concretely, the automatic search of Section 3.5 is run once, ahead of everything in this section, on the three raw timeseries; this produces a single shared pair of values for the time delay and the embedding dimension, reused for all three signals here, so that the three resulting point clouds are directly comparable to one another.

The Takens embedding is not applied one timeseries at a time: `giotto-tda`'s `TakensEmbedding` instead expects the whole collection of timeseries to embed, gathered into a single Python list, and embeds every one of them independently. We first collect our three raw signals into such a list,

```
1 list_timeseries = [pure_cosine, pure_noise, cosine_plus_noise]
```

and only then call the embedder on the whole list at once:

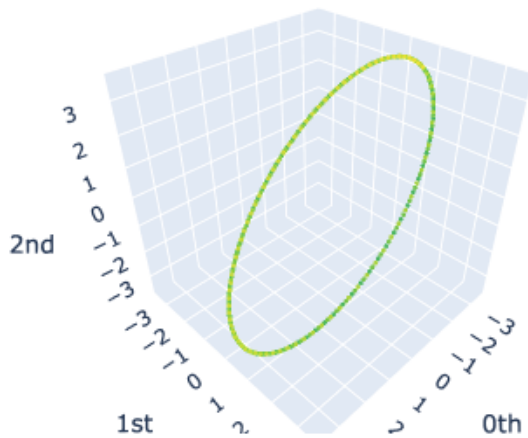
```

1 embedder_dim3 = TakensEmbedding(
2     time_delay=TIME_DELAY,    # from the search of Sec. 3.5, shared across
3     dimension=3,              # forced by hand, so the cloud can be plotted
4     stride=STRIDE,
5 )
6 raw_dim3_clouds = embedder_dim3.fit_transform(list_timeseries)

```

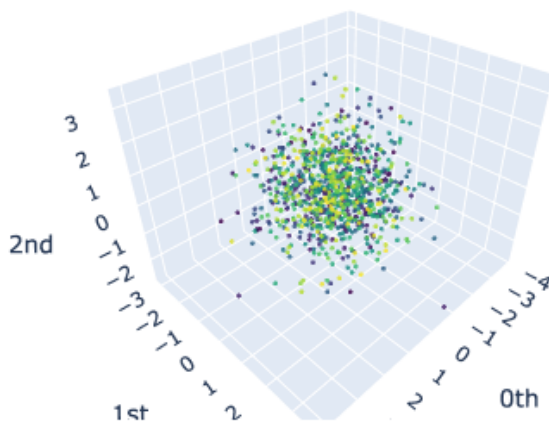
The result, `raw_dim3_clouds`, is again a list of three point clouds, one per input timeseries and in the same order, each living in $\mathbb{R}^3$.

Even before computing anything, these three point clouds already hint at what persistent homology should find.



For the **pure cosine** signal, being periodic, we expect the embedding to trace out a loop; more precisely, an ellipse, exactly as in the worked example of Example 2.6, where we already saw a cosine signal embed into an ellipse. We expect TDA to detect this loop as a single long-lived 1-dimensional feature in the persistence diagram.

Figure 3.2: Point cloud obtained by embedding the pure cosine signal in dimension 3.



For the **pure noise** signal, having no repeating structure at all, we instead expect a shapeless cloud. Indeed no visible patterns or genuine long-lived loops are noticed.

Figure 3.3: Point cloud obtained by embedding the pure white-noise signal in dimension 3.

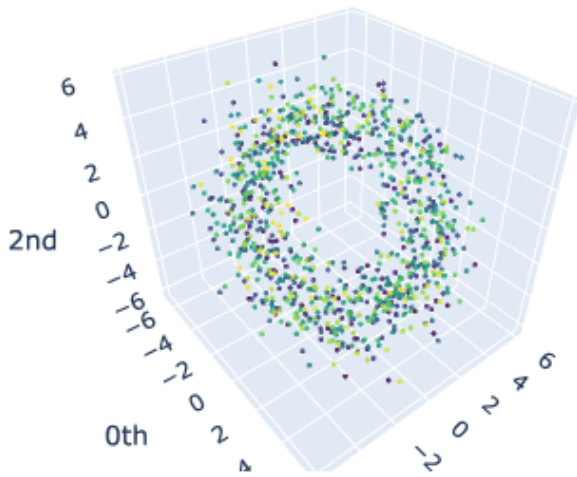


Figure 3.4: Point cloud obtained by embedding the cosine+noise signal in dimension 3.

For the **cosine+noise** signal, we expect something in between: a noisy version of the same loop seen for the pure cosine, and in fact we can see the loop, but perturbed and thickened, like a noisy thickening of the ellipse. Because of this noise, we expect the persistent homology to detect a few short-lived 1-dimensional features (**topological noise**) but at the same time to still detect the genuine loop as a long-lived feature, exactly as in the pure cosine case.

3.3 Computing persistent homology

With a point cloud in hand, we need to choose a filtration to compute its persistent homology on; we decide to build a **Rips filtration** on it, exactly the construction of Section 2.1.2. Briefly and intuitively, recall that building a Rips filtration on a point cloud means:

- growing a ball of radius t around every point of the cloud;
- connecting two points as soon as their balls start to overlap;
- letting the scale t increase progressively, so that more and more points end up connected, and tracking how the resulting complex's Betti numbers change as t grows.

Giotto-tda's `VietorisRipsPersistence` does this in a single call. What this function computes is exactly the study of how the connected components, the loops, the voids¹ (and so on) evolve as the filtration scale t grows — in other words, persistent homology, computed here in the very same way we already computed it by hand in Section 1.6.3:

```
1 persistence = VietorisRipsPersistence(
2     homology_dimensions=HOMOLOGY_DIMENSIONS, # [0, 1, 2]: which Betti
3     max_edge_length=global_max_distance,      # largest scale t the
4     collapse_edges=BOOL_COLLAPSE,            # speeds up the Rips
5     n_jobs=-1,                               # parallelize over point clouds
6 )
7 list_PH_diagrams = persistence.fit_transform(point_clouds)
```

Each parameter here is something we have already met:

- `homology_dimensions=[0, 1, 2]` fixes which Betti numbers we want, β_0 , β_1 and β_2 (Definition 2.17) — more precisely, it tells the algorithm to compute only the 0-dimensional homological features (connected components), the 1-dimensional ones (loops), and the 2-dimensional ones (cavities), and nothing beyond that. This parameter matters more than it might look: asking for one extra dimension, e.g. `homology_dimensions=[0, 1, 2, 3]`, can increase the computation time substantially, because detecting a k -dimensional hole requires the algorithm to examine simplices of dimension k and $k + 1$ built out of the point cloud, and the number of such higher-dimensional simplices grows very fast (combinatorially) with the number of points — so every extra homological dimension we ask for is noticeably more expensive than the last.

¹For example if points arrange in the surface of a sphere those points enclose a void.

- `max_edge_length` is the largest scale t the Rips filtration (Section 2.1.2) is grown to; here it is set to the largest pairwise distance actually present in the point cloud, so the filtration is grown just far enough to see every point merge into a single component. It is possible to set `max_edge_length` to a smaller value instead, to make the computation run faster — but at a cost: any hole that is only born, or only dies, at a scale t beyond this cutoff will be missed entirely, since the filtration is never grown far enough to see it.
- `collapse_edges=True` is a purely computational shortcut — it removes redundant edges during construction without changing the resulting homology, letting the Rips complex be built faster [34]. This edge-collapse step turns out to be remarkably effective at cutting down computation time while leaving the homology exactly unchanged; a closer discussion and some timing tests are left to Section 3.6.
- `n_jobs=-1` asks for the point clouds to be processed in parallel across several CPU cores at once, rather than one after another; concretely, with our three point clouds (cosine, noise, cosine+noise) this means a machine with (at least) three CPU cores can compute all three persistent homologies simultaneously, one per core, instead of waiting for each to finish before starting the next. This is purely a matter of computational speed, and has nothing to do with the topology itself.

The output, `list_PH_diagrams`, collects one persistence diagram per point cloud. Concretely:

- it is a list of three diagrams, one for the cosine signal, one for the noise, and one for the sum, in the same order as the input point clouds — i.e. we mean `list_PH_diagrams = [diagram_pure_cosine, diagram_pure_noise, diagram_cosine_plus_noise]`;
- each diagram is itself a list of triples (b, d, k) , one per topological feature found: b is the birth scale, d is the death scale, and $k \in \{0, 1, 2\}$ says which homological dimension the feature belongs to (a component, a loop, or a cavity).²

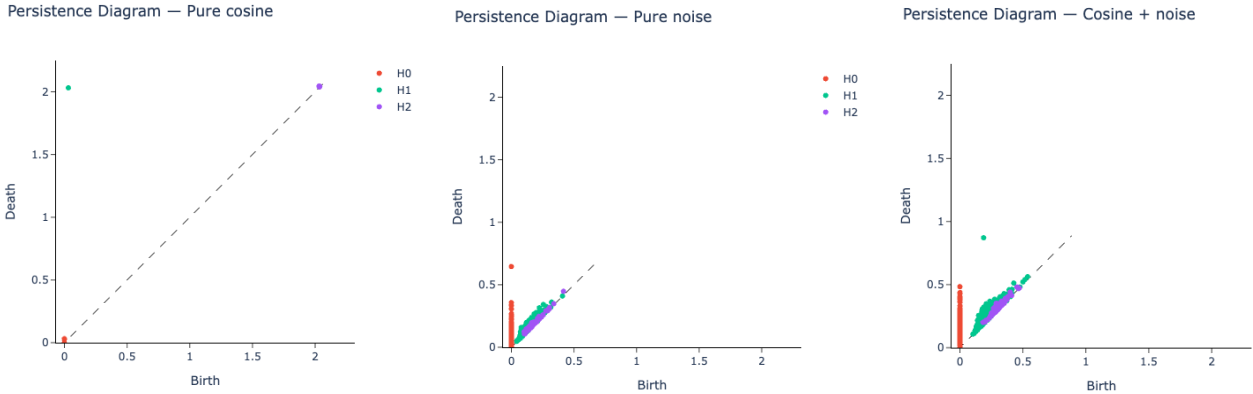


Figure 3.5: Persistence diagrams of the dimension-3 Takens embeddings of Section 3.2: pure cosine (left), pure noise (centre), cosine+noise (right).

Reading these three diagrams side by side:

- **Left, pure cosine.** A single green point sits clearly apart from the rest, in the upper-left region of the diagram: this is the 1-dimensional feature (a loop, H_1) coming from the ellipse of Figure 3.2. It is born early in the filtration scale t and dies very late, so — as expected for a genuine feature of the signal — it has a long lifetime, i.e. it is highly persistent. There are also a few violet points close to the diagonal: these are 2-dimensional cavities (H_2). Being so close to the diagonal, they are born and almost immediately filled back in, so they are trivial, short-lived features; in practice these arise as a spurious side effect of the numerical computation, not as genuine cavities we would expect to find in a loop-shaped point cloud.

²A small practical wrinkle: some features never die within the range explored, and are recorded with an infinite death time. Since downstream code cannot work with infinities, every such death is capped at the largest *finite* death time observed, before the diagrams are used any further — a bookkeeping fix for the code, not a change to the underlying mathematics; the features being capped are exactly the ones recorded as infinite bars $[s, \infty)$ in Section 1.6.2, features that survive to the end of the filtration without ever dying.

- **Centre, pure noise.** Here the points — of every homological dimension, components, loops and cavities alike — are clustered close to the origin and roughly along the diagonal: they are all born and die at similarly small scales t , so all of them have short lifetimes. This is exactly **topological noise**: many features, none of them long-lived. It is exactly what we would expect from this signal, since pure white noise embeds into a shapeless, amorphous point cloud (Figure 3.3) with no genuine geometric structure to detect.
- **Right, cosine+noise.** This diagram looks like a combination of the two diagrams just described: the same cluster of short-lived topological noise near the origin and the diagonal, together with one clearly persistent green point standing above that cluster — the same genuine loop already seen for the pure cosine, now surviving inside the noisy, donut-like point cloud of Figure 3.4.

3.4 Topological summaries

A persistence diagram is already a topological summary of the point cloud, but for many purposes — comparing many signals at a glance, or feeding the result to further numerical processing — it helps to compress each diagram further, into a handful of numbers or a fixed small image. We now compute, on our three persistence diagrams, exactly the summaries introduced in Section 2.4.1, Section 2.4.3 and Section 2.4.2.

3.4.1 Scalar summaries

To compute persistent entropy, maximum persistence, total persistence, and the number of points, we do not need anything more than the list of persistence diagrams, `list_PH_diagrams`, built in Section 3.3: giving it as input to

```
1 features_of_homology = topological_features.fit_transform(list_PH_diagrams)
```

computes all four scalar summaries at once, where `topological_features` bundles the needed functions together as a single `FeatureUnion`:

```
1 topological_features = FeatureUnion([
2     ("entropy", PersistenceEntropy(normalize=False)),           # Def.
3     ("maxPersistence", compute_max_persistence_transformer),    # Def. maximum
4     ("totalPersistence", compute_total_persistence_transformer), # Def. total
5     ("numFeatures", NumberOfPoints(n_jobs=-1)),                # Def. number
6 ])
```

`FeatureUnion` is a scikit-learn tool that does nothing topological by itself: it simply takes several transformers — here, `PersistenceEntropy`, `compute_max_persistence_transformer`, `compute_total_persistence_transformer` and `NumberOfPoints` — and runs all of them on the same input, collecting their outputs side by side into a single array. Each of the four transformers computes exactly one of the quantities already defined: the persistent entropy of Definition 2.16, the maximum persistence of Definition 2.14, the total persistence of Definition 2.13, and the number of points of Definition 2.15 — each computed once per homology dimension, so that `features_of_homology` collects, for every signal, one row of $4 \times 3 = 12$ numbers (four summaries, three homology dimensions).

Each of these four summaries is actually computed once per homology dimension H_0, H_1, H_2 , giving three numbers per summary per signal; for brevity, in what follows we report and discuss only the values in dimension H_1 (loops), since this is the dimension where the genuine feature of interest — the loop — lives, and where the contrast between the three signals is most informative.

H_1 summary	Cosine	Noise	Cosine+noise
Persistent entropy	0.00	8.39	8.19
Maximum persistence	2.00	0.09	0.68
Total persistence	2.00	9.60	15.87
Number of points	1	483	449

Table 3.1: The four H_1 (loop) scalar summaries of `features_of_homology`, for the three signals.

- **Persistent entropy** is exactly 0 for the pure cosine, since there is a single, completely dominant loop (recall from Definition 2.16 that entropy is 0 precisely when all the persistence is concentrated in one feature); it is instead large for pure noise, since many loops of comparable, short lifetimes share the persistence among themselves; cosine+noise falls in between, closer to the noise value, since entropy is quite sensitive to the presence of many small noisy loops even when one genuine loop is also present.
- **Maximum persistence** shows the pure cosine’s single loop dominating completely, with a lifetime of 2.00, against just 0.09 for the most persistent loop of pure noise. Cosine+noise sits in between, at 0.68: still clearly closer to the pure cosine’s genuine loop than to pure noise, but visibly reduced from 2.00 — a sign that the added noise perturbs the loop enough to shorten its lifetime, without destroying it.
- **Total persistence**, being a *sum* over every loop in the diagram rather than just the longest-lived one, tells a different story: it is largest for cosine+noise, since summing adds up the many small, short-lived noisy loops on top of the one genuine, long-lived loop already present in the pure cosine.
- **Number of points**, in dimension H_1 , simply counts how many points — i.e. how many loops born at some point in the Rips filtration on the point cloud — appear in the persistence diagram, regardless of how long they persist. Here there is the same contrast: only 1 loop for the pure cosine, against 483 and 449 for noise and cosine+noise respectively — a direct numerical confirmation that the pure cosine’s point cloud contains a single, clean loop, while the other two are full of many small, spurious ones.

3.4.2 Betti curves and their area

Recall from Definition 2.17 that a Betti curve simply tracks, as the scale t of the Rips filtration grows, how a Betti number evolves:

- $\beta_0(t)$ counts the number of connected components, at every value of t ;
- $\beta_1(t)$ counts the number of loops, at every value of t ;
- $\beta_2(t)$ counts the number of cavities, at every value of t .

The Betti curves are computed with a single `BettiCurve` transformer:

```
1 BC = BettiCurve(n_bins=N_BC_BINS, n_jobs=-1)
2 list_bc = BC.fit_transform(list_PH_diagrams)
```

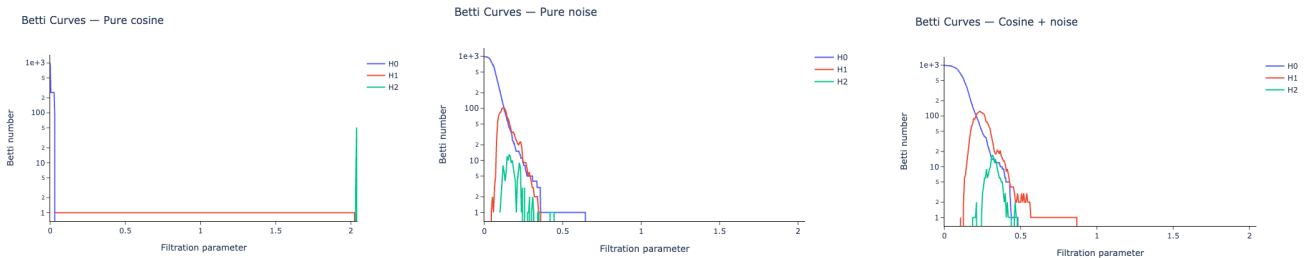


Figure 3.6: Betti curves of the dimension-3 Takens embeddings of Section 3.2, plotted with a logarithmic y -axis: pure cosine (left), pure noise (centre), cosine+noise (right).

Reading these three panels, and focusing on the red curve, $\beta_1(t)$, the one that counts loops:

- **Left, pure cosine.** The red curve sits at a constant 1 across almost the whole range of t : a single loop is created early in the filtration and survives all the way to the end — the big loop of the ellipse already seen in Figure 3.2.
- **Centre, pure noise.** The red curve instead spikes up and back down repeatedly, never settling: many short-lived loops are created and destroyed throughout the filtration, none of them surviving for long — exactly the topological noise already discussed for the point cloud of Figure 3.3.
- **Right, cosine+noise.** The red curve looks like a combination of the two: the same short-lived spikes of topological noise, but this time sitting on top of a curve that also settles at 1 for a long stretch — the genuine loop of Figure 3.4, perturbed but still present.

A Betti curve, being a whole function of t , is still an awkward object to compare across many signals at a glance — so, as in Section 2.4.3, we replace it by its area under the curve, computed one window at a time:

- we cut the range of the scale t into `N_INTEGRALS` equal windows;
- in each window, we replace the Betti curve by the area underneath it there, i.e. by how much of that homological feature (a component, a loop, or a cavity, depending on the dimension) was present, on average, during that stretch of the filtration;
- doing this for every window turns one whole curve into a short list of numbers, one area per window, which is much easier to compare across different signals than the full curve.

As just seen in Figure 3.6, only the pure cosine’s and the cosine+noise’s loops survive to high values of t ;³ pure noise’s do not. This is exactly what makes the area under $\beta_1(t)$ in the last window discriminative: non-zero for the two signals with a genuine loop, 0 for pure noise. Computing this is exactly what `betti_partial_integrals` does; we do not show the code itself, since computing the area under a curve numerically can be done in Python in many equivalent ways, none of them specific to topological data analysis.

3.4.3 Persistence images

Recall from Section 2.4.2 that a persistence image simply turns a persistence diagram into a small picture, by smoothing and weighting its points and then reading off pixel intensities. We focus here on one particular way of weighting the points, obtained with:

```
1 PI_quad = PersistenceImage(sigma=PI_SIGMA, n_bins=PI_N_BINS,
2                             weight_function=lambda x: x**2, n_jobs=-1)
```

Recall from Section 2.4.2 that weighting is exactly the second step of building a persistence image: every point of the diagram is first turned into a small, smooth bump, and the argument `weight_function` decides how tall that bump is allowed to be, as a function of the point’s lifetime x (its persistence, i.e. how long that topological feature survived in the filtration). Here `weight_function=lambda x: x**2` says: take the lifetime x of the point, and square it, and use that squared value as the height of its bump. Two consequences follow from squaring rather than, say, leaving the lifetime unweighted:

- a point with a very short lifetime, close to the diagonal of the persistence diagram, gets an even smaller squared value, so its bump ends up almost flat — it contributes almost nothing to the final picture;
- a point with a long lifetime, far from the diagonal, gets a much larger squared value than its lifetime alone would give it, so its bump becomes tall and prominent, standing out clearly in the final picture.

³Formally, we should say that these loops survive up to high values of the filtration scale t , rather than “the end,” since the filtration does not literally terminate.

For example, the persistence diagram of the pure cosine (Figure 3.5, left panel) has a very persistent point in dimension H_1 : the loop of the ellipse-shaped point cloud. Being so long-lived, this point gets a large squared weight, and so it is exactly this point that maps into a single, very bright region of the resulting persistence image, clearly standing out from the rest of the (dim) picture:

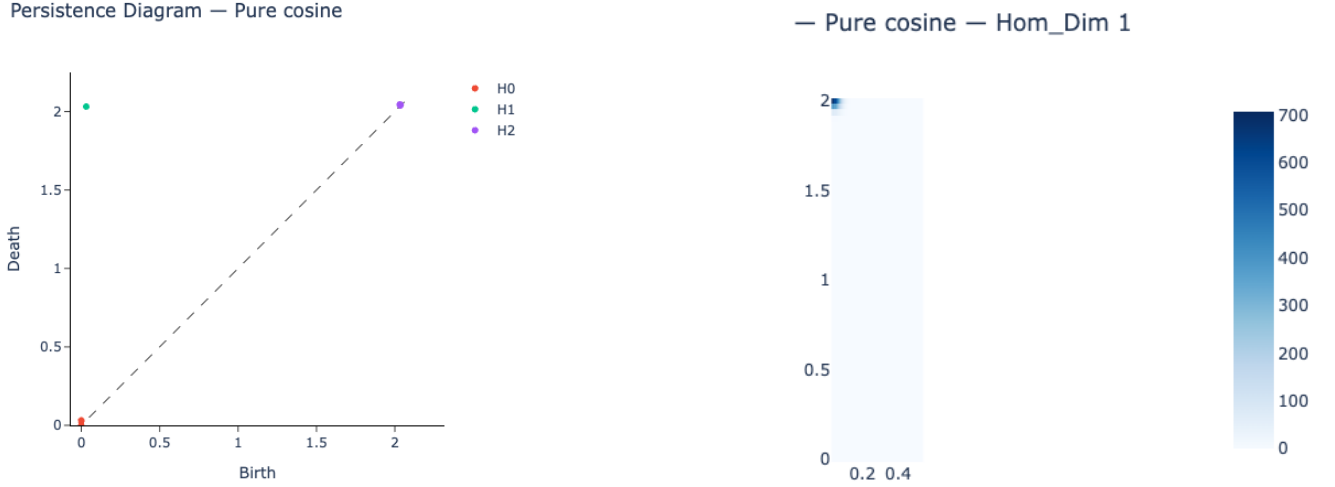


Figure 3.7: The persistence diagram of the pure cosine signal (left) and the persistence image obtained from it with quadratic weighting (right): the one persistent point becomes the one bright region.

In other words, squaring the lifetime does not just favour long-lived points over short-lived ones, as any increasing weighting would: it favours them more aggressively than a weighting proportional to the lifetime itself would, pushing the short-lived points — the topological noise already discussed in Section 3.2 — further down still, and the genuine, long-lived features further up. This is exactly why we focus on this particular weighting here: it produces the persistence images in which the contrast between genuine features and noise is sharpest. We can also chose different milder weightings (no weighting at all, and a weighting directly proportional to the lifetime) to see them look in Appendix C.

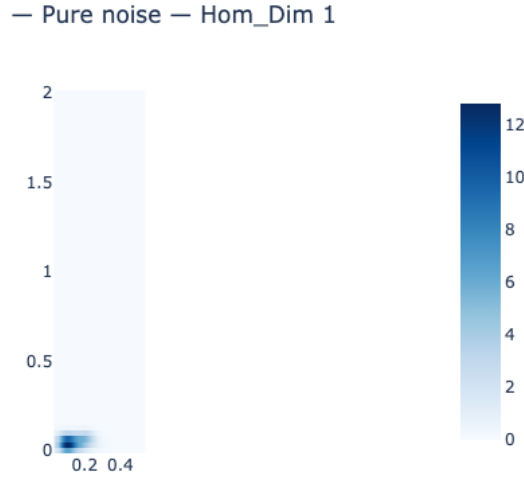


Figure 3.8: Persistence image (β_1 , quadratic weighting) of the pure noise signal.

3.5 Searching for the embedding, without forcing three dimensions

Throughout Section 3.2–Section 3.4 we forced the Takens embedding dimension to 3 by hand, purely so that the resulting point cloud could be plotted directly. Nothing forces this choice: Section 2.3.4 already showed how to let the embedding dimension and time delay be found automatically, by searching for the smallest dimension at which false neighbours disappear and the time delay at which consecutive coordinates carry the least redundant (mutual) information — we only briefly recall this here and point

back to Section 2.3.4 for the full explanation.

To be able to search for the optimal parameters (the embedding dimension d and the time delay τ), we need a different function from `giotto-tda`, `SingleTakensEmbedding`. As the name suggests, this function, like the `TakensEmbedding` we already used in Section 3.2, builds a Takens embedding; unlike it, though, it works on a single timeseries at a time (hence “Single”), and it is capable of searching for the optimal d and τ itself, rather than just applying values we hand it. Setting its `parameters_type` argument to “search” is exactly what turns this search on:

```
1 ste = SingleTakensEmbedding(  
2     parameters_type="search",  
3     time_delay=SEARCH_MAX_TIME_DELAY,  
4     dimension=SEARCH_MAX_EMBEDDING_DIMENSION,  
5     stride=STRIDE,  
6     n_jobs=-1,  
7 )
```

Commenting on some of these lines:

- `parameters_type="search"` activates the search mode described above, so that the optimal embedding dimension d and time delay τ are found automatically, rather than fixed by hand.
- `time_delay=SEARCH_MAX_TIME_DELAY` tells the search the largest time delay τ it is allowed to consider: this upper bound must be set reasonably high, so that the search does not miss the true optimal time delay by stopping short of it.
- `dimension=SEARCH_MAX_EMBEDDING_DIMENSION` plays the same role for the embedding dimension d : an upper bound set high enough that the true optimal dimension is not missed.
- `stride=STRIDE` sets the stride of the Takens embedding, exactly as defined in Section 2.3.1; `n_jobs=-1` simply parallelizes the search across the available CPU cores, and has nothing to do with the topology itself.

Running this search independently on each of the three timeseries finds a different optimal pair (d, τ) for each:

- dimension 4 and time delay 131 for the pure cosine;
- dimension 5 and time delay 72 for pure noise;
- dimension 6 and time delay 139 for cosine+noise.

We then (arbitrarily) decide to use, as the single shared embedding dimension and time delay for all three signals, the median of these three found dimensions and the median of these three found time delays:

```
1 EMBEDDING_DIMENSION = int(np.median(found_dimensions))  
2 TIME_DELAY           = int(np.median(found_time_delays))
```

which gives `EMBEDDING_DIMENSION = 5` and `TIME_DELAY = 131` — using a single shared pair, rather than each signal’s own optimum, is exactly so that the three resulting point clouds are directly comparable Section 3.2. These are the values of `TIME_DELAY` and `EMBEDDING_DIMENSION` used, ahead of time, in Section 3.2; here they are used to build the Takens embedding (recall from Section 2.3.1 that this is exactly what turns a timeseries into a point cloud) of all three signals, this time without forcing the dimension down to 3.

The embedding dimension found this way is typically larger than 3, which is precisely why Section 3.2 forced it down to 3 by hand in the first place: a cloud in more than 3 dimensions cannot be plotted directly. To still visualise it, we fall back on the principal component analysis of Section 2.3.5, used here purely as a visualisation aid:

```
1 pca_plot_clouds = [PCA(n_components=3).fit_transform(cloud) for cloud in  
    embedded_clouds]
```

PCA is the scikit-learn function that performs the principal component analysis already described in Section 2.3.5. Writing `PCA(n_components=3)` tells it that we want to reduce our point cloud `cloud` — a higher dimensional point cloud — down to a 3-dimensional point cloud, so that it can be plotted; more precisely, this is done by computing its first 3 principal components, exactly as detailed in Section 2.3.5. It is important to be precise about what this PCA step is, and is not, used for:

- `pca_plot_clouds` is used *only* to draw the point cloud shown below;
- the persistent homology, Betti curves, area under the curve, and persistence images of Section 3.3–Section 3.4 are instead recomputed, in exactly the same way as before, directly on the original, un-reduced, higher-dimensional embedding, so that no topological information is discarded before the homology is computed.

Let us see in practice what happens to a point cloud once we find the optimal embedding dimension d and time delay τ as explained above: Figure 3.9 shows the pure **cosine’s point cloud**, obtained in two different ways.

- **Left.** The point cloud already presented in Figure 3.2, obtained without the search algorithm, namely with the Takens embedding forced to embed directly in dimension 3. Being the embedding of a cosine, we get, as expected, an ellipse-shaped point cloud.
- **Right.** The point cloud of the same cosine signal, but this time embedded at the searched, optimal dimension $d = 5$, and then projected down to 3 dimensions by PCA, purely so that it can be plotted. The loop structure is still clearly visible, even though this time the embedding is not a perfect ellipse: there is more structure than that, with the points appearing to be distributed over the surface of a cylinder rather than lying flat on a single ellipse. We already discussed, in Section 2.3.1, that a periodic signal like our cosine must trace out an elliptical point cloud whatever the embedding dimension d — so this must be an ellipse too, living in a 5-dimensional space; it must therefore be the PCA projection, from this 5-dimensional space down to 3 dimensions purely for plotting, that produces this cylindrical appearance.



Figure 3.9: The pure cosine’s point cloud embedded at dimension 3 by hand (left, Figure 3.2) versus embedded at the searched dimension $d = 5$ and then projected to 3 dimensions by PCA (right).

This higher-dimensional embedding is also worth pausing on for a moment, since it changes what is even possible to compute:

- before, the point clouds were forced to be 3-dimensional, so we could only investigate up to 3-dimensional cavities, namely H_2 homological features;
- now that the point clouds are built at the optimal embedding dimension 5 (i.e. 5-dimensional point clouds), we could in principle also compute higher homological dimensions, for instance searching for 4-dimensional holes (H_3 elements). Unfortunately, as already discussed in Section 3.3, the computation time for this explodes combinatorially, so this particular computation is not carried out here — but in principle, nothing stops us from doing it.

A priori, though, we do not know in which dimension the dynamics of a given signal actually unfolds — this is precisely why, throughout the rest of this work, we always use the search algorithm to find the optimal embedding dimension and time delay, rather than fixing them by hand.

3.6 Edge collapses

Recall from Section 3.3 that growing a Rips filtration means, at every scale t , connecting points whose balls overlap and then building not just these connections but also every higher-dimensional simplex (triangle, tetrahedron, and so on) that sits on top of them — and that this is exactly why computing higher homological dimensions is so expensive: the number of such simplices grows combinatorially with the number of points. **Edge collapse** is a preprocessing trick that thins out these connections themselves, before any of these higher-dimensional simplices are ever built.

- **What edge collapse does.** Intuitively, some connections are redundant: removing them does not change the overall shape traced out by the rest of the point cloud. Think of a triangle subdivided into several smaller triangles by extra inner edges: erasing one such inner edge, as long as doing so does not disconnect anything or open up a new hole, leaves the same overall triangular shape behind. Edge collapse is exactly the systematic procedure of finding and removing such redundant connections, one at a time.
- **Why it is computationally valuable.** Deciding whether a given connection is redundant in this sense can be done by looking only at which points are connected to which, without ever constructing the higher-dimensional simplices that the Rips complex would otherwise build on top of it. The whole reduction is therefore pure bookkeeping on this web of connections, carried out before the expensive part of the computation (building and processing simplices in high dimensions) ever begins.
- **The theorem.** Boissonnat and Pritam [41] prove that applying edge collapse to every complex of a filtration, one scale at a time, produces a smaller filtration with exactly the same persistence diagram as the original one — the formal result behind Section 3.3's claim that `collapse_edges=True` speeds up the computation "without changing the resulting homology."
- **Why we test it ourselves anyway.** Understanding the proof in full detail is outside the scope of this thesis — it rests on a broader theory of *simple collapses* of simplicial complexes preserving persistence, of which edge collapse is only a special case — so instead we verify empirically, on our own signals, that turning `collapse_edges` on or off really does leave the persistence diagram unchanged, and measure directly the resulting speed-up.

For this test we set aside the cosine signal of the rest of this chapter, and instead use a single, more complex timeseries. Its analytic form is a piecewise function of time t , split at a fixed instant $t = 0$:

$$x(t) = \begin{cases} A \left(\frac{t_{\text{reg}}}{t_{\text{reg}} - t} \right)^{0.25} \cos(\phi_0 - \alpha(t_{\text{reg}} - t)^{0.625}), & t \leq 0, \\ A e^{-t/\tau} \cos(\phi_0 + 2\pi f_0 t), & t > 0. \end{cases}$$

In words: before $t = 0$ both the amplitude and the frequency of the oscillation grow, reaching their largest values exactly at $t = 0$; after $t = 0$ the amplitude instead decays exponentially, at a rate set by τ , while the oscillation settles to the fixed frequency f_0 . The two pieces are built to match up at $t = 0$, so the timeseries has no jump at the transition.

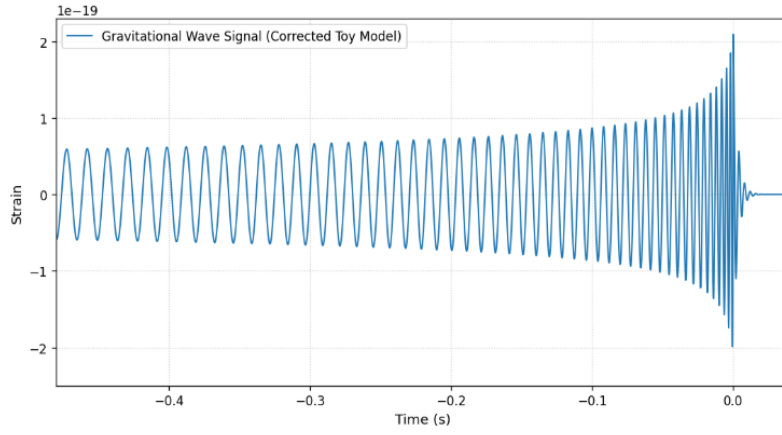


Figure 3.10: The more complex timeseries used for the edge-collapse tests of this section: this timeseries has a growing amplitude and frequency before the 0 s, while after 0 s the amplitude decays to 0.

A more demanding signal is exactly what makes a check of this kind meaningful, since any difference between the two branches would be far easier to miss on a clean, simple signal like the cosine. As before, this timeseries is turned into a point cloud (via a Takens embedding, this time also reduced to 3 dimensions with PCA before computing homology).

Our first objective is to check, empirically, whether `collapse_edges` really leaves the persistence diagram unchanged, exactly as the theorem above predicts. We have already seen, in Section 3.3, all the steps needed to get from a point cloud to a persistence diagram; here we only need to toggle one parameter:

- **Without edge collapse.** It is enough to call `VietorisRipsPersistence` with `collapse_edges=False`:

```
1 persistence = VietorisRipsPersistence(
2     homology_dimensions=HOMOLOGY_DIMENSIONS, # [0, 1, 2]: which Betti
3     collapse_edges=False,
4     n_jobs=-1,
5 )
```

- **With edge collapse.** Symmetrically, setting `collapse_edges=True` instead computes the persistence diagram of the very same point cloud *with* edge collapse applied.

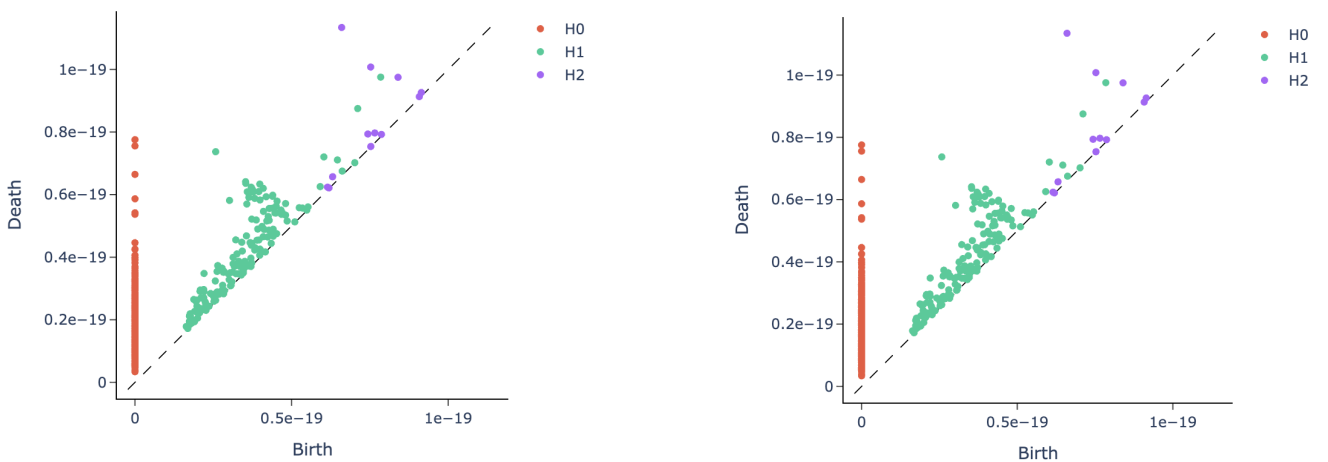


Figure 3.11: The persistence diagram of the same point cloud, computed without edge collapse (left) and with edge collapse (right).

Figure 3.11 shows the two resulting diagrams side by side: they are exactly the same. Turning `collapse_edges` on changes nothing about which topological features are found, or when they are born and die — exactly as the theorem of Boissonnat and Pritam [41] predicts.

3.6.1 Timing test

Having checked that edge collapse leaves the persistence diagram itself untouched, our next objective is to see, in practice, how much computation time it actually saves.

Part 1: all three homology dimensions. Here is, step by step, how this first timing test is set up:

- **We first build several shorter versions of the same timeseries.** The idea is to have signals of different lengths to time the computation on, so that we can later see how the computation time changes as the signal gets longer. We do this simply by cropping the timeseries down to several different lengths, from a short crop up to the full signal.
- **Each of these 4 cropped signals is then turned into its own point cloud,** using the same Takens embedding procedure already described in Section 3.2 — so at the end of this step we have 4 point clouds of different sizes, one per cropped signal, ready to be fed into the persistent homology computation.
- **We then time how long VietorisRipsPersistence takes on each of these 4 point clouds, twice:** once with `collapse_edges` turned on, and once with it turned off, so that for every point cloud we end up with two measured computation times to compare against each other.
- **In both cases, we ask for all three homology dimensions, H_0 , H_1 and H_2** — this is the setting we are specifically interested in for this first part of the test, since H_2 is the expensive dimension that edge collapse is meant to help with.

Figure 3.12 plots the resulting computation time against what percentage of the signal’s samples were kept in each cropped version. Reading the two panels side by side, collapsing edges gives a clear speedup, and that speedup grows the longer the (cropped) signal gets.

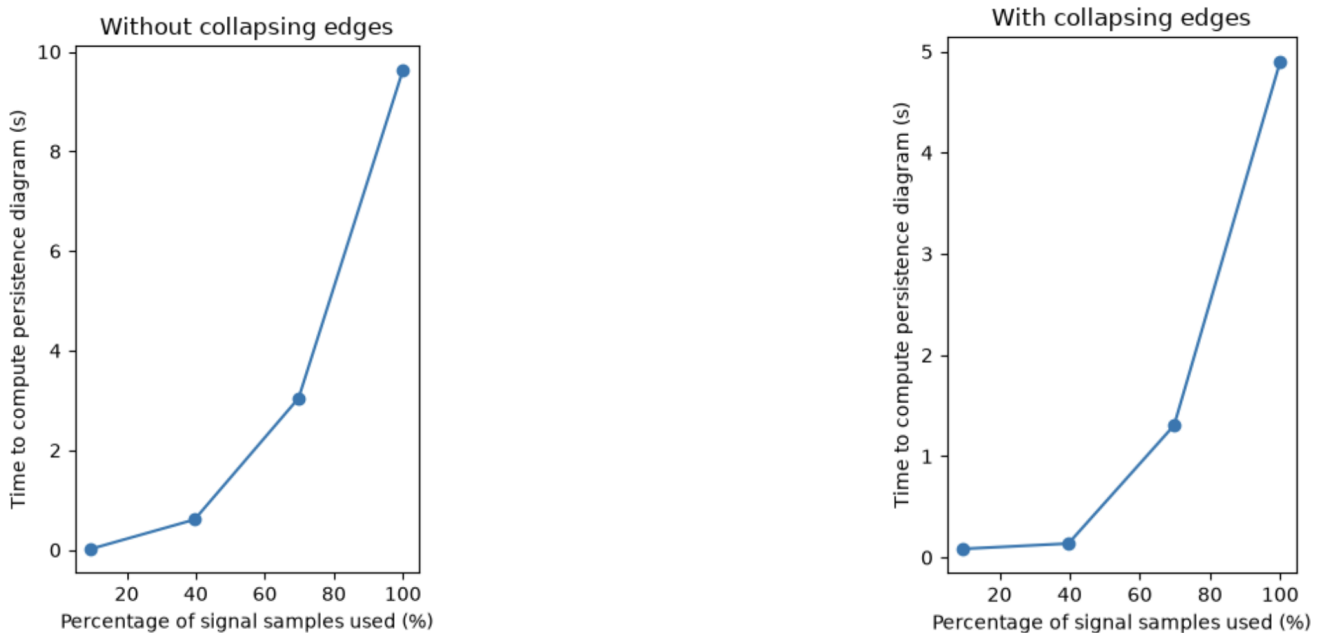


Figure 3.12: Time to compute the persistence diagram, with and without edge collapse, asking for all three homology dimensions H_0, H_1, H_2 : without edge collapse (left) and with edge collapse (right). On the x -axis, “percentage of signal samples used” is simply how much of the full signal each cropped version kept: 100% is the uncropped signal, while a smaller percentage means a shorter crop, keeping only that fraction of the samples. On the full, uncropped signal, edge collapse very nearly halves the computation time (from about 9.6s to about 4.9s), a speed-up of roughly a factor of 2.

Part 2: only the two lower dimensions. We now repeat the exact same timing test, with one single change: this time we ask only for H_0 and H_1 , leaving H_2 out entirely. Doing so flips the result: `collapse_edges` now makes the computation *slower*, not faster. To see why, it helps to break the reasoning into two steps:

- **Pruning away redundant connections is itself an extra step, and this step costs some time.** Before the rest of the computation even begins, edge collapse first has to go through the point cloud’s connections and decide which ones are redundant and can be removed — this bookkeeping does not happen for free, it takes some amount of time on its own, regardless of what is computed afterwards.
- **This upfront cost is only worth paying if it saves us time later, on the truly expensive part of the computation — building and processing simplices in high dimensions.** If that expensive part never actually happens, then the time spent pruning connections was simply wasted: we paid the cost of collapsing edges without ever getting the benefit it is meant to provide.

Asking only for H_0 and H_1 never reaches that expensive part in the first place, so there is nothing left for edge collapse to save — only its own overhead remains, which is why it ends up making things slower rather than faster. Figure 3.13 plots the resulting computation time, exactly as in Part 1 but for H_0, H_1 only.

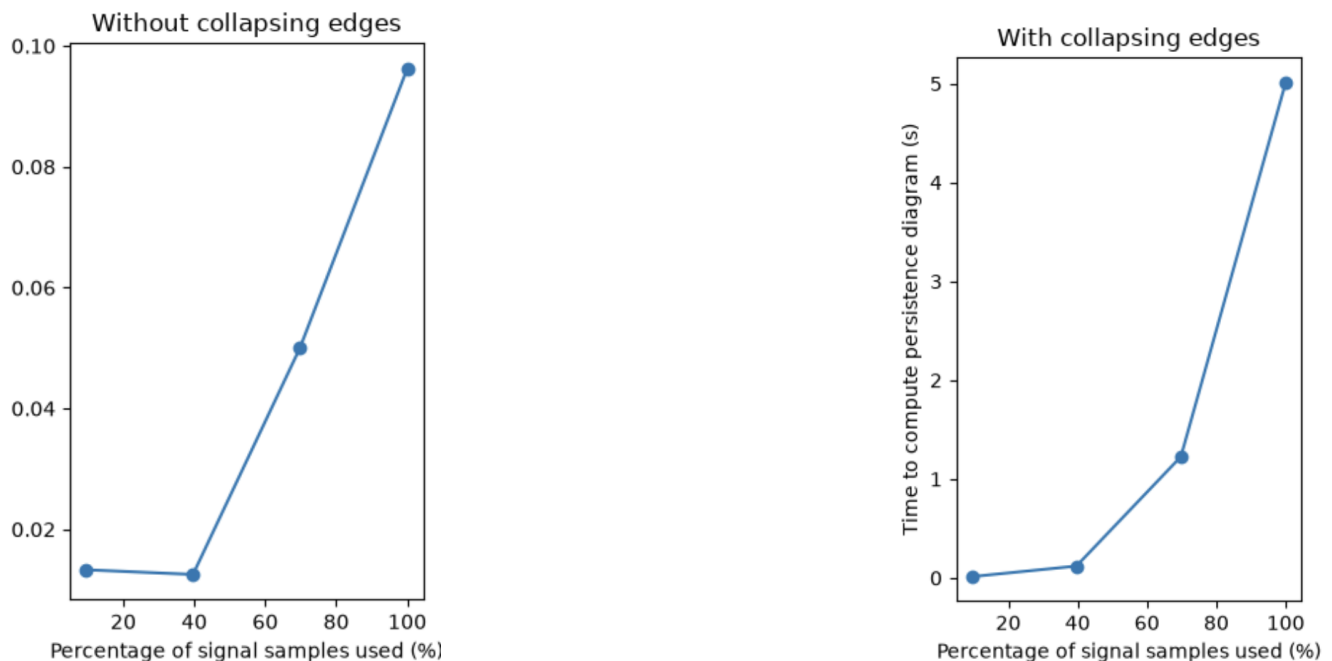


Figure 3.13: Time to compute the persistence diagram, with and without edge collapse, asking only for H_0 and H_1 : without edge collapse (left) and with edge collapse (right). Note the very different vertical scales: on the full, uncropped signal, computing without edge collapse takes well under a tenth of a second, while collapsing edges first now takes about 5s — roughly 50 times slower, confirming that here edge collapse only adds overhead without ever reaching the expensive computation it is meant to save time on.

Motivated by this, we tried to get a sense of how expensive computing H_2 would become for a realistic, several-seconds-long signal, well beyond the short, cropped versions used above. Fitting a simple exponential curve to the measured computation times against the number of samples, and extrapolating it out to a few seconds’ worth of samples, gives an estimated time of about 83 billion seconds — roughly 2 600 years — to compute H_2 for a signal only 3 seconds long, as shown in Figure 3.14 below.

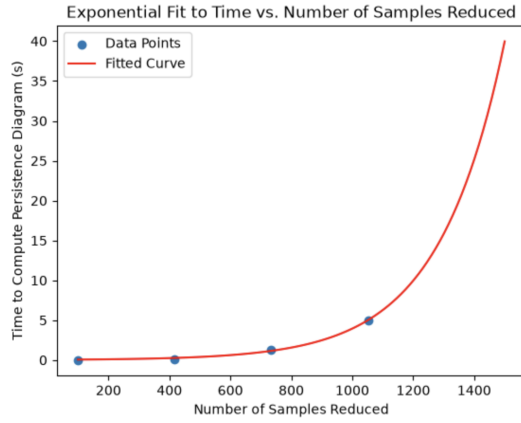


Figure 3.14: The exponential curve fitted to the measured computation times, used to extrapolate how the computation time grows for longer signals than the ones actually measured.

We stress that this extrapolation is not meant to be rigorous or precise: fitting a curve to just a handful of data points and pushing it far beyond the range it was fitted on is, at best, a rough order-of-magnitude guess. Its purpose here is only to give an idea of how quickly the computation becomes infeasible, not to predict an exact runtime.

Even as a rough guess, though, this estimate makes one thing clear: edge collapse alone is not enough to make computing H_2 feasible on a realistic signal. The practical fix this suggests is to increase the **stride** of the Takens embedding (Section 2.3.1). Recall that the stride controls how densely the embedding samples the timeseries: a higher stride means downsampling it more, i.e. skipping more samples between one embedded point and the next. A larger stride therefore directly produces a point cloud with fewer points, and since the cost of building and processing a Rips complex grows so fast with the number of points, even a modest reduction in the number of points can turn an infeasible computation into a feasible one. Increasing the stride matters especially once this pipeline is no longer run just once, but repeated on many signals in turn: later in this work, when we apply it to gravitational-wave data, the same homology computation has to be run again and again, once per signal, so keeping each individual run fast, by keeping the point cloud small, becomes essential rather than optional.

Chapter 4

Gravitational Waves: Theory and Detection

4.1 Introduction and outline

Gravitational waves (GWs) are ripples in the curvature of space-time, predicted by Einstein's theory of general relativity (GR) and generated by the accelerated motion of mass distributions. A century after they were first inferred, they were directly detected by the LIGO and Virgo interferometers, opening an entirely new observational window on the Universe. The present chapter has two purposes. The first is to assemble, in a self-contained way, the theoretical background needed to understand gravitational radiation and its detection: starting from GR we linearise Einstein's equations, identify the two propagating polarisations of the wave, and follow the radiation from its quadrupolar generation to its interaction with an interferometric detector. The second purpose, which occupies the latter half of the chapter, is to address the more practical question that motivates the rest of this thesis: *given the output of a real detector, how do we decide whether a gravitational wave is present, and how do we reconstruct the properties of its source?*

This second, data-analysis, theme is the central concern of the work. Rather than only computing waveforms, we shall be concerned with how the detector responds, how its noise is characterised, and what can be inferred from a noisy time series. The treatment of the theoretical foundations follows the standard textbook [42], while the data-analysis part follows closely [12], complemented by the data-analysis chapters taken from the "GW Open Data Workshops", organised by the LIGO-Virgo-KAGRA Scientific Collaboration.

The reason for developing this material in detail is that the rest of the thesis proposes and studies a *new* detection strategy, based on *topological data analysis* (TDA) and, in particular, on the persistent homology used in combination with machine learning techniques. The mathematical machinery of persistent homology was developed in the previous chapters; here we are interested in the physical and statistical context into which such a method must fit. To understand why a topological approach to GW is interesting one must first understand, in some depth, the technique it intends to complement — *matched filtering* — together with its remarkable strengths and its very concrete limitations.

Notation

Before presenting an introduction to GW physics and GW data analysis, let us fix here some conventions in the notation that we will use in the following.

Constants and units

The speed of light c and the Newton constant G are always written explicitly; we never set either of them to 1. The solar mass is denoted by $M_{\odot}$.

Indices and metric signature

Greek indices, such as $\alpha, \beta, \dots$ or $\mu, \nu, \dots$, take the values $0, \dots, 3$ and label spacetime components, while spatial components alone are labelled by Latin indices, $i, j, \dots = 1, 2, 3$; repeated upper and lower indices are summed over (the Einstein summation convention). The flat-space (Minkowski) metric is

$$\eta_{\mu\nu} = (-, +, +, +),$$

the sign convention most common in general relativity (the opposite one is more common in particle physics). Spacetime coordinates are written $x^\mu = (x^0, \mathbf{x})$ with $x^0 = ct$, and $\partial_\mu = \partial/\partial x^\mu$.

Fourier transform

A tilde denotes the Fourier transform of a function of time, with the convention

$$\tilde{x}(f) = \int_{-\infty}^{\infty} dt x(t) e^{2\pi i f t}, \quad x(t) = \int_{-\infty}^{\infty} df \tilde{x}(f) e^{-2\pi i f t}, \quad (4.1)$$

i.e. frequency f is used throughout rather than angular frequency.

4.2 General relativity, gravitational waves and compact binaries

In this part we review the background needed for the rest of the thesis: the theory of general relativity and, in its linearized limit, the description of gravitational waves.

4.2.1 General relativity

General relativity was developed by Albert Einstein and published in 1915 [1]. It gives a geometric description of gravity. What we usually think of as the gravitational force is, in this theory, just the effect of the geometry of spacetime on matter and energy; in turn, that geometry is fixed by the matter and energy that are present in spacetime. This picture is precisely what makes Einstein's equivalence principle come out naturally. The principle states that, around any point of spacetime, we can always choose a coordinate system in which the laws of physics look like those of special relativity, as long as we stay in a small enough region.

To put these ideas in a precise form, general relativity is written in the language of differential geometry: spacetime is a Lorentzian manifold, and physical quantities are tensor fields on it, so that the equations keep their form under any diffeomorphism (a smooth, invertible coordinate change). In particular it is always possible to construct local inertial coordinates in which the metric is Minkowskian, up to tidal effects, and all such frames describe the same physics — general relativity therefore has a very large local gauge symmetry, invariance under the group of arbitrary diffeomorphisms [12].

General relativity can be derived from a least-action principle. One starts by taking the gravitational action to be the Einstein–Hilbert action [42]

$$S_{EH} = \frac{c^3}{16\pi G} \int d^4x \sqrt{-g} R, \quad (4.2)$$

where $g \equiv \det(g_{\mu\nu}) < 0$ is the determinant of the metric tensor. To couple gravity to matter, one adds a matter action S_M . This is usually built from a flat-space Lagrangian using the minimal coupling rule: the flat metric is replaced by a general one ($\eta_{\mu\nu} \rightarrow g_{\mu\nu}(x)$), ordinary derivatives become covariant derivatives ($\partial_\mu \rightarrow \nabla_\mu$), and the spacetime volume element is taken to be $dV = d^4x \sqrt{-g}$:

$$S_M = \frac{1}{c} \int d^4x \sqrt{-g} \mathcal{L}. \quad (4.3)$$

This action also gives the definition of the energy–momentum tensor $T_{\mu\nu}$ [12]:

$$T_{\mu\nu} \equiv \frac{2c}{\sqrt{-g}} \frac{\delta S_M}{\delta g^{\mu\nu}}. \quad (4.4)$$

Varying the full action $S = S_{EH} + S_M$ with respect to the inverse metric $g^{\mu\nu}$ gives Einstein's equations [42]:

$$R_{\mu\nu} - \frac{1}{2}g_{\mu\nu}R = \frac{8\pi G}{c^4}T_{\mu\nu}, \quad (4.5)$$

where while $R_{\mu\nu}$ is the Ricci tensor, R the Ricci scalar and $G = 6.674\,30(15)\,\text{m}^3\,\text{kg}^{-1}\,\text{s}^{-2}$ is Newton's constant [43]. The left-hand side is also called the Einstein tensor, namely $G_{\mu\nu} = R_{\mu\nu} - \frac{1}{2}g_{\mu\nu}R$, and this G_0 satisfies the contracted Bianchi identities $\nabla^\mu G_{\mu\nu} = 0$ [42, 44].

Einstein's equations are ten in number, because the tensors in them are symmetric in the indices μ and ν . The four Bianchi identities, however, act as constraints on the initial data, so only six of the equations are truly dynamical (second order in time). This is why, to evolve the equations without ambiguity, one must add four conditions that fix the coordinate system [45]. As we will see in section 4.2.3, this leaves general relativity with only two radiative degrees of freedom.

In general these equations are hard to solve exactly. This is especially true when gravity is coupled to matter, or when the system has little or no symmetry, which is the case in most realistic situations. For this reason, only a few exact solutions are known. Apart from the Minkowski metric, the best known stationary vacuum solutions that are asymptotically flat are:

- the *Schwarzschild metric*, for a spherically symmetric body that has only mass;
- the *Kerr metric*, for a rotating, axially symmetric body;
- the *Reissner–Nordström metric*, for a spherically symmetric charged body, obtained once electromagnetism is added as a matter field in (4.3);
- the *Kerr–Newman metric*, for a rotating, axially symmetric charged body [42].

Historical overview

Einstein predicted the existence of gravitational waves in 1916, one year after formulating general relativity [2, 6], but for decades it remained unclear whether they were physical or merely an artifact of the theory's large gauge freedom. The question was settled only at the 1957 Chapel Hill conference, where Pirani and Feynman showed that gravitational waves have measurable, gauge-independent effects, most famously through Feynman's “sticky bead” argument [3, 46, 47]. The first evidence for their existence was indirect: the binary pulsar PSR B1913+16, discovered by Hulse and Taylor in 1974, loses orbital energy at exactly the rate general relativity predicts for a system radiating gravitational waves [4, 5, 12]. A direct detection came only decades later, in 2015, when the two LIGO interferometers observed the merger of a binary black hole [6]. A joint detection with electromagnetic telescopes of a binary neutron star merger followed in 2017, opening the era of multi-messenger astronomy [48, 49]. Looking ahead, ground-based detectors such as the Einstein Telescope and Cosmic Explorer [9, 50, 51], and the space-based LISA mission [11], are expected to extend this sensitivity considerably.

4.2.2 Gravitational waves in general relativity

Phenomenology

In this section we outline some phenomenology of gravitational waves. In this work we are mainly interested in binary systems of two compact objects that orbit each other under their mutual gravity. Such systems are one of the main ways in which gravitational waves are produced. In fact, all the waves detected directly so far come from this kind of source.

In these systems, as discussed in section 4.2.5, the motion of the two bodies produces gravitational waves. The waves carry energy away from the system, so the two bodies move closer and closer. This is the *inspiral* phase. The wave amplitude is larger for more massive systems, and both the amplitude and the frequency grow as the bodies get closer. This rising signal is called a *chirp*. If the bodies are compact enough, which requires that they be heavy enough, they eventually get so close that strong relativistic effects take over. The orbit then breaks down into a *plunge*, followed by the *merger*. The result of the merger is a single compact body. It starts in an excited state, because of the violent process that created it, and it loses energy by emitting gravitational waves during the *ringdown*, until it settles into its final

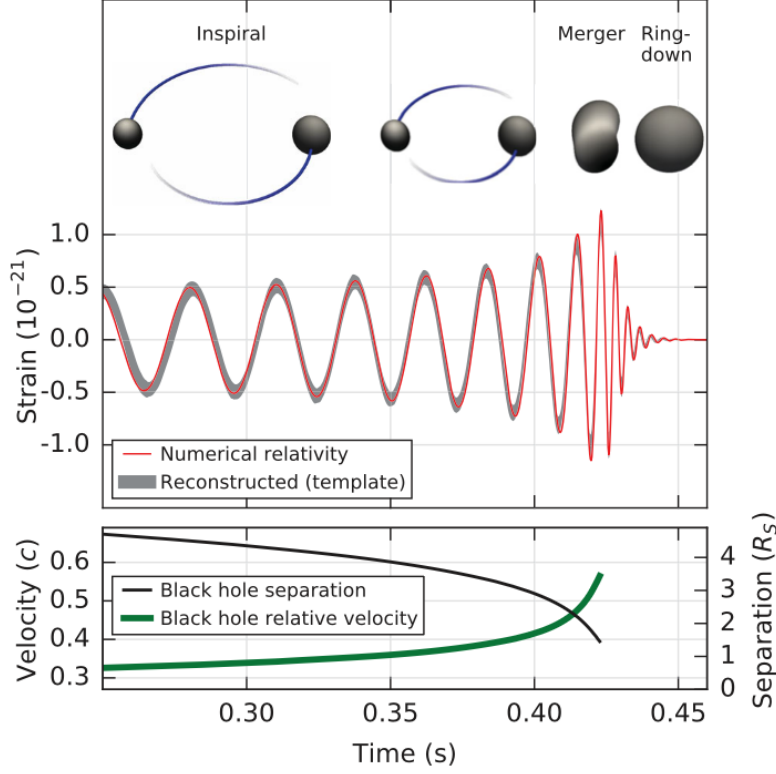


Figure 4.1: Time evolution of the binary black hole found in event GW150914, the first direct detection [6, 7]. The top panel shows the strain that the wave induced at the LIGO Hanford observatory. It is reconstructed both from a dedicated numerical-relativity simulation (red line) and from template waveforms (grey band), with the most likely parameters. The phases the black holes went through are sketched above: the inspiral that brought them together, then the merger and the ringdown that produced the final black hole. The strain peaks around the moment of merger. The bottom panel shows the effective Keplerian separation R/R_s , in units of the Schwarzschild radius R_s defined in Eq. (4.6), and the effective relative velocity v/c . Image has been collected from [6].

state.

These systems are usually very far away, so the wave amplitude that reaches us is very small. For this reason we can hope to detect the signal only for very massive systems, and only when the two bodies are very close, that is, near the end of the long inspiral and during the merger and ringdown. The inspiral phase is in fact extremely long, and at the start it evolves very slowly. As an example, a binary with a total mass of a few $M_\odot$, and an age similar to that of its host galaxy, could have merged by now only if its initial orbital period was shorter than about a day [12]. Figure 4.1 shows the final phases of the first binary black hole ever observed to merge.

The length scale at which strong relativistic effects become important, which is roughly where the inspiral turns into the plunge, is set by the Schwarzschild radius of the mass m ,

$$R_s \equiv \frac{2Gm}{c^2} \approx 2.95 \text{ km} \left(\frac{m}{M_\odot} \right), \quad (4.6)$$

where $M_\odot = 1.98841(4) \cdot 10^{30} \text{ kg}$ is the solar mass [52].

From this discussion we see that the binaries we can hope to detect are made of heavy and compact objects. By “compact” we mean objects whose size is close to their Schwarzschild radius. This compactness lets them get very close to each other before they merge or are torn apart by tides. The astrophysical objects that satisfy these conditions are neutron stars (NS) and black holes (BH). The binaries they form are called binary neutron stars (BNS), binary black holes (BBH), and neutron-star–black-hole systems (NSBH).

Compact binaries are not the only way to produce gravitational waves. A rotating neutron star can radiate if its shape is not perfectly symmetric, for example because of small “mountains” or density

bumps on its surface. These neutron stars are expected to emit continuous gravitational waves [53]. A supernova can also release a large amount of gravitational waves if the explosion is asymmetric enough. In addition, we expect the Universe to be filled with a stochastic gravitational-wave background, made by the superposition of waves from many different sources. Of special interest is the cosmological background produced in the early Universe, for example by inflation, but possibly also by phase transitions or more exotic phenomena such as cosmic strings [54, 55]. On this point, the NANOGrav collaboration has recently found evidence, using pulsar-timing arrays, for a stochastic background at nanohertz frequencies. This signal is probably the superposition of waves from many supermassive black hole binaries, although a cosmological or more exotic origin cannot be ruled out [56].

4.2.3 General relativity at linear order and gravitational waves

In the following we linearize Einstein's equations (4.5) around the flat Minkowski metric. This gives a set of linear partial differential equations, which are well understood, and it works under the single assumption that the gravitational field is weak. This section is based mostly on [12].

We start from the full Einstein equations (4.5) and look for solutions that are close to flat spacetime. To do this, we write the metric as the flat Minkowski metric $\eta_{\mu\nu}$ plus a small perturbation $h_{\mu\nu}$,

$$g_{\mu\nu} = \eta_{\mu\nu} + h_{\mu\nu}, \quad |h_{\mu\nu}| \ll 1, \quad (4.7)$$

and we keep only terms that are linear in $h_{\mu\nu}$. The theory obtained in this way is called *linearized theory*. The components of a tensor depend on the reference frame we choose, so (4.7) cannot literally hold in every frame at once. What we actually require is weaker, namely that *some* frame exists in which it holds, over a region of space large enough for our purposes [12]. In linearized theory indices are raised and lowered with the flat metric $\eta_{\mu\nu}$.

Choosing such a frame does not fix the coordinates completely. A residual freedom remains, in the form of a small change of coordinates $x^\mu \rightarrow x^\mu + \xi^\mu(x)$, where the derivatives $|\partial_\mu \xi_\nu|$ are at most as small as $|h_{\mu\nu}|$. Under this change the perturbation transforms as [12]

$$h_{\mu\nu}(x) \rightarrow h_{\mu\nu}(x) - (\partial_\mu \xi_\nu + \partial_\nu \xi_\mu). \quad (4.8)$$

This is the gauge freedom of linearized gravity, and it plays the same role as the gauge freedom of electromagnetism. It is convenient to introduce the *trace-reversed* perturbation

$$\bar{h}_{\mu\nu} \equiv h_{\mu\nu} - \frac{1}{2}\eta_{\mu\nu}h, \quad h \equiv \eta^{\mu\nu}h_{\mu\nu}, \quad (4.9)$$

because in terms of $\bar{h}_{\mu\nu}$ the linearized Einstein equations take a simpler form. We can use the gauge freedom (4.8) to impose the *Lorenz gauge* (also called the harmonic or De Donder gauge) [12],

$$\partial^\nu \bar{h}_{\mu\nu} = 0. \quad (4.10)$$

In this gauge the linearized Einstein equations reduce to a simple wave equation sourced by the energy-momentum tensor [12],

$$\square \bar{h}_{\mu\nu} = -\frac{16\pi G}{c^4} T_{\mu\nu}, \quad (4.11)$$

where $\square = \eta^{\mu\nu}\partial_\mu\partial_\nu = -\frac{1}{c^2}\partial_t^2 + \nabla^2$ is the flat-space d'Alembertian. The Lorenz gauge (4.10) gives four conditions, so it reduces the ten independent components of the symmetric matrix $h_{\mu\nu}$ to six.

Equation (4.11) is the basic result for the *generation* of gravitational waves. To study instead their *propagation* and their interaction with a detector, we look at this equation outside the source, where $T_{\mu\nu} = 0$ [12]:

$$\square \bar{h}_{\mu\nu} = 0 \quad (\text{outside the source}). \quad (4.12)$$

Since $\square = -\frac{1}{c^2}\partial_t^2 + \nabla^2$, this equation tells us right away that gravitational waves travel at the speed of light.

4.2.4 The transverse-traceless gauge and the two polarizations

Outside the source we can simplify the metric even further. The Lorenz condition (4.10) does not use up all the gauge freedom: a further change of coordinates with $\square \xi^\mu = 0$ leaves the Lorenz gauge intact, and it can be used to impose four more conditions [12]. We use this remaining freedom to set the time components to zero and to make the perturbation traceless. The result is the *transverse-traceless* (TT) gauge,

$$h_{0\mu} = 0, \quad h^i_i = 0, \quad \partial^j h_{ij} = 0, \quad (4.13)$$

in which only the spatial components survive; we denote the metric in this gauge by h_{ij}^{TT} . This gauge-fixing procedure lets us isolate the genuine radiative content of the gravitational field, and a simple counting argument makes this precise. The symmetric tensor $\bar{h}_{\mu\nu}$ starts out with ten independent components. Imposing the Lorenz gauge condition (4.10) is itself four conditions, so it removes four of them, leaving six. In vacuum we have exactly the residual gauge freedom used above to impose the TT conditions (4.13) — the four functions ξ^μ with $\square \xi^\mu = 0$ — and using it to reach the TT gauge removes four more. Only two independent components survive: these are the two physical, propagating degrees of freedom of general relativity, matching the two polarizations of the wave [12].

The two degrees of freedom become explicit once we look at a plane wave. Equation (4.12) has plane-wave solutions, and for a wave travelling along a direction $\hat{n}$ the transversality condition in (4.13) forces the non-zero components of h_{ij}^{TT} to lie in the plane orthogonal to $\hat{n}$ [12]. Taking $\hat{n}$ along the z axis, and using that h_{ij}^{TT} is symmetric and traceless, the wave can be written as

$$h_{ij}^{TT}(t, z) = \begin{pmatrix} h_+ & h_\times & 0 \\ h_\times & -h_+ & 0 \\ 0 & 0 & 0 \end{pmatrix}_{ij} \cos[\omega(t - z/c)], \quad (4.14)$$

where h_+ and $h_\times$ are the amplitudes of the “plus” and “cross” polarizations [12]. The two polarizations have a clear physical meaning: the plus mode stretches one transverse axis while it squeezes the other, and the cross mode does the same along axes rotated by 45° . It is this differential, quadrupolar stretching of space that a detector is built to measure (section 4.3.2).

4.2.5 Inspiral of compact binaries and the chirp signal

We now specialize the formalism to the sources of interest in this work: binaries of two compact objects. We treat the binary, to the lowest (Newtonian) order, as two point masses m_1 and m_2 on a circular orbit, with total mass $m = m_1 + m_2$ and reduced mass $\mu = m_1 m_2 / m$. For a circular orbit of radius R , Kepler’s law relates the orbital frequency ω_s to the radius [12],

$$\omega_s^2 = \frac{Gm}{R^3}. \quad (4.15)$$

A binary is one of the most efficient sources of gravitational waves, because its mass distribution has a large, rapidly changing quadrupole moment. The gravitational wave it emits is quadrupolar, so it comes out at twice the orbital frequency, $\omega_{gw} = 2\omega_s$ [12].

Applying the quadrupole formula to this system, and eliminating the radius R in favour of the frequency through (4.15), the two polarization amplitudes take the compact form [12]

$$h_+(t) = \frac{4}{r} \left(\frac{G\mathcal{M}_c}{c^2} \right)^{5/3} \left(\frac{\pi f_{gw}}{c} \right)^{2/3} \frac{1 + \cos^2 \theta}{2} \cos(2\pi f_{gw} t_{ret} + 2\phi), \quad (4.16)$$

$$h_\times(t) = \frac{4}{r} \left(\frac{G\mathcal{M}_c}{c^2} \right)^{5/3} \left(\frac{\pi f_{gw}}{c} \right)^{2/3} \cos \theta \sin(2\pi f_{gw} t_{ret} + 2\phi), \quad (4.17)$$

where r is the distance to the source, θ the angle between the line of sight and the normal to the orbit, ϕ a reference phase, and $t_{ret} = t - r/c$ the retarded time. The masses enter these expressions only through one combination, the *chirp mass* [12]

$$\mathcal{M}_c = \frac{(m_1 m_2)^{3/5}}{(m_1 + m_2)^{1/5}}. \quad (4.18)$$

The chirp mass is therefore the parameter that controls the amplitude of the signal, and, as we will see in a moment, also the rate at which its frequency grows. For this reason it is the best-measured parameter of an inspiral.

The emission of gravitational waves carries energy away from the binary. As a consequence the orbit shrinks, the orbital frequency increases, and so does the frequency of the wave. This slow drift is what produces the characteristic rising signal called the *chirp*. As long as the orbit evolves slowly compared with the orbital period (the so-called quasi-circular regime), one finds that the wave frequency obeys the simple evolution law [12]

$$\dot{f}_{gw} = \frac{96}{5} \pi^{8/3} \left(\frac{G\mathcal{M}_c}{c^3} \right)^{5/3} f_{gw}^{11/3}. \quad (4.19)$$

Integrating (4.19), the frequency formally diverges at a finite time, which we call t_{coal} , the moment of coalescence. Writing the time to coalescence as $\tau \equiv t_{coal} - t$, the solution of (4.19) is [12]

$$f_{gw}(\tau) = \frac{1}{\pi} \left(\frac{5}{256} \frac{1}{\tau} \right)^{3/8} \left(\frac{G\mathcal{M}_c}{c^3} \right)^{-5/8}. \quad (4.20)$$

Putting in typical numbers for a binary neutron star, this can be written in the convenient form [12]

$$f_{gw}(\tau) \approx 134 \text{ Hz} \left(\frac{1.21 M_\odot}{\mathcal{M}_c} \right)^{5/8} \left(\frac{1 \text{ s}}{\tau} \right)^{3/8}, \quad (4.21)$$

which shows that, for a chirp mass $\mathcal{M}_c \approx 1.21 M_\odot$ (the value for two $1.4 M_\odot$ neutron stars), the signal sweeps up through the detector band in the last second or so before merger. The reason is the steep dependence $f_{gw} \propto \tau^{-3/8}$: the frequency rises ever faster as the time to coalescence τ shrinks, so even though the inspiral lasts a very long time, almost all of the frequency sweep is squeezed into the final moments before merger. It is this short but rapidly evolving part of the chirp that the detection methods discussed in the rest of the chapter aim to capture.

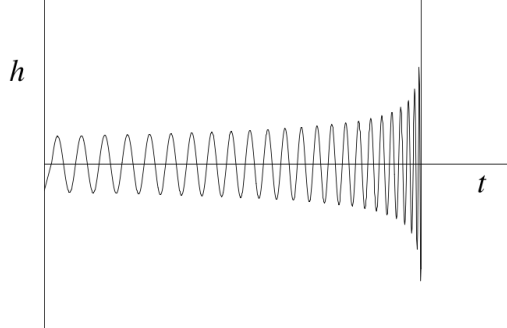


Figure 4.2: **Time evolution of the gravitational-wave strain emitted during the inspiral of a compact binary.** Both the amplitude and the frequency increase as the two bodies spiral closer, producing the characteristic “chirp”. This rising waveform is the signature that the detection methods discussed in the rest of the chapter — matched filtering and, later, the topological method developed in this thesis — are designed to recognize. Image has been collected from [12].

The expressions above hold at the lowest, Newtonian order, and they describe only the inspiral phase. As the bodies approach the strong-field regime, one must go beyond this approximation; the post-Newtonian expansion. For the purposes of this thesis the important point is qualitative: the inspiral of a compact binary produces a well-defined, rising chirp, and it is this signal — buried in the detector noise — that we will try to detect.

4.3 The detector as a linear system and the noise spectral density

4.3.1 The Michelson interferometer

The idea of using an interferometer to detect gravitational waves is old, and in principle remarkably simple. It goes back to 1962, when it was first proposed by the Russian theorists M. Gertsenshtein and

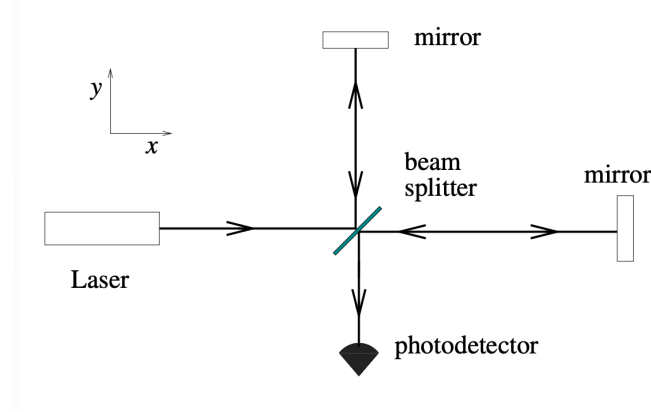


Figure 4.3: Layout of a simple Michelson interferometer. Image has been collected from [12].

V. I. Pustovoit, before being developed further in the late 1960s. Turning this simple idea into a working instrument, however, has taken decades of effort and large international collaborations, since a real interferometer has an enormous number of degrees of freedom that must all be controlled to extraordinary precision. We start from the simplest possible setting: a basic Michelson interferometer [12].

A Michelson interferometer is the same kind of instrument that Michelson and Morley famously used in 1887 to look for the ether, and it is an extremely sensitive way of comparing how long light takes to travel along two different paths. Figure 4.3 sketches the idea:

- a laser beam hits a beam-splitter, which sends half the light down one arm and half down a second, perpendicular arm;
- a mirror at the end of each arm reflects the light straight back;
- the two beams meet again at the beam-splitter and recombine into a single beam;
- part of that recombined light reaches a photodetector, which simply measures how bright it is.

To see why this setup is sensitive to the length of the arms, it helps to think of the laser light as a wave with a well-defined phase, oscillating at the laser frequency ω_L , with wavenumber $k_L = \omega_L/c$. Because the two arms generally have slightly different lengths L_x and L_y , the beam sent down one arm travels a different distance, out to its mirror and back, than the beam sent down the other. A light wave's phase keeps track of exactly how far it has travelled, so two beams that have covered different distances come back to the beam-splitter slightly out of phase with one another. It is this mismatch that makes them interfere when they recombine: rather than simply adding up, two waves that are out of phase can reinforce each other, partially cancel, or land anywhere in between, depending on how far out of phase they are. Working this out explicitly, one finds that the power reaching the photodetector obeys a strikingly simple law [12],

$$|E_{\text{out}}|^2 = E_0^2 \sin^2[k_L(L_y - L_x)], \quad (4.22)$$

where E_0 is the amplitude of the input laser field: the measured power depends only on the *difference* $L_y - L_x$ between the two arm lengths, oscillating between fully dark and fully bright as that difference changes by a fraction of the wavelength of the laser light. Therefore any variation in the length of an arm results in a corresponding variation of the power at the photodetector.

This is precisely the effect that a gravitational wave can produce. As we saw in Section 4.2.4, a wave stretches one transverse direction while squeezing the other — exactly the differential, quadrupolar deformation of space that we identified earlier as the signature of a gravitational wave. If the two arms of the interferometer are oriented along these two directions, a passing wave changes L_x and L_y , so that even a tiny strain h produces a genuine, non-cancelling difference $L_y - L_x$, and hence a change in the power measured at the photodetector. This is the basic principle that lets a Michelson interferometer, in spite of its conceptual simplicity, measure length changes many orders of magnitude smaller than the size of a proton.

4.3.2 Input, output and the detector tensor

The output of any gravitational-wave detector is a time series: it describes, for instance, the oscillation state of a resonant bar or, in the case of interest to us, the phase difference of the light recombined after travelling along the two arms of an interferometer. Moreover, this output is simply a long stream of numbers, and the whole art of detection consists in deciding whether a faint, deterministic signal is hidden in it. To set up the problem cleanly it is convenient, following [12], to think of the detector as a *linear system*.

The physical input is a gravitational wave, described by a tensor field $h_{ij}(t)$, whereas the input and output of the instrument are scalar quantities. The two are related by a constant tensor D^{ij} , the *detector tensor*, that encodes the geometry of the apparatus,

$$h(t) = D^{ij} h_{ij}(t). \quad (4.23)$$

For a resonant bar oriented along the x axis only the (x, x) component of h_{ij} drives the detector, so $D^{ij} = 1$ for $i = j = 1$ and zero otherwise [12]; for an interferometer with arms along $\hat{x}$ and $\hat{y}$ one finds $D^{ij} = \frac{1}{2}(\hat{x}^i \hat{x}^j - \hat{y}^i \hat{y}^j)$ [12]. The key point is that, apart from this geometrical factor of order unity, the scalar $h(t)$ depends only on the incoming wave and not on the internal workings of the detector.

For a linear system the output in the absence of noise is related to the input, in frequency space, by a *transfer function* $T(f)$,

$$\tilde{h}_{\text{out}}(f) = T(f) \tilde{h}(f). \quad (4.24)$$

A real detector is in practice a cascade of stages $i = 1, \dots, N$, each with its own transfer function $T_i(f)$, so that $T(f) = \prod_i T_i(f)$, and noise can be injected at any stage. Each noise source propagates to the output with the transfer function of the stages downstream of the point at which it first appeared, and all such contributions add up to a total output noise $n_{\text{out}}(t)$. The output that we actually record is therefore

$$s_{\text{out}}(t) = h_{\text{out}}(t) + n_{\text{out}}(t). \quad (4.25)$$

It is enormously convenient to refer everything to the detector input. We define a fictitious input noise $n(t)$ through

$$\tilde{n}(f) = T^{-1}(f) \tilde{n}_{\text{out}}(f), \quad (4.26)$$

in words: $n(t)$ is the fictitious noise that, fed alone into the detector's input,¹ would propagate through the transfer function to reproduce the actual noise $n_{\text{out}}(t)$ we observe at the output. The advantage is immediate: $n(t)$ can be compared directly with the signal $h(t)$, because both are now referred to the same point, whereas $h_{\text{out}}(t)$ depends on a transfer function that can differ by many orders of magnitude between different instruments. With this convention we simply write

$$s(t) = h(t) + n(t), \quad (4.27)$$

and regard $s(t)$ as the detector output, composed of a signal $h(t)$ and a noise $n(t)$. The whole detection problem is the problem of distinguishing the GW signal $h(t)$ from the noise $n(t)$.

4.3.3 The noise spectral density

We now characterise the noise statistically. We assume that $n(t)$ is a *stationary* stochastic process with zero mean,

$$\langle n(t) \rangle = 0, \quad (4.28)$$

where $\langle \cdot \rangle$ denotes an ensemble average: this ensemble average is, in principle, an average over many independent “realizations” of the system. In practice we have a single physical system (our detector) but we can record it over time, so the ensemble average is traded for a time average². Concretely, one measures the noise $n(t)$ over a stretch of duration T and treats it as one realization, from which the Fourier transform $\tilde{n}(f)$ is obtained with a frequency resolution $\Delta f = 1/T$. The same is then done on a later stretch, also of length T and far enough from the first that the noise in the two can be regarded as

¹That is, assuming nothing else contributes anywhere else in the instrument.

²This step assumes the system to be ergodic [12].

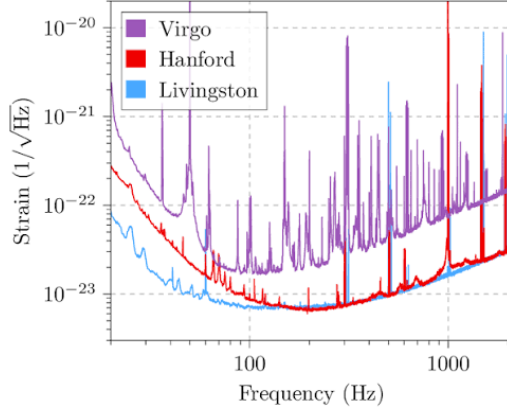


Figure 4.4: Amplitude spectral density (strain sensitivity $S_n^{1/2}(f)$, in $\text{Hz}^{-1/2}$) of the Virgo, LIGO Hanford and LIGO Livingston interferometers, on a log-log scale. The steep rise at low frequency is the seismic wall, the broad minimum is the mid-band thermal-noise floor — the band in which the instrument is most sensitive — and the rise at high frequency is due to quantum (shot) noise; outside this band the detector is effectively blind. The many narrow spikes are instrumental and environmental lines (e.g. mechanical resonances, mains harmonics), not part of the smooth noise floor discussed in the text.

uncorrelated, giving a second independent realization; averaging $\tilde{n}(f)$ over many such stretches yields the ensemble average. Note that a timescale T is built into this procedure, and it will reappear in the expressions below [12]. Stationarity means that the different Fourier components are uncorrelated, so the two-point function in frequency space has the form

$$\langle \tilde{n}^*(f) \tilde{n}(f') \rangle = \frac{1}{2} \delta(f - f') S_n(f). \quad (4.29)$$

Equation (4.29) defines the (single-sided) *noise spectral density* $S_n(f)$ ³. Since $n(t)$ is real, $\tilde{n}(-f) = \tilde{n}^*(f)$ and hence $S_n(-f) = S_n(f)$: you can find a proof of this relation in Appendix A.34. If $n(t)$ is taken dimensionless — as is natural for a strain — then $S_n(f)$ has dimensions of Hz^{-1} .⁴

The square root $S_n^{1/2}(f)$, with dimensions $\text{Hz}^{-1/2}$, is the quantity usually quoted as the *strain sensitivity* or *amplitude spectral density* of the detector. It is the most compact way of summarising the performance of an instrument as a function of frequency, and the following noise mechanisms (that we study in Section 4.3.4) contribute to it: seismic and Newtonian noise at low frequency, thermal noise of the mirror coatings and suspensions in the central band, and quantum (shot and radiation-pressure) noise at high frequency all contribute to it. A typical strain sensitivity curve is sketched in Figure 4.4.

The essential message of eq. (4.29), which we shall use repeatedly, is that the noise is not white: it is much more intense in some frequency regions than in others. An optimal analysis must therefore weight the data according to how noisy the detector is at each frequency, a feature that will emerge naturally from the matched-filtering derivation of Section 4.4.

4.3.4 The physical origin of the noise

It is worth recalling, if only qualitatively, which physical mechanisms build up the spectral density $S_n(f)$ of Figure 4.4, because the shape of that curve dictates which sources a detector can and cannot see.⁵ The broad features of a second-generation interferometer’s sensitivity are governed by three families of noise.

At *low frequency*, below roughly 10 Hz, the dominant disturbances are *seismic noise* — ground motion transmitted to the mirrors despite multi-stage suspension and active isolation — and *Newtonian* (or gravity-gradient) *noise*, the direct gravitational pull of fluctuating mass densities in the ground and atmosphere, which no suspension can screen. These produce the steep “seismic wall” that sets the lower

³Since $n(t)$ is real one has $\tilde{n}(-f) = \tilde{n}^*(f)$, so all the information is contained in the positive frequencies; $S_n(f)$ is therefore taken to be defined only for $f \geq 0$, which is the meaning of “single-sided”. The factor $\frac{1}{2}$ is a convention chosen so that integrating over positive frequencies reproduces the variance of the noise [12].

⁴To see that $S_n(f)$ has dimensions of Hz^{-1} , simple notions of Fourier transforms are enough: see Appendix A.35.

⁵This will be made precise by the optimal-SNR formula (4.44), derived later in this chapter.

edge of the usable band — the sharp rise on the left side of the amplitude spectral density curve of Figure 4.4. Heavier binaries emit at lower gravitational-wave frequencies so this low-frequency wall directly limits the reach for the heaviest compact binaries in particular⁶, and lowering it is a central goal of the third-generation designs mentioned like Einstein Telescope and LISA.

In the *central band*, from tens to a few hundred hertz, the limiting contribution is *thermal noise*. Because the mirrors, their reflective coatings and the fibres that suspend them all sit at a finite temperature, their atoms are in constant random motion. This microscopic jiggling makes the reflecting surface shift back and forth by tiny amounts, and since the interferometer measures exactly the position of that surface, the random motion shows up as noise.⁷

At *high frequency* the sensitivity is set by *quantum noise*. This has two faces: *shot noise*, the Poisson fluctuation in the arrival rate of photons at the output photodiode, which limits phase resolution and rises at high frequency; and *quantum radiation-pressure noise*, the random momentum kicks imparted to the mirrors by the same photons, which dominates at low frequency for high laser power. The two are conjugate, bounded below by the Heisenberg relation, and their balance — the “standard quantum limit” — can be reshaped by injecting frequency-dependent squeezed states of light, [12].

On top of these fundamental noises sits a long list of *technical* noises (laser frequency and intensity fluctuations, scattered-light recombination, electronic and magnetic couplings [12]) many of which are neither stationary nor Gaussian, and which return to haunt us in Section 4.7.3. For the purposes of the idealised derivation that follows we set them aside and treat $n(t)$ as stationary and Gaussian; it is essential to remember that this is an idealisation.

4.4 Matched filtering

We have established that the detector output has the general form $s(t) = h(t) + n(t)$, with $n(t)$ characterised by the spectral density $S_n(f)$ of eq. (4.29). The naive expectation would be that a wave can be detected only when $|h(t)|$ rises above $|n(t)|$. This would be catastrophic, because with realistic estimates of the signals bathing the Earth and with the sensitivity of current instruments we are firmly in the opposite regime,

$$|h(t)| \ll |n(t)|. \quad (4.30)$$

The fundamental question of this section is therefore how to dig a signal out of a noise that may be orders of magnitude larger. The answer, well known from radio engineering and radio astronomy, is that values of $h(t)$ far below the noise floor can be recovered *provided we know, to some accuracy, the form of the signal we are looking for*.

4.4.1 The intuitive idea

A simple version of the procedure already exhibits the essential physics. Suppose we know the waveform $h(t)$ we are hunting for. We multiply the output $s(t)$ by $h(t)$, integrate over an observation time T , and divide by T ,

$$\frac{1}{T} \int_0^T dt s(t) h(t) = \frac{1}{T} \int_0^T dt h^2(t) + \frac{1}{T} \int_0^T dt n(t) h(t). \quad (4.31)$$

The two terms behave very differently as T grows. The integrand of the first is positive definite (think of $\cos^2 \omega t$ times a slowly varying envelope) so the integral grows linearly in T and its average is of order the squared signal amplitude [12],

$$\frac{1}{T} \int_0^T dt h^2(t) \sim h_0^2. \quad (4.32)$$

⁶This is made precise by the inspiral SNR integral (4.65), derived later in this chapter.

⁷This connection between the random thermal motion of a body and its mechanical losses is the content of the *fluctuation-dissipation theorem*: at a given temperature, a component that dissipates more energy also fluctuates more. It is the reason low-loss materials are so important here, and the reason a lossy coating produces so much noise. [12]

The second term, by contrast, is a product of two uncorrelated oscillating functions $n(t)h(t)$: the integral of this product $n(t)h(t)$ will grow as $T^{1/2}$ for large T (as typical for random-walk systems [12]), so

$$\frac{1}{T} \int_0^T dt n(t) h(t) \sim \left(\frac{\tau_0}{T} \right)^{1/2} n_0 h_0, \quad (4.33)$$

where n_0 and h_0 are the characteristic amplitudes of n and h and τ_0 is a typical characteristic time of the signal (e.g the period of the oscillating function $h(t)$). In the limit $T \rightarrow \infty$ the noise term 4.33 averages to zero and the noise has been “filtered out”. In practice T is finite, limited either by the duration of the signal or by the available observing time, but the conclusion already stands: to detect the signal it is *not* necessary that $h_0 > n_0$; it suffices that the noise term in eq. (4.33) is smaller than the signal term in eq. (4.32), which gives the condition

$$h_0 > \left(\frac{\tau_0}{T} \right)^{1/2} n_0. \quad (4.34)$$

For a periodic source with characteristic time of the GW signal $\tau_0 \sim 1$ ms observed for $T = 1$ yr, the gain factor is $(\tau_0/T)^{1/2} \sim 10^{-5}$: one can dig deep into the noise floor. This single estimate already explains why this coherent integration of a *known* waveform is so powerful and foreshadows why *losing* that knowledge, as happens for poorly modelled sources, is so costly.

4.4.2 The optimal linear filter

We now make the procedure precise and optimal. Restricting ourselves to linear filters, we define the filtered statistic

$$\hat{s} = \int_{-\infty}^{\infty} dt s(t) K(t), \quad (4.35)$$

where $K(t)$ is the *filter function*, to be determined. We assume the form of the sought signal $h(t)$ is known. Because $K(t)$ is matched to the signal $s(t)$, the technique is called *matched filtering*.

We define the signal-to-noise ratio (SNR) as S/N , where [12]:

- S is the expected value of $\hat{s}$ when the signal is present.

$$S = \int_{-\infty}^{\infty} dt \langle s(t) \rangle K(t) \stackrel{\langle s \rangle = \langle h+n \rangle = h}{=} \int_{-\infty}^{\infty} dt h(t) K(t) = \int_{-\infty}^{\infty} df \tilde{h}(f) \tilde{K}^*(f). \quad (4.36)$$

Where we used the stationary assumption (4.28), $\langle n \rangle = 0$, namely that the time average of the noise is zero, and rewrote the last integral by passing to the frequency domain with a Fourier transform.

- The denominator N is defined as

$$\begin{aligned} N^2 &= [\langle \hat{s}^2(t) \rangle - \langle \hat{s}(t) \rangle^2]_{h=0} \\ &\stackrel{8}{=} \langle \hat{s}^2(t) \rangle_{h=0} \\ &\stackrel{(4.35)}{=} \int_{-\infty}^{\infty} dt dt' K(t) K(t') \langle n(t) n(t') \rangle \\ &\stackrel{(4.1)}{=} \int_{-\infty}^{\infty} dt dt' K(t) K(t') \int_{-\infty}^{\infty} df df' e^{2\pi i f t - 2\pi i f' t'} \langle \tilde{n}^*(f) \tilde{n}(f') \rangle. \end{aligned} \quad (4.37)$$

Combining the two we get

$$\frac{S}{N} = \frac{\int_{-\infty}^{\infty} df \tilde{h}(f) \tilde{K}^*(f)}{\left[\int_{-\infty}^{\infty} df \frac{1}{2} S_n(f) |\tilde{K}(f)|^2 \right]^{1/2}}. \quad (4.38)$$

⁸The second term drops out because the mean of $\hat{s}$ vanishes when $h = 0$: in that case $s(t) \stackrel{s=h+n}{=} h(t) + n(t) \stackrel{h=0}{=} n(t)$, so that the second term $\langle \hat{s}(t) \rangle_{h=0} = \int dt \langle n(t) \rangle K(t) = 0$, where we used the zero-mean property of the noise 4.28 $\langle n(t) \rangle = 0$. Hence $\langle \hat{s}(t) \rangle_{h=0}^2 = 0$ and $N^2 = \langle \hat{s}(t)^2 \rangle_{h=0}$.

Our objective is to extract the signal from the noise, namely to choose the filter $K(t)$ that maximises the signal-to-noise ratio S/N of eq. (4.38). This variational problem is solved most elegantly by introducing the noise-weighted inner product between two real functions $A(t)$ and $B(t)$,

$$(A|B) \equiv \text{Re} \int_{-\infty}^{\infty} df \frac{\tilde{A}^*(f)\tilde{B}(f)}{\frac{1}{2}S_n(f)} = 4 \text{Re} \int_0^{\infty} df \frac{\tilde{A}^*(f)\tilde{B}(f)}{S_n(f)}, \quad (4.39)$$

where the second equality holds for real functions because then $\tilde{A}(-f) = \tilde{A}^*(f)$ and $S_n(-f) = S_n(f)$.⁹ The geometry of this space is the natural language of matched filtering. Introduce the function $u(t)$ whose Fourier transform is

$$\tilde{u}(f) = \frac{1}{2} S_n(f) \tilde{K}(f), \quad (4.40)$$

so that eq. (4.38) becomes simply

$$\frac{S}{N} = \frac{(u|h)}{(u|u)^{1/2}}. \quad (4.41)$$

Written this way the answer is geometric and immediate: we are seeking the unit vector $\hat{u} = u/(u|u)^{1/2}$ whose projection onto the fixed vector h is largest. By the Cauchy–Schwarz inequality this is achieved when $\hat{u}$ is parallel to h , i.e. when $\tilde{u}(f) \propto \tilde{h}(f)$.¹⁰ Inverting eq. (4.40), the optimal filter is therefore

$$\boxed{\tilde{K}(f) = \text{const.} \times \frac{\tilde{h}(f)}{S_n(f)}}. \quad (4.42)$$

This is the *matched filter*, or *Wiener filter*. The arbitrary constant is irrelevant, since an overall rescaling of $\hat{s}$ does not change its $\frac{S}{N}$ in equation 4.41.

Equation (4.42) has a transparent interpretation. If the noise is white, $S_n(f) = \text{const.}$, the best filter is the signal itself, recovering the naive procedure of eq. (4.31). When the noise is coloured, the filter weights the data by $1/S_n(f)$: it suppresses the frequency bands where the detector is noisy and emphasises those where it is quiet: exactly the behaviour anticipated at the end of Section 4.3.3.

4.4.3 The optimal signal-to-noise ratio

Substituting the optimal filter (4.42) back into eq. (4.41), the arbitrary constant cancels and one obtains the maximum achievable SNR,

$$\left(\frac{S}{N}\right)_{\text{opt}} = (h|h)^{1/2}, \quad (4.43)$$

that is, explicitly,

$$\boxed{\left(\frac{S}{N}\right)_{\text{opt}}^2 \stackrel{(4.39)}{=} 4 \int_0^{\infty} df \frac{|\tilde{h}(f)|^2}{S_n(f)}}. \quad (4.44)$$

Equation (4.44) is one of the most important formulae in gravitational-wave data analysis. It holds for any waveform $\tilde{h}(f)$ and it states that the detectability of a source is governed by the overlap of its spectrum $|\tilde{h}(f)|^2$ with the inverse noise spectral density $S_n(f)$. A signal whose power lies where the detector is quiet is easy to see; a signal whose power lies where the detector is noisy is effectively invisible, no matter how large its intrinsic amplitude. The norm $(h|h)^{1/2}$ is often called the *optimal SNR* and denoted ρ_{opt} ; it is the figure of merit against which every alternative detection statistic, including the ones developed later in this thesis, must ultimately be compared.

It is worth pausing on the logical structure of this result, because it is precisely where the strengths and the weaknesses of matched filtering are joined. The derivation assumed two things: that the noise is a stationary Gaussian process fully described by $S_n(f)$, and that the waveform $\tilde{h}(f)$ is known. Under these

⁹Since $S_n(f) > 0$, this product is positive definite, and it endows the space of finite-norm waveforms with the structure of a real Hilbert space [12].

¹⁰Once the noise-weighted inner product is introduced, the signal-to-noise ratio takes the form $S/N = (u|h)/(u|u)^{1/2}$, to which the Cauchy–Schwarz inequality applies directly. For any two vectors in an inner-product space, $(u|h) \leq (u|u)^{1/2}(h|h)^{1/2}$, with equality if and only if $u \propto h$. Dividing by $(u|u)^{1/2}$ gives $(u|h)/(u|u)^{1/2} \leq (h|h)^{1/2}$, so the ratio is bounded above by $(h|h)^{1/2}$, and this maximum is attained precisely when $u \propto h$.

hypotheses matched filtering is *provably optimal* — no linear statistic can do better. The price is that both hypotheses must hold. If we are not sure of the exact shape of the waveform $\tilde{h}(f)$ (for example, for poorly studied sources), the optimality is lost; if the noise develops non-Gaussian, non-stationary features the very notion of $S_n(f)$, and with it the optimality of eq. (4.42), is undermined. We shall return to both points in Section 4.7.

4.4.4 Matched filtering in practice

The derivation above assumed that the waveform $\tilde{h}(f)$ is known in advance,

$$\left(\frac{S}{N}\right)_{opt}^2 \stackrel{(4.39)}{=} 4 \int_0^\infty df \frac{|\tilde{h}(f)|^2}{S_n(f)}, \quad (4.45)$$

together with stationary Gaussian noise fully characterised by $S_n(f)$. In reality neither the waveform nor its parameters are known ahead of time: the source parameters $\boldsymbol{\theta}$ (the two masses, the spins, and so on) are precisely what we are trying to measure.

The scalar $(S/N)_{opt}$ above is a theoretical ceiling, one we could only reach if we had the exact true waveform coming from the source — but in practice we do not have it; what is actually computed, for each template, is the SNR as a function of the trial arrival time t_* — a whole time series, not a single number. The following exercise makes this concrete.

Example 4.1 (Computing the SNR in practice). We now show, concretely, how the SNR is actually computed in practice. Recall that the real data coming out of an interferometer is a single time series $s(t)$: the detector’s dimensionless *strain*, the fractional stretching $\Delta L/L$ of its arms produced by a passing wave (introduced in Section 4.3.2), which combines a putative gravitational-wave signal with noise, $s(t) = h(t) + n(t)$. There is no single template, but a whole *family* of candidate waveforms,

$$h(t; \boldsymbol{\theta}), \quad (4.46)$$

indexed by the parameter vector $\boldsymbol{\theta} = (\theta_1, \dots, \theta_N)$ introduced above; for a compact binary this is the inspiral waveform of Section 4.6.1, with $\boldsymbol{\theta}$ collecting the two masses, the distance, the sky and orbital orientation, the arrival time, and the phase. Given one such template $h(t; \boldsymbol{\theta})$, we can see what the filtered statistic $\hat{s} = \int dt s(t)K(t)$ of (4.35) becomes once we plug in the optimal filter itself:

- Start from the general filtered statistic, $\hat{s} = \int dt s(t)K(t)$ of (4.35), with $K(t)$ still an arbitrary filter.
- Substitute the matched filter itself (that we derived in Section 4.4.2), $\tilde{K}(f) = \text{const.} \times \tilde{h}(f)/S_n(f)$ of (4.42).
- Passing to the frequency domain, as in (4.36), and dropping the irrelevant overall constant, gives

$$\hat{s} \propto \int_{-\infty}^\infty df \frac{\tilde{h}^*(f) \tilde{s}(f)}{S_n(f)} = (h | s). \quad (4.47)$$

which is exactly the noise-weighted scalar product $(A | B) = 4 \text{Re} \int_0^\infty df \tilde{A}^*(f) \tilde{B}(f)/S_n(f)$ of (4.39), evaluated between the template h and the data s . In other words, the abstract statistic $\hat{s}$ (the numerator of the SNR) reduces to the concrete, computable overlap $(h | s)$ between template and data.

If the template’s arrival time t_* is also unknown — as it generically is — the template is shifted by t_* relative to the data, which shifts its Fourier transform by a pure phase, $\tilde{h}(f; \boldsymbol{\theta}, t_*) = \tilde{h}(f; \boldsymbol{\theta}) e^{2\pi i f t_*}$; the overlap then becomes a function of the trial arrival time t_* ,

$$(h(\boldsymbol{\theta}, t_*) | s) = 4 \text{Re} \int_0^\infty df \frac{\tilde{h}^*(f; \boldsymbol{\theta}) \tilde{s}(f)}{S_n(f)} e^{2\pi i f t_*}, \quad (4.48)$$

^a The value of (4.48) alone is not yet the SNR: it still scales with however the template happens to be normalised. Dividing by the template’s own norm,

$$(h|h) = 4 \int_0^\infty df \frac{|\tilde{h}(f; \boldsymbol{\theta})|^2}{S_n(f)}, \quad (4.49)$$

removes this ambiguity: unlike (4.48), $(h|h)$ does not depend on t_* , only on the shape of the template, so it is a single fixed number for each choice of $\boldsymbol{\theta}$.^b This number is exactly the denominator that entered the general SNR formula (4.38), $N^2 = \int df \frac{1}{2} S_n(f) |\tilde{K}(f)|^2$. Combining the two gives the SNR as a genuine time series in the arrival time t_* ,

$$\rho(t_*) = \frac{\hat{s}(t_*)}{N} = \frac{(h(\boldsymbol{\theta}, t_*)|s)}{(h|h)^{1/2}}, \quad (4.50)$$

but, crucially, this SNR time series $\rho(t_*)$ is still for one fixed, precise combination of the remaining parameters $\boldsymbol{\theta}$, e.g. one particular choice of the two component masses m_1, m_2 and spins. In practice this whole procedure is repeated for many templates, i.e. many choices of $\boldsymbol{\theta}$, and for each one $\rho(t_*)$ is scanned over many candidate arrival times, producing a large collection of “events”. Deciding which, if any, of these events amounts to a genuine detection is exactly the statistical question taken up in the next Section 4.5.

^aThe same construction is worked out concretely, together with its computational cost, for the compact-binary waveform in Section 4.6.3.

^bShifting a template in time does not change $(h|h)$ because a time shift only multiplies its Fourier transform by a phase, $\tilde{h}(f; \boldsymbol{\theta}, t_*) = \tilde{h}(f; \boldsymbol{\theta}) e^{2\pi i f t_*}$, and this phase disappears once we take the squared modulus $|\tilde{h}(f; \boldsymbol{\theta}, t_*)|^2 = |\tilde{h}(f; \boldsymbol{\theta})|^2$ appearing in (4.49).

The practical strategy is therefore to matched-filter the data against many different *templates* — candidate waveforms $h(t; \boldsymbol{\theta})$, each computed for one hypothesised set of source parameters $\boldsymbol{\theta}$; how a whole bank of such templates is chosen and spaced is discussed in Section 4.6.4. Running the matched filter against every template in the bank produces a list of “events”: cases in which the SNR, for some template, rises above a predetermined threshold. Applying the maximum-likelihood criterion (or maximising the posterior probability, if prior information on $\boldsymbol{\theta}$ is available) to the events found this way then gives the most probable value of the parameters $\boldsymbol{\theta}$, under the hypothesis that a gravitational-wave signal $h(t; \boldsymbol{\theta})$ is genuinely present.¹¹

4.5 Probability, statistics and detection

Matched filtering tells us how to maximise the SNR *assuming a signal is present*. The experimenter, however, faces a logically prior question. We do not know whether a gravitational wave is present at all, and we know even less its precise waveform. In practice one applies the filter many times, with many candidate arrival times and many template waveforms, and extracts a collection of “events” with various values of S/N . What can be concluded? When may we claim a detection? These questions are statistical, and we treat them following the discussion in [12].

4.5.1 Frequentist and Bayesian frameworks

Probability can be axiomatised abstractly through the Kolmogorov axioms and the definition of the conditional probability

$$P(A|B) = \frac{P(A \cap B)}{P(B)}, \quad (4.51)$$

but the interpretation of the events $A, B, \dots$ splits into two schools.

In the *frequentist* interpretation, $A, B, \dots$ are outcomes of a repeatable experiment and $P(A)$ is the limiting frequency of A over infinitely many repetitions. Quantities such as $P(\text{data}|\text{hypothesis})$ are then well defined, but one may never speak of the probability that a hypothesis is true or that a parameter takes a given value: a hypothesis is simply right or wrong, and the true value of a parameter is what it is.

¹¹We do not go into the detail of parameter inference in this thesis, look at [12] for further details.

In the *Bayesian* interpretation, instead, one is allowed to assign a probability to a hypothesis or to a parameter value. Starting from the identity $P(A \cap B) = P(A|B)P(B) = P(B|A)P(A)$ one obtains *Bayes' theorem*,

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}, \quad (4.52)$$

and, applying it with $A = \text{hypothesis}$ and $B = \text{data}$,

$$\underbrace{P(\text{hyp} | \text{data})}_{\text{posterior}} \propto \underbrace{P(\text{data} | \text{hyp})}_{\text{likelihood}} \underbrace{P(\text{hyp})}_{\text{prior}}. \quad (4.53)$$

The posterior is the degree of belief in the hypothesis after the measurement; the prior encodes the belief before it, and is in general subjective.

The choice between the two frameworks is, for a working physicist, largely pragmatic, and gravitational-wave astrophysics offers a clean illustration of when each is appropriate.

- *Negative results* — upper limits on the rate of some class of events — are naturally cast in frequentist terms, because they concern an ensemble of repeated observations [12].
- The parameter estimation that follows a *positive* detection, on the other hand, is intrinsically Bayesian: a coalescing binary detected at a given instant is a unique event, and there is no ensemble of identical binaries at the same position with the same masses. We just have that one event, and we want to extract from it the most probable values of the source's distance, direction, masses and spins.

We therefore adopt a frequentist language for thresholds and false-alarm rates, and a Bayesian language for parameter estimation.

4.5.2 Detection statistic, threshold and false alarms

To recap where we stand: the general strategy is to matched-filter the data against many templates $h(t; \theta)$ (Section 4.4.4), producing a list of “events” wherever the SNR, for some template, rises above a predetermined threshold; maximum likelihood then extracts the most probable parameters θ under the hypothesis that a real signal $h(t; \theta)$ is present. What we have not yet asked is how much any single event should be trusted: given that we found an event at some level of SNR, what is its statistical significance? When, in other words, are we allowed to claim a detection?

The answer depends entirely on the statistical behaviour of the noise, and here it is useful to distinguish two very different regimes.

- On one hand there is “well-behaved” *Gaussian* noise, whose probability of producing a given value falls off extremely fast — like $e^{-x^2/2}$ — the further that value sits from zero. Gaussian noise can be beaten simply by setting the SNR threshold high enough, since very large fluctuations are exponentially rare.
- On the other hand there is everything else, generically called *non-Gaussian* noise. It typically has long tails that decay only as a power law, so it can produce large SNR values far more often than a Gaussian ever would; no matter how high the threshold is pushed, some non-Gaussian noise still leaks through. In principle one could measure the true noise distribution and raise the threshold until even non-Gaussian fluctuations become rare — but in practice the resulting threshold would be so high that it would badly degrade the detector's sensitivity to genuine signals.

We therefore proceed in two steps. In the remainder of this subsection we work out the statistical significance of a given SNR value under the idealised assumption that only Gaussian noise is present; in the following we'll fix a threshold above which, at some chosen confidence, an event cannot be attributed to Gaussian noise alone, and we'll turn the raw output of the detector into a list of candidate events. Only afterwards, in Section 4.7.3, do we confront the fact that real detector noise is not purely Gaussian.

So far the SNR of (4.38) was defined through the *expectation value* of the filtered output $\hat{s}$ of (4.35); to study its full statistical distribution, rather than just this one number, we need the random variable itself. Define

$$\rho \equiv \frac{\hat{s}}{N}, \quad (4.54)$$

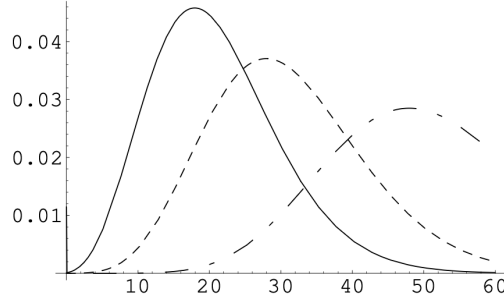


Figure 4.5: The energy-SNR distribution $P(R | \bar{R})$ of (4.57), plotted against R for three values of the optimal energy SNR: $\bar{R} = 20$ (solid), $\bar{R} = 30$ (dotted) and $\bar{R} = 50$ (dot-dashed). A stronger signal shifts the distribution to higher R and broadens it. Image has been collected from [12].

with N the same normalisation introduced in Section 4.4.2 ((4.37)).¹² This is the same construction as S/N of (4.38), except that the numerator is now the random variable $\hat{s}$ itself, rather than its average $S = \langle \hat{s} \rangle$; consequently the SNR we discussed in Section 4.4.2 is just $S/N = \langle \rho \rangle$.

- With simple statistical steps we can prove that, in the absence of a signal ($h = 0$), the probability distribution $p(\rho | h = 0) d\rho$ of the random variable ρ is a unit-variance Gaussian with zero mean,

$$p(\rho | h = 0) d\rho = \frac{1}{\sqrt{2\pi}} e^{-\rho^2/2} d\rho. \quad (4.55)$$

To see the passages that lead to this Gaussian distribution, see Appendix A.36.

- Similarly, if a genuine signal with optimal SNR $\bar{\rho}$ ((4.44)) is present, the same kind of steps show that ρ is Gaussian again, but now centred on $\bar{\rho}$ rather than on zero,

$$p(\rho | \bar{\rho}) d\rho = \frac{1}{\sqrt{2\pi}} e^{-(\rho - \bar{\rho})^2/2} d\rho. \quad (4.56)$$

A genuine signal, in other words, does not change the *shape* of the distribution at all: it simply slides the same bell curve sideways by $\bar{\rho}$. The derivation is in Appendix A.37.

- The sign of ρ carries no physical meaning, and the energy carried by a gravitational wave is quadratic in its amplitude, so it is more natural to work with the SNR in energy, $R \equiv \rho^2 \geq 0$. Combining the two branches $\rho = \pm\sqrt{R}$ of (4.56) gives its distribution,

$$P(R | \bar{R}) dR = \frac{1}{\sqrt{2\pi R}} e^{-(\bar{R} + R)/2} \cosh(\sqrt{R\bar{R}}) dR, \quad (4.57)$$

with $\bar{R} \equiv \bar{\rho}^2$; setting $\bar{R} = 0$ recovers the noise-only case (Figure 4.6 below). The change-of-variables argument behind (4.57) is in Appendix A.38.

- Its mean is

$$\langle R \rangle = 1 + \bar{R}, \quad (4.58)$$

proved in Appendix A.39: writing $R = E/kT_n$, with T_n playing the role of an effective noise temperature, this reads $\langle E \rangle = kT_n + \bar{E}$ — the detected energy is, on average, exactly the signal energy $\bar{E}$ plus one “unit” of noise kT_n , a reassuringly simple result.

The different behaviour of the two distributions in Figure 4.6 is precisely what makes a detection possible: choosing a threshold R_t that sits well above the noise-only distribution, while still capturing a good fraction of the signal distribution, lets us discriminate a real gravitational-wave signal from a fluctuation of Gaussian noise. This choice cannot eliminate errors altogether, since the two distributions

¹²This N the r.m.s. of $\hat{s}$ in the absence of a signal [12].

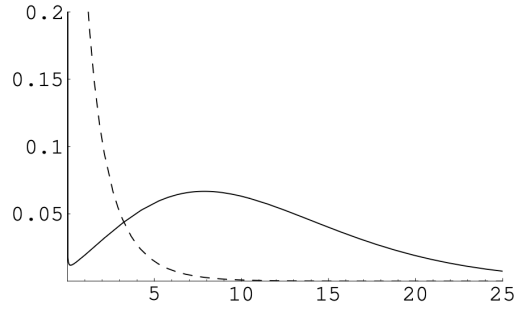


Figure 4.6: The energy-SNR distribution $P(R | \bar{R})$ for a signal of optimal energy SNR $\bar{R} = 10$ (solid), compared against the noise-only distribution $P(R | \bar{R} = 0)$ (dashed). Even at this comparatively modest signal strength the two curves are already visibly separated. Image has been collected from [12].

always overlap somewhat. There is a *false-alarm probability*, the chance that noise alone exceeds the threshold:

$$p_{\text{FA}} = \int_{R_t}^{\infty} dR P(R | \bar{R} = 0) \stackrel{(4.57)}{=} 2 \int_{\rho_t}^{\infty} d\rho e^{-\rho^2/2} = 2 \operatorname{erfc}\left(\frac{\rho_t}{\sqrt{2}}\right), \quad (4.59)$$

¹³ With erfc the complementary error function, and we can also compute the *false-dismissal probability*, the chance of missing a genuine signal,

$$p_{\text{FD}} = \int_0^{R_t} dR P(R | \bar{R}). \quad (4.60)$$

14

Therefore the threshold R_t is fixed by deciding how many false alarms can be tolerated, and this depends sharply on how many independent trials the search performs — how many templates, and how many candidate arrival times, are tried. We work out this trials-counting argument concretely, and the resulting numerical threshold, once the template bank itself has been introduced in Section 4.6.4.

This whole construction, however, rests on the assumption that the noise is Gaussian. Real instrumental noise is not: it contains *glitches*, short and loud non-Gaussian disturbances discussed in Section 4.7.3, which populate the tail of (4.55) far more heavily than the idealised Gaussian formula predicts, and this is what undermines the whole construction above. A threshold calibrated on Gaussian statistics does nothing to suppress a population of noise transients that are, by construction, not Gaussian. The standard remedy is to require *coincidence*: a genuine gravitational wave should register, at compatible times, in every operating detector, whereas an instrumental glitch is local to one instrument and will essentially never mimic a coincident signal across a network. This is one of the main reasons several detectors are built and operated together, as a network, rather than relying on any single instrument.

4.6 Application to coalescing compact binaries

We now specialise the general matched-filtering framework of Section 4.4.4 to the concrete case of compact binary coalescences (CBCs): we work out the explicit inspiral waveform, its optimal SNR, and how a bank of templates is built to cover its parameter space.

4.6.1 The inspiral waveform and the chirp mass

During the inspiral the two bodies follow a quasi-circular orbit that shrinks as energy is radiated, so that both the gravitational-wave frequency $f_{\text{gw}}(t)$ and the amplitude grow with time — the characteristic

¹³In the first step we integrate $dR P(R | \bar{R} = 0)$, the distribution of the energy SNR R in the case where no gravitational wave is present ($\bar{R} = 0$), over the interval $[R_t, \infty)$ where the energy SNR exceeds the threshold; the second step then rewrites this integral in terms of $\rho = \sqrt{R}$ using (4.57) at $\bar{R} = 0$.

¹⁴Here we integrate $dR P(R | \bar{R})$, the distribution of the energy SNR R in the case where a genuine gravitational-wave signal of optimal energy SNR $\bar{R}$ is present, over the interval $[0, R_t)$ where the energy SNR falls *below* the threshold: exactly the cases in which a real signal would fail to trigger a detection.

“chirp”. To leading (Newtonian, or 0PN) order the strain at the detector can be written as [12]

$$h(t) = A \left[\frac{\pi f_{\text{gw}}(t)}{c} \right]^{2/3} \cos[\Phi(f_{\text{gw}}(t)) + \varphi], \quad (4.61)$$

where the amplitude $A = (A_+^2 + A_\times^2)^{1/2}$ collects the two polarisation amplitudes

$$A_+ = \frac{4}{r} \left(\frac{G\mathcal{M}_c}{c^2} \right)^{5/3} F_+(\theta, \phi) \frac{1 + \cos^2 \iota}{2}, \quad (4.62)$$

$$A_\times = \frac{4}{r} \left(\frac{G\mathcal{M}_c}{c^2} \right)^{5/3} F_\times(\theta, \phi) \cos \iota, \quad (4.63)$$

r is the distance to the source, ι the inclination of the orbital plane to the line of sight, and $F_+, F_\times$ are the detector’s pattern functions, encoding how its response depends on the sky position and polarisation angle of the source; their derivation is deferred to Appendix B. The all-important combination of masses is again the chirp mass $\mathcal{M}_c$ of (4.18).

$$f_{\text{gw}}(\tau) = \frac{1}{\pi} \left(\frac{5}{256} \frac{1}{\tau} \right)^{3/8} \left(\frac{G\mathcal{M}_c}{c^3} \right)^{-5/8} \quad (4.64)$$

The chirp mass governs both the amplitude (4.63) and, at leading order, the rate of frequency sweep¹⁵ (4.64), and is consequently the best-measured parameter of an inspiral. The frequency evolution $f_{\text{gw}}(t)$ follow from the quadrupole formula (see [12]) applied to a quasi-circular orbit and are known, in the post-Newtonian expansion, up to high order; their accuracy is what makes the template faithful.

4.6.2 The optimal SNR and the sight distance

Inserting the inspiral waveform into the optimal-SNR formula (4.44) gives a result of great practical importance (see [12] for a precise derivation): the SNR of an inspiral scales as

$$\frac{S}{N} \propto \frac{\mathcal{M}_c^{5/6}}{r} \left[\int df \frac{f^{-7/3}}{S_n(f)} \right]^{1/2}, \quad (4.65)$$

so that, for a fixed detector and a fixed threshold, the maximum distance out to which a source can be seen — the *sight distance* or *horizon* — grows as $\mathcal{M}_c^{5/6}$. Heavier binaries are visible farther away, and the volume surveyed, and hence the expected event rate, scales steeply with chirp mass. The integral in eq. (4.65) is dominated by the low-frequency end, which is why pushing the seismic wall of Figure 4.4 downward in frequency is such a high priority for next-generation instruments: it directly increases the reach for inspirals. Figure 4.7 illustrates the chirping waveform together with its time–frequency representation.

4.6.3 Eliminating the extrinsic parameters

The class of sources of direct relevance to this thesis is that of *compact binary coalescences* (CBCs): the inspiral and merger of two black holes or neutron stars. These are *short* and *modelled* signals:

- *short*, because a stellar-mass binary sweeps through the detectable band in seconds;
- *modelled*, because general relativity predicts the waveform with great accuracy.

They are therefore the textbook case for matched filtering, and indeed every confident ground-based¹⁶ detection to date belongs to this class [57]. Precisely because the method works so well here, the CBC

¹⁵The frequency sweep refers to the continuous acceleration of the gravitational-wave frequency over time. As the binary system loses energy and angular momentum via radiation, the orbit shrinks, forcing the bodies to revolve faster. Mathematically, the rate of this sweep is given at leading Newtonian order by [12] $\frac{df_{\text{gw}}}{dt} \propto \mathcal{M}_c^{5/3} f_{\text{gw}}^{11/3}$.

¹⁶Remember that with ground-based detectors we mean detectors like Virgo, that are located in the surface on Earth, differently from future detectors like LISA that are space-based detectors.

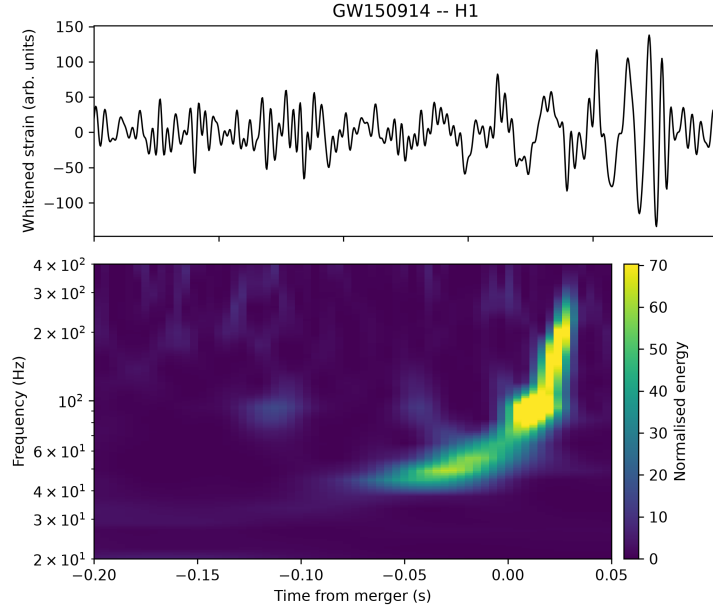


Figure 4.7: Inspiral–merger–ringdown signal of a coalescing binary and its time–frequency representation. The rising “chirp” track is the signature that template-based and, later, topological methods both attempt to recognise.

problem is also the cleanest setting in which to see *why* matched filtering becomes expensive, and where a complementary strategy could help.

A full inspiral waveform depends on fifteen parameters¹⁷. Searching a fifteen-dimensional grid would be hopeless, but a crucial simplification is that several parameters are *extrinsic*: they affect the overall amplitude, phase or timing of the waveform but not its intrinsic morphology, and can be maximised over analytically or with a single transform rather than by gridding.

The clearest example is the time of arrival t_* , defined as the instant the signal enters the band (say at $f_{\text{gw}} = 10$ Hz). A template arriving at t_* is a time translation of the template at $t_* = 0$, so its Fourier transform acquires only a phase $e^{2\pi i f t_*}$, and using the definition of the scalar product 4.39 the overlap is¹⁸

$$(h(\boldsymbol{\theta}, t_*) | s) = 4 \operatorname{Re} \int_0^\infty df \frac{\tilde{h}^*(f; \boldsymbol{\theta}) \tilde{s}(f)}{S_n(f)} e^{2\pi i f t_*} \quad (4.66)$$

which is the Fourier transform of $\tilde{h}^*(f; \boldsymbol{\theta}) \tilde{s}(f) / S_n(f)$. A *single* fast Fourier transform therefore evaluates the statistic for all arrival times at once [12]. The computational saving is enormous: scanning a year of data ($\sim 3 \times 10^7$ s) with a 3 ms time step would require $\sim 10^{10}$ scalar products per template, whereas one FFT, at cost $\mathcal{O}(N \log N)$ rather than $\mathcal{O}(N^2)$, does the same job [12]. Figure 4.8 sketches the resulting trigger. The amplitude and the reference phase are likewise extrinsic and can be maximised analytically. What remains are the *intrinsic* parameters — chiefly the two masses and the spins — which genuinely change the shape of the waveform and must be searched explicitly.

It is the *peak* of this time series $\rho(t_*)$ — maximised, in practice, both over t_* and over every template in the bank — that is compared against the statistical threshold ρ_t introduced in Section 4.5.2 to decide whether an event is significant.

4.6.4 Template banks

Even after eliminating the extrinsic variables (as discussed in Section 4.6.3), the intrinsic parameter space must be covered by a discrete set of templates — a *template bank*. The bank must be dense enough

¹⁷Characterizing a compact binary merger within a post-Newtonian framework requires a set of 15 independent parameters θ_i [12]. This parameters space encompasses the source distance, its sky coordinates (two angles), the spatial orientation of the orbital angular momentum vector (two additional angles), the time at which the signal enters the detector’s sensitivity band alongside the corresponding initial orbital phase, the individual masses of the two objects, and the six components specifying their respective three-dimensional spin vectors.

¹⁸Notice that $s(t)$ is the detector output while the $\tilde{s}(f)$ is its Fourier transform. The template we are testing is $h(\boldsymbol{\theta}, t_*)$ that arrives at a time t_* .

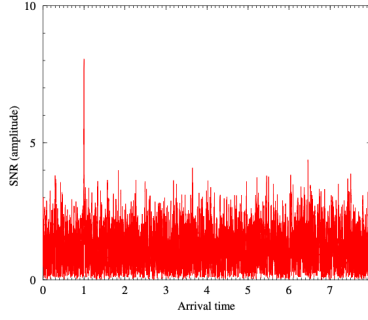


Figure 4.8: Matched-filter SNR as a function of trial arrival time. The single FFT of Eq. (4.66) produces a spike at the true arrival time, turning a costly time scan into one transform. Image has been collected from [12].

that any real signal lies close, in the sense of the scalar product (4.39), to some template; the standard requirement is that the *minimal match* between neighbouring templates exceed a chosen value [12], so that the loss of SNR due to discreteness is at most a few percent. The number of templates needed grows rapidly with the dimension of the intrinsic space and with the bandwidth and low-frequency reach of the detector: modern compact-binary banks contain $\mathcal{O}(10^5\text{--}10^6)$ templates [12], and each must be filtered against the data of every detector continuously and, ideally, in low latency. Spinning, precessing and eccentric systems enlarge the space further, increasing the computational cost.

This bank size feeds directly into the false-alarm threshold ρ_t of Section 4.5.2: this threshold ρ_t is fixed by deciding how many false alarms can be tolerated, which depends sharply on how many independent trials the search performs. For example, for a coalescing binary GW detection, one can estimate that one has to try $\sim 10^{15}$ templates for one year of data.¹⁹ The false-alarm level is usually set so that the expected number of false alarms in a run will be of order a few:

- If we search for a coalescence in a single detector, with one year of data and 10^{15} templates, we could choose a threshold in amplitude $\rho_t \simeq 8$, since this gives, via (4.59), $p_{\text{FA}} \sim 2.5 \times 10^{-15}$ (assuming only Gaussian noise, which unfortunately is not the case in real scenarios).
- And indeed, multiplying this probability by the number of independent trials gives exactly the handful of false alarms we were aiming for: $p_{\text{FA}} \times 10^{15} \approx 2.5 \times 10^{-15} \times 10^{15} \approx 2.5$, of order a few, as promised.

With multiple detectors the situation improves further, since it becomes much harder for noise alone to fake a detection in more than one instrument at once; we refer to [12] for a precise discussion.

Matched filtering is optimal, but its optimality is bought with (i) an accurate waveform model for every point of the search space and (ii) a computational budget that scales with the size of the template bank. Although both costs are tolerable for the well-characterized CBC signals, they remain significant enough that low-latency searches require a substantial engineering effort [12].

4.7 The case for new search strategies

The preceding sections have built up matched filtering as the optimal linear detection statistic and have shown how it is deployed for compact binaries. Along the way, three assumptions have repeatedly entered the derivation, each of which is, in the real world, only approximately true:

- that we *know the waveform* we are looking for, since the filter (4.42) is built out of the signal itself;
- that the parameter space can be *searched exhaustively*, covering every candidate waveform and arrival time within an affordable computational budget, as the template bank of Section 4.6.4 presumes;

¹⁹Before, we just said that the number of templates necessary for a binary coalescence was $\mathcal{O}(10^5\text{--}10^6)$, but now we say that one actually has to try $\mathcal{O}(10^{15})$ templates. The reason is that here we are analysing a full year of data: to see where this $\mathcal{O}(10^{15})$ comes from, see Appendix A.40.

- that the noise is *stationary and Gaussian*, fully described by the single spectral density $S_n(f)$ of (4.29), on which the detection statistics of Section 4.5.2 rest.

When they hold, matched filtering is unbeatable; when they fail, its performance degrades and its cost can explode. We collect these limitations here, because together they constitute the motivation for the methods developed in the remainder of this thesis.

4.7.1 The need for an accurate waveform model

The whole power of matched filtering rests on the assumption, emphasized back in Section 4.4, that we know the form of the signal. The optimal SNR (4.44) is reached only by a template that reproduces the *true* waveform,

$$\left(\frac{S}{N}\right)_{\text{opt}}^2 = 4 \int_0^\infty df \frac{|\tilde{h}_{\text{true}}(f)|^2}{S_n(f)}, \quad (4.67)$$

where $\tilde{h}_{\text{true}}(f)$ is the waveform the source actually emitted. Everything below is about what happens when we cannot write that waveform down, or can write down only an approximation to it.

The first problem is that certain sources are simply not modelled: we are not able to compute the gravitational waveform they generate. Following the standard classification of sources [57], signals are either modelled or unmodelled and either short or long:

- **Compact binary coalescences** — short and modelled. The natural home of matched filtering.
- **Continuous waves** — long and modelled, emitted for instance by spinning neutron stars. Modelled in principle, but their long duration makes a fully coherent matched-filter search computationally intractable (see below).
- **Bursts** — short and *unmodelled*, expected from core-collapse supernovae or neutron-star glitches. Because the waveform cannot be reliably predicted, no template bank can be built, and one must fall back on generic excess-power or coherent-network methods.
- **Stochastic backgrounds** — long and unmodelled, cosmological or astrophysical in origin, sought through cross-correlation of detector pairs rather than matched filtering.

Even when a source *is* modelled, however, we never hold the true waveform in our hands. What the search actually uses is a template $\tilde{h}_t(f; \boldsymbol{\theta}_t)$, computed for some trial parameters $\boldsymbol{\theta}_t$ taken from the bank, and this is at best close to — never identical with — the emitted waveform $\tilde{h}_{\text{true}}(f; \boldsymbol{\theta}_{\text{true}})$. It is worth seeing what this costs us. The data still contain the true signal, $s = h_{\text{true}} + n$, but the filter (4.42) is built out of h_t , so the statistic (4.50) that the search evaluates is

$$\rho = \frac{(h_t | s)}{(h_t | h_t)^{1/2}}, \quad \text{whose average over the noise is} \quad \langle \rho \rangle = \frac{(h_{\text{true}} | h_t)}{(h_t | h_t)^{1/2}}, \quad (4.68)$$

since the noise term averages to zero by (4.28). In words: what we recover is not the full strength of the signal, but only the part of it that the template manages to overlap. Comparing with (4.43), the Cauchy–Schwarz inequality — the same one that gave us the optimal filter in Section 4.4.2 — guarantees that

$$\langle \rho \rangle \leq (h_{\text{true}} | h_{\text{true}})^{1/2} = \left(\frac{S}{N}\right)_{\text{opt}}, \quad (4.69)$$

with equality only if h_t is proportional to h_{true} . A template that is merely *close* to the truth therefore always yields *less* than the optimal signal-to-noise ratio, and the further the template drifts, the lower is the SNR we measure.

Unmodelled sources: for the unmodelled classes the matched filter is not even available: one cannot match what one cannot write down. What is done instead is instructive. For *bursts* the template is abandoned altogether, and one looks for an unexplained excess of power appearing in the same stretch of time and the same band of frequencies in several detectors at once — the strategy of pipelines such as coherent WaveBurst [58]. The trade is a sharp one:

- **What is given up.** Sensitivity. With no template to overlap with, the search cannot collect the signal the way (4.68) does, and it is less sensitive than a matched filter would be on a signal it knew how to model — this is (4.69) pushed to its extreme.
- **What is gained.** Generality. Nothing about the shape of the signal is assumed: all that is asked of a candidate is that it be loud, brief, and seen by more than one instrument, so that *any* sufficiently loud transient can be found, whatever its shape.

Modelled sources: The degradation (4.69) is not a worry confined to the unmodelled sky. Our templates are accurate, but not exact — post-Newtonian and numerical waveforms carry their own systematic errors — and a bank built on general relativity is, by construction, a bank of the waveforms we already expect. A signal from physics we have not anticipated, such as a strong-field deviation from general relativity or an exotic compact object, would be treated by the search as an imperfect template: recovered with degraded signal-to-noise ratio, or, if the difference is large enough, not recovered at all. A method that recognises a signal from its generic structure, rather than from a precise template, could be therefore valuable both as a safety net for the modelled searches and as a primary tool for the unmodelled ones.

4.7.2 Computational cost and the template-bank bottleneck

The second limitation is cost. Matched filtering has an important computational cost and we have already met, scattered through this chapter, the three factors that multiply together to set the cost for a compact-binary search. It is worth collecting them in one place:

- **We must try many waveform shapes.** The masses and spins are unknown, so the intrinsic parameter space has to be covered by a discrete bank of 10^5 – 10^6 templates (Section 4.6.4), each of which is a separate filtering of the data.
- **We must try many arrival times.** We do not know when the signal came in either, so every template has to be tested against every plausible arrival time — some 10^{10} of them in a year of data. Here we are rescued by the trick of Section 4.6.3: a single fast Fourier transform tries all arrival times at once, turning what would be an $\mathcal{O}(N^2)$ scan into an $\mathcal{O}(N \log N)$ transform (Appendix A.41).
- **The two costs multiply.** Shapes times arrival times gives the $\sim 10^{15}$ independent trials of Appendix A.40 — the very number that forced the high detection threshold of Section 4.5.2. Searching harder therefore does not only cost computing time: it also raises the bar a candidate must clear before we may believe it.

This is not, today, a crippling problem. Compact-binary searches run continuously on real data, and they run fast enough to issue alerts while a telescope might still catch the afterglow. But the arithmetic above is fragile in two directions:

The first is the *dimension* of the parameter space. The bank size quoted above is the cost of covering only the masses and a limited treatment of spin [12]. Every physical effect we stop neglecting adds new directions to that space — fully precessing spins, orbital eccentricity, higher-order multipoles of the radiation — and because the bank must stay dense in *every* direction at once, each addition multiplies the number of templates.

The second is the *low-frequency reach* of the detectors.

- **Why it matters.** This follows from a formula we have already written down: the frequency-sweep relation (4.64) gives the time a binary spends in band as $\tau \propto f_{\text{gw}}^{-8/3}$, so listening from a lower starting frequency lengthens the signal steeply. A neutron-star binary entering the interferometer band at 20 Hz chirps for a couple of minutes; entering the interferometer band at a few hertz, as the next generation of observatories intends, it would stay in band for hours.
- **Why that costs more.** A longer signal has to be matched by a correspondingly longer template, so each filtering operation runs over a larger number of samples N — and, by the $\mathcal{O}(N \log N)$ scaling of Appendix A.41, every template in the bank becomes more expensive to try.

- **What it does not mean.** It does not make matched filtering unusable, but it does mean that the brute-force strategy of Section 4.6.4 will have to be supplemented — by hierarchical schemes, by cheaper approximate filters, or by methods of a different kind altogether. That is precisely the opening this thesis is interested in: a fast, generic first pass that can narrow down where the expensive matched filtering machinery needs to be pointed.

4.7.3 Non-Gaussian, non-stationary noise and glitches

The matched filter and its statistics were built on the assumption that the noise is stationary and Gaussian, fully described by a single spectral density $S_n(f)$. Real interferometer noise is neither. It drifts on timescales of minutes to hours (non-stationarity), and it is contaminated by a lot of short, loud, non-Gaussian transients known as *glitches* — scattered-light arches, blips, whistles and many others — arising from the instrument and its environment. These transients populate the tail of the detection statistic far more heavily than the Gaussian prediction (4.55), so that the clean relation (4.59) between threshold and false-alarm probability no longer holds. A glitch can ring up a template and produce a spurious high-SNR trigger, while a glitch overlapping a real signal can corrupt its parameter estimation.

There are, of course, defences. The best known is the χ^2 *time-frequency discriminator*, which checks whether the power that produced a trigger is distributed across frequency in the way the template says it should be; we refer to Allen [59] for the construction and to [12] for the complementary strategies of environmental vetoes and multi-detector coincidence.

A great deal of practical work, then, goes into telling glitches apart from real signals. What is worth noticing is *how* this is increasingly done. Glitches are not random, formless noise: each family has its own repeated, recognisable shape in the time-frequency plane. Telling a glitch from a chirp, then, is less like checking whether the data match one specific, pre-computed waveform, and more like recognising which of a few familiar shapes is present. Matched filtering is not built for that job: by construction it only ever asks “how well do the data agree with *this one* template?”, one template at a time. A method built instead to recognise general shapes — of real signals and of the glitches that imitate them alike — is better suited to this kind of classification. This is the third and last limitation of matched filtering that we set out to examine, and like the other two, it points towards the same kind of alternative.

4.7.4 The case for a new detection strategy

The three limitations above point in a consistent direction. There is room — alongside, not instead of, matched filtering — for a detection strategy that is (i) fast, so that it can pre-screen long data streams before the expensive matched-filter and human-vetting stages; (ii) generic, so that it responds to the broad morphology of a chirp rather than to a specific template, and is therefore robust to waveform uncertainty and to the diversity of source parameters; and (iii) resilient to non-Gaussian noise, because it does not assume a Gaussian likelihood.

Machine-learning methods, and convolutional neural networks (CNNs) in particular, have been proposed in exactly this role. The pioneering work [23], and its many successors, showed that an appropriately trained CNN can flag the presence of a compact-binary chirp directly in noisy strain data, acting as an excellent first-pass filter that precedes the more specific and computationally expensive matched-filter and electromagnetic-follow-up steps. A trained network evaluates a data segment in essentially constant time, sidestepping the template-bank bottleneck of Section 4.7.2, and — because it learns from examples rather than from an explicit likelihood — it can in principle accommodate non-Gaussian noise and signals with varied signatures. Its weaknesses, however, are equally well documented: a plain CNN on raw strain requires a very large and carefully constructed training set, and its accuracy degrades badly at low SNR, precisely the regime where help is most needed.

This chapter set out to make precise what matched filtering achieves, why it is optimal under its assumptions, and where exactly those assumptions break down. We can now restate the breakdown as a single, closing list:

- we do not always *know the waveform* we are looking for: whole classes of sources are unmodelled, and even a modelled template is never exact, as Section 4.7.1 showed;

- the parameter space cannot always be *searched exhaustively*: the template bank of Section 4.6.4 already runs to 10^5 – 10^6 templates for compact binaries, and Section 4.7.2 showed how quickly that cost can become prohibitive;
- the noise is not, in practice, *stationary and Gaussian*: it drifts, and it is contaminated by the non-Gaussian glitches of Section 4.7.3, which break exactly the statistics on which the detection threshold of Section 4.5.2 relies.

In other words, none of the three assumptions of matched filtering is exactly true. The next generation of ground-based detectors, such as the Einstein Telescope [9, 50, 51], is designed to reach lower frequencies precisely where Section 4.7.2 showed the computational burden grows fastest; and a space-based observatory such as LISA [11] will face overlapping signals and non-stationary instrumental artefacts of its own. This is exactly the space in which it is worth developing a genuinely new detection technique — fast, generic, and robust in the ways matched filtering, by construction, is not. The following chapter tries to develop a new approach for GW detection.

Chapter 5

Topological Data Analysis for Gravitational Waves

Chapter 4 showed that matched filtering is a powerful tool, but it has some limitations. The waveform is assumed known exactly, yet some sources have no precise gravitational-wave template [57] and even binary coalescences are modelled only approximately [12]. The noise is assumed Gaussian and stationary, yet real detectors drift and carry short, loud transients, or glitches, which can imitate a real signal [12]. And since the source parameters are unknown in advance, matched filtering must scan a bank of thousands to hundreds of thousands of waveform templates [12], which means that the method is computationally intensive. These conditions motivate the search for complementary methods for gravitational-wave detection: in this thesis we investigate whether a topological description of the shape of the data carries usable information about the presence of a gravitational wave, and at what computational cost it can be obtained.

The strategy we develop augments some machine learning classifiers with a *feature-extraction* stage based on *topological data analysis*, specifically the persistent homology of Takens embeddings of the time series, elaborating on the approach of [24]. The idea is to exploit a simple but powerful geometric fact: when a time series is embedded into a high-dimensional space by the Takens-embedding construction (2.3.1), a periodic signal traces out a closed loop, a decaying chirp traces a spiral, and noise fills a featureless “ball”. These shapes are distinguished by their *homology* (1.4) — loosely, by their numbers of connected components and independent loops — and persistent homology (Definition 1.35) provides a stable, quantitative summary of this topological structure, the *persistence diagram* (Definition 1.36).

In the following chapter the computational analysis was performed in Python. Numerical operations and data manipulation relied primarily on NumPy [60] and SciPy [61]. The gravitational-wave data analysis was carried out using PyCBC [27], while the topological data analysis, including Takens embeddings, persistent homology and the extraction of topological descriptors, was implemented using Giotto-TDA [34]. Machine-learning models and the associated statistical analysis were implemented using scikit-learn [62] and PyTorch [63]. Matplotlib [64] and Plotly [65] were used for data visualization.

5.1 Preparing a phenomenologically realistic dataset

Our first objective is to prepare a dataset of time series representing the noisy output of a gravitational-wave interferometer, into which an actual gravitational-wave signal has been injected. Building such a dataset in a way that reflects how real gravitational-wave data actually behaves is itself a design problem. We take as our main reference [25], together with [66] and [28], all three of which train neural-network classifiers on simulated gravitational-wave data and are correspondingly careful about how that data is generated.

More precisely, we are trying to generate a dataset with these overall characteristics:

- The dataset is composed of N time series, and each time series is the noisy output (the strain) of an interferometer, which may or may not contain a gravitational-wave signal. Every time series has the same fixed length, chosen in advance.

- The gravitational-wave signal present in some of these time series is generated by a built-in waveform function of PyCBC [27], an open-source Python software package, developed within the LIGO–Virgo–KAGRA community, that provides the standard tools of gravitational-wave data analysis: waveform generation, matched filtering, and the handling of detector noise and of its spectral density.
- These simulated time series, built to resemble the raw output of an interferometer as closely as possible, are then processed with standard techniques, with the objective of cleaning the raw data so that a signal, if present, can emerge as much as possible against the noise, without distorting the physical information the signal carries. We discuss these techniques in detail later in this chapter (Section 5.1.5).
- For the simulated signal coming from a binary, the two component masses are not fixed to a single pair of values: they are drawn at random, independently of one another, within ranges that we choose in advance and that are wide enough to cover a realistic population of coalescing black holes.
- The noise into which these signals are injected is not flat, featureless white noise, but *coloured* noise, whose power is spread across frequencies the way a real interferometer’s is, following the design sensitivity of the LIGO detector.

We now go through this construction step by step: we comment a handful of short, selected extracts of code, chosen to show what is noteworthy, rather than reproducing the underlying notebook line by line. The complete code for dataset generation is given in Appendix D.

5.1.1 Generating and tapering the waveform

The actual gravitational-wave signal is generated using a standard waveform approximant of the PyCBC library [27]: a model, of the same family already discussed in Chapter 4, that predicts the strain produced by a coalescing binary with a certain pair of masses:

```

1 hp, _ = get_td_waveform(
2     approximant=cfg.approximant,      # a waveform model, as in Ch. 4
3     mass1=params["mass1"],
4     mass2=params["mass2"],
5     delta_t=cfg.dt,                  # time between samples, i.e. 1/(sampling rate)
6     f_lower=cfg.f_gen,               # lowest frequency the waveform is generated from
7 )

```

For every signal we generate, the two component masses `mass1` and `mass2` are not read off a fixed grid, but sampled at random. Concretely:

- Rather than drawing the two masses directly, we first draw two other, more physically intuitive quantities: the **total mass** of the binary (the sum of the two component masses) and the **mass ratio** (how many times heavier the larger mass is than the smaller one). Each of these two quantities is drawn independently, uniformly at random within a range we fix in advance.
- Having drawn a total mass and a mass ratio, simple algebra converts this pair back into the two component masses `mass1` and `mass2` themselves.
- The binary’s coalescence phase — the orbital phase at the moment of merger, not shown among the arguments above — is likewise drawn uniformly at random, rather than fixed to a single value.

The effect is a dataset that never sees the same source twice, spread smoothly over the whole mass range we have chosen to cover, rather than a small, fixed catalogue of signals repeated with different noise.

The output `hp` is simply a time series: the plus-polarised strain $h(t)$ the source would produce, sampled at the same rate as everything else in the pipeline. This is exactly what the argument `delta_t` above fixes: it is the time gap between one sample and the next, so the sampling rate — how many samples are recorded every second — is simply its reciprocal, $1/\text{delta_t}$. In Figure 5.1 there is an example of a waveform `hp` generated by this function, `get_td_waveform` of PyCBC.

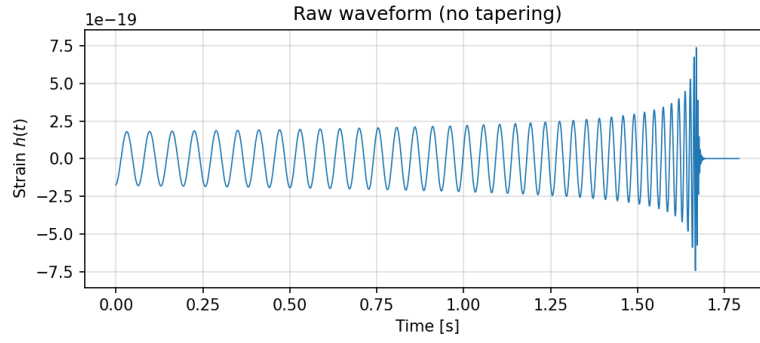


Figure 5.1: An example gravitational-wave waveform generated with PyCBC’s `get_td_waveform`, before tapering, with component masses $m_1 = 35 M_\odot$, $m_2 = 30 M_\odot$, sampled at $1/\text{delta_t} = 2048$ Hz and generated from a lower frequency $f_{\text{lower}} = 20$ Hz: the frequency at which the waveform model starts evaluating the signal, chosen low enough that nothing of astrophysical interest is cut off from the start of the time series.

As Figure 5.1 shows, a raw waveform like this starts and ends abruptly, at whatever amplitude the model happens to reach at the edges of the requested time window; left alone, such a sharp edge behaves, in frequency space, like a sudden switching-on and switching-off, and it contaminates the whole spectrum with spurious high-frequency content that has nothing to do with the astrophysical signal itself (a simple worked example of this effect is given in Appendix A.42). To avoid this we *taper* the waveform: we multiply its start and end by a short, smooth ramp that rises from (and falls back to) zero, so that the signal is eased in and out rather than switched on and off:

```
1 x = np.arange(n_start)
2 h[:n_start] *= 0.5 * (1.0 - np.cos(np.pi * x / (n_start - 1))) # ramp up
```

This single line applies a half-cosine ramp over the first `n_start` samples of the waveform (and a mirrored ramp, not shown, over the last few samples): at the very first sample the multiplying factor is 0, it climbs smoothly through the ramp, and by its end it reaches 1, leaving the rest of the waveform untouched. In Figure 5.2 we see the same waveform of Figure 5.1, this time with tapering applied: the edges are now visibly smoothed rather than abrupt as before.

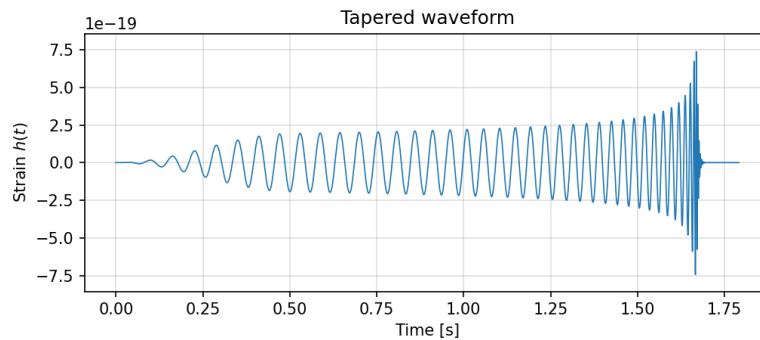


Figure 5.2: The waveform of Figure 5.1 after tapering, generated with the same parameters: component masses $m_1 = 35 M_\odot$, $m_2 = 30 M_\odot$, sampled at $1/\text{delta_t} = 2048$ Hz, lower frequency $f_{\text{lower}} = 20$ Hz.

The effect is purely cosmetic on the time series itself, but is necessary for the correct behaviour of the matched filtering (Section 5.1.5).

5.1.2 Generating coloured detector noise

Recall from Section 4.3.3 that the detector noise $n(t)$ is treated as a stationary random process, and that its noise spectral density $S_n(f)$ (4.29) is precisely what says how much power that noise carries at each frequency — large where the detector is noisy, small where it is quiet. The simulated noise of the interferometer is generated from an analytic model of this same $S_n(f)$, the Advanced LIGO noise spectral density.

We first build $S_n(f)$ itself, evaluated at the frequencies the noise realisation will need¹:

```
1 psd = aLIGOZeroDetHighPower(n_samples // 2 + 1, delta_f, cfg.f_low)
```

`aLIGOZeroDetHighPower` is simply PyCBC’s built-in analytic model of one specific, standard operating configuration of Advanced LIGO and it returns exactly the noise spectral density $S_n(f)$ of Section 4.3.3. Figure 5.3 shows the shape of this particular $S_n(f)$.

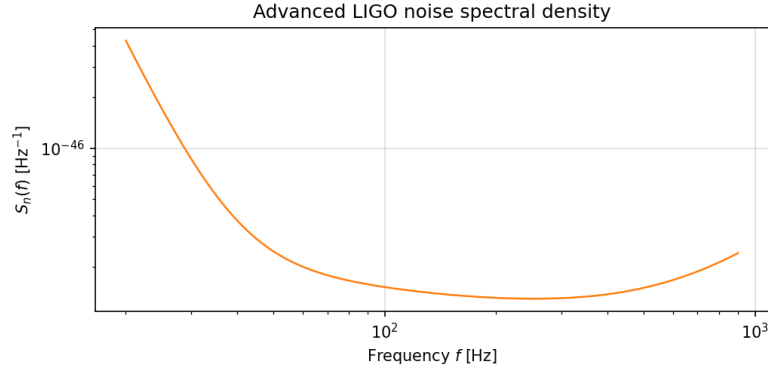


Figure 5.3: The Advanced LIGO noise spectral density used to generate the coloured noise of this section.

Recall also from Section 4.3.3 that the noise $n(t)$ is not one fixed, known signal, but a *random* process: $S_n(f)$ only fixes its statistics (how much power it carries, on average, at each frequency), not any one particular realisation of it. To actually generate a noise time series we therefore need to draw one such realisation, consistent with the statistics fixed by $S_n(f)$ above.²

```
1 noise_ts = noise_from_psd(length=n_samples, delta_t=cfg.dt, psd=psd)
```

`noise_from_psd` is PyCBC’s function for exactly this: given a target power spectral density (here, `psd`, computed above), the length and sample spacing the output should have, it draws one concrete, random noise time series whose statistics follow that PSD.

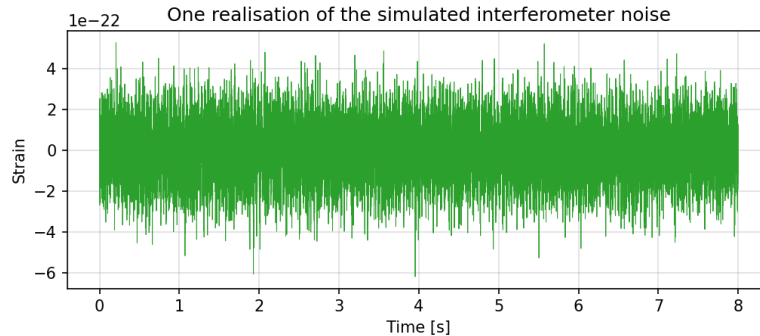


Figure 5.4: An example realisation of the simulated interferometer noise.

This is exactly the *coloured* noise already mentioned above: quiet where the detector is quiet, loud where the detector is loud, following the same $S_n(f)$ that Chapter 4 built the entire matched-filtering machinery around.

5.1.3 Placing the signal at a random merger time

With a realistic noise time series already in hand, we can now decide whether, and where, to add a gravitational-wave signal to it. Concretely:

¹The function used, `aLIGOZeroDetHighPower`, takes three arguments: the number of frequency samples the output should have, the spacing `delta_f` between one frequency sample and the next, and a low-frequency cutoff below which the model is not valid. See [27] for further details.

²`noise_from_psd` takes four arguments: the number of time samples to generate (`length`), the time spacing `delta_t` between one sample and the next and the target power spectral density `psd` itself.

- The tapered waveform of Section 5.1.1 is then added to the noise generated above in only about half of the cases (see Figure 5.5).
- The other half of the dataset is left as noise on its own, with no injected signal at all.

When a signal *is* added, its merger — the instant of peak amplitude — is placed at a randomly chosen position within the timeseries, rather than always sitting at the same fixed instant. This mirrors the fact that, in a real search, the arrival time of a signal is not known in advance; a machine learning classifier trained on signals that always merge at the same instant would learn to exploit that artefact instead of learning to recognise the signal itself. Figure 5.5 illustrates this with two independent draws of the same procedure, in which the same gravitational-wave signal ends up merging at two different random instants.

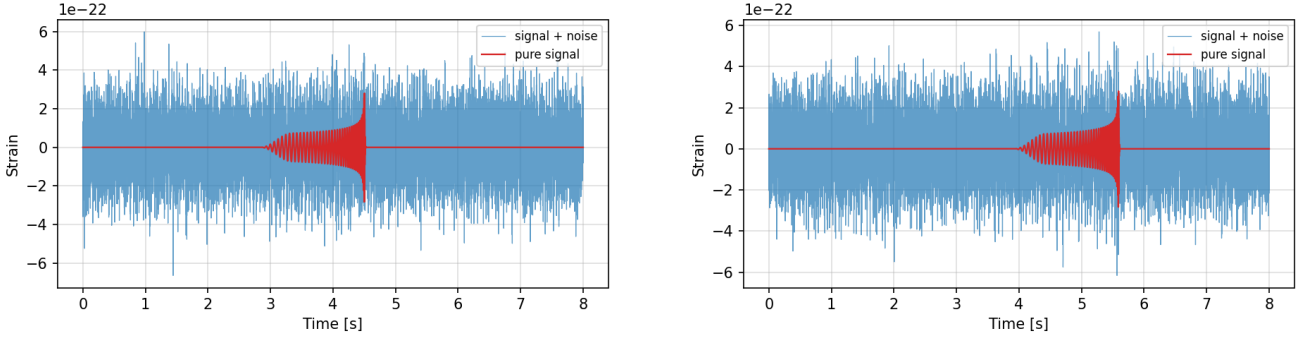


Figure 5.5: Two independent draws of the merger-time placement of Section 5.1.3, showing the same gravitational-wave signal merging at two different random instants.

5.1.4 Rescaling the signal, not the noise, to a target SNR

Before rescaling anything, it is worth recalling briefly what the SNR — the signal-to-noise ratio — actually is: a single number comparing the strength of a signal to the level of the noise it is buried in, large when the signal stands out clearly and small when it is barely distinguishable from the noise around it. Section 4.4.3 made this precise, defining the *optimal* SNR — the best value achievable by any linear filter — as

$$\rho_{\text{opt}}^2 = 4 \int_0^\infty df \frac{|\tilde{h}(f)|^2}{S_n(f)},$$

exactly (4.44): an integral that weighs the waveform’s own spectrum $|\tilde{h}(f)|^2$ against the noise spectral density $S_n(f)$ of Section 5.1.2, frequency by frequency. Evaluating this integral exactly requires knowing the true waveform $\tilde{h}(f)$ itself, which is precisely what a real search does *not* have (hence the whole apparatus of template banks, Section 4.6.4, approximating an unknown true signal). Here, by contrast, we generated the waveform ourselves with `get_td_waveform` (Section 5.1.1), so its analytic form is known exactly, and (4.44) can be evaluated without approximation. We first compute this quantity for the freshly generated, still unscaled, signal:

```
1 rho0 = sigma(signal_ts, psd=psd,
2               low_frequency_cutoff=cfg.f_low,
3               high_frequency_cutoff=cfg.f_high_safe) # unscaled optimal SNR
```

`sigma` is PyCBC’s implementation of exactly the optimal-SNR formula (4.44), $\rho_{\text{opt}} = (h|h)^{1/2} = \left[4 \int_0^\infty df \frac{|\tilde{h}(f)|^2}{S_n(f)} \right]^{1/2}$, evaluated against the same coloured noise spectrum generated above.³

Knowing `rho0`, the (unscaled) optimal SNR the signal would already have, we can rescale the *signal itself* by whatever factor is needed to reach a target SNR value: if we multiply the waveform by a factor $A = \text{target_snr}/\text{rho0}$, we get that the optimal SNR of the rescaled waveform is precisely the SNR we want, `target\_snr`. This can be seen directly from (4.44); the proof is given in Appendix A.43.

³PyCBC’s own name for this quantity, `sigma`, is the norm $(h|h)^{1/2}$ itself: the same normalisation factor that appears in the matched-filter statistic (4.38), here computed for the signal against the noise alone (rather than for a template against real data), which is exactly the unscaled optimal SNR ρ_{opt} of the freshly generated signal. See [27] for further details.

```

1 A = target_snr / rho0
2 scaled_signal_buffer = A * signal_buffer    # only the SIGNAL is rescaled

```

Only the signal amplitude is multiplied by A ; the noise generated in Section 5.1.2 is added to it afterwards, completely untouched. This is an improved approach compared to [24] where the noise amplitude was scaled down to reach a target SNR while the signal was left fixed: here the detector’s noise level stays exactly as generated, and the SNR is instead set by the amplitude of the injected signal, exactly as it would be set, in reality, by the distance and masses of the source.

Now that we have this machinery to rescale a waveform to any SNR we want, we can use it to insert, into the dataset described at the start of this section, signals whose SNR is not fixed to a single value but sampled at random from an interval we choose in advance.

5.1.5 Whitening and band-passing

Before any of this data is used further, it is *whitened*. The idea is best understood by recalling what the noise spectral density $S_n(f)$ (Section 4.3.3) actually says: at some frequencies the detector is quiet, and at others it is noisy, so the noise has a power spectrum that is spread *unevenly* across frequencies — exactly the “colour” of the coloured noise generated above. Whitening simply undoes this unevenness: working in the frequency domain, we divide the data by (the square root of) $S_n(f)$ itself, frequency by frequency, so that every frequency ends up contributing the same amount of noise power as every other. What is left, after this rescaling, is noise whose spectrum is flat⁴ — *white* noise. Before whitening, a signal is partly masked wherever it overlaps a frequency at which the detector happens to be noisiest; once the noise floor is the same at every frequency any genuine signal can stand out uniformly against the flat background [27].

Before dividing by $S_n^{1/2}(f)$, we first apply one further, more technical step to it:

```

1 psd = inverse_spectrum_truncation(psd)

```

`inverse_spectrum_truncation` does exactly this: it shortens the inverse of $S_n^{1/2}(f)$ down to at most `max_filter_len` samples in the time domain, and hands back a correspondingly smoothed version of $S_n(f)$ itself. We explain precisely what this function does, and why it is needed, in Appendix A.44.

Concretely, this division is performed with a single line of PyCBC code:

```

1 whitened = (ts.to_frequencieseries() / psd ** 0.5).to_timeseries()

```

`ts.to_frequencieseries()` moves the data into the frequency domain. There, dividing by `psd ** 0.5` — i.e. by the smoothed $S_n^{1/2}(f)$ obtained above — performs exactly the rescaling just described. The result is then moved back into the time domain with `.to_timeseries()`.

After whitening, the data is further restricted to the frequency band the detector is actually sensitive in, by applying a high-pass and a low-pass filter in turn — discarding the very low and very high frequencies where the detector contributes essentially only noise, as Figure 5.7 shows.

⁴For a formal definition of whitening, see the PyCBC documentation [27].

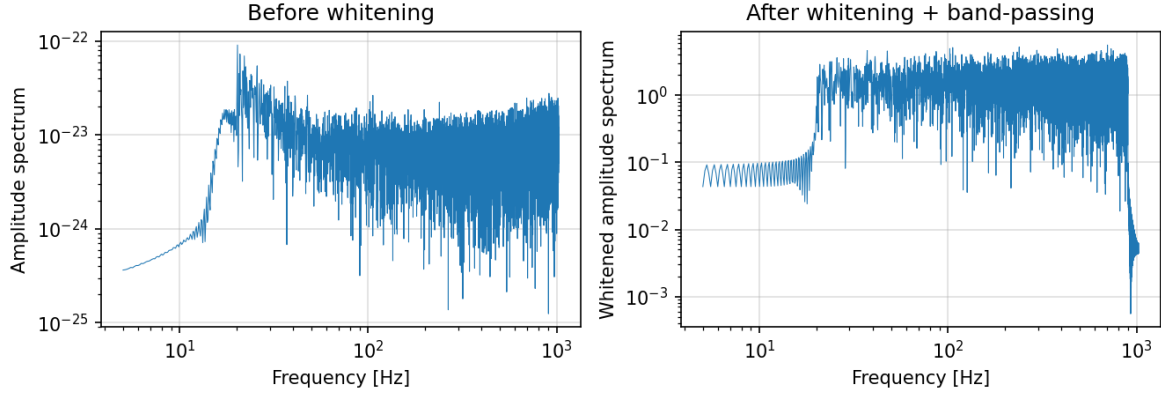


Figure 5.6: The effect of whitening on a signal-plus-noise sample, shown in the frequency domain (amplitude spectrum).

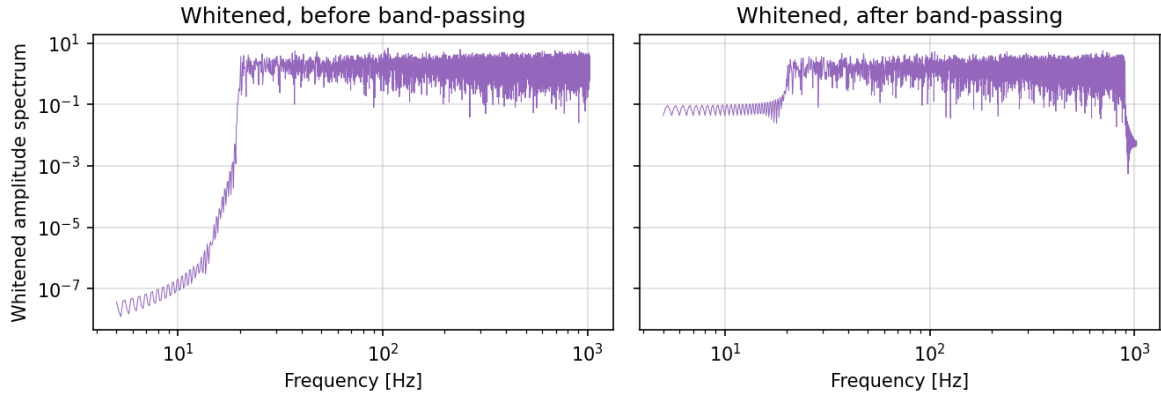


Figure 5.7: The effect of band-passing (high-pass and low-pass filtering) on the whitened data of Figure 5.6.

5.1.6 Summary of the resulting dataset

Repeating the whole procedure above many times builds up the dataset we set out to construct at the start of this section. Briefly:

- The dataset is composed of a number N of time series, half of which contain a gravitational-wave signal (buried in coloured noise) and half of which are noise on its own, with no injected signal at all — the two classes a classifier has to learn to tell apart.
- Each gravitational-wave signal comes from a binary whose total mass and mass ratio are drawn uniformly at random from an interval we choose in advance (Section 5.1.1), rather than from a fixed, repeated catalogue of sources. Also this dataset feature is an improvement over [24].
- Each signal is also given its own random target SNR, drawn uniformly at random from an interval we choose in advance (Section 5.1.4), by rescaling the signal amplitude itself rather than the noise level.
- Every example is built from coloured noise following the actual Advanced LIGO noise spectral density (Section 5.1.2). Also this dataset feature is a phenomenological improvement over [24].
- When a signal is present, its merger time is placed at a random instant within the time series, rather than always at the same fixed position (Section 5.1.3).
- Every time series, whether it contains a signal or not, is whitened and band-passed before use (Section 5.1.5), so that the dataset is ready to be fed into the Takens-embedding and persistent-homology pipeline developed earlier in this thesis (Section 2.3.1).

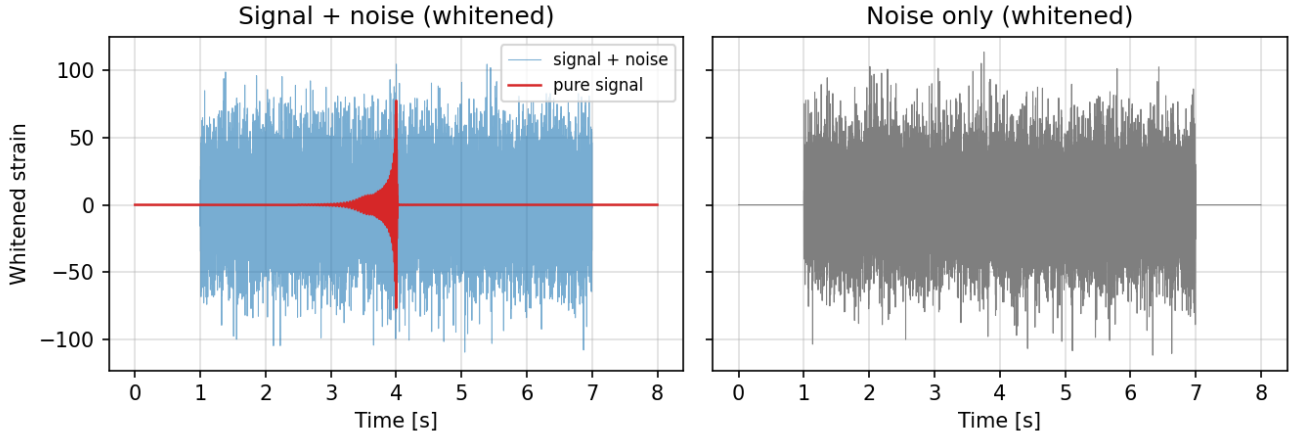


Figure 5.8: A representative signal-plus-noise sample and a representative noise-only sample from the final dataset.

Once built, the whole dataset — every time series together with its label, its target SNR, and the source parameters it was generated with — is saved to disk in a single file, so that it only has to be generated once and can then be reused across the rest of this chapter.

5.2 Running the topological pipeline on a single sample

The dataset of Section 5.1.6 is now ready to be analysed, but before let’s look visually the topological features of a GW. This is the same exercise carried out in Chapter 3 on a cosine, and it is repeated here for one honest reason: a chirp buried in coloured detector noise is a far less simple object than a cosine buried in white noise.

The pipeline is unchanged from Chapter 3, so we go through it more quickly here and point back rather than re-explaining. We will discuss about: Takens embedding (Section 2.3.1) to turn the time series into a point cloud, a Rips filtration (Section 2.1.2) on that cloud, its persistent homology (Section 3.3), and finally the scalar summaries, Betti curves and persistence images of Section 3.4.

5.2.1 The three signals

We fix one example and keep it fixed for the whole section, so that every figure below refers to the same three time series. The binary has component masses $35 M_{\odot}$ and $30 M_{\odot}$, and the injected signal is rescaled to an optimal SNR of 40 by the procedure of Section 5.1.4. This SNR is deliberately high — far above the threshold at which a real search would call a detection — because the point of this section is pedagogical: we want whatever topological difference exists between signal and noise to be as visible as it can possibly be.

The three signals we compare are exactly the three of Chapter 3, transposed to the gravitational-wave setting:

- the **pure chirp**, the injected waveform on its own, with no noise added at all;
- the **pure noise**, one realisation of the coloured Advanced LIGO noise of Section 5.1.2, with no signal in it;
- the **chirp + noise**, the sum of the two — the only one of the three that a detector could actually produce.

Crucially, all three are put through the *same* whitening and band-passing of Section 5.1.5 before anything topological is done to them, including the pure chirp. Each of the three time series is then cropped to the same 2 seconds around the merger, which at the sampling rate we use (12288 Hz, i.e. 2048×6 Hz) gives 24576 samples per signal.

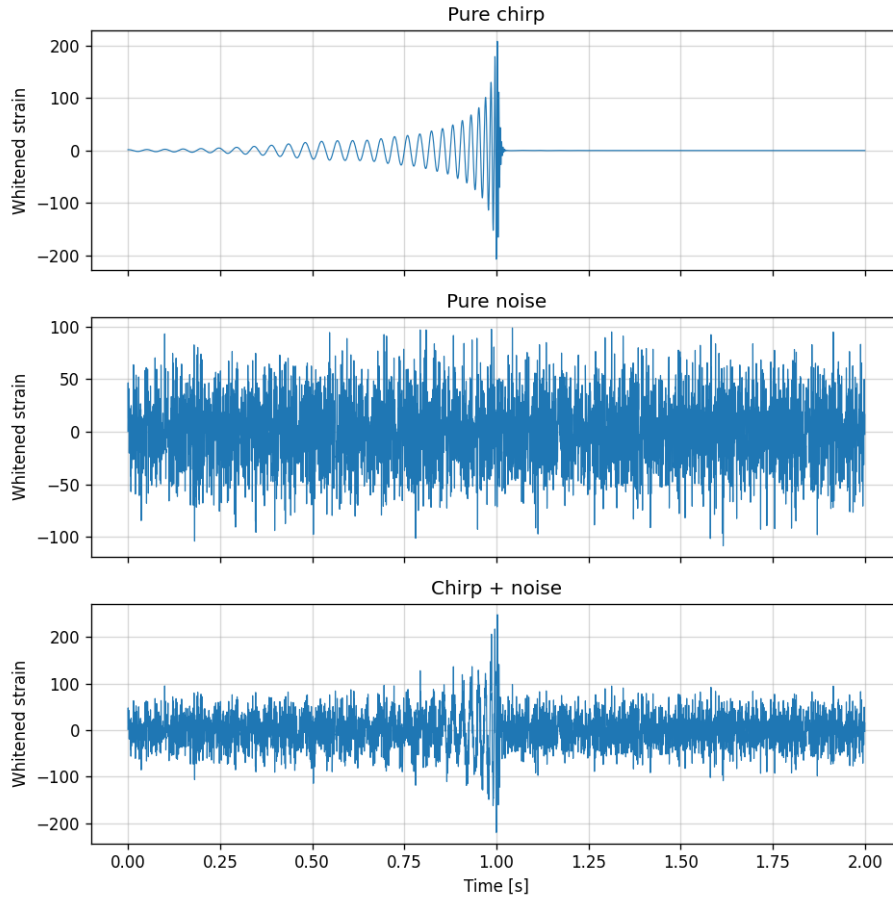


Figure 5.9: The three whitened, band-passed time series used throughout this section: the pure chirp (top), one realisation of coloured detector noise (centre), and their sum (bottom). The merger sits at $t = 1$ s. Even at SNR 40 the chirp is only marginally visible by eye in the bottom panel, and only in the last tenth of a second before merger.

Already Figure 5.9 says something the cosine example could not. The chirp is not a stationary signal: essentially all of its amplitude is concentrated in the final fraction of a second before merger, and it is quite silent before that. Whatever topological structure the chirp produces has to be produced by a small minority of the samples (the samples close to the merger), while the rest of the series — which for the noisy case is essentially just noise — contributes the same shapeless mess as the noise-only signal does. This is the single most important structural difference between this section and Chapter 3, and most of what follows is a consequence of it.

5.2.2 From time series to point cloud

The embedding parameters are found automatically, with the search mode of `SingleTakensEmbedding` already used in Section 3.5, rather than being fixed by hand. Run separately on each of our three signals, the search returns:

Signal	dimension d	time delay τ
Pure chirp	17	103
Pure noise	11	75
Chirp + noise	15	517

As in Chapter 3 we take the median across the three, giving $d = 15$ and $\tau = 103$, so that all three point clouds live in the same space and can be compared at all.

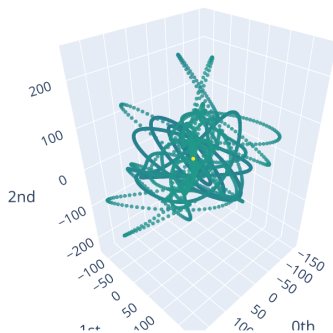
There is a second choice to make, and it is one Chapter 3 never had to face. Recall that the embedding also has a *stride*: having built the delay vectors, we may keep every one of them, or only every s -th one. With 24576 samples and stride 1 the embedding yields 23134 points in $\mathbb{R}^{15}$ — a point cloud far too large for a Rips filtration, whose cost grows steeply with the number of points (Section 3.6).

We therefore use *two* strides for two different purposes:

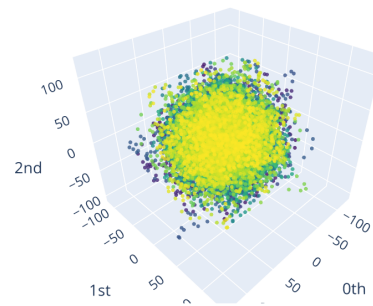
- a **dense** embedding, stride 1, giving 23134 points — used *only* to draw the point clouds below, where we want to see the geometry as sharply as possible;
- a **coarse** embedding, stride 48, giving 1007 points — used for everything else, i.e. every persistent homology computation and every summary derived from it.

Finally, both embeddings are projected down to three dimensions by PCA (Section 2.3.5), so that the Rips filtration is computed in a space where it is computationally affordable; the result is shown in Figure 5.10.

Takens embedding + 3D PCA -- Pure chirp



Takens embedding + 3D PCA -- Pure noise



Takens embedding + 3D PCA -- Chirp + noise

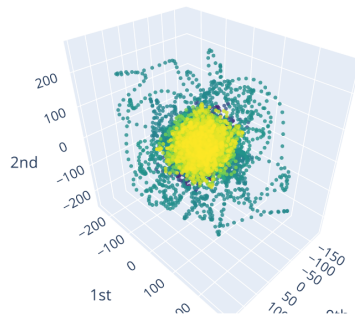


Figure 5.10: Takens embeddings of the three signals, projected to three dimensions by PCA: pure chirp (left), pure noise (centre), chirp + noise (right). The chirp produces a bundle of nested loops of growing radius; the noise produces a structureless ball; their sum produces the ball with a few of the widest loops still visible around its outside.

The **pure chirp** (left) does not give one clean loop. It gives a whole *nest* of loops, of steadily increasing radius, tangled around a dense core. This is exactly what it should give, and follows directly from what a chirp is: each successive cycle of the waveform has a larger amplitude than the last, so each successive turn of the embedded trajectory is a wider loop than the one before. A cosine, whose every cycle is identical, retraces the same closed curve forever and so gives one loop; a chirp never retraces anything, and so gives many. The dense core is the long, quiet inspiral phase at the start of the series, where the amplitude is small and the trajectory stays close to the origin, and the few straight radial spikes shooting outward are the very last cycles before merger, where the amplitude grows so fast between one sample and the next that consecutive points of the cloud are displaced far apart.

This is worth stating plainly, because the introduction to this chapter promised a spiral. The spiral is there — a trajectory whose radius grows as it turns is a spiral — but it is a spiral wound in a

fifteen-dimensional space and then flattened onto three, and what survives that flattening looks like a tangle of nested rings rather than a textbook spiral drawn on paper. The topological content is what matters and it is intact: many loops, of many different sizes.

The **pure noise** (centre) behaves exactly as its white-noise analogue did in the cosine example of Chapter 3 (Section 3.2): a featureless, roughly spherical ball of points with no structure of any kind. Coloured noise, after whitening, embeds in a similar way to white noise.

The **chirp + noise** (right). It is dominated by the noise ball — the dense yellow core — with only the largest few of the chirp’s loops still traceable around its outside, and the smaller ones swallowed entirely. Even at SNR 40, most of the chirp’s topological structure does not survive contact with the noise. Only the loops wide enough to reach outside the ball are still there.

5.2.3 Persistent homology

Persistent homology is computed on the three coarse, PCA-reduced point clouds exactly as in Section 3.3 — same `VietorisRipsPersistence` call, same homology dimensions H_0 , H_1 and H_2 , same edge-collapse speed-up. The only difference is the input.

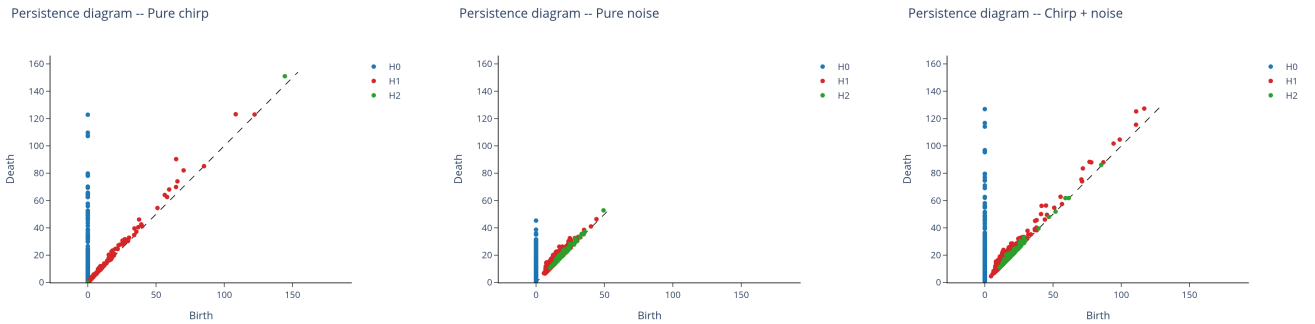


Figure 5.11: Persistence diagrams of the three point clouds of Figure 5.10: pure chirp (left), pure noise (centre), chirp + noise (right). All three panels are drawn on the same axis ranges, so they may be compared directly. The discriminating quantity is not the height of any single point above the diagonal but how far along the filtration the points extend.

Reading the three diagrams of Figure 5.11 side by side:

- **Centre, pure noise.** Everything is compressed into the bottom-left corner. Every feature of every dimension is born and dies below a scale of about 50^5 , and the H_1 and H_2 points lie in a tight band close to the diagonal. This is textbook topological noise, and it is the same picture the white-noise toy model gave: many features, none long-lived, and — the new part — none reaching far along the filtration.
- **Left, pure chirp.** There is no single dominant loop — nothing like the one isolated, far-from-the-diagonal green point that the cosine produced. This is the direct consequence of the nested-loop geometry just described: the chirp’s persistence is *shared* among many loops of moderate lifetime instead of being concentrated in one. What does stand out is the *reach* of the diagram. The blue H_0 points (connected components) climb to a death scale of about 123^6 , and there are red H_1 points born as late as 120: the point cloud is still merging, and still forming loops, at filtration scales far beyond anything the noise reaches. There is also a single green H_2 point, alone at around (145, 151).
- **Right, chirp + noise.** As with the point clouds, this is approximately a superposition of the two cases. The dense diagonal band of the noise is there essentially unchanged, but a scatter of H_1 points now extends out to births of about 115 and deaths of about 127, well past the noise’s cutoff at 50, and the H_0 points climb correspondingly higher; the diagram has *stretched* to larger scales.

⁵More precisely at filtration scale $t=50$, remember that the filtration scale for Rips complexes is the radius of the growing balls around the points.

⁶More precisely at filtration scale $t=123$, remember that the filtration scale for Rips complexes is the radius of the growing balls around the points.

A number of these red points sit reasonably far above the diagonal, rather than hugging it the way the noise cluster does: these are the persistent, small loops still visible in the corresponding point cloud of Figure 5.10, surviving inside the noise.

These three panels indicate that the signature to look for in gravitational-wave data is not the one suggested by Chapter 3. There, the relevant question was whether a single, very persistent loop is present. Here, the relevant question is instead how far along the filtration any feature survives at all — a question about the support of the diagram rather than about any individual point in it. A second, complementary signature is also present: a small number of moderately persistent loops, appearing as points reasonably far above the diagonal, as already noted for the chirp + noise diagram above. Both signatures — an extended support, and a handful of loops of non-trivial persistence — are consequences of the same underlying fact, the nested-loop structure of the chirp’s point cloud, and both are used in the topological summaries of the next subsection.

5.2.4 Topological summaries

We now compress each of the three diagrams into the summaries of Section 3.4, computed with the same `FeatureUnion` of persistent entropy, maximum persistence, total persistence and number of points.

Scalar summaries

Chapter 3 reported only the H_1 values, on the grounds that the loop was the only feature of interest and H_2 contained nothing but numerical debris. Here we prefer to report all three homology dimensions H_0 , H_1 , and H_2 for completeness.

Summary	Dim.	Chirp	Noise	Chirp + noise
Persistent entropy	H_0	7.52	9.77	9.50
	H_1	6.13	8.49	8.43
	H_2	0.00	6.24	6.21
Maximum persistence	H_0	122.8	45.3	126.9
	H_1	25.6	9.2	14.5
	H_2	6.6	3.6	4.6
Total persistence	H_0	4441.7	8541.0	11325.1
	H_1	258.1	1059.9	1254.4
	H_2	6.6	119.7	117.8
Number of points	H_0	1006	1006	1006
	H_1	319	514	517
	H_2	10	107	117

Table 5.1: The four scalar summaries of Section 2.4.1, in all three homology dimensions, for the three signals. Only *maximum persistence* separates all three columns; the remaining summaries separate the pure chirp from the other two but place chirp + noise essentially on top of pure noise.

Reading the table, and taking the summaries in order of how useful they turn out to be:

- **Maximum persistence** is the one scalar that differentiates the three signals (pure noise, pure chirp, and chirp + noise): in H_1 , the chirp’s longest-lived loop survives for 25.6, the noise’s for only 9.2, and the mixture’s for 14.5 — comfortably above the noise value. In H_0 the separation of noise from the other two is starker still (45.3 against 122.8 and 126.9), which is the numerical form of the observation already made about Figure 5.11: noise stops early, and anything containing a chirp does not. That maximum persistence works in differentiating the three signals, while the other summaries struggle and this is not a coincidence. It is the only one of the four that measures how *far* the diagram reaches rather than how its persistence is distributed, and reach is exactly what the chirp adds (Section 5.2.3).

- **Persistent entropy** fails in differentiating the three signals. It separates the pure chirp cleanly from the other two — 6.13 against 8.49 and 8.43 in H_1 , and a striking 0.00 against 6.24 and 6.21 in H_2 — but it cannot differentiate between pure noise and chirp + noise at all: 8.49 against 8.43 is a difference of well under one percent. The reason follows from Definition 2.16: entropy does not measure how large the total persistence is, only how evenly it is shared out among features. Adding the chirp in the time series adds only a handful of long-lived features to the several hundred short-lived noisy ones already present, so among hundreds of features the extra few barely change the overall balance: the evenness that entropy measures is almost the same with or without them. Entropy is doing what it is defined to do; it is simply not measuring the thing that distinguishes these two signals. (The H_2 entropy of 0.00 for the pure chirp is a different matter, and a slightly deceptive one: it is 0 because there is essentially only one H_2 feature, and entropy of a single feature is zero by construction, not because that feature is meaningful.)
- **Total persistence** and the **number of points** behave the same way as entropy, and for a related reason. Both are dominated by the sheer count of short-lived noisy features, of which chirp + noise (517 loops) has essentially as many as pure noise (514). Both are in fact *anti*-informative if read naively: the pure chirp has the *lowest* total persistence and the *fewest* features of the three, so a reader expecting "more signal means more topology" would draw exactly the wrong conclusion. What the chirp does is not add features; it adds a small number of *long* ones and leaves the count almost unchanged.
- The H_0 **number of points** is 1006 for all three signals, which is no information at all — it simply counts the points of the cloud, and all three clouds have the same size by construction. It is listed for completeness.

Betti curves

The Betti curves make graphically the point that maximum persistence made numerically. Recall from Section 2.4.3 that the Betti curve of a diagram just plots, against the filtration scale t , how many features of each dimension are alive at that scale — so the horizontal extent of the curve is exactly the "reach" of the diagram we have been tracking. They are plotted here on a logarithmic vertical axis, so that the small counts at large t remain visible next to the very large counts at small t .

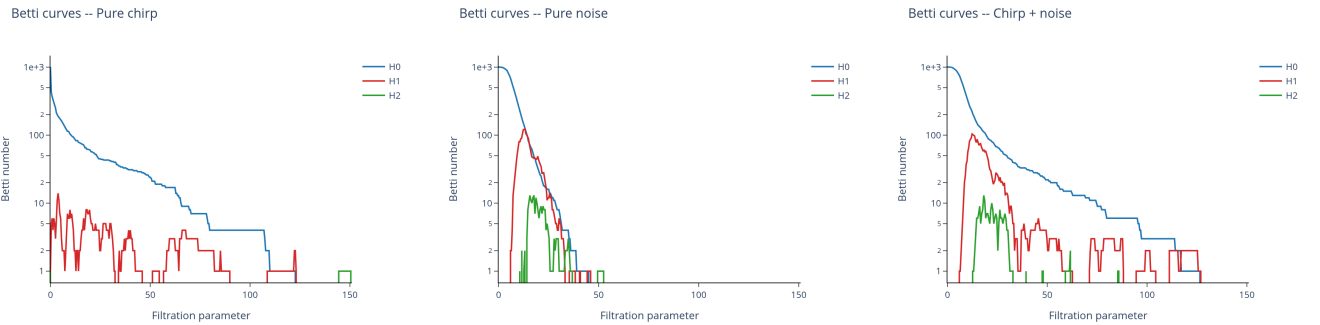


Figure 5.12: Betti curves (β_0 blue, β_1 red, β_2 green, vertical axis logarithmic) for the pure chirp (left), the pure noise (centre) and their sum (right), all on the same horizontal range. The noise curves have collapsed to zero by $t \approx 50$, while both signals containing a chirp still have live features past $t \approx 120$.

The three panels of Figure 5.12 separate the signals more convincingly than any single number in Table 5.1:

- **Pure noise** (centre) produces a compact, self-contained bump. Every curve rises, peaks and returns to zero well before $t = 50$, and past that point the panel is simply empty. Note also the characteristic shape: β_0 stays flat at 1006 for a while before falling, because in a uniformly dense ball of points nothing merges until the filtration radius reaches the typical inter-point spacing, and then everything merges at once.

- **Pure chirp** (left) produces the opposite shape. Its β_0 begins falling immediately from $t = 0$ — there is no characteristic spacing, because the cloud contains both the dense core and the far-flung spikes — and then decays slowly and unevenly all the way out to $t \approx 110$, with visible plateaux where the surviving components are the well-separated spikes visible in the pure-chirp point cloud of Figure 5.10. Its β_1 -curve is small in magnitude but ragged and long-lived, still non-zero out past $t = 120$: these are the nested loops, dying one by one as the filtration radius grows past each ring’s size in turn. The lone β_2 -curve bump at $t \approx 145$ is the single cavity noted in the persistence diagram: at that filtration scale, the balls growing around a handful of points have become large enough to close up around an empty three-dimensional region, so a cavity is born; at a slightly larger scale, the same balls grow far enough to intrude into that region and fill it, so the cavity dies almost immediately after.
- **Chirp + noise** (right) is approximately the superposition of the other two, and this is the most useful panel of the three. Up to $t \approx 40$ it is essentially the noise panel: the same flat-then-falling β_0 , the same tall β_1 peak of order 100, the same β_2 bump. Past $t \approx 50$, where the noise panel is empty, it instead continues in the manner of the chirp panel — a slowly decaying β_0 out to $t \approx 115$ and a ragged β_1 hovering between 1 and 3 out to $t \approx 127$. The signal has not changed the noisy part of the curve; it has added a long, low tail where the noise had nothing.

This is the clearest statement available of what topology contributes here. A detector output containing a chirp and one containing only noise agree closely at small filtration scales and differ completely at large ones, and it is the tail of the Betti curve — not its peak — that carries the signal.

This observation identifies one of the most discriminative features available to us: the *area under the Betti curve*, introduced in Section 2.4.3 (Definition 2.18). Briefly, instead of comparing whole curves, we divide the range of the filtration scale t into a number of equal windows and, in each window, replace the curve by the area underneath it there — that is, by how much of that homological feature was present, on average, during that stretch of the filtration. One curve thus becomes a short list of numbers, one per window.

The reason this feature discriminates so effectively here follows directly from the panels just discussed. The three signals differ not in the height of their Betti curves but in the range of t over which those curves are non-zero, and the windowed areas convert exactly that difference into numbers. In the early windows, where noise and chirp + noise agree closely, the areas are comparable and carry little information. In the late windows — beyond $t \approx 50$, where the pure noise curves have already returned to zero — the area is 0 for pure noise, and strictly positive for both signals containing a chirp. A quantity that vanishes identically for one class and not for the other is precisely what a machine-learning classifier can exploit, and it requires no prior decision on our part about which single scale or which individual feature to examine.

Persistence images

Finally we turn each diagram into a persistence image (Section 2.4.2), with the same quadratic weighting used in Section 3.4, which suppresses short-lived points near the diagonal and amplifies long-lived ones. We show the H_1 images, where the loops live.

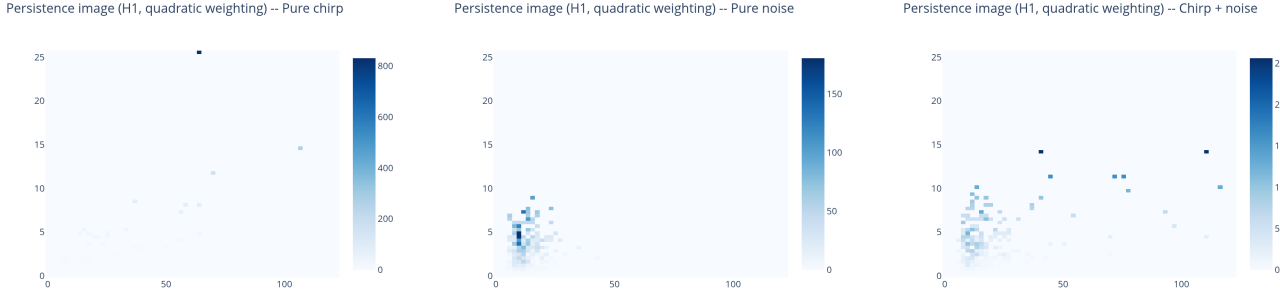


Figure 5.13: Persistence images in dimension H_1 with quadratic weighting, for the pure chirp (left), the pure noise (centre) and their sum (right). The horizontal axis is the birth scale and the vertical axis is the persistence (lifetime) of the feature. **Each panel carries its own colour scale**, so the panels should be compared by *where* the bright cells sit, not by how bright they are.

A caution before reading them: because each panel is normalised to its own maximum, the absolute brightness is not comparable across panels — the brightest cell of the chirp image corresponds to a value roughly three times larger than the brightest cell of the chirp + noise image, even though both appear as the same dark square. What *is* comparable across the three panels of Figure 5.13 is the layout, and the layout is where the difference lies:

- the **pure chirp** image is almost entirely empty, with a handful of isolated bright cells scattered at high persistence (remember that persistence is the y-axis of the images): one prominent cell has a lifetime of about 25, while several smaller ones, with lifetimes between 10 and 15, are scattered across birth scales ranging up to 110 (birth scale is the x-axis). This is the nested-loop structure seen once more: a few well-separated loops, of different sizes, born at different scales;
- the **pure noise** image is the reverse — a single dense clump packed into the bottom-left corner at low birth and low persistence, and nothing anywhere else. All its loops are short-lived and born early, which is precisely what topological noise means;
- the **chirp + noise** image contains both patterns at once and is visually distinct from either: the noise clump in the lower left is there, but so are several bright isolated cells at persistence 10–15 reaching out to birth scales of 110, in a region of the image that is completely empty for pure noise.

This last observation is the one of practical relevance. For the present example, the presence or absence of any bright pixels in the upper-right region of the image separates pure noise from chirp + noise unambiguously. Moreover these images can be supplied to a machine-learning classifier directly.

5.2.5 What this example does, and does not, establish

It is useful to summarize what this example has shown, since it motivates the analysis that follows. Pure detector noise and detector noise containing a chirp exhibit a clear topological difference. We observe this difference in three complementary ways: the persistence diagrams extend to larger scales, the Betti curves develop a longer tail, and the persistence images acquire weight in regions that remain empty for pure noise. These results show that a chirp leaves a detectable topological signature, and that the pipeline introduced in Chapter 3 is sensitive to it.

The chirp’s signature is more distributed than the one observed in the cosine example. The cosine produces one dominant persistent loop, whereas the chirp generates many loops with different persistence values. Its topological information is therefore spread over a broader set of features, rather than concentrated in a single conspicuous structure.

The scalar summaries confirm this picture. Maximum persistence provides a clear separation between pure noise and chirp + noise, while persistent entropy, total persistence and the number of persistence points remain much closer between the two classes. This indicates that the discriminating information is carried by a *combination* of topological observables, rather than by any one scalar quantity alone. For this reason, the present analysis uses the full topological feature set, combining information from different homology dimensions, scalar summaries, Betti curves, and persistence images.

This broader representation makes it possible to identify the topological structure associated with a gravitational-wave signal more effectively. In the present example, at SNR 40, the signal produces systematic changes across several topological descriptors; these changes provide the basis for distinguishing chirp + noise data from pure detector noise.

The appropriate next step is therefore to compute the complete feature vector for every sample in the dataset introduced in Section 5.1. A classifier can then learn, directly from many examples, which combinations of topological features provide the strongest discrimination between noise and gravitational-wave signals. In this way, the detection strategy uses the collective information contained in the topology, rather than relying on a single predefined observable. This is the approach developed in the remainder of this chapter.

5.3 Extracting topological features from the whole dataset

In the previous section, Section 5.2, we followed a single example through the whole pipeline by hand. We now run that same pipeline over the entire dataset built in Section 5.1: each of its thousands of time series is turned into a point cloud, then into a persistence diagram, and finally into the topological summaries we have already defined.

Nothing about the pipeline itself changes, so we do not describe it again. We will discuss only new things, like: choosing a single embedding that works for every time series, computing all the homology in one go for every time series, putting the diagrams on a common scale so that their summaries can be compared, and collecting the results into one table. That table — one row of numbers per time series — is the input the machine-learning classifiers of the next sections will work with.

5.3.1 One embedding for the whole dataset

Every point cloud has to be built with the same embedding dimension and the same time delay, so that we can compare those point clouds: embedding two time series with different values would place their point clouds in different spaces, making the persistence diagrams obtained from them impossible to compare in any meaningful way. One pair of values must therefore be chosen once and then used for the whole dataset.

For the three time series of Section 5.2 this was easy: we ran the automatic search (Section 2.3.4) on each of them and took the median. On thousands of time series that is no longer possible, because the search is itself a costly computation, and running it once per time series would take a considerable amount of time.

The way out is to run the search on only a part of the dataset — in practice half of it — taking equally many time series from each of the two classes (recall from Section 5.1 that class 1 is a gravitational-wave signal buried in noise, and class 0 is noise on its own). Using only a part is what makes the search affordable. Taking equally many from each class matters because signals buried in noise and pure noise have different geometry: if most of the sampled series happened to be noise, the parameters would be tuned to detect mostly noise and would suit the signal-bearing time series in a worse way.

The search is then run once on each sampled time series, and the values of embedding dimension d and time delay τ it returns are collected with the following code snippet:

```

1  for idx in search_indices:                                # the sampled time series
2      ste = SingleTakensEmbedding(
3          parameters_type="search",                        # look for the parameters, don't fix them
4          ...,                                              # other arguments as in the main
5              embedding_call
6      )
7      ste.fit(np.array(list_of_timeseries[idx]))
8      found_dimensions.append(ste.dimension_)               # winning dimension
9      found_time_delays.append(ste.time_delay_)             # winning time delay
10
11 best_embedding_dimension = int(np.median(found_dimensions))
12 best_time_delay          = int(np.median(found_time_delays))

```

The argument `parameters_type="search"` is what turns the transformer⁷ `SingleTakensEmbedding` from one that *uses* a dimension and a delay we give it into one that *looks for* them itself.⁸ Figure 5.14 shows the resulting distribution of the optimal embedding dimension d and time delay τ found for each sampled time series. To pick the single value of d and of τ used for the whole dataset, we take the median of each histogram⁹.

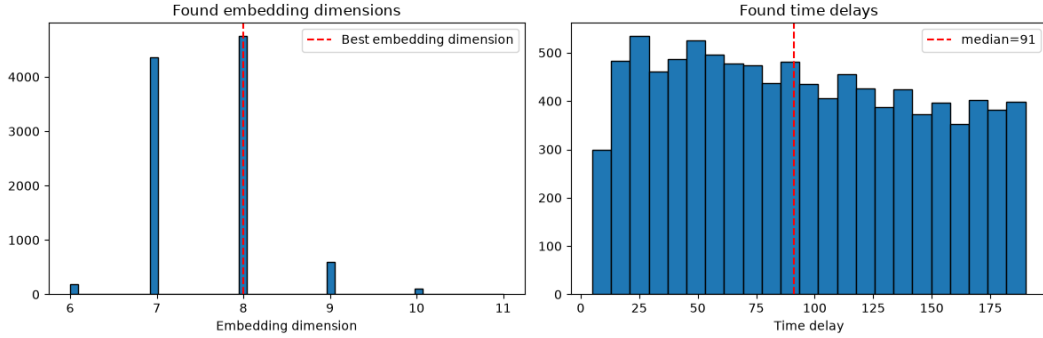


Figure 5.14: How often each embedding dimension (left) and each time delay (right) was chosen by the automatic search, over the sampled time series; the dashed line marks the median that is then used for the whole dataset. The two panels behave very differently: the dimension is concentrated on essentially two values, while the delay is spread almost evenly over the entire range the search was allowed to try.

Figure 5.14 is worth looking at closely:

- The **dimension** (left) is well determined: almost every time series picks one of two neighbouring values, so the median summarises a genuine agreement among thousands of independent time series.
- The **delay**: its distribution is nearly flat over the whole range the search was allowed to explore, with no peak anywhere, which means the sampled time series do not agree on a delay even approximately. We use the median time delay anyway, for the reason given at the start of this subsection: a common embedding is what makes the diagrams comparable in the first place.

5.3.2 Reducing the point clouds

Each point cloud may then be projected onto a small number of principal components (Section 2.3.5) before its homology is computed, for the reason already given in Section 5.2.2: the cost of a Rips filtration grows with both the dimension and the number of points.

It is worth asking what such a projection throws away. Figure 5.15 answers this by showing, for every time series, how much of the variance each principal component accounts for.

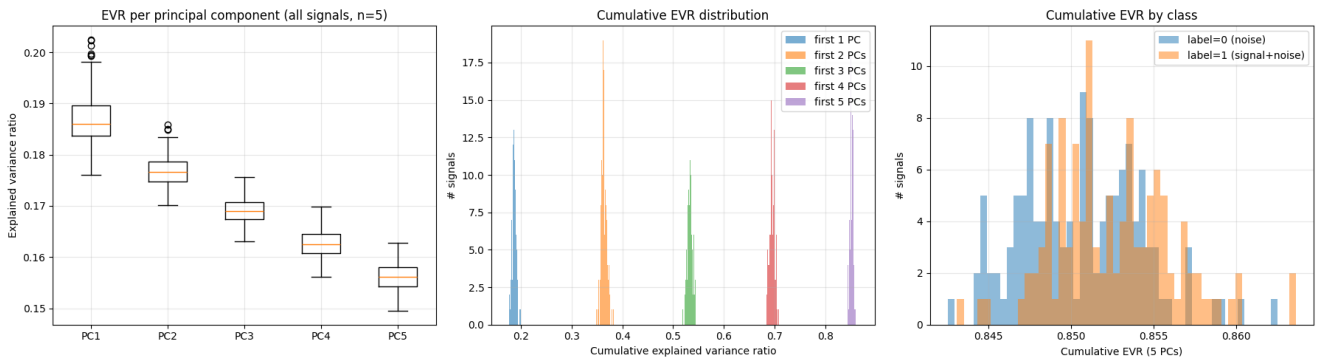


Figure 5.15: How the variance of the embedded point clouds is divided among the principal components. Left: the fraction carried by each component, over all time series. Centre: the fraction carried by the first k components together. Right: the same total, with the two classes plotted separately. The fraction falls only very slowly from one component to the next, and the two classes lie almost on top of each other.

⁷The function `SingleTakensEmbedding` can be called a *transformer* in machine learning field.

⁸The delay is chosen as the first minimum of the mutual information between the series and a delayed copy of itself, and the dimension by the method of false nearest neighbours; both are described in Section 2.3.4.

⁹We prefer not to take the mean since the median is more stable to outliers.

Three things can be read from it.¹⁰

- **The variance is spread almost evenly.** The first component carries about 18.7% of it and the fifth still about 15.6%, decaying only gently in between. This is not what a point cloud with a clear shape looks like: a cloud shaped like a single loop would put most of its variance into the first two or three directions and almost none into the rest. Here no direction is preferred over the others — the cloud is spread out much the same way whichever way we look at it. The projection is therefore a compromise made for speed, not the discovery of some simpler shape hidden in the data, since each direction we throw away carries about as much as each one we keep.

This is exactly the picture we already saw by eye in Figure 5.10. The chirp + noise point cloud was dominated by the noise ball, with only the widest of the chirp’s loops still traceable around its outside. A ball looks the same in every direction, and so divides its variance evenly among all of them; the loops that carry the actual gravitational-wave signal sit on its surface and account for very little of it.

- **It does not separate the two classes.** In the right-hand panel the noise-only and signal-bearing distributions overlap almost completely, the signal one being shifted only very slightly. A chirp does add a little regular structure that the projection notices, but far too little for this quantity to be useful as a distinctive feature of the presence of a GW on its own — which is consistent with what Section 5.2.5 already found.

5.3.3 Persistent homology of the whole collection

The persistent homology of every point cloud is computed by a single call, which gives back one persistence diagram per time series, in the same order as the input. The parameters are the ones already discussed in Section 3.3 and used again in Section 5.2.3, so we do not go through them again:

```
1 persistence = VietorisRipsPersistence(  
2     \dots                # same parameters as in "Persistent homology"  
3     n_jobs=-1,           # use every available CPU core  
4 )  
5 list_PH_diagrams = persistence.fit_transform(list_point_clouds)
```

The one argument worth a comment here is `n_jobs=-1`. The point clouds are completely independent of one another, so there is no reason to treat them one after the other: this argument tells the library to spread them across all the processor cores available, computing several diagrams at the same time. It changes how long the computation takes and nothing else — not one feature of the resulting topology.

5.3.4 Computing the summaries

A common scale for every diagram. Persistence values are lengths measured along the filtration scale, and that scale depends on the overall size of the point cloud it came from. A summary such as the maximum persistence of Definition 2.14 is therefore a number whose magnitude depends on that overall size.

This is the job of `Scaler()`. Its `fit` stage computes a *single* number from the whole collection of diagrams — essentially, the size of the most persistent feature found anywhere in it — and its `transform` stage divides every birth and every death of every diagram by that one number. The word *single* carries the weight here: one factor is computed for the entire collection, *not* one factor per diagram. One global factor changes the units of all the diagrams together and leaves the differences between them untouched¹¹.

¹⁰These come from an earlier, smaller run of the same notebook that computes the persistent homology, in which the search returned an embedding dimension of 6 and the clouds were projected onto 5 components.

¹¹Rescaling each diagram by its own size, instead of by one size shared across the whole collection, would stretch every diagram onto the same range regardless of how far it originally reached. A small-reach noise diagram and a far-reaching signal diagram would then come out looking the same size. That is exactly the difference we want to keep, since Section 5.2.3 found that signal diagrams reach *further* along the filtration than noise diagrams do.

The scalar summaries. The four scalar summaries are then computed exactly as in Section 3.4, bundled into a `FeatureUnion` — the tool, met there, that runs several transformers¹² on the same input and puts their outputs side by side:

```
1 topological_features = FeatureUnion([
2     ("entropy", entropy),
3     ("maxPersistence", compute_max_persistence_transformer),
4     ("totalPersistence", compute_total_persistence_transformer),
5     ("numFeatures", NumberOfPoints(n_jobs=-1)),
6 ])
```

The four objects listed in the `FeatureUnion` compute the four quantities of the same name:

- `entropy` computes the persistent entropy of Definition 2.16;
- `compute_max_persistence_transformer` computes the maximum persistence of Definition 2.14;
- `compute_total_persistence_transformer` computes the total persistence of Definition 2.13;
- `NumberOfPoints` computes the number of points of Definition 2.15.

Each is computed once per homology dimension, so together they give one value per dimension per time series.

- We can compose the steps needed to compute these topological features (persistent entropy, maximum persistence, and so on) into a single pipeline, applying first the scaling and then the summaries just described:

```
1 steps = [
2     ("scaling", Scaler()),           # first: one common scale
3     ("topological_features", topological_features), # then: the
4     ]                                summaries
5 transformer_for_homology = Pipeline(steps)
```

- and then give the command that actually computes these topological features, from the list of persistence diagrams `list_PH_diagrams`:

```
1 features_of_homology = transformer_for_homology.fit_transform(
    list_PH_diagrams)
```

Betti curves and persistence images. The remaining summaries to compute are the Betti curves and the persistence images, with code similar to what was already shown in Section 3.4. As before, both are built from transformers (`BettiCurve` and `PersistenceImage`) applied to the whole list of diagrams *after* the common scaling, for the same reason as the scalar summaries; Table 5.2 summarises the Betti curves and persistence images.

¹²In scikit-learn, a *transformer* is simply an object exposing two methods, `.fit(X)` to learn whatever the transformation needs from the data and `.transform(X)` to apply it; see [62] for the general framework.

Summary	What it is	Why kept
Betti curve (Section 2.4.3, Def. 2.17)	One curve per time series and per homology dimension, sampled on a fixed grid of filtration values	Late windows vanish for noise but not for signals (Section 5.2.4)
Windowed areas (Def. 2.18)	Betti curve reduced to a short list of numbers, area under the Betti curve over each window of the filtration scale ¹³	Same distinction as above, expressed as plain area values rather than a curve
Persistence image (Section 2.4.2)	One image per time series and per homology dimension, with the same quadratic weighting used in Section 5.2.4	Upper-right region empty for noise but not for signals (Section 5.2.4)

Table 5.2: The remaining topological summaries computed for every time series, alongside the scalar summaries of Section 5.3.4.

The resulting table. Everything computed above is finally collected into a single table, one row per time series and saved to disk so that the homological computation — by far the expensive part — is done once and reused afterwards. This dataset of topological features is now ready to be analysed with machine-learning techniques, with the purpose of detecting a gravitational-wave signal immersed in overwhelming noise — the subject of the next chapter.

¹³As in Section 3.4 we do not show the code, since integrating a curve numerically has nothing to do with topology.

Chapter 6

Machine Learning Detection of Gravitational Waves from Topological Features

Chapter 5 turned every time series of the dataset into a persistence diagram, and then into a handful of topological summaries — scalar quantities, Betti curves, persistence images — computed once and saved to disk. This chapter takes that table of summaries and asks whether it is enough to tell a gravitational wave from noise. We first introduce, in a self-contained way and without any topology, the machine-learning classifiers used throughout (Section 6.1); we then train and validate them on the topological summaries (Section 6.2); and we close by asking which features the classifiers actually rely on, how weak a signal can still be found, and how the cost of the whole approach compares with matched filtering (Section 6.3).

6.1 Machine learning classifiers

This section is therefore a self-contained account of the machine-learning tools we use: machine learning (ML) is a tool we use in this work, so we describe what each method does and why we chose it, and leave the formal ML theory behind in the references [21].

6.1.1 Supervised binary classification

The situation we are in is the following. We hold a large collection of example(or samples), each of which has been reduced to a fixed list of measured numbers, and for each sample we also know the answer: whether it belongs to one class or the other. We want a rule that, shown the list of numbers for an example we have never seen tells us which class it belongs to. Learning such a rule from examples that come with their answers attached is called *supervised* learning, and when there are exactly two classes the problem is called *binary classification*.

Binary classification is exactly the problem this thesis has been working towards. Detecting a gravitational wave buried in the detector noise means deciding, for a given stretch of data, which of two classes it belongs to: the one in which a gravitational-wave signal is present, and the one in which it is not.

Making that precise takes very little notation. The examples we learn from is called the *training set*

$$S = ((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)),$$

which is simply our m examples listed one after another, each paired with its answer. Here $\mathbf{x}_i \in \mathbb{R}^n$ is the *feature vector* of the i -th example — the list of n numbers we have chosen to describe it, and the only thing about that example the algorithm will ever see — and $y_i \in \{0, 1\}$ is its *label*, the number that records which of the two classes the example actually belongs to. In our case we fix the convention that $y_i = 1$ marks a stretch of data containing a gravitational-wave signal and $y_i = 0$ one containing only noise. A *classifier* is then nothing more than a function

$$h : \mathbb{R}^n \longrightarrow \{0, 1\},$$

that is, a rule which accepts any list of n numbers and returns one of the two labels. A learning algorithm is a procedure that reads the training set S and returns such a rule h .

In particular, we are interested in how well the machine-learning classifier performs on examples it has never seen. A rule can achieve a perfect score on the training set by memorising it — storing every example together with its answer and returning that answer when it recognises one — and such a rule is worthless, because it has learned nothing that transfers to a new example. This failure is called *overfitting* [21], and it is not a rare pathology but the central difficulty of the field: the more flexible we allow a rule to be, the better it can fit the examples in front of it. The quantity we want to estimate is therefore the error the rule would make on fresh examples.

We use five machine learning classifiers, which fall into two groups according to what kind of input they expect. Section 6.1.2 describes them.

6.1.2 The classifiers

The distinction that organises our choice of methods is what the input to the classifier looks like. In the setting of Section 6.1.1 an example is a feature vector $\mathbf{x}$: a short, fixed-length list of numbers in which *the order of the entries carries no meaning*. The third entry is one measurement and the fourth is a different measurement, and the fact that they were written next to each other says nothing — shuffling the columns of the whole table, consistently for every example, would change nothing about the problem. This is the shape of the table assembled in Section 5.3: one row per time series, each row a fixed list of numbers summarising it. Most classical classifiers are built for exactly this kind of input, and we use three of them.

Some inputs, however, are not like that. If an example is a *curve* sampled on a grid, or an *image*, then neighbouring entries are related to one another, and that adjacency is itself information: a bump occupying three consecutive samples is a feature of the curve. Flattening such an input into a plain list of numbers throws that structure away. Convolutional neural networks are the standard tool designed to exploit this type of input, and we use two of them.

Logistic regression. Figure 6.1 shows the situation this classifier is built for: forty examples, each described by two measured numbers, so that each is a point in the plane of the two features, and each labelled by its class. The two classes occupy different regions but overlap where they meet. Logistic regression looks for the simplest rule available — a single straight line across the plane, one class claimed on either side.

That line is described by a weighted sum. Each feature x_j is given a weight w_j saying how much it counts and in which direction, and the feature vector is collapsed into the single number

$$\langle \mathbf{w}, \mathbf{x} \rangle + b = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b, \quad (6.1)$$

called the *score*, where b is a constant offset letting the line sit away from the origin. The score is positive on one side of the line, negative on the other, and larger the further from it the example lies. Fitting means tuning the weights and the offset until the line falls where it best separates the training examples.¹ The decision boundary is the set of points scoring exactly zero — a straight line for two features, a plane for three, a hyperplane in general — drawn in Figure 6.1(a) together with the weight vector $\mathbf{w}$.² Since the classes overlap, a few points fall on the wrong side; no straight line could have done better.

The score (6.1) can be any real number, positive or negative and of any size, whereas a probability has to lie between 0 and 1. We can compute a probability composing (6.1) with the *sigmoid* function

$$\phi_{\text{sig}}(z) = \frac{1}{1 + \exp(-z)},$$

an S-shaped curve which squeezes the whole real line into the interval $(0, 1)$: it is close to 0 for very negative z , close to 1 for very positive z , and equal to $\frac{1}{2}$ at $z = 0$. Applying it to the score — that is,

¹The weights are chosen to minimise the *logistic loss*, a measure of disagreement between the predicted probabilities and the observed labels. It is a convex function of the weights, so the minimisation has no local minima to be trapped in and standard numerical methods find the optimum reliably. See [21] for the derivation.

²As Figure 6.1(a) shows, $\mathbf{w}$ is perpendicular to the boundary and points in the direction along which the score grows fastest. This is not a coincidence of the example: [21] for further details.

composing it with the weighted sum (6.1) — is the classifier itself,

$$h(\mathbf{x}) = \phi_{\text{sig}}(\langle \mathbf{w}, \mathbf{x} \rangle + b) = \frac{1}{1 + \exp(-(\langle \mathbf{w}, \mathbf{x} \rangle + b))} \in (0, 1), \quad (6.2)$$

a two-step recipe: first reduce the example to its score, then bend that score into a probability in the unit interval. The number $h(\mathbf{x})$ is the model's output for the example $\mathbf{x}$, and it is read as the probability that $\mathbf{x}$ belongs to class 1 — so that $1 - h(\mathbf{x})$ is the probability it belongs to class 0. Since the task demands an actual label rather than a probability, we finally commit to whichever class the model considers more likely, predicting class 1 when $h(\mathbf{x}) > \frac{1}{2}$ and class 0 otherwise. Figure 6.1(b) shows the same forty points arranged by their distance from the boundary, lying along that curve. This sigmoid adds a graded confidence — points far from the boundary receive probabilities near 0 or 1, points near it probabilities near $\frac{1}{2}$.

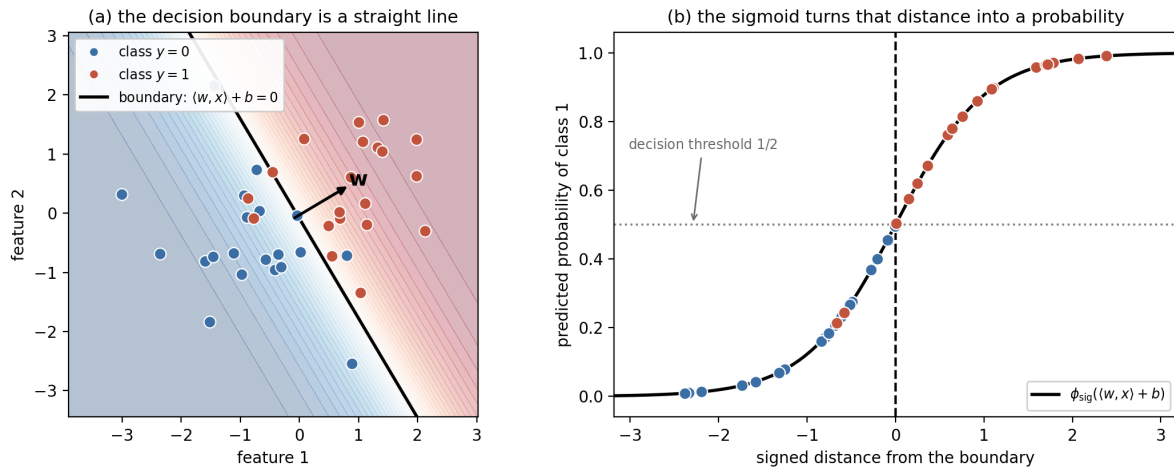


Figure 6.1: Logistic regression on a small two-feature example, forty points in two classes. (a) The fitted decision boundary, the straight line along which the score $\langle \mathbf{w}, \mathbf{x} \rangle + b$ vanishes; the arrow marks the weight vector $\mathbf{w}$, the direction in which the score grows, and the shading is the predicted probability of class 1. Because the two classes overlap, a few points fall on the wrong side of the line — a flat cut cannot do better here. (b) The same forty points placed by their signed distance from that line, the single number the linear part produces. The sigmoid maps that number to a probability, and points lying near the boundary are assigned probabilities near $\frac{1}{2}$.

Because each feature enters the score (6.1) through exactly one weight, the fitted weights can be read off and compared: a large weight moves the score a great deal and so matters to the decision, while a weight near zero is effectively ignored. Figure 6.2(a) shows a five-feature problem in which only two features were built to carry information; the fit gives those two large weights and leaves the other three near zero, recovering by itself which measurements are worth anything.³ No other model we use offers this as directly, and we include logistic regression principally as this interpretable baseline.

The limitation is the flatness. If no flat cut divides the two classes, the method has no utility: Figure 6.2(b) shows one class forming a ring around the other, and of course no straight line separates a disc from the annulus surrounding it. This is the gap the remaining classifiers are meant to fill.

³Two cautions apply. Weights are comparable only when the features they multiply are on comparable scales, which is why the features are put on a common footing before fitting (Section 6.1.3). And when several features are strongly correlated the individual weights stop being interpretable: a large weight on one may be offset by a compensating weight on another, so the fit can distribute the credit among them almost arbitrarily. We meet exactly this situation in the next section we apply ML to the GW case.

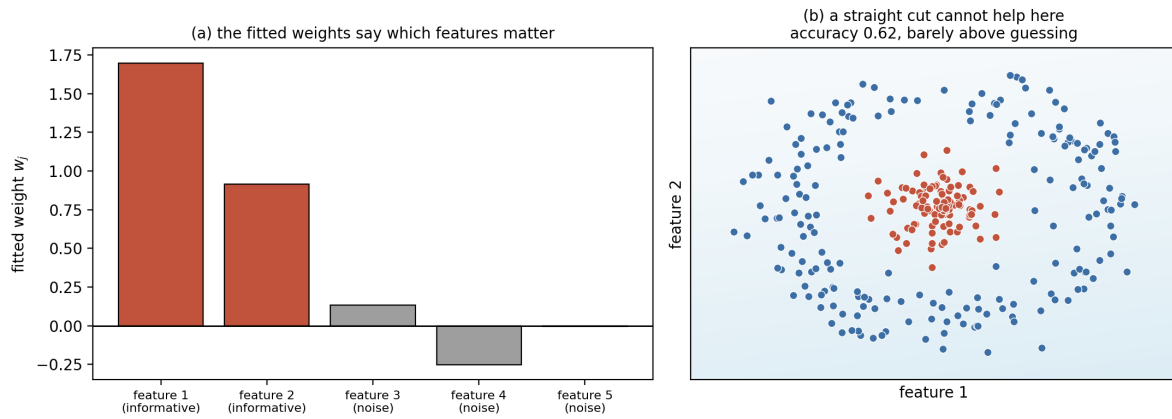


Figure 6.2: (a) Weights fitted on a problem with five features. The two informative features receive by far the largest weights and the three uninformative features have a weight that fall close to zero. (b) Two classes arranged as a disc inside a ring. No straight line can separate them.

Random forest. The second classifier answers the same question with a different shape of the classification rule. Figure 6.6 shows it on ninety points of another two-feature example.. A *decision tree* classifies a point by applying a sequence of tests, each comparing a single feature against a threshold, and following one branch or the other according to the outcome until it reaches a leaf, where a label is recorded. To clarify this the next three figures follow such a tree as it is built, one test at a time, on sixty points of a two-feature example in which class 1 occupies one corner of the plane.

The first test, in Figure 6.3, is $x_2 < -0.01$: it consults the second feature alone and ignores the first, and so divides the plane along a horizontal line, everything below it labelled class 0 and everything above class 1. This is the general pattern — a test on one feature is a cut perpendicular to that feature’s axis — and it already classifies 88% of the points, but it cannot do better, since the upper region still holds a mixture of both classes.

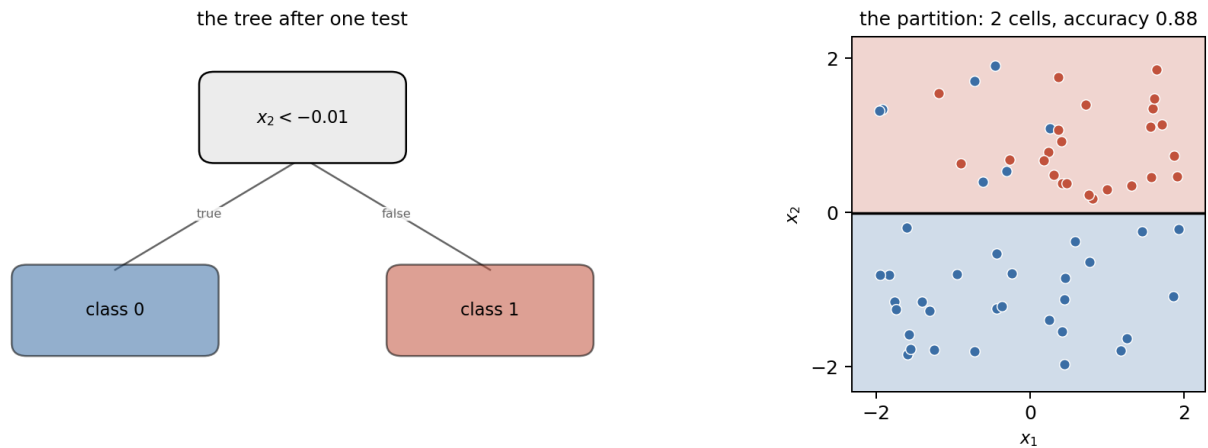


Figure 6.3: The tree after one test, and the partition it produces. The test consults x_2 alone, so it cuts the plane along a horizontal line. Everything below is labelled class 0 and everything above class 1, which is correct for 88% of the sixty points.

The tree therefore splits that region again. The second test divide that region with a vertical line at $x_1 = -0.29$ and isolates the corner where class 1 (red) actually lives. Two tests have produced three rectangular cells, each carrying one label, and the accuracy rises to 95%. This is worth comparing with logistic regression: a corner is bounded by two perpendicular edges, and no single straight line can separate it from the rest of the plane, so the classifier of the previous paragraphs cannot represent this arrangement at all, while two axis-aligned cuts capture it exactly. Every further test cuts one cell in two, so the cells stay rectangular and the boundary is always a staircase of axis-aligned steps. Given enough steps, such a staircase can follow a curve that no straight line approaches.⁴

⁴Which test to apply at each step is chosen greedily, by maximising a measure of how much purer the two resulting

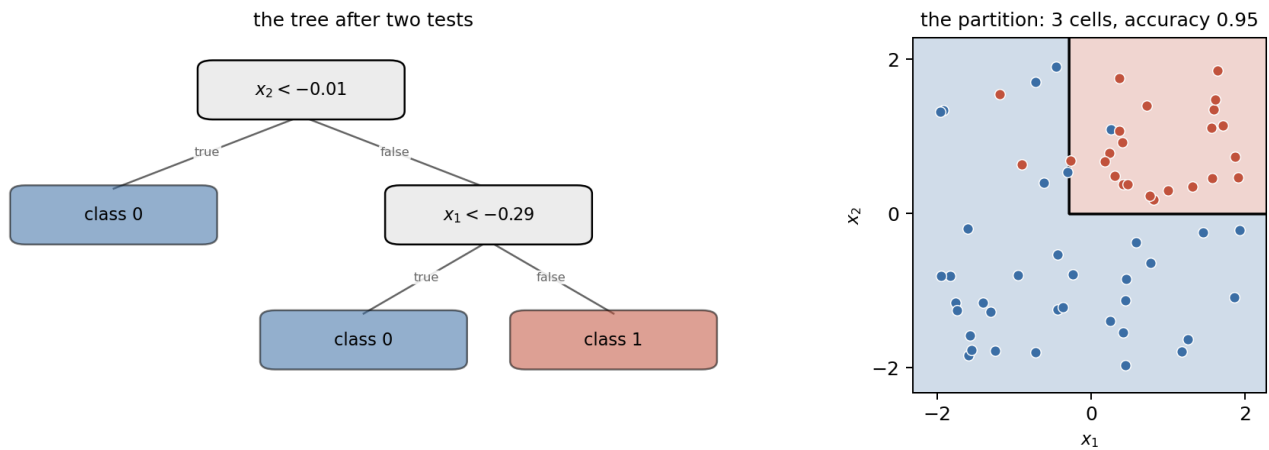


Figure 6.4: The tree after two tests. The second test splits only the upper region, along a vertical line, isolating the corner in which class 1 lies. Three rectangular cells now give 95% accuracy — and no single straight line could have separated this corner from the rest of the plane.

That flexibility carries a risk, shown in Figure 6.5. Left unrestricted, the tree keeps splitting until no cell contains both classes, which here takes seven cells and the tree classifies every training point correctly. For example, in the upper left of Figure 6.5 there is an outlier: a point labelled class 1 (red) sitting in the middle of a cluster of class 0 points (blue). The tree has made a special split for that one point, isolating it in a narrow strip of its own. This is memorising the training set rather than learning the underlying structure of the data, namely the tree is overfitting, exactly the failure named in Section 6.1.1. The 100% accuracy of this stage, Figure 6.5, is worth less than the 95% of the previous stage, Figure 6.4.

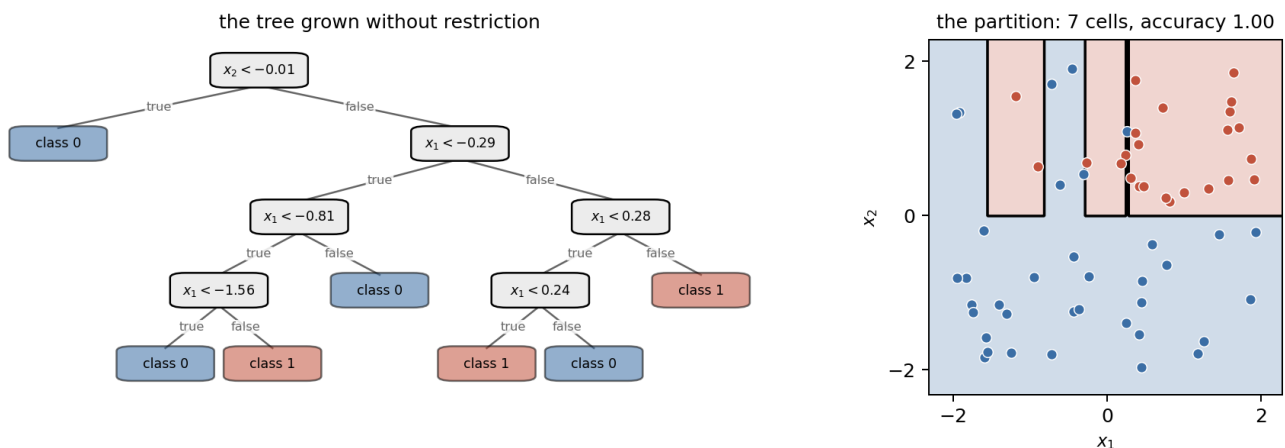


Figure 6.5: The tree grown without restriction. It reaches 100% accuracy with seven cells, but does so by carving a narrow strip around each of the four points whose labels were flipped when the data was made. These strips fit the noise of this sample rather than the structure of the data, and would misclassify fresh points falling near them.

The *random forest* [67] addresses this by combining many trees rather than relying on a single one. Each tree is grown on a random subsample of the training examples and is restricted, at every test, to a random subset of the features, and the forest's prediction is the majority vote of its trees [21].

These two sources of randomness are essential to the construction, since they ensure that the trees differ slightly from one another, and it is worth being precise about what is the voting mechanism of random forests. Every tree in the forest classifies the point in question, on its own, into class 0 or class 1; the forest simply counts how many trees said 1 and divides by the number of trees, giving a number between 0 and 1 that is treated exactly like the probability of Section 6.1.1 — the forest's label is whichever class more than half the trees voted for. Each tree still overfits, but each overfits *differently*, so

groups are than the group being divided; the usual choices are the information gain and the Gini index. See Shalev-Shwartz and Ben-David [21] for both, and for the pruning strategies that limit the fragmentation described below.

a tree that misclassifies a given point is generally outvoted by the trees that do not, and the individual errors are largely cancelled once enough trees are combined [21].

The forest of this example is built from three hundred trees. Each point of the plane can be handed to the forest, which assigns it a class by the vote just described. Repeating this at every point separates the plane into the region the forest labels 0 and the region it labels 1; the border between the two is the forest’s decision boundary, drawn in Figure 6.6(c), and it is considerably smoother than the boundary of any single tree. To see the arithmetic itself, Table 6.1 follows it at six representative points. Three hundred columns cannot be printed, so the table displays the individual votes of only three of the trees — the same three plotted in Figure 6.6(b), which places the boundary in appreciably different positions — while the fraction reported next to them is taken over the full three hundred. The first four rows are precisely the interesting cases in which tree do not agree: in each of them the three displayed trees fail to reach a consensus, or reach one that the remaining trees overturn, and the forest is nonetheless correct at the end, after every tree has voted.

votes of 3 of the 300 trees (those of Figure 6.6(b))			fraction of all 300 trees for class 1	forest verdict	true	
tree 1	tree 2	tree 3			class	
1	0	1	0.23	0	0	✓
0	1	0	0.33	0	0	✓
1	0	0	0.70	1	1	✓
1	1	0	0.85	1	1	✓
0	0	0	0.00	0	0	✓
1	1	1	1.00	1	1	✓

Table 6.1: How the vote resolves disagreement, at six representative points of the example of Figure 6.6. The first three columns give the votes of just three of the forest’s three hundred trees, the same three drawn in panel (b); the fourth gives the fraction of *all* three hundred voting for class 1, which is the forest’s graded output; the fifth is the label that follows by majority. In the top block the three displayed trees never agree unanimously, but the forest is correct each time thanks to the voting mechanism. In the bottom block the fractions of 0.00 and 1.00 show two points on which every tree in the forest agrees.

The cost of this construction is interpretability. A single tree can be read and followed step by step, whereas the combined verdict of three hundred trees cannot.

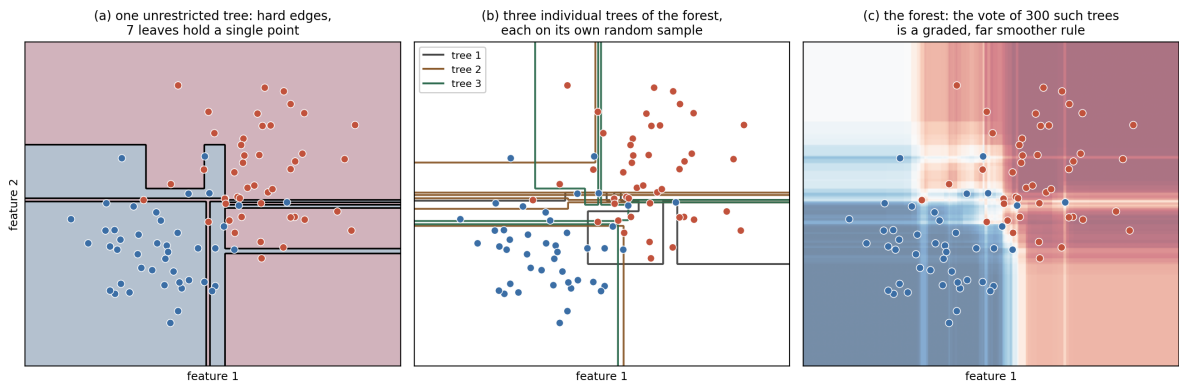


Figure 6.6: Why a forest is preferred to a tree, on ninety points of a small two-feature example with two overlapping classes. (a) A single tree grown without restriction: the regions are boxes bounded by cuts perpendicular to the axes, every training point is classified correctly, and seven of the leaves contain just one point — the thin horizontal slivers on the right are each there to accommodate a single awkward example. (b) The boundaries of three individual trees of a forest, each grown on its own random sample of the data and of the features; they agree on the broad picture and disagree in the details. (c) The forest’s verdict, the vote of three hundred such trees, shaded by the fraction voting for each class: the fragmentation of the single tree has averaged away, and the result is graded rather than hard-edged.

Gradient boosting. Our third classifier is built from the same trees, combined on the opposite principle, and this contrast is the reason for including it alongside the forest. A forest grows its trees independently

and lets them vote. In *boosting*, the trees are constructed *in sequence*, each one fitted to the errors that the trees built so far have not yet corrected [21].

Figure 6.7 follows this on the same ninety points used for the forest, with the training points the ensemble still misclassifies ringed in black. The first tree, in panel (a), is a single cut of the plane: it is deliberately shallow, and on its own it leaves eleven points on the wrong side. The subsequent trees do not attempt the problem afresh. Each is fitted to what the accumulated sum still gets wrong, so the ringed points are precisely what the next tree is asked to address, and as trees are added the rings disappear: ten remain after five trees in panel (b), five after fifty in panel (c), by which stage the boundary has acquired a structure that no individual stump could express. The strength of the method therefore comes from accumulation, not from the quality of its parts.⁵ Where the forest reduces overfitting by averaging strong predictors that disagree, boosting assembles a single strong predictor from many weak ones.

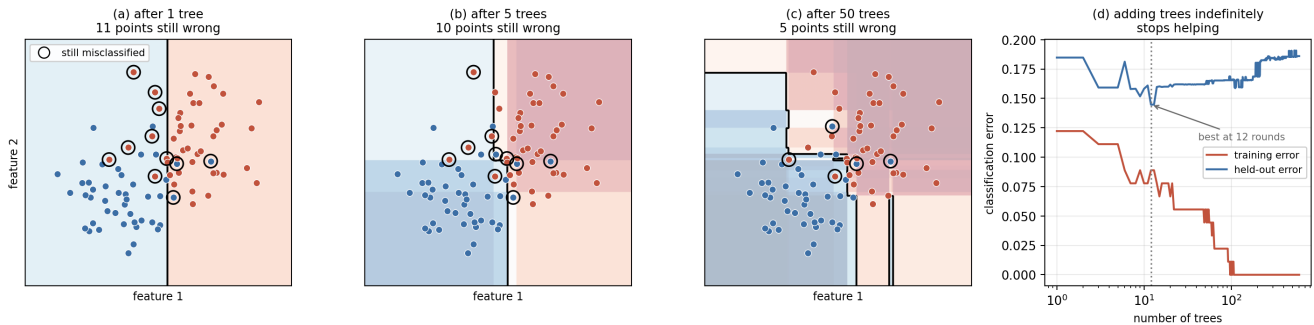


Figure 6.7: Boosting on the ninety points of Figure 6.6, with the training points still misclassified ringed in black. (a) A single shallow tree cuts the plane once and leaves eleven points wrong. (b) and (c) After five and fifty trees, each fitted to the errors its predecessors left behind, the ringed points reduce to ten and then five, and the boundary acquires structure no single cut could produce. (d) Error against the number of trees, on a logarithmic scale: the training error (red) falls to zero, while the error on a new samples of the dataset (blue) is lowest after about twelve rounds and then rises: the ensemble having begun to fit the noise of the training set rather than the structure shared with fresh data.

The sequential construction is also the limitation, in two respects. Since each tree depends on all those preceding it, the trees cannot be built in parallel as a forest’s can. And because every round is spent fitting the errors that remain, the procedure will eventually begin fitting the noise, which is exactly what panel (d) of Figure 6.7 shows: the training error falls monotonically and reaches zero, while the error on data the model has not seen bottoms out after about a dozen rounds and then deteriorates steadily.⁶ The number of rounds is therefore a quantity requiring control rather than simple increase, and we set it by the procedure described in Section 6.1.3.

Convolutional neural networks. The remaining two classifiers address the second kind of input described above, in which the example is a *curve* or an *image* rather than an unordered list of numbers. Both are *convolutional neural networks*, and it is worth saying now what is a neural network (NN) in general.

A neural network is a computing structure loosely modelled on the brain [21]. It is built from a large number of very simple units, often called neurons, arranged in successive layers. Each unit receives a handful of numbers, combines them into one, and passes that result on to the units of the next layer; no single unit does anything clever, and the capability of the network comes entirely from how many of them there are and how they are connected. Learning consists of adjusting the strengths of those connections until the whole assembly maps its inputs to the right answers. Given enough units, a network of this kind can represent extremely intricate relationships between input and output — far beyond the single flat cut available to logistic regression.

⁵The book develops this through AdaBoost, which reweights the training examples at each round. The variant we use, gradient boosting [22], instead fits each new tree to the residual errors of the current sum; the principle described here is unchanged.

⁶The *training error* is simply the fraction of the training examples the model currently gets wrong; it is what each new tree is fitted to reduce, which is why it falls every round and eventually reaches zero.

An ordinary network of this sort treats each position of its input as a separate thing to be learned about. If the pattern that distinguishes the two classes may appear anywhere along a curve (or along an image), such a network has to learn to recognise it separately at every position it might occupy, which is wasteful and requires a great deal of data. A *convolutional* network [68] removes this duplication with one idea. Rather than learning a separate rule for every position, it learns a single small *detector* — a compact group of weights, small enough to look at only a short stretch of the input at a time, whose job is to recognise one particular pattern — and then applies that very same detector at every position in turn, sliding it across the input from one end to the other. One pattern is therefore learned once and recognised wherever it occurs. This both reduces enormously the number of quantities to be fitted and builds in the assumption that position of the pattern should not matter.

Figure 6.8 shows how such a network is assembled, and the stages read from left to right as follows.

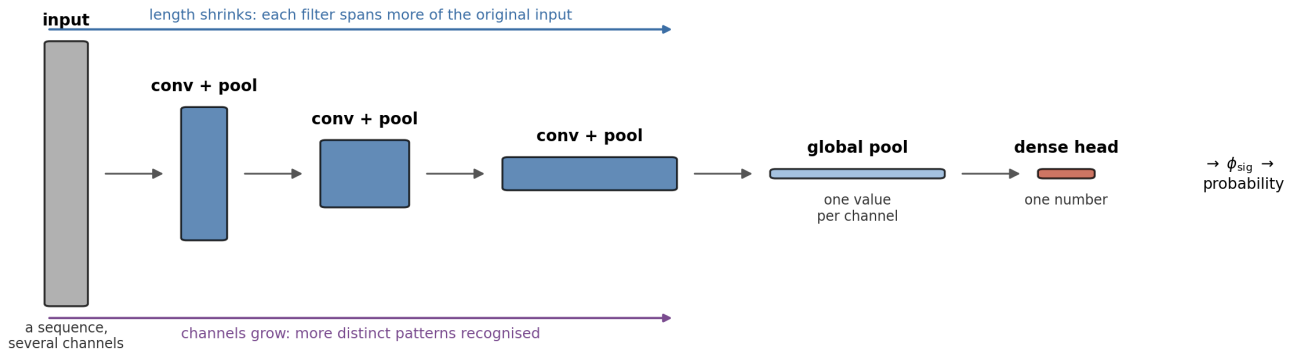


Figure 6.8: The convolutional network used in this work. Its input, on the left, is a curve or an image. Each block is one stage of the network, and the two arrows record what happens as the input passes through them.

- **The input.** A curve sampled on a grid, or an image. It may carry several *channels* — several parallel measurements at each position — which the detectors read together, so that a pattern involving more than one of them at once can be picked up.
- **Convolution.** Each small detector, called a *filter*, is slid across the input and reports at every position how strongly the pattern⁷ is present there. A stage uses many filters at once, each looking for a different pattern, and each produces its own map of responses.
- **Pooling.** The responses are then summarised over neighbouring positions, which shortens what is passed to the next stage. This is what makes stacking worthwhile: a filter of the same small size, applied to an already shortened input, effectively covers a wider span of the original curve, so successive stages respond to progressively coarser structure while the early ones respond to fine detail.
- **Global pooling.** After the last stage, there is a pooling that produces a number per channel: how strongly that pattern was found anywhere in the input. This is the step that discards position altogether, leaving only which patterns were present and how strongly.
- **The dense head.** A small ordinary layer reduces those per-pattern strengths to one number, which the sigmoid of (6.2) converts into a probability, read exactly as in logistic regression.

A network usually has considerably more parameters to fit than the other three classifiers, and therefore requires correspondingly more data before it generalises reliably. It is also the least transparent of the four: there is no analogue of reading off one weight per feature, and there is no simple way to say what pattern any given filter has learned to detect.

Nothing about a convolutional network is fixed in advance: the architecture of Figure 6.8 is one arrangement among a great many possible ones. We chose to use three convolutional stages, but there could have been two or ten; we chose how the pooling steps summarise neighbouring positions, and how much they shorten the input at each stage. Every one of these is a decision made before any fitting takes

⁷The pattern that precise filter is looking for.

place, and quantities of this kind — settings we fix ourselves rather than values the fitting procedure determines — are called *hyperparameters*.

The other classifiers have those hyperparameters too, and several have already appeared above: how many trees a forest contains and how deep each is allowed to grow, how many rounds boosting is run for. There is no architecture or set of hyperparameters that is right in general. The appropriate choice depends on the kind of input and on how much data is available, and it cannot simply be read off: a network with more stages, like a forest of deeper trees or a boosting run continued for too many rounds, will fit the training data better while generalising worse, which is the overfitting problem of Section 6.1.1 appearing once again.

6.1.3 Validating the classifiers

What remains is to specify how we measure whether any of this works, which by Section 6.1.1 means estimating the error a trained classifier would make on examples it has never seen.

The obvious procedure is to set aside part of the data before training, fit the model on what remains, and score it on the part held back [21]. To see what it gives in practice, consider a dataset of one hundred labelled examples:

- Hold back twenty of the examples at random.
- Fit a logistic regression to the remaining eighty.
- Score it on the twenty held back.

Doing this once returned an accuracy of 0.85⁸. Repeating it with a different random twenty — same data, same classifier, same procedure, only a different draw — returned 0.95. Over twelve such draws the accuracy ranged from 0.75 to 0.95.

That spread of 0.20 is the weakness of the method. The number it returns depends on *which* examples happened to land in the held-out part. The twenty held back are scored, so a difficult example placed among them is one the classifier can get wrong, whereas the same example placed among the eighty used for training costs nothing. Since each of the twenty is worth five percentage points of the final score, moving only a few difficult examples from one part to the other already shifts the accuracy substantially. With a single split we have no way of telling how much of the score is the classifier and how much is the luck of the draw.

We therefore use *k-fold cross-validation* [21], with $k = 5$, illustrated in Figure 6.9. The procedure is as follows.

- The data is divided into five parts of equal size.
- In the first round, the first part is held back, the model is fitted on the other four, and scored on the part held back.
- In the second round the model is discarded and fitted afresh on a different four parts, the second being held back this time, and so on for five rounds in all, each part serving once as the held-out one.
- The five scores obtained this way are averaged, and that average is what we report.

⁸Returned this accuracy in one round of a machine learning code in my machine.

the 100 examples, divided into five equal parts						
round 1	held out	train	train	train	train	score 0.75
round 2	train	held out	train	train	train	score 0.90
round 3	train	train	held out	train	train	score 0.80
round 4	train	train	train	held out	train	score 0.90
round 5	train	train	train	train	held out	score 0.95
						average 0.86

Figure 6.9: Five-fold cross-validation on the hundred-example dataset described in the text. Each row is one round: four parts are used for fitting and the remaining one, marked, is used for scoring. Every part is held out exactly once, so every example is scored exactly once by a model that was not fitted on it. The five scores obtained are averaged to give the reported figure.

Two things are gained. Every example is used for fitting in four of the five rounds, so no data is wasted; and every example is scored exactly once, by a model that never saw it. The gain is visible in the same example: the five rounds of Figure 6.9 scored 0.75, 0.90, 0.80, 0.90 and 0.95, averaging 0.86. Those individual scores vary as much as the single splits did, which is expected since each rests on twenty examples — but the average does not. Repeating the whole procedure with the data divided up differently moved that average only between 0.84 and 0.88, a spread of 0.04 against the 0.20 of a single split.

We make the folds *stratified*, meaning that each of the five parts is constructed to contain the two classes in the same proportion as the full dataset, so that no round is fitted or scored on an accidentally unbalanced sample.

Leakage. All of this rests on one assumption: that the held-out fold really was unknown to the model that scores it. When that assumption fails — when information about the held-out examples reaches the model by some route — the estimate is inflated, and the failure is called *leakage*. The situation is that of a candidate who has seen the examination paper in advance. The mark obtained is genuine in the sense that it was really earned on those questions, but it no longer measures what it was intended to measure, and it will not be reproduced on a paper the candidate has not seen. Leakage is insidious because it never announces itself: it makes results look better, not broken, so there is nothing obviously wrong to notice.

What we report. The single number one reaches for first is the *accuracy*, the fraction of examples labelled correctly. On its own the accuracy is not enough, and a concrete case shows why. Consider a dataset of one thousand examples of which only fifty belong to class 1, and a classifier whose predictions on them are collected in Table 6.2.⁹ A rule that ignored its input entirely and answered class 0 every time would already be right nine hundred and fifty times out of a thousand, an accuracy of 0.950. The classifier of the table achieves 0.963, only a little higher — a reminder that when one class is much larger than the other, accuracy is weighted heavily towards how well that large class is handled, and a high accuracy do not mean that the classifier has learnt to recognize the smaller class.

⁹Neither the origin of this dataset nor which classifier produced these predictions matters for the point being made here; the four counts of Table 6.2 are all that is used below.

		predicted		
		class 1	class 0	total
truly	class 1	15	35	50
	class 0	2	948	950
total		17	983	1000

Table 6.2: The four outcomes of a classifier on one thousand examples, of which fifty belong to class 1. Of those fifty, fifteen were found and thirty-five missed ($15 + 35 = 50$); of the nine hundred and fifty belonging to class 0, two were wrongly assigned to class 1 and the rest correctly rejected ($2 + 948 = 950$). The bottom row totals what the classifier assigned to each predicted class ($15 + 2 = 17$ assigned to class 1, $35 + 948 = 983$ to class 0), and both margins sum to the one thousand examples. Every quantity discussed in the text is computed from the four counts in the top-left block.

What the table makes visible is that there are two quite different ways of being wrong, and that the accuracy cannot report both. We therefore use, for the class of interest, the three standard quantities alongside accuracy: *precision*, *recall* and the F_1 score [62]:

- The *precision* is the fraction of the examples assigned to the class that really belong to it: here 15 of the 17 examples called class 1 were genuinely class 1, giving $15/17 = 0.88$. It measures how far a positive answer can be trusted.
- The *recall* is the fraction of the examples truly in the class that were found: here 15 of the 50, giving $15/50 = 0.30$. It measures how much the classifier misses.
- The F_1 score is the harmonic mean of the two, here 0.45. It is a single number that is high only when precision and recall are both high, so it cannot be inflated by sacrificing one for the other.

The pair says more here than the accuracy alone did. A precision of 0.88 with a recall of 0.30 suggests that this classifier is cautious: when it commits to class 1 it is usually right, but a recall of 0.30 means it only picks out three examples in ten that truly belong to class 1, leaving the other seven in ten wrongly labelled class 0 — a distinction the accuracy of 0.963 did not make visible on its own.

The threshold, and the ROC curve. Precision and recall are not fixed properties of a fitted classifier. Every quantity above follows from comparing the predicted probability against the threshold (the threshold that in before example was 0.50), and that threshold is a choice we are free to change. Lowering it makes the classifier more willing to answer class 1, raising recall and lowering precision; raising it does the reverse. Table 6.3 shows the same fitted model, unchanged, read at three different thresholds.

threshold	found	missed	false alarms	precision	recall	F_1
0.20	28	22	40	0.41	0.56	0.47
0.50	15	35	2	0.88	0.30	0.45
0.80	7	43	0	1.00	0.14	0.25

Table 6.3: One fitted classifier, read at three thresholds, on the dataset of Table 6.2. At 0.80 every example it commits to is correct, but it finds only seven of the fifty; at 0.20 it finds twenty-eight, at the cost of forty false alarms. Nothing about the model changes between rows — only the value the predicted probability must exceed.

Reporting precision and recall at one threshold therefore describes one operating point of a classifier rather than the classifier itself. The *receiver operating characteristic* (ROC) curve avoids this by plotting, for *every* threshold at once, two quantities against each other:

- the fraction of class-1 examples found;
- the fraction of class-0 examples wrongly flagged.

Figure 6.10 shows the curve for this classifier, with the three rows of Table 6.3 marked on it: sliding the threshold moves the operating point along the curve, and choosing a threshold amounts to choosing where on it to sit.

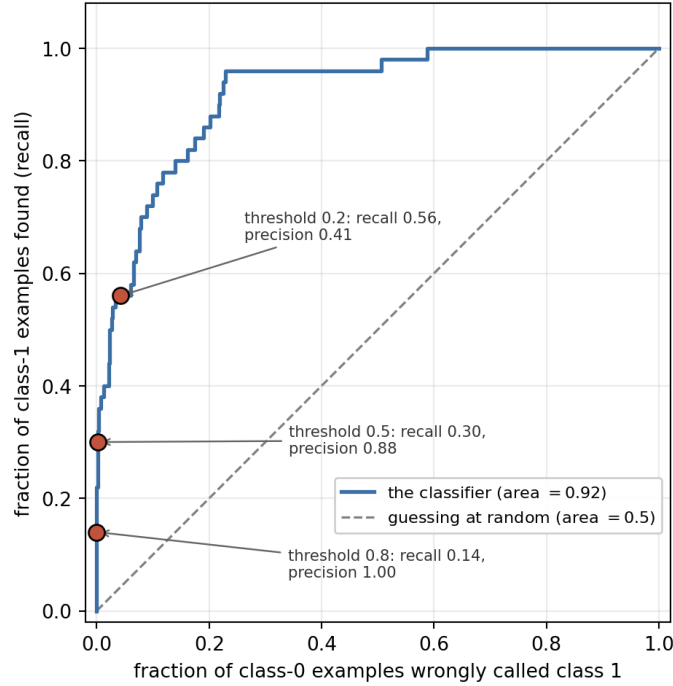


Figure 6.10: The ROC curve of the classifier of Table 6.2, with the three thresholds of Table 6.3 marked on it, against the diagonal traced out by a classifier that guesses at random.

The area beneath that curve — the $ROC-AUC$ ¹⁰ — summarises the whole curve in one threshold-free number. What that number means is easiest to see by considering the two extreme cases first.

- **A classifier that guesses at random.** Such a classifier assigns probabilities that carry no information about the class, so at any threshold it flags class-1 and class-0 examples in the same proportion: if it catches half of the class-1 examples it also wrongly flags half of the class-0 ones. Its two coordinates in the ROC plot are therefore always equal, and the curve is the diagonal line from corner to corner, shown dashed in Figure 6.10. The area beneath a diagonal across a unit square is half of it, so the area is 0.5. This is the floor: no useful classifier should score below it.
- **A perfect classifier.** Here every class-1 example receives a higher probability than every class-0 example, so some threshold sits cleanly between the two groups. At that threshold all fifty class-1 examples are found and no class-0 example is flagged wrongly, which is the point at the top-left corner of the plot. The curve passes through that corner and so encloses the entire square, an area of 1. This is the ceiling.
- **The classifier of our example.** Its curve, drawn solid in Figure 6.10, rises steeply from the origin and then flattens, sitting well above the diagonal but short of the corner, and encloses an area of 0.92. Being much closer to 1 than to 0.5 says that the probabilities it assigns do classify the examples reasonably well.

6.2 Results

Section 5.3 left us with a table of topological features, one row per time series, and Section 6.1 described the ML classifiers that can be trained on a generic dataset. In this section we combine the two: we train ML classifiers on the topological features of the gravitational-wave dataset and report the results.

¹⁰AUC stands for Area Under the Curve.

Unlike the previous sections, no code is shown here. Fitting a classifier and cross-validating it is standard work that can be done with any of several libraries, and none of the choices involved is specific to topological data analysis.

6.2.1 The dataset and the features

Everything reported below comes from a single run of the pipeline built over the course of this chapter. The dataset is the one whose construction was described in Section 5.1: half of its time series contain a gravitational-wave signal buried in coloured detector noise, the other half contain noise alone. That section deliberately left its parameters symbolic, since the procedure does not depend on them; Table 6.4 now fixes the values actually used.

Parameter	Value
Number of time series	20 000
Signal-bearing / noise-only	10 000 / 10 000
Target SNR of the injected signals (Section 5.1.4)	drawn in $[6, 30]$
Sampling frequency	4096 Hz

Table 6.4: The parameters of the gravitational-wave dataset the results of this section are based on. The two classes are exactly balanced, so a classifier that ignored its input entirely could not do better than one half.

The topological features are those of Section 5.3, computed with the settings collected in Table 6.5.

Parameter	Value
Takens embedding dimension (Section 5.3.1)	8
Projection onto principal components (Section 5.3.2)	none
Homology dimensions computed	H_0 and H_1
Bins per Betti curve (Section 2.4.3)	300
Windows per Betti curve (Def. 2.18)	10
Persistence image resolution (Section 2.4.2)	32×32
Resulting scalar features (entropy, max persistence, ...)	28

Table 6.5: The settings of the topological pipeline used to produce the features analysed in this section. The 28 scalar features are the four summaries of Section 5.3.4 in each of the two homology dimensions, together with the ten windowed Betti-curve areas in each of the two dimensions.

Two of these settings are worth a word, since both depart from what earlier sections did.

No projection onto principal components. Section 5.3.2 introduced the option of projecting each point cloud onto a few principal components before computing its homology, as a way of keeping the cost of the Rips filtration down, and then examined what such a projection costs. Two findings there bear on the choice. First, the variance of these clouds is spread almost evenly across the principal components, rather than concentrated in the first few, so every direction carries a comparable share of the cloud’s structure and discarding any of them discards a comparable amount of it. Second, the projected clouds of the two classes overlap, so the projection is of little help in telling signal from noise. For the run reported here we therefore paid the computational price and dropped the projection, computing the homology of the full eight-dimensional embedded clouds.¹¹

Only two homology dimensions. Only H_0 and H_1 — connected components and loops — are computed, not H_2 (3d-cavities). This is the practical limit already met in Section 3.6, where the cost of reaching the second homological dimension on a realistic time series was found to be very large. Nothing

¹¹This embedding dimension, 8, is not chosen by hand: it is the value found by the search algorithm of Section 5.3.1, which looks for the smallest dimension at which the embedding stops introducing false neighbours.

in the results below depends on H_2 cavities, and Section 5.2.4 we already studied how the presence of a GW signal affects the H_0 and H_1 topological features, so we can use these H_0 and H_1 features to detect the presence of a GW in the noise. As it turns out, this restriction costs us little: the H_2 features carry comparatively little information about whether a GW signal is present, a point we return to below, once the random forest’s feature importances are explained in Section 6.2.4.

6.2.2 The classical classifiers

Three of the five classifiers take a feature vector: a plain list of numbers, the first family of Section 6.1.2. Concretely, what each of them is given, for every time series, is one row of 28 numbers: the four scalar summaries of Section 5.3.4 (persistent entropy, maximum persistence, total persistence, number of points), each computed once per homology dimension, together with the windowed Betti-curve areas of the same section, ten per homology dimension. Table 6.6 sets these out.

Summary		What it measures	Count H_0, H_1
Persistent (Def. 2.16)	entropy	how evenly the persistence is spread across the diagram’s points	2
Maximum (Def. 2.14)	persistence	the lifetime of the single longest-lived feature	2
Total persistence (Def. 2.13)		the sum of the lifetimes of every feature	2
Number of points (Def. 2.15)		how many features the diagram contains	2
Windowed Betti-curve area (Def. 2.18)		the Betti curve’s area, one number per equal window of the filtration scale	20
Total			28

Table 6.6: The 28 scalar features handed to logistic regression, gradient boosting and the random forest, one row of these 28 numbers per time series. Every summary is exactly the one defined in Section 5.3.4; this table only collects them.

Briefly, and with Section 6.1.2 for the details:

- **logistic regression** weighs the features, sums them, and separates the two classes with a hyperplane;
- the **random forest** grows many decision trees on randomised subsets of the data and takes their vote;
- **gradient boosting** grows shallow trees one after another, each fitting what the ones before it still get wrong.

Table 6.7 collects the outcome, for these three and for the two convolutional networks discussed in Section 6.2.3. The training and validation accuracy are both computed with the stratified k -fold cross-validation of Section 6.1.3¹², whose purpose is to give a reliable estimate of how well a model generalises to time series it has not been trained on, by repeatedly training and testing it on different subsets of the dataset (see 6.1.3 for further details). Concretely:

- the **training accuracy** is how often the model is correct on the very examples it was fitted on;
- the **validation accuracy** is how often it is correct on the one fold held back from that fit, which the model never saw during training. This is really a measure of how well the model generalises to new data, namely how well the classifier is able to detect GW signals in new data which has not been used in the training of the model.

A model that has genuinely learned something about the data, rather than merely memorised it, should do about as well on the second item as on the first item (Section 6.1.1). And this is what happens in the results of Table 6.7: the training and validation accuracies are very close to one another.

¹²Here $k = 5$.

Input	Classifier	Train acc.	Valid. acc.	Valid. AUC
28 scalar features	Logistic regression	0.836	0.836 ± 0.003	0.901 ± 0.003
	Gradient boosting	0.847	0.839 ± 0.002	0.901 ± 0.003
	Random forest	0.851	0.838 ± 0.001	0.902 ± 0.004

Table 6.7: Accuracy and area under the ROC curve for the three feature-vector classifiers, averaged over the five cross-validation folds. The quoted spread is the variation from one fold to the next: the three models agree with one another to within a few thousandths. The two convolutional networks are reported separately in Section 6.2.3.

The first thing to read from it is how little separates the three:

- their **validation accuracies** fall within 0.003 of one another;
- their **areas under the ROC curve** fall within 0.001 of one another. This area, usually abbreviated AUC, is a single number summarising the whole curve of Section 6.1.3: the closer it is to 1, the closer the classifier is to perfect, while a classifier that guesses at random has an AUC of 0.5.

Logistic regression, the simplest model available and the only linear one, performs as well as the non-linear models such as the random forest and the gradient boosting. That is not what one expects, and Section 6.2.4 takes it up.

What the errors look like. An accuracy alone is not a complete measure of a classifier’s performance, so Table 6.8 also reports recall, precision, and F_1 scores. Recall that precision is the fraction of a classifier’s positive calls that are correct, recall the fraction of the examples truly in the class that it manages to find, and F_1 the single number that is high only when both are (Section 6.1.3).

Classifier	Class	Precision	Recall	F_1	Accuracy
Logistic regression	Noise-only	0.800	0.895	0.845	0.836
	Signal	0.881	0.776	0.825	
Gradient boosting	Noise-only	0.792	0.921	0.851	0.839
	Signal	0.906	0.758	0.825	
Random forest	Noise-only	0.790	0.921	0.850	0.838
	Signal	0.905	0.755	0.823	

Table 6.8: Precision, recall and F_1 for each class, computed on the predictions the five-fold cross-validation makes for each time series while it is held out. All three classifiers are markedly better at recognising noise than at finding signals: high precision on the signal class against distinctly lower recall.

The asymmetry is the same for all three classifiers, and it is worth stating plainly:

- on the signal class the **precision** is high, around 0.88 to 0.91: when one of these classifiers announces a gravitational wave, it is usually right;
- on the signal class the **recall** is lower, around 0.76: it fails to identify roughly one signal in four that is genuinely present. Together with the high precision above, this indicates that these classifiers are cautious: they announce a gravitational wave only once the evidence is strong enough to leave little doubt.
- on the signal class the F_1 **score**, which is high only when both precision and recall are, sits between the two at around 0.82 to 0.83 — pulled down from the precision by the weaker recall.

The precision above 0.88 means only a small fraction of the classifier’s positive calls are wrong, so false alarms are rare, while a recall of 0.76 means a comparatively larger fraction of the genuine signals are never flagged at all by the machine learning classifier. Section 6.2.5 identifies which signals get missed. As Section 6.1.3 explains, these precision and recall values are not fixed properties of a classifier: every one of them follows from comparing the classifier’s predicted probability against a threshold, one half by default,

above which it answers that a signal is present. Move that threshold and precision and recall move with the threshold so the numbers above describe this one choice of threshold rather than the classifier itself. A threshold-free comparison of all five classifiers, including the two convolutional networks, is given in Section 6.2.3.

How the models behave as the training set grows. Figure 6.11 plots, for each model, its training accuracy and its validation accuracy¹³ as the amount of training data is increased.

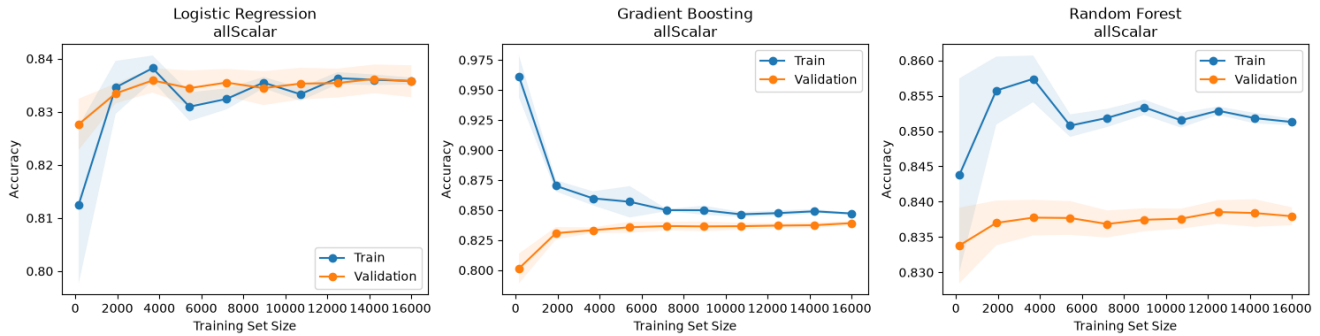


Figure 6.11: Accuracy on the training data and on validation data, against the number of examples used for training, for the three feature-vector classifiers. The shaded band around each curve shows how much that accuracy varies across the cross-validation folds. Note that each panel has its own vertical scale.

The three panels look quite different from one another:

- for **logistic regression** the two curves sit on top of each other over the whole range: the model does no better on the examples it was fitted on than on ones it has never seen;
- for **gradient boosting** the training accuracy starts far above the validation accuracy — about 0.96 against 0.80 with only a few hundred examples — and falls steadily as more data is added, while the validation accuracy rises, the two ending up only about 0.01 apart;
- for the **random forest** a gap of roughly one and a half percentage points opens early and persists across the whole range.

These curves are worth reading in general terms as well as panel by panel:

- the training accuracy is expected to start out higher than the validation accuracy, since it is measured on the very examples the model used to learn and find patterns, while the validation accuracy is measured on examples it has never seen;
- as the training set grows, the validation accuracy is expected to rise and approach the training accuracy¹⁴ — the gap closing is what it looks like for a model to keep learning genuine, generalisable patterns as it sees more examples.

This is exactly the pattern seen for logistic regression and gradient boosting: in both panels the validation curve rises to meet the training curve. The random forest is the exception, where the two curves stay a fixed distance apart across the whole range instead of closing. A plausible reason is that the forest's individual trees are grown under fixed regularisation settings — how deep a tree is allowed to go, how few examples a split or a leaf may contain — that do not relax as the training set grows, so the model's effective capacity relative to the task stays roughly constant, and with it the size of the gap; we have not investigated this further.

¹³Defined in Section 6.2.2: the training accuracy on the examples the model was fitted on, the validation accuracy on the fold held back from that fit.

¹⁴The training accuracy itself is usually expected to fall as the training set grows: a classifier flexible enough to fit a small dataset almost perfectly finds it harder to fit every individual example once there are many more of them, so its training accuracy settles towards whatever it can genuinely achieve, rather than staying near the near-perfect fit a small dataset allows.

6.2.3 The convolutional networks (CNNs)

The remaining two classifiers belong to the second family of Section 6.1.2, the one whose input is:

- A Betti curve (Section 2.4.3) is a sequence, sampled on a grid of x -values¹⁵. It records how many topological features of a given dimension are alive at each scale, so neighbouring entries report the count at neighbouring scales.
- A persistence image (Section 2.4.2) is an image. Each pixel measures how much persistence sits in one small region of the persistence diagram, so neighbouring pixels describe neighbouring regions of that diagram.

In both cases the position of an entry carries meaning: it says *where* in the filtration, or *where* in the diagram, that number was measured.

Flattening is what one has to do to hand such an input to a classifier that expects a plain feature vector (like the logistic regression): the entries are read off in some fixed order and written out as one long list. For a persistence image the usual order is row by row, so a small 3×3 image becomes

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \mapsto (a_{11}, a_{12}, a_{13}, a_{21}, a_{22}, a_{23}, a_{31}, a_{32}, a_{33}).$$

No number has been changed, but the geometry has been discarded. In the image, a_{11} and a_{21} sit one directly above the other, and are therefore neighbours; in the list they are three places apart. A classifier reading the list has no way to recover which entries were adjacent in the picture, because for such a classifier the list is just 9 separate measurements whose order carries no meaning at all — entry 3 is simply a different number from entry 4. Everything the arrangement of the pixels was expressing is lost.

A convolutional network avoids this precisely because it does not flatten: it works on the sequence or the image as it stands, sliding a small detector across it (Section 6.1.2) and so using the very adjacency that flattening destroys. The one-dimensional network is given the Betti curves, the two-dimensional one the persistence images¹⁶. Both are cross-validated on the same five folds as the models above. Table 6.9 and Table 6.10 give their results, in the same form as Table 6.7 and Table 6.8 did for the three classical models.

Input	Classifier	Train acc.	Valid. acc.	Valid. AUC
Betti curves	One-dimensional CNN	0.608	0.607 ± 0.089	0.896 ± 0.004
Persistence images	Two-dimensional CNN	0.816	0.814 ± 0.013	0.901 ± 0.004

Table 6.9: Accuracy and area under the ROC curve for the two convolutional networks, in the same form as Table 6.7. The quoted spread is the variation from one cross-validation fold to the next.

Classifier	Class	Precision	Recall	F_1	Accuracy
One-dimensional CNN	Noise-only	0.607	0.610	0.609	0.608
	Signal	0.608	0.605	0.606	
Two-dimensional CNN	Noise-only	0.751	0.940	0.835	0.814
	Signal	0.920	0.689	0.788	

Table 6.10: Precision, recall and F_1 for each class, for the two convolutional networks, in the same form as Table 6.8.

The two-dimensional network. On persistence images the network performs comparably to the three classical classifiers:

- its validation accuracy is 0.814, against roughly 0.84 for logistic regression, gradient boosting and the random forest;

¹⁵Remember the x axis of a Betti curve plot is the filtration scale t .

¹⁶In both cases with the two homology dimensions supplied as separate channels — channels are defined in Section 6.1.2.

- its area under the ROC curve is 0.901, matching the 0.901 to 0.902 of those three.

The errors of the CNNs are also of the same kind of the errors of the classical models: a precision of 0.920 on the signal class against a recall of 0.689, so it misses more signals than the classical models do while raising fewer false alarms.

This deserves a comment. A convolutional network is a far more elaborate model than a random forest (Section 6.1.2), with many more quantities to fit, so one might expect it to outperform the simpler classifiers rather than merely match them. Two considerations bear on this.

- **The architecture is itself a choice, and not an obvious one.** As discussed in Section 6.1.2, how many convolutional stages a network has, how wide its filters are and how its pooling steps summarise neighbouring positions are all hyperparameters, fixed before any fitting takes place. There is no way to read the right values off the problem in advance, and finding an architecture well suited to a particular input is a matter of systematic experimentation. The architecture used here was chosen as a reasonable default rather than arrived at by such a search, so the CNN architecture is not necessarily the best for this specific dataset.
- **The dataset may not be large enough to reward the extra complexity.** A model with more quantities to fit generally requires more data before it generalises reliably. It is possible that with a substantially larger dataset the network would begin to exploit structure in the persistence images that the scalar summaries cannot express, and pull ahead of the classical models. We have not tested this.

The one-dimensional network. On Betti curves the CNN performs markedly worse. Its validation accuracy is 0.607 — better than guessing, but far below every other model here.

A firm explanation for this would require further study rather than the reinterpretation of the numbers already in hand. One possibility among several is simply that the architecture used — the stack of convolutions and pooling steps described in Section 6.1.2 — is not well suited to inputs such the Betti curves, and that a different arrangement of the same ingredients would do better. We record the result as it stands and leave the question open.

Comparing all five classifiers. The tables above report accuracy and per-class scores separately for the three feature-vector classifiers of Section 6.2.2 and the two convolutional networks of this subsection. Figure 6.12 instead plots the ROC curve (Section 6.1.3) of all five classifiers. Briefly, the curve is obtained by sweeping the classification threshold across its whole range and, at each value, plotting the fraction of signals correctly found against the fraction of noise-only time series wrongly flagged; a curve that hugs the top-left corner describes a classifier that finds signals without raising false alarms, while the diagonal is what a classifier guessing at random would trace out.

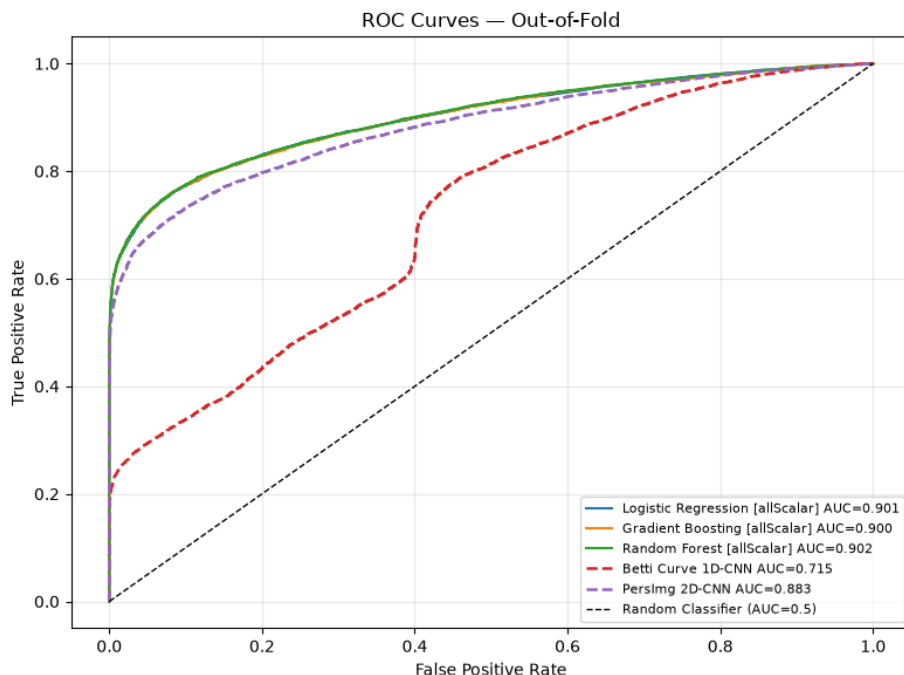


Figure 6.12: ROC curves for all five classifiers, traced out by sweeping the decision threshold across its whole range. The three feature-vector models lie so nearly on top of one another that their curves are hard to tell apart; the two-dimensional network follows them closely, and the one-dimensional network trails well below.

The three feature-vector curves are indistinguishable by eye, all enclosing an area of about 0.90; the two-dimensional network’s curve sits close behind them, enclosing an area of 0.883¹⁷; the one-dimensional network’s curve is visibly the lowest of the five, enclosing an area of only 0.715.

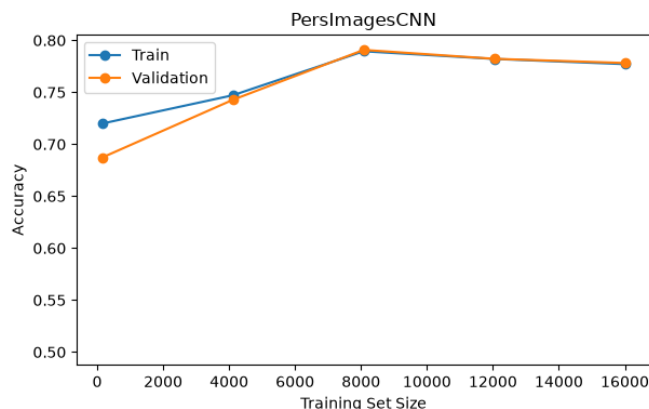


Figure 6.13: Accuracy on training and held-out data against the number of training examples, for the two-dimensional network. The two curves climb together and flatten, close to one another throughout.

Figure 6.13 plots the training and validation accuracy¹⁸ of the two-dimensional network as the amount of training data is increased. It behaves as the classical models did: both curves climb and then flatten, close together.

6.2.4 Results interpretation

Two questions are left over from Section 6.2.2: which of the 28 topological features actually carry the distinction between the two classes, and why a linear model does as well as anything else.

¹⁷This is slightly below the 0.901 reported for the same network in Table 6.9. The two numbers are computed differently; see Appendix A.45 for precisely how.

¹⁸Defined in Section 6.2.2: the training accuracy on the examples the model was fitted on, the validation accuracy on the fold held back from that fit.

Which features matter

Which features the forest uses. A random forest offers a direct way to answer the first question, because its internal workings can be inspected one step at a time. Recall from Section 6.1.2 that a tree classifies by asking a sequence of questions, each of the form “is this one feature above or below some threshold?”, and that each such question — a *split* — divides a group of examples into two smaller groups. Some splits accomplish more than others, and a feature used in more productive splits is doing more of the work of separating the classes. This is the basis for scoring each feature’s importance. Concretely:

- A group of examples is called *pure* when all of its members belong to the same class, and *impure* when the two classes are mixed within it. Purity is what a tree is working towards: once a group is pure, there is nothing left to decide about it, and it can simply be labelled with the class it contains.
- A split is useful when it leaves the two groups it produces purer than the group it divided — when it sends most of the signal-bearing time series one way and most of the noise-only ones the other. How much purer they are can be measured, and the difference before and after is the *decrease in impurity* that split achieved.¹⁹
- Each split uses exactly one feature. A feature that repeatedly enables large decreases in impurity is doing much of the work of separating the classes, while a feature whose splits barely improve anything is contributing little.
- For a given feature, we take every split that uses it, across all 500 trees of the forest, and add up the decreases in impurity those splits achieved. The total is a single number per feature, its *importance*. This is the quantity plotted in Figure 6.14, on a logarithmic vertical scale.

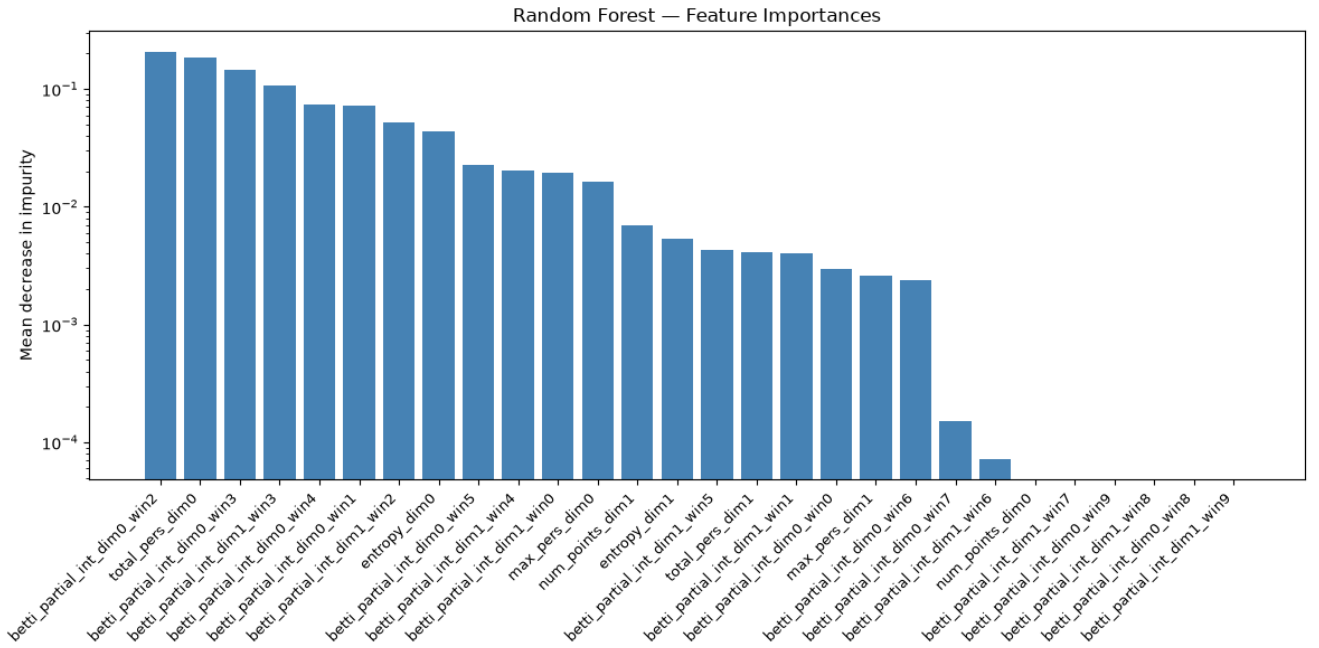


Figure 6.14: How much each of the 28 scalar features contributes to the random forest’s splits, sorted from largest to smallest, on a logarithmic vertical scale. A handful of features dominate; the last several contribute essentially nothing.

The distribution is very uneven, and it is worth going through what it says. Since the names of the leading features all refer to windowed Betti-curve areas, we recall briefly what those are. A Betti curve $\beta_i(t)$ counts how many features of dimension i are alive at each scale t of the filtration, so it is a whole curve rather than a single number. To hand it to a classifier we summarise it as in Definition 2.18: the filtration range is cut into ten equal intervals, called *windows*, and the area under the curve is computed

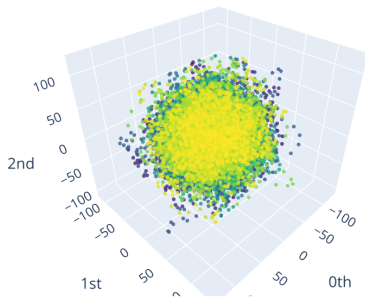
¹⁹The precise measures of impurity, and how a tree selects its splits, are those already referenced qualitatively in Section 6.1.2; see [62] for a more precise discussion, and for the specific measure used by the implementation of the random forest employed here.

separately over each one, turning the curve into ten numbers. Each of them measures how many features were alive during one stretch of the filtration, and for how long. The windows are numbered 0 to 9 in order of increasing scale, matching the labels in Figure 6.14: window 0 covers the earliest part of the filtration, where the balls are still small, and window 9 the latest, where they have grown large.

- **Components dominate.** The three largest contributions all concern dimension H_0 , that is connected components: the area under the H_0 Betti curve in window 2, the total persistence in dimension H_0 , and the area in window 3. Windows 4 and 1 of the same curve follow shortly behind.

This is exactly what the point clouds of Section 5.2.2 lead one to expect. Figure 5.10 showed that a chirp embeds not as one loop but as a nest of loops of steadily growing radius, the widest of them produced by the final cycles before the merger, where the waveform’s amplitude is largest. When noise is added, the bulk of the embedding becomes a dense ball, but those widest loops are wide enough to survive it: in the right-hand panel of Figure 5.10 they appear as sparse arcs standing outside the ball. The points making up these arcs are far from the dense core and far from one another, so in a Rips filtration (Section 2.1.2) they remain separate components until the scale t has grown large — long after every point of a noise-only cloud has merged into one. A signal-bearing time series therefore keeps a handful of components alive at filtration scales where a noise-only one has none, which is precisely what the H_0 Betti curve records — the same conclusion already reached in Section 5.2.4.

Takens embedding + 3D PCA -- Pure noise



Takens embedding + 3D PCA -- Chirp + noise

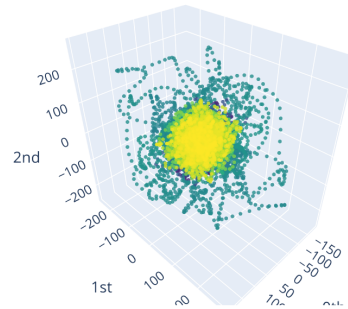


Figure 6.15: The noise-only and chirp + noise panels of Figure 5.10 (Section 5.2.2), repeated here for convenience. Pure noise (left) is a structureless ball; chirp + noise (right) is the same ball with sparse arcs — the widest of the chirp’s nested loops — still visible around its outside.

Figure 6.16 shows this directly. For pure noise the H_0 curve has reached zero before window 4; for chirp + noise it is still non-zero in window 9. Take window 2 or window 3 as an example: reading off the blue H_0 curve of Figure 6.16 at those windows, the chirp + noise sample still has many connected components alive, while the pure noise sample has almost none left. This is exactly why the areas in these windows rank so high in Figure 6.14: a feature that is large for one class and near zero for the other is precisely what a classifier needs. The *total persistence* in dimension H_0 ranks second in importance overall for the same reason expressed: it sums the lifetimes of all components, so the late-dying components of a signal-bearing cloud add to the total persistence a contribution that noise-only clouds simply do not have.

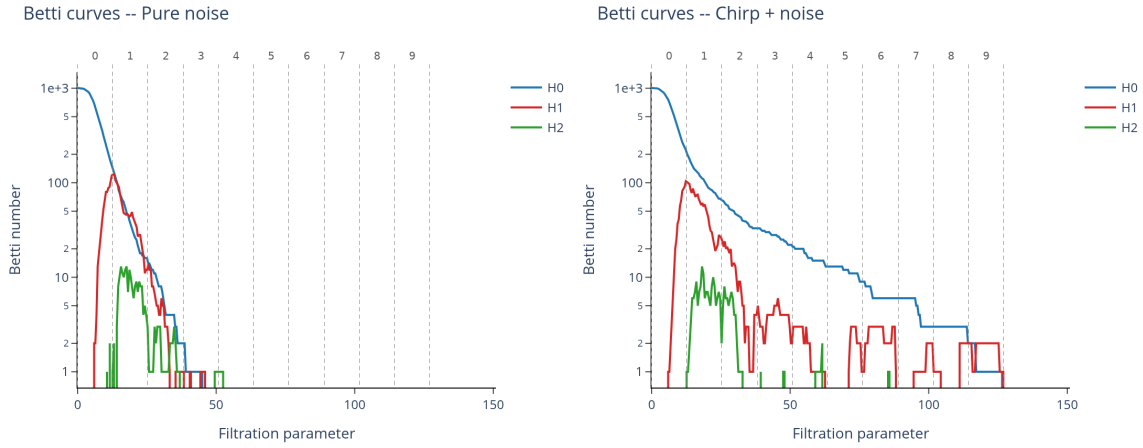


Figure 6.16: The Betti curves of Figure 5.12 for the noise-only sample (left) and the signal-bearing sample (right), with the ten windows over which the areas of Definition 2.18 are computed marked by dashed lines and numbered along the top. The noise curves have vanished by window 4, while the signal curves remain non-zero into window 9: the windows in between are where the two classes differ most, and the last windows are empty for both.

- **Loops matter too, though less.** The next tier in importance belongs to dimension H_1 , which counts loops: the area under the H_1 Betti curve in window 3 is the fourth most important feature overall, with windows 2 and 4 of the same curve close behind.

The intuition is the same one already used for H_0 . Section 5.2.3 found that, in the chirp + noise point cloud of Figure 5.10, a handful of the chirp’s nested loops remain visible as loops even with the noise added, surviving further along the filtration than any loop made of noise alone. These surviving loops are exactly what the H_1 Betti curve is counting.

Figure 6.16 shows the same pattern for H_1 as for H_0 : the red curve for pure noise has collapsed to zero by around window 3, while the red curve for chirp + noise is still non-zero much later. In windows 2 to 4, then, chirp + noise still has several loops alive while pure noise already has none — again a feature that is large for one class and near zero for the other, which is why these windows rank highly.

Still, these H_1 features rank below their H_0 counterparts, and the likely reason is simple: a loop is a more fragile thing than an isolated component. An isolated component only needs one point, or a small group of points, to stay away from everything else; a loop needs a whole ring of points to close up around empty space, with nothing wandering into the middle to fill it in. Noise is exactly the kind of disturbance that can put a stray point where it should not be, so loops are disrupted more easily than the isolated arcs are.²⁰

- **Many features contribute nothing, and this is not mysterious.** The vertical scale of Figure 6.14 is logarithmic because the smallest non-zero contributions lie some four orders of magnitude below the largest. At the far right sit features whose contribution is zero exactly: the number of points in dimension H_0 , and the areas in the last windows of both dimensions H_1 and H_2 . For the windowed areas the reason is visible in Figure 6.16: the late windows lie beyond the scale at which the Betti curves have fallen to zero for *every* time series in the dataset, signal or not. A feature that takes the same value (zero) for every example cannot separate signals from noise, and a tree will never choose to split on it.

²⁰There may be a further, geometric reason for this fragility. The pure cosine’s point cloud (Figure 3.2) is a single loop lying flat in one plane, because the cosine’s frequency never changes; the chirp’s point cloud (Figure 5.10), by contrast, is a tangle of loops at many different orientations, none confined to a single plane, because the chirp’s frequency is constantly changing (Section 2.3.1). A loop that is not flat may be harder for the Rips filtration to recognise cleanly than a flat one is, which would be a second reason H_1 features underperform H_0 ones here, independent of noise. This suggests a possible fix, which we have not investigated: applying the Takens embedding not to the whole time series at once but to a short sliding window of it. Within a short enough window the chirp’s frequency barely changes, so the embedded loop should come out close to flat, as in the pure cosine case, and so easier for the filtration to detect. We record this only as a plausible direction, not as a result of this thesis.

Why the H_2 features were left out. Section 6.2.1 noted that the results of this section use only H_0 and H_1 , for reasons of computational cost, and left open how much this actually gives up. The random forest’s feature importances answer this directly. We repeated the whole procedure with all three homology dimensions computed. The 28 scalar features of Table 6.5 are then joined by the H_2 versions of the same quantities: its entropy, maximum persistence, total persistence, number of points and windowed areas. We then read off the importances of this larger set exactly as above.

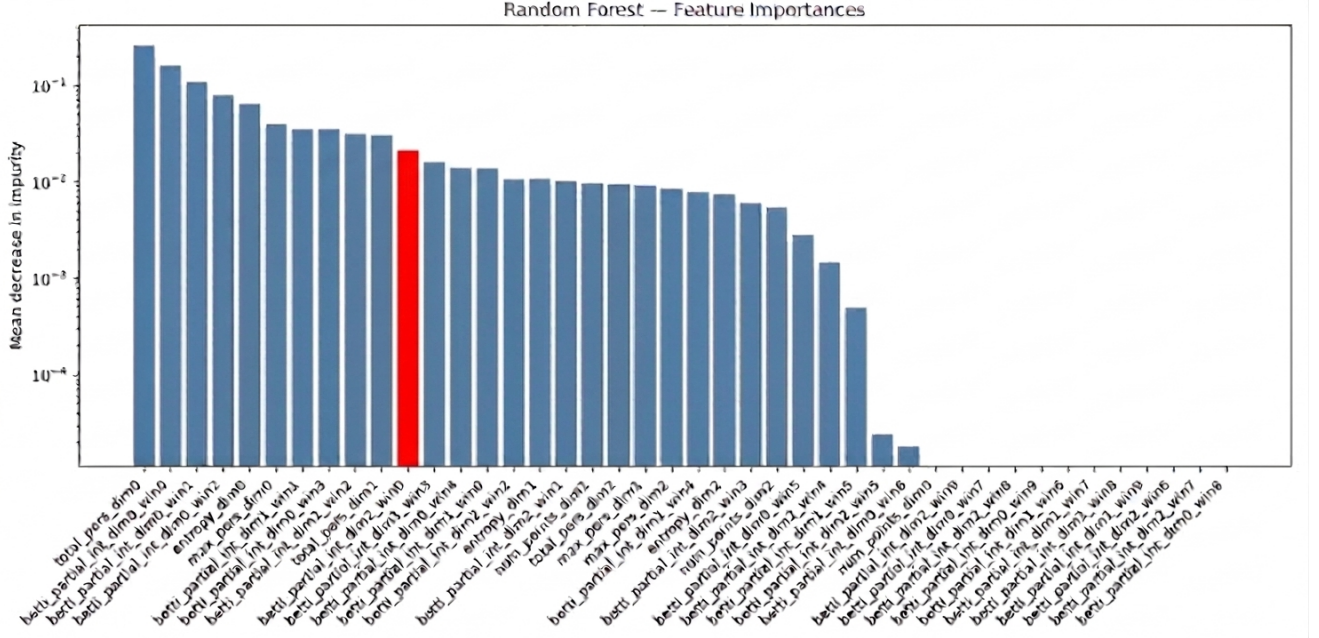


Figure 6.17: Random-forest feature importances, as in Figure 6.14, but for a run in which H_2 was also computed. The most important H_2 feature, the windowed Betti-curve area `betti_partial_int_dim2_win0`, is highlighted in red.

Figure 6.17 shows the outcome, again on a logarithmic scale. The single most useful H_2 feature, the windowed Betti-curve area in its first window, sits well down the ranking: about an order of magnitude below the most important feature overall, the total persistence in dimension H_0 . Every other H_2 feature ranks lower still, and several vanish entirely (namely there are not useful at all for the classification task using a random forest). The 3-dimensional cavities that H_2 tracks are, for this dataset, largely uninformative about whether a gravitational wave is present, which is consistent with computing only H_0 and H_1 throughout the rest of this section.

Which features the linear model uses. Logistic regression answers the same question differently. As Section 6.1.2 explained, this classifier fits one weight per feature during training, and that weight tells how strongly that feature pushes the decision towards one class or the other.²¹ Looking at the fitted weights therefore tells us which topological features the model actually relies on to detect a gravitational wave. Figure 6.18 shows them, sorted by size and signed.

²¹This reading is only valid because the features were put on a common numerical scale before fitting. The four independent scalar summaries (entropy, maximum persistence, total persistence, number of points) are each rescaled to mean 0 and standard deviation 1; the windowed Betti-curve areas, being correlated with one another by construction, are instead all divided by one shared factor, so as to preserve their relative shape across windows and dimensions rather than standardising each in isolation. Without some such rescaling, a feature naturally measured in the thousands would receive a small weight and a feature naturally measured in fractions a large one, for reasons having nothing to do with which feature is actually more informative. Putting every feature on a comparable scale also helps the fitting procedure converge faster, since gradient-based optimisation is slower when features differ wildly in scale.

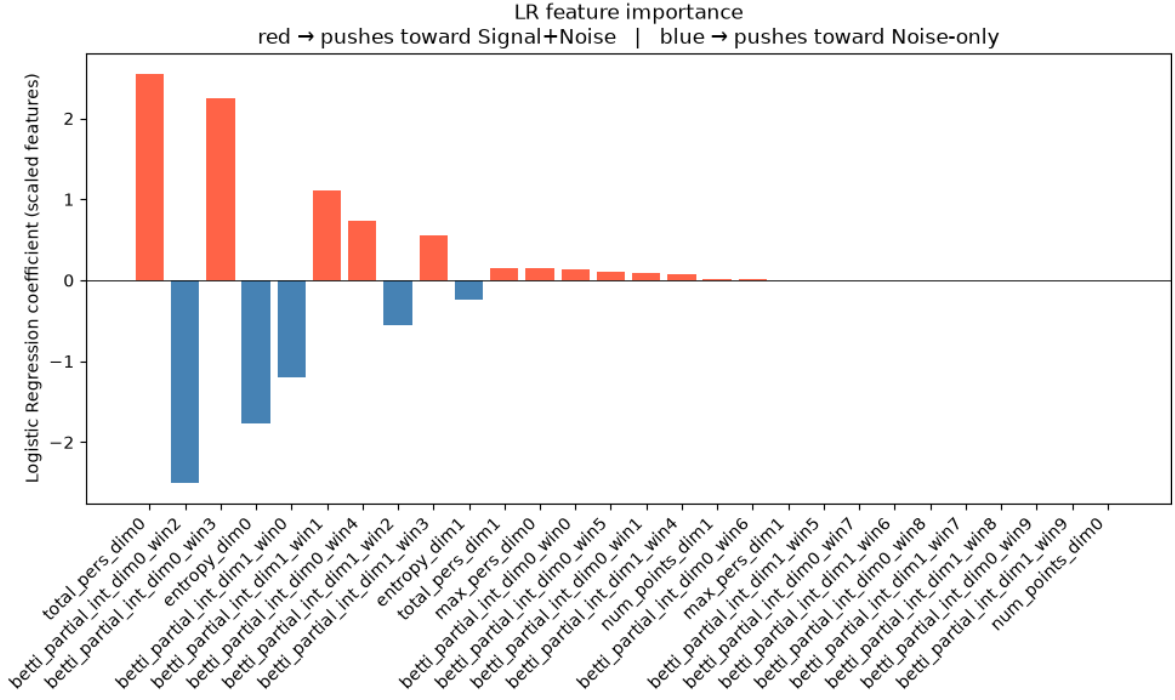


Figure 6.18: The fitted logistic-regression weights, sorted by magnitude. Bars above the axis push the decision towards the signal class, bars below it towards noise-only. As with the forest, a few features carry almost all of the weight.

The two rankings agree. The three largest weights belong to the total persistence in dimension H_0 (+2.55), the H_0 Betti-curve area in the third window (-2.50) and in the fourth (+2.25) — the same three features that also the random forest point as important. Just below those features, in fifth and sixth place, sit two H_1 features: the Betti-curve area of dimension H_1 in the first window (-1.20) and in the second (+1.10) — so the loops discussed above are visible and important also here. The logistic regression, a linear combination of scaled features, and the random forest, an ensemble of decision trees, are different objects fitted by different procedures. Their concurring on which handful of topological summaries matters is therefore evidence about which features actually carry the distinction between the two classes, GW signal versus noise-only.

This agreement comes with a caution, though: the signs in Figure 6.18 should not be read one at a time, and Figure 6.19 says why.

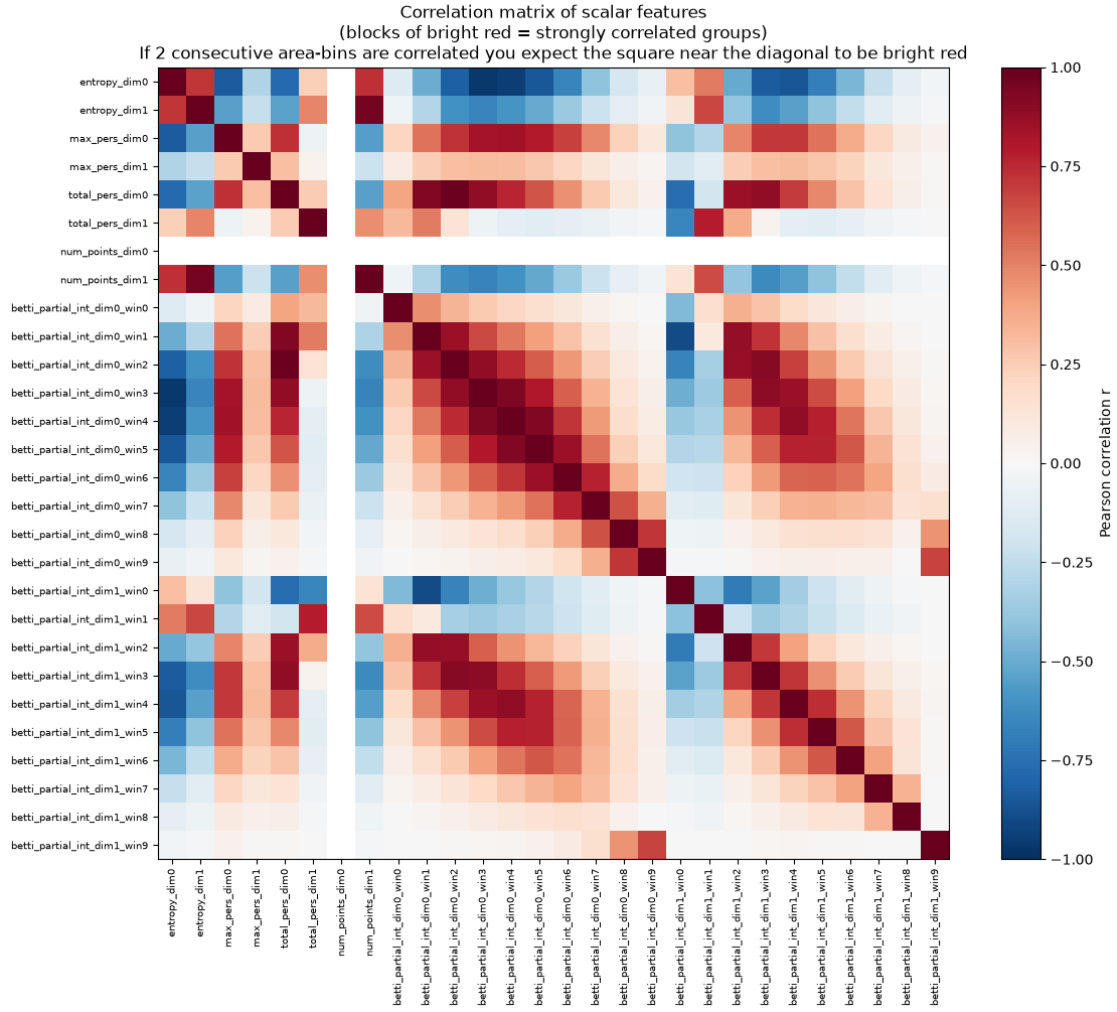


Figure 6.19: Correlation between every pair of the 28 scalar features.

Consecutive windows of the same Betti curve are strongly correlated — in dimension H_0 the average correlation between one window and the next is +0.78, reaching +0.94 for some pairs — which is only to be expected, since neighbouring windows integrate the same slowly varying curve over adjacent widows. Some correlations cut across the different kinds of summary as well: the persistent entropy in dimension H_0 and the Betti-curve area in the fourth window have a correlation of -0.96 , so the two are very nearly the same measurement with opposite sign.

When two features carry nearly the same information, the fitted weights are free to distribute themselves between them almost arbitrarily — one large positive weight against one large negative weight can produce the same predictions as two moderate ones²² — so an individual coefficient is not a stable quantity.

Why the linear model is enough

An unexpected result. Logistic regression is a *linear* classifier²³ — the simplest of the five models used in this chapter, with far fewer quantities to fit than an ensemble of trees. Given exactly the same 28 topological features (Table 6.6), it nonetheless performs on a par with a much more elaborate, non-linear model such as gradient boosting: their validation accuracies are 0.836 against 0.839, and their areas under the ROC curve 0.901 against 0.900, differences of a few thousandths in both cases (Table 6.7). Section 6.2.2 already flagged this as not what one would expect, and it is worth asking why.

What this means geometrically. A linear classifier can only do one thing: draw a single flat cut through feature space and put every example on one side of it or the other. It cannot bend around a

²²To see precisely why, and a worked example of it, see Appendix A.46.

²³As discussed in Section 6.1.2: it separates the two classes with a hyperplane in feature space.

class, or wrap itself around a cluster, the way a random forest or a gradient-boosted ensemble can. So if a linear classifier matches in performance with those more flexible models, the two classes it is separating must already be sitting in positions where a single flat cut is nearly enough to tell them apart — the two classes, gravitational-wave time series and noise-only time series, are close to linearly separable once expressed as these 28 topological numbers.

This says something about the pipeline built over the course of this chapter, not just about logistic regression. Section 5.2 through Section 5.3 took each raw, noisy time series and turned it into 28 topological numbers, by embedding it, computing its persistent homology, and summarising the result. Separating a gravitational-wave signal from detector noise is a genuinely hard problem, and turning the raw time series into these 28 numbers is where that hardness is resolved: once this is done, what remains is close to the easiest kind of classification problem there is, separating two classes with a straight cut. In other words, detecting a GW wave inside the noise became (approximately) a linear classification problem.²⁴

The first check is to examine the 20 000 time series directly, as points in the 28-dimensional feature space, and ask whether the two classes are actually separable by a hyperplane there. A 28-dimensional space cannot be examined visually, so the usual way around this is to project it down to the few dimensions we can plot. We use principal component analysis (Section 2.3.5) for this, since it projects onto the directions along which the data varies most, in the hope that whatever linear separation survives the projection. Figure 6.20 shows the result, projected down to the three directions of greatest variance.

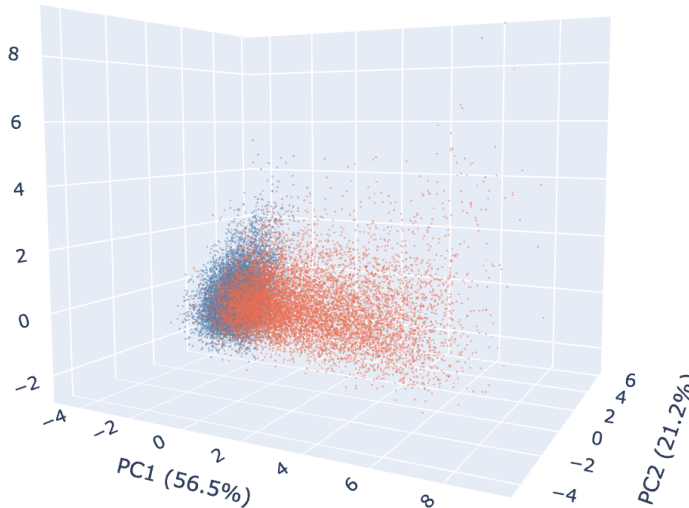


Figure 6.20: The 20 000 time series projected onto the first three principal components of the scaled feature space, coloured by class (blue: noise-only, orange: signal). Together these three directions carry 89.8% of the variance.

These three directions already carry most of the variance: 56.5%, 21.2% and 12.0%, 89.8% together, so this is not a case of looking at some minor corner of the data. The two classes (red and blue)²⁵ are not cleanly split apart in this view — there is a dense region where both colours are present together — but they are not on top of one another either: the noise-only class stays concentrated close to the origin, while the signal class spreads out much further along all three directions. This picture alone does not settle whether the classes are linearly separable, though: even if they really are separable in the full 28-dimensional space, this projection down to three directions could still make them look mixed together, for a reason explained next.

The reason is that principal component analysis is *unsupervised*: it finds the directions along which the combined cloud of both classes is most spread out, and it never looks at the labels of the classes. There is

²⁴Only *nearly*, not exactly: a straight cut is not a perfect description of the two classes even after the TDA pipeline, since Section 6.2.2 already showed that none of the five classifiers, including the non-linear ones, gets every example right. Some curvature separating the classes remains that a hyperplane cannot capture, but it is little enough that a random forest or a gradient-boosted ensemble, both free to bend around it, gain almost nothing from being able to.

²⁵The two classes are, as usual, the class of timeseries that has the GW signal immersed in the noise and the class of timeseries composed only by noise.

no reason for the direction of greatest spread (which is the direction PCA projects onto, Section 2.3.5) to be the direction that separates the two classes, and it commonly is not. Figure 6.21 shows a small example of exactly this: two classes, each drawn as a cloud elongated in the same diagonal direction and offset only slightly, perpendicular to that elongation. Most of the spread in the combined cloud comes from how far each individual cloud stretches out along the diagonal, so that is the direction PCA picks out as PC1 — but sliding along that direction stays inside one class the whole way, so it does not separate anything. The much smaller, perpendicular direction is where the two classes actually differ, and it is this direction, not the one of greatest variance, that tells the two classes apart. PCA has no way to know this in advance, since it never looks at which points belong to which class; it simply ranks directions by how much the data spreads out along them.

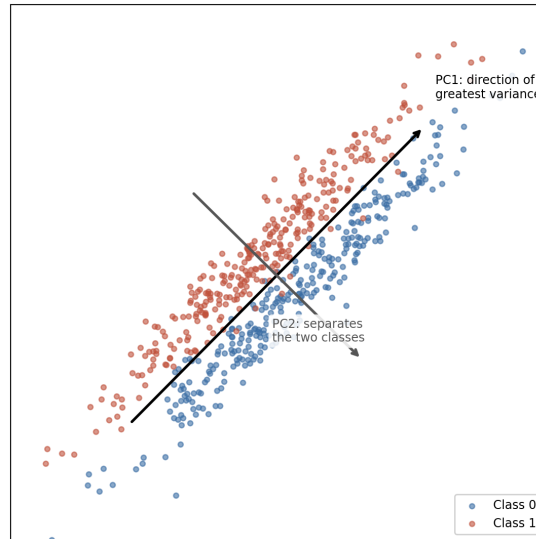


Figure 6.21: A synthetic illustration, unrelated to the gravitational-wave data, of why the direction of greatest variance need not separate two classes. Both clouds are stretched along the same diagonal and offset only slightly, perpendicular to it. PCA’s first component (PC1) runs along the stretch and mixes the two classes; the much shorter second component (PC2) is the one that actually separates them.

The same observation was already made briefly in Section 5.3.2 about the point clouds themselves.

So we should instead look along the direction the classifier actually uses. Logistic regression reduces each time series to the single number $\mathbf{w}^\top \mathbf{x} + b$ — the weighted sum of its features (eq. (6.1)). This one number is all the classifier looks at: if it is positive, the classifier answers signal; if it is negative, it answers noise-only; the decision boundary itself, the hyperplane of Section 6.1.2, is exactly where this number is zero. Unlike the principal components, which are chosen without regard to the labels, this one-dimensional projection was chosen specifically to make that sign as informative as possible about the class. Figure 6.22 shows how the two classes are distributed along it.

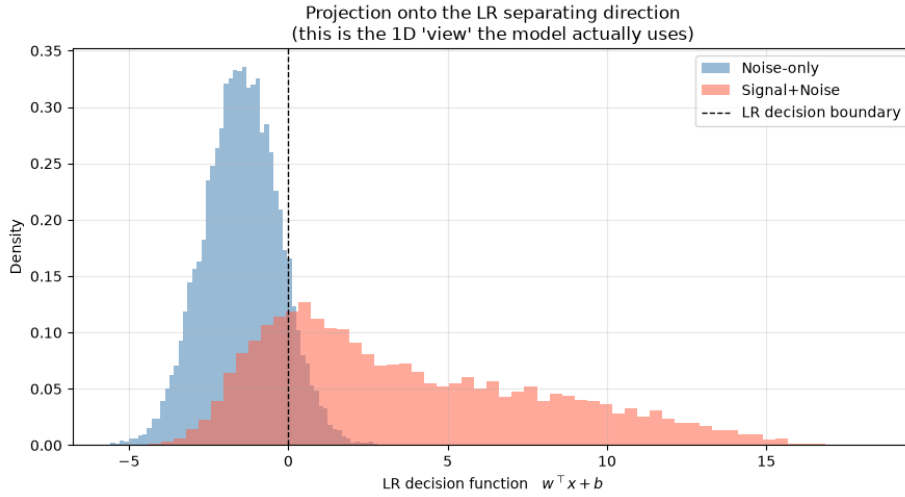


Figure 6.22: The two classes distributed along the single direction the fitted logistic regression uses, with its decision boundary at zero. The noise-only distribution is compact and lies mostly to the left of the boundary; the signal distribution is broad, with a long tail extending far to the right, and overlaps the noise only near the boundary itself.

Along this direction the two classes are plainly distinguishable. The noise-only distribution is narrow and centred well to the left of the boundary; the signal distribution is much broader and reaches far to the right, out to values the noise class never attains. Looking at these two histograms, the classes really are close to linearly separable in these topological features — a hyperplane is simply enough to describe the distinction, and there is little left for a more flexible model to add. That is the explanation for the near-identical performance of the three classifiers in Table 6.7.

6.2.5 Detection sensitivity as a function of signal strength

The signals in this dataset were injected with target SNRs drawn anywhere in $[6, 30]$ (Table 6.4), a range spanning signals that are loud and signals that are barely there at all, and the question that we examine now is how weak can a signal be to still be detected by the machine learning classifiers.

This is answered by grouping the signal-bearing time series by the SNR and computing, within each group, the fraction that the classifier correctly identifies. That fraction is the recall of the signal class, restricted to one band of SNR at a time; in this context it is usually called the *sensitivity*. As with every other result in this section, this fraction is computed under cross-validation (Section 6.1.3).

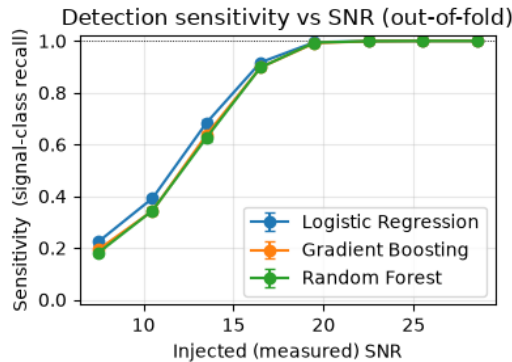


Figure 6.23: The fraction of injected signals recovered, against the SNR they were injected with, for the three feature-vector classifiers. Error bars give the uncertainty from the number of signals in each band. The curve rises steeply through the middle of the range and saturates once the signals are loud enough.

Figure 6.23 shows the result, with the SNR range divided into eight bands. Its shape is the expected one for a detector, and it is steep. In the lowest band, around an SNR of 7.5, only about one signal in five is recovered. Detection improves rapidly from there: by an SNR of around 13.5 roughly two thirds

are found, and by an SNR of about 20 every GW signal is detected. The three classifiers are again almost indistinguishable, with logistic regression very slightly ahead of the other two in the lowest bands.

Taken together, the results of this section say something reasonably definite. The topological summaries of Section 5.3 do carry the presence of a gravitational wave: three quite different classifiers independently extract the same information from them, agree on which summaries matter, and recover essentially every signal above a moderate SNR.

6.3 Comparing the cost against matched filtering

Matched filtering, derived in Section 4.4, is the state of the art in gravitational-wave detection and is the method the LIGO–Virgo–KAGRA detectors actually run: correlating the data against a known waveform is provably the best one can do when that waveform is known and the noise is Gaussian. One of the weakness of this approach is its high computational cost. The masses of the two bodies are not known in advance, so the search cannot use a single waveform; it must try a whole *bank* of them, some 10^5 templates covering the range of masses a source might plausibly have, and each one of those templates is a separate filtering of the data (Section 4.7.2). The cost of a search therefore grows in proportion to the size of the bank.

6.3.1 Measuring the difference

Both methods were evaluated on the same task: deciding, from one four-second time series sampled at 4096 Hz, whether a gravitational wave is present. The time series is the same in both cases — the same injected waveform in the same realisation of noise — and both methods were timed on the same computer, with each restricted to a single processor core so that the two numbers are comparable.

The two costs behave quite differently as the bank grows. Matched filtering was timed against banks of 10, 25, 50 and 100 templates, and the measured times fall on a straight line: each additional template adds a fixed cost of about one millisecond, because each one is an independent correlation with the data. The topological pipeline, by contrast, does not have a bank at all. It embeds the time series once, computes the persistent homology of the resulting point cloud once, and reads off the features once, and none of that work depends on how many source parameter combinations a bank would have had to cover. Its cost is a flat line.

Two lines with different slopes cross somewhere and we call that crossing point N^* , the bank size at which matched filtering and the topological pipeline cost the same; below it matched filtering is cheaper, above it the topological pipeline is cheaper. Table 6.11 gives the measured costs and the resulting crossover for three sizes of point cloud, since the size of the cloud is one of the quantities that governs how expensive the persistent homology is.

The number of points in the cloud is not itself a free parameter we set directly: it is a consequence of the *stride* of the Takens embedding, introduced in Section 5.2.2. Recall that the stride controls how many of the delay vectors built by the embedding are kept — keeping every one of them gives the largest, densest cloud, and keeping only every s -th one gives a cloud roughly s times smaller. A coarser stride discards more of the embedded trajectory and gives a sparser cloud, which is cheaper to run through a Rips filtration but a rougher description of the trajectory’s shape; a finer stride keeps more of embedded trajectory, at a correspondingly higher cost. This is the same trade-off already met when the dense and coarse embeddings were introduced in Section 5.2.2, and Table 6.11 shows what it costs in wall-clock time rather than just in the number of points.

Points in the cloud	Cost per time series	Crossover N^*
1313 (stride 12)	0.25 s	250
2625 (stride 6)	1.20 s	1200
5249 (stride 3)	7.73 s	7700

Table 6.11: Cost of classifying one four-second time series with the topological pipeline, for three point-cloud sizes obtained by varying the embedding stride, together with the crossover bank size N^* at which matched filtering becomes the more expensive of the two. The last row is the configuration used everywhere else in this chapter. Crossover values are rounded.

At the configuration used throughout this chapter — a point cloud of 5249 points — the topological pipeline takes 7.73 seconds per time series (on one machine), and the crossover sits at roughly 7700 templates. So the topological pipeline becomes cheaper than matched filtering once the bank grows beyond 7700 templates, but real banks used to do GW detection are much larger: a production search of the kind described in Section 4.6.4 uses on the order of 250 000 templates.²⁶

We can use Figure 6.24 to estimate how long detecting a gravitational wave with a bank of that realistic size, 250 000 templates, would actually take on a single machine. The figure plots, on a log-log scale, the cost of classifying one time series against the size of the bank: the rising line is matched filtering, measured directly up to 100 templates (circles) and then extrapolated as a straight line, since its cost grows in fixed steps per template; the flat lines are the topological pipeline at the three point-cloud sizes of Table 6.11, which do not depend on the bank at all. Reading the matched-filtering line at 250 000 templates — far to the right of the measured points, but still on the same straight line — gives a cost of about 250 seconds per time series. This is the number a real search needs, not an arbitrary one: matched filtering only recovers a signal if the bank contains a template close enough to the true waveform buried in the noise, as discussed in Section 4.7.1, so using a bank smaller than the realistic one would not be a fair comparison. Against that 250 seconds, the topological pipeline still costs 7.73 seconds: topology is, in this setup, about thirty times faster than matched filtering run against a bank large enough to be used in practice.

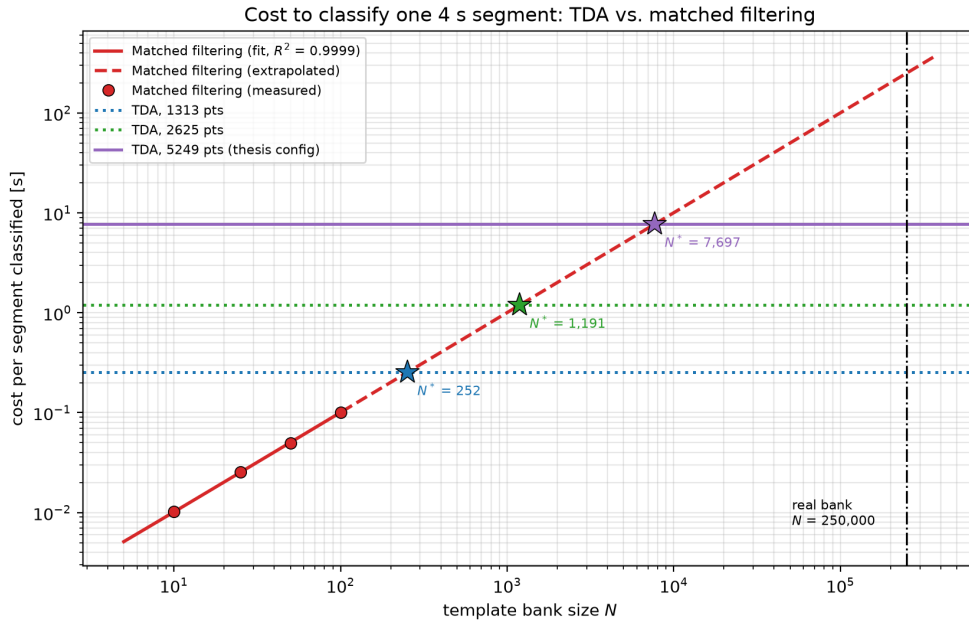


Figure 6.24: Cost of answering “is a gravitational wave present?” for one four-second time series, against the size of the template bank. The rising line is matched filtering, measured at the four bank sizes marked with circles and extrapolated beyond them as a dashed line; the horizontal lines are the topological pipeline at three point-cloud sizes, flat because its cost does not depend on the bank. Stars mark the crossover bank sizes of Table 6.11, and the vertical line marks the size of a real production bank. Both axes are logarithmic.

²⁶Such large banks are needed because the parameters of a coalescing binary — the two masses, and, in a more complete search, the spins — are themselves multi-dimensional and unknown in advance, so the bank must be dense in every one of those directions at once; Section 4.6.4 discusses this further.

6.3.2 What the speed does not buy

Must be noticed that matched filtering performs GW detection and also estimates the parameters of the source. Because the search records *which* template produced the loudest correlation, and each template corresponds to a definite pair of masses, the same computation that detects the signal also estimates the parameters of the source that produced it. While the topological pipeline simply detects the presence of a gravitational wave, without performing parameter inference. Estimating physical parameters from topological summaries is not something that follows immediately from what we have built, and we have not attempted it in this thesis.

A possible way to perform parameter inference with this topological pipeline is the following. Deep neural networks trained directly on the noisy time series — with no templates and no topology — have been shown to detect compact-binary chirps with performance comparable to matched filtering, and to estimate the parameters of the source at the same time [23]. Inserting topological features into a pipeline of that kind can lead to:

- **Cheaper training.** A convolutional network given persistent-homology features alongside the raw time series is shown to require less training, and to need a significantly smaller training set to reach comparable performance, than the same network given the time series alone [24].
- **Parameter estimation that already works well.** Evidence that this route can succeed is not only hypothetical: [23] report that, for high SNR, their deep-learning predictor reaches a mean relative error on the estimated masses below 10%, while matched filtering restricted to the same discrete template bank stays above 10%. What this chapter adds to that picture is evidence that topological features specifically carry real information about the presence of a gravitational wave: three quite different classifiers, none of them elaborate, all detect the GW presence from the same topological summaries, and agree with one another about which topological features matter (Section 6.2.4) in GW detection. The promising preliminary results of this thesis therefore suggest a possible route to a topological pipeline that both detects and estimates the parameters of a gravitational wave.

Conclusions

In this thesis we addressed the problem of gravitational wave detection, with an interdisciplinary approach combining general relativity, geometry and computer science. In particular, we developed an approach based on topological data analysis and machine learning to implement a novel algorithm for the discrimination of Gravitational Wave signal generated by black-hole binary merging. Detecting a gravitational wave is a challenging problem. A passing gravitational wave changes the length of an interferometer’s arms by a fraction of their length so small that the resulting displacement is a small fraction of the size of a proton [12], and the resulting signal sits far beneath the instrument’s noise.

Therefore our motivating question was whether the shape of gravitational-wave data, measured topologically, carries enough information to detect gravitational wave signals.

To reach that goal, we relied on the mathematical properties of topological data analysis, and in particular of persistent homology. **Topological data analysis is a set of tools**, borrowed from a branch of mathematics known as algebraic topology, **for studying the *shape* of data**. Its central tool is persistent homology, built on the observation that a collection of measurements often carries a structure that is geometric rather than statistical: the data points cluster together, close up into loops, or enclose empty regions. These are qualitative features of the data and a conventional analysis based on averages and correlations could miss them [15].

Detecting gravitational waves with topological data analysis required us to study two further elements. The first was the core formalism of general relativity, needed as the basis for defining gravitational waves. The second was the field of gravitational-wave data analysis: the research area concerned with extracting astrophysical information from the signals recorded by interferometric detectors such as LIGO, Virgo and KAGRA. Its central challenge is that the gravitational waves carried by these signals are extremely weak, and sit buried in instrumental noise. This field focuses also on **detection: identifying the presence of an astrophysical signal in the data**, typically through techniques such as matched filtering, which is suited to well-modelled sources such as the coalescence of compact binaries.

Deciding whether a gravitational wave is present is a further step, and the one taken here is machine learning. Machine learning (ML) is the branch of artificial intelligence concerned with systems that learn patterns from data [21]. In particular, **a machine-learning model adjusts its own internal parameters until it captures the patterns that the data contain, aiming not to reproduce the examples it was shown but to perform well on data it has never seen**. One family of machine learning is supervised learning, which learns from labelled examples [21]: each training example is an input together with the correct answer for it, and the model is adjusted until its own answers agree with those provided. Classification is the task of selecting an answer from a fixed set of alternatives, such as determining whether a stretch of data contains a signal [22].

In this thesis **we combined these three fields of research** — gravitational-wave data analysis, persistent homology and machine learning classification — **to develop a novel algorithm, or pipeline, for the detection of a gravitational wave immersed in noise**. We implemented this pipeline, along with the dataset-generation and analysis code that supports it, entirely in Python, in more than 7 000 lines of code.

We first simulated the gravitational waveforms ourselves, using PyCBC [27], an open-source Python package for gravitational-wave data analysis built to detect and study the mergers of black holes and neutron stars. PyCBC allows us to generate as many waveforms as we needed, each one with its own values for the parameters of the coalescing binary.

We then embedded each simulated waveform in noise, itself simulated to reproduce the noise of the LIGO interferometers. In particular we have built a **GW dataset that tries to mimic the real data**

of an interferometer. For this reason, the masses of the two bodies, whose merger produces the wave, were drawn at random for every waveform, so no source was ever generated twice. Each waveform was also embedded at a randomly chosen instant within its stretch of noise, and at a randomly chosen amplitude, so that the dataset held both strong and weak gravitational waves. The dataset was then put through the standard preprocessing of gravitational-wave data analysis [25, 28]. Its purpose was to flatten the noise spectrum, so that no band of frequencies dominated artificially and the injected waveform could emerge as clearly as possible.

With the GW dataset ready, we applied the topological methods described above. **Each time series in the dataset** — a stretch of interferometer noise, with or without a gravitational-wave signal buried in it — **was turned into a cloud of points in $\mathbb{R}^n$ by an algorithm known as the *Takens embedding*** [29]. The resulting point cloud had a shape, and that shape depended on what the time series was doing. A gravitational-wave signal, whose frequency and amplitude grow as the two bodies spiral together, produced a cloud that wound into a spiral. A stretch of pure noise produced a shapeless ball. The two clouds differed in a way that was topological rather than numerical, and it was that difference we set out to detect.

We then applied persistent homology to each point cloud, which recorded how the points were arranged: where they gathered into clusters, where they closed up into loops, where they surrounded an empty region. Concretely, we implemented the *Rips filtration* and applied it to each point cloud. Around every point we drew a ball of radius r , and whenever the balls of two nearby points intersected, we connected the corresponding points. The radius started at zero and was then increased: at the beginning every point stood on its own, and as r grew the points progressively connected, forming the clusters, loops and cavities described above. For very large radius r all the points were connected in a single cluster and all the loops and cavities get filled by the connections: loops and cavities disappeared. **Persistent homology then detected these structures** — by means of linear algebra, through the reduction of a matrix — **and reported, for each topological feature (like a cluster, a loop, or a cavity), how long it survived as the radius grew.** In particular we found that **a gravitational wave produces a point cloud that winds into an elongated spiral**, whose loops persistent homology detected and reported as long-lived features, **while a stretch of pure noise produced a shapeless ball** with little structure, whose clusters and loops were all short-lived. The final result of persistent homology is a collection of numbers and images — we refer to them collectively as *topological summaries* — that summarised the shape of the cloud that each time series produced.

Finally, we gave these topological summaries as input for learning to machine-learning classifiers. We considered a range of classifiers, from logistic regression — a simple and linear model — to convolutional neural networks, which required images as input. What each of those ML models learned, in the end, was a statement about shape: that a gravitational-wave signal arranges the points of its cloud in particular ways, gathering them into clusters and closing them into loops. Once trained, the machine-learning classifiers were shown time series they had never met before, and on these they detected the presence of a gravitational wave reliably: most of the ML models we trained reached a **detection accuracy of about 83%**, on a dataset built from waveforms of variable mass and a signal-to-noise ratio sampled at random from a broad interval, from 6 (a very faint gravitational wave) to 30 (a strong gravitational-wave signal). In particular, for the models we trained we drew the ROC curve: a curve that shows, for every possible decision threshold, how well a classifier tells signal and noise apart. **This ROC curve confirmed that the topological pipeline detects gravitational waves reliably:** the curve traced out by our models came reasonably close to the ideal classification curve. Indeed, using a plot of sensitivity against signal-to-noise ratio, we found that **this topological approach detects gravitational-wave signals reliably in the range of signal-to-noise ratio** [15, 20], while it can still detect signals of lower signal-to-noise ratio [7, 15], though with reduced performance. A further interesting result of this thesis was that a linear machine-learning classifier, logistic regression, detected the presence of a gravitational wave with a performance comparable to that of more sophisticated machine-learning methods. A linear classifier is one that separates two classes of data (GW signal vs. pure noise) by drawing a straight line, or more generally a hyperplane, in the space of the topological summaries, cutting it as cleanly as possible into the regions belonging to each class. We understood from this that, **once topological data analysis has been applied, the classification of a gravitational wave becomes approximately a linear classification problem.**

In particular we also investigated which topological summaries were the most discriminative in gravitational-wave detection, according to the ML classifiers. **The most informative features were the topological summaries built on the H_0 Betti curve**, the curve that counts the number of components, while the loops of dimension H_1 were still informative but markedly less so. From the fact that **two machine learning models with nothing in common singled out the same topological features for gravitational-wave detection**, we understood that what they were reading was a real property of the data, and not an artefact of one fitting procedure.

We also compared the computational cost of this topological approach against matched filtering, the state of the art for gravitational-wave detection, timing both methods on the same task and the same machine. Matched filtering’s cost grows with the size of its template bank, since every template is a separate correlation with the data, and detecting a gravitational wave this way requires a bank of order 10^5 templates. **The topological pipeline, instead, has no bank**: it embeds the time series, computes its homology, and reads off the topological summaries once. Extrapolating our timings to a realistic bank size, seems to suggest that the topological pipeline could be a more efficient approach to gravitational-wave detection compared to matched filtering.

Summary of the main results

In summary, the main achievements of this thesis are the development and evaluation of a complete topological pipeline for gravitational-wave detection; the construction of a diverse simulated dataset containing binary-black-hole signals with varying component masses, arrival times and signal-to-noise ratios; and the demonstration that persistent-homology summaries contain discriminative information capable of separating gravitational-wave signals from noise.

- Across several machine-learning classifiers, the proposed approach achieved a detection accuracy of approximately 80%, performing reliably for signal-to-noise ratios between 15 and 20 while retaining some sensitivity to weaker signals.
- Notably, logistic regression performed comparably to more sophisticated models, suggesting that a linear decision rule already captures nearly all of the discriminative information the topological summaries contain.
- The analysis also identified summaries derived from the H_0 Betti curve (the curve of connected components) as the most informative features, while H_1 features (related to loops) provided additional, though less pronounced, information.
- Finally, we compared the computational cost of the topological pipeline against matched filtering, the state-of-the-art method for gravitational-wave detection. The topological approach avoids the large template banks required by matched filtering, and extrapolating our own timings to a realistic bank size seem to indicate that topological data analysis could be a more efficient approach.

Taken together, these results provide evidence that topological data analysis offers an effective, interpretable and computationally promising complementary approach to gravitational-wave detection.

Possible future developments

Since this thesis work encompassed several topics, there are many possibilities to develop it further.

From a binary answer to parameter inference. Matched filtering is also able to perform parameter inference, namely this method can estimate the parameters of the source. Our topological pipeline, on the other hand, produces only a binary decision: signal present or absent. Extending it is the most natural next step, and the two halves of the idea have each already been demonstrated separately: neural networks trained directly on noisy detector data detect chirps and estimate the parameters of the source at the same time [23], and topological features supplied to such a network are known to reduce the needed training set [24]. Therefore the idea is to place the topological summaries inside a machine learning pipeline built for parameter inference — the family of techniques known as *simulation-based inference*,

in which the posterior over source parameters is learned from simulated examples. The preliminary results of this thesis suggest that the topological summaries carry information about the presence of a gravitational wave signal, but whether they carry enough to constrain the source parameters is an open question (Section 6.3.2).

The regime where a generic method has the most to offer. The pipeline was tested here on binary coalescences. That choice was deliberate, and it was the conservative one: coalescences are the single class of source for which matched filtering is available, and therefore coalescence is the ground on which the two methods can be compared at all. They are also the class matched filtering handles best. The classes where matched filtering struggles are catalogued in Section 4.7.1: continuous waves, which are modelled in principle but applying naively matched filtering is computationally prohibitive [69]; bursts from core-collapse supernovae and several other plausible signals, which are not modelled well enough to use matched filtering, since the physical parameter space is too large to effectively cover with a template bank [70]. To these problematic scenarios of matched filtering should be added the problem of instrumental glitches (Section 4.7.3). Glitches are short, loud transients of instrumental or environmental origin — scattered-light arches, blips, whistles and many others — which are not random, formless noise: each family has its own repeated, recognisable shape in the time–frequency plane, and a glitch can produce a spurious high-SNR trigger. A method like TDA that reads generic shape rather than a specific waveform is, by construction, available in every one of these cases where matched filtering is not suitable, and this is where topological data analysis could give remarkable results.

Embedding a sliding window rather than the whole series. The ranking of the most important topological features for GW detection placed the loops of dimension H_1 below the connected components of dimension H_0 . Part of the reason is simply that a loop is a more fragile object than an isolated component: a component needs only one point, or a small group of points, to stay away from everything else, whereas a loop needs a whole ring of points to close up around empty space, with nothing wandering into the middle to fill it in. Noise is exactly the kind of disturbance that puts a stray point where it should not be (Section 6.2.4).

There may also be a geometric reason. The point cloud of a pure cosine is a single loop lying flat in one plane, because the cosine’s frequency never changes; the point cloud of a GW chirp is a tangle of loops at many different orientations, none of them confined to a single plane, because the chirp’s frequency is constantly changing. A loop that is not flat may be harder for the Rips filtration to recognise cleanly than a flat one is. To solve this problem one could apply the Takens embedding not to the whole time series at once, but to a short sliding window of it. Within a window short enough, the chirp’s frequency barely changes, so the embedded loop should come out close to flat, as in the pure cosine case, and correspondingly easier for persistent homology to detect.

Acknowledgments

First of all, I would like to thank in particular my supervisors, Prof. Pierpaolo Mastrolia and Dr. Manoj K. Mandal, for proposing this project to me and for their continuous and present support throughout the study and implementation of this work. Moreover, I would like to thank PhD student Matteo Pegorin for his guidance and for the precious discussions we had on the subject of gravitational waves.

I am also grateful to Prof. Pierpaolo Mastrolia for encouraging and pushing me to take an active part in the scientific community, letting me attend two PhD schools and encouraging me to present the thesis project at the GRASS 2026 workshop, which I would also like to thank for giving me the opportunity to present it. I am also very grateful to Dr. Manoj Mandal for his support during the practical implementation of the work; our discussions were crucial to the success of the project.

A special thanks goes to my family and especially to my mother, who has always supported and encouraged me throughout the years, and at the same time I would like to thank my friends for always being there and for all the fun we had together over the years.

Appendix A

Deferred proofs and constructions

A.1 The formal definition of a metric

In Section 1.1.1 we work with spaces carrying a notion of distance without spelling out the axioms such a distance must satisfy; they are recorded here.

Definition A.1 (Metric space, [16]). A **metric space** (X, d) is a set X together with a function $d: X \times X \rightarrow [0, \infty)$ such that, for all $x, y, z \in X$,

- $d(x, y) = 0$ if and only if $x = y$;
- $d(x, y) = d(y, x)$ (symmetry);
- $d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality).

The function d is called a **distance** or a **metric**.

A.2 Every isometry is a homeomorphism

In Section 1.1.1 we claimed that a map satisfying the definition of an isometry automatically satisfies that of a homeomorphism. This proof is based on [16].

Recall the definition of continuity used implicitly in Definition 1.2: a map $f: X \rightarrow Y$ between metric spaces is **continuous** if for each $x \in X$ and each $\varepsilon > 0$ there exists $\delta > 0$ such that, for all $y \in X$,

$$d_X(x, y) < \delta \implies d_Y(f(x), f(y)) < \varepsilon \quad (\text{A.1})$$

Proposition A.2. *Every isometry $f: X \rightarrow Y$ between metric spaces is a homeomorphism.*

Proof. By hypothesis f is a bijection and $d_Y(f(x), f(y)) = d_X(x, y)$ for all $x, y \in X$. There are two things to check.

f is continuous. Fix $x \in X$ and $\varepsilon > 0$, and take $\delta = \varepsilon$. If $d_X(x, y) < \delta$ then

$$d_Y(f(x), f(y)) = d_X(x, y) < \delta = \varepsilon,$$

so eq. (A.1) holds. The two sides of the implication are in fact the same inequality, which is why no estimate is needed.

f^{-1} is continuous. Since f is a bijection, $f^{-1}: Y \rightarrow X$ exists. It is again distance-preserving: given $u, v \in Y$, write $u = f(x)$ and $v = f(y)$, so that

$$d_X(f^{-1}(u), f^{-1}(v)) = d_X(x, y) = d_Y(f(x), f(y)) = d_Y(u, v).$$

Hence f^{-1} is itself an isometry, and the previous paragraph applies to it verbatim.

So f is a continuous bijection with continuous inverse, which is Definition 1.2. □

The converse is false: a stretched ellipse and a circle, for instance, are homeomorphic but plainly not isometric, since a homeomorphism is free to change every distance.

A.3 The Euclidean topology

This continues Section 1.1.2, giving the formal construction of the Euclidean topology on $\mathbb{R}^n$ out of open balls, of which Section 1.1.2 only records the informal picture.

Definition A.3 (Open ball). Let $x \in \mathbb{R}^n$ and $r > 0$. The **open ball of centre x and radius r** , denoted $B(x, r)$, is defined as

$$B(x, r) = \{y \in \mathbb{R}^n : \|x - y\| < r\}.$$

Definition A.4 (Open sets of $\mathbb{R}^n$). Let $A \subset \mathbb{R}^n$ be a subset and $x \in A$. We say that A is **open around x** if there exists $r > 0$ such that $B(x, r) \subset A$. We say that A is **open** if for every $x \in A$, A is open around x . We denote the set of such open sets by $\mathcal{T}_{\mathbb{R}^n}$.

Proposition A.5. $\mathcal{T}_{\mathbb{R}^n}$ is a topology on $\mathbb{R}^n$.

The proof of this proposition can be found in [30].

A.4 Continuous images of connected components

This continues Section 1.1.2, giving the statement and proof of a lemma used there and in Appendix A.6.

Lemma A.6. *Let $f: X \rightarrow Y$ be continuous and let O be a connected component of X . Then its image $f(O) \subset Y$ is connected.*

Proof. Write $O' = f(O)$ and test it against Definition 1.7. So suppose O' were split as $O' = A \cup A'$ by two disjoint open subsets A, A' of Y .

Pulling back along f gives two open sets $f^{-1}(A)$ and $f^{-1}(A')$ of X , again disjoint, and they cover O :

$$O \subset f^{-1}(O') = f^{-1}(A \cup A') = f^{-1}(A) \cup f^{-1}(A').$$

As O is connected, one of the two must be empty, and hence so must the corresponding one of A, A' . No genuine splitting of O' exists, which is to say O' is connected. $\square$

A.5 Homeomorphic spaces have the same number of connected components

This continues Section 1.1.2, giving the formal definition of connected components. The proof that homeomorphic spaces have the same number of connected components (Proposition 1.8) is given in [30].

Definition A.7 (Connected components). Let $(X, \mathcal{T})$ be a topological space. If X can be written as a union

$$\bigcup_{1 \leq i \leq n} O_i = X$$

of n open sets $O_1, \dots, O_n$ that are non-empty, pairwise disjoint and connected, we say that X **admits n connected components**.

A.6 Homotopy equivalent spaces have the same number of connected components

Proof of Proposition A.25, following Tinarrage [30].

Proof. Let X and Y be topological spaces and let $f: X \rightarrow Y$, $g: Y \rightarrow X$ be a homotopy equivalence. The claim is that f matches the components of X with those of Y one for one.

The key step is to show that $g \circ f$ moves no component out of itself. Let $F: X \times [0, 1] \rightarrow X$ be a homotopy from $g \circ f$ to the identity, take $x \in X$, and let O be its component. The cylinder $O \times [0, 1]$ is connected, so Lemma A.6 makes its image $F(O \times [0, 1])$ connected as well. That image contains $O = F(O \times \{1\})$; but O was already the *largest* connected set around x , so the two coincide, $O = F(O \times [0, 1])$. Reading off the other end of the homotopy,

$$g \circ f(O) = F(O \times \{0\}) \subset F(O \times [0, 1]) = O.$$

With $g \circ f(O) \subset O$ in hand the count follows. Say X has components $O_1, \dots, O_n$ and Y has m of them, and suppose $m < n$. Then f must send two distinct components O_i and O_j into one and the same component of Y , whence $g \circ f(O_i)$ and $g \circ f(O_j)$ lie in a single component of X . That is impossible: the first is contained in O_i and the second in O_j , which are different components.

Exchanging the roles of X and Y rules out $m > n$ by the same argument, so $m = n$. $\square$

A.7 Geometric simplices and simplicial complexes

This continues Section 1.2.2, giving the formal definitions behind the geometric complexes drawn there.

Remark A.8 (Convex hull). For any collection of points $a_1, \dots, a_k \in \mathbb{R}^n$, we define their **convex hull** as

$$\text{Conv}(\{a_1 \dots a_k\}) = \left\{ \sum_{1 \leq i \leq k} t_i a_i, \quad t_1 + \dots + t_k = 1, \quad t_1, \dots, t_k \geq 0 \right\}. \quad (\text{A.2})$$

Definition A.9 (Geometric simplex, [16]). Let $k, d \in \{0, 1, \dots\}$ with $k \leq d$. A **geometric k -simplex** σ in $\mathbb{R}^d$ is the convex hull of an affinely independent family^a $V = \{v_0, v_1, \dots, v_k\} \subset \mathbb{R}^d$, that is, $\sigma = \text{Conv}(V)$ in the sense of Remark A.8.

^aA family $\{v_0, \dots, v_k\}$ is **affinely independent** when none of its members can be written as an affine combination of the others; equivalently, when the k difference vectors $v_1 - v_0, \dots, v_k - v_0$ are linearly independent [16]. In low dimensions this is the familiar general-position requirement: two points must be distinct, three must not be collinear, four must not be coplanar. It is what stops a triple of points from spanning a segment rather than a triangle.

A geometric simplex $\sigma = \text{Conv}(\{v_0, \dots, v_k\})$ carries the following vocabulary.

- Its **dimension** is $\dim(\sigma) = k$, which we occasionally record as a superscript, writing $\sigma = \sigma^k$.
- Its **vertices** are the points $v_0, v_1, \dots, v_k$, and σ is said to be **spanned** by them.
- Its **edges** are the convex hulls of pairs of vertices.
- If a simplex τ is spanned by a subset of the vertices of σ , then τ is a **face** of σ and σ a **coface** of τ . A face one dimension below σ , that is with $\dim(\tau) = \dim(\sigma) - 1$, is called a **facet**.

Assembling these blocks into larger shapes calls for one condition beyond the obvious one. Keeping all the faces is clearly necessary; but we must also forbid two simplices from cutting across each other, meeting in some region that is not a face of either.

Definition A.10 (Geometric simplicial complex, [16]). Let $d \in \{0, 1, \dots\}$. A **geometric simplicial complex** $L \subset \mathbb{R}^d$ is a finite collection of geometric simplices such that:

- if $\sigma \in L$ and τ is a face of σ , then $\tau \in L$;
- if $\sigma, \tau \in L$, then $\sigma \cap \tau$ is either empty or a common face of both.

A.8 The topological realization of an abstract complex

The construction sketched in Section 1.2.2 and carried out for one complex in Example 1.13, stated in general. The definition is Tinarrage [30]; only the notation is ours, for the reason given below.

Definition A.11 (Topological realization, [30]). Let K be an abstract simplicial complex, with vertex set $V = \{1, \dots, n\}$. In $\mathbb{R}^{n+1}$, consider, for every $i \in \{0, \dots, n\}$, the vector $e_i = (0, \dots, 1, 0, \dots, 0)$ (i^{th} coordinate 1, the other ones 0). Let $|K|$ be the subset of $\mathbb{R}^{n+1}$ defined as

$$|K| = \bigcup_{\sigma \in K} \text{Conv}(\{e_j, j \in \sigma\}), \quad (\text{A.3})$$

where Conv represents the convex hull (see Remark A.8). Endowed with the subspace topology, $(|K|, \mathcal{T}_{|K|})$ is a topological space, that we call the **topological realization** of K .

Two remarks on the statement.

The ambient space is $\mathbb{R}^{n+1}$ itself: the set $|K|$ of eq. (A.3) is a subset of it, $\mathbb{R}^{n+1}$ carries the Euclidean topology of Definition A.4, and the subspace topology is the one of Definition 1.6, namely $\mathcal{T}_{|K|} = \{O \cap |K| : O \text{ open in } \mathbb{R}^{n+1}\}$. Nothing new is introduced; the two definitions of Section 1.1.2 are applied to the set just built.

The indexing is slightly generous. With vertices labelled $1, \dots, n$ the vector e_0 never occurs in eq. (A.3), so $\mathbb{R}^n$ would already suffice, and Theorem 1.14 does better still with $\mathbb{R}^{n-1}$. The point of this construction is not economy of dimension but that it requires no verification: distinct basis vectors are affinely independent whatever subset one takes, so condition (b) of Definition A.10 holds automatically.

Finally, the notation. Tinarrage [30] writes $|K|$ here, and since 2026-08-01 this thesis uses K for an abstract complex too (see notation.md), so the symbol carries over unchanged. $|L|$ is reserved for Virk's *body* of a geometric complex (Section 1.2.2). The mathematics is unchanged.

A.9 The formal definition of a triangulation

This continues Section 1.2.2, giving the formal definition behind the intuitive notion of a triangulation stated there.

Definition A.12 (Triangulation, [30]). Let $(X, \mathcal{T})$ be a topological space. A **triangulation** of X is a simplicial complex K such that its topological realization $(|K|, \mathcal{T}_{|K|})$ is homeomorphic to $(X, \mathcal{T})$.

The topological realization $|K|$ of Definition A.11 is one specific choice of geometric embedding of K ; any other geometric realization of K (Definition A.10) is homeomorphic to it, since all are homeomorphic to K 's abstract structure alike. So this formal definition agrees with the intuitive one of Section 1.2.2: a triangulation of X is a simplicial complex whose geometric picture, however it is drawn in some $\mathbb{R}^n$, is homeomorphic to X .

A.10 Dimension of a quotient vector space

Let's prove that for a $\mathbb{Z}/2\mathbb{Z}$ -vector space V and a linear subspace W ,

$$\dim V/W = \dim V - \dim W.$$

The following proof is based on [30].

Proof. Write $n = \dim V$ and $m = \dim W$. Every finite $\mathbb{Z}/2\mathbb{Z}$ -vector space of dimension k is isomorphic to $(\mathbb{Z}/2\mathbb{Z})^k$ and so has cardinal 2^k ; hence V has cardinal 2^n and W has cardinal 2^m .

The cosets $v + W = \{v + w : w \in W\}$ of W in V partition V , and every coset $v + W$ is in bijection with W itself via $w \mapsto v + w$, so every coset has the same cardinal 2^m . Hence the number of cosets is

$$|V/W| = \frac{|V|}{|W|} = \frac{2^n}{2^m} = 2^{n-m}.$$

The quotient V/W , with $\oplus$ defined coset-wise by $(v+W) \oplus (v'+W) = (v+v')+W$, is itself a $\mathbb{Z}/2\mathbb{Z}$ -vector space, so the same cardinality fact applies to it as well: there exists $d \geq 0$ such that $|V/W| = 2^d$ and $d = \dim V/W$. Comparing with $|V/W| = 2^{n-m}$ gives $d = n - m$, that is,

$$\dim V/W = \dim V - \dim W.$$

□

A.11 The boundary of a boundary is zero

Proof of Lemma 1.20, based on [30].

Proof. Suppose $q \geq 2$, the result being trivial otherwise. Since the boundary operators are linear, it is enough to prove that $\partial_{q-1} \circ \partial_q(\sigma) = 0$ for every simplex $\sigma \in K_{(q)}$. By definition,

$$\partial_q(\sigma) = \sum_{\substack{\tau \subset \sigma \\ |\tau| = |\sigma| - 1}} \tau,$$

and

$$\partial_{q-1} \circ \partial_q(\sigma) = \sum_{\substack{\tau \subset \sigma \\ |\tau| = |\sigma| - 1}} \partial_{q-1}(\tau) = \sum_{\substack{\tau \subset \sigma \\ |\tau| = |\sigma| - 1}} \sum_{\substack{\nu \subset \tau \\ |\nu| = |\tau| - 1}} \nu.$$

We can write this last sum as

$$\sum_{\substack{\tau \subset \sigma \\ |\tau| = |\sigma| - 1}} \sum_{\substack{\nu \subset \tau \\ |\nu| = |\tau| - 1}} \nu = \sum_{\substack{\nu \subset \sigma \\ |\nu| = |\sigma| - 2}} \alpha_\nu \cdot \nu,$$

where $\alpha_\nu = |\{\tau \subset \sigma : |\tau| = |\sigma| - 1, \nu \subset \tau\}|$. It is easy to see that for every ν with $|\nu| = |\sigma| - 2$, we have $\alpha_\nu = 2 = 0$. □

A.12 Every boundary is a cycle

This continues Section 1.3, giving the proof of Corollary 1.21. This proof is based on [30].

Proof. Let $\mathbf{b} \in B_q(K; F)$. By definition, there exists $\mathbf{c} \in C_{q+1}(K; F)$ such that $\mathbf{b} = \partial_{q+1}(\mathbf{c})$. Using Lemma 1.20, we obtain

$$\partial_q(\mathbf{b}) = \partial_q \partial_{q+1}(\mathbf{c}) = 0,$$

hence $\mathbf{b} \in Z_q(K; F)$. □

A.13 Induced orientation of a facet

Section 1.3.1 promised the precise rule by which a choice of orientation on σ fixes one on each of its facets. We follow [16] for the statement.

Definition A.13 (Induced orientation). Suppose $\sigma = \langle v_0, v_1, \dots, v_k \rangle$ is an oriented simplex and $p \in \{0, 1, \dots, k\}$. Then the **induced orientation** of the facet of σ obtained by dropping v_p is

$$(-1)^p \langle v_0, v_1, \dots, v_{p-1}, v_{p+1}, \dots, v_k \rangle.$$

Oriented edge $\langle x, y \rangle$ induces orientations $\langle y \rangle$ and $-\langle x \rangle$ on its facets (vertices). Oriented triangle $\langle x, y, z \rangle$ induces orientations $\langle y, z \rangle$, $-\langle x, z \rangle$ and $\langle x, y \rangle$ on its facets (edges).

A.14 Parity of 0-boundaries

We follow Edelsbrunner and Harer [31] for this argument, used in Example 1.25.

Proposition A.14. *Let K be a connected simplicial complex with vertex set V , and let $S \subseteq V$. The 0-chain $\sum_{v \in S} \{v\}$ is a 0-boundary if and only if S has even cardinal. Consequently, exactly half of the 0-cycles of K are 0-boundaries.*

Proof. Every 1-chain is a sum of edges, and each edge contributes exactly two endpoints to the boundary. Taking the boundary of a sum of edges cancels duplicate endpoints in pairs, so the boundary of any 1-chain always consists of an even number of distinct vertices. This rules out every odd-cardinal S .

Conversely, suppose $S \subseteq V$ has even cardinal. Since K is connected, we may pair up the vertices of S arbitrarily and choose, for each pair, a path in K joining them. Summing the edges of all these paths into a single 1-chain $\mathbf{c}$, every vertex of S is the endpoint of exactly one path and so appears an odd number of times in $\partial_1 \mathbf{c}$, while every other vertex is either untouched or an interior point of some path and so appears an even number of times. Hence $\partial_1 \mathbf{c} = \sum_{v \in S} \{v\}$, and S is a 0-boundary.

Since ∂_0 is the zero map (Definition 1.17), every 0-chain is a 0-cycle, so the 0-cycles of K are exactly the subsets $S \subseteq V$. Among the $2^{|V|}$ subsets of a non-empty finite set, exactly half have even cardinal and half odd,¹ so exactly half the 0-cycles are 0-boundaries. $\square$

A.15 Homologous cycles

This continues Section 1.4, giving the definition of homologous chains and the computation, promised in Example 1.24, that its two non-trivial 1-cycles other than $\{0, 2\} + \{2, 3\} + \{0, 3\}$ enclose the same hole.

Definition A.15 (Homologous chains, [30]). Two chains $\mathbf{c}, \mathbf{c}' \in C_q(K; F)$ are **homologous** if there exists $\mathbf{b} \in B_q(K; F)$ such that $\mathbf{c} = \mathbf{c}' + \mathbf{b}$: in other words, if they are equal up to a boundary.

Recall from Example 1.24 the complex

$$K = \{\{0\}, \{1\}, \{2\}, \{3\}, \{0, 1\}, \{0, 2\}, \{1, 2\}, \{0, 3\}, \{2, 3\}, \{0, 1, 2\}\},$$

with 1-cycles $Z_1(K; F) = \{0, \{0, 1\} + \{1, 2\} + \{0, 2\}, \{0, 2\} + \{2, 3\} + \{0, 3\}, \{0, 1\} + \{1, 2\} + \{2, 3\} + \{0, 3\}\}$ and boundaries $B_1(K; F) = \{0, \{0, 1\} + \{1, 2\} + \{0, 2\}\}$.

The chains $\{0, 2\} + \{2, 3\} + \{0, 3\}$ and $\{0, 1\} + \{1, 2\} + \{2, 3\} + \{0, 3\}$ are homologous. Indeed,

$$\{0, 2\} + \{2, 3\} + \{0, 3\} = \{0, 1\} + \{1, 2\} + \{2, 3\} + \{0, 3\} + (\{0, 1\} + \{0, 2\} + \{1, 2\}),$$

and the term added on the right, $\{0, 1\} + \{0, 2\} + \{1, 2\}$, is exactly the non-trivial boundary of $B_1(K; F)$. Figure A.1 draws the two homologous cycles side by side.



Figure A.1: Two homologous 1-cycles of the complex of Example 1.24: they differ by the boundary of the filled triangle $\{0, 1, 2\}$.

A.16 Computing the Betti numbers of the square

This continues Section 1.4, giving the computation behind the Betti numbers stated in Example 1.25 for

$$K = \{\{0\}, \{1\}, \{2\}, \{3\}, \{0, 1\}, \{1, 2\}, \{2, 3\}, \{0, 3\}\}.$$

¹Expand $(1 + 1)^{|V|}$ and $(1 - 1)^{|V|}$ by the binomial theorem: the first equals $2^{|V|}$, the sum of all subset counts, and the second equals 0, the alternating sum of even-cardinal counts minus odd-cardinal counts. Adding and subtracting the two identities gives both sums equal to $2^{|V|-1}$.

K is built from 0-chains, such as $\{0\} + \{1\}$, and 1-chains, such as $\{1, 2\} + \{2, 3\}$. It has no 2-chains beyond the null one, since K contains no 2-simplex.

- $\beta_2(K)$. Since K has no 2-simplex, $C_2(K; F)$ contains only the null chain 0, hence so do $Z_2(K; F) \subset C_2(K; F)$ and $B_2(K; F) \subset Z_2(K; F)$ (Corollary 1.21). Both have cardinal $1 = 2^0$, so $\dim Z_2(K; F) = \dim B_2(K; F) = 0$, and

$$\beta_2(K) = \dim Z_2(K; F) - \dim B_2(K; F) = 0 - 0 = 0.$$

- $\beta_1(K)$. Computing as in Example 1.19, the 1-cycles $Z_1(K; F)$ (namely the 1-chains with zero boundary) are

$$Z_1(K; F) = \{0, \{0, 1\} + \{1, 2\} + \{2, 3\} + \{0, 3\}\} :$$

the null chain, and the sum of all four edges, whose boundary is zero since every vertex of the square lies on exactly two of its edges, so each term cancels in pairs. So $|Z_1(K; F)| = 2 = 2^1$, giving $\dim Z_1(K; F) = 1$. And $B_1(K; F) = \text{im}(\partial_2)$; since K has no 2-simplex, $C_2(K; F)$ is trivial, so its image $B_1(K; F)$ is trivial too, $\dim B_1(K; F) = 0$. Hence

$$\beta_1(K) = \dim Z_1(K; F) - \dim B_1(K; F) = 1 - 0 = 1.$$

- $\beta_0(K)$. Since ∂_0 is the zero map (Definition 1.17), every 0-chain is a 0-cycle, so $Z_0(K; F) = C_0(K; F)$, giving $|Z_0(K; F)| = 2^4 = 16$ and $\dim Z_0(K; F) = 4$. Since K is connected, Proposition A.14 says that exactly half of its 0-cycles are 0-boundaries, so $|B_0(K; F)| = 8 = 2^3$ and $\dim B_0(K; F) = 3$. Hence

$$\beta_0(K) = \dim Z_0(K; F) - \dim B_0(K; F) = 4 - 3 = 1.$$

A.17 The zeroth Betti number counts components

The proof of Proposition 1.28 is due to Botnan [33], and uses his own notion of path-connectedness (his Def. 2.12):

Definition A.16 (Path-connected). We say that a simplicial complex K is path-connected if any two vertices in K are connected by an edge path.

Proof. Assume that K is path-connected, and let $\{p_1, \dots, p_m\}$ denote the vertex set of K . A basis for $Z_0(K; F)$ is given by $\{p_1\}, \{p_2\} + \{p_1\}, \dots, \{p_m\} + \{p_1\}$. Since K is path-connected, we can find, for every $j \neq 1$, a $\mathbf{c} \in C_1(K; F)$ such that $\partial_1(\mathbf{c}) = \{p_1\} + \{p_j\}$. This shows that $\beta_0(K) \leq 1$. To show that $\beta_0 = 1$, observe that a vertex cannot be in the image of ∂_1 : for any $\mathbf{c} \in C_1(K; F)$, $\partial_1(\mathbf{c})$ is either 0 or a non-trivial sum of an even number of vertices. Hence, $\beta_0(K) = \dim Z_0(K; F) - \dim B_0(K; F) = 1$.

For the more general case, observe that if $K = K_1 \cup K_2$ where K_1 and K_2 are disjoint, then the boundary operator decomposes as a direct sum

$$\partial_1: C_1(K; F) = C_1(K_1; F) \oplus C_1(K_2; F) \xrightarrow{\partial_1^1 \oplus \partial_1^2} C_0(K_1; F) \oplus C_0(K_2; F) = C_0(K; F).$$

Where $\partial_1^i: C_1(K_i; F) \rightarrow C_0(K_i; F)$ is the first boundary operator for K_i . In particular,

$$\begin{aligned} \beta_0(K) &= \dim C_0(K; F) - \dim \text{Im } \partial_1 \\ &= \dim C_0(K_1; F) + \dim C_0(K_2; F) - \dim \text{Im } \partial_1^1 - \dim \text{Im } \partial_1^2 \\ &= \beta_0(K_1) + \beta_0(K_2). \end{aligned}$$

The result follows by induction. □

A.18 Euler–Poincaré formula

The proof of Theorem 1.32 is due to Botnan [33].

Proof. From linear algebra, a linear map $T: V \rightarrow W$ satisfies $\dim V = \dim \ker T + \dim \operatorname{Im} T$; applied to the boundary map ∂_q , this gives $\dim C_q(K; F) = \dim Z_q(K; F) + \dim B_{q-1}(K; F)$. Setting $B_{-1}(K; F) = 0$,

$$\begin{aligned}\chi(K) &= \sum_{q=0}^{\infty} (-1)^q \dim C_q(K; F) \\ &= \sum_{q=0}^{\infty} (-1)^q (\dim Z_q(K; F) + \dim B_{q-1}(K; F)) \\ &= \sum_{q=0}^{\infty} (-1)^q (\dim Z_q(K; F) - \dim B_q(K; F)) \\ &= \sum_{q=0}^{\infty} (-1)^q \beta_q(K),\end{aligned}$$

the third line obtained by re-indexing the $B_{q-1}(K; F)$ sum by one and absorbing the sign flip. $\square$

A.19 Rank, nullity, and the Betti-number formula

We follow Virk [16] for this appendix section.

Proposition A.17. *Let $f: A \rightarrow B$ be a linear map of vector spaces. Then:*

1. $\dim A = \dim(\ker f) + \operatorname{rank} f$;
2. $\dim(B/\operatorname{Im} f) = \dim B - \operatorname{rank} f$.

Proof. Part 1 is the rank–nullity theorem of linear algebra. Part 2 is exactly the identity $\dim V/W = \dim V - \dim W$ proved in Appendix A.10, applied with $\operatorname{Im} f$ as the linear subspace. $\square$

Proof of Proposition 1.33. Part 1 follows from part 1 of Proposition A.17, since $n_q = \dim C_q(K; F)$. Part 2 follows from part 2 of Proposition A.17 and part 1. $\square$

A.20 Extracting representing cycles from the Smith normal form

We follow Virk [16] for this appendix section, which works Example 1.31 out in full.

The recipe is the one stated in Section 1.5. Given the annotated Smith normal form of a boundary matrix, with r the number of 1s on the diagonal:

- the annotations of the columns from index $r + 1$ on form a basis of the kernel;
- the boundaries of the annotations of the columns of index up to r form a basis of the image.

A basis of the homology group is then obtained by completing the basis of the image to a basis of the kernel; the vectors added in the completion are the representatives.

Applied to the complex K of Example 1.31, dimension by dimension:

Dimension 0.

1. $\ker \partial_0$ has basis $\langle a \rangle, \langle b \rangle, \langle c \rangle, \langle d \rangle, \langle e \rangle$.
2. $\operatorname{Im} \partial_1$ has a basis formed by the images of the first three annotated columns of the Smith normal form M_1'' , that is of $\langle a, b \rangle$, $\langle b, c \rangle$ and $\langle b, d \rangle - \langle b, c \rangle$. The basis obtained this way is

$$\langle b \rangle - \langle a \rangle, \quad \langle c \rangle - \langle b \rangle, \quad \langle d \rangle - \langle b \rangle + \langle c \rangle - \langle b \rangle.$$

3. We may complete the basis from 2. to the basis of $\ker \partial_0$ by, for example, adding $\langle a \rangle$ and $\langle e \rangle$, and thus $\langle a \rangle$ and $\langle e \rangle$ represent the two 0-holes spanning $H_0(K; F)$.

Dimension 1.

1. $\ker \partial_1$ has basis $\langle a, c \rangle - \langle b, c \rangle - \langle a, b \rangle$ and $\langle c, d \rangle - \langle b, d \rangle + \langle b, c \rangle$ — the annotations of the last two columns of M_1'' .
2. $\text{Im } \partial_1$ has a basis formed by the images (boundaries) of the first annotated column of the Smith normal form M_2'' , that is of $\langle a, b, c \rangle$. The basis obtained this way is

$$\langle a, b \rangle + \langle b, c \rangle - \langle a, c \rangle.$$

3. We may complete the basis from 2. to the basis of $\ker \partial_1$ by, for example, adding $\langle c, d \rangle - \langle b, d \rangle + \langle b, c \rangle$, and thus $\langle c, d \rangle - \langle b, d \rangle + \langle b, c \rangle$ represents the 1-hole spanning $H_1(K; F)$.²

A.21 Simplicial maps

We follow Virk [16] for this appendix section.

Chapter 1 looks at one complex at a time, until filtrations arrive in Section 1.6: a growing family of complexes needs a way to relate one complex to the next, that is, a notion of “map” between simplicial complexes that respects their combinatorial structure, sending simplices to simplices and never tearing one apart. That is the simplicial map defined below, and the functoriality of the following appendix section is what lets the induced homology maps of a filtration be strung together into a single sequence.

Definition A.18 (Simplicial map). Suppose K and L are abstract simplicial complexes. A simplicial map between K and L is an assignment $f: K^{(0)} \rightarrow L^{(0)}$ on vertices, such that for each abstract simplex $\{v_0, v_1, \dots, v_k\} \in K$ its image $\{f(v_0), f(v_1), \dots, f(v_k)\}$ is an abstract simplex in L .

Remark A.19. A simplicial map will be usually denoted by $f: K \rightarrow L$. However, since K and L are collections of sets, such a notation would formally include maps that map, say, a vertex to an edge, a highly unfavourable occurrence. When talking about simplicial maps $K \rightarrow L$ we thus always consider only maps in the sense of Definition A.18, i.e., maps that map a vertex to a vertex, while images on simplices are always determined by the values on the vertices:

- For each vertex $v \in K$ we define a corresponding vertex $f(v)$.
- For each abstract simplex $s = \{v_0, v_1, \dots, v_k\} \in K$ its image $f(s) = \{f(v_0), f(v_1), \dots, f(v_k)\}$ is determined by the values on its vertices.
- Note that $f(s)$ is a set, meaning there are no repetitions of elements. In particular this means that each vertex appears at most once in $f(s)$, even if it appears multiple times as $f(v_i)$. As a result the image of an n -dimensional simplex can be of dimension less than or equal to n , but never more than n .

Example A.1. Let K be the simplicial complex on vertices a, b, c, d, e with triangle $\{a, b, c\}$, edge $\{c, d\}$, and vertex e attached (as in Virk, Fig. 3.22). The assignment $a \mapsto a, b \mapsto c, c \mapsto c, d \mapsto d, e \mapsto b$ can be verified to induce a simplicial map $K \rightarrow K$. Note that triangle $\{a, b, c\}$ gets mapped to edge $\{a, c\}$.

A.22 Functoriality of homology

We follow [16] for this appendix section.

²The fact that the basis element from 2. is a member of the basis of 1. lets us see this completion immediately. Such a situation is an exception, and a completion of basis typically involves some work with linear algebra.

Definition A.20 (Functoriality of homology). Suppose $f: K \rightarrow L$ is a simplicial map between simplicial complexes, $q \in \{0, 1, \dots\}$, and F is a field. The induced maps $f_{\#}$ and f_* are defined as follows:

- $f_{\#}: C_q(K; F) \rightarrow C_q(L; F)$ is the linear map of chain groups defined as

$$f_{\#}\left(\sum_i a_i s_i\right) = \sum_{i: \dim(f(s_i))=q} a_i f(s_i), \quad a_i \in F, s_i^q \in K.$$

- $f_*: H_q(K; F) \rightarrow H_q(L; F)$ is the linear map defined as $f_*([\alpha]) = [f_{\#}(\alpha)]$.

We refrain from specifying q and F in the notation f_* in order not to overload it with the indices. As such f_* represents the induced map on homology in any dimension or with any coefficients. The relevant choice of the dimension(s) and coefficients should always be apparent from the context.

Given a simplex $s \in K$ of dimension q , its image $f(s)$ is a simplex of dimension q or less. The condition on dimension in the definition of $f_{\#}$ means that only the images of those simplices s_i , which are of full dimension q , are taken into account. In particular,

$$f_{\#}(s) = \begin{cases} f(s), & \dim(f(s)) = q \\ 0, & \text{else.} \end{cases}$$

The induced map f_* turns out to be well defined, i.e., if $[\alpha] = [\beta]$ then $f_*([\alpha]) = f_*([\beta])$.

Proposition A.21 (Functoriality of the induced maps). Suppose maps $f: K \rightarrow L$ and $g: L \rightarrow M$ between simplicial complexes are simplicial. Then for each $q \in \{0, 1, \dots\}$ and for each F we have

$$(g \circ f)_{\#} = g_{\#} \circ f_{\#}, \quad \text{and} \quad (g \circ f)_* = g_* \circ f_*.$$

Proof. The proof follows straight from the definition. □

Every inclusion $i_{s,t}: K_s \hookrightarrow K_t$ used in a filtration (Section 1.6) is in particular a simplicial map (Definition A.18), so Definition A.20 applies to it: the induced chain map $(i_{s,t})_{\#}$ is just the inclusion of chains, and the induced homology map $(i_{s,t})_*$ is the one used throughout Section 1.6.1. By Proposition A.21, applied to the composition $i_{u,t} \circ i_{s,u} = i_{s,t}$ for $s \leq u \leq t$, we get $(i_{u,t})_* \circ (i_{s,u})_* = (i_{s,t})_*$: this is exactly the compatibility that lets the homology groups of a filtration be strung together into the sequence of maps of eq. (1.4).

A.23 The barcode, formally

This continues the “Barcodes” discussion of Section 1.6.2, where the counting formula (1.7) was stated but not derived. This more formal definition is taken from [16]. Throughout, fix a field F , a dimension $q \in \{0, 1, \dots\}$, and a filtration $K_1 \leq K_2 \leq \dots \leq K_m = K$. We drop q from the notation $\beta_{s,t}^q$ and simply write $\beta_{s,t}$, since the dimension is fixed; for $s = 0$ we set $(i_{0,t})_*$ to be the trivial map.

A bar $[s, t]$ is meant to represent a homology element born *precisely* at s and terminating *precisely* at t , not merely alive somewhere in between. We build up, one piece at a time, exactly how many bars start where and end where:

1. The number of bars containing s and passing through t equals $\beta_{s,t}$, by definition.
2. Homology born at s is what remains of $H_q(K_s; F)$ once the classes already visible at the previous stage are quotiented away, $H_q(K_s; F)/(\text{Im } i_{s-1,s})_*$,³ its dimension, $\beta_{s,s} - \beta_{s-1,s}$,⁴ is the number of bars starting at s .

³ $(\text{Im } i_{s-1,s})_*$ is the part of $H_q(K_s; F)$ that came from K_{s-1} , so what remains after quotienting is only the homology genuinely new at K_s .

⁴By the identity $\dim(V/W) = \dim V - \dim W$ for a subspace $W \subset V$, proved in Appendix A.10. Taking $V = H_q(K_s; F)$ and $W = (\text{Im } i_{s-1,s})_*$: since $(i_{s,s})_*$ is the identity, (1.6) gives $\dim V = \beta_{s,s}$, and $\dim W = \text{rank}(i_{s-1,s})_* = \beta_{s-1,s}$, again by (1.6).

3. Symmetrically, homology terminating at t is $\ker(i_{t-1,t})_*$, and its dimension, $\beta_{t-1,t-1} - \beta_{t-1,t}$, is the number of bars terminating at t .
4. The quantity $\beta_{s,t} - \beta_{s-1,t}$ is the dimension of homology born at s that is still alive at t : the number of bars starting at s and passing through t .⁵
5. Combining a birth count at s with a survival count up to t isolates exactly the bars born at s and terminating at t , which is the counting formula (1.7) of Section 1.6.2,

$$n_{s,t} = \beta_{s,t-1} - \beta_{s-1,t-1} - (\beta_{s,t} - \beta_{s-1,t}).$$

6. Finally $n_{s,\infty} = \beta_{s,m} - \beta_{s-1,m}$ counts the bars born at s that are still alive at the very end of the filtration.

Definition A.22 (Barcode). The q -dimensional **barcode** of the filtration is the collection of intervals

$$[s, t) \text{ for } 1 \leq s < t \leq m, \quad [s, \infty) \text{ for } 1 \leq s < m,$$

where each interval $[s, t)$ occurs with multiplicity $n_{s,t}$, and each $[s, \infty)$ with multiplicity $n_{s,\infty}$.

A.24 Fundamental lemma of persistent homology

This continues Section 1.6.2: the statement is Lemma 1.37. The following proof is taken from [16].

Proof. The formula can be verified explicitly. However, the statement is apparent from the definitions, as

- $\beta_{s,t}$ represents the homology born at s or before and terminating after t ;
- $n_{s',t'}$ represents the homology born precisely at s' and terminating precisely at t' .

□

A.25 Computing persistent homology: matrix reduction, extraction, and representatives

The following discussion is based on [16]. Throughout this section we fix a field F and a filtration $K_1 \leq K_2 \leq \dots \leq K_m = K$; parameter $q \in \{0, 1, \dots\}$ denotes the dimension of a considered object.

Matrix reduction. This part could be called an annotated matrix reduction using only column operations from the left.

1. Order simplices consistently with the filtration. For each q order all q -simplices in an order in which they appear in the filtration. If more simplices appear at the same time their internal order is immaterial.
2. For each q construct the boundary matrix M_q (Definition 1.23) using the order chosen in 1. to label columns and rows.
3. For each q reduce M_q from the left using a single type of column operation: the addition of an F -multiple of any of the previous columns to a treated column. Specifically, starting with the leftmost column, go through all the columns by passing to the right and for each column:
 - (a) Determine the pivot (the lowest non-trivial entry in the column).
 - (b) If any of the previous columns on the left has a pivot in the same row, subtract the appropriate multiple of that column so that the pivot of the current column either disappears or its location is moved up.

⁵This follows from item 1 above.

(c) Repeat as long as there are matching pivots on the left.

For each q the resulting matrix is denoted by M'_q . Each of its columns is either trivial or has a pivot, whose row is unique amongst all pivots.

Extracting persistence. Given a simplicial complex, the addition of a single q -simplex can change the homology in two ways:

- If its boundary is a linear combination of boundaries of other q -simplices⁶, then the simplex gives birth to a non-trivial q -dimensional homology element. In this case we call the simplex a **birth simplex**.
- If its boundary is *not* a linear combination of boundaries of other q -simplices, then the simplex terminates a non-trivial $(q - 1)$ -dimensional homology element. In this case we call the simplex a **terminal simplex**.

A filtration can be considered to be a sequence of incremental expansions. At each stage of the filtration we may assume we first add all vertices according to the ordering in 1. above, then all edges, etc. Combining such an ordering through all stages we get a sequence of incremental expansions inducing boundary matrices M_q and their reduced forms M'_q .

Based on such an ordering each simplex of K is either a terminal simplex or a birth simplex. We are now in a position to extract the persistence diagram and barcode:

- For each terminal q -simplex τ there exists a paired birth $(q - 1)$ -simplex σ , which is the label of the pivot in the column τ . Such a pair induces a bar $[s, t)$ in the corresponding barcode or, equivalently, a point (s, t) in the corresponding persistence diagram, where s, t are the stages of the filtration at which σ and τ appear.
- Each birth simplex which is not paired to a terminal simplex induces a bar $[s, \infty)$ in the corresponding barcode or, equivalently, a point $(s, m + 1)$ in the corresponding persistence diagram, where s is the stage of the filtration at which σ appears.

As each row contains at most one pivot, each birth simplex is paired to at most one terminal simplex; a terminal simplex cannot appear as the label of a pivot column. Note that if $s = t$ we obtain an empty interval in the barcode and a point on the diagonal in the persistence diagram, both of which we ignore in the visualisation, as they represent elements of persistence zero; this is consistent with our interpretation of persistent homology, which measures only holes that persist through at least one stage of the filtration. It turns out that the presented definition and computation of the barcode respects the elder rule mentioned in Section 1.6.1.

Representatives. Occasionally we are also interested in homology representatives of the bars and points of persistence diagrams. These can be extracted from the reduction process. Given a bar with the birth simplex σ and the terminal simplex τ we define:

- The **birth representative** of σ as the chain formulated by the reduction of the column corresponding to σ to the zero column in the column reduction scheme. In particular, if the linear combination turning column σ into the zero column in our column reduction scheme is encoded in terms of columns as $\partial\sigma - \sum_i \lambda_i \partial\sigma_i = 0$, then the birth representative is $\alpha = \sigma - \sum_i \lambda_i \sigma_i$. The birth representative gives a homology class $[\alpha]$ that is born by the addition of σ .
- The **terminal representative** is encoded by the column corresponding to τ in the reduced matrix.

The birth representative and the terminal representative typically do not represent the same homology class. The birth representative may not even be a good representative of the corresponding bar in the sense that it may remain homologically non-trivial beyond the appearance of the corresponding terminal simplex. On the other hand, infinite intervals do not have a terminal representative. As a result we define the representative of a bar as follows:

⁶ I.e., if, after adding the simplex to the boundary matrix as the rightmost column, its column gets reduced to the trivial column by the above reduction.

1. If the bar is a finite interval, the representative of the bar is the terminal representative.
2. If the bar is an infinite interval, the representative of the bar is the birth representative.

This choice of representatives is algebraically sound in the sense that the representatives form a basis of the elementary intervals of the decomposition described in the structure theorem for persistence modules, a result stated (but not proved) later in the thesis. This statement includes the fact that the lifespan of each representative matches the lifespan of the corresponding bar, and that the representatives are linearly independent at all times.

In practice we sometimes deviate from the algebraically orthodox choice when declaring the representatives of 0-dimensional bars: we choose the birth representative as a bar representative even if the bar is bounded. The worked example below illustrates this geometrically motivated exception on the pair $(\langle b \rangle, \langle a, b \rangle)$, which induces a 0-dimensional bar. Geometrically we would like to think of this bar as a representation of the component containing b merging with a larger component, hence the choice of the birth representative $\langle b \rangle$, which fits this geometric intuition. However, we should be aware that the homological element $[\langle b \rangle]$ does not become trivial after adding $\langle a, b \rangle$: in terms of homology, the appearance of $\langle a, b \rangle$ identifies $[\langle a \rangle] = [\langle b \rangle]$ rather than setting $[\langle b \rangle] = 0$.

A worked example. We now run the whole procedure on the filtration $K_1 \leq K_2 \leq K_3 \leq K_4$ of Example 1.33, with the simplices of K labelled explicitly as in Figure A.2.⁷ Throughout, we fix a field F .

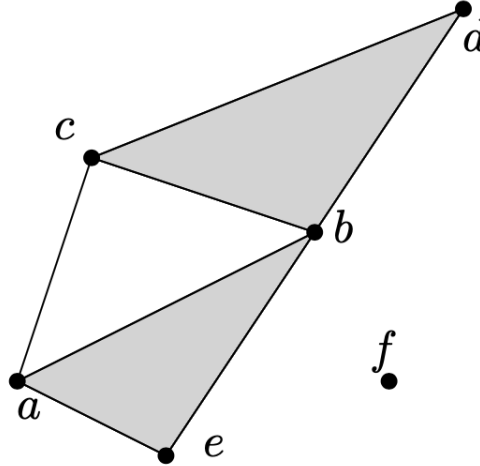


Figure A.2: The complex K of Figure 1.36, with its simplices labelled. Image has been collected from [16].

The reduction. Ordering simplices consistently with the filtration, the boundary matrices (Definition 1.23) of K are, with vertical and horizontal lines dividing simplices from different stages of the filtration,

$$M_1 = \left(\begin{array}{c|cc|cc|cc} & \langle b, c \rangle & \langle b, d \rangle & \langle a, b \rangle & \langle c, d \rangle & \langle a, c \rangle & \langle a, e \rangle & \langle b, e \rangle \\ \hline \langle a \rangle & & & -1 & & -1 & -1 & \\ \langle b \rangle & -1 & -1 & 1 & & & & -1 \\ \langle c \rangle & 1 & & & -1 & 1 & & \\ \langle d \rangle & & 1 & & 1 & & & \\ \hline \langle e \rangle & & & & & & 1 & 1 \\ \langle f \rangle & & & & & & & \end{array} \right)$$

⁷The vertices in Figure A.2 are named as in Virk [16] and do *not* carry the names used in Figure 1.36: the two pictures show the same filtration with its vertices labelled differently. Only the names change, so each bar found below is the same bar under either labelling.

$$M_2 = M'_2 = \left(\begin{array}{cc|c} & \langle b, c, d \rangle & \langle a, b, e \rangle \\ \hline \langle b, c \rangle & 1 & \\ \langle b, d \rangle & -1 & \\ \hline \langle a, b \rangle & & 1 \\ \langle c, d \rangle & \mathbf{1} & \\ \hline \langle a, c \rangle & & \\ \langle a, e \rangle & & -1 \\ \langle b, e \rangle & & \mathbf{1} \end{array} \right).$$

The matrix M_2 is already reduced (each non-trivial column already has a pivot — its lowest non-trivial entry, shown in bold — in a row no other column's pivot occupies), so we only need to reduce M_1 . We go through its columns left to right, and whenever a column's pivot lands in the same row as an earlier column's pivot, we subtract a multiple of that earlier column (coloured red) from the treated column (coloured blue) to clear it:

$$\left(\begin{array}{cc|cc|ccc} & \langle b, c \rangle & \langle b, d \rangle & \langle a, b \rangle & \langle c, d \rangle & \langle a, c \rangle & \langle a, e \rangle & \langle b, e \rangle \\ \hline \langle a \rangle & & & -1 & & -1 & -1 & \\ \langle b \rangle & -1 & -1 & 1 & & & & -1 \\ \langle c \rangle & 1 & & & -1 & 1 & & \\ \langle d \rangle & & 1 & & 1 & & & \\ \hline \langle e \rangle & & & & & & 1 & 1 \\ \langle f \rangle & & & & & & & \end{array} \right) \quad \partial \langle c, d \rangle = \partial \langle b, d \rangle - \partial \langle b, c \rangle.$$

$$\left(\begin{array}{cc|cc|ccc} & \langle b, c \rangle & \langle b, d \rangle & \langle a, b \rangle & \langle c, d \rangle & \langle a, c \rangle & \langle a, e \rangle & \langle b, e \rangle \\ \hline \langle a \rangle & & & -1 & & -1 & -1 & \\ \langle b \rangle & -1 & -1 & 1 & & & & -1 \\ \langle c \rangle & 1 & & & & 1 & & \\ \langle d \rangle & & 1 & & & & & \\ \hline \langle e \rangle & & & & & & 1 & 1 \\ \langle f \rangle & & & & & & & \end{array} \right) \quad \partial \langle a, c \rangle = \partial \langle a, b \rangle + \partial \langle b, c \rangle.$$

$$\left(\begin{array}{cc|cc|ccc} & \langle b, c \rangle & \langle b, d \rangle & \langle a, b \rangle & \langle c, d \rangle & \langle a, c \rangle & \langle a, e \rangle & \langle b, e \rangle \\ \hline \langle a \rangle & & & -1 & & & -1 & \\ \langle b \rangle & -1 & -1 & 1 & & & & -1 \\ \langle c \rangle & 1 & & & & & & \\ \langle d \rangle & & 1 & & & & & \\ \hline \langle e \rangle & & & & & & 1 & 1 \\ \langle f \rangle & & & & & & & \end{array} \right) \quad \partial \langle b, e \rangle = \partial \langle a, e \rangle - \partial \langle a, b \rangle.$$

Each of these three identities says precisely that the column in question reduces to zero: the columns of $\langle c, d \rangle$, $\langle a, c \rangle$ and $\langle b, e \rangle$ all become trivial, while $\langle b, c \rangle$, $\langle b, d \rangle$, $\langle a, b \rangle$ and $\langle a, e \rangle$ are left untouched. The resulting reduced matrix is

$$M'_1 = \left(\begin{array}{cc|cc|ccc} & \langle b, c \rangle & \langle b, d \rangle & \langle a, b \rangle & \langle c, d \rangle & \langle a, c \rangle & \langle a, e \rangle & \langle b, e \rangle \\ \hline \langle a \rangle & & & -1 & & & -1 & \\ \langle b \rangle & -1 & -1 & \mathbf{1} & & & & \\ \langle c \rangle & \mathbf{1} & & & & & & \\ \langle d \rangle & & \mathbf{1} & & & & & \\ \hline \langle e \rangle & & & & & & \mathbf{1} & \\ \langle f \rangle & & & & & & & \end{array} \right).$$

Births and terminations in the example. In the sense of the “Extracting persistence” discussion above, we may now read off M'_1 and $M_2 = M'_2$: the birth simplices are $\langle a \rangle, \langle b \rangle, \langle c \rangle, \langle d \rangle, \langle e \rangle, \langle f \rangle$ (all six vertices) and $\langle c, d \rangle, \langle a, c \rangle, \langle b, e \rangle$ (the three edges whose columns vanished); the terminal simplices are $\langle b, c \rangle, \langle b, d \rangle, \langle a, b \rangle, \langle a, e \rangle$ (pivots of M'_1) and $\langle b, c, d \rangle, \langle a, b, e \rangle$ (pivots of M'_2).

The barcode of the example. Each terminal simplex is paired with the birth simplex labelling its pivot's row, and each unpaired birth simplex gives an infinite bar. From M'_1 :

- Pair $(\langle b \rangle, \langle a, b \rangle)$ induces the 0-dimensional bar $[1, 2)$,⁸ represented by $[\langle b \rangle]$.⁹
- Pairs $(\langle c \rangle, \langle b, c \rangle)$ and $(\langle d \rangle, \langle b, d \rangle)$ pair a birth and a terminal simplex appearing at the same stage K_1 , so they contribute nothing to the barcode (an empty interval $[1, 1)$).
- Pair $(\langle e \rangle, \langle a, e \rangle)$ induces the 0-dimensional bar $[2, 3)$, represented by $[\langle e \rangle]$.
- $\langle a \rangle$ and $\langle f \rangle$ are unpaired, inducing the 0-dimensional bars $[1, \infty)$ (generated by $[\langle a \rangle]$) and $[2, \infty)$ (generated by $[\langle f \rangle]$).

These four bars are exactly the $\beta_{s,t}^0$ column of the table in Example 1.33, obtained here without computing a single rank. We next extract the one-dimensional barcode from the pivots of M'_2 and unpaired edges:

- Pair $(\langle c, d \rangle, \langle b, c, d \rangle)$ induces the 1-dimensional bar $[2, 3)$, represented by $[\langle c, d \rangle - \langle b, d \rangle + \langle b, c \rangle]$.¹⁰
- Pair $(\langle b, e \rangle, \langle a, b, e \rangle)$ induces the 1-dimensional bar $[3, 4)$, represented by $[\langle b, e \rangle - \langle a, e \rangle + \langle a, b \rangle]$.
- $\langle a, c \rangle$ is unpaired,¹¹ inducing the 1-dimensional bar $[3, \infty)$, generated by $[\langle a, c \rangle - \langle a, b \rangle - \langle b, c \rangle]$.

Why is $\langle a, c \rangle$, and not $\langle a, b \rangle$, the unpaired edge? In the worked example above, $\langle a, b \rangle$ is a pivot column of M'_1 (paired with the birth simplex $\langle b \rangle$) and so is not itself a birth simplex; only a birth simplex can be unpaired in the first place, since pairing only ever matches a birth q -simplex to a terminal $(q + 1)$ -simplex. By footnote 6, the birth 1-simplices are exactly the edges whose columns reduce to the trivial column in M'_1 : reading off the reduction above, these are $\langle c, d \rangle$, $\langle a, c \rangle$ and $\langle b, e \rangle$. Of these three, $\langle c, d \rangle$ is paired with the terminal simplex $\langle b, c, d \rangle$ and $\langle b, e \rangle$ is paired with the terminal simplex $\langle a, b, e \rangle$ (both pivots of M'_2), leaving $\langle a, c \rangle$ as the only birth simplex among the three with no terminal simplex to pair it to — hence $\langle a, c \rangle$, and only $\langle a, c \rangle$, is unpaired.

A.26 Rips and Čech filtrations (continuous formulation)

This continues Section 2.1.2 and Section 2.1.3, giving the fully formal version of the Rips and Čech filtrations: indexed continuously by every scale $r \geq 0$, rather than only by the finitely many stages at which $\text{Rips}(S, r)$ or $\check{\text{Cech}}(S, r)$ actually changes.

Definition A.23 (Rips filtration). Let X be a metric space and let a sample $S \subset X$ be a finite subset. The Rips filtration on S is the collection of abstract simplicial complexes $\{\text{Rips}(S, r)\}_{r \geq 0}$ along with inclusions

$$i_{r_1, r_2} : \text{Rips}(S, r_1) \hookrightarrow \text{Rips}(S, r_2)$$

for all $r_1 \leq r_2$. A Rips filtration provides the collection of all Rips complexes on S . While a single Rips complex depends on the choice of the scale, the filtration does not.

12

Definition A.24 (Čech filtration). Let X be a metric space and let a sample $S \subset X$ be a finite subset. The Čech filtration of S is the collection of abstract simplicial complexes $\{\check{\text{Cech}}(S, r)\}_{r \geq 0}$

⁸ $\langle b \rangle$ appears at K_1 while $\langle a, b \rangle$ appears at K_2 , hence the values of the endpoints.

⁹Intuitively, this bar records the component containing b merging into the larger component containing a , and $\langle b \rangle$ is a natural representative point of that disappearing component. The formal definition of a representative, and of this particular choice for 0-dimensional bars, is given in the “Representatives” discussion above.

¹⁰ $\langle c, d \rangle$ appears at K_2 while $\langle b, c, d \rangle$ appears at K_3 , hence the endpoints. The linear combination that made the column corresponding to $\langle c, d \rangle$ trivial in M'_1 was $\partial \langle c, d \rangle - \partial \langle b, d \rangle + \partial \langle b, c \rangle$, and hence the representative. See the “Representatives” discussion above for the formal definition of a representative.

¹¹One might object: $\langle a, b \rangle$ also does not label any pivot column of M'_2 , so why is $\langle a, c \rangle$ singled out as “unpaired” and not $\langle a, b \rangle$? This is resolved in the final paragraph of this section.

¹²More generally, a filtration of a simplicial complex K is a family of subcomplexes $\{K_r\}_{r \geq 0}$ indexed by a parameter r , such that $K_r \leq K_{r'}$ for all $r < r'$. The Rips filtration on a finite set S is a filtration of the $(|S| - 1)$ -simplex.

along with inclusions

$$i_{r_1, r_2}: \check{\text{Cech}}(S, r_1) \hookrightarrow \check{\text{Cech}}(S, r_2)$$

for all $r_1 \leq r_2$. As was the case with Rips filtrations, Čech filtrations also provide a scale-free approximation of the underlying set by simplicial complexes.

A.27 Transitivity of homeomorphism

This continues Section 1.1.1, spelling out why homeomorphism is an equivalence relation [30]: reflexivity and symmetry directly from the definition, and transitivity via a short argument.

Reflexivity: $X \cong X$ for every X , since the identity map is a bijection, is continuous, and is its own inverse.

Symmetry: $X \cong Y$ if and only if $Y \cong X$. If f is a homeomorphism then f^{-1} is a bijection, is continuous by hypothesis, and has continuous inverse f ; so f^{-1} is a homeomorphism too.

Transitivity is the only one needing an argument.

Proof of transitivity of homeomorphism (Section 1.1.1). Pick homeomorphisms $f: X \rightarrow Y$ and $g: Y \rightarrow Z$, and look at the composite $g \circ f: X \rightarrow Z$. Composing two bijections leaves a bijection, and composing two continuous maps leaves a continuous map, so $g \circ f$ is a continuous bijection. Its inverse is $f^{-1} \circ g^{-1}$, continuous for the same reason. All three conditions of Definition 1.2 therefore hold, and $X \cong Z$. $\square$

A.28 Invariants under homotopy equivalence

Both invariants met so far were shown to survive homeomorphism. It is worth asking how they fare under the weaker relation of homotopy equivalence. The first result is the counterpart of Proposition 1.8.

Proposition A.25 ([30]). *Homotopy equivalent topological spaces have the same number of connected components.*

The proof is given in Appendix A.6.

The count of components is thus an invariant of homotopy equivalence too, and serves the same purpose there: a difference in the count rules out any homotopy equivalence between two spaces.

Example A.2. Let $n, m \geq 0$ with $n \neq m$. The subspaces $\{1, \dots, n\}$ and $\{1, \dots, m\}$ of $\mathbb{R}$ are not homotopy equivalent, the first having n connected components and the second m .

A.29 Why the nerve theorem applies to a collection of balls

This continues Section 2.1.4, spelling out why the nerve theorem (Theorem 2.8) may be applied to the collection of r -balls $\{B(s, r)\}_{s \in S}$ used to build the Čech complex $\check{\text{Cech}}(S, r)$.

The nerve theorem requires the sets in the collection $\mathcal{U}$ to be closed and convex subsets of $\mathbb{R}^n$. A closed ball $B(x, r) = \{y \in \mathbb{R}^n \mid \|x - y\| \leq r\}$ is closed by definition, and it is convex: given any two points $y_1, y_2 \in B(x, r)$ and any $t \in [0, 1]$, the point on the segment between them satisfies

$$\|x - (ty_1 + (1 - t)y_2)\| \leq t\|x - y_1\| + (1 - t)\|x - y_2\| \leq tr + (1 - t)r = r,$$

using the triangle inequality, so the whole segment between y_1 and y_2 stays inside $B(x, r)$. Every ball in the collection $\{B(s, r)\}_{s \in S}$ is therefore a closed convex subset of $\mathbb{R}^n$, so the hypotheses of Theorem 2.8 hold, and the theorem gives

$$\mathcal{N}(\{B(s, r)\}_{s \in S}) \simeq \bigcup_{s \in S} B(s, r).$$

A.30 Passing between discrete and continuous filtrations

This continues Section 2.2.1, spelling out how a discrete filtration and a continuous filtration can always be converted into one another without losing information.

1. Given a discrete filtration, there is an obvious extension of it as the sublevel filtration of the annotation function.
2. Given a continuous sublevel filtration $\{K_r\}_{r \geq 0}$ associated to a filtration function f , there are two ways of generating a discrete filtration:
 - (a) By restriction to $K_1 \leq K_2 \leq \dots \leq K_{\lceil \max f \rceil}$. While mathematically convenient, this approach has many drawbacks and is mostly avoided: the information about the sequence of changes between each pair of integer scales is lost.
 - (b) A more beneficial way of thinking about the index i of a discrete filtration is not as the scale parameter but rather as the index of the **critical scale** of the continuous filtration.¹³ Formally, define critical scales $r_1 < r_2 < \dots < r_k$ as the enumeration of the image of f , and define $K_i = \{\sigma \in K \mid f(\sigma) \leq r_i\}$. The corresponding finite filtration contains information about all changes in the original continuous filtration.

From this point on, whenever we mention an unspecified filtration, or consider a transition from a finite to continuous filtration or vice versa, the underlying interplays we have in mind are item 1 and item 2(b) above.

A.31 The stability theorem, complete statement

This continues Section 2.2.2, spelling out the full hypotheses of the stability theorem, stated there only qualitatively.

Theorem A.26 (Stability theorem). *Assume persistence diagrams $\mathcal{A}$ and $\mathcal{B}$ represent the persistent homology of filtrations $\mathcal{V}$ and $\mathcal{W}$ obtained by one of the following procedures:*

1. *as the sublevel filtrations of filtration functions f and g that are uniformly close, $\|f - g\|_\infty \leq \varepsilon$, namely no simplex's two labels ever differ by more than ε ;*
2. *as the Rips filtrations of two point clouds X and Y that are close as shapes, $d_{GH}(X, Y) \leq \varepsilon/2$, where d_{GH} is the Gromov–Hausdorff distance, a way of measuring how far two metric spaces are from being isometric even when they do not sit inside the same ambient space [30];*
3. *as the Čech filtrations of two such point clouds, with $d_{GH}(X, Y) \leq \varepsilon$.*

Then $d_B(\mathcal{A}, \mathcal{B}) \leq \varepsilon$.

A.32 Persistence images: formal construction

This continues Section 2.4.2, giving the fully formal version of the persistence-image construction sketched there. Given a persistence diagram D , its persistence image is obtained as follows.

1. A transformation $T: \mathbb{R}^2 \rightarrow \mathbb{R}^2$, where $T(x, y) = (x, y - x)$, is applied to the diagram D . The transformed diagram $T(D)$ is now in birth-persistence coordinates.
2. $\Phi_u: \mathbb{R}^2 \rightarrow \mathbb{R}$, a differentiable probability distribution with mean $u \in \mathbb{R}^2$, and $f: \mathbb{R}^2 \rightarrow \mathbb{R}$, a continuous, piece-wise differentiable weight function which is zero on points on the x -axis, are chosen. For instance, Φ_u can be a normalized Gaussian distribution with mean u , and f a weighting function that depends only on the y -persistence coordinate.

¹³A scale r of a continuous filtration is critical if at least one simplex appears at r .

3. The persistence surface $P_D: \mathbb{R}^2 \rightarrow \mathbb{R}$ is given by

$$P_D(z) = \sum_{u \in T(D)} f(u) \Phi(z).$$

The weights corresponding to points along the diagonal in D are zero, as T maps them onto the x -axis and these do not contribute to the above sum.

4. The persistence image $I(P_D)$ is obtained from the persistence surface by discretising the relevant region using a grid and integrating over each box/pixel. That is, for any pixel p in the persistence image, the intensity is given by

$$I(P_D)_p = \iint_p P_D \, dy \, dx.$$

The persistence images offer a lot of flexibility through user choices for the distribution Φ_u and weight function f . They can also be integrated with machine learning algorithms easily.

A.33 Shannon entropy: an intuitive discussion

The following is an intuitive discussion of Shannon entropy on its own, away from persistence diagrams, based on the more formal treatment of Saraiva [71]; it supplements the persistent entropy of Section 2.4.1 with the underlying information-theoretic idea in its original, simpler setting.

Shannon entropy measures the average uncertainty, randomness, or surprise in a set of outcomes and their probabilities. It was introduced by Claude Shannon in 1948 and is the central quantity of information theory, where it measures how much “information” a probability distribution carries. For a random variable X with possible outcomes x and probabilities $p(x)$, the Shannon entropy is

$$H(X) = - \sum_x p(x) \log_2 p(x).$$

In words: $H(X)$ is obtained by weighting the surprise of each outcome, $-\log_2 p(x)$ (large when $p(x)$ is small, i.e. the outcome is rare and hence surprising), by how often that outcome actually occurs, $p(x)$, and summing over every possible outcome; the minus sign in front simply compensates for the fact that $\log_2$ of a probability (a number between 0 and 1) is itself negative, so that $H(X)$ comes out non-negative. The result, $H(X)$, is conventionally measured in bits, since the logarithm is taken in base 2.

Two extreme cases build the right intuition:

- If the system is completely predictable — one outcome has probability 1 and every other outcome has probability 0, as for a coin that always lands on the same face — there is no uncertainty left to measure, and $H(X) = 0$.
- If the system is as unpredictable as possible — every outcome equally likely — the uncertainty is maximal, and $H(X)$ attains its largest possible value.

Example A.3 (A fair coin flip). A fair coin has two outcomes, heads and tails, each with probability $p(\text{heads}) = p(\text{tails}) = 0.5$. Substituting into the formula for $H(X)$,

$$H(X) = -[0.5 \log_2(0.5) + 0.5 \log_2(0.5)].$$

Since $\log_2(0.5) = -1$, this is $H(X) = -[0.5(-1) + 0.5(-1)] = 1$ bit: a fair coin flip carries exactly one bit of entropy, the maximum possible for two outcomes.

Example A.4 (A completely biased coin). Now suppose the coin is rigged to always land on heads, so $p(\text{heads}) = 1$ and $p(\text{tails}) = 0$. Then $\log_2(1) = 0$, and the tails term vanishes too: by convention $0 \log_2(0) = 0$, since an outcome that never happens contributes no uncertainty. Hence $H(X) = -[0 + 0] = 0$ bits, matching the intuition above: since the outcome is certain, flipping this

coin teaches us nothing new.

Example A.5 (A fair six-sided die). A fair die has six equally likely outcomes, each with probability $p(i) = 1/6$. Because all six terms are identical, the sum collapses to six copies of the same term,

$$H(X) = - \sum_{i=1}^6 p(i) \log_2 p(i) = -6 \cdot \frac{1}{6} \log_2 \left(\frac{1}{6} \right) = -\log_2 \left(\frac{1}{6} \right) = \log_2(6) \approx 2.585 \text{ bits.}$$

A die roll is more uncertain than a coin flip, and indeed carries more entropy: more possible outcomes mean more uncertainty about which one will occur, and hence a larger entropy.

A.34 The noise spectral density of a real signal is symmetric

Proposition A.27. Let $n(t)$ be a real-valued, stationary noise process, with Fourier transform $\tilde{n}(f)$ and noise spectral density $S_n(f)$ defined by

$$\langle \tilde{n}^*(f) \tilde{n}(f') \rangle = \frac{1}{2} \delta(f - f') S_n(f).$$

Then $S_n(-f) = S_n(f)$.

Proof. Since $n(t)$ is real, $n(t) = n^*(t)$, so its Fourier transform satisfies

$$\tilde{n}(-f) = \int_{-\infty}^{\infty} n(t) e^{2\pi i f t} dt = \left[\int_{-\infty}^{\infty} n(t) e^{-2\pi i f t} dt \right]^* = \tilde{n}^*(f).$$

Replacing $f \rightarrow -f$ and $f' \rightarrow -f'$ in the definition of $S_n(f)$ yields

$$\langle \tilde{n}^*(-f) \tilde{n}(-f') \rangle = \frac{1}{2} \delta(-f + f') S_n(-f) = \frac{1}{2} \delta(f - f') S_n(-f).$$

On the other hand, substituting $\tilde{n}(-f) = \tilde{n}^*(f)$ directly into the left-hand side of the same ensemble average gives

$$\langle \tilde{n}(f) \tilde{n}^*(f') \rangle = \langle \tilde{n}^*(f') \tilde{n}(f) \rangle = \frac{1}{2} \delta(f' - f) S_n(f').$$

Since the Dirac delta function $\delta(f - f')$ forces $f' = f$, this expression reduces to $\frac{1}{2} \delta(f - f') S_n(f)$, matching the two results and proving that $S_n(-f) = S_n(f)$. $\square$

A.35 Dimensions of the noise spectral density

Proposition A.28. Let $n(t)$ be dimensionless, as is natural for a strain, with noise spectral density $S_n(f)$ defined by

$$\langle \tilde{n}^*(f) \tilde{n}(f') \rangle = \frac{1}{2} \delta(f - f') S_n(f).$$

Then $S_n(f)$ has dimensions of Hz^{-1} .

Proof. This proof is an adaptation of the argument of Maggiore [12]. It traces back to the finite-time averaging procedure used to define the ensemble average in practice: the noise $n(t)$ is measured over a stretch of duration T , giving a Fourier transform $\tilde{n}(f)$ with frequency resolution $\Delta f = 1/T$.

At exact coincidence, $f = f'$, the right-hand side of the definition of $S_n(f)$ diverges, because $\delta(f - f')$ is only meaningful as a distribution. For a finite observation time T , restricting the time integral defining $\tilde{n}(f)$ to the interval $-T/2 < t < T/2$ regularizes this: evaluated at $f = 0$, the finite-time integral gives exactly

$$\left[\int_{-T/2}^{T/2} dt e^{i2\pi f t} \right]_{f=0} = T,$$

since the integrand is simply 1 at $f = 0$. This is precisely the finite-time, regularized stand-in for $\delta(f = 0)$: instead of an infinitely narrow, infinitely tall spike, restricting to a window of duration T replaces it with the finite value T . Substituting this coincidence value into the definition of $S_n(f)$ then gives

$$\langle |\tilde{n}(f)|^2 \rangle = \frac{1}{2} S_n(f) T,$$

i.e. $S_n(f) = (2/T) \langle |\tilde{n}(f)|^2 \rangle$.

Recalling the Fourier convention $\tilde{n}(f) = \int_{-\infty}^{\infty} dt n(t) e^{2\pi i f t}$, the transform picks up one power of time from the dt it integrates over, so $|\tilde{n}(f)|^2$ has dimensions of (time)²; dividing by T leaves exactly one power of time, i.e. Hz⁻¹. $\square$

A.36 The detection statistic is Gaussian in the absence of a signal

Proposition A.29. *Let $\hat{s}$ be the filtered statistic of (4.35) and N its r.m.s. in the absence of a signal ((4.37)). In the absence of a signal ($h = 0$), the normalised statistic $\rho \equiv \hat{s}/N$ has probability distribution*

$$p(\rho | h = 0) d\rho = \frac{1}{\sqrt{2\pi}} e^{-\rho^2/2} d\rho.$$

Proof. The detector noise $n(t)$ is assumed to be a stationary Gaussian stochastic process, and any linear operation acting on Gaussian noise preserves its Gaussian character. The filter output $\hat{s}$ of (4.35) is a linear functional of the data $s(t)$; therefore, under the null hypothesis $h = 0$, where the detector output consists entirely of noise, $s(t) = n(t)$, $\hat{s}$ is itself a Gaussian random variable.

Since the noise has vanishing ensemble average, $\langle n(t) \rangle = 0$ ((4.28)), the mean of $\hat{s}$ likewise vanishes, $\langle \hat{s} \rangle_{h=0} = 0$. Recalling that $\rho = \hat{s}/N$, with N the standard deviation of $\hat{s}$ when $h = 0$, it follows that ρ is a zero-mean, unit-variance Gaussian random variable, whose probability density function is the standard normal distribution stated above. $\square$

A.37 The detection statistic in the presence of a signal

Proposition A.30. *If the data contains a gravitational-wave signal $h(t)$ with optimal signal-to-noise ratio $\bar{\rho}$ ((4.44)), then*

$$p(\rho | \bar{\rho}) d\rho = \frac{1}{\sqrt{2\pi}} e^{-(\rho - \bar{\rho})^2/2} d\rho.$$

Proof. With a signal present, the total filter output becomes $\rho = \bar{\rho} + \hat{n}/N$, where $\hat{n} = \int_{-\infty}^{\infty} dt n(t) K(t)$ is the noise's own contribution to $\hat{s}$. Because N is, by definition ((4.37)), exactly the r.m.s. value of $\hat{n}$, the same argument as in Proposition A.29 — applied to $\hat{n}$ rather than to $\hat{s}$ — shows that $\hat{n}/N$ is a zero-mean, unit-variance Gaussian random variable. Hence $\rho - \bar{\rho}$ is a standard Gaussian, which is exactly the statement above. $\square$

A.38 The distribution of the SNR in energy

Proposition A.31. *Let $R \equiv \rho^2$ and $\bar{R} \equiv \bar{\rho}^2$. Then*

$$P(R | \bar{R}) dR = \frac{1}{\sqrt{2\pi R}} e^{-(\bar{R} + R)/2} \cosh(\sqrt{R\bar{R}}) dR.$$

Proof. A given value of R corresponds to two values of ρ , namely $\rho = \pm\sqrt{R}$, so the probability of an energy SNR between R and $R + dR$ is the sum of the probabilities of both branches, evaluated using Proposition A.30 and the change of variable $d\rho = dR/(2\sqrt{R})$:

$$P(R | \bar{R}) dR = \left[p(\sqrt{R} | \bar{\rho}) + p(-\sqrt{R} | \bar{\rho}) \right] \frac{dR}{2\sqrt{R}}.$$

Writing out both Gaussians and taking $\bar{\rho} = \sqrt{\bar{R}} \geq 0$ without loss of generality, the two exponents are

$$-\frac{(\sqrt{R} - \sqrt{\bar{R}})^2}{2} = -\frac{R + \bar{R}}{2} + \sqrt{R\bar{R}}, \quad -\frac{(-\sqrt{R} - \sqrt{\bar{R}})^2}{2} = -\frac{R + \bar{R}}{2} - \sqrt{R\bar{R}},$$

so the two terms share the common factor $e^{-(R+\bar{R})/2}$, and their sum is

$$e^{-(R+\bar{R})/2} (e^{\sqrt{R\bar{R}}} + e^{-\sqrt{R\bar{R}}}) = 2 e^{-(R+\bar{R})/2} \cosh(\sqrt{R\bar{R}}).$$

Substituting back,

$$P(R | \bar{R}) dR = \frac{1}{\sqrt{2\pi}} \cdot 2 e^{-(R+\bar{R})/2} \cosh(\sqrt{R\bar{R}}) \cdot \frac{dR}{2\sqrt{R}} = \frac{1}{\sqrt{2\pi R}} e^{-(\bar{R}+R)/2} \cosh(\sqrt{R\bar{R}}) dR,$$

which is the stated distribution. Setting $\bar{R} = 0$ recovers the noise-only distribution of Proposition A.29 in the variable R . $\square$

A.39 The mean detected energy SNR

Proposition A.32. $\langle R \rangle = 1 + \bar{R}$.

Proof. By Proposition A.30, ρ is a Gaussian random variable with mean $\bar{\rho}$ and variance 1, so $\langle \rho^2 \rangle = \text{Var}(\rho) + \langle \rho \rangle^2 = 1 + \bar{\rho}^2$. Since $R = \rho^2$ and $\bar{R} = \bar{\rho}^2$ by definition, this is exactly $\langle R \rangle = 1 + \bar{R}$. $\square$

A.40 From 10^5 templates to 10^{15} trials

Section 4.6.4 quotes two different numbers for "how many templates" a compact-binary search has to try: $\mathcal{O}(10^5\text{--}10^6)$ for the template bank itself, and $\mathcal{O}(10^{15})$ once the search is actually run on a year of data. The two are not in tension — they are counting different things, and the second is simply the first multiplied by the number of arrival times that must also be tried.

A template bank of $\sim 10^5$ templates covers the *intrinsic* parameters (masses and spins): each point of the bank is one waveform shape. But every one of those shapes must also be tested against every plausible arrival time t_* at which the signal could have entered the detector's band, since t_* is not known in advance either. Two gravitational waves emitted seconds apart are, as far as the search is concerned, essentially two different signals to look for, so each arrival time tried counts as its own independent trial, on top of the 10^5 waveform shapes.

How many arrival times must be tried follows from a simple timing argument. The templates and the real signal need to line up to within about 3 ms for the overlap between them not to degrade appreciably; scanning one year of data ($\sim 3 \times 10^7$ s) in steps of 3 ms means trying

$$\frac{3 \times 10^7 \text{ s}}{3 \times 10^{-3} \text{ s}} \sim 10^{10}$$

distinct arrival times. Each of the $\sim 10^5$ waveform shapes must be tried at each of these $\sim 10^{10}$ arrival times, so the total number of independent trials is their product,

$$10^5 \times 10^{10} = 10^{15},$$

which is exactly the number quoted in Section 4.6.4. (In practice this scan over arrival times is not done one trial at a time, which would be hopelessly slow: Section 4.6.3 shows that a single Fast Fourier Transform evaluates the overlap at every arrival time simultaneously. But for the purpose of counting how many *independent* trials the false-alarm statistics of Section 4.5.2 must account for, the 10^{15} figure above is the relevant one, however cheaply it is computed in practice.)

A.41 Computational complexity of the matched-filter statistic

This continues Section 4.7.2, which points here for the derivation of why a brute-force, time-domain search over arrival times costs $\mathcal{O}(N^2)$, and why passing to the frequency domain — the trick of Section 4.6.3, (4.66) — brings it down to $\mathcal{O}(N \log N)$.

Proposition A.33. *Let the detector strain $s(t)$ and the template $h(t; \theta)$ be digitised into time series of length N , sampled at uniform intervals Δt . Testing every possible arrival time by direct time-domain cross-correlation costs $\mathcal{O}(N^2)$ operations; a single Fast Fourier Transform of the same data costs $\mathcal{O}(N \log N)$.*

Proof. To test whether a signal entered the detector at the discrete time step $j \in \{0, 1, \dots, N-1\}$, we must evaluate the discrete cross-correlation

$$y[j] = \sum_{k=0}^{N-1} s[k] h[k-j; \theta], \quad (\text{A.4})$$

the discrete analogue of the scalar product (4.39). Two factors compound:

- **Cost per trial.** For one fixed j , (A.4) is a dot product of two length- N vectors: N multiplications and $N-1$ additions, i.e. $\mathcal{O}(N)$ operations.
- **Number of trials.** The arrival time is not known in advance, so every one of the N samples in the data stream is a candidate, and (A.4) must be evaluated for all N values of j .

Multiplying the two, a brute-force search over all arrival times costs

$$N \text{ (trials)} \times \mathcal{O}(N) \text{ (cost per trial)} = \mathcal{O}(N^2).$$

The Fast Fourier Transform evaluates (A.4) for *every* j at once, at a cost of $\mathcal{O}(N \log N)$ — this is the standard complexity of the FFT algorithm, not something specific to matched filtering, and we do not re-derive it here. $\square$

For a typical observing run, $N \sim 10^{10}$ samples (Appendix A.40), so $\mathcal{O}(N^2)$ operations per template is far beyond what is affordable, while $\mathcal{O}(N \log N)$ is exactly what makes the single-FFT trick of Section 4.6.3 practical.

A.42 Spectral leakage from an abruptly truncated signal

This continues Section 5.1.1, which points here for a simple worked example of why an abruptly truncated (untapered) waveform contaminates its own spectrum with spurious high-frequency content, and why tapering the edges instead of cutting them off sharply avoids this.

Take the simplest possible periodic signal, a pure cosine of frequency f_0 , and suppose it is recorded only for a finite stretch of time, of duration T , starting at $t = 0$. Recording only a finite stretch is exactly what *truncation* means, and it can be written as a product:

$$x(t) = \cos(2\pi f_0 t) \cdot w(t), \quad w(t) = \begin{cases} 1 & 0 \leq t \leq T \\ 0 & \text{otherwise,} \end{cases}$$

where $w(t)$, equal to 1 inside the recorded window and 0 outside it, is called a **rectangular window**: multiplying by it is exactly what an abrupt start and end does to an otherwise infinite signal.

A cosine lasting forever has an extremely simple spectrum: a single pair of spikes, at $f = \pm f_0$ and nowhere else. Truncating it changes this completely, because of a general fact about the Fourier transform: multiplying two functions in time corresponds to *convolving* their Fourier transforms in frequency. The Fourier transform of the rectangular window $w(t)$ itself is

$$\tilde{w}(f) = T \operatorname{sinc}(fT), \quad \operatorname{sinc}(x) \equiv \frac{\sin(\pi x)}{\pi x},$$

a central lobe at $f = 0$ together with an infinite sequence of smaller **side lobes** on either side of it, decaying only as slowly as $1/f$ as f moves away from 0. Convolution of the two clean spikes of the infinite cosine with this sinc function replaces each spike with a copy of $\tilde{w}(f)$ centred on it: the spectrum of the truncated cosine $x(t)$ is no longer a clean pair of spikes, but a pair of central lobes at $\pm f_0$ each trailing the same slowly-decaying side lobes, spreading power into frequencies far from f_0 that were never present in the original, untruncated signal. This is exactly the **spectral leakage** referred to in Section 5.1.1: power “leaking” out of the true frequency f_0 into spurious ones, including high-frequency ones, purely as an artefact of the sharp, rectangular cutoff.

Tapering avoids this because it replaces the rectangular window $w(t)$ — a function with a sudden jump at its two edges — with a smooth window that rises from 0 to 1 (and back down) gradually, with no jump and no sharp corner. The Fourier transform of a smooth, gradually-rising window decays much faster than the slow $1/f$ tail of the sinc function above; qualitatively, the smoother a function is in time, the faster its Fourier transform decays in frequency, and a rectangular window, with a discontinuous jump at each edge, is about as far from smooth as a bounded function can be. Easing the signal in and out, as the tapering of Section 5.1.1 does, therefore leaves a spectrum with much weaker side lobes, and correspondingly far less spurious high-frequency content.

A.43 Rescaling a waveform to a target optimal SNR

This continues Section 5.1.4, which points here for the proof that rescaling a waveform’s amplitude by $A = \text{target_snr}/\rho_0$ produces a waveform whose optimal SNR is exactly `target\_snr`.

Proposition A.34. *Let $h(t)$ be a waveform with (unscaled) optimal SNR $\rho_0 = \rho_{\text{opt}}[h]$, computed against a fixed noise spectral density $S_n(f)$ via (4.44), and suppose $\rho_0 > 0$. For any target value $\rho_{\text{target}} > 0$, the rescaled waveform $Ah(t)$, with*

$$A = \frac{\rho_{\text{target}}}{\rho_0},$$

has optimal SNR $\rho_{\text{opt}}[Ah] = \rho_{\text{target}}$ exactly.

Proof. Write $\tilde{h}(f)$ for the Fourier transform of $h(t)$. The Fourier transform is linear, so rescaling the waveform in the time domain by the real constant A rescales its Fourier transform by the same constant:

$$\widetilde{Ah}(f) = A\tilde{h}(f).$$

By definition (4.44), the optimal SNR of Ah is

$$\rho_{\text{opt}}^2[Ah] = 4 \int_0^\infty df \frac{|\widetilde{Ah}(f)|^2}{S_n(f)} = 4 \int_0^\infty df \frac{A^2 |\tilde{h}(f)|^2}{S_n(f)} = A^2 \cdot 4 \int_0^\infty df \frac{|\tilde{h}(f)|^2}{S_n(f)} = A^2 \rho_{\text{opt}}^2[h],$$

where the constant A^2 was pulled out of the integral since it does not depend on f . Taking the (positive) square root of both sides,

$$\rho_{\text{opt}}[Ah] = A \rho_{\text{opt}}[h] = A \rho_0.$$

Substituting $A = \rho_{\text{target}}/\rho_0$ gives

$$\rho_{\text{opt}}[Ah] = \frac{\rho_{\text{target}}}{\rho_0} \cdot \rho_0 = \rho_{\text{target}},$$

as claimed. □

The content of this proof is simple: the optimal SNR is, by (4.44), the square root of an integral that is *quadratic* in the waveform $\tilde{h}(f)$, so it scales linearly — not quadratically — with an overall amplitude rescaling of the waveform. This is exactly the property **sigma** relies on when Section 5.1.4 rescales a signal to a target SNR by a single multiplication, with nothing left to check numerically.

A.44 What `inverse_spectrum_truncation` does, and why

This continues Section 5.1.5, which points here for a fuller explanation of the `inverse_spectrum_truncation` step applied to the noise spectral density $S_n(f)$ before whitening.

Why it is needed. Whitening divides the data by $S_n^{1/2}(f)$ in the frequency domain. Dividing by something in the frequency domain is, in the time domain, the same as *convolving* with the inverse of that something: so dividing by $S_n^{1/2}(f)$ is equivalent to convolving the data with a filter whose frequency-domain form is $1/S_n^{1/2}(f)$. If $S_n(f)$ takes very small values, or varies sharply, at some frequencies, this inverse filter can have a very long impulse response in the time domain — in the worst case, arbitrarily long. A filter that long is both expensive to apply and, on a signal of finite duration, produces significant artefacts (*ringing*) near the edges of the data, of exactly the same origin as the sharp-edge spectral leakage of Appendix A.42, but now affecting the noise spectrum itself rather than a waveform. `inverse_spectrum_truncation` exists to prevent this, by guaranteeing in advance that the whitening filter’s impulse response cannot exceed a chosen maximum length [72].

What it does. Concretely, given the PSD $S_n(f)$ and a maximum length `max_filter_len` (in samples), the function performs the following steps [27]:

1. It builds the inverse amplitude spectral density $1/S_n^{1/2}(f)$ in the frequency domain — exactly the filter that whitening is about to convolve the data with — setting it to a fixed value below the chosen low-frequency cutoff, where $S_n(f)$ is not considered reliable.
2. It computes the inverse Fourier transform of this filter, to see what it actually looks like in the time domain: this is its impulse response.
3. It *truncates* this impulse response, keeping only `max_filter_len` samples of it and setting the rest to zero — discarding whatever part of the filter would have extended beyond the allowed length. When `trunc_method="hann"`, the edge of this truncation is tapered smoothly with a Hann window, rather than cut off sharply, for exactly the reason of Appendix A.42: a sharp cutoff in time introduces its own spurious high-frequency content, which a smooth taper avoids.
4. It Fourier transforms this truncated impulse response back to the frequency domain, obtaining a new, smoothed version of the inverse amplitude spectral density — one that is, by construction, exactly representable by a filter of the requested finite length.
5. Finally, it inverts this smoothed quantity once more, handing back a new noise spectral density $S_n(f)$: not the original one, but one whose own inverse-amplitude-spectral-density filter is guaranteed to have a bounded, well-behaved impulse response.

The net effect is a version of $S_n(f)$ that has been coarse-grained, or smoothed, just enough that whitening with it corresponds to a short, finite, well-behaved filter — at the cost of a small, controlled loss of fine spectral detail, in exchange for a whitening step free of the long-filter and edge-ringing problems described above.

A.45 Two ways to average an AUC under cross-validation

Two numbers reported for the same classifier under k -fold cross-validation (Section 6.1.3) are both called the area under the ROC curve, or AUC, and they can differ. Both are built from the same k sets of held-out predictions, one set per fold, but combined in different orders.

- **Averaging per-fold AUCs.** For each of the k folds separately, compute an AUC using only that fold’s own held-out probabilities and labels, then average the k resulting numbers. Every comparison entering any of these k AUCs is between two scores produced by the same trained model, since each fold has its own model.

- **Pooling, then computing one AUC.** Instead, collect the held-out predictions of all k folds into a single list spanning the whole dataset, and compute one AUC on that combined list. This mixes comparisons across folds: a signal-class example scored by one fold’s model can be compared against a noise-class example scored by a different fold’s model.

An AUC is built entirely from comparisons of this kind: for every pair of one signal-class and one noise-class example, it asks whether the signal-class example’s score is the higher of the two, and averages the answer over all such pairs. Averaging per-fold AUCs never compares scores from two different trained models, so it is only ever asking whether each model ranks its *own* examples correctly. Pooling first asks the same question across models as well, and a model trained independently on a different fold need not output probabilities on quite the same scale as another — one may typically report slightly higher or lower numbers than another, even when both rank their own examples equally well. Such a difference in scale costs nothing in the per-fold average, since it never has to compare across it, but it can cost the pooled AUC every pair it touches.

A small example makes the mechanism concrete. Suppose $k = 2$, and each fold’s model perfectly separates its own two held-out examples: fold A gives its signal-class example a score of 0.9 and its noise-class example 0.1; fold B, whose model happens to report higher numbers across the board, gives its signal-class example 0.99 and its noise-class example 0.6. Within each fold the signal score exceeds the noise score, so both per-fold AUCs equal 1, and so does their average. Pooled together, though, fold B’s noise-class score of 0.6 now sits *above* fold A’s signal-class score of 0.9 — not because fold B’s model ranks its examples any worse than fold A’s does, but because fold B’s scores run higher overall. This one ordering mistake, invisible to the per-fold average, is exactly what lowers the pooled AUC below 1.

This is the situation for the convolutional networks of Section 6.2.3: five independently trained networks, each confident in its own ranking, whose output scales are not perfectly aligned with one another. The two-dimensional network’s average of 0.901 falls to 0.883 once pooled; the effect is much larger for the one-dimensional network, whose average of 0.896 falls to 0.715, consistent with the fold-to-fold instability already noted for it in Section 6.2.3.

A.46 Correlated features and unstable coefficients

Section 6.2.4 noted that a fitted logistic-regression weight is not a stable quantity when two features carry nearly the same information, since the weights can then trade off against one another without changing a single prediction. The following toy example makes precise why.

Suppose a price is to be predicted from the area of a house, and that area happens to be recorded twice, once in square metres and once in square feet:

$$x_1 = \text{area in square metres}, \quad x_2 = \text{area in square feet} = 10.76 x_1.$$

The two features carry identical information, since one is just a fixed multiple of the other. Fitting a linear model

$$\text{price} = w_1 x_1 + w_2 x_2 + b$$

to data generated by the true relationship $\text{price} = 3000 x_1$ does not pick out one pair of weights: because $x_2 = 10.76 x_1$ identically, any pair (w_1, w_2) satisfying $w_1 + 10.76 w_2 = 3000$ reproduces the same predictions on every example, and so fits the data exactly as well as any other. A few of the infinitely many solutions:

- $w_1 = 3000, w_2 = 0$;
- $w_1 = 0, w_2 \approx 278.8$;
- $w_1 = 10,000, w_2 \approx -650.5$ — a large positive weight and a large negative one, nearly cancelling;
- $w_1 = -50,000, w_2 \approx 4924$ — the same cancellation, pushed further still.

All four give identical predictions, and nothing in the fitting procedure prefers one over another: a small change to the training data, such as moving to a different cross-validation fold, can move the fitted weights from one of these solutions to a completely different one. The model itself is not at fault, and its predictions do not change; what changes is only the individual weights, which carry no meaning of

their own once the features are this redundant — "the weight on x_1 is +3000" says nothing about how important area in square metres really is, since a refit could just as easily have reported $-50,000$.

The windowed Betti-curve areas of Section 6.2.4 are not perfectly correlated in the way x_1 and x_2 are here, but Figure 6.19 shows several pairs correlated above 0.9, close enough to the same mechanism: the fitted weights on such a pair are free to move against one another over a wide range while leaving the model's predictions essentially unchanged, which is why Figure 6.18's individual signs and sizes should be read with the caution given there.

Appendix B

Pattern functions and the angular response

It is useful, for what follows, to rewrite (4.14) in a way that separates the polarization structure from the wave itself. We introduce the two polarization tensors $e_{ij}^A(\hat{n})$, with $A = +, \times$, defined for a wave along $\hat{n}$ from an orthonormal pair of vectors in the transverse plane [12]. A general wave propagating along $\hat{n}$ is then a superposition over frequencies of both polarizations,

$$h_{ij}(t, \mathbf{x}) = \sum_{A=+, \times} \int_{-\infty}^{\infty} df e_{ij}^A(\hat{n}) \tilde{h}_A(f) e^{-2\pi i f (t - \hat{n} \cdot \mathbf{x}/c)}. \quad (\text{B.1})$$

Equation (B.1) is exactly the form we will use later: it is the starting point for the discussion of the detector pattern functions and of the angular sensitivity below, where the geometry of the detector selects a specific combination of e_{ij}^+ and $e_{ij}^\times$.

Before turning to the filtering problem we recall briefly how the abstract scalar input $h(t)$ of eq. (4.23) is related to the physical wave arriving from a definite direction in the sky. A gravitational wave propagating along a direction $\hat{n}$ carries two polarisations, conventionally denoted $+$ and $\times$, and the scalar strain seen by the detector is a linear combination of them,

$$h(t) = F_+(\theta, \phi; \psi) h_+(t) + F_\times(\theta, \phi; \psi) h_\times(t), \quad (\text{B.2})$$

where (θ, ϕ) are the polar angles giving the source position in a frame attached to the detector and ψ is the polarisation angle. The coefficients F_+ and $F_\times$ are the *pattern functions* (or antenna response functions); they follow from contracting the detector tensor D^{ij} with the polarisation tensors $e_{ij}^+(\hat{n})$ and $e_{ij}^\times(\hat{n})$. For an interferometer with orthogonal arms one finds the familiar expressions

$$F_+(\theta, \phi; \psi) = \frac{1}{2}(1 + \cos^2 \theta) \cos 2\phi \cos 2\psi - \cos \theta \sin 2\phi \sin 2\psi, \quad (\text{B.3})$$

$$F_\times(\theta, \phi; \psi) = \frac{1}{2}(1 + \cos^2 \theta) \cos 2\phi \sin 2\psi + \cos \theta \sin 2\phi \cos 2\psi. \quad (\text{B.4})$$

Two features of these functions matter for what follows. First, a single interferometer is an intrinsically *wide-angle* instrument: its response is appreciable over a large fraction of the sky and falls to zero only along a few blind directions. This is excellent for catching rare transients but very poor for localising them, which is why a *network* of detectors is needed to triangulate a source. Second, the pattern functions are of order unity and vary slowly; for a ground-based instrument following a binary coalescence for the few seconds to minutes it spends in band, (θ, ϕ) can be treated as constant, and the antenna response contributes only an overall amplitude and phase. This is what allows the sky position to be largely decoupled from the intrinsic detection problem and, as we shall see in Section 4.6.3, to be treated as an *extrinsic* parameter. The characteristic antenna pattern is a three-dimensional "peanut" shape, nearly omnidirectional, which is why sky localisation requires a network of geographically separated detectors.

Appendix C

Full code listing: the toy-model TDA pipeline

This appendix reproduces, in full, the code behind the toy-model pipeline discussed and commented in Section 3: a self-contained notebook that builds three raw timeseries (a cosine burst, white Gaussian noise, and their sum), embeds each into a point cloud via a Takens embedding, and runs the full topological pipeline (Vietoris-Rips persistent homology, Betti curves and their area, scalar summaries, persistence images) on each. The code is split below into the same eight sections as the original notebook.

C.1 Imports

```
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4
5 from gtda.time_series import SingleTakensEmbedding, TakensEmbedding
6 from gtda.homology import VietorisRipsPersistence
7 from gtda.diagrams import BettiCurve, PersistenceEntropy, Scaler,
8     NumberOfPoints, PersistenceImage
9 from gtda.plotting import plot_point_cloud, plot_diagram
10 from gtda.pipeline import Pipeline
11
12 from sklearn.decomposition import PCA
13 from sklearn.metrics import pairwise_distances
14 from sklearn.preprocessing import FunctionTransformer
15 from sklearn.pipeline import FeatureUnion
16
17 from tqdm.auto import tqdm
18
19 plt.rcParams.update({"figure.dpi": 120, "axes.grid": True, "grid.alpha":
20     0.5})
```

C.2 Configuration

```
1 ##### CONFIGURATION #####
2
3 # --- Sampling ---
4 SAMPLE_RATE = 2048 # Hz
5 DURATION = 4.0 # s, length of every timeseries
6
7 # --- Cosine burst parameters ---
8 COSINE_FREQUENCY = 25.0 # Hz
9 COSINE_AMPLITUDE = 3.5
```

```

10 COSINE_DURATION = DURATION * 10/10 # s, length of the cosine burst
    inside the 4s window
11 COSINE_CENTER_TIME = 2.0 # s, where the burst is centered inside the 4s
    window
12
13 # --- Noise parameters ---
14 NOISE_STD = 1.0 # standard deviation of the white Gaussian noise
15 NOISE_SEED = 1234
16
17 # --- Takens embedding: automatic parameter search ---
18 SEARCH_MAX_EMBEDDING_DIMENSION = 35
19 SEARCH_MAX_TIME_DELAY = 140
20 STRIDE = 18
21
22 # --- PCA dimension used for the Rips complex / all topological features (PCA
    branch) ---
23 PCA_N_COMPONENTS = 3
24
25 # --- Persistent homology ---
26 HOMOLOGY_DIMENSIONS = [0, 1, 2]
27 BOOL_COLLAPSE = True
28
29 # --- Betti curves / partial integrals ---
30 N_BC_BINS = 300
31 N_INTEGRALS = 10
32
33 # --- Persistence images ---
34 PI_SIGMA = 0.03
35 PI_N_BINS = 64
36
37 # --- Point cloud plot ---
38 POINT_CLOUD_MARKER_SIZE = 1.5 # smaller = easier to see internal structure
39 #####
40
41 assert COSINE_DURATION <= DURATION, "Cosine burst must fit inside the 4s
    window."
42 _half = COSINE_DURATION / 2.0
43 assert 0.0 <= COSINE_CENTER_TIME - _half and COSINE_CENTER_TIME + _half <=
    DURATION, (
44     "Cosine burst (centered at COSINE_CENTER_TIME, length COSINE_DURATION) "
45     "must fit fully inside [0, DURATION]."
46 )

```

C.3 Build the three timeseries

```

1 def generate_cosine_burst(sample_rate, duration, amplitude, frequency,
2     burst_duration, center_time):
3     """Cosine burst (no tapering) placed inside a 'duration'-long zero buffer
4     ."""
5     n_total = int(round(duration * sample_rate))
6     n_burst = int(round(burst_duration * sample_rate))
7
8     t_burst = np.arange(n_burst) / sample_rate
9     burst = amplitude * np.cos(2.0 * np.pi * frequency * t_burst)
10
11     buf = np.zeros(n_total, dtype=np.float64)
12     start = int(round(center_time * sample_rate)) - n_burst // 2
13     end = start + n_burst
14     if start < 0 or end > n_total:
15         raise ValueError("Cosine burst extends outside the buffer - adjust "

```

```

15         "COSINE_CENTER_TIME / COSINE_DURATION.")
16     buf[start:end] = burst
17     return buf
18
19
20 def generate_white_noise(sample_rate, duration, std, seed):
21     n_total = int(round(duration * sample_rate))
22     rng = np.random.default_rng(seed)
23     return rng.normal(loc=0.0, scale=std, size=n_total)

```

```

1 time_axis = np.arange(int(round(DURATION * SAMPLE_RATE))) / SAMPLE_RATE
2
3 cosine_ts = generate_cosine_burst(
4     SAMPLE_RATE, DURATION, COSINE_AMPLITUDE, COSINE_FREQUENCY,
5     COSINE_DURATION, COSINE_CENTER_TIME,
6 )
7 noise_ts = generate_white_noise(SAMPLE_RATE, DURATION, NOISE_STD, NOISE_SEED)
8 sum_ts = cosine_ts + noise_ts
9
10 sample_names = ["Pure cosine", "Pure noise", "Cosine + noise"]
11 list_timeseries = [cosine_ts, noise_ts, sum_ts]
12
13 print(f"Each timeseries has {len(cosine_ts)} samples ({DURATION}s @ {
14     SAMPLE_RATE}Hz)")

```

Plot the three raw timeseries.

```

1 fig, axes = plt.subplots(3, 1, figsize=(8, 8), sharex=True)
2 for ax, name, ts in zip(axes, sample_names, list_timeseries):
3     ax.plot(time_axis, ts)
4     ax.set_ylabel("Amplitude")
5     ax.set_xlabel("Time [s]")
6     ax.set_title(name)
7
8 fig.tight_layout()
9 plt.show()

```

C.4 Takens embedding — automatic parameter search

We use `SingleTakensEmbedding(parameters_type='search')` independently on each of the 3 signals, then take the median of the found (`dimension`, `time_delay`) pairs. Using one shared pair for all three keeps the point clouds on the same axes, so they are directly comparable.

```

1 found_dimensions = []
2 found_time_delays = []
3
4 for name, ts in zip(sample_names, tqdm(list_timeseries, desc="Searching
5     embedding parameters")):
6     ste = SingleTakensEmbedding(
7         parameters_type="search",
8         time_delay=SEARCH_MAX_TIME_DELAY,
9         dimension=SEARCH_MAX_EMBEDDING_DIMENSION,
10        stride=STRIDE,
11        n_jobs=-1,
12    )
13    ste.fit(ts)
14    found_dimensions.append(ste.dimension_)
15    found_time_delays.append(ste.time_delay_)
16    print(f" {name}: best dimension={ste.dimension_}, best time_delay={ste.
17        time_delay_}")

```

```

16 EMBEDDING_DIMENSION = int(np.median(found_dimensions))
17 TIME_DELAY          = int(np.median(found_time_delays))
18
19 print(f"\nOK EMBEDDING_DIMENSION (median across the 3 signals) = {
    EMBEDDING_DIMENSION}")
20 print(f"OK TIME_DELAY          (median across the 3 signals) = {TIME_DELAY}")
21 print(f"OK STRIDE              = {STRIDE}")

```

C.5 Takens embedding — three point clouds per signal

From here on we carry three parallel branches, always clearly labeled: a **PCA branch** (each raw timeseries is embedded with the shared (EMBEDDING_DIMENSION, TIME_DELAY, STRIDE) found above, then PCA-reduced to PCA_N_COMPONENTS dimensions); a **no-PCA (dim=3) branch** (each raw timeseries is embedded directly with dimension=3, no PCA at all, using the same TIME_DELAY/STRIDE found above); and a **no-PCA (searched dim) branch** (the very same embedded_clouds from the search above, dimension = EMBEDDING_DIMENSION, used directly for the Rips complex / topological features — no PCA, no dimensionality reduction at all). A separate PCA projection, always fixed at 3 components, is also computed purely for plotting the PCA-branch point cloud.

```

1 embedder = TakensEmbedding(
2     time_delay=TIME_DELAY,
3     dimension=EMBEDDING_DIMENSION,
4     stride=STRIDE,
5 )
6
7 # TakensEmbedding (unlike SingleTakensEmbedding) operates on the whole
8 # collection of series at once, embedding each one independently.
9 embedded_clouds = embedder.fit_transform(list_timeseries)
10 for name, cloud in zip(sample_names, embedded_clouds):
11     print(f"{name}: embedded point cloud shape (dimension={
        EMBEDDING_DIMENSION}) = {cloud.shape}")
12
13 # --- PCA branch: PCA-reduce the (found-dimension) embedded clouds ---
14 pca_feature_clouds = [PCA(n_components=PCA_N_COMPONENTS).fit_transform(cloud)
15     for cloud in embedded_clouds]
16
17 # Separate, ALWAYS-3D projection used only for plotting the PCA-branch point
18 # cloud
19 PLOT_PCA_N_COMPONENTS = 3
20 pca_plot_clouds = [PCA(n_components=PLOT_PCA_N_COMPONENTS).fit_transform(
21     cloud)
22     for cloud in embedded_clouds]
23
24 # --- No-PCA branch: embed directly with dimension=3 (same TIME_DELAY/STRIDE)
25 # ---
26 embedder_dim3 = TakensEmbedding(
27     time_delay=TIME_DELAY,
28     dimension=3,
29     stride=STRIDE,
30 )
31 raw_dim3_clouds = embedder_dim3.fit_transform(list_timeseries)
32 for name, cloud in zip(sample_names, raw_dim3_clouds):
33     print(f"{name}: raw dimension-3 embedded point cloud shape = {cloud.shape}
34         ")

```

Plot the point clouds with giotto-tda's built-in `plot_point_cloud` — PCA branch first, then the no-PCA (dim=3) branch. The no-PCA (searched dim) branch is only plotted if EMBEDDING_DIMENSION happens to be 2 or 3; otherwise we just print a note and skip it.

```

1 print("### PCA branch (Takens embedding + PCA to 3D) ###")

```

```

2 for name, cloud in zip(sample_names, pca_plot_clouds):
3     fig = plot_point_cloud(cloud)
4     fig.update_traces(marker=dict(size=POINT_CLOUD_MARKER_SIZE))
5     fig.update_layout(title=f"[PCA] Takens embedding + 3D PCA - {name}")
6     fig.show()
7
8 print("### No-PCA (dim=3) branch (raw dimension-3 Takens embedding) ###")
9 for name, cloud in zip(sample_names, raw_dim3_clouds):
10     fig = plot_point_cloud(cloud)
11     fig.update_traces(marker=dict(size=POINT_CLOUD_MARKER_SIZE))
12     fig.update_layout(title=f"[No PCA dim=3] Raw dimension-3 Takens embedding
13                          - {name}")
14     fig.show()
15
16 print(f"### No-PCA (searched dim={EMBEDDING_DIMENSION}) branch ###")
17 if EMBEDDING_DIMENSION in (2, 3):
18     for name, cloud in zip(sample_names, embedded_clouds):
19         fig = plot_point_cloud(cloud)
20         fig.update_traces(marker=dict(size=POINT_CLOUD_MARKER_SIZE))
21         fig.update_layout(title=f"[No PCA searched dim={EMBEDDING_DIMENSION}]
22                          Raw Takens embedding - {name}")
23         fig.show()
24 else:
25     print(f"    skipping plot: EMBEDDING_DIMENSION = {EMBEDDING_DIMENSION}, "
26           f"    plot_point_cloud needs a 2D or 3D point cloud.")

```

C.6 Helper functions for the topological pipeline

Factored into plain functions so they can be reused identically for every branch below.

```

1 def sanitize_diagrams(diagrams):
2     """Make diagrams finite for the feature pipeline.
3
4     Vietoris-Rips can leave a few deaths as +inf (features that never die).
5     giotto's feature transformers reject non-finite values, so we cap each
6     infinite death at the largest *finite* death seen anywhere in the whole
7     collection. Any remaining non-finite point is turned into a trivial
8     zero-lifetime point.
9     """
10    d = diagrams.copy()
11    births = d[:, :, 0]
12    deaths = d[:, :, 1]
13
14    finite_deaths = deaths[np.isfinite(deaths)]
15    cap = finite_deaths.max() if finite_deaths.size else 0.0
16    inf_mask = np.isposinf(deaths)
17    n_inf = int(inf_mask.sum())
18    deaths[inf_mask] = cap
19
20    bad = ~np.isfinite(births) | ~np.isfinite(deaths)
21    n_bad = int(bad.sum())
22    births[bad] = 0.0
23    deaths[bad] = 0.0
24
25    if n_inf:
26        print(f"    sanitize_diagrams: capped {n_inf} infinite death(s) at {cap
27              :.4f}")
28    if n_bad:
29        print(f"    sanitize_diagrams: zeroed {n_bad} remaining non-finite
30              point(s)")
31    return d

```

```

1 def compute_max_persistence(persistence_diagrams, homology_dimensions=
  HOMOMOLOGY_DIMENSIONS):
2     """Maximum persistence (longest lifetime) for each homology dimension."""
3     n_samples = persistence_diagrams.shape[0]
4     n_dims = len(homology_dimensions)
5     features = np.zeros((n_samples, n_dims))
6     for i in range(n_samples):
7         for d_idx, dim in enumerate(homology_dimensions):
8             mask = persistence_diagrams[i, :, 2] == dim
9             lifetimes = persistence_diagrams[i, mask, 1] -
              persistence_diagrams[i, mask, 0]
10            if len(lifetimes) > 0:
11                features[i, d_idx] = np.max(lifetimes)
12    return features
13
14
15 def compute_total_persistence(persistence_diagrams, homology_dimensions=
  HOMOMOLOGY_DIMENSIONS):
16     """Sum of all lifetimes (total persistence) for each homology dimension."
17     """
18     n_samples = persistence_diagrams.shape[0]
19     n_dims = len(homology_dimensions)
20     features = np.zeros((n_samples, n_dims))
21     for i in range(n_samples):
22         for d_idx, dim in enumerate(homology_dimensions):
23             mask = persistence_diagrams[i, :, 2] == dim
24             lifetimes = persistence_diagrams[i, mask, 1] -
              persistence_diagrams[i, mask, 0]
25             if len(lifetimes) > 0:
26                 features[i, d_idx] = np.sum(lifetimes)
27    return features
28
29
30 def betti_partial_integrals(list_bc, bc_samplings, homology_dimensions,
  n_integrals=N_INTEGRALS):
31     """Partial integrals of Betti curves over equally-spaced filtration
32     windows."""
33     n_samples = list_bc.shape[0]
34     n_dims = len(homology_dimensions)
35     result = np.zeros((n_samples, n_dims, n_integrals), dtype=np.float64)
36     window_edges = {}
37
38     for d_idx, di in enumerate(homology_dimensions):
39         filt_axis = bc_samplings[di]
40         edges = np.linspace(filt_axis.min(), filt_axis.max(), n_integrals +
41                             1)
42         window_edges[di] = edges
43
44         for k in range(n_integrals):
45             if k < n_integrals - 1:
46                 mask = (filt_axis >= edges[k]) & (filt_axis < edges[k + 1])
47             else:
48                 mask = (filt_axis >= edges[k]) & (filt_axis <= edges[k + 1])
49
50             x_chunk = filt_axis[mask]
51             y_chunk = list_bc[:, di, :][:, mask]
52
53             if len(x_chunk) >= 2:
54                 result[:, d_idx, k] = np.trapz(y_chunk, x=x_chunk, axis=1)
55

```

```

25 flat = result.reshape(n_samples, n_dims * n_integrals)
26 return flat, window_edges

1 def max_birth_death(diagrams, homology_dimensions=HOMOLOGY_DIMENSIONS):
2     max_val = -np.inf
3     for diagram in diagrams:
4         mask = np.isin(diagram[:, 2], homology_dimensions)
5         filtered = diagram[mask]
6         birth = filtered[:, 0]
7         death = filtered[:, 1]
8         finite_death = death[np.isfinite(death)]
9         if len(birth) > 0:
10             max_val = max(max_val, np.max(birth))
11         if len(finite_death) > 0:
12             max_val = max(max_val, np.max(finite_death))
13     return max_val

```

C.7 Topological pipeline — Rips complex + all features, run once per branch

`run_topology_pipeline` takes a list of point clouds (one per sample) and a `branch_label` string, runs Vietoris-Rips persistence, sanitizes the diagrams, extracts scalar features (entropy, max/total persistence, num. points), Betti curves + partial integrals, persistence images, and finally plots everything, with every title/print prefixed by `branch_label` so results from different branches are never ambiguous.

```

1 def run_topology_pipeline(point_clouds, branch_label, sample_names=
sample_names):
2     print("\n" + "#" * 70)
3     print(f"# TOPOLOGICAL PIPELINE - [{branch_label}]" )
4     print("#" * 70)
5
6     # --- Vietoris-Rips persistent homology ---
7     max_distances = []
8     for cloud in point_clouds:
9         distances = pairwise_distances(cloud)
10        upper = distances[distances > 0]
11        max_distances.append(np.max(upper))
12    global_max_distance = max(max_distances)
13    print(f"[{branch_label}] Global max distance across the 3 point clouds: {
global_max_distance:.6f}")
14
15    persistence = VietorisRipsPersistence(
16        homology_dimensions=HOMOLOGY_DIMENSIONS,
17        max_edge_length=global_max_distance,
18        collapse_edges=BOOL_COLLAPSE,
19        n_jobs=-1,
20    )
21    list_PH_diagrams = persistence.fit_transform(point_clouds)
22    print(f"[{branch_label}] Persistence diagrams computed - shape: {
list_PH_diagrams.shape}")
23
24    # --- Sanitize ---
25    list_PH_diagrams = sanitize_diagrams(list_PH_diagrams)
26    print(f"[{branch_label}] After sanitize - NaNs: {np.isnan(
list_PH_diagrams).sum()}")
27        f"  infs: {np.isinf(list_PH_diagrams).sum()}")
28
29    # --- Scalar features ---
30    entropy = PersistenceEntropy(normalize=False, nan_fill_value=-10)

```

```

31 compute_max_persistence_transformer = FunctionTransformer(
32     func=compute_max_persistence, kw_args={"homology_dimensions":
        HOMOLOGY_DIMENSIONS})
33 compute_total_persistence_transformer = FunctionTransformer(
34     func=compute_total_persistence, kw_args={"homology_dimensions":
        HOMOLOGY_DIMENSIONS})
35 topological_features = FeatureUnion([
36     ("entropy", entropy),
37     ("maxPersistence", compute_max_persistence_transformer),
38     ("totalPersistence", compute_total_persistence_transformer),
39     ("numFeatures", NumberOfPoints(n_jobs=-1)),
40 ])
41 transformer_for_homology = Pipeline([
42     ("scaling", Scaler()),
43     ("topological_features", topological_features),
44 ])
45 features_of_homology = transformer_for_homology.fit_transform(
    list_PH_diagrams)
46 print(f"[{branch_label}] features_of_homology shape: {
    features_of_homology.shape}")
47 list_PH_diagrams_scaled = transformer_for_homology.named_steps["scaling"
    ].transform(list_PH_diagrams)
48
49 # --- Betti curves ---
50 BC = BettiCurve(n_bins=N_BC_BINS, n_jobs=-1)
51 list_bc = BC.fit_transform(list_PH_diagrams_scaled)
52 bc_samplings = BC.samplings_
53 betti_partial_ints, betti_window_edges = betti_partial_integrals(
54     list_bc, bc_samplings, homology_dimensions=HOMOLOGY_DIMENSIONS,
        n_integrals=N_INTEGRALS)
55 print(f"[{branch_label}] Betti partial integrals shape: {
    betti_partial_ints.shape}")
56
57 # --- Persistence images ---
58 PI_flat = PersistenceImage(sigma=PI_SIGMA, n_bins=PI_N_BINS, n_jobs=-1)
59 PI_linear = PersistenceImage(sigma=PI_SIGMA, n_bins=PI_N_BINS,
60     weight_function=lambda x: x, n_jobs=-1)
61 PI_quad = PersistenceImage(sigma=PI_SIGMA, n_bins=PI_N_BINS,
62     weight_function=lambda x: x**2, n_jobs=-1)
63 list_PI_flat = PI_flat.fit_transform(list_PH_diagrams_scaled)
64 list_PI_linear = PI_linear.fit_transform(list_PH_diagrams_scaled)
65 list_PI_quad = PI_quad.fit_transform(list_PH_diagrams_scaled)
66 pi_samplings_quad = PI_quad.samplings_
67 print(f"[{branch_label}] Persistence Images computed - PI_quad shape: {
    list_PI_quad.shape}")
68
69 # --- DataFrame ---
70 dims = HOMOLOGY_DIMENSIONS
71 column_names = (
72     [f"entropy_dim{d}" for d in dims] +
73     [f"max_pers_dim{d}" for d in dims] +
74     [f"total_pers_dim{d}" for d in dims] +
75     [f"num_points_dim{d}" for d in dims]
76 )
77 df = pd.DataFrame(features_of_homology, columns=column_names)
78 df.insert(0, "sample_name", sample_names)
79 df.insert(0, "branch", branch_label)
80 for d_idx, d in enumerate(HOMOLOGY_DIMENSIONS):
81     for k in range(N_INTEGRALS):
82         col_idx = d_idx * N_INTEGRALS + k
83         df[f"betti_partial_int_dim{d}_win{k}"] = betti_partial_ints[:,
            col_idx]

```

```

84 df["betti_curve"] = [list_bc[i] for i in range(len(list_bc))]
85 df["pers_img_quad"] = [list_PI_quad[i] for i in range(len(list_PI_quad))]
86 print(f"[{branch_label}] DataFrame shape: {df.shape}")
87
88 # --- Plot: persistence diagrams ---
89 global_max_ = max_birth_death(list_PH_diagrams_scaled,
    homology_dimensions=HOMOLOGY_DIMENSIONS)
90 print(f"[{branch_label}] Global max birth/death value across all diagrams
    and dimensions: {global_max_: .3f}")
91 for i, name in enumerate(sample_names):
92     fig_diag = plot_diagram(list_PH_diagrams_scaled[i])
93     fig_diag.update_layout(
94         title=f"[{branch_label}] Persistence Diagram - {name}",
95         xaxis_title="Birth", yaxis_title="Death",
96     )
97     fig_diag.update_xaxes(range=[0, global_max_ * 1.1])
98     fig_diag.update_yaxes(range=[0, global_max_ * 1.1])
99     fig_diag.show()
100
101 # --- Scalar feature diagnostics ---
102 for i, name in enumerate(sample_names):
103     print("\n" + "=" * 60)
104     print(f"[{branch_label}] DIAGNOSTICS FOR {name}")
105     print("=" * 60)
106     header = f"{'Scalar Feature:':<25}"
107     for d in HOMOLOGY_DIMENSIONS:
108         header += f" | Dim {d:<7}"
109     print(header)
110     print("-" * len(header))
111     scalar_feats = ["entropy", "max_pers", "total_pers", "num_points"]
112     for feat in scalar_feats:
113         row_str = f"{feat:<25}"
114         for d in HOMOLOGY_DIMENSIONS:
115             val = df[f"{feat}_dim{d}"].iloc[i]
116             row_str += f" | {val:<10.4f}"
117         print(row_str)
118
119 # --- Betti curve plots ---
120 BC_plotter = BettiCurve()
121 BC_plotter.homology_dimensions_ = tuple(HOMOLOGY_DIMENSIONS)
122 BC_plotter.samplings_ = bc_samplings
123 all_bc_to_plot = np.stack(df["betti_curve"])
124 for i, name in enumerate(sample_names):
125     fig = BC_plotter.plot(all_bc_to_plot, sample=i, homology_dimensions=
        HOMOLOGY_DIMENSIONS)
126     fig.update_layout(title=f"[{branch_label}] Betti Curves - {name}")
127     fig.update_yaxes(type="log")
128     fig.show()
129
130 # --- Persistence image plots ---
131 all_PI_quad_to_plot = np.stack(df["pers_img_quad"])
132 PI_plotter = PersistenceImage()
133 PI_plotter.homology_dimensions_ = tuple(HOMOLOGY_DIMENSIONS)
134 PI_plotter.samplings_ = pi_samplings_quad
135 for d in HOMOLOGY_DIMENSIONS:
136     for i, name in enumerate(sample_names):
137         fig_pi = PI_plotter.plot(all_PI_quad_to_plot, sample=i,
            homology_dimension_idx=d)
138         fig_pi.update_layout(title=f"[{branch_label}] Persistence Image (
            Quad) - {name} - Hom_Dim {d}")
139         fig_pi.show()
140

```

```

141     return {
142         "df": df,
143         "list_PH_diagrams": list_PH_diagrams,
144         "list_PH_diagrams_scaled": list_PH_diagrams_scaled,
145         "bc_samplings": bc_samplings,
146         "pi_samplings_quad": pi_samplings_quad,
147         "global_max_distance": global_max_distance,
148     }

```

C.8 Run the pipeline for all three branches

```

1 results_pca = run_topology_pipeline(pca_feature_clouds, "PCA")
2 results_nopca_dim3 = run_topology_pipeline(raw_dim3_clouds, "No PCA dim
   =3")
3 results_nopca_searched = run_topology_pipeline(embedded_clouds, f"No PCA
   searched dim={EMBEDDING_DIMENSION}")

```

Appendix D

Full code listing: the dataset-generation pipeline

This appendix reproduces, in full, the code behind the dataset-generation pipeline discussed and commented section by section in Section 5.1: a self-contained notebook that samples random binary parameters, generates and tapers a gravitational-wave waveform, places it at a random merger time, generates coloured detector noise from the Advanced LIGO noise spectral density, rescales the signal (not the noise) to a target SNR, whitens and band-passes the result, and assembles and saves the resulting dataset. The code is split below into the same cells as the original notebook.

D.1 Imports

```
1 from dataclasses import dataclass, field
2 from functools import lru_cache
3 from typing import Optional
4
5 import numpy as np
6 import matplotlib.pyplot as plt
7
8 from pycbc.waveform import get_td_waveform
9 from pycbc.psd import aLIGOZeroDetHighPower, inverse_spectrum_truncation
10 from pycbc.noise import noise_from_psd
11 from pycbc.types import TimeSeries
12 from pycbc.filter import sigma
13
14 import pickle
15 import os
16 from tqdm.notebook import tqdm
17
18 import warnings
19 warnings.filterwarnings("ignore", "Wswiglal-redirect-stdio")
20
21 plt.rcParams.update({"figure.dpi": 120, "axes.grid": True, "grid.alpha":
    0.5})
```

D.2 Configuration

```
1 @dataclass
2 class GWConfig:
3     # --- Sampling ---
4     sample_rate: int = 2048 # Hz
5
6     # --- Frequencies [Hz] ---
```

```

7   f_gen: float = 15.0      # waveform generation lower freq
8   f_low: float = 20.0     # SNR / whitening lower freq
9   f_high: float = 900.0   # SNR / whitening upper freq
10
11  # --- Time windows [s] ---
12  buffer_duration: float = 16.0
13  crop_duration: float = 4.0
14  crop_start_time: float = 6.0
15  tc_min_in_crop: float = 3.0
16  tc_max_in_crop: float = 3.8
17
18  # --- BBH parameter ranges ---
19  mtot_min: float = 30.0
20  mtot_max: float = 100.0
21  q_min: float = 1.0
22  q_max: float = 5.0
23  m1_max: float = 80.0
24  m2_min: float = 5.0
25
26  # --- Target SNR ---
27  snr_min: float = 6.0
28  snr_max: float = 30.0
29
30  # --- Waveform model ---
31  approximant: str = "SEOBNRv4"
32  distance_mpc: float = 1.0
33
34  # --- Tapering [s] ---
35  start_taper: float = 0.5
36  end_taper: float = 0.05
37
38  # --- Whitening ---
39  max_filter_duration: float = 2.0
40
41  # --- Output ---
42  dtype: str = "float32"
43
44  # --- Random seed ---
45  seed: int = 1234
46
47  # --- Derived ---
48  @property
49  def dt(self) -> float:
50      return 1.0 / self.sample_rate
51  @property
52  def buffer_n(self) -> int:
53      return int(self.buffer_duration * self.sample_rate)
54  @property
55  def crop_n(self) -> int:
56      return int(self.crop_duration * self.sample_rate)
57  @property
58  def crop_end_time(self) -> float:
59      return self.crop_start_time + self.crop_duration
60  @property
61  def crop_slice(self) -> slice:
62      i0 = int(self.crop_start_time * self.sample_rate)
63      return slice(i0, i0 + self.crop_n)
64  @property
65  def f_high_safe(self) -> float:
66      return min(self.f_high, 0.5 * self.sample_rate)

```

D.3 Dataset-level parameters

```
1 ##### DATASET-LEVEL PARAMETERS #####
2 folder = "../Datasets5"          # output folder (matches the extractor
   default)
3
4 num_signals = 10000              # total number of ML samples to generate
5                                  # (balanced: ~half signal+noise, ~half noise-
   only)
6
7 # Bool, if True also save the per-sample whitened pure signal / pure noise
   crops
8 BOOL_SAVE_PURE_SIGNAL_AND_NOISE = False
9
10 # Small "test topological features" file: [pure signal, pure noise, signal+
   noise]
11 # built from a SINGLE fixed waveform + noise realisation (all whitened crops)
   .
12 BOOL_TEST_TOPOLOGICAL_FEATURES = False
13
14
15 # Single GWConfig instance drives the whole generation pipeline.
16 cfg = GWConfig()
17
18 # ---- Bookkeeping values kept ONLY so the extractor's dict keys still exist.
   ----
19 # The new pipeline samples mass & SNR per-signal, so these scalars are now
   just
20 # metadata describing the configured ranges rather than fixed grid values.
21 PSD_CONST_VALUE = np.nan        # no longer a flat-PSD constant (
   coloured aLIGO PSD now)
22 M_total = cfg.mtot_max          # representative; true total mass is
   per-signal
23 EPSILON = np.nan               # not used by the new amplitude
   scaling
24 num_snr_values = 0              # 0 => "continuously sampled", not a
   grid
25 num_mass_ratio_values = 0       # 0 => "continuously sampled", not a
   grid
26
27 # Parameters for the optional topological-test file
28 mtot_test = 60.0
29 q_test = 2.0
30 SNR_TEST_MODE = 50.0
31 #####
```

D.4 Sampling binary parameters

```
1 def sample_bbh_parameters(cfg: GWConfig,
2                             rng: np.random.Generator,
3                             max_tries: int = 100) -> dict:
4     """Sample (m1, m2) from total mass and mass ratio q = m1/m2 >= 1."""
5     for _ in range(max_tries):
6         mtot = rng.uniform(cfg.mtot_min, cfg.mtot_max)
7         q = rng.uniform(cfg.q_min, cfg.q_max)
8         m2 = mtot / (1.0 + q)
9         m1 = q * m2
10        if cfg.m2_min <= m2 <= m1 <= cfg.m1_max:
11            return {"mass1": float(m1),
```

```

12         "mass2": float(m2),
13         "q": float(q),
14         "total_mass": float(mtot),
15         "coa_phase": float(rng.uniform(0.0, 2.0 * np.pi))}
16     raise RuntimeError("Could not sample valid masses within range.")

```

D.5 Tapering the waveform edges

```

1 def apply_edge_taper(h: np.ndarray, cfg: GWConfig) -> np.ndarray:
2     """Half-Hann ramp at start, short cosine ramp at end."""
3     h = h.copy()
4     n_start = min(int(cfg.start_taper * cfg.sample_rate), len(h))
5     n_end = min(int(cfg.end_taper * cfg.sample_rate), len(h))
6     if n_start > 1:
7         x = np.arange(n_start)
8         h[:n_start] *= 0.5 * (1.0 - np.cos(np.pi * x / (n_start - 1)))
9     if n_end > 1:
10        x = np.arange(n_end)
11        h[-n_end:] *= 0.5 * (1.0 + np.cos(np.pi * x / (n_end - 1)))
12    return h

```

D.6 Generating the waveform

```

1 def generate_waveform(params: dict, cfg: GWConfig) -> np.ndarray:
2     """Generate a tapered h_plus BBH waveform. Returns h (np.ndarray)."""
3     hp, _ = get_td_waveform(
4         approximant=cfg.approximant,
5         mass1=params["mass1"],
6         mass2=params["mass2"],
7         delta_t=cfg.dt,
8         f_lower=cfg.f_gen,
9         distance=cfg.distance_mpc,
10        coa_phase=params["coa_phase"],
11        taper=None,
12    )
13    h = np.asarray(hp, dtype=np.float64)
14    return apply_edge_taper(h, cfg)

```

D.7 Placing the signal in the buffer

```

1 def place_in_buffer(h: np.ndarray, cfg: GWConfig,
2                     tc_in_crop: float) -> np.ndarray:
3     """Insert 'h' into a 'cfg.buffer_n'-length zero buffer.
4
5     The merger (peak |h|) is placed at crop_start_time + tc_in_crop.
6     """
7     buf = np.zeros(cfg.buffer_n, dtype=np.float64)
8
9     peak_target = int(round((cfg.crop_start_time + tc_in_crop) * cfg.
10                             sample_rate))
11     peak_h = int(np.argmax(np.abs(h)))
12     start, end = peak_target - peak_h, peak_target - peak_h + len(h)
13
14     if end > cfg.buffer_n:
15         raise ValueError("Waveform extends past the right edge of the buffer.")

```

```

15     src_start = max(0, -start)          # if start < 0 we drop the early
16         inspiral
17     dst_start = max(0, start)
18     buf[dst_start:end] = h[src_start:]
19     return buf

```

D.8 Noise spectral density and coloured noise

```

1 @lru_cache(maxsize=8)
2 def _psd_cached(n_samples: int, sample_rate: int, f_low: float):
3     delta_f = sample_rate / n_samples
4     return aLIGOZeroDetHighPower(n_samples // 2 + 1, delta_f, f_low)
5
6
7 def make_psd(n_samples: int, cfg: GWConfig):
8     """Analytic aLIGOZeroDetHighPower PSD matched to a time-series length."""
9     return _psd_cached(n_samples, cfg.sample_rate, cfg.f_low)
10
11
12 def generate_colored_noise(cfg: GWConfig, seed: int) -> np.ndarray:
13     """Colored Gaussian noise of length 'cfg.buffer_n'."""
14     psd = make_psd(cfg.buffer_n, cfg)
15     ts = noise_from_psd(length=cfg.buffer_n, delta_t=cfg.dt, psd=psd, seed=
16         seed)
17     return np.asarray(ts, dtype=np.float64)

```

D.9 Computing and rescaling the optimal SNR

```

1 def optimal_snr(signal_crop: np.ndarray, cfg: GWConfig, psd=None) -> float:
2     """Optimal matched-filter SNR of the signal inside the crop."""
3     ts = TimeSeries(signal_crop, delta_t=cfg.dt)
4     if psd is None:
5         psd = make_psd(len(signal_crop), cfg)
6     return float(sigma(ts, psd=psd,
7         low_frequency_cutoff=cfg.f_low,
8         high_frequency_cutoff=cfg.f_high_safe))
9
10
11 def scale_to_snr(signal_buffer: np.ndarray, cfg: GWConfig,
12     target_snr: float):
13     """Return 'signal_buffer * A' so that its crop has SNR = target_snr."""
14     rho0 = optimal_snr(signal_buffer[cfg.crop_slice], cfg)
15     if not np.isfinite(rho0) or rho0 <= 0:
16         raise ValueError(f"Invalid unscaled SNR: {rho0}")
17
18     A = target_snr / rho0
19     scaled_signal_buffer = A * signal_buffer
20     measured = optimal_snr(scaled_signal_buffer[cfg.crop_slice], cfg)
21
22     info = {
23         "target_snr": float(target_snr),
24         "unscaled_snr_crop": float(rho0),
25         "amplitude_scale_A": float(A),
26         "measured_snr_crop": float(measured),
27     }
28     return scaled_signal_buffer, info

```

D.10 Whitening and band-passing

```
1 def whiten_and_bandpass(buf: np.ndarray, cfg: GWConfig,
2                          fir_order_duration: float = 1.0,
3                          beta: float = 5.0) -> np.ndarray:
4     """Whiten with the analytic PSD, then highpass/lowpass with FIR filters."""
5     ts = TimeSeries(np.asarray(buf, dtype=np.float64), delta_t=cfg.dt)
6     psd = make_psd(len(buf), cfg)
7     psd = inverse_spectrum_truncation(
8         psd,
9         max_filter_len=int(cfg.max_filter_duration * cfg.sample_rate),
10        low_frequency_cutoff=cfg.f_low,
11        trunc_method="hann",
12    )
13
14    out = (ts.to_frequencyseries() / psd ** 0.5).to_timeseries()
15
16    order = int(fir_order_duration * cfg.sample_rate)
17    out = out.highpass_fir(cfg.f_low, order=order, beta=beta,
18        remove_corrupted=False)
19    out = out.lowpass_fir(cfg.f_high_safe, order=order, beta=beta,
20        remove_corrupted=False)
21
22    # Defensive: preserve original buffer length
23    arr = np.asarray(out, dtype=np.float64)
24    if len(arr) > len(buf): arr = arr[:len(buf)]
25    elif len(arr) < len(buf): arr = np.pad(arr, (0, len(buf) - len(arr)))
26    return arr
```

D.11 Building one signal or noise sample

```
1 def generate_signal_sample(cfg: GWConfig,
2                             rng: np.random.Generator,
3                             max_tries: int = 20,
4                             return_debug: bool = False):
5     """One signal+noise ML sample (label = 1) - whitened crop."""
6     for attempt in range(max_tries):
7         try:
8             p = sample_bbh_parameters(cfg, rng)
9             target_snr = float(rng.uniform(cfg.snr_min, cfg.snr_max))
10            tc = float(rng.uniform(cfg.tc_min_in_crop, cfg.
11                tc_max_in_crop))
12            noise_seed = int(rng.integers(0, 2**31 - 1))
13
14            h = generate_waveform(p, cfg)
15            signal_buffer = place_in_buffer(h, cfg, tc)
16            signal_buffer, snr_info = scale_to_snr(signal_buffer, cfg,
17                target_snr)
18            noise_buffer = generate_colored_noise(cfg, seed=noise_seed)
19
20            x_white = whiten_and_bandpass(signal_buffer + noise_buffer, cfg)
21            x_ml = x_white[cfg.crop_slice].astype(cfg.dtype)
22
23            meta = {"label": 1, "attempt": attempt, "noise_seed": noise_seed,
24                "tc_in_crop": tc, "target_snr": target_snr, **p, **
25                snr_info}
26
27            if return_debug:
```

```

25         signal_white = whiten_and_bandpass(signal_buffer, cfg)
26         noise_white = whiten_and_bandpass(noise_buffer, cfg)
27         debug = {
28             "signal_white_crop": signal_white[cfg.crop_slice].astype(
29                 cfg.dtype),
30             "noise_white_crop": noise_white[cfg.crop_slice].astype(
31                 cfg.dtype),
32             "x_white_crop": x_ml,
33         }
34         return x_ml, meta, debug
35     return x_ml, meta
36
37 except ValueError:
38     continue
39 raise RuntimeError(f"No valid signal sample after {max_tries} tries.")
40
41 def generate_noise_sample(cfg: GWConfig, rng: np.random.Generator,
42                           return_debug: bool = False):
43     """One noise-only ML sample (label = 0) - whitened crop."""
44     noise_seed = int(rng.integers(0, 2**31 - 1))
45     noise_buffer = generate_colored_noise(cfg, seed=noise_seed)
46     x_white = whiten_and_bandpass(noise_buffer, cfg)
47     x_ml = x_white[cfg.crop_slice].astype(cfg.dtype)
48
49     meta = {"label": 0, "noise_seed": noise_seed, "target_snr": 0.0}
50     if return_debug:
51         return x_ml, meta, {"noise_white_crop": x_ml, "x_white_crop": x_ml}
52     return x_ml, meta

```

D.12 Plotting helpers

```

1 def as_timeseries(crop: np.ndarray, cfg: GWConfig) -> TimeSeries:
2     """Wrap a whitened crop array as a pyCBC TimeSeries (epoch = 0)."""
3     return TimeSeries(np.asarray(crop), delta_t=cfg.dt, epoch=0.)
4
5
6 '''def plot_whitened_crop(ts: TimeSeries, label, snr=None, idx=None, ax=None)
7 :
8     if ax is None:
9         fig, ax = plt.subplots(figsize=(20, 4))
10        ax.plot(ts.sample_times, ts, label=f"signal {idx} | label={label}")
11        ax.set_xlabel("Time [s]")
12        ax.set_ylabel("Whitened strain")
13        title = f"Signal {idx} | label={label}"
14        if snr is not None:
15            title += f" | SNR={snr:.2f}"
16        ax.set_title(title)
17        ax.legend()'''
18
19 def plot_whitened_crop(noisy_ts, label, snr=None, idx=None,
20                        signal_ts=None, noise_ts=None, ax=None):
21     """Overlay pure signal, pure noise and signal+noise on one axis.
22
23     'signal_ts' / 'noise_ts' are optional (whitened pure-signal / pure-noise
24     crops). When given, all three traces are plotted with transparency so you
25     can visually check that signal + noise == the noisy data.
26     """
27     if ax is None:
28         fig, ax = plt.subplots(figsize=(20, 4))

```

```

28
29 # Plot noise first (background), then signal+noise, then pure signal on
   top
30 if noise_ts is not None:
31     ax.plot(noise_ts.sample_times, noise_ts,
32             label="noise", color="C7", alpha=0.35, lw=0.8)
33 ax.plot(noisy_ts.sample_times, noisy_ts,
34         label="signal + noise", color="C0", alpha=0.5, lw=0.9)
35 if signal_ts is not None:
36     ax.plot(signal_ts.sample_times, signal_ts,
37             label="signal", color="C3", alpha=0.9, lw=1.2)
38
39 ax.set_xlabel("Time inside crop [s]")
40 ax.set_ylabel("Whitened strain")
41 title = f"Signal {idx} | label={label}"
42 if snr is not None:
43     title += f" | SNR={snr:.2f}"
44 ax.set_title(title)
45 ax.legend(loc="upper right")

```

D.13 Generating the full dataset

```

1 if BOOL_SAVE_PURE_SIGNAL_AND_NOISE:
2     list_pure_signals = []
3     list_pure_noises = []
4     list_signals_plus_noises = []
5
6 if BOOL_TEST_TOPOLOGICAL_FEATURES == False:
7     rng = np.random.default_rng(cfg.seed)
8
9     list_ml_set_signals = []
10    labels = []
11    list_SNR_values = np.zeros(num_signals)
12    list_mass_ratio_coef = []
13
14    n_signal = 0
15    n_noise = 0
16
17    for i in tqdm(range(num_signals), desc="Generating signals"):
18        is_signal = (i % 2 == 0) # interleave: signal, noise, signal, ...
19
20        if is_signal:
21            x_ml, meta, dbg = generate_signal_sample(cfg, rng, return_debug=
22                True)
23            ts = as_timeseries(x_ml, cfg)
24            list_ml_set_signals.append(ts)
25            labels.append(1)
26            list_SNR_values[i] = meta["measured_snr_crop"]
27            list_mass_ratio_coef.append(meta["q"])
28            n_signal += 1
29
30            if BOOL_SAVE_PURE_SIGNAL_AND_NOISE:
31                list_pure_signals.append(as_timeseries(dbg["signal_white_crop"],
32                    cfg))
33                list_pure_noises.append(as_timeseries(dbg["noise_white_crop"],
34                    cfg))
35                list_signals_plus_noises.append(as_timeseries(dbg["
36                    x_white_crop"], cfg))
37            else:

```

```

34     x_ml, meta, dbg = generate_noise_sample(cfg, rng, return_debug=
35         True)
36     ts = as_timeseries(x_ml, cfg)
37     list_ml_set_signals.append(ts)
38     labels.append(0)
39     list_SNR_values[i] = 0.0
40     list_mass_ratio_coef.append(np.nan)    # no mass for noise-only
41     n_noise += 1
42
43     if BOOL_SAVE_PURE_SIGNAL_AND_NOISE:
44         # No injected signal: pure-signal crop is zeros; pure-noise
45         # crop is the data.
46         zero_crop = np.zeros(cfg.crop_n, dtype=cfg.dtype)
47         list_pure_signals.append(as_timeseries(zero_crop, cfg))
48         list_pure_noises.append(as_timeseries(dbg["noise_white_crop"],
49         cfg))
50         list_signals_plus_noises.append(as_timeseries(dbg["
51         x_white_crop"], cfg))
52
53     print(f"Generated {num_signals} samples ({n_signal} signal+noise, {
54         n_noise} noise-only)")
55     _nonzero = [s for s in list_SNR_values if s > 0]
56     if _nonzero:
57         print(f"Measured-SNR range (signal class): {min(_nonzero):.2f} - {max
58         (_nonzero):.2f}")
59
60     # Visual check of the first few samples
61     '''for i in range(min(7, num_signals)):
62         plot_whitened_crop(list_ml_set_signals[i], labels[i],
63             snr=list_SNR_values[i] if list_SNR_values[i] > 0
64             else None,
65             idx=i)
66     plt.show()'''
67     for i in range(min(7, num_signals)):
68         if BOOL_SAVE_PURE_SIGNAL_AND_NOISE:
69             plot_whitened_crop(
70                 list_signals_plus_noises[i], labels[i],
71                 snr=list_SNR_values[i] if list_SNR_values[i] > 0 else None,
72                 idx=i,
73                 signal_ts=list_pure_signals[i] if labels[i] == 1 else None,
74                 noise_ts=list_pure_noises[i],
75             )
76         else:
77             plot_whitened_crop(
78                 list_ml_set_signals[i], labels[i],
79                 snr=list_SNR_values[i] if list_SNR_values[i] > 0 else None,
80                 idx=i,
81             )
82     plt.show()

```

D.14 Topological-feature test mode

```

1  if BOOL_TEST_TOPOLOGICAL_FEATURES == True:
2      print("--- RUNNING IN TOPOLOGICAL TEST MODE ---")
3      rng = np.random.default_rng(cfg.seed)
4
5      # Fixed parameters for the test waveform
6      p = {"mass1": q_test * mtot_test / (1.0 + q_test),
7          "mass2": mtot_test / (1.0 + q_test),
8          "q": q_test, "total_mass": mtot_test, "coa_phase": 0.0}

```

```

9
10 #target_snr = float(0.5 * (cfg.snr_min + cfg.snr_max)) # mid-range
    target
11 target_snr = float(SNR_TEST_MODE)
12 tc = float(0.5 * (cfg.tc_min_in_crop + cfg.tc_max_in_crop))
13 noise_seed = int(rng.integers(0, 2**31 - 1))
14
15 h = generate_waveform(p, cfg)
16 signal_buffer = place_in_buffer(h, cfg, tc)
17 signal_buffer, snr_info = scale_to_snr(signal_buffer, cfg, target_snr)
18 noise_buffer = generate_colored_noise(cfg, seed=noise_seed)
19
20 signal_white = whiten_and_bandpass(signal_buffer, cfg)[cfg.crop_slice].
    astype(cfg.dtype)
21 noise_white = whiten_and_bandpass(noise_buffer, cfg)[cfg.crop_slice].
    astype(cfg.dtype)
22 x_white = whiten_and_bandpass(signal_buffer + noise_buffer, cfg)[cfg
    .crop_slice].astype(cfg.dtype)
23
24 SNR = snr_info["measured_snr_crop"]
25
26 # [pure signal, pure noise, signal + noise]
27 list_ml_set_signals = [as_timeseries(signal_white, cfg),
28                        as_timeseries(noise_white, cfg),
29                        as_timeseries(x_white, cfg)]
30 labels = [1, 0, 1]
31 list_SNR_values = [0.0, 0.0, SNR]
32 list_mass_ratio_coef = [q_test, np.nan, q_test]
33 num_signals = 3
34 realistic_sample_idx = 2 # the signal+noise sample is the "realistic"
    one
35
36 fig, ax = plt.subplots(figsize=(20, 4))
37 ax.plot(list_ml_set_signals[2].sample_times, list_ml_set_signals[2],
38        label="signal + noise (whitened)", alpha=0.7)
39 ax.plot(list_ml_set_signals[0].sample_times, list_ml_set_signals[0],
40        label="pure signal (whitened)")
41 ax.set_xlabel("Time inside crop [s]"); ax.set_ylabel("Whitened strain")
42 ax.set_title(f"Topological test set | SNR = {SNR:.2f}")
43 ax.legend(); plt.show()
44
45 if BOOL_SAVE_PURE_SIGNAL_AND_NOISE:
46     list_pure_signals = [as_timeseries(signal_white, cfg)] * 3
47     list_pure_noises = [as_timeseries(noise_white, cfg)] * 3
48     list_signals_plus_noises = [as_timeseries(x_white, cfg)] * 3

```

D.15 Saving the dataset

```

1 dataset_to_save = {
2     "list_ml_set_signals": list_ml_set_signals,
3     "labels": labels,
4     "list_SNR_values": list_SNR_values,
5     "PSD_CONST_VALUE": PSD_CONST_VALUE, # NaN: coloured aLIGO PSD
    now
6     "list_mass_ratio_coef": list_mass_ratio_coef,
7     "SAMPLE_RATE": cfg.sample_rate,
8     "M_total": M_total, # representative only
9     "DURATION_TOTAL": cfg.buffer_duration, # full buffer length [s]
10    "DURATION_SIGNAL": cfg.crop_duration, # whitened crop length [s]
    ] (what is saved)

```

```

11     "EPSILON":                EPSILON,                # NaN: unused by new
12         scaling
13     "MODEL":                  cfg.approximant,
14     "LOW_FREQUENCY_CUTOFF":   cfg.f_low,
15     "num_snr_values":         num_snr_values,          # 0 => sampled, not a
16         grid
17     "num_mass_ratio_values":  num_mass_ratio_values,   # 0 => sampled, not a
18         grid
19     # ---- extra provenance (extractor ignores these) ----
20     "config":                 vars(cfg),
21     "CROP_N":                 cfg.crop_n,
22     "PIPELINE":               "01_Generate_Waveform_correct (whitened crop)",
23 }
24
25 if BOOL_SAVE_PURE_SIGNAL_AND_NOISE:
26     dataset_to_save["list_pure_signals"] = list_pure_signals
27     dataset_to_save["list_pure_noises"] = list_pure_noises
28     dataset_to_save["list_signals_plus_noises"] = list_signals_plus_noises
29
30 os.makedirs(folder, exist_ok=True)
31
32 # Filename mirrors the old naming scheme so the extractor's 'filename' is
33 familiar
34 _nonzero = [x for x in list_SNR_values if x > 0]
35 snr_min_f = min(_nonzero) if _nonzero else 0.0
36 snr_max_f = max(list_SNR_values) if len(list_SNR_values) else 0.0
37
38 if BOOL_TEST_TOPOLOGICAL_FEATURES == False:
39     filename = (f"{folder}/gw_data_{num_signals}_signals_"
40                f"SNR_{snr_min_f:.2f}_{snr_max_f:.2f}_"
41                f"num_snr_values_{num_snr_values}_"
42                f"MassTotal_{M_total}_numMassRatios_{num_mass_ratio_values}."
43                f"pkl")
44 else:
45     dataset_to_save["BOOL_TEST_TOPOLOGICAL_FEATURES"] =
46         BOOL_TEST_TOPOLOGICAL_FEATURES
47     dataset_to_save["realistic_sample_idx"] = realistic_sample_idx
48     print(f"BOOL_TEST_TOPOLOGICAL_FEATURES = {BOOL_TEST_TOPOLOGICAL_FEATURES}"
49           ")
50     filename = (f"{folder}/gw_data_TEST_TOPOLOGICAL_FEATURES_"
51                f"SNR_{snr_max_f:.2f}_MassTotal_{M_total}_samplingRate_{cfg."
52                f"sample_rate}.pkl")
53
54 with open(filename, "wb") as f:
55     pickle.dump(dataset_to_save, f)
56
57 print(f"Dataset and parameters saved successfully to:\n {filename}")

```

Bibliography

- [1] A. Einstein. Die grundlage der allgemeinen relativitätstheorie. *Annalen der Physik*, 354(7):769–822, January 1916. ISSN 1521-3889. doi: 10.1002/andp.19163540702. URL <http://dx.doi.org/10.1002/andp.19163540702>.
- [2] A. Einstein. Über gravitationswellen. Sitzungsber. K. Preuss. Akad. Wiss., 1918.
- [3] J. Cervantes-Cota, S. Galindo-Uribarri, and G. Smoot. A brief history of gravitational waves. *Universe*, 2(3):22, 2016. doi: 10.3390/universe2030022.
- [4] R. A. Hulse and J. H. Taylor. Discovery of a pulsar in a binary system. *Astrophys. J. Letters*, 195: L51–L53, 1975. doi: 10.1086/181708.
- [5] J. H. Taylor, L. A. Fowler, and P. M. McCulloch. Measurements of general relativistic effects in the binary pulsar PSR1913+16. *Nature*, 277(5696):437–440, 1979. doi: 10.1038/277437a0.
- [6] B. P. Abbott et al. Observation of gravitational waves from a binary black hole merger. *Phys. Rev. Lett.*, 116:061102, 2016. doi: 10.1103/PhysRevLett.116.061102.
- [7] B. P. Abbott et al. Properties of the binary black hole merger gw150914. *Physical Review Letters*, 116(24), 2016. doi: 10.1103/physrevlett.116.241102.
- [8] LIGO scientific collaboration. Observing capabilities, 2022. URL <https://emfollow.docs.ligo.org/userguide/capabilities.html>.
- [9] Punturo, M. et al. The Einstein Telescope: a third-generation gravitational wave observatory. *Classical and Quantum Gravity*, 27(19):194002, 2010. doi: 10.1088/0264-9381/27/19/194002.
- [10] Reitze, D. et al. Cosmic explorer: The U.S. contribution to gravitational-wave astronomy beyond LIGO, 2019. URL <https://arxiv.org/abs/1907.04833>.
- [11] Amaro-Seoane, P. et al. Laser interferometer space antenna, 2017. URL <https://doi.org/10.48550/arXiv.1702.00786>.
- [12] M. Maggiore. *Gravitational Waves: Volume 1: Theory and Experiments*. OUP Oxford, 2008. ISBN 9780198570745. doi: 10.1093/acprof:oso/9780198570745.001.0001.
- [13] Abbott et al. (LIGO Scientific Collaboration, Virgo Collaboration, and KAGRA Collaboration). An optically targeted search for gravitational waves emitted by core-collapse supernovae during the third observing run of Advanced LIGO and Advanced Virgo. *The Astrophysical Journal*, 955:155, 2023. doi: 10.3847/1538-4357/aced98.
- [14] Michael Pürrer and Carl-Johan Haster. Ready for what lies ahead? – gravitational waveform accuracy requirements for future ground based detectors, 2019. URL <https://arxiv.org/abs/1912.10055>.
- [15] Gunnar Carlsson. Topology and data. *Bulletin of the American Mathematical Society*, 46(2):255–308, 2009.
- [16] Žiga Virk. *Introduction to Persistent Homology*. Založba UL FRI, Ljubljana, 2022.

- [17] Kelin Xia and Guo-Wei Wei. Persistent homology analysis of protein structure, flexibility and folding. *International Journal for Numerical Methods in Biomedical Engineering*, 30:814–844, 2014. doi: 10.1002/cnm.2655.
- [18] Luigi Caputi, Anna Pidnebesna, and Jaroslav Hlinka. Promises and pitfalls of topological data analysis for brain connectivity analysis. *NeuroImage*, 238:118245, 2021. doi: 10.1016/j.neuroimage.2021.118245.
- [19] Mickaël Buchet, Yasuaki Hiraoka, and Ippei Obayashi. Persistent homology and materials informatics. In Isao Tanaka, editor, *Nanoinformatics*, pages 75–95. Springer, Singapore, 2018. doi: 10.1007/978-981-10-7617-6_5.
- [20] Matteo Biagetti, Alex Cole, and Gary Shiu. The persistence of large scale structures i: primordial non-gaussianity. *Journal of Cosmology and Astroparticle Physics*, 04:061, 2021. doi: 10.1088/1475-7516/2021/04/061.
- [21] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, New York, NY, USA, 2014. ISBN 978-1-107-05713-5.
- [22] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer, 2001.
- [23] Daniel George and E.A. Huerta. Deep learning for real-time gravitational wave detection and parameter estimation: Results with advanced ligo data. *Physics Letters B*, 778:64–70, March 2018. ISSN 0370-2693. doi: 10.1016/j.physletb.2017.12.053. URL <http://dx.doi.org/10.1016/j.physletb.2017.12.053>.
- [24] Christopher Bresten and Jae-Hun Jung. Detection of gravitational waves using topological data analysis and convolutional neural network: An improved approach, 2019. URL <https://arxiv.org/abs/1910.08245>.
- [25] Hunter Gabbard, Michael Williams, Fergus Hayes, and Chris Messenger. Matching matched filtering with deep networks for gravitational-wave astronomy, 2017. URL <https://arxiv.org/abs/1712.06041>.
- [26] LIGO Scientific Collaboration and Virgo Collaboration and KAGRA Collaboration. GWTC-4.0: An introduction to version 4.0 of the gravitational-wave transient catalog. *arXiv preprint*, 2025.
- [27] Alex Nitz, Ian Harry, Duncan Brown, Christopher M. Biwer, and Josh Willis. gwastro/pycbc: v2.4.0 release of pycbc, 2023. URL <https://doi.org/10.5281/zenodo.10013996>.
- [28] Timothy D. Gebhard, Niki Kilbertus, Ian Harry, and Bernhard Schölkopf. Convolutional neural networks: A magic bullet for gravitational-wave detection?, 2019. URL <https://arxiv.org/abs/1904.08693>.
- [29] giotto-ai. giotto-tda: A topological data analysis toolkit for machine learning and data exploration. <https://github.com/giotto-ai/giotto-tda>, 2021. Source code repository, including the example notebook this section follows.
- [30] Raphaël Tinarrage. Topological data analysis with persistent homology. EMap Summer Course lecture notes, Fundação Getulio Vargas, 2021. Last update 10 February 2021.
- [31] Herbert Edelsbrunner and John Harer. *Computational Topology: An Introduction*. American Mathematical Society, Providence, RI, 2010.
- [32] Jeff Murugan and Duncan Robertson. An introduction to topological data analysis for physicists: From lgm to frbs. *arXiv preprint arXiv:1904.11044*, 2019.
- [33] Magnus Bakke Botnan. Topological data analysis. <https://www.few.vu.nl/~botnan/>, 2024. Mastermath lecture notes, version of 17 June 2024.

- [34] Guillaume Tauzin, Umberto Lupo, Lewis Tunstall, Julian Burella Pérez, Matteo Caorsi, Anibal Medina-Mardones, Alberto Dassatti, and Kathryn Hess. giotto-tda: A topological data analysis toolkit for machine learning and data exploration, 2020. URL <https://arxiv.org/abs/2004.02551>.
- [35] Andrew M. Fraser and Harry L. Swinney. Independent coordinates for strange attractors from mutual information. *Physical Review A*, 33(2):1134–1140, 1986. doi: 10.1103/PhysRevA.33.1134.
- [36] Felipe L. Gewers, Gustavo R. Ferreira, Henrique F. De Arruda, Filipi N. Silva, Cesar H. Comin, Diego R. Amancio, and Luciano Da F. Costa. Principal component analysis: A natural approach to data exploration. *ACM Comput. Surv.*, 54(4), May 2021. ISSN 0360-0300. doi: 10.1145/3447755. URL <https://doi.org/10.1145/3447755>.
- [37] S. Shambhavi. Applications of topology to data analysis. Master’s thesis, Indian Institute of Science Education and Research Pune, 2021. BS-MS dual degree thesis, supervised by Priyavrat Deshpande.
- [38] Harish Chintakunta, Thanos Gentimis, Rocio Gonzalez-Diaz, Maria-Jose Jimenez, and Hamid Krim. An entropy-based persistence barcode. *Pattern Recognition*, 48(2):391–401, 2015. ISSN 0031-3203. doi: <https://doi.org/10.1016/j.patcog.2014.06.023>. URL <https://www.sciencedirect.com/science/article/pii/S0031320314002453>.
- [39] Henry Adams, Tegan Emerson, Michael Kirby, Rachel Neville, Chris Peterson, Patrick Shipman, Sofya Chepushtanova, Eric Hanson, Francis Motta, and Lori Ziegelmeier. Persistence images: A stable vector representation of persistent homology. *Journal of Machine Learning Research*, 18(8): 1–35, 2017. URL <http://jmlr.org/papers/v18/16-337.html>.
- [40] The GUDHI Project. *GUDHI User and Reference Manual*. GUDHI Editorial Board, 3.13.0 edition, 2026. URL <https://gudhi.inria.fr/doc/3.13.0/>.
- [41] Jean-Daniel Boissonnat and Siddharth Pritam. Edge Collapse and Persistence of Flag Complexes. In Sergio Cabello and Danny Z. Chen, editors, *36th International Symposium on Computational Geometry (SoCG 2020)*, volume 164 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 19:1–19:15, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-143-6. doi: 10.4230/LIPIcs.SoCG.2020.19. URL <https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.SoCG.2020.19>.
- [42] S. M. Carroll. *Spacetime and Geometry: An Introduction to General Relativity*. Cambridge University Press, 2019. doi: 10.1017/9781108770385.
- [43] E. Tiesinga, P. J. Mohr, D. B. Newell, and B. N. Taylor. CODATA recommended values of the fundamental physical constants: 2018. *Rev. Mod. Phys.*, 93(2):025010, 2021. doi: 10.1103/RevModPhys.93.025010.
- [44] M. P. Hobson, G. P. Efstathiou, and A. N. Lasenby. *General Relativity: An Introduction for Physicists*. Cambridge University Press, 2006. doi: 10.1017/CBO9780511790904.
- [45] S. Weinberg. *Gravitation and cosmology: principles and applications of the general theory of relativity*. Wiley, New York, 1972. ISBN 9780471925675.
- [46] C. W. Misner, K. S. Thorne, and J. A. Wheeler. *Gravitation*. W. H. Freeman, San Francisco, 1973. ISBN 0716703343.
- [47] P. R. Saulson. Josh goldberg and the physical reality of gravitational waves. *General Relativity and Gravitation*, 43(12):3289–3299, 2011.
- [48] B. P. Abbott et al. GW170814: A three-detector observation of gravitational waves from a binary black hole coalescence. *Phys. Rev. Lett.*, 119:141101, 2017. doi: 10.1103/PhysRevLett.119.141101.
- [49] B. P. Abbott et al. GW170817: Observation of gravitational waves from a binary neutron star inspiral. *Phys. Rev. Lett.*, 119:161101, 2017. doi: 10.1103/PhysRevLett.119.161101.

- [50] Maggiore, M. et al. Science case for the Einstein telescope. *J. Cosmol. Astropart. Phys.*, 2020(03):050, 2020. doi: 10.1088/1475-7516/2020/03/050.
- [51] Branchesi, M. et al. Science with the Einstein Telescope: a comparison of different designs. *J. Cosmol. Astropart. Phys.*, 2023(07):068, 2023. doi: 10.1088/1475-7516/2023/07/068.
- [52] Particle Data Group et al. Review of particle physics. *Progress of Theoretical and Experimental Physics*, 2022(8):083C01, 2022. doi: 10.1093/ptep/ptac097.
- [53] Karl Wette. Searches for continuous gravitational waves from neutron stars: A twenty-year retrospective. *Astroparticle Physics*, 153:102880, November 2023. ISSN 0927-6505. doi: 10.1016/j.astropartphys.2023.102880. URL <http://dx.doi.org/10.1016/j.astropartphys.2023.102880>.
- [54] M. Maggiore. *Gravitational Waves: Volume 2: Astrophysics and Cosmology*. Oxford University Press, 2018. ISBN 9780198570899. doi: 10.1093/oso/9780198570899.001.0001.
- [55] *La Rivista del Nuovo Cimento*, 39(9):399–495, 2016. ISSN 0393697X. doi: 10.1393/ncr/i2016-10127-1.
- [56] The NANOGrav Collaboration. The NANOGrav 15 yr data set: Evidence for a gravitational-wave background. *Astrophys. J. Letters*, 951(1):L8, 2023. doi: 10.3847/2041-8213/acdac6.
- [57] Marc Andrés Carcasona. *Searches for primordial black holes using gravitational waves*. PhD thesis, Barcelona, Autònoma U., Spain, 2025.
- [58] M. Drago, V. Gayathri, S. Klimenko, C. Lazzaro, E. Milotti, G. Mitselmakher, V. Necula, B. O’Brian, G. A. Prodi, F. Salemi, M. Szczepanczyk, S. Tiwari, V. Tiwari, G. Vedovato, and I. Yakushin. Coherent WaveBurst, a pipeline for unmodeled gravitational-wave data analysis. *SoftwareX*, 14: 100678, 2021. doi: 10.1016/j.softx.2021.100678.
- [59] Bruce Allen. A χ^2 time-frequency discriminator for gravitational wave detection. *Physical Review D*, 71:062001, 2005. doi: 10.1103/PhysRevD.71.062001.
- [60] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, 2020. doi: 10.1038/s41586-020-2649-2.
- [61] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C. J. Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. Scipy 1.0: Fundamental algorithms for scientific computing in python. *Nature Methods*, 17(3):261–272, 2020. doi: 10.1038/s41592-019-0686-2.
- [62] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830, 2011.
- [63] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, pages 8024–8035, 2019.

- [64] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3): 90–95, 2007. doi: 10.1109/MCSE.2007.55.
- [65] Plotly Technologies Inc. Collaborative data science, 2015. URL <https://plot.ly>.
- [66] Plamen G. Krastev. Real-time detection of gravitational waves from binary neutron stars using artificial neural networks, 2020. URL <https://arxiv.org/abs/1908.03151>.
- [67] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.
- [68] Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech and time series. In *The Handbook of Brain Theory and Neural Networks*, pages 255–258. The MIT Press, 1995.
- [69] LIGO Scientific Collaboration and Virgo Collaboration. Results of the deepest all-sky survey for continuous gravitational waves on ligo s6 data running on the einstein@home volunteer distributed computing project. *Physical Review D*, 94(10), November 2016. ISSN 2470-0029. doi: 10.1103/physrevd.94.102002. URL <http://dx.doi.org/10.1103/PhysRevD.94.102002>.
- [70] Peter S Shawhan. Gravitational-wave astronomy: observational results and their impact. *Classical and Quantum Gravity*, 27(8):084017, April 2010. ISSN 1361-6382. doi: 10.1088/0264-9381/27/8/084017. URL <http://dx.doi.org/10.1088/0264-9381/27/8/084017>.
- [71] Paulo Saraiva. On shannon entropy and its applications. *Kuwait Journal of Science*, 50(3):194–199, 2023. ISSN 2307-4108. doi: <https://doi.org/10.1016/j.kjs.2023.05.004>. URL <https://www.sciencedirect.com/science/article/pii/S2307410823000433>.
- [72] Bruce Allen, Warren G. Anderson, Patrick R. Brady, Duncan A. Brown, and Jolien D. E. Creighton. FINDCHIRP: An algorithm for detection of gravitational waves from inspiraling compact binaries. *Physical Review D*, 85:122006, 2012. doi: 10.1103/PhysRevD.85.122006.