Geometric Multimedia Time Series

by

Christopher J Tralie

Department of Electrical and Computer Engineering
Duke University

Ph.D. Dissertation
2019

Abstract

This thesis provides a new take on problems in multimedia times series analysis by using a shape-based perspective to quantify patterns in time, which is complementary to more traditional analysis-based time series techniques. Inspired by the dynamical systems community, we turn time series into shapes via sliding window embeddings, which we refer to as "time-ordered point clouds" (TOPCs). This framework has traditionally been used on a single 1D observation function for deterministic systems, but we generalize the sliding window technique so that it not only applies to multivariate data (e.g. videos), but that it also applies to data which is not stationary (e.g. music).

The geometry of our time-ordered point clouds can be quite informative. For periodic signals, the point clouds fill out topological loops, which, depending on harmonic content, reside on various high dimensional tori. For quasiperiodic signals, the point clouds are dense on a torus. We use modern tools from topological data analysis (TDA) to quantify degrees of periodicity and quasiperiodicity by looking at these shapes, and we show that this can be used to detect anomalies in videos of vibrating vocal folds. In the case of videos, this has the advantage of substantially reducing the amount of preprocessing, as no motion tracking is needed, and the technique operates on raw pixels. This is also one of the first known uses of persistent H_2 in a high dimensional setting.

Periodic processes represent only a sliver of possible dynamics, and we also show that sequences of arbitrary normalized sliding window point clouds are approximately isometric between "cover songs," or different versions of the same song, possibly with radically different spectral content. Surprisingly, in this application, an incredibly simple geometric descriptor based on self-similarity matrices performs the best, and it also enables us to use MFCC features for this task, which was previously thought not to be possible due to significant timbral differences that can exist between versions. When combined with traditional pitch-based features using similarity metric fusion, we obtain state of the art results on automatic cover song identification.

In addition to being used as a geometric descriptor, self-similarity matrices provide a unifying description of phenomena in time-ordered point clouds throughout our work, and we use them to illustrate properties such as recurrence, mirror symmetry in time, and harmonics in periodic processes. They also provide the base representation for designing *isometry blind* time warping algorithms, which we use to synchronize time-ordered point clouds that are shifted versions of each other in space without ever having to do a spatial alignment. In particular, we devise an algorithm that lower bounds the 1-stress between two time-ordered point clouds, which is related to the Gromov-Hausdorff distance.

Overall, we show a proof-of-concept and promise of the nascent field of *geometric signal processing*, which is worthy of further study in applications of music structure, multimodal data analysis, and video analysis.

To my parents, John Tralie and Mary Poteau-Tralie, my brother James Tralie, and my partner Celia Litovsky for providing strong support and love during an incredibly unpredictable Ph.D. experience

Contents

List of Abbreviations

CAFMap Community-accepted feature map. vii, 9

CQT Constant-Q Transform. vii, 82

CSM Cross-Similarity Matrix: A matrix of distances between two time-ordered point clouds. vii, 11–13, 15, 65, 68, 70, 90, 91, 93, 96–98, 107

CSTWM Cross-Similarity Time-Warp Matrix: A cross-similarity matrix which stores cost of time warping all pairs of rows between self-similarity matrices. vii, 151, 153, 155, 156

DTW Dynamic Time Warping: An algorithm for aligning time-ordered point clouds in some metric space. vii, 63–70, 92, 93, 143, 144, 146, 149, 151, 152, 154–156

FLDTW First Last Dynamic Time Warping: An new distance for aligning time-ordered point clouds that have been transformed or are in different spaces. Weaker and easier to solve than IBDTW. vii, 149

GPU Graphics processing unit. vii, 155

IBDTW Isometry Blind Dynamic Time Warping: An new distance for aligning time-ordered point clouds that have been transformed or are in different spaces. A restricted version of the Gromov-Hausdorff Distance. vii, 149, 151–153, 156, 164, 166, 167

MFCC Mel-Frequency Cepstral Coefficients. vii, 55, 72, 75, 82–86, 89, 91, 93, 97, 98, 102–105, 107, 108

PCA Principal Component Analysis (also known as the Karhunen-Loève Transform or Empirical Orthogonal Functions). vii, 5, 21, 33, 40, 43, 44, 46, 79, 86, 112, 116, 118, 119, 125, 126, 137–141

SSM Self-Similarity Matrix: A time-ordered symmetric matrix of all pairwise distances between points in a time-ordered point cloud. vii, 11, 12, 45, 54, 57, 83–86, 89–91, 96–98, 102–104, 107, 108, 113, 121, 124, 132, 143, 144, 146, 151, 156, 157, 159–164, 166, 167

STFT Short-Time Fourier Transform. vii, 9, 70–72, 74

TOPC Time-Ordered Point Cloud. vii, 4–7, 10, 11, 13, 15–19, 21, 26, 30, 63–66, 68, 69, 79, 130, 143, 148, 149, 151–153, 159, 164, 166, 167

List of Symbols

$B_r(x)$ A ball of radius r in some metric space. vii, 31

β_k The k^{th} Betti number. vii, 29, 30, 35, 36

∂ Boundary operator on a chain complex. vii, 26–30, 34, 35, 47

δ Co-boundary operator on a chain complex. vii

$\mathcal{G}$ A community-accepted feature map. vii, 9

$\mathcal{C}$ Correspondence. vii, 20, 148

Π Set of All Correspondences. vii, 148

$\mathcal{F}$ A Fourier Transform (either DFT or continuous Fourier Transform, based on context). vii, 9, 120

$\mathbb{F}$ An algebraic field. vii, 26

$||X||_F$ The Frobenius norm of a matrix (the square root of the sum of the squares of the entries). vii

κ Unsigned Curvature. vii, 169, 171

$(\mathcal{M}, d)$ A metric space with distances d. vii, 2, 6, 7, 11, 12, 31, 32, 69

$\mathcal{M}$ A Riemannian manifold. vii, 39, 40

$\mathbb{C}$ The complex numbers. vii

$\mathbb{Z}$ The integers. vii

$\mathbb{R}$ The real numbers. vii

ψ A state transition function in a dynamical system. vii, 39–41, 43

$\mathcal{S}$ p-Stress. vii, 148, 149

τ Unsigned Torsion or Kendall-Tau score (clear from context). vii, 169

$\mathcal{W}$ Warping Path. vii, 63, 149, 151, 152, 154

Ω Set of All Warping Paths. vii, 63, 149

Acknowledgements

To my primary mentors, special thanks Guillermo Sapiro for rescuing me when my first adviser left Duke, for being incredibly flexible with my choices of research topics, and for cultivating such a rich and vibrant community of international researchers, from whom I benefited tremendously. Thanks to John Harer for his kind, collaborative spirit, for taking me under his wing with patience and for putting me on the path to being a mathematician in addition to an engineer (which has always been a dream of mine), and for investing so much care into my personal growth and professional development. Thanks to Paul Bendich for being such an excellent close collaborator, for showing me how to mentor by example, and for normalizing a range of difficult and wacky experiences that have arisen for me as a budding academic (not to mention opening my eyes to some amazing Szechuan places in the Triangle area). Thanks to Jose Perea for so much help and guidance during the latter part of my grad career, for putting me up at the Wild Goose Inn(!) at Michigan State, for inviting me to Munich, for having such incredible patience for the gaps in my math knowledge, and for being a great role model of an early career research academic with such a positive attitude and attention to refinement of ideas (not to mention an enormous inspiration for this dissertation with his Sw1Pers and time series work). And thanks to Loren Nolte for being on every single one of my committees before and after I switched areas, and for teaching some of my favorite classes in ECE (I'm sorry I never got to T.A. for you!). Finally, thank you to Martha Absher for accepting me to the Duke Engineering REU 8 years ago, and for valuable advice on my NSF Graduate Fellowship application. It's largely due to you that I started on this path!

During my time at Duke I was in three different research groups all with some pretty incredible people. From the Sapiro group, shout outs in particular to Jordan Hashemi, George Chang, Alasdair Newson, and Mauricio Delbracio for being great engineering collaborators in one way or another. From the Harer group, thanks to Ellen Gasparovic for work on geometric models, thanks to Justin Curry for some recent great collaborations on topological distances that I hope can continue for a long time, thanks to Rann Bar-On for being a crazy coder and a generally hilarious presence, and thanks to Francis Motta for being a model topology teacher and possibly the nicest person I've ever met. And from my RFID lab back in the day, thanks to Stewart Thomas and Josh Ensworth for being awesome friends and colleagues, with whom I've sadly lost touch but whose company I really valued early on (and

who helped keep me sane). Finally, even though he wasn't officially in any of my groups, thanks to Dmitry Vagner for being a great sounding board for ideas about research, teaching, and mental health (and also a rare shared appreciation of Lil B).

On the research environment side and grant side, I must give a hearty thanks to Information Initiative at Duke (IID), especially Paul Bendich, Robert Calderbank, and Kathy Peterson, for providing an excellent place to work, collaborate, and see lots of talks on the third floor of Gross Hall. This was orders of magnitude better than the lab closet I worked in my first two years. Thanks to the staff in the ECE department, and in particular to Amy Kostrewa for her hard work and for helping me through some difficult transitions, to helping to secure sixth year funding and travel funding, for tolerating a lot of mistakes I made, and for maintaining such a great balance between personal and professional while caring for all grad students in the department. Also thanks to an NSF Graduate fellowship and NSF Research Training Grant "Structure in Complex Data," NSF-DMS 1045133. I would have been forced to drop out of the Ph.D. or transfer to another school without that safety net! I hope our government expands these types of research safety nets. Also, a sincere thanks to Mythlogic for making a crazy custom laptop which was my "lab on the run," which was funded in part by Data Expeditions at Duke. And on a related note, thanks to the makers of Doom 2016 for providing a clutch avenue for me to blow off steam during the latter part of my dissertation writing.

On the teaching side thanks to Hugh Crumley for his mentorship in the College Certificate in Teaching (CCT) and for putting me in touch with so many other people at Duke who care about teaching. Thanks also to the Bass Family Instructional Teaching Fellowship for giving me the opportunity to design my own course in 3D geometry from scratch so early on in my career, and for the Duke CS and Math DGSs for being so accommodating with my course design. Special thanks to Steven Espinosa for being so open and honest about his own teaching experiences and for voluntarily checking in on me before every single lecture I gave that semester, providing tons of valuable feedback. Thank you also to all of my students in that class and all of the students I mentored in independent studies at Duke and NC Science And Math. These were some of the most fun and valuable experiences during my Ph.D., and I can't even quantify how much I learned from you (and continue to as you reach back out to me and share things).

Special thanks are also due to my spiritual advisers. I thank Sumi Kim for being an excellent, open, and caring Buddhist Chaplain, who was gives so much of herself to the community and to individuals and who was positively a joy to work with during my time as BMCD president. I also thank Holly Rogers for providing incredible mindful meditation seminars and invaluable individual help and firm guidance on holistic approaches to mental health that radically changed my world view and helped to de-stigmatize a lot of things I had experienced during my life. Finally, I thank Dr. Mazella Fuller for seeing me through the ups and downs of my grad school experience over the years and teaching me how to view the nuances and complexities of all of my experiences and to see their value in helping me to be a better mentor, partner,

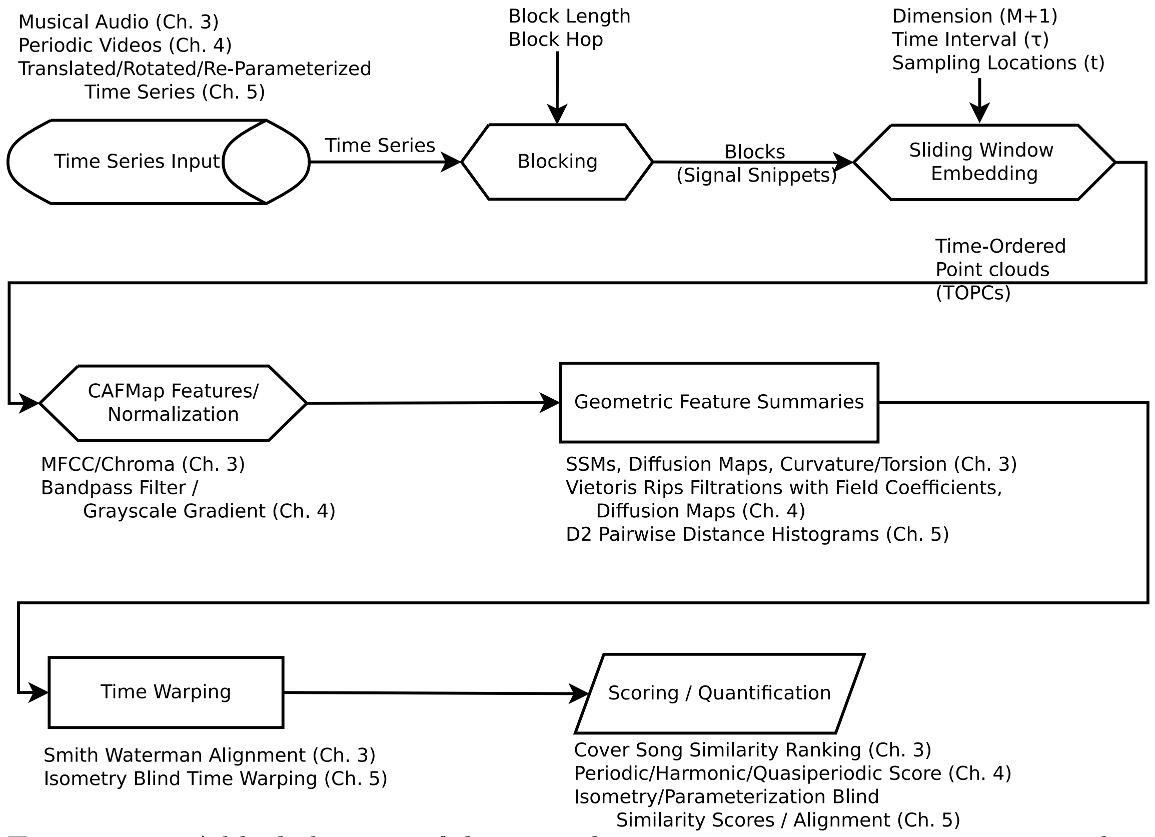
and member of my community.

I also owe some thanks to Prince Rogers Nelson, who was a huge musical inspiration and anchor during undergrad and grad school, and who I sadly found out passed away right in the middle of an NSF workshop last spring. On that note, thanks also to Terrace F. Club (FOOD = LOVE) at Princeton for taking me back, feeding me, and providing awesome shows at random intervals during grad school, including the single best concert of my life by Phil Lesh. And thanks to Harry Davidson from the Duke Symphony Orchestra for being the best conductor I ever had.

Finally, I owe so much gratitude to my family and closest friends who showed me unconditional love throughout my time at Duke. When the going got tough, my parents made sure I always had a safe place to land back in Fort Washington, PA, and we all continued to grow together during these six years. By the same token, thank you to my aunts and uncles (all 14 pairs of them) and cousins (all 32 of them) for keeping up with me after I had to move away, particularly to Aunt Patty, Uncle Bill, Aunt Leslie, Uncle Mark, Aunt Lynne, Mike/Jen, Alison/Andrea, and Pat/Tim. I'm looking forward to being closer to everyone again. Thanks also to Dan Creamer, a teacher at George Washington High School in Philadelphia, who is an incredible public servant and a lifelong friend who cares deeply about my personal development and who's kept me honest as a new teacher. Thanks to my rad group of college friends (esp. Carlton, Johannes, Anna, Jason, Sully, Chi, Vinayak, Chris K) who kept on lovin me even after I became a workaholic in grad school. Thanks to Craig LaBoda for being a partner in crime in ECE and a mosh pit and crazy movie buddy throughout grad school. Thanks to Jordan "Jordon Bensoff" Besnoff for being the best d00d and roommate I could have asked for the past 5 years, and not only for tolerating my crazy antics but, much to his chagrin, for actually starting to imitate some of them! (Not to mention some pr0 moments at the Durham sk8park). Thanks to Javier Zapata for being a hidden gem from the business school put up with me for a year as a roommate after finding me on Duke list (and who provided much hilarity at much needed times). Finally, thank you to my partner Celia Litovsky for being such a sweetie / ride or die chick and for being the most supportive and nurturing partner anyone could ask for. And of course, thanks to the amazing bearded dragon Titan Besnoff and the best feline I've ever met, Layla Ophelia Litovsky (LOL).

1

Introduction



Musical Audio (Ch. 3)
Periodic Videos (Ch. 4)
Translated/Rotated/Re-Parameterized Time Series (Ch. 5)
Time Series Input
Time Series
Block Length
Block Hop
Blocking
Blocks (Signal Snippets)
Dimension (M+1)
Time Interval (τ)
Sampling Locations (t)
Sliding Window Embedding
Time-Ordered Point clouds (TOPCs)
CAFMap Features/ Normalization
MFCC/Chroma (Ch. 3)
Bandpass Filter / Grayscale Gradient (Ch. 4)
Geometric Feature Summaries
SSMs, Diffusion Maps, Curvature/Torsion (Ch. 3)
Vietoris Rips Filtrations with Field Coefficients, Diffusion Maps (Ch. 4)
D2 Pairwise Distance Histograms (Ch. 5)
Time Warping
Smith Waterman Alignment (Ch. 3)
Isometry Blind Time Warping (Ch. 5)
Scoring / Quantification
Cover Song Similarity Ranking (Ch. 3)
Periodic/Harmonic/Quasiperiodic Score (Ch. 4)
Isometry/Parameterization Blind Similarity Scores / Alignment (Ch. 5)

FIGURE 1.1: A block diagram of the general geometric time series processing pipeline that unifies all of the work in this thesis. Each stage is annotated with some of the specifics from each chapter where applicable.

This work is the marriage between time series problems in engineering and applied signal processing and concepts in metric geometry and topology. While one's first instinct is often to tackle time series problems with analysis and traditional statistical modeling, we show some surprising advantages to adding geometric language to the signal processing pipeline. This allows us to make contributions in diverse areas of multimedia data analytics. In particular, in music information retrieval (Chapter 3), we show that adding information about the "shape" of a time series of MFCC features (a common set of audio features covered in the background in Section 2.5) allows them to be used for cover song identification, where they were previously thought not to capture any relevant information. We demonstrate that combining these new shape-based MFCC features with more traditional pitch-based features significantly improves cover song clique finding accuracy. In the areas of video processing and health-related video monitoring, we show that the "shape" of chunks of videos can help to rank periodicity of the videos or to determine the presence of quasiperiodic phenomena correlated to biological anomalies in high speed videos of resonating vocal cords (Chapter 4). We highlight the ways in which our work complements off-the-shelf Fourier analysis techniques. Finally, in multimodal data processing (e.g. video to audio), geometry and topology give a way of expressing data that is closer to being domain independent, and we demonstrate this by showing how to use geometric and topological ideas to do time warping without aligning shapes which are rotated/translated/flipped versions of each other (Chapter 5).

In all of the above applications, there are common tools which we will introduce in this chapter, as indicated in the general pipeline in Figure 1.1, and we will summarize them in this chapter. But first, we start with the most general possible definition of a time series:

Definition 1. *A* time series X *is a metric curve parameterized by the real numbers* $\mathbb{R}^+$*. It is endowed with a metric space* $(\mathcal{M}, d)$ *for measuring distances between the objects. A* N-length sampled time series *is a subset* $X_N \subset X$ *indexed by a subset of the natural numbers, often written as* $X_1, X_2, ..., X_N$

Many people start by defining time series as a sequence indexed by a subset of the natural numbers, but when we do theory, it will often be useful to think of the time series as a function from the positive reals $f : t \rightarrow (\mathcal{M}, d)$. This makes it possible to defer discussions of sampling until the very end. However, most of the real data we deal with has been uniformly sampled and is treated in a discrete fashion.

Examples of time series include sequences of real numbers or integers, as is the case with audio (Chapter 3), which is often referred to as a *1D time series*. In this case, the metric between values is simply the absolute distance between their values.

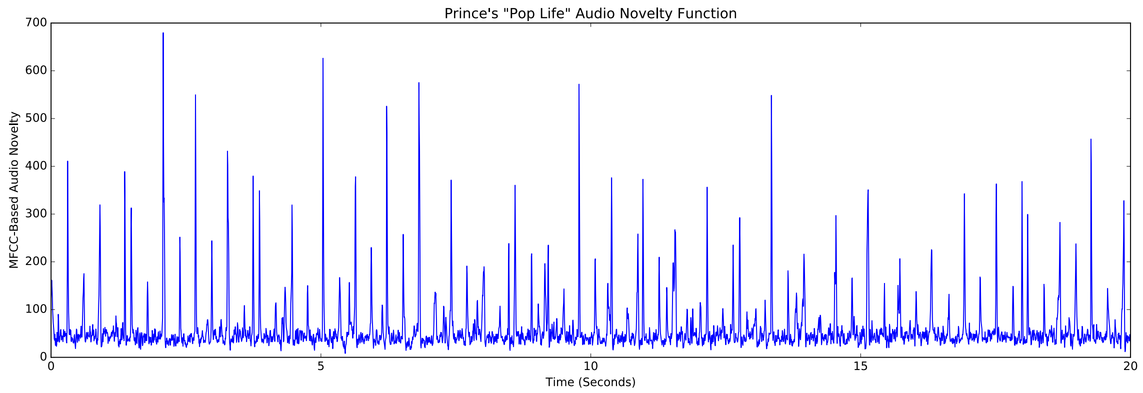


FIGURE 1.2: An audio novelty function (Ellis (2007a), Böck and Widmer (2013)) for Prince's "Pop Life." This signal has been designed to preserve percussive events correlated with beat onsets, while being at a much lower sample rate than the often cumbersome 44100hz standard music sample rate.

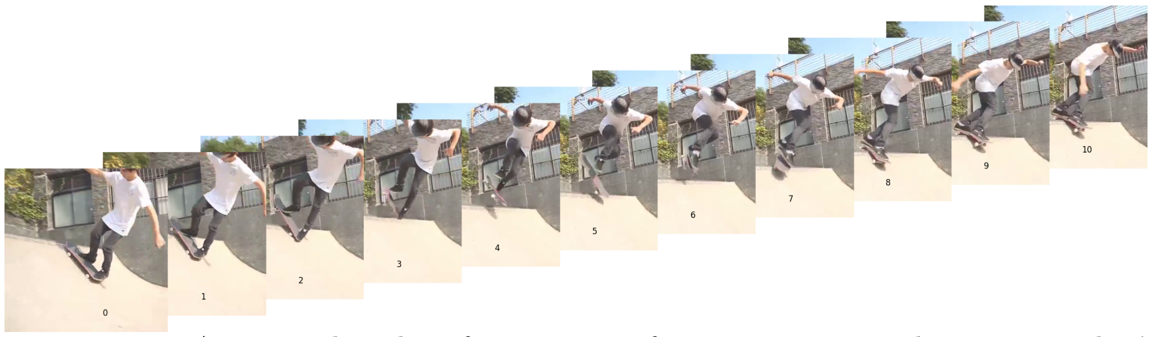


FIGURE 1.3: An example video of someone performing a 360 pop shove-it to rock n' roll on a quarter pipe. We analyze such videos as geometric time series in Chapter 4.

Figure 1.2 shows an example of a 1D sampled, real-valued time series correlating to rhythmic events in a Prince song. But the objects in question can be much more complex. For instance, a video is a sequence of images (Chapters 4, Figure 1.3)), and one can even consider a sequence of 3D shapes in this framework (Figure 1.4). In these cases, the chosen metric spaces need to be designed with more care.

One of the advantages of geometric and topological tools is that, in the abstract, our techniques can be applied equally to this diverse array of applications, regardless of the complexity of the underlying representations. For example, we think of a periodic time series as a topological loop, regardless of whether it represents repetitions in audio (Chapter 3) or expansion or contraction of the vocal folds in high speed videos of speech (Chapter 4). But an abstract mathematical approach is usually not enough, and we also often need to apply application-specific preprocessing techniques. When we do this, we usually leverage a long line of work from whatever community we are looking at, which we will highlight carefully in each chapter. As

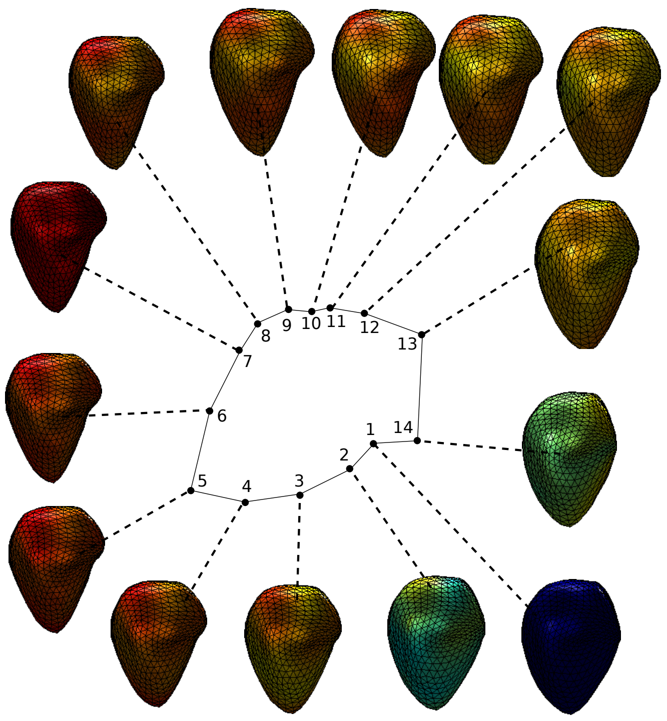


FIGURE 1.4: An example 3D video time series of a mesh of an expanding and contracting heart. Each mesh has been projected to $\mathbb{R}^2$ using classic multidimensional scaling after applying the Gromov-Hausdorff metric (Section 5.3.2). Large positive metric stress with respect to frame 1 is indicated with hot colors. Ultrasonic heartbeat data is courtesy of `http://lazax.com/www.cs.columbia.edu/~laza/html/heart3D/`

such, this work is equal parts math and engineering.

Figure 1.1 shows a block diagram which outlines the general pipeline we follow across all applications. For the rest of the intro, we will provide some high level details of that pipeline, including blocking and sliding window embeddings (Section 1.1), Community-Accepted Features (Section 1.1.3), geometric feature summaries (Section 1.3) (including self-similarity and cross-similarity matrices (Section 1.2)), and time warping (Section 1.4).

1.1 Block Windowing of Time Series/Time-Ordered Point Clouds

One of the key tools we use for translating time series into geometric objects is the so-called "delay embedding" or "sliding window" procedure, which is commonly used in dynamical systems and nonlinear time series analysis (Kantz and Schreiber (2004)). Sliding window embeddings convert time series into *time-ordered point clouds* (TOPCs), which live in what is often referred to as the *phase space* of the dynamical system. Our particular flavor of this is a so-called "block/window" pro-

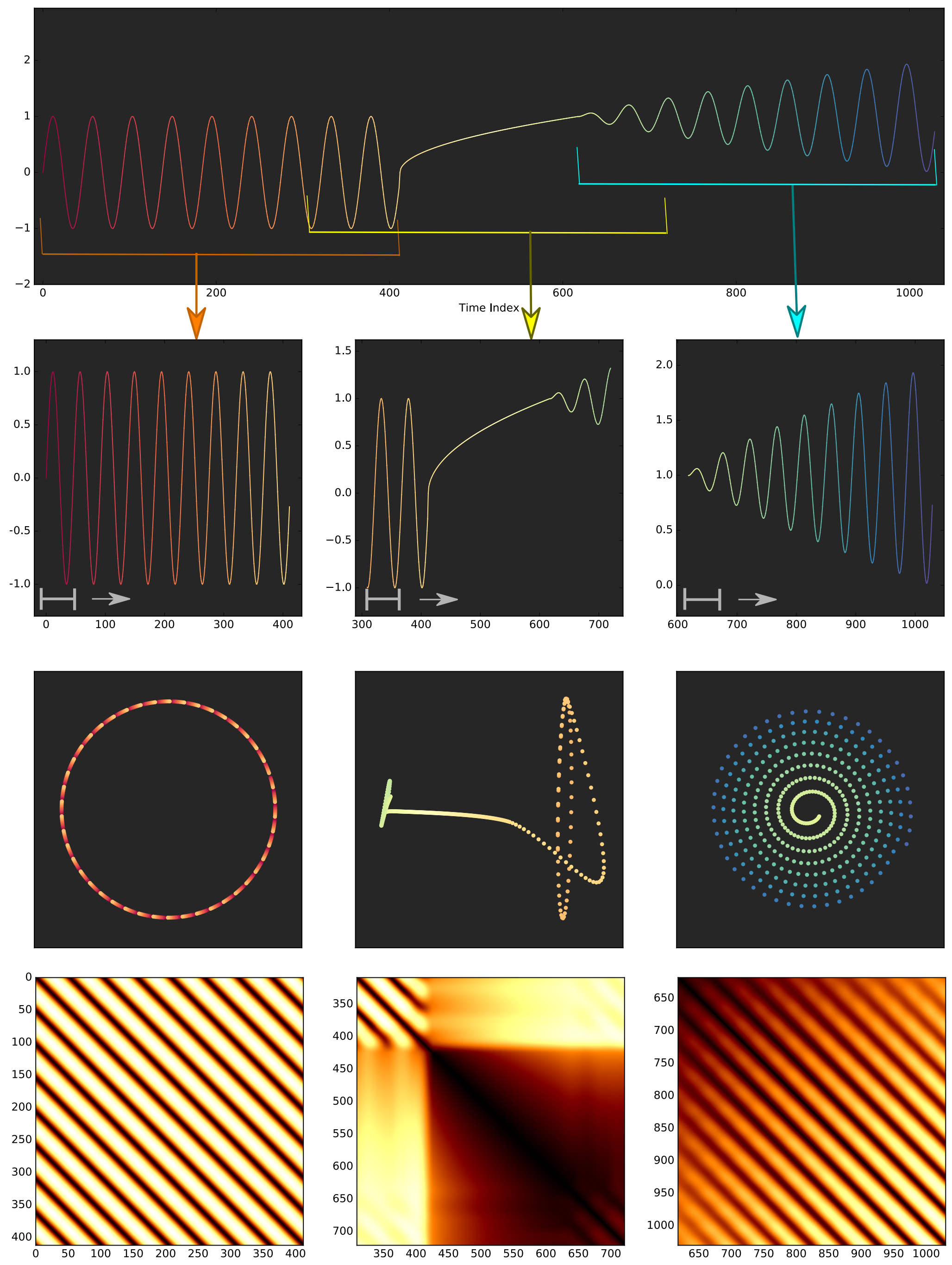


FIGURE 1.5: An example of block-windowing a 1D time series which is colored by time. A time series is shown on the top row, and three overlapping blocks are extracted. In each block, a sliding window is then taken (gray interval) and slid along to the right (gray arrow). The second to bottom row shows 2D PCA of the resulting Time-Ordered Point Cloud (TOPC)s, where the colors of the points correspond to the time at which the window started in the original time series. The bottom row shows the corresponding self-similarity matrices for each block.

cedure, and all of the applications in this thesis are a variation on the theme. A block is a contiguous chunk of a time series that is pulled out for processing. Within each block, a window is an even smaller chunk of the time series. A collection of windows within the block, ordered by time and endowed with some metric, forms a time-ordered point cloud associated with that block, and the union of all blocks covers the whole time series. Blocks and windows are similar to the "texture windows" and "analysis windows," respectively, in a well known paper on music genre recognition by Tzanetakis and Cook (2002), but we take a more explicitly geometric interpretation than the authors of that paper.

1.1.1 Formal Definitions

We now define block/windowing more formally, and link it to some similar schemes in the open literature

Definition 2. *Given a time series X defined in an interval $[a, b]$, a* block *is a subset of the time series defined on the interval $[a_i, b_i]$, where $a_i < b_i$. "Blocking" the time series refers to taking a set of K such blocks i so that*

- $a_i < a_{i+1}$, $b_i < b_{i+1}$
- $\bigcup_{i=1}^{K} [a_i, b_i] = [a, b]$

Blocking is useful when we want to process small chunks of the time series independently. Blocking is also useful when there is drift present in the time series (Section 1.1.2), because we can apply normalization within small chunks of the time series which haven't drifted.

Often, blocks overlap each other (i.e. $a_{i+1} < b_i < b_{i+1}$), so that some of the data is redundant between blocks. One reason for this is robustness, since it increases the chance that a block will exactly capture a section of the signal we are looking for. It can also be viewed as a form of data augmentation, which can help to get a better picture of the time series with a larger collection of blocks. Overlapping blocks have been used, for example, to improve hashing in audio fingerprinting (Wang et al. (2003)) (aka the "Shazam" service)) and to improve recovery guarantees in phase retrieval from Fourier Magnitude coefficients (Eldar et al. (2015)). More generally, some nontrivial overlap is usually present in standard spectrogram presentations of 1D time series. Note also that very similar schemes are present in image and video processing in applications of denoising and inpainting with overlapping patches (Buades et al. (2005), Barnes et al. (2009)).

Now within each block, we have a sliding window defined as follows:

Definition 3. *Start with a time series $f : t \to (\mathcal{M}, d)$ and a block B_i in the interval $[a_i, b_i]$. Given a time index $t \in [a_i, b_i]$, an integer $M \geqslant 0$, and a real number $\tau > 0$, then the* sliding window *in block B_i at time t is*

$$S_{M,\tau} f(t) = \begin{bmatrix} f(t) \\ f(t+\tau) \\ \vdots \\ f(t+M\tau) \end{bmatrix} \in (\mathcal{M}, d)^{M+1} \tag{1.1}$$

where $(\mathcal{M}, d)^{M+1}$ is some product metric over the product space $\mathbb{M}^{M+1}$

A collection of sliding windows at K different times $t_1, t_2, ..., t_K$ within a block, ordered by time, is known as the Time-Ordered Point Cloud *within that block. The product $M\tau$ is referred to as the* window size *of the sliding window.*

Note that a "time-ordered point cloud" can refer to any collection of points in a metric space indexed by time, but we often use it to refer specifically to a sliding window embedding. Note also that we usually chose the K time samples in the block to be uniformly spaced over $[a_i, b_i]$. And in practice, we are usually given samples of a time series as opposed to a continuous specification. If interpolation is possible in the metric space $(\mathcal{M}, d)$, then we do it (e.g. linear, sinc, or spline interpolation for 1D time series). Sometimes, however, interpolation is not possible, and we are limited to the samples we have ($\tau = 1$ by default). In this case, it is helpful to use tools from metric geometry that don't care about means, derivatives, etc. (Chapter 5).

Blocking aside, sliding window embeddings have been leveraged in a diverse set of applications, including cover song identification Serra et al. (2009) (which will be explained more in context in Chapter 3), music structure analysis (Bello (2011)), EEG data analysis (Stam (2005); Plesnik et al. (2014)), activity and gait recognition (Frank et al. (2010)), and gene expression time series data (Perea et al. (2015)), to name a few. There have also been several applications of this idea to video analysis specifically (Schödl et al. (2000), Huang et al. (2010), Venkataraman and Turaga (2016), Tralie (2016)), which we will explain more in context in Chapter 4.

Finally, Figure 1.5 shows an example of block-windowing a synthetic 1D time series, using the Euclidean metric. Notice how in each block, the time-ordered point cloud captures the dynamics of the signal in the block geometrically. The first block corresponds to a perfectly periodic region, and as we will explain more in Chapter 4, matching the window size to the length of the period gives rise to a circular point cloud. The middle block captures a transition, which starts off in a periodic section and then moves over to another periodic section in a different part of the embedding space. The rightmost block encapsulates an amplitude modulated periodic signal, which turns into a spiral in the embedding.

1.1.2 Sliding Window Length and Normalization

We have now defined precisely what we mean by blocks and windows, but how do we choose the parameters such as window size, interval sampling, etc? This is usually application specific, but there are a few common themes. First of all, a longer window size (i.e. larger M in Equation 1.1) means that there is more *context* summarized geometrically. In a way which will be made more precise with Taken's Delay Embedding Theorem (Theorem 2), a choice of more delays means we can effectively express higher order derivative information about the time series and more accurately reconstruct the state space of the underlying dynamical system. For a quick suggestive example, note that a linear transformation (specifically, a shear transformation) of the single delay embedding in a window $S_{1,\tau}[f(t)]$ can change into a coordinate system of a time series point and a discrete derivative, and similar transformations can be shown for higher order derivatives.

$$\begin{bmatrix} 1 & 0 \\ -1/\tau & 1/\tau \end{bmatrix} \begin{bmatrix} f(t) \\ f(t+\tau) \end{bmatrix} = \begin{bmatrix} f(t) \\ \frac{1}{\tau}(f(t+\tau) - f(\tau)) \end{bmatrix} \approx \begin{bmatrix} f(t) \\ f'(t) \end{bmatrix} \tag{1.2}$$

Knowing a derivative at a particular time point can tell us the difference between "going up" and "going down," even if we are passing through the same point, and higher order derivatives provide further disambiguation. However, when the window size is too long, there can be practical concerns about memory, especially when the time series objects already take up a lot of memory, such as video frames. We provide a computational scheme whose processing time and memory requirements are independent of window size for videos in (Chapter 4), given that $\tau = 1$ and there is no interpolation.

There are similar questions for how to choose the time sample intervals within each block, where choosing more guarantees better ϵ sampling of the continuous geometric object we want to summarize, but costs more memory and computation time. Finally, given that M is sufficiently large, choosing τ is important in so far as the window size $M\tau$ affects the geometry. For periodic processes, Broomhead and King (1986) suggests choosing a window size equal to the period, but as we will show in Chapter 4, any integer of a half multiple of a period works when quantifying periodicity.

There is one more very important practical concern which will be addressed in an application-specific manner. Sometimes, there is drift present in the time series, or there are changes in amplitude between blocks of a time series that are otherwise very similar. To mitigate this, we introduce various normalization schemes. Two basic normalization schemes that we sometimes employ are "Z-normalization" and "Point-Centering/Sphere Normalizing" (Perea and Harer (2015)) and they are defined as follows for point clouds in Euclidean spaces

Definition 4. *Given a point cloud X in a Euclidean space, let $\hat{X}$ denote its mean. Then for each $x_i \in X$ thought of as a column vector, the* z-normalized *version is*

$$\boldsymbol{z_i} = \frac{\boldsymbol{x_i} - \hat{X}}{||\boldsymbol{x_i} - \hat{X}||} \tag{1.3}$$

the point-centered/sphere-normalized *version is*

$$\boldsymbol{p_i} = \frac{x_i - (\boldsymbol{x_i}^T \mathbf{1})\mathbf{1}}{||\boldsymbol{x_i} - (\boldsymbol{x_i}^T \mathbf{1})\mathbf{1}||} \tag{1.4}$$

These two types of normalization are very similar, except Z-normalization subtracts off the mean point from all points, while the latter subtracts the mean coordinate off of each point point-wise. Both are attempts to mitigate drift. Amplitude is then controlled for by making the points all unit norm in both cases. Point-centering has theoretical reasons for being a good choice for sliding window videos when trying to quantify periodicity (Chapter 4), while Z-normalization is shown empirically to be a crucial step for cover song identification where there are lots of variations in loudness between cover versions of the same song (Chapter 3).

1.1.3 Community-Accepted Feature Maps (CAFMaps)

We now have a well-defined scheme for turning signals into point clouds and some idea of parameter choices and normalization schemes. However, a raw sliding window embedding on blocks is sometimes unwise, and it can be advantageous to additionally apply feature transformations in the sliding window space before further processing. For instance, in audio applications, the sample rate is 44100hz, and trying to capture windows of a half of a second leads to a noisy raw embedding spanning 22050 samples. Instead, audio information retrieval communities often summarize these windows with much lower dimensional features that are designed to retain perceptually relevant information (these features will be described more in Section 2.5). Figure 1.6 shows the drastic difference applying these "community-accepted features" (CAFs) onto the sliding window points can have on the resulting geometry.

More formally, we have the following soft definition of a "Community-Accepted Feature Map"(CAFMap):

Definition 5. *Given a sliding window block $S_{M,\tau}[f(t)] \subset (M_1, d_1)$, a* Community-Accepted Feature Map *is a (often nonlinear and non-invertible) map $\mathcal{G}$*

$$\mathcal{G} : S_{M,\tau}[f(t)] \subset (M_1, d_1) \to (M_2, d_2) \tag{1.5}$$

designed to preserve relevant information in a particular application domain.

We note right away that the Short-Time Fourier Transform (STFT) is a special case where $\mathcal{G} = \mathcal{F}$, and $\mathcal{F}$ is simply the Discrete Fourier Transform on each window independently. In this special case, $\mathcal{G}$ happens to be an isometry. Another common choice is the *audio spectrogram*, where $\mathcal{G} = |\mathcal{F}|$; or in other words, the Short-Time Fourier Magnitude Coefficients. This is no longer even invertible, but inversion is possible in some cases when the windows overlap enough, a problem known as "phase retrieval" (Eldar et al. (2015), Griffin and Lim (1984)). Likewise, in video applications, one could choose a map from color frames to grayscale, which is also sometimes invertible with enough prior information.

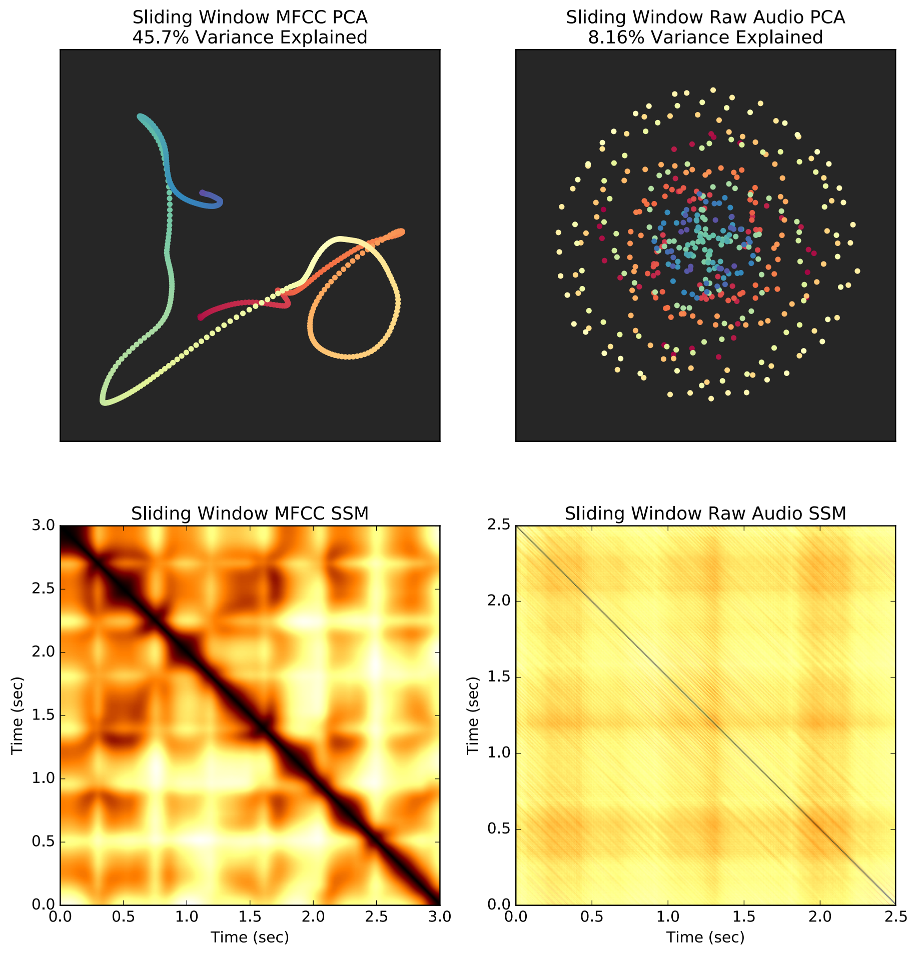


FIGURE 1.6: Comparing sliding window embeddings with and without summarizing community-accepted features (CAFs) on audio at a sample rate of 44100hz and a window size of 22050 (0.5 seconds). The sliding window embedding of the raw audio is much more spread out and noisy and can only be fully explained in a higher dimensional space, while summarizing each sliding window with 12 MFCC coefficients leads to clearer patterns with fewer dimensions.

More commonly, $\mathcal{G}$ is not invertible, but it preserves something relevant about the task at hand. For an extremely "lossy" example, most audio novelty functions (e.g. Figure 1.2), are obtained with a $\mathcal{G}$ that maps to a single real number for every window, which is often some function of the velocity of the sliding window at that point in time. In other words, broadband percussive events cause the sliding window embeding to "jump" from one point to another, so this function is useful for retaining rhythmic information. We provide many more examples of community-accepted features from music information retrieval in Section 2.5, such as "MFCC" and "Chroma," which we use in cover song applications (Chapter 3).

Finally, we note that when we need to normalize after applying these maps, Z-normalization makes the most sense here. Since these time-ordered point clouds no longer apply to sliding window embeddings, and the coordinates aren't simply lags of each other and are likely unrelated, it doesn't make sense to subtract the average coordinate. This is why we exploit Z-normalization with our cover songs application.

1.2 Self-Similarity Matrices (SSMs) And Cross-Similarity Matrices (CSMs)

1.2.1 Self-Similarity Matrices

Now that we have shown how to turn a time series into a time-ordered geometric object, we introduce one of the most important time-ordered geometric descriptors that unifies all of the work in this thesis:

Definition 6. *Given a time-ordered space curve parameterized by the unit interval* $\gamma : [0, 1] \rightarrow (\mathcal{M}, d)$*, a* Self-Similarity Image *is a function* $D : [0, 1] \times [0, 1] \rightarrow \mathbb{R}$ *so that*

$$D_{\gamma}(i, j) = d(\gamma(i), \gamma(j)) \tag{1.6}$$

If d *is the L2 metric on* $\mathbb{R}^d$*, a common choice, then this is referred to as a* Euclidean Self-Similarity Image.

The self-similarity image completely summarizes the metric information of the curve, disregarding a fixed ambient space with a fixed position and orientation in that space. In other words, self-similarity images are naturally *blind to isometries* of the underlying space curve. This makes the self-similarity image attractive for applications where it is desirable to be invariant to translation/rotation/flips of the curve.

Note that a "time-ordered space curve" is simply a continuous version of a time-ordered point cloud, and as such a self-similarity image is defined on a continuous 2D domain. In practice, self-similarity images must be sampled and treated as discrete objects for time-ordered point clouds.

Definition 7. *Given a time-ordered space curve* $\gamma : [0, 1] \rightarrow (\mathcal{M}, d)$*, its corresponding self-similarity image* D_{γ} *and a set of time indices* $t_1, t_2, ..., t_K$ *corresponding to a time-ordered point cloud, so that* $t_1 = 0$*,* $t_K = 1$*, and* $t_i < t_{i+1}$*, the* Self-Similarity Matrix(SSM) *is an* $N \times N$ *matrix* M_{γ} *so that*

$$M_{\gamma}[i, j] = D_{\gamma}(t_i, t_j) \tag{1.7}$$

As an example, self-similarity matrices are shown on the bottom row of Figure 1.5, corresponding to the respective blocks that they summarize in the blocked time series.

Self-similarity matrices have been used as a tool in pattern recognition across many domains. Junejo et al. (2008) apply self-similarity matrices to the problem of human activity recognition in video, where a video is treated as a time-ordered point cloud mapped into a feature space designed to summarize the frames of the video. They show that straightforward approaches using texture descriptors on self-similarity matrices to match the resulting curves have shown promising results for activity recognition, and this work inspired our approach to cover songs in Chapter 3. Self-similarity matrices were also used as a tool to detect periodicity and symmetry in video motion Cutler and Davis (2000) and in detecting copyright infringement in online video uploads. In the music information retrieval community, self-similarity matrices have been used on audio summarized in feature spaces, which can again be viewed as curves in those feature spaces, starting with the pioneering work of Foote (2000) which used a descriptor related to curvature in these images to detect note boundaries, with much followup work for music structure understanding and segmentation including more recent work by Bello (2009), Kaiser and Sikora (2010), Serra et al. (2012) and McFee and Ellis (2014). In the dynamical systems community, Euclidean self-similarity matrices are used as a format for reconstructing trajectories up to isometry, from which dynamical statistics can be computed McGuire et al. (1997).

In this thesis, self-similarity matrices are the unifying geometric descriptor across all applications. In the chapter on sliding window videos (Chapter 4), they are used to explain and motivate the reason to use sliding windows when quantifying time series of periodic processes. In the cover songs application (Chapter 3), they are used as geometric feature descirptors in their own right. In particular, we map from the sliding window point clouds with N points into the space $\mathbb{R}^{N \times N}$ under the Euclidean metric. This makes sense, because these images tend to look similar for the same songs performed with different instruments/singers/etc, even though the raw audio and derived MFCC features are quite different (figure 3.3 shows an example). This implies that small chunks of audio normalized properly are approximately isometries of each other between cover version of the same song. Finally, we use self-similarity images as the primary representation for designing isometry blind time warping in Chapter 5.

1.2.2 Cross-Similarity Matrices

In addition to 2D symmetric matrices which compare a time-ordered point cloud to itself, we also sometimes use general (not necessarily symmetric or even square) 2D matrices which compare two time-ordered point clouds in the same metric space to each other, known as "cross-similarity matrices" (CSMs).

Definition 8. *Given a time-ordered space curve $\gamma_1 : \mathbb{R}^+ \to (\mathcal{M}, d)$ sampled at times $s_1, s_2, ..., s_M$ and another time-ordered space curve $\gamma_2 : \mathbb{R}^+ \to (\mathcal{M}, d)$ sampled at times $t_1, t_2, ..., t_N$, the corresponding* Cross-Similarity Matrix (CSM) *between sampled γ_1 and γ_2 is an $M \times N$ matrix C_γ so that*

$$C_{\gamma_1,\gamma_2}[i,j] = d(\gamma_1(s_i), \gamma_2(t_j)) \tag{1.8}$$

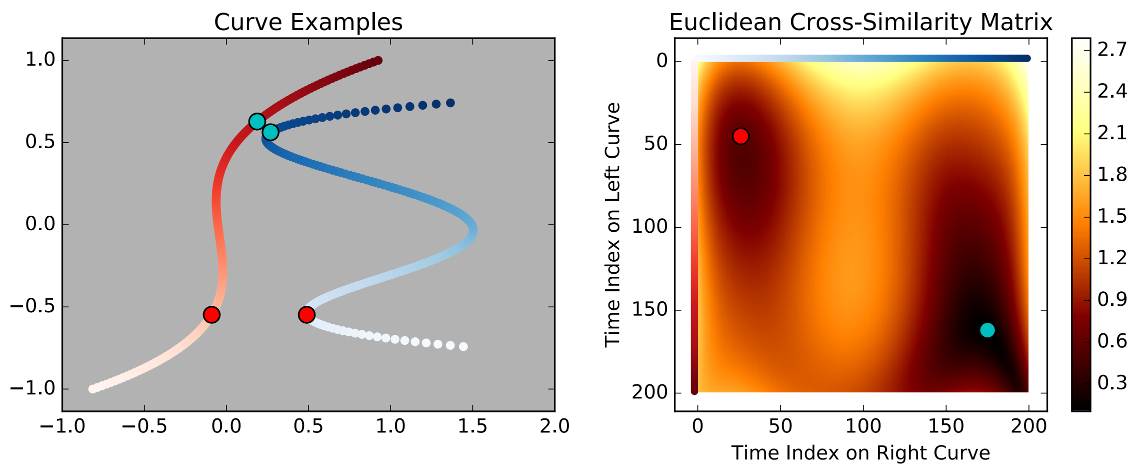


FIGURE 1.7: An example of a Euclidean cross-similarity matrix describing the distances between two curves. The time axes of the rows and columns are colored in accordance with the time parameter in the respective curves. Two example points are marked in the CSM where the curves get closer to each other relative to the neighborhoods of the points.

Note that a CSM from a curve to itself sampled at the same time indices is an SSM, but the CSM concept applies more generally to two curves. CSMs show up mainly in alignment applications, where for time-ordered point clouds the ij^{th} pixel is the distance between the i^{th} sliding window in the first point cloud and the j^{th} sliding window in the second point cloud. The alignment problem reduces to finding a minimum cost path from the upper left to the lower right of the CSM image (Figure 2.20).

Binary versions of these matrices can also be defined, both with an ϵ threshold

$$B^{\epsilon}_{\gamma_1,\gamma_2}[i,j] = \left\{ \begin{array}{cc} 1 & d(\gamma_1(s_i), \gamma_2(t_j)) < \epsilon \\ 0 & \text{otherwise} \end{array} \right\} \tag{1.9}$$

or with a mutual k-nearest neighbors threshold

$$B^{k}_{\gamma_1,\gamma_2}[i,j] = \left\{ \begin{array}{cc} 1, & d(\gamma_1(s_i), \gamma_2(t_j)) < \min(\epsilon^k_{1i}, \epsilon^k_{2j}) \\ 0, & \text{otherwise} \end{array} \right\} \tag{1.10}$$

Where ϵ^k_{1i} refers to the distance of the k^{th} nearest neighbor of $\gamma_1(s_i)$ in $\gamma_2(t_1, t_2, ..., t_M)$, and vice versa for ϵ^k_{2j}. These are referred to as "cross-recurrence plots" (Marwan et al. (2000), Serra et al. (2009)). An analogous version for self-similarity matrices, used for analysis of dynamical systems, is known simply as a "recurrence plot" (Eckmann et al. (1987), Marwan et al. (2007)) [1]. Diagonal lines of ones in these matrices indi-

[1] For more information on recurrence and cross-recurrence plots, the web site `http://www.recurrence-plot.tk` has an exhaustive literature review on the subject, as of the time of writing (2/2017).

```
def getCSM(X, Y):
    """
    Return the Euclidean cross-similarity matrix between the M points
    in the Mxd matrix X and the N points in the Nxd matrix Y.
    :param X: An Mxd matrix holding the coordinates of M points
    :param Y: An Nxd matrix holding the coordinates of N points
    :return D: An MxN Euclidean cross-similarity matrix
    """
    #First, compute the squared magnitude of each point in X and Y
    XSqr = np.sum(X**2, 1)
    YSqr = np.sum(Y**2, 1)
    #The squared distance between x_i and y_j is
    #||x_i||^2 + ||y_j||^2 - 2x_i*y_j.
    #Broadcast the ||X||^2 along each column and ||Y||^2 along each row
    C = XSqr[:, None] + YSqr[None, :] - 2*X.dot(Y.T)
    C[C < 0] = 0 #Before taking the square root, do this in case
    #numerical instability caused one of the entries to be
    #less than zero
    return np.sqrt(C)
```

Listing 1.1: Fast Python/Numpy code for Euclidean SSM and CSM computation exploiting matrix mutiplication and broadcasting

cate entire sequences matching between two time-ordered point clouds, a fact which we will exploit heavily in our cover songs work.

In our work, we find the k-nearest neighbor version of the cross-recurrence plot substantially boosts performance when aligning cover songs (Chapter 3). Intuitively, in some cases, using binary k-nearest neighbors discards unnecessary and possibly noisy metric information that could degrade performance.

1.2.3 Fast Code

Code for SSMs / CSMs

In our cover songs application (Chapter 3), we need to compute thousands of self-similarity or cross-similarity matrices between TOPCs in high dimensional Euclidean spaces, which necessitates some computational optimization. In this special Euclidean case, it is possible to write very efficient code, based on the following mathematical fact about squared Euclidean distances between two vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$:

$$||\boldsymbol{x} - \boldsymbol{y}||_2^2 = \boldsymbol{x} \cdot \boldsymbol{x} + \boldsymbol{y} \cdot \boldsymbol{y} - 2\boldsymbol{x} \cdot \boldsymbol{y} \tag{1.11}$$

Now let X be an $M \times d$ matrix with each Euclidean point x_i arranged along a row, and let Y be the analogous $N \times d$ matrix for the y_js. Let $|X|$ be the $N \times 1$ matrix with the squared magnitudes of all of the x_is along the rows, and let $|Y|$ be the analogous $M \times 1$ matrix for the y_js. Then the Euclidean cross-similarity matrix can be written as the square root of the sum of three matrix multiplications:

```python
def CSMToBinary(D, Kappa):
    """
    Turn a cross-similarity matrix into a binary cross-simlarity matrix
    If Kappa = 0, take all neighbors
    If Kappa < 1 it is the fraction of mutual neighbors to consider
    Otherwise Kappa is the number of mutual neighbors to consider
    """
    N = D.shape[0]
    M = D.shape[1]
    if Kappa == 0:
        return np.ones((N, M))
    elif Kappa < 1:
        NNeighbs = int(np.round(Kappa*M))
    else:
        NNeighbs = Kappa
    #Do a merge partition on each row
    J = np.argpartition(D, NNeighbs, 1)[:, 0:NNeighbs]
    #Set up a sparse binary cross-similarity matrix
    I = np.tile(np.arange(N)[:, None], (1, NNeighbs))
    V = np.ones(I.size)
    [I, J] = [I.flatten(), J.flatten()]
    ret = sparse.coo_matrix((V, (I, J)), shape=(N, M))
    return ret.toarray()
```

Listing 1.2: Fast Python/Numpy code using merge partition to determine nearest neighbors if a CSM has been precomputed

$$\text{CSM} = \sqrt{|X|\mathbf{1}^T + \mathbf{1}|Y|^T - 2XY^T} \tag{1.12}$$

where $\mathbf{1}$ refers to an appropriate column matrix of all 1s. Incidentally, this can be used to show that the rank of a squared Euclidean cross-similarity matrix between points in $\mathbb{R}^d$ is bounded by $d + 2$, regardless of the number of points in X and Y, since the cross-similarity matrix is the sum of two rank-1 matrices and one rank-d matrix.

Listing 1.1 shows how to exploit this fact in Python/Numpy using matrix multiplication and row/column broadcasting (in place of the outer product on the rank-1 matrices), which is quite fast in practice and can be easily parallelized on the GPU.

Fast Code for Nearest Neighbors

If we have already computed a full CSM, the fastest way to find all sets of k-nearest neighbors is to use *merge partition* on each row to partition all of the k^{th} smallest elements to be at index k or less. In practice, this is faster than sorting, as we don't care that they are in order, just that they are all contiguous so that we can easily extract them. The average time complexity of merge partition is $O(N)$ per row, with a worst case $O(N^2)$, but in practice this is substantially faster than the $\Theta(N \log N)$

time complexity of merge sort, especially since it needs to be done on each row. The code in Listing 1.2 shows how to do this in Python/Numpy.

1.3 Geometric Feature Summaries

Though self-similarity matrices play a key role in our work, there are many other geometric descriptors can be used to summarize time-ordered point clouds in blocks. For example, Tzanetakis and Cook (2002), who introduced a block-windowing procedure to the music information retrieval community, summarize each block by the mean and variance of each CAF sliding window dimension over all windows in the block, which could be viewed a simplified ellipsoid fitting of the time-ordered point cloud in each block (ignoring covariance). Also, Venkataraman and Turaga (2016) use a histogram of pairwise distances to summarize sliding window embeddings of videos for activity recognition. Variants of this feature, often referred to as "D2 shape histograms" have been popular in the computer graphics community for describing 3D shapes (Osada et al. (2002), Mahmoudi and Sapiro (2009)), because, like self-similarity matrices, they are blind to isometries, but unlike self-similarity matrices, they require no alignment.

We now briefly summarize a few of the additional geometric features we use in this work, which will be explained more in the background and as needed. In general, we want features which are blind to isometries of the underlying space, but which, unlike the previous two examples, also retain information about the time order of the time-ordered point clouds in each window. To demonstrate these features, we generate two synthetic TOPCs. The first is what as known as "Viviani's figure 8" (Figure 1.9), which is the intersection of the sphere $x^2 + y^2 + z^2 = 4a^2$ and the cylinder $(x - a)^2 + y^2 = a^2$. In particular, it is parameterized as

$$\vec{X_V}(t) = \begin{bmatrix} a(1 + \cos(t)) \\ a\sin(t) \\ 2a\sin(t/2) \end{bmatrix} \tag{1.13}$$

The second one is a 3-5 torus knot (Figure 1.8), which is parameterized by

$$\boldsymbol{X_{T3,5}}(t) = \begin{bmatrix} r\cos(3t) \\ r\sin(3t) \\ -\sin(5t) \end{bmatrix} \tag{1.14}$$

We find variants of the Figure 8 are good examples for isometry blind time warping (Chapter 5), while the torus knot is similar to some patterns we see in sliding window embeddings of harmonic periodic video sequences (Chapter 4), so they are both worthy examples for motivating our feature choices.

As before, we compute the SSMs, which describe all of the geometry up to an isometry, retaining the time order of the point cloud. We also compute *diffusion maps* (Coifman and Lafon (2006)) and provide their associated SSMs. Diffusion

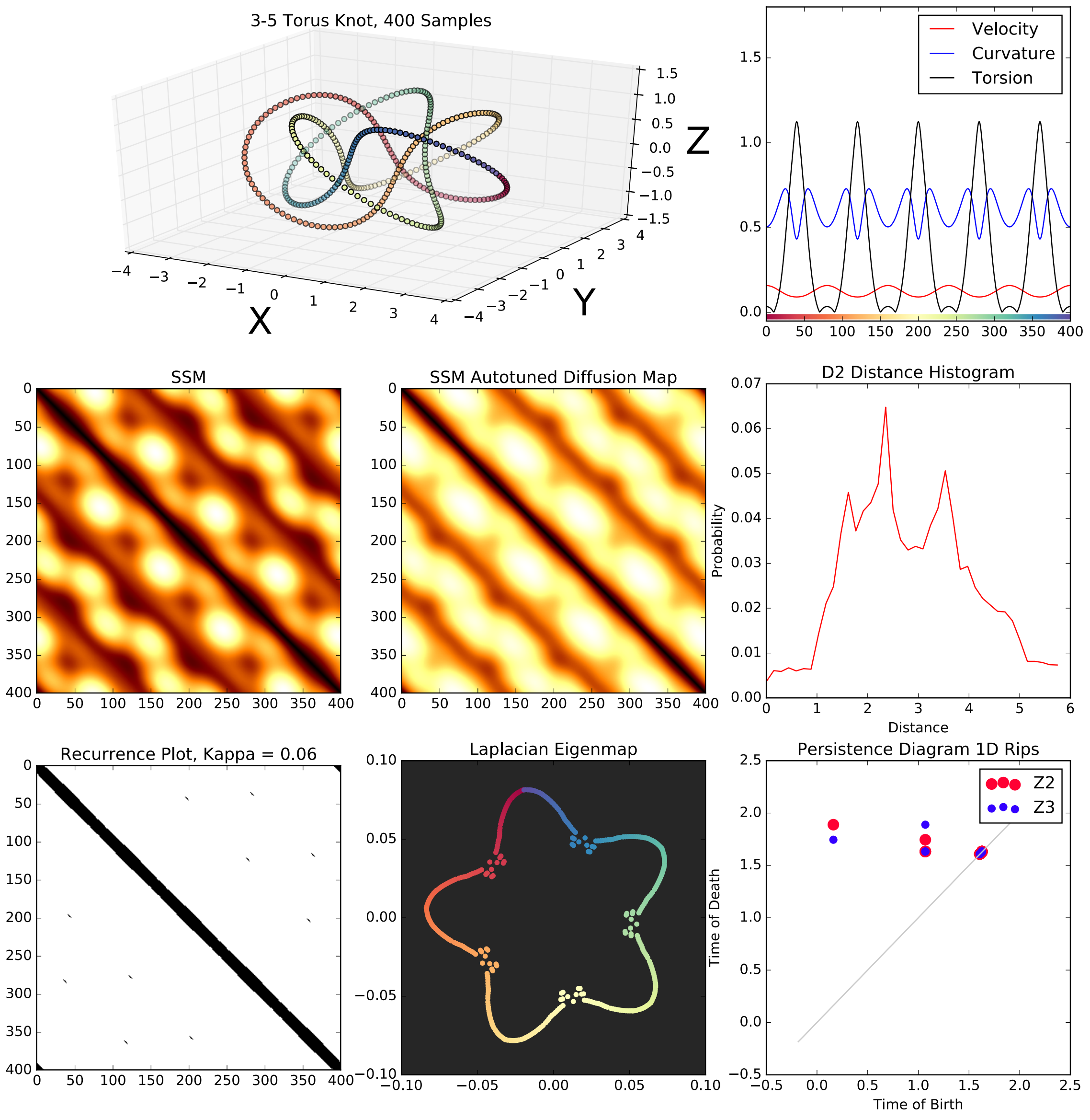


FIGURE 1.8: Different geometric features describing a time-ordered point cloud sampled from a 5-3 torus knot. The autotuned diffusion map reduces the impact of the 5-part (smaller radius loop), effectively flattening the torus knot to be closer to three loops. Unweighted Laplacian Eigenmaps actually completely un-knot the knot and map it to one loop around a topological circle, and for an even smaller κ, the curve would be a perfect geometric circle. Finally, the complicated geometry causes some swapping between the $\mathbb{Z}_2$ and $\mathbb{Z}_3$ field coefficients in the 1D persistence diagram. We exploit a similar phenomenon with paths on high dimensional flat tori to detect harmonics frequencies in sliding window videos of vocal cords (Chapter 4).

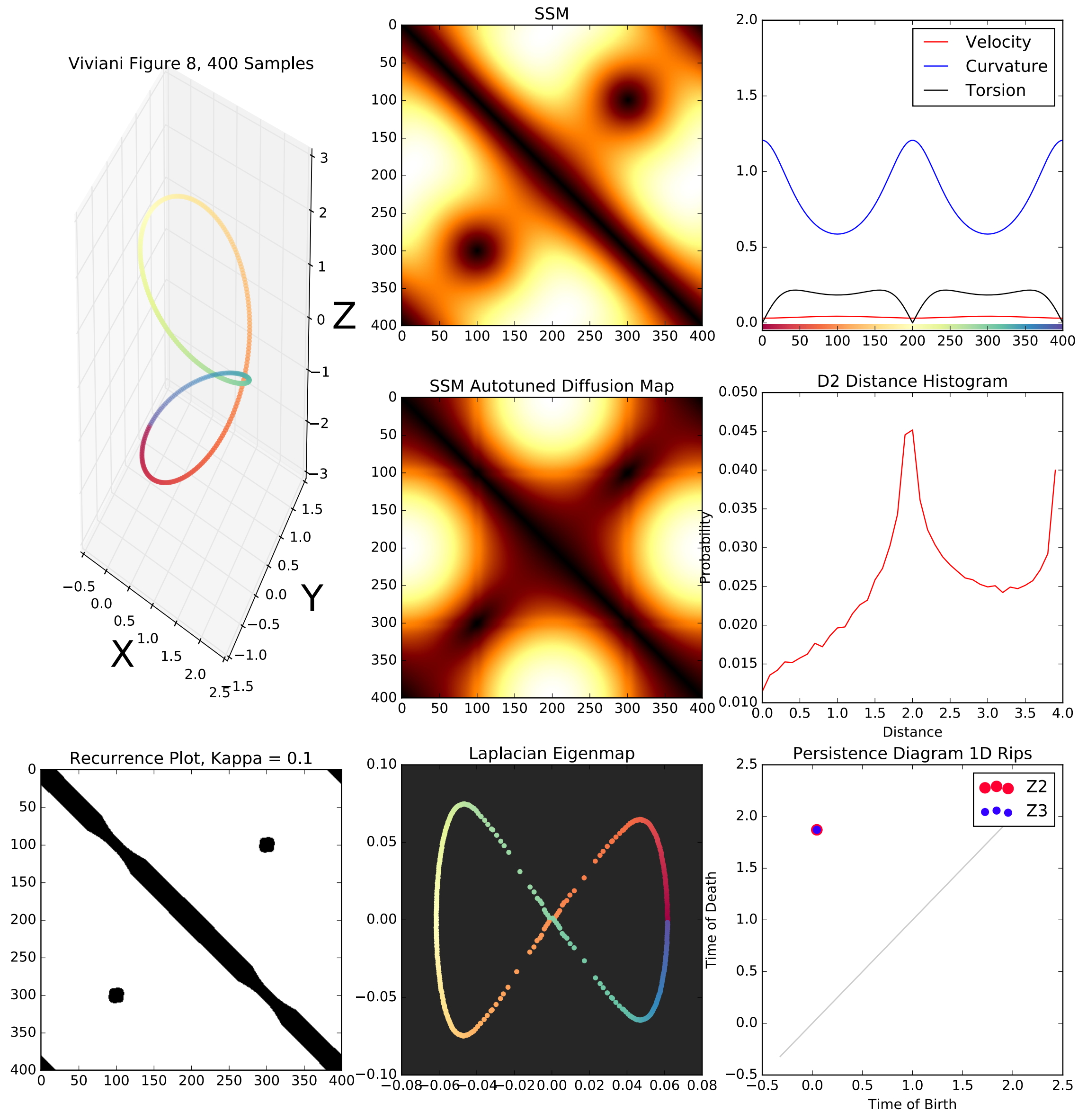


FIGURE 1.9: Different geometric features describing a time-ordered point cloud sampled from a Viviani Figure 8 (intersection of a sphere and cylinder, also type of "Clelia curve" on the sphere). Curvature peaks at the two extremities of the figure 8, which is apparent in the curve rendering. The first two eigenvectors of the Laplacian Eigenmap are a planar Lissajous curve (the curve that results on an oscilloscope when two harmonic signals are plotted against each other). The $\mathbb{Z}_2$ and $\mathbb{Z}_3$ coefficient 1D persistence diagrams are the same in this case, and they both have a point of multiplicity two (one for each loop in the figure 8).

maps are designed to preserve the *intrinsic geometry* of a point cloud, factoring out unnecessary curvature in the ambient space. There are many parameters to tune with these maps, but we provide a scheme to autotune them in Section 2.3.4 that works reasonably well. Noting that diagonals in an SSM indicate repeating sections, we notice the autotuned diffusion maps have effectively flattened the torus knot to 3 loops in Figure 1.8.

In addition to diffusion maps, we also use Laplacian Eigenmaps (Belkin and Niyogi (2003)) on nearest neighbor graphs, which have a similar purpose. In Figure 1.9 and Figure 1.8, we compute the Laplacian Eigenmaps on the unweighted graph obtained from the mutual k-nearest neighbor recurrence plot. We note that in the torus knot example, this has the effect of unraveling the knot and mapping it to one revolution around a loop in the plane. Because of this, these tools are particularly useful when the time order is not known for periodic processes, and maps to the circle need to be found in spite of complicated geometry. To complement diffusion maps and Laplacian Eigenmaps, we also compute numerical estimates of velocity, curvature, and torsion for the time-ordered point clouds.

Finally, we compute two types of features which lose the time order, but which can still be useful. First, we compute the D2 histogram we mentioned at the beginning of this section, for comparison. Secondly, and more importantly, we compute features from topological data analysis (Edelsbrunner and Harer (2010)), known as "Rips Filtrations," which will be explained much more in the background. Roughly, they describe the multiscale prominence of topological loops in the time-ordered point clouds. As such, they are particularly apt for quantifying the periodicity of time series, which give rise to topological loops in the sliding window embedding, as demonstrated in Figure 1.5 (Chapter 4). Using coefficients in different fields can also be used to find *algebraic torsion*, indicates if our loops bound non-orientable surfaces. This is surprisingly useful for detecting if a periodic sliding window video consists of a base frequency and an integer harmonic.

1.4 Isometry Blind Time Warping And Alignment

We have now outlined the majority of our pipeline, but there is one more issue that needs to be addressed. In the cover songs application, we often need to match songs which are at different tempos, and which potentially have missing or added beats and sections. In addition, changes in instrumentation cause the sliding window point clouds to move to different parts of the state space, even if the relative shapes are the same. The latter issue can be addressed by sticking to features which are blind to isometries, assuming the instrument changes are represented roughly by isometries after normalization (an assumption which appears to hold up well in Chapter 3). But for the former issue, we need alignment techniques which we have not yet discussed. Our solution for the cover songs is to apply off-the-shelf algorithms for gene sequencing, such as Smith Waterman (Smith and Waterman (1981)), to match the

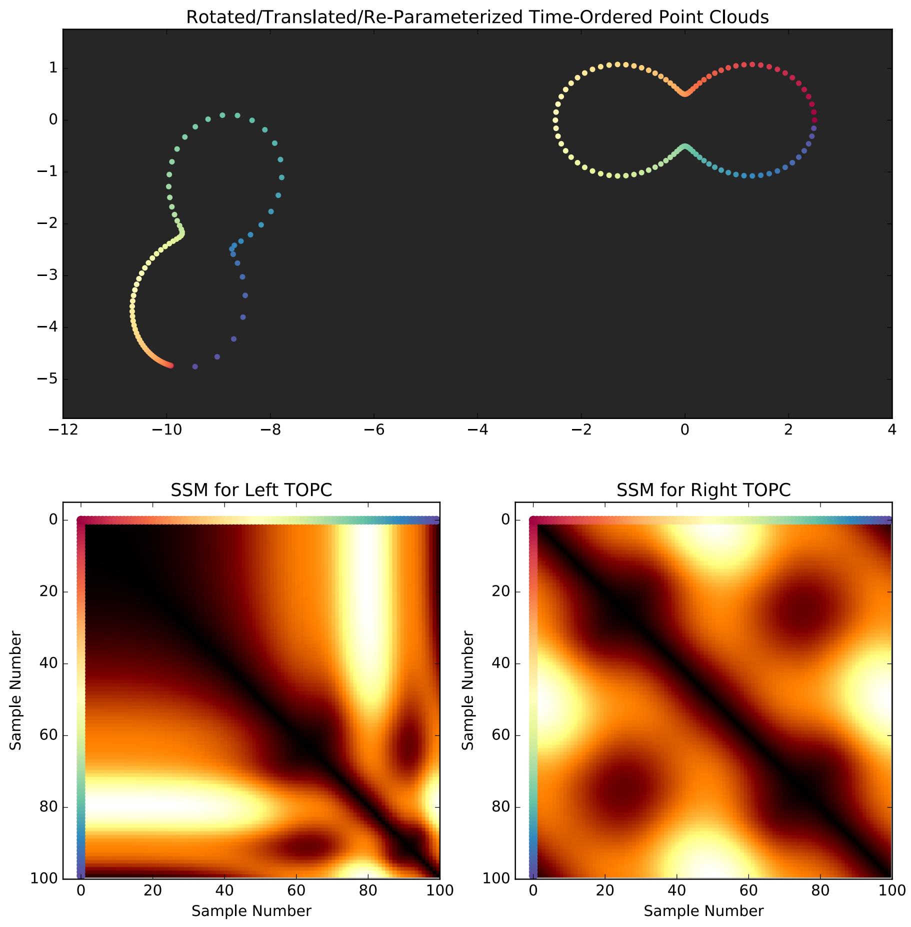


FIGURE 1.10: An example of what happens to self-similarity matrices after time warping and rotation/translation. Since SSMs are naturally blind to isometries, the rotation/translation has no effect, but the re-parameterization induces an image warping on the SSMs, which we explore more in Chapter 5.

blocks via a cross-recurrence plot.

While working on this problem for cover songs, however, we were also motivated to tackle the more general problem of isometry blind time alignment. This is a difficult problem, since not only are the point clouds time warped version of each other, but they are also not aligned spatially. It also has real applications in addition to cover song identification. For example, one might want to synchronize two motion capture gestures from different people performing the same action (Hsu et al. (2005)). Each person is in a different part of the space (approximate isometry if they are a similar body shape), and they likely perform the actions at a different rate (time warping).

Figure 1.10 shows an example of rotating/translating/re-parameterizing a time-ordered point cloud. SSMs are a good base representation, because they are naturally blind to isometries, so we can focus on discovering an image warp between them to uncover the underlying re-parameterization. The goal is to find an optimal *correspondence* between the top TOPCs, which aligns the SSMs as optimally as possible under some objective function. We will be more precise about this in Chapter 5, but for now we formally define a correspondence:

Definition 9. *Given two sets X and Y, a* correspondence *$\mathcal{C}$ between the two sets is such that $\mathcal{C} \subset X \times Y$ and $\forall x \in X \exists y \in y$ s.t. $(x, y) \in \mathcal{C}$ and $\forall y \in Y \exists x \in X$ s.t. $(x, y) \in \mathcal{C}$*

In other words, a correspondence is a matching between two sets X and Y so that each element in X is matched to at least one element in Y, and each element of Y is matched to at least one element in X. Searching over all correspondences would lead to a combinatorial explosion, and it was recently shown that finding the optimal metric correspondence, known as the Gromov-Hausdorff Distance, is NP complete (Agarwal et al. (2015)). However, if we impose constraints on valid correspondences, particularly constraints which preserve a time order, then we are able to design polynomial time algorithms which address this problem. We discuss these issues and present our new algorithms in the mostly theoretical Chapter 5.

As a bonus, even if the time-ordered point clouds aren't in the same metric space, we would still like to compare them. This means our techniques could be potentially used in cross-modal applications, such as aligning audio to video (Patterson et al. (2002)). As we will show in Section 4.3, different modalities can have much different geometries, even if they are describing the same underlying object. We use diffusion maps as a preprocessing step to make cross-modal time-ordered point clouds closer to being isometries across modalities.

1.5 Summary of Novel Contributions

Since this thesis straddles electrical engineering, math, and computer science, our background is extensive. To help the reader, we created an annotated list of novel contributions in this work below, by chapter

- Chapter 2: Background

  Although this chapter is mostly explaining existing concepts in our own words and with a consistent notation set, we did include a few small ideas of our own that would not be a chapter unto themselves

  - We demonstrate how Laplacian eigenmaps of sliding window embeddings of periodic videos can be used to re-arrange the samples of the signal to get a "slow motion" view of a single period, even if the sample rate is low (Section 2.3.3)
  - We create a novel user interface to synchronize PCA of feature summaries on sliding window embeddings of audio, known as "Loop Ditty" (`http://www.loopditty.net`, Section 2.5.6)

- Chapter 3: Automatic Cover Song Identification
  - We show how block-normalized MFCC features and SSMs of MFCC features can be used in the cover songs problem, where MFCC features were thought not to have any relevant information before
  - We show how fusing MFCC and HPCP features with Similarity Network Fusion improves cover song identification over either one alone.
  - We show a novel low level metric fusion technique that significantly improves classification performance if applied before local alignment, achieving state of the art in cover song identification.
  - We show that our timbral features are more appropriate for assessing music similarity than chroma features in the recent "Blurred Lines" music copyright controversy (Section 3.4.2).
- Chapter 4: Sliding Window Videos
  - We extend the delay reconstruction technique from 1D time series to video as a concatenated delay series of all pixels (Section 4.1)
  - We come up with theoretical models to explain the geometry that results from these delay embeddings (Section 4.3). We prove that the geometry lives on curved hypertori even for signals which oscillate with only one frequency, and we show that harmonic signals often live on the boundary of a Möbius strip.
  - We provide techniques to reduce computation and cut down on drift without requiring any video tracking (Section 4.4).
  - We devise periodic, quasiperiodic, and harmonic scores of motion in videos (Section 4.4.3). The quasiperiodicity score in particular has one of the first known uses of persistent H_2 in high dimensions for a practical purpose.
  - We provide a way to autotune the window size of delay embeddings by using techniques from audio processing on 1D surrogate signals derived from the video using diffusion maps (Section 4.4.5).
  - We show that our periodicity score of videos agrees well with aggregated human rankings from the Amazon Mechanical Turk (Section 4.5)
  - We show that our method can be used to quantify anomalies in high speed videos of vibrating vocal cords without any motion tracking (Section 4.6)
- Chapter 5: Isometry Blind Time Warping
  - We introduce a generalization of dynamic time warping which can synchronize two time-ordered point clouds which have been spatially shifted without aligning them spatially (Section 5.3). We show that this lower

bounds the 1-stress between the pairwise distances of both point clouds restricted to warping paths, and we provide a GPU algorithm to parallelize computation

- We show that critical points in self-similarity images are preserved under re-parameterizations of the underlying time series, and we provide interpretations of what different critical points say about geometry (Section 5.4)
- We provide a persistence watershed algorithm to quantify critical points in self-similarity images if time-ordered point clouds (Section 5.4.3), which complements the rips complex approach that is normally used on point clouds

2

Background

To keep this thesis self-contained, we have some background to cover from a diverse array of topics, in addition to the over-arching signal to geometry pipeline given in the previous chapter. One of the contributions of our work is making connections between disparate fields, and we hope to highlight the ways in which non-traditional tools from mathematics are useful when combined with traditional signal processing techniques. Surprisingly, most of the topics below are connected to each other, and we try to highlight the connections where applicable.

2.1 Topology / Topological Data Analysis

The word "topos" is the Ancient Greek word for "place." So "topology" is the "study of place." Like its cousin geometry, topology is concerned with the study of shape, but it is not concerned with metric properties such as length, area, angles, etc. Rather, it is the study of how spaces are "connected," and topological spaces are thought of as equivalent under deformations that do not involve gluing or tearing. For example, a line segment and a curve are topologically equivalent, but they are not topologically equivalent to a loop. Common topological properties of interest are connected components, loops/cycles (and their higher dimensional analogues tunnels/voids), and orientability/twists. More information on topological spaces can be found in Munkres (1975), but we briefly repeat some of the basics here at a high level for engineers or computer scientists who may never have seen this before.

Topological spaces can be defined purely in terms of designated "open sets," and proximity in these spaces can be understood by studying these open set systems. In particular,

Definition 10. *Given a set X, a collection of subsets τ of X forms a* topological space (X, τ) *if the three following properties are satisfied*

- $\varnothing, X \in \tau$
- *A union of arbitrary collections of sets in τ is in τ, whether the union is finite, countably infinite, uncountably infinite, etc.*
- *The intersection of any* finite number *of sets in τ is in τ*

The sets in τ are known as "open sets" and the complement of any open set is known as a "closed set."

Though this definition may seem abstract, it turns out to be what's needed to capture all of the properties of interest in topology while also generalizing long-standing notions of "open" and "closed" on the real line. Note that a topology can be defined for any metric space as the union of balls in that metric space, and it can be shown, for example, that unions of L^p balls in $\mathbb{R}^d$ give rise to the same topology, regardless of p (though there are topologies which aren't realizable this way). This means that many important properties from metric spaces can be understood in a much more general (and often simpler) framework. For example, continuity is purely in terms of open sets:

Definition 11. *Given two topological spaces X and Y, a map f between them is* continuous *if for every open set $V \in Y$, the inverse image $f^{-1}(V)$ is open in X*

It can be shown that the "delta/epsilon" version of continuity between metric functions is a special case of this definition with open balls defined in the domain and range spaces, respectively.

The primary equivalence between topological spaces exists between spaces which are *homeomorphic*. More precisely:

Definition 12. *A* homeomorphism *h between two topological spaces X and Y is a continuous, bijective map whose inverse is also continuous. Two spaces between which a homeomorphism exists are said to be* homeomorphic*.*

Intuitively, we must be able to find a map between two topological spaces that doesn't take things "too far" away from each other in the forward or backwards direction, where proximity is understood in terms of the open sets at hand via the definition of continuity above. Homeomorphisms precisely disallow the "gluing" and "tearing" we seek to avoid. A common non-example of a homeomorphism is the following function between the half open unit interval and the circle S^1 (each with "subspace topologies" inherited from a topology generated on $\mathbb{R}^2$ by intersections of the segment and circle with arbitrary unions of open balls in $\mathbb{R}^2$):

$$f : t \in [0, 1) \to S^1, \text{given by} f(t) = (\cos(2\pi t), \sin(2\pi t)) \tag{2.1}$$

This function is certainly bijective, and it's continuous in the forward direction, but it fails continuity in the inverse direction at the point $(1, 0)$, because there is no open set in S^1 around $(1,0)$ that contains an open set mapped by f around $t=0$. Intuitively, the map f brings points around $t=0$ very close to points around $t=1$ in the range space (i.e. there are open sets containing their image in the range space), where they were far in the domain space. This is a very precise way of saying it would be necessary to glue the line segment at its endpoints to establish topological equivalence with the circle. On the other hand, the map from the circle to the square (in polar coordinates)

$$f : (1,\theta) \in S^1 \to (\max(|\cos(\theta)|, |\sin(\theta)|), \theta) \tag{2.2}$$

is, in fact, a homeomorphism between the circle and the square (it maps the circle to its inscribed square diamond), so a circle is topologically equivalent to a square. Note that this map is non-smooth (i.e. not a *diffeomorphism*) when considering the metrics of the square and circle, but topology is blind to this.

If two spaces are homeomorphic, then those two spaces have all topological properties of interest in common, so in categorical language they are "isomorphisms in the category of topological spaces." It is important to note, however, that while particular maps may fail to be homeomorphisms, it is in general quite difficult to establish that no homeomorphisms exist between two spaces. For example, though the example in Equation 2.1 is encouraging, it is not enough to prove that the circle and the line segment are not topologically equivalent. In fact, the general problem of homeomorphy is *undecidable*. Therefore, it is useful to develop *topological invariants*, or quantities that can be computed over topological spaces which are preserved under homeomorphisms. If these topological invariants are different between two different topological spaces, then that is enough to prove that they are not homeomorphic. We will be primarily concerned with a topological invariant known as *homology* in this work, since it is one that can be computed with standard tools in linear algebra, but there are many others (e.g. connectedness, compactness, homotopy).

We conclude this introduction to topology by noting that while it was long thought to be merely an abstract branch of mathematics, it has found many interesting applications in science and engineering, surprisingly demonstrating its practical worth in recent times. The author's personal introduction to the subject was via computer graphics. For instance, sometimes it is desirable to find and fill in holes in a surface that occurred during scanning (Zhao et al. (2007)), or surfaces with nontrivial topology need to be cut so they can be effectively flattened to the plane for texture mapping (Erickson and Whittlesey (2005)) [1]. But there are many other applications. In fact, the most recent nobel prize in physics (at the time of writing of this thesis) was awarded for work that showed that topological invariants are useful for detecting phase transitions in 2D collections of matter, a set of phenomena which was extremely difficult to quantify with a traditional dynamical systems approach.

[1] Some nice animations of algorithms for cutting and flattening surfaces can be found at `https://www.cs.cmu.edu/~kmcrane/Projects/LoopsOnSurfaces/`.

In our work, we show that a particular flavor of applied topology known as "topological data analysis" (Edelsbrunner et al. (2000), Edelsbrunner and Harer (2008)) is well-suited to quantify high dimensional point clouds such as our time-ordered point clouds, and which has found applications in signal analysis as a result. We now build up some additional algebraic tools to establish a class of topological invariants.

2.1.1 Simplicial Homology

One way of describing a topological space in a purely combinatorial way is with a *simplicial complex*:

Definition 13. *An* abstract simplicial complex *is a finite set of sets K such that $\sigma \in K$ and $\alpha \subset \sigma \implies \alpha \in K$. Each $\sigma \in K$ is known as a* simplex*, and each $\alpha \subset \sigma$ is a* face *of σ ($\sigma \supset \alpha$ is known as a* co-face *of α). The* dimension *$d(\sigma)$ of a simplex σ is the cardinality of σ minus one.*

For example, $\{\{1,2,3\},\{1,2\},\{2,3\},\{1,3\},\{1\},\{2\},\{3\}\}$ is an abstract simplicial complex, but $\{\{1,2\},\{1\}\}$ is not ($\{2\} \in \{1,2\}$ but $\{2\} \notin K$). The one-dimensional simplices can be viewed as points, the two dimensional simplices as edges connecting points, the 3 dimensional simplices as triangles connecting the edges, the 4 dimensional simplices as tetrahedra, etc. The requirement that $Y \subset X \implies Y \in S$ says that if a simplex exists in the simplicial complex, then all of its faces must also be part of the collection. Another way of defining a simplicial complexis as a "hypergraph" which is closed under the face relation.

We can compute a topological invariant known as homology on simplicial complexes, but we need a few more definitions first. All of the definitions can be found in Hatcher (2002) and are re-iterated here briefly for convenience.

Definition 14. *A p-*chain *c is a formal sum of p-simplices with coefficients in some field $\mathbb{F}$*

$$c = \sum_i a_i \sigma_i \tag{2.3}$$

where $a_i \in \mathbb{F}$. The set of all p-chains over a set of p-simplices in a simplicial complex K with the addition operator under the field $\mathbb{F}$forms a group denoted $C_p(K)$

Often in practice, the field is taken to be $\mathbb{Z}_2$, or the binary field, in which addition is analogous to the XOR operation ($1 + 1 = 0$), but in Chapter 4 we show some utility for using other fields, such as $\mathbb{Z}_3$ (briefly, $\mathbb{Z}_2$ is not a strong enough invariant to detect "twists").

Now define the boundary operator ∂_p of a p-simplex, which is a map from a p simplex to a $(p-1)$-chain under the same field

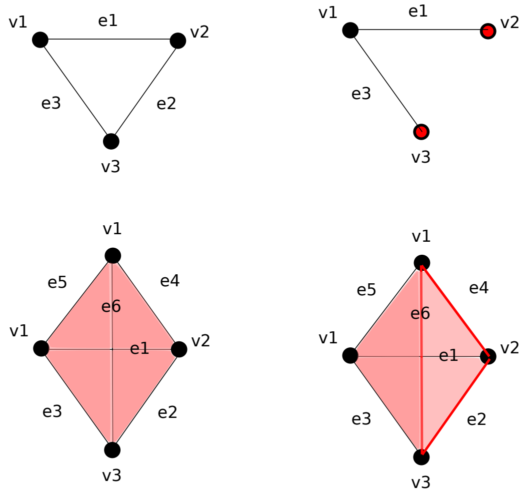


FIGURE 2.1: Examples of the 1-boundary resulting from removing an edge from a triangle (top, boundary shown with red dots), and an example of a 2-boundary resulting from removing a triangle from a tetrahedron (bottom, boundary shown with red line segments)

Definition 15. *Given a p-simplex $\sigma = \{u_0, u_1, ..., u_p\}$*

$$\partial_p \sigma = \sum_{j=0}^{p+1} (-1)^j \{u_0, u_1, ..., \hat{u_j}, ..., u_p\} \tag{2.4}$$

where $\hat{u_j}$ indicates the omission of the vertex u_j to form a $(p-1)$-simplex.

Note that for $\mathbb{Z}_2$ coefficients, the boundary operator is simply

$$\partial_p \sigma = \sum_{j=0}^{p+1} \{u_0, u_1, ..., \hat{u_j}, ..., u_p\} \tag{2.5}$$

The boundary operator extends linearly to all simplices in a p-chain c, so

$$\partial_p c = \sum_i a_i \partial_p \sigma_i \tag{2.6}$$

The group of $(p-1)$-chains in the image of the boundary operator for a group of p-chains in a simplicial complex K is itself a group, which is a subgroup of the group of $(p-1)$ chains in C. The notation for the group of p chains which result from the boundary operator of $(p+1)$-chains is denoted as $B_p(K) \subset C_p(K)$.

This definition may seem abstract, but it expresses a concept that is very intuitive, especially with binary coefficients as the field, which we will use to explain the examples in Figure 2.1. On the top of Figure 2.1, 3 1-simplices are connected to each other by edges $\{e1 = \{v1, v2\}, e2 = \{v2, v3\}, e3 = \{v1, v3\}\}$. Since every vertex shows up in exactly two edges, the boundary operator on this chain of edges gives zero (1+1 = 0 with binary coefficients)

$$\partial_1(e1+e2+e3) = (v1+v2)+(v2+v3)+(v1+v3) = (v1+v1)+(v2+v2)+(v3+v3) = 0 \tag{2.7}$$

However, when edge $e2$ is removed, a boundary is now present, as the vertices $v1$ and $v2$ show up only once after the boundary map

$$\partial_1(e1 + e3) = (v1 + v2) + (v1 + v3) = (v1 + v1) + v2 + v3 = v2 + v3 \tag{2.8}$$

On the bottom of Figure 2.1, a similar example is shown with a 2-boundary. On the bottom left, 6 edges of the tetrahedron are connected to each other by 4 triangles: $\{T1 = \{e3, e6, e5\}, T2 = \{e2, e4, e6\}, T3 = \{e1, e2, e3\}, T4 = \{e1, e5, e4\}\}$ In this case, applying the 2-boundary operator ∂_2 on the chain formed by these four triangles yields zero, because every edge is part of exactly two triangles. However, once T2, the triangle on the right, is removed, three edges show up exactly once in the new chain, and those three edges survive the boundary operator.

Now the language exists for describing what is meant by a *cycle*

Definition 16. *A p-chain c whose boundary is the zero element is called a p-*cycle*. The group of all p-cycles in a simplicial complex K is denoted as $Z_p(K)$, and is a subgroup of $C_p(K)$.*

By this definition, the three edges in the top left of Figure 2.1 form a 1-cycle, and the four triangles in the bottom right of Figure 2.1 form a 2-cycle. In other words, there are no "holes" at that dimension where a boundary could be nonzero, so everything is closed. A 1-cycle is often referred to as simply a "cycle," and a 2-cycle is referred to as a "void." Now note a very important theorem relating boundaries to cycles.

Lemma 1. The fundamental lemma of of homology *(Edelsbrunner and Harer (2010)) Given a $(p + 1)$-chain c in a simplicial complex K,*

$$\partial_p \partial_{p+1} c = 0 \tag{2.9}$$

This implies $B_p(K) \subset Z_p(k)$

In other words, every boundary of a $(p + 1)$-simplex is a p-cycle, so applying the boundary operator twice sends everything to zero. This makes sequences of p-chains with boundary operators between them a special case of a *chain complex*. One can use this property to quantify exactly how many holes of a particular dimension exist in a simplicial complex K. The idea is that a p-cycle will count as a hole as long as it is not the boundary of some $(p + 1)$ simplex in K. Intuitively, the p-boundary in a simplicial complex K are the p-simplices in K that aren't "filled in" by $(p + 1)$-simplices. To express this mathematically, the notion of a *homology group* is used

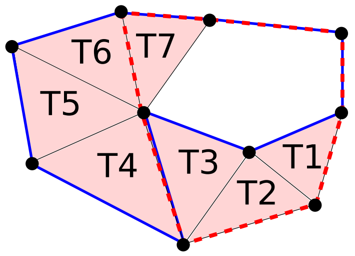


FIGURE 2.2: An example of a simplicial complex with two homologous loops (i.e. the red path is expressible as the blue path plus only boundaries of triangles). Triangles which exist in the complex are shaded in pink, and the two different paths are drawn in red and blue. The blue cycle B can be turned in the red cycle R by adding only boundaries: $R = B + \partial(T_1 + T_2 + T_3) - \partial(T_4 + T_5 + T_6)$.

Definition 17. *Given a simplicial complex, the p^{th}* homology group *$H_p(K)$ is*

$$H_p(K) = Z_p(K)/B_p(K) \tag{2.10}$$

or the group of p-cycles modulo the group of p-boundaries. The rank of the group H_k is the k^{th} Betti number *β_k.*

Thus, in order for a k-cycle to count as a hole contributing to β_k, it must not be a linear combination of the boundaries of $(k + 1)$-simplices under the chosen field. Therefore, if $\beta_k = x$; there are x linearly independent k-cycles which cannot be written as a linear combination of $(k + 1)$-boundaries. The set of cycles that are equal under the equivalence relation "equality mod the boundary $B_p(K)$" are part of an equivalence class called a *homology class*, so there are x k-homology classes if $\beta_k = x$. This elegant framework for classifying topological spaces is known as *homology.*

One attractive property that homology has that many engineering approaches lack is a certain "immunity to arbitrary choices." For instance, a computer scientist's first instinct when looking for "loops" in a simplicial complex might be to run some kind of graph traversal algorithm. But there are many loops which are *homologous*, i.e. part of the same homology class. Figure 2.2 shows an example of two different loops which could represent the same homology class. By creating these equivalence classes and simply counting the number of classes, the basis for the homology classes doesn't matter at all. In addition, it turns out that the choice of simplices used to represent a topological space does not affect the homology either. This should serve as a relief to people in graphics, for example, since there are many ways to mesh (triangulate) the same point cloud which intuitively do not change the topology. More details can be found in Hatcher (2002).

Unfortunately, the independence of basis can be a double-edged sword when one wants to actually find representative cycles of a particular homology class, or to "localize" the homology (i.e. visualize where the cycles actually reside in a simplicial complex). Even when attempting to constrain the problem to cycles of a minimum length, the problem is NP-hard to approximate within even a constant factor in general with $\mathbb{Z}_2$ coefficients (Chen and Freedman (2011)). There is hope for orientable complexes when lifting to $\mathbb{Z}$ coefficients, however (Dey et al. (2011)).

Computing Betti numbers

To compute Betti numbers in practice, it is useful to fix a basis for the simplices and explicitly write out the boundary operators from p chains to $(p-1)$ chains as matrices with coordinates. A sparse matrix reduction algorithm can then be used to find the rank of the image and kernel of the boundary matrix, which are needed to compute the Betti numbers. Below, a matrix is defined where the basis is fixed to be each p simplex (we abuse notation slightly by using the same symbol ∂ for this matrix as we do for the abstract operator without a basis)

Definition 18. *Given a simplicial complex K with N p-simplices $\{\sigma_j^p\}_{j=1}^N$ and M $(p-1)$-simplices $\{\sigma_i^{p-1}\}_{i=1}^M$, then the* boundary matrix ∂_p *is an $M \times N$ matrix defined as follows*

$$\partial_p(i,j) = \left\{ \begin{array}{cc} -1^k & \sigma_i^{p-1} = \{v_{j_1}, v_{j_2}, ..., \hat{v_{j_k}}, ..., v_{j_{p+1}}\} \subset \sigma_j^p = \{v_{j_1}, v_{j_2}, ..., v_{j_{p+1}}\} \\ 0 & \textit{otherwise} \end{array} \right\} \tag{2.11}$$

where the ordering of the vertices in σ_j is made in accordance with some global order of the vertices, so that $\partial_p(i,j)$ is consistent with Definition 15

Hence, adding a $(p-1)$ simplex to the complex adds a row to ∂_p, and adding a p-simplex adds a column. Note that these matrices are extremely sparse, since each p simplex has only $(p+1)$ co-dimension 1 faces, so the boundary matrix for a simplicial complex with N p-simplices needs $N(p+1)$ storage. Therefore, sparse matrix reduction algorithms can be used to significantly reduce storage and computation time.

Given these matrices, all Betti numbers can be computed as

$$\beta_k = \text{rank}(\text{kernel}(\partial_k)) - \text{rank}(\text{image}(\partial_{k+1})) \tag{2.12}$$

2.1.2 Vietoris-Rips Filtrations And Persistent Homology

Vietoris-Rips Complexes, Filtrations, And Persistence

Like everything in our toolset, we would like topology to be in the service of analyzing time-ordered point clouds. However, the topology of a point cloud is quite dull. A point cloud with N points consists of N connected components ($\beta_0 = N$), and otherwise trivial topology ($\beta_k = 0, k > 0$). However, we can think of point clouds as sampled from unions of manifolds/stratified spaces, which themselves can have interesting topology. For example, Figure 2.3 shows a point cloud which appears

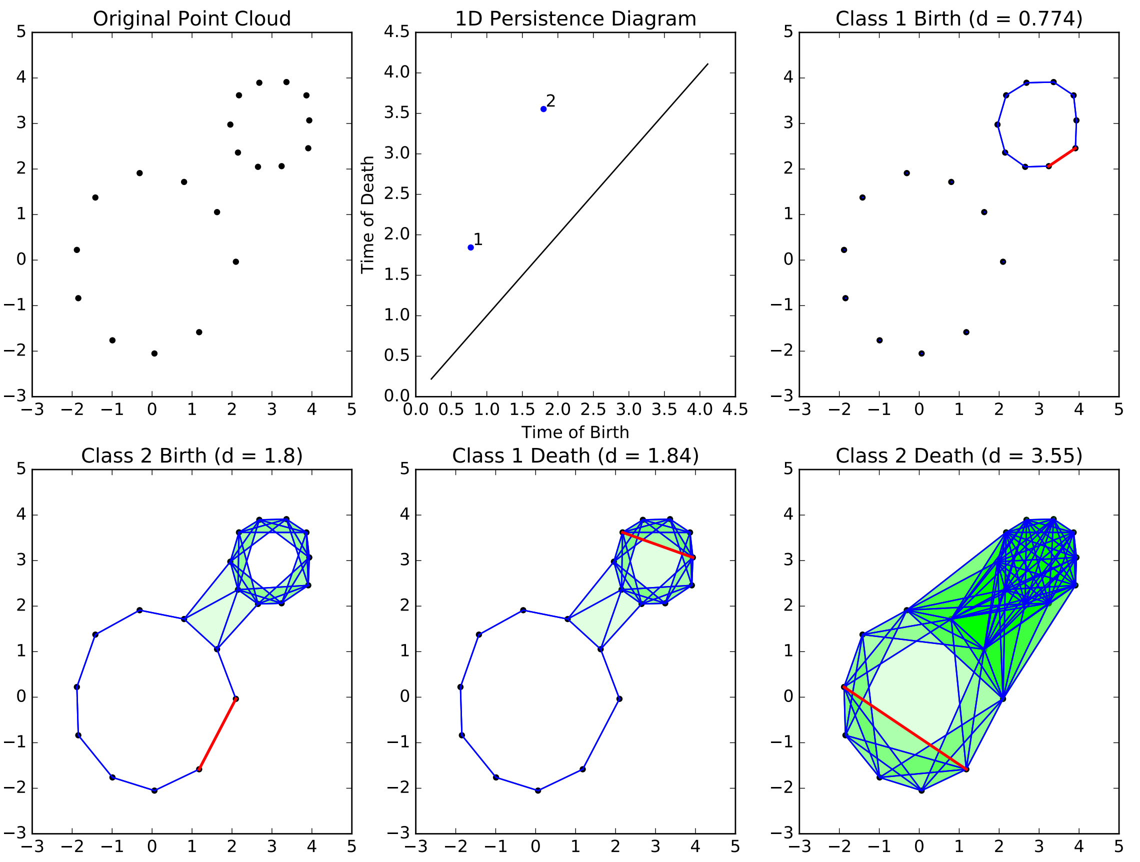


FIGURE 2.3: The evolution of simplicial complexes built on a point cloud as the Vietoris-Rips Filtration progresses, and the 1D persistence diagram summarizing the evolution of 1 dimensional homology during this progression. Connected edges are drawn in blue, and filled in triangles are drawn in green.

to have come from two different loops. But how do we quantify this, not actually knowing the underlying manifolds? The key idea is to grow metric balls around each point, and to examine the homology of the resulting space resulting of the union of the balls. Since a ball is a metric concept, this ends up being a nice combination of geometry and topology, or "multiscale topology."

To get started, we define something called a Čech Complex

Definition 19. *Given a point cloud X in a metric space $(\mathcal{M}, d)$, the* Čech Complex *is the following abstract simplicial complex:*

$$C_X(r) = \{\sigma \subset X | \bigcap_{x \in \sigma} B_r(x) \neq \varnothing\} \tag{2.13}$$

In other words, if a simplex is *covered* by a union of balls, then add it to the simplicial complex. This simplicial complex is known as a *nerve* of the topological space obtained from the union of balls of radius r around each $x \in X$, and by

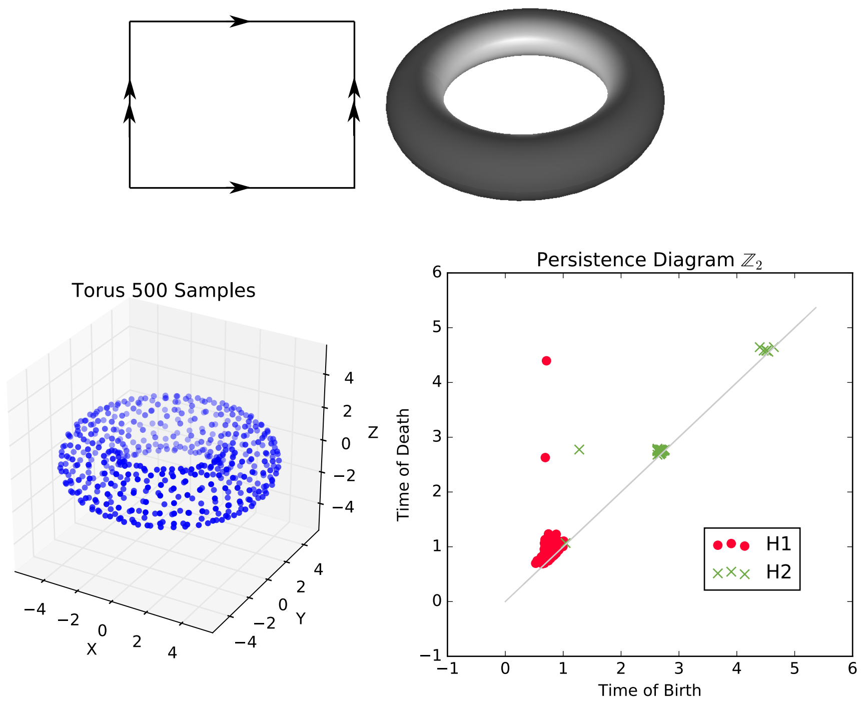


FIGURE 2.4: On the top, a diagram showing which identifications to make to form a torus. On the bottom, 500 points are sampled on the torus. Two 1 cycles and one 2 cycle are evident.

something called the "Nerve Lemma" (Steenrod and Eilenberg (1952)) it turns out to be "homotopy equivalent" to the topological space. Homotopy equivalence is a stronger invariant than homology, so the homology of the Čech Complex captures the homology of this ball union space.

One drawback of Čech Complexes is that it can be difficult to compute them in high dimensions, because checking if a simplex is covered amounts to checking if there is a ball enclosing the points that make it up, and computing minimum enclosing balls is expensive in high dimensions (and needs to be done for all possible simplices!). A substantially simpler complex is what's known as the *Vietoris-Rips Complex*

Definition 20. *Given a point cloud X in a metric space $(\mathcal{M}, d)$, the* Vietoris-Rips Complex *is the following abstract simplicial complex:*

$$V_r(X) = \{\sigma \subset X | d(x_1, x_2) \leqslant r, \forall (x_i, x_j) \in \sigma\} \tag{2.14}$$

In other words, a simplex exists in $V_X(r)$ if all of its edges have length $\leqslant r$.

Note that the Vietoris-Rips Complex is not equivalent to the Čech Complex. In fact, $C_X(r) \subset V_X(r)$, which follows readily from the definitions (Figure 2.6 shows an example). Another way of saying this is that the Vietoris-Rips Complex is an L_∞

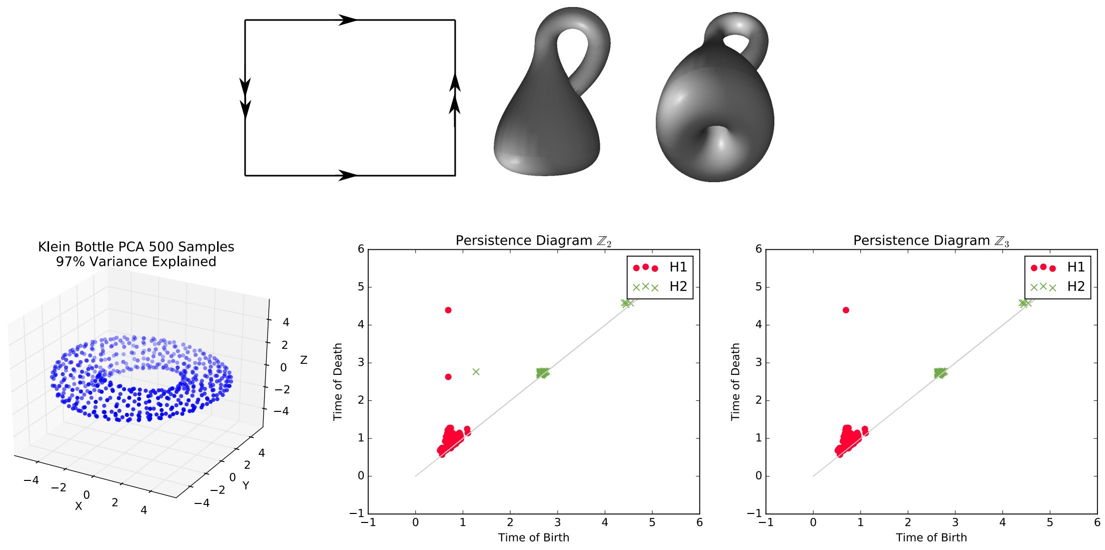


FIGURE 2.5: On the top, a diagram showing which identifications to make to form a Klein bottle (same as the torus, except one of the identifications has a twist). On the bottom, 500 points are sampled on the Klein bottle, and it is projected to 3D with PCA. When using $\mathbb{Z}_2$ coefficients, the diagram looks the same as a torus, but with $\mathbb{Z}_3$ coefficients, it changes.

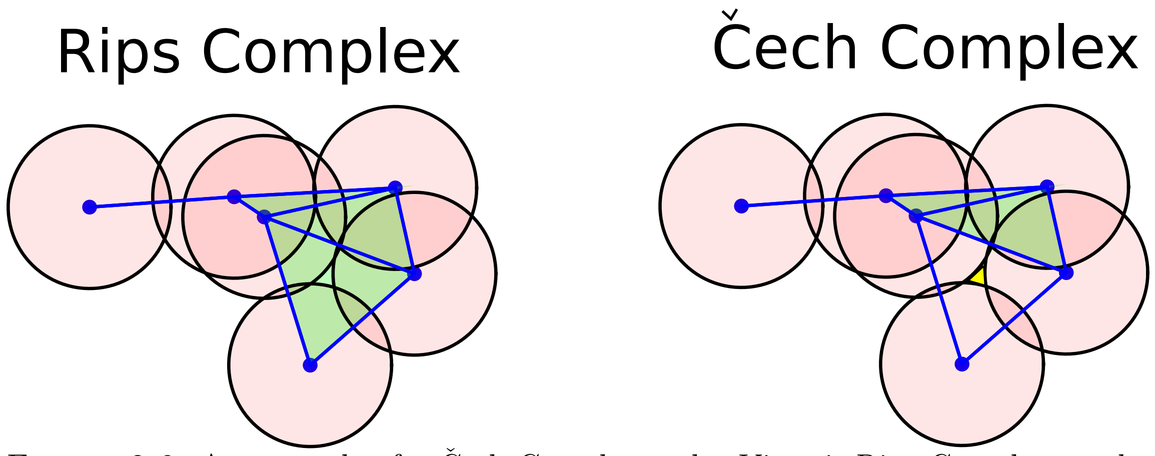


FIGURE 2.6: An example of a Čech Complex and a Vietoris-Rips Complex on the same point cloud. There is one triangle that Rips adds here which is not actually covered by balls. The uncovered portion is highlighted in yellow in the Čech Complex.

version of the Čech Complex. It is necessarily, therefore, an approximation to the geometry/topology combination we're after, but it is a good tool in practice, since it only requires knowledge of edge lengths (making it usable in general discrete metric spaces even when it's impossible to compute enclosing balls). The Vietoris-Rips Complex is also a special case of a *flag complex*, which is a simplicial complex where if a clique exists in the 1-skeleton of that complex, then the corresponding simplex exists. Because of this, in addition to avoiding ball enclosing computations, simpler data structures are possible for storing higher simplices, since only the 1-skeleton (graph) is needed (Boissonnat et al. (2016)).

Both Čech Complexes and Vietoris-Rips Complexes have the property that the complex at a smaller scale is contained in the complex at a larger scale. This makes it possible to define what's known as a "filtration"

Definition 21. *A given sequence of simplicial complexes $S_1, S_2, ...S_N$ is a* filtration *if $S_1 = \varnothing$ and $S_i \subset S_j$, $\forall i < j$*

To create a Vietoris-Rips Filtration, for example, simply come up with the sequence $V_X(0) \subset V_X(r_1) \subset ... \subset V_X(r_N)$, where r_i is the length of the i^{th} longest edge and N is the total number edges to be added. Note that the Vietoris-Rips Complex only changes when an edge is added, so we can ignore scales between edge lengths.

We are finally ready to define persistent homology. Given a filtration $S_1, S_2, ...S_N$, define all of the inclusion maps between S_i and S_{i+1}. This induces maps at the homology level, and we can study the kernel and co-kernel of these maps. In particular, if a p simplex is added at time i which increases the kernel of ∂_p, then a new linearly independent p-homology class forms. This is called a *birth event*, and this type of p-simplex is referred to as a *positive (+) p-simplex.* If, on the other hand, a p simplex is added which increases the image of ∂_p, then a $p-1$ homology class becomes homologous to zero. This is called a *death event*, and this type of p-simplex is referred to as a *negative (-) p-simplex.* We can create an object to describe these events known as a *persistence diagram.* For each homology class that exists at some time, we record the time at which it was born, and the time at which it died. The difference between these two values is known as the *persistence.* This yields a multiset of points in the plane, which can be plotted with birth on the x-axis and death on the y-axis. For technical reasons, we also include the diagonal line with infinite multiplicity, which represents all classes that born as soon as they die.

Figure 2.3 shows an example 1D persistence diagram of a Vietoris-Rips Filtration on a point cloud in $\mathbb{R}^2$. The Vietoris-Rips Complex is shown at every stage of the algorithm where either a birth or death event happens. There are two persistence dots in the image; one corresponding to the smaller loop and one corresponding to the larger loop. This makes more precise the notion that there are two loops in the point cloud at different scales. Figure 2.4 shows the 1D and 2D persistence diagrams of a sampled torus. Here, there are two dots with large persistence that stand out, implying that there are two independent 1-cycles over a large range of distances in the rips filtration, each corresponding to a principal circle of the torus. There is also a 2D cycle that rises far above the diagonal (birth = death), which implies that there is a 2D cycle, or a "void," that exists over many scales. This is a pretty convincing signature of a torus, which is made of the product of two circles and which has $\beta_1 = 2$ and $\beta_2 = 1$. This is a signature which we will search for in Chapter 4.

Figure 2.5, on the other hand, shows the persistence diagram for a Klein bottle which has been sampled in $\mathbb{R}^4$. Using $\mathbb{Z}_2$ homology, this appears the same as the torus. However, the Klein bottle does not bound a void; there is no inside or outside ($\beta_2 = 0$). As it turns out, with $\mathbb{Z}_3$ homology, it should have $\beta_1 = 1, \beta_2 = 0$, and the difference is visible in the third most persistence diagram of Figure 2.5.

The Persistence Algorithm

The homology computation given in Section 2.1.1 only works for simplicial complexes which are fixed. We now modify homology computation so that it fits into the persistence framework, where simplices are added incrementally and the ranks of the images and kernels of all of the boundary matrices can change. When a p simplex σ is added to a simplicial complex K, it adds a column to ∂_p and a row to $\partial_{p+1,p}$. The added row is all zeros because of the filtration rule that the $(p+1)$ co-faces of σ have not been added yet, so the rank of the image and kernel of $\partial_{p+1,p}$ remain unchanged. However, adding the column to ∂_p either increases the rank of the kernel of ∂_p if the column is a linear combination of the previous columns, or it increases the rank of the image of ∂_p if the column is linearly independent of the previous columns. This leads to the definition positive (+) and negative (+) p-simplices:

Definition 22. *When a p-simplex is added to a simplicial complex K and a column is added to ∂_p*

- *If the column is a linear combination of previous columns, a birth of a p-cycle happens, since the rank of the kernel of ∂_p is increased. Then by Equation 2.12, $\beta_k = \beta_k + 1$. Thus, this is a* positive *p-simplex.*
- *If the column is linearly independent of previous columns, a death of a $(p-1)$ cycle happens, since the rank of the image of ∂_p is increased. Then by Equation 2.12, $\beta_{k-1} = \beta_{k-1} - 1$. Thus, this is a* negative (-) p-simplex

To complete the modifications for the persistence algorithm, rather than choosing an arbitrary indexing set, the simplices can be ordered by their filtration function values, and the births and death times can be determined as biproducts of reducing the entire simplicial complex (an N-simplex over a set of N points in the case of a point cloud with a rips filtration). Algorithm 1 provides the details. Most of the details are very similar to the case of a static simplicial complex. When a zero column is encountered in Line 16 for a k-simplex, the kernel of ∂_k is increasing by one, so a k-cycle must be born. The most subtle new detail is on Line 25, when the rank of ∂_k increases by one, and a $(k-1)$-class is killed. Since the columns are

Algorithm 1 Algorithm for Persistence Reduction and Generators

```
1: procedure ComputePersistenceDgmsReduce($K, f$)    ▷ Input is a simplicial complexand a valid filtration $f$
2:     $N \leftarrow$ number of simplices in $K$
3:     Sort simplices by their order of occurrence in the filtration
4:     Initialize an $N \times N$ matrix M of all zeros    ▷ Full boundary matrix
5:     Initialize an $N \times N$ identity matrix K    ▷ Matrix to be co-reduced
6:     $\beta_k \leftarrow 0$ for all $k$    ▷ Initialize all Betti numbers to zero
7:     $\mathtt{I}_k \leftarrow \{\}$    ▷ List of maps for storing transient $k$-homology classes for all $k$
8:     $\mathtt{G}_k \leftarrow \{\}$    ▷ List for storing generators of homologyclasses
9:     $\sigma_i \leftarrow$ the $i^{\text{th}}$ simplex added in the filtration, for all $i = 1 : N$
10:    for $i = 1 : N$ do    ▷ Initialize the sparse matrix
11:        for $\sigma_j \in \sigma_i, |\sigma_j| = |\sigma_i| - 1$ do
12:            $\mathtt{M}(j, i) \leftarrow 1$    ▷ Add a 1 to the column $i$ of the matrix for each co-dimension one face of $\sigma_i$, at the $j^{\text{th}}$ row
13:        end for
14:    end for
15:    for $i = 1 : N$ do    ▷ Reduce the sparse matrix
16:        if $\mathtt{M}(:, i) == 0$ then    ▷ If this is a column of all zeros
17:            $k \leftarrow |\sigma_i|$
18:            $\beta_k \leftarrow \beta_k + 1$
19:            $\mathtt{I}_k\{\sigma_i\} =$NULL    ▷ A $k$-cycle is born at time $f(\sigma_i)$
20:            $\mathtt{G}_k\{\sigma_i\} = \{\sigma_j ; \mathtt{K}(j, i) == 1\}$    ▷ Read off generator from co-reduced matrix
21:        else
22:            $k \leftarrow |\sigma_i|$
23:            $\beta_{k-1} \leftarrow \beta_{k-1} - 1$
24:            $j = \text{low}(\mathtt{M}(:, i))$    ▷ Index of lowest nonzero element in column $i$ of M
25:            $\mathtt{I}_{k-1}\{\sigma_j\} = \sigma_i$    ▷ At time $f(\sigma_i)$, $\sigma_i$ kills the class born at $f(\sigma_j)$
26:            for $k = i + 1 : N$ do
27:                if $\mathtt{M}(j, k) == 1$ then
28:                    $\mathtt{M}(:, k) \leftarrow \mathtt{M}(:, k) + \mathtt{M}(:, i)$
29:                    $\mathtt{K}(:, k) \leftarrow \mathtt{K}(:, k) + \mathtt{K}(:, i)$
30:                end if
31:            end for
32:        end if
33:    end for
34: end procedure
```

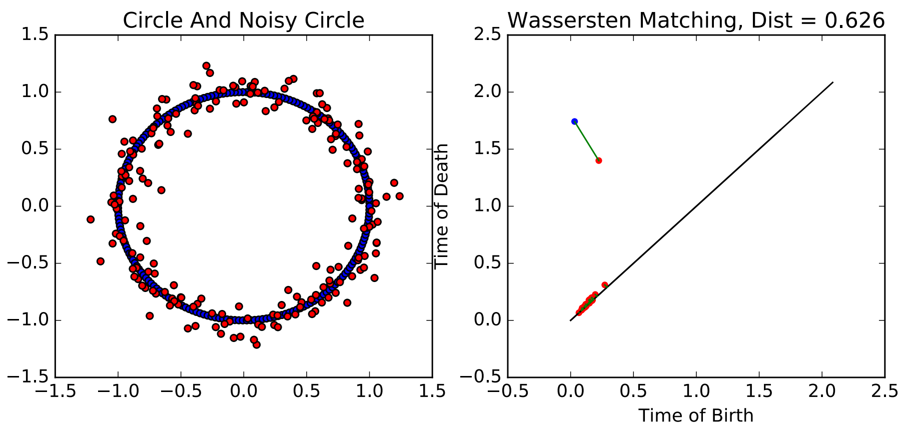


FIGURE 2.7: L_2 Wasserstein distance between persistence diagrams of a circle and a noisy circle

sorted in order of filtration time, the lowest element in that column, which is not zero (since it increases the rank), must have given birth to the $(k-1)$-class that is being killed. Thus, it is possible to pair the death times with the correct birth times in the filtration. Also, when $\mathbb{Z}_2$ coefficients are used, it is possible to store the sparse matrices so that each column is a list. With this representation, Line 29 can be implemented as an XOR list merge. Finally, at the end, the information needed to construct the persistence diagrams can be read off of the list of maps defined on Line 7.

2.1.3 Persistence Diagram Comparison And Stability

With the definition and examples of persistence diagrams now in hand, it is natural to ask how robust they are to noise and inaccuracies in the point cloud and metric. Something known as the *Bottleneck Distance* can be used to quantify this. Below are a few definitions which build up to an explanation of bottleneck distance.

Definition 23. *Given two persistence diagrams P_{S_1} and P_{S_2} on filtrations S_1 and S_2, respectively, the p-*Wasserstein Distance $d^p(P_{S_1}, P_{S_2})$ *is defined as*

$$d_W^p(P_{S_1}, P_{S_2}) = \inf_{\gamma \in \Gamma} \left(\sum_{i=1}^{|\gamma|} ||P_{S_1}(i), \gamma(P_{S_1}(i))||^p \right)^{\frac{1}{p}} \tag{2.15}$$

where Γ is the set of all perfect bipartite matchings (1 to 1 correspondences between points in $P_{S_1}(i)$ to those in $P_{S_2}(i)$), where all diagonal points are included in each with infinite multiplicity.

If $p = \infty$, this is known as the bottleneck distance *between two persistence diagram.*

The bottleneck distance has been shown to be Lipschitz-continuous with respect to the Hausdorff distance (minimum maximum distance under a correspondence) between two point sets by Cohen-Steiner et al. (2007), so the bottleneck distance is stable with respect to Hausdorff noise[2]. The same is not true of the p-Wasserstein distance in general. For instance, an arbitrarily large set of points close to the diagonal can, in aggregate, blow up the distance. However, Cohen-Steiner et al. (2010) show stability of the p-Wasserstein distance for filtrations which come from Lipschitz functions, so it is still a useful similarity measure between persistence diagrams for a large class of data in practice. It also incorporates more information than simply the max distance, which can be more discerning.

To compute the Wasserstein distance, perform a minimum weight bipartite matching $\gamma : P_{S_1} \to P_{S_2}$ between the points of the two diagrams P_{S_1} and P_{S_2}. This can be performed in $O(N^3)$ time with the Hungarian algorithm, where N is the number of points in the persistence diagram. The diagonal is included with infinite multiplicity, so if a point in $|P_{S_1}|$ does not get matched to a point in $|P_{S_2}|$ or vice versa, then it can be matched to the diagonal. This takes care of small persistence points which arise close to the diagonal when noise is added. For the bottleneck distance, there exists an $O(N^{1.5}\log(N))$ algorithm for computing the optimal matching (Kerber et al. (2016)).

Figure 2.7 shows an example point cloud and the minimum-weight bipartite matching between points for $d_W^2(P_{S_1}, P_{S_2})$. Notice how noisy points are matched to the diagonal.

Note that there has been a lot of recent work on other ways to match persistence diagrams that are stable. For instance, Bubenik (2015) creates "persistence landscapes," which turns a diagram into a set of triangles for each point whose areas are proportional to the square of persistence of that point. Reininghaus et al. (2015) create a multiscale kernel using heat diffusion around each point in the diagram. And Carrière et al. (2015) use the D2 histogram of points in a diagram. When we need to compare diagrams in Section 5.4.3, however, we simply stick to the Wasserstein distance.

2.1.4 Computational Complexity

It is often computationally infeasible to apply the naive persistence algorithm (Algorithm 1) to analyze point clouds. To see this, assume that one wants to compute Betti numbers up to β_k for a Rips filtration over a point cloud with N points. To compute β_k, all $(k+1)$-simplices need to be included. There are $\binom{N}{p+1} = O(N^{(p+i)})$ possible p-simplices to be added for each for each p, so the total number of simplices that would need to be considered is $O(N^{k+2})$, meaning the full boundary matrix would have $O(N^{k+2})$ columns. Furthermore, computing Betti numbers requires determining rank, which calls for a matrix reduction that is $O(n^3)$ for n columns. This compounds the complexity even further to $O((N^{k+2})^3) = O(N^{3k+6})$. For β_1, this

[2] Though it has been proved that the diagrams are stable to Hausdorff noise under the bottleneck distance, it should be noted that the diagrams are not in general robust to other types of additive noise, such as Gaussian or uniformly distributed random noise.

requires a worst case complexity of $O(N^9)$, and for β_2, the complexity is $O(N^{12})$. Since we need to compute persistent β_2 in Chapter 4 on point clouds with 600 points, this bound needs to be improved somehow.

In low dimensions, *alpha complexes*, or simplicial complexes that are subcomplexes of the Delaunay triangulation, can be used to substantially reduce the number of simplices that need to be checked compared to a Čech Complex (Edelsbrunner et al. (1983), Edelsbrunner (1993)). There has also been some recent work using cover trees (Beygelzimer et al. (2006)) to approximate Vietoris-Rips Filtrations (Sheehy (2013), Cavanna et al. (2015)). In this work, we use recent fast code developed by Ulrich Bauer at TUM, known as "Ripser" (Bauer (2017)). This code can compute persistent H_2 for point clouds of 600 points in a matter of seconds, which sets it apart from other currently available techniques. It can also handle field coefficients other than $\mathbb{Z}_2$, which we exploit in Section 4.3.3.

2.2 Nonlinear Time Series Analysis / Dynamical Systems

We draw a lot of inspiration from the dynamical systems community in our work. At times, it will be helpful to take an explicitly dynamical systems view, particularly in our chapter on sliding window videos (Chapter 4). Because of this, we find it necessary to review some basic tools from this field here. We also take this as an opportunity to highlight how topological data analysis is useful for describing dynamical systems, which we will show through various examples in this section. This combination forms the basis for a powerful toolset in this thesis.

2.2.1 Takens' Delay Theorem

The main theorem we will make use of is known as *Takens' Theorem* (Takens (1981)), which is tied to *discrete-time dynamical systems*

Definition 24. *A* discrete-time dynamical system *is a time-evolving process which is identified with a* state space *on a d-dimensional Riemannian manifold $\mathcal{M}$ with a state transition self-map on $\mathcal{M}$, $\psi : \mathcal{M} \to \mathcal{M}$, which is smooth.*

The state transition map ψ describes how the dynamical system progresses from its present state into the future, and so ψ is often referred to as the *dynamics* of the system. Since ψ is a smooth function determined a priori, a discrete-time dynamical system is inherently *deterministic*, though it can still yield *chaotic* trajectories through $\mathcal{M}$ (i.e trajectories which are high sensitivity to the initial state, which means they are impossible to predict long term with finite precision measurements).

Now we can state Takens' Delay Theorem

Theorem 2. *Given the two following objects:*

- *A discrete-time dynamical system on a manifold $\mathcal{M}$ with a state transition function ψ*

- *A generic twice differentiable scalar function* $f \in \mathcal{C}^2(\mathcal{M}, \mathbb{R})$

then it is a generic property that the map $\mathcal{M} \to \mathbb{R}^{2m+1}$ *given by*

$$S(x) = \begin{bmatrix} f(x) \\ f(\psi(x)) \\ \vdots \\ f(\psi^{2d}(x)) \end{bmatrix} \tag{2.16}$$

where $\psi^k(x)$ *represents composition with* ψ k *times, is an* embedding *of* $\mathcal{M}$ *(i.e. there exists a diffeomorphism between* $\mathcal{M}$ *and the image of* S*).* S *is known as the* delay reconstruction *or* delay embedding *of* f*.*

Note the similarity of this framework to the sliding window embedding established in Equation 1.1. In fact, this is theoretical justification why one cares about using a sliding window; if one wants to truly describe the state of a system geometrically without ambiguity (i.e. the injectivity of the diffeomorphism), then it is possible that up to $2d+1$ lags are needed. We will see some examples of this in Chapter 4. Also note that this theorem is basically a corollary of the *Whitney embedding theorem* (Adachi (2012)), which states that any d-dimensional Riemannian manifold can be embedded into $\mathbb{R}^{2d+1}$ (e.g. a knot, which is a 1-manifold, can be embedded in $\mathbb{R}^3$). Though Takens' Theorem shows that, rather than needing $2d + 1$ independent dimensions, it is enough to use a single scalar function to generate them all under the right conditions. Finally, a sliding window embedding is often referred to as a "phase space reconstruction" in the dynamical systems community. A interesting analysis of the historical development of phase space can be found in Nolte (2010). Basically, the name "phase space" is not suggestive of the generality of the above framework, and it took some retrospective analysis to figure out where that nomenclature started.

To see Takens' Theorem in action, it is useful to reverse engineer the theorem to get a few examples. First, let's suppose that we start with a state space which is the two sphere S^2. Let the sphere be parameterized in spherical coordinates by (θ, ϕ), so that

$$s(\theta, \phi) = (\cos(\theta)\cos(\phi), \sin(\theta)\cos(\phi), \sin(\phi)) \tag{2.17}$$

Define the following state transition map

$$\psi_{\Delta\theta,\Delta\phi}(\theta, \phi) = \left\{ \begin{array}{cc} ((\theta + \Delta\theta)\ 2\pi, \phi + \Delta\phi), & -\pi/2 < \phi + \delta\phi < \pi/2 \\ (\theta, \phi), & \text{otherwise} \end{array} \right\} \tag{2.18}$$

and define an observation function

$$f(\theta, \phi) = \boldsymbol{u} \cdot s(\theta, \phi) \tag{2.19}$$

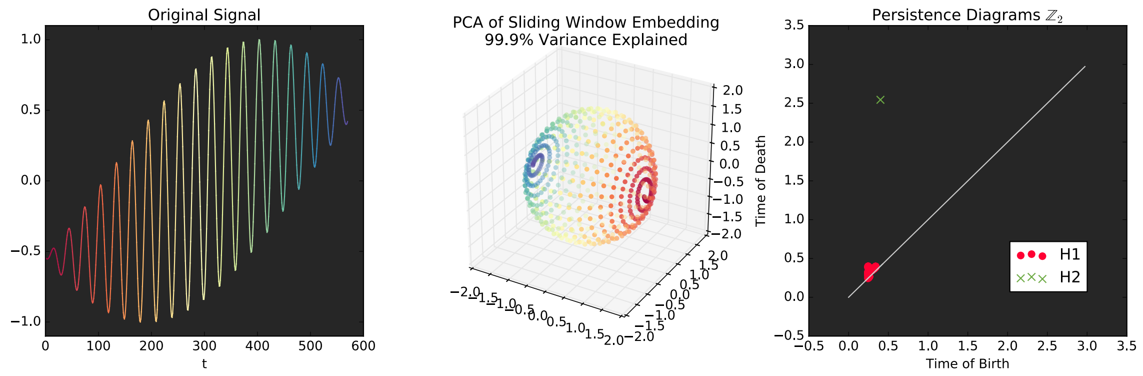


FIGURE 2.8: An example of reverse engineering Taken's delay theorem to derive a time series whose sliding window embedding lives on the sphere S^2. In this case, the observation function is the angle between the state vector on the sphere and some random vector $\boldsymbol{u}$ which isn't on the equator or poles. As suggested by PCA and confirmed by 2D persistence, this does indeed lie on a sphere

where $\boldsymbol{u} \in \mathbb{R}^3 = (a, b, c)$ is some random unit vector so that $a, b, c \neq 0$. In other words, the observation function is the cosine of the angle a state variable on the sphere makes with some vector $\boldsymbol{u}$ which isn't on the north or south pole or on the equator. Ruling out the poles and the equator keeps the function *generic*, or in this case that it observes all degrees of freedom of the state variable. Equation 2.19 can be rewritten as

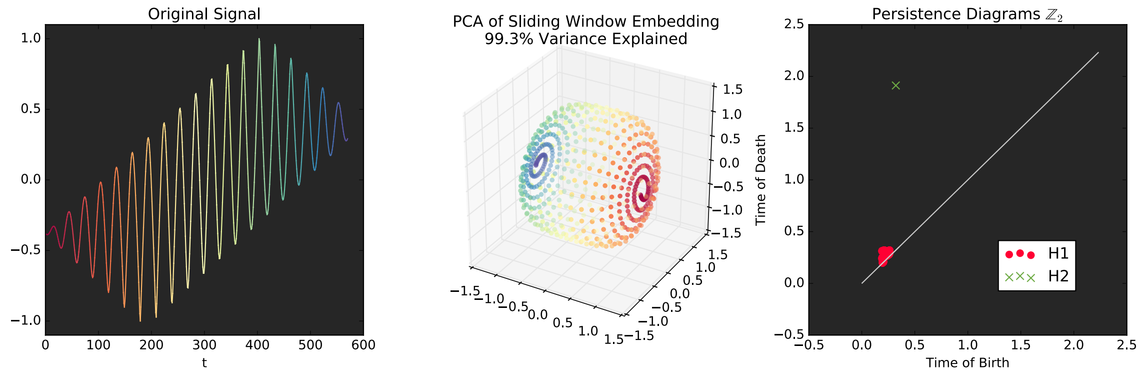


FIGURE 2.9: An example of reverse engineering Taken's delay theorem to derive a time series whose sliding window embedding lives on the sphere S^2. In this case, the observation function is the geodesic distance of the state vector on the sphere to some random vector $\boldsymbol{u}$ which isn't on the equator or poles. Like Figure 2.8, this lies on a topological sphere which is diffeomorphic to the geometric sphere, though the embedding has changed due to the use of a different observation function.

$$f(\theta, \phi) = (a\cos(\theta) + b\sin(\theta))\cos(\phi) + c\sin(\phi) \tag{2.20}$$
$$= a'\cos(\theta - b')\cos(\phi) + c\sin(\phi) \tag{2.21}$$

where $a' = \sqrt{a^2 + b^2}$ and $b' = \tan^{-1}(b/a)$. Now given a period length T and a frequency f, and taking the limit as $\Delta\theta$ and $\Delta\phi$ approach zero, we can write a continuous time series of the form

$$f(t) = A\cos(2\pi f t - \phi)\cos\left(\pi(t/T - 1/2)\right) + B\sin\left(\pi(t/T - 1/2)\right) \tag{2.22}$$

for some constants A, B, and ϕ. In other words, the signal is a phase-shifted cosine which is amplitude modulated by a cosine over the interval $[-\pi/2, \pi/2]$, which starts off at amplitude 0, is amplitude 1 halfway through, and is amplitude 0 at the end, which is added to a function which increases monotonically. As can be seen in Figure 2.8, the delay reconstruction of this signal does indeed lie on the sphere. If we tweak the observation function to instead measure geodesic distance from $\boldsymbol{u}$ (which is proportional to the angle with $\boldsymbol{u}$), we still get a signal which lies on a space that is diffeomorphic to the sphere, but it has been slightly warped (Figure 2.9).

Now we will look at examples of dynamics which live on the torus, which call for their own section.

2.2.2 Torus State Spaces: Periodicity And Quasiperiodicity

Dynamics on the $T - torus$ have a very important role in describing periodic processes. To show this, as before, we will reverse engineer Takens' Delay Theorem. This will end up yielding a geometric alternative to Fourier analysis of periodic signals which is quite useful. We will show another way to derive all of this in Section 4.1.2 which may be more intuitive for engineers, and which will also allow us to show the similarities between 1D time series and video for periodic processes, but for now we follow the same blueprint we outlined for dynamical systems on the sphere. For convenience, start with a state space of the *flat torus* (also known as a "Clifford Torus") in $\mathbb{R}^4$, which is parameterized by

$$T(\theta, \phi) = (A\cos(\theta), A\sin(\theta), B\cos(\phi), B\sin(\phi)) \tag{2.23}$$

This is a 2-torus which has no curvature and which topologically results from identifying the two pairs of opposite sides of a rectangle. It cannot be metrically realized in $\mathbb{R}^3$, although it is diffeomorphic to the standard 2-torus in $\mathbb{R}^3$. Now create the observation function similarly to Equation 2.19. That is take the dot product of some random vector $\boldsymbol{u} = (a, b, c, d)$, so that $(a, b), (c, d) \neq (0, 0)$ (this is like saying the scalar function must witness both principal circles of the torus)

$$f(\theta, \phi) = \boldsymbol{u} \cdot T(\theta, \phi) \tag{2.24}$$
$$= A' \cos(\theta - \phi_1) + B' \cos(\phi - \phi_2) \tag{2.25}$$

where $A' = A\sqrt{a^2 + b^2}, \phi_1 = \tan^{-1}(b/a), B' = B\sqrt{c^2 + d^2}, \phi_2 = \tan^{-1}(d/c)$. The dynamics can be defined as

$$\psi_{\Delta\theta,\Delta\phi}(\theta, \phi) = ((\theta + \Delta\theta) \bmod 2\pi, (\phi + \Delta\phi) \bmod 2\pi) \tag{2.26}$$

As before, we can take the limit to continuous dynamics and see that a continuous function whose sliding window embedding lives on the torus is of the form

$$f(t) = A\cos(2\pi f_1 t - \phi_1) + B\cos(2\pi f_2 t - \phi_2) \tag{2.27}$$

This can be generalized to the T-torus as

$$f(t) = \sum_{t=1}^{T} A_t \cos(2\pi f_i t - \phi_t) \tag{2.28}$$

Since there are T degrees of freedom in a T-torus, Takens' Delay Theorem would suggest that we potentially need the upper bound of $2T + 1$ lags in the delay embedding to properly reconstruct the state space. In fact, as shown by Perea and Harer (2015), we only need $2T$. Intuitively, there is a dimension required to describe both amplitude and phase of each sinusoid. More rigorously, we can write the sinusoids as complex exponentials and show that the $N \times 2k$ matrix containing all of the sliding windows contains an upper $2T \times 2T$ Vandermonde matrix, whose rank is $2k$ (Perea (2016)). This is equivalent to the statement that each sinusoid adds an elliptic trajectory which lines on a linearly independent plane.

Though signals which are sums of sinuosids all live on a torus state space, the dynamics can induce very different trajectories. For example, Figure 2.10 shows the sliding window embedding for the following harmonic 1D signal

$$f_h(t) = \cos\left(\frac{\pi}{5}t\right) + \cos\left(\frac{\pi}{15}t\right) \tag{2.29}$$

which is the sum of two sinusoids with periods 10 and 30. Though $f_h(t)$ lives in 4D on the flat 2-torus (we are viewing a 2D projection in Figure 2.10), the embedding traces and re-traces a topological loop, and the traced loop has intrinsic dimension of only 1. Figure 2.10, on the other hand, shows the sliding window embedding for the following *quasiperiodic signal*

$$f_q(t) = \cos\left(\frac{\pi}{5}t\right) + \cos\left(\frac{1}{5}t\right) \tag{2.30}$$

which is the sum of two sinusoids with periods 10 and 10π. Although the two periods in $f_q(t)$ are similar to those in $f_h(t)$, geometry of the embedding is quite

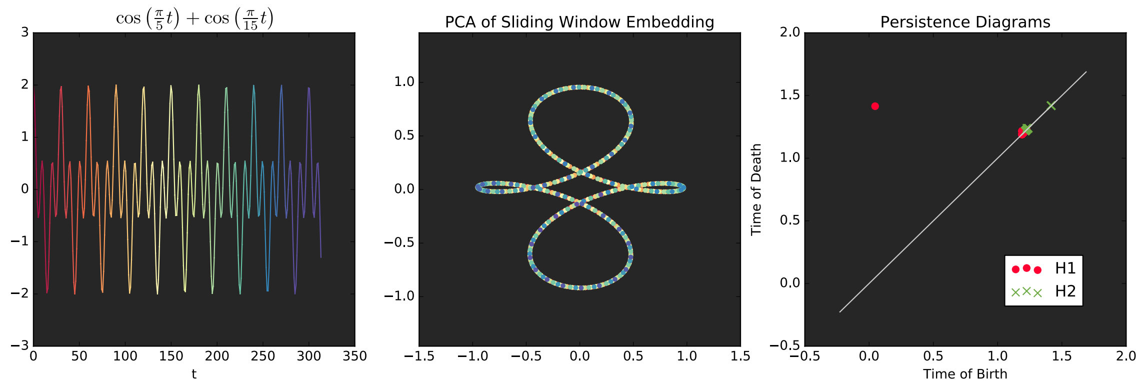


FIGURE 2.10: Sliding window embedding of a harmonic signal with a ratio of 3 between the two frequencies. Colors in the signal correspond to colors of the points in the PCA sliding window. The embedding forms a topological loop, as reflected by the single strong 1D persistent dot

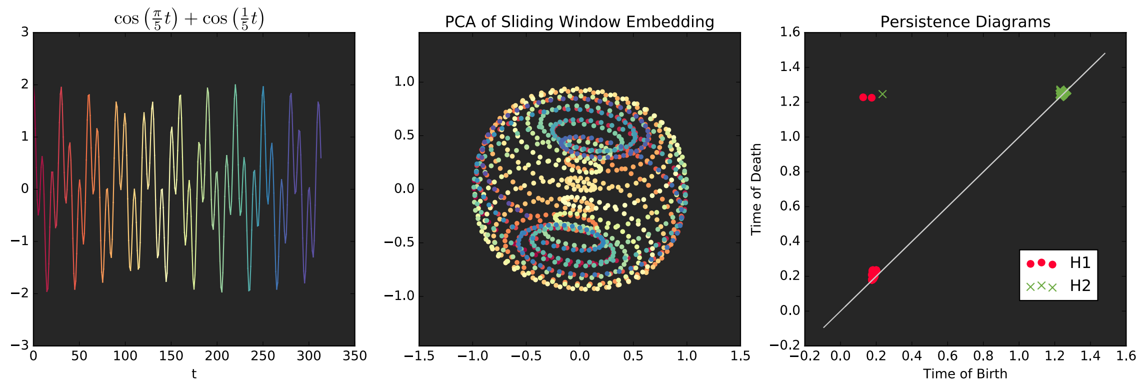


FIGURE 2.11: Sliding window embedding of a synthetic quasiperiodic signal with a ratio of π between the two frequencies. Colors in the signal correspond to colors of the points in the PCA sliding window. The embedding fills out the surface of the 2-torus, as reflected by the two strong 1D persistent dots and the single strongly persistent 2D dot.

different. The points actually fill out the entire surface of the 2-torus, since no window is ever exactly the same as any other window for all $t \in [0, \infty)$. This is an application of Kronecker's theorem (Kronecker (1884)). The difference in geometry is stark compared to the difference between traditional power spectral densities of these two signals, as shown in Figure 2.12. Because of the limits in frequency resolution with these low number of samples, it would be difficult to design an algorithm to differentiate harmonic from quasiperiodic based on the power spectrum alone. More specifically, it would be impossible to come up with a sampling rate so that all of the frequencies coincide directly with a frequency bin in the fixed FFT basis, which means that there will be sinc bleed into adjacent bins (Figure 2.12).

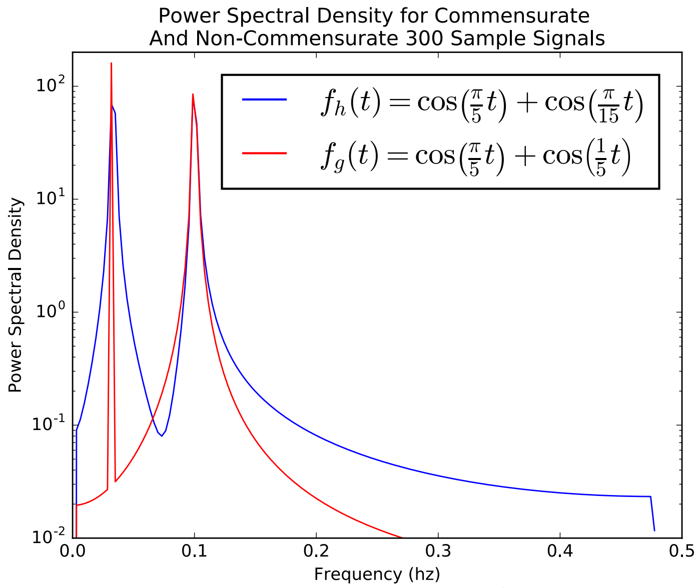


FIGURE 2.12: The power spectral densities of a commensurate and non-commensurate signal with relative harmonics at ratios 3 and π. The difference is not nearly as evident as it is with the geometry of our sliding window embeddings, and there are issue with sinc bleed for frequencies which are not commensurate with the sampling.

This toy example demonstrates one of our main contributions in this work, *quasiperiodicity detection*, and it forms the basis of our algorithm for detecting anomalies in videos of vocal cords in Section 4.6 (Figure 4.25), which have delay embeddings that behave very similarly to Figure 2.11. Note that our definition of "quasiperiodic" is different from some others in the literature (e.g. Wang et al. (2009)). While many in the engineering community define quasiperiodic as any deviation from perfect repetition, we use a different, more specific definition. In particular, we define quasiperiodic signals as those containing *non-commensurate* frequencies (i.e. frequencies that are not rational with respect to each other). This case is covered in detail theoretically by Perea (2016).

2.2.3 Connections To SSMs and Fourier Analysis

There are some connections between dynamical systems work and SSMs. Iwanski and Bradley (1998) make the case for recurrence plots on raw signals without delay. They show experimentally that the delay dimension doesn't affect recurrence plots much for certain low dimensional systems exhibiting chaos. This has been used as an argument that delay embeddings are not needed when looking at SSMs (Junejo et al. (2011)). However, we know that the delay is important for our applications with periodic and quasiperiodic signals, and we will show how SSMs and delay embeddings work in harmony in Chapter 4. It is also unclear how to tune the nearest neighbors threshold for recurrence plots, so this just trades one potentially arbitrary parameter choice (the window length) for another.

Also, though sliding window embeddings seem quite different from Fourier analysis, there are connections between the two. If a signal is perfectly periodic and the window length is equal to the period length, then PCA on sliding window embedding coincides with the Discrete Fourier Transform, as the sliding windows arranged in columns form a circulant matrix.

Finally, one might be tempted to use PCA as an alternative for techniques like false nearest neighbors (Kennel et al. (1992)) for determining the embedding dimension, but care must be taken when using singular value thresholds for determining the embedding dimension (Mees et al. (1987)).

2.3 Manifold And Metric Learning

In this section, we review some practical methods for *manifold learning* on point cloud data. As with topological data analysis, we will build structures on top of a point cloud, but the goal here is inherently geometric. We seek to find a set of lower dimensional coordinates to approximate the *intrinsic metric* of the manifold from which the points are assumed to be sampled. We will review two techniques that work off of the 1-skeleton (graph) of a discrete approximation of a manifold at some particular scale: the Graph Laplacian and Diffusion Maps. We also show an extension of diffusion maps to metric learning on multimodal data known as *Similarity Network Fusion*, which we exploit heavily in our cover songs work to fuse our new features with more classical ones. The main disadvantage of these methods compared to topological data analysis is the requirement to choose a scale a priori when the sampling density of the presumed manifold is unknown. However, we will demonstrate some "tricks of the trade" for autotuning these thresholds, which we leverage in our work.

2.3.1 The Graph Laplacian

We start by defining an important linear operator on graphs which is the discrete analogue to the Laplace Beltrami operator on manifolds (∇^2 in $\mathbb{R}^d$). It is like a "second derivative operator" for functions defined on a graph.

Definition 25. *Given an undirected graph $G = (V, E)$, with $|V| = N$ the* adjacency matrix *A is the following $N \times N$ matrix*

$$A_{ij} = \left[\begin{array}{cc} 1 & (v_i, v_j) \in E \\ 0 & otherwise \end{array} \right] \tag{2.31}$$

the degree matrix *D is a diagonal matrix which counts the number of edges*

$$D_{ii} = |e \in E s.t. v_i \in E| D_{ij,i \neq j} = 0 \tag{2.32}$$

Note that the value of D_{ii} is simply the row sum of the i^{th} row of A. Note also that the matrix A is sparse if there are only $O(N)$ edges, which is a common case in k-nearest neighbor graphs for small k, a common choice for manifold learning. Finally, note that the recurrence plot is a special case of an adjacency matrix when the vertices are endowed with a time order. Because of this, we will sometimes use the two interchangeably.

We can now define the Graph Laplacian(Chung (1997))

Definition 26. *Given a graph $G = (V, E)$ with $|V| = N$ and corresponding adjacency and degree matrices A and D, then the* unweighted graph Laplacian *is defined as*

$$L = D - A \tag{2.33}$$

Let $f \in \mathbb{R}^n$ be a scalar function defined on the vertices. Then Lf at each vertex is the sum of the differences of f at that vertex and f at its neighbors

$$\sum_{j \in N(i)} f_i - f_j \tag{2.34}$$

where $N(i)$ are the indices of the vertex neighbors of vertex i. In this way, it is like a second derivative of f[3]. Note also that given the boundary matrix ∂_1 defined using $\mathbb{Z}$ coefficients and with a basis fixed in accordance with the vertices and edges, the graph Laplacian can be written as

$$L = \partial_1 \partial_1^T \tag{2.35}$$

This is one way of showing that the graph Laplacian is symmetric positive definite, which means it has all real, positive eigenvalues. In this context, ∂_1^T is sometimes referred to as the "edge adjacency matrix," but in topological parlance this is actually the "first coboundary matrix." This connection to topology is no mathematical accident, and it is part of something much more general known as the *Hodge Laplacian* (Jiang et al. (2011)). There is a general theorem that says the kernel of the p-Hodge Laplacian is isomorphic to the p^{th} homology group H_p. In the case of L, which is the 0-Hodge Laplacian, what this says is that the rank of the kernel of L indicates the number of connected components. This is easy to verify directly from Definition 26. Beyond L, we will use the 1-Hodge Laplacian to help us go from pairwise rankings to a global ranking in Section 4.5.2.

2.3.2 Laplacian Eigenmaps / Generalized Fourier Modes

Beyond the connections of the kernel of the graph Laplacian to topology, there are some very interesting properties of the eigenvectors of this matrix, which, when

[3] If the weights are chosen non-uniformly and intelligently, this will converge to the Laplace-Beltrami operator at fine enough sampling. An example is the cotangent weights in polygon meshes in 3D Pinkall and Polthier (1993)

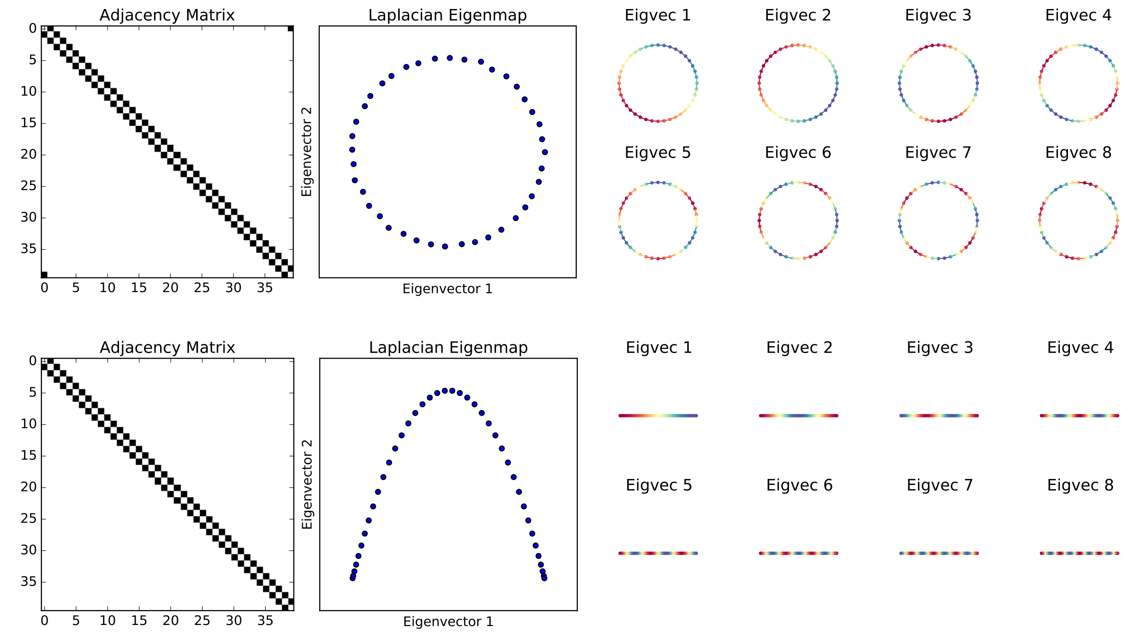


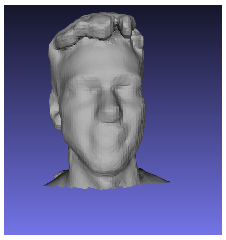
FIGURE 2.13: Examples of eigenvectors of the graph Laplacian on a circle graph and a path graph. Vertex coordiante values are plotted on the graph as colors. The eigenvectors of the circle graph recover the modes of the Discrete Fourier Transform, and they come in pairs which are out of phase by $\pi/2$.

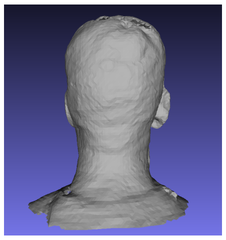
taken as a set of Euclidean coordinates, are referred to as *Laplacian Eigenmaps* (Belkin and Niyogi (2003)). A truncated set of the first few eigenvectors can be seen as a dimensionality reduction of the intrinsic geometry of the graph [4]. The second eigenvector is known as the *fielder vector*, and for connected graphs, it can be thought of as the slowest varying harmonic of the graph. It is useful, for example, for partitioning a graph into two parts (Berger et al. (2010)). In general, there is a strong analogy between the eigenvectors and Fourier decomposition. To see why, take the example of a circle graph of size N, where the adjacency matrix A^C is defined as follows:

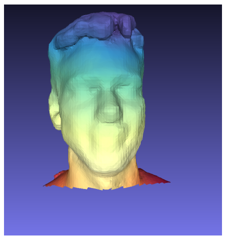
$$A^C_{ij} = \begin{bmatrix} 1 & |i-j| = 1 \\ 1 & i=1, j=N \\ 1 & i=N, j=1 \\ 0 & \text{otherwise} \end{bmatrix} \tag{2.36}$$

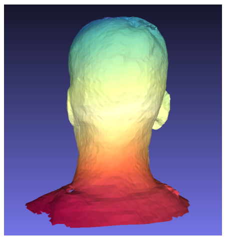
The first eigenvector is all 1s corresponding to an eigenvalue of 0 (as it would be for any connected graph), and the subsequent eigenvectors come in pairs of two (all

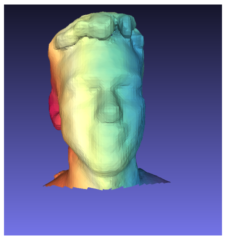
[4] When we refer to eigenvectors as occurring in an order, we are assuming that they are sorted in ascending order by their eigenvalue, so the “first nonzero eigenvector” would refer to the eigenvector corresponding to the eigenvalue with smallest magnitude

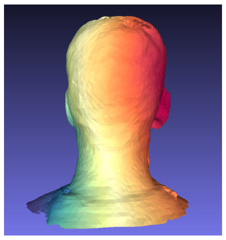

(a) Original Front

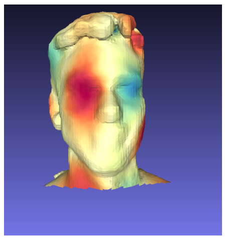

(b) Original Back

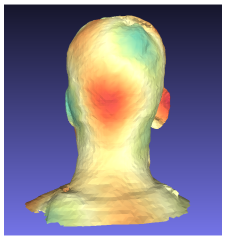

(c) Mode 1 Front

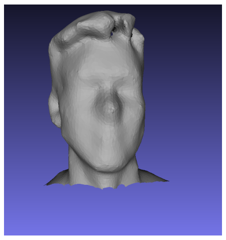

(d) Mode 1 Back


(e) Mode 2 Front


(f) Mode 2 Back


(g) Mode 33 Front


(h) Mode 33 Back


(i) 400 Modes Reconstructed Front

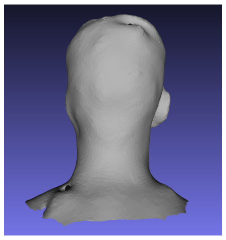

(j) 400 Modes Reconstructed Back

FIGURE 2.14: Eigenvectors of the graph Laplacian on a graph of the author's face embedded in 3D (20325 nodes, 60478 edges). The first nonzero mode captures a single oscillation along the elevation, and the second one captures an oscillation along the azimuth. As with the Discrete Fourier Transform, higher modes capture finer detail. Projecting the x, y, and z coordinates onto the first 400 modes gives a smoothed version of the geometry (at about a 20x compression ratio)

other eigenvalues have multiplicity 2)

$$x_k(i) = \sin(2\pi ki/N), y_k(i) = \cos(2\pi ki/N) \tag{2.37}$$

with corresponding eigenvalues $\lambda_{x_k} = \lambda_{y_k} = 2 - 2\cos(2\pi k/N)$. In other words, the eigenvectors of the graph Laplacian of an N-point circle *recover the Discrete Fourier Transform*. Note how higher frequencies have higher eigenvalues. This is analogous to the fact that

$$\frac{d^2}{dt^2}\cos(\omega t) = -\omega^2 \cos(t) \tag{2.38}$$

or that cosine eigenfunctions of the second derivative operator have higher eigenvalues for higher frequencies.

The top image of Figure 2.13 shows the first 8 nonzero eigenvectors of A^C with $N = 40$. In general, graphs which have symmetries, such as the circle graph, will have repeated eigenvalues. This means that the basis returned for the eigenvectors may not be numerically stable. For instance, the sine and cosine functions are arbitrarily rotated in Figure 2.13. But since L is symmetric, the eigenvectors must always be mutually orthogonal, and they will always span the same space. For this graph, this means the numerically returned eigenvectors with the same eigenvalue are always $\pi/2$ out of phase with respect to their period, so that when they are plotted against each other, they form a circle.

It is also instructive to examine the graph Laplacian of the *path graph* of N points, which is the same as the circle graph, except it does not have the edge between vertex 1 and vertex N. Though the path graph is only missing one edge compared to the circle graph, this causes it to only retain the cosine modes that descend from the quotient graph of a circle of length $2N$ since the cosine has mirror symmetry around π [5] (note that we will exploit the mirror symmetry of the cosine in a similar way in Section 4.3.1 for periodic videos in Eulerian coordinates). In particular, the eigenvectors beyond the first constant eigenvector with eigenvalue 0 are

$$v_k(i) = \cos(\pi k i/N - \pi k/2N) \tag{2.39}$$

with corresponding eigenvalues $\lambda_k = 2 - 2\cos(2\pi k/2N)$. In other words, the first eigenvector is a half cosine, the next eigenvector is a cosine, the following eigenvector is a cosine with 1.5 periods, etc. Since $\cos(2t) = 2\cos^2(t) - 1$, we see a characteristic parabolic shape when plotting the first two nonzero eigenvectors next to each other in Figure 2.13.

In addition to making quotient identifications to derive eigenvectors of graphs, we can also derive the eigenvectors of special types of product graphs (Mahadevan (2008)):

Definition 27. *Given two undirected graphs $G_1 = (U, E_1)$ and $G_2 = (V, E_2)$, the* Kronecker sum *of the two graphs is the graph $G = G_1 \bigoplus G_2$ that has the vertex set $U \times V$ and the edge set $E(n_1, n_2) = 1$ in the following two cases*

- *$n_1 = (u_i, v_j), n_2 = (u_k, v_j)$, and the edge between u_i and u_k exists in E_1*
- *$n_1 = (u_i, v_j), n_2 = (u_i, v_k)$, and the edge between v_j and v_k exists in E_2*

A remarkable fact is that the eigenvectors of the Laplacian of $G = G_1 \bigoplus G_2$ are the outer product of the eigenvectors of the Laplacians of G_1 and G_2: (Mahadevan (2008)):

[5] This is related to the concept of covering spaces in topology

Lemma 3. *If $G = G_1 \bigoplus G_2$, $L_{G_1} v = \lambda v$, and $L_{G_2} u = \mu u$, then, treating u and v as column vectors,*

$$L_{G_1 \bigoplus G_2}(vu^T) = (\lambda + \mu) vu^T \tag{2.40}$$

For instance, the *Kronecker sum* of two circle graphs is a torus graph, and this is a way to see how the Discrete Fourier Transform in higher dimensions is an outer product of 1D DFTs. This is analogous to the fact that

$$\nabla^2(\cos(\omega_1 x)\cos(\omega_2 y)) = -(\omega_1^2 + \omega_2^2)\cos(\omega_1 x)\cos(\omega_2 y) \tag{2.41}$$

which is the outer product of the eigenfunctions in Equation 2.38 with eigenvalue the sum of the two eigenvalues. The torus topology also elegantly expresses the assumption that signals are assumed to be periodic along every dimension when deriving the DFT.

Applications

In general, the Laplacian eigenvectors can be viewed as a "Fourier Transform on graphs." For instance, Figure 2.14 shows the Laplacian eigenvectors of a graph built on the author's face. This way of thinking enables a slew of applications, such as lowpass filtering / smoothing of functions on graphs (Taubin (1995), Figure 2.14 bottom), graph function compression (Karni and Gotsman (2000)), and spectral descriptors for graph classification (Reuter et al. (2006)), to name a few. In a particularly exotic application, it has even been used to come up with a basis for image texture patches that out-performs the DCT by exploiting the fact that natural image patches concentrate on the Klein bottle (Perea and Carlsson (2014)). In other words, the eigenvectors of a Klein bottle graph form a slightly better basis for image patches than Fourier-based approaches.

2.3.3 Reordering Signals with Laplacian of Sliding Windows

We now briefly develop one novel application of the graph Laplacian that ties into the dynamical systems background. As shown in Section 2.2.2, the sliding window embeddings of periodic signals form a topological loop. This implies that, at the right scale, a mutual nearest neighbor graph (Definition 1.10) built on the sliding window point cloud should be a circle graph, which means the first two nonzero eigenvectors should be a sine and cosine, or orthogonal linear combinations therein. When plotted against each other, they make an approximate circle, and the arctangent of the two eigenvector coordinates at every window can be used to determine the *phase* of the corresponding window in the periodic signal [6]. The first sample of each window can then be re-sorted by phase. Figure 2.15 shows an example of running this algorithm

[6] Note that we are only modeling the phase of a full period, rather than the phase of each harmonic, so the state space is a loop rather than a torus

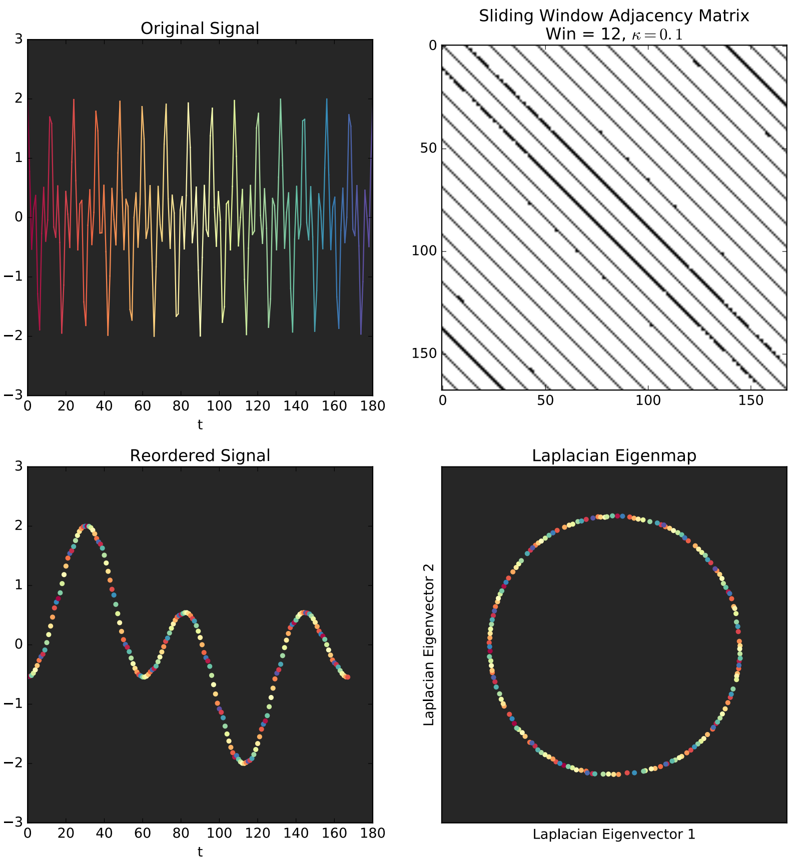


FIGURE 2.15: An example of using the first two eigenvectors of the graph Laplacian built on top of a delay embedding of a 1D signal to re-order the samples of that signal to go through exactly one period. The re-sorted points are colored according to where they occurred in the original signal. Note that the period is arbitrarily circularly shifted due to the numerical instability of the two eigenvectors with the same eigenvalue.

on a sampled signal $f(t) = \cos(t) + \cos(3t)$. There are only 12 samples per period in the original signal, so the details of each period are rather coarse. However, once they are re-sorted, we get a nice, fine-detailed representation of one period. This can be used to fake temporal super-resolution or "slow motion" for signals. Eventually, we hope to add some image processing tools to make aesthetically pleasing slow motion representations of periodic videos using this technique.

Similar tricks with the graph Laplacian have been used to re-arrange images around a loop as a pre-processing step for structure from motion (Averbuch-Elor and Cohen-Or (2015)) and to re-order the frames of microscope images a developing embryo (Dsilva et al. (2015)). A more topological approach with cohomology circular coordinates was used to parameterize a sliding window embedding of the Lorenz attractor (de Silva et al. (2012)).

2.3.4 Diffusion Maps

Now that we have developed the Graph Laplacian, we have the mathematical sophistication necessary to describe Diffusion Maps. Roughly speaking, diffusion maps are a way of defining a distance between two vertices based on random walks which initiate at each vertex, where the random walks are modeled by first order Markov chains. The transition probabilities in the Markov chains are defined by a *similarity kernel*

Definition 28. *Given a point cloud X and a distance measure ρ between points in X, we define a* similarity kernel W *as*

$$W_{ij} = \exp\left(-\frac{\rho^2(x_i, x_j)}{\sigma_{ij}}\right) \tag{2.42}$$

This is a similarity measure rather than a distance, as points which are far have a score close to 0, and points which are close have a score closer to 1. Often, σ_{ij} is set to a constant, but a smarter choice is to autotune it based on the average distance to the nearest neighbors of x_i and x_j. In particular, we can set

$$\sigma_{ij}^{\kappa} = \frac{\mu}{3}\left(\frac{1}{\kappa N}\left(\sum_{k\in N^{\kappa}(i)} \rho(x_i, x_k)\right) + \left(\frac{1}{\kappa N}\sum_{k\in N^{\kappa}(j)} \rho(x_j, x_k)\right) + \rho(x_i, x_j)\right) \tag{2.43}$$

where κ is the proportion of nearest neighbors taken (we usually choose $\kappa = 0.1$), $N^{\kappa}(i)$ refers to the κN nearest neighbors of x_i, and μ is a parameter that can be tweaked (usually in the range $[0.3, 0.8]$). This trick was used by Wang et al. (2014), for example. Of course, kernels other than the Gaussian are possible, but we find this works well in practice for our applications.

Once a similarity kernel W has been established, we can construct a Markov chain by normalizing each row sum of W to be 1. That is, define the Markov probability matrix P as

$$P_{ij} = \frac{W_{ij}}{\sum_{k=1}^{N} W_{ik}} \tag{2.44}$$

If we create a diagonal matrix holding the row sums of P, this can be written as

$$P = D^{-1}W \tag{2.45}$$

The matrix exponential P_{ij}^{t} then gives the probabilities of starting at x_i and ending at x_j after t steps. This is not yet a metric, however, as P is not even symmetric. To ameliorate this, define the following distance:

Definition 29. *Given a Markov transition matrix P, the* diffusion distance *at time t is defined as the following distance between x_i and x_j*

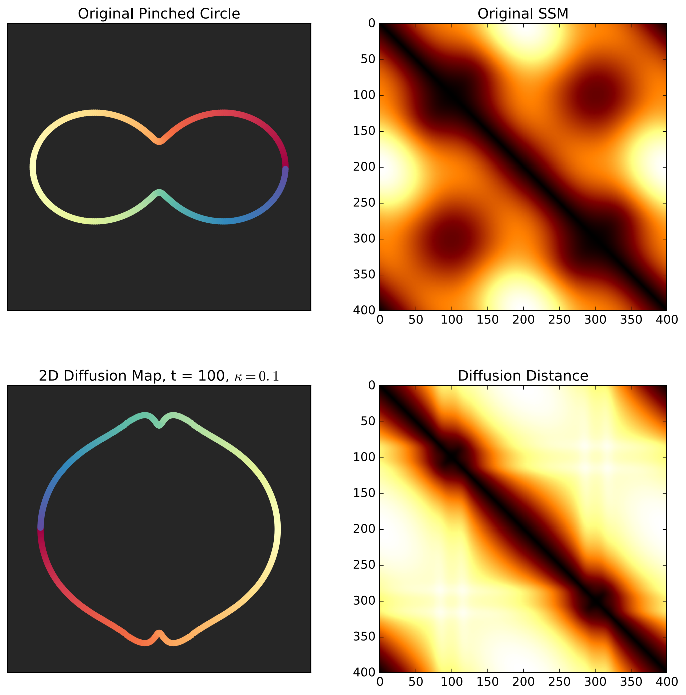


FIGURE 2.16: An example of applying diffusion maps to a pinched circle. The result is a point cloud which is closer to a rounder circle, as evidenced by the first two coordinates of the diffusion map and the SSM.

$$D_t(x_i, x_j) = \sum_{x_k \in X} (P^t_{ik} - P^t_{jk})^2 \tag{2.46}$$

In other words, the diffusion distance $D_t(x_i, x_j)$ is the squared Euclidean distance between rows i and j of P^t. Since it is a Euclidean distance, it is a metric. The diffusion distance will be low if there are many vertices which have a high probability of being reached from both x_i and x_j after t steps, or high otherwise. Alternatively, it can be visualized as the squared L_2 distance between two bumps that have diffused from delta functions at x_i and x_j after t time steps.

Unlike Laplacian eigenmaps, the diffusion distance in the form of Definition 29 does not have a natural way of decomposing into low and high frequency coordinates for dimension reduction / smoothing. To obtain such a decomposition, we need to once again do an eigendecomposition, but this time of P. The trouble is that P is not a symmetric matrix[7]. However, it can be shown (De la Porte et al. (2008)) that the eigenvalues of the "symmetric normalized transition matrix" P'

$$P' = D^{1/2} P D^{-1/2} = U \Lambda U^T \tag{2.47}$$

[7] Note that there are many version of Markov transition matrices, many of which are symmetric, but we prefer the non-symmetric one given in this section with neighborhood sizes autotuned mutually between two points, as it makes more sense in the presence of outliers.

are equal to the eigenvalues $\lambda_1, \lambda_2, ..., \lambda_N$ of P, that the right eigenvectors of P are given by the columns of $D^{-1/2}U$, and that the left eigenvectors of P are given by the rows of $U^T D^{1/2}$. For ease of explanation, we assume for the moment that $P = P'$, which can be easily generalized to the non-symmetric case with the above observations. Let the eigenvectors of P' be v_k with corresponding eigenvalues λ_k. Then the diffusion distance can be written as

$$D_t(x_i, x_j)^2 = \sum_k \lambda_k^{2t} ||v_k[i] - v_k[j]||_2^2 \tag{2.48}$$

In other words, the diffusion maps can be realized by the following Euclidean vector with squared distances

$$x[i] = \begin{bmatrix} \lambda_1 v_1[i] \\ \lambda_2 v_2[i] \\ \vdots \\ \lambda_k v_k[i] \end{bmatrix} \tag{2.49}$$

Assuming the eigenvalues are sorted in *decreasing order* now (as contrasted with the graph laplacian), a truncated approximation can be obtained by taking the first $l < k$ vectors. Hence, diffusion maps can now be used as a dimension reduction technique.

Figure 2.16 shows an example of applying diffusion maps to a pinched circle. The first 2 diffusion coordinates are plotted ($l = 2$). This map has the desired effect of lessening the pinch effect, since most random walks would go around the half circle rather than jumping across the pinch points. It is for this reason that Bendich et al. (2011) used diffusion maps as a pre-processing step before applying Vietoris-Rips Complexes on point clouds, since 1D persistence is higher for rounder point clouds.

Advantages over Graph Laplacian

The symmetric normalized Markov matrix can also be used to derive weights in a Laplacian eigenmap, so the two have similar spectral decompositions. The main advantage of diffusion maps lies in its robustness to topological noise. As demonstrated in Figure 2.13, the addition of a single edge can have a drastic effect on the Laplacian eigenmodes. By contrast, diffusion maps average over many random walks, so this effect will be less pronounced. Some nice demonstrations of these differences in nonrigid 3D shape quantification are given by Bronstein et al. (2009).

2.3.5 Similarity Network Fusion

Diffusion is useful beyond factoring out the extrinsic geometry with a single similarity kernel. Joint diffusion can also be used to fuse the results of two or more different kernels on the same point cloud to come up with a better fused diffusion map, assuming the kernels capture complementary similarity information. This process, introduced by Wang et al. (2012) with a followup by Wang et al. (2014), is known as *similarity network fusion*. It is a type of metric learning which has proved to be very successful when applied to fuse the results of different Smith Waterman schemes in cover song identification, for instance (Chen et al. (2017)). We also show its utility in fusing MFCC and HPCP features for cover song identification.

We now describe this process in more detail. First, starting with distance functions ρ_1 and ρ_2, create kernels W_1 and W_2 with the neighborhood autotuned exponential of Equation 2.43. Now, create the self-similarity normalized Markov transition matrices

$$P(i,j) = \left\{ \begin{array}{cc} \frac{1}{2}\frac{W(i,j)}{\sum_{k \neq i} W(i,k)} & j \neq i \\ 1/2 & \text{otherwise} \end{array} \right\} \tag{2.50}$$

This is very similar to the markov chains defined in Section 2.3.4, but with a regularized diagonal to promote self-recurrence. Once this matrix has been obtained, we create a truncated k-nearest neighbor version of this matrix

$$S(i,j) = \left\{ \begin{array}{cc} \frac{W(i,j)}{\sum_{k \in N(i)} W(i,k)} & j \in N(i) \\ 0 & \text{otherwise} \end{array} \right\} \tag{2.51}$$

where $N(i)$ are the k nearest neighbors of vertex i, for some chosen k. Now we are ready to describe the algorithm, which updates the Markov transition probability matrices by a joint multiplication process to represent cross-diffusion (as opposed to ordinary matrix multiplication for ordinary diffusion). Let $P^1_{t=0} = P^1$, $P^2_{t=0} = P_2$. Then the joint multiplications each step are as follows

$$P^1_{t+1} = S_1 P^2_t (S^1)^T \tag{2.52}$$

$$P^2_{t+1} = S_2 P^1_t (S^2)^T \tag{2.53}$$

In other words, a random walk is occurring but with probabilities that are modulated by two similarity kernels. This process can be generalized to more than two similarity kernels to m similarity kernels by the following update rule

$$P^v_{t+1} = S^v \left(\frac{\sum_{k \neq v} P^k_t}{m-1} \right) (S^V)^T \tag{2.54}$$

As shown by Wang et al. (2012), this process will eventually converge, but we can cut it off early. Whenever it stops, the final fused transition probabilities are

$$\hat{P}_t = \frac{1}{m} \sum_{k=1}^{M} P^k_t \tag{2.55}$$

Figure 2.17 shows a synthetic example of similarity network fusion on 3 matrices for a dataset with 120 points, using 5 nearest neighbors and 100 iterations. Though

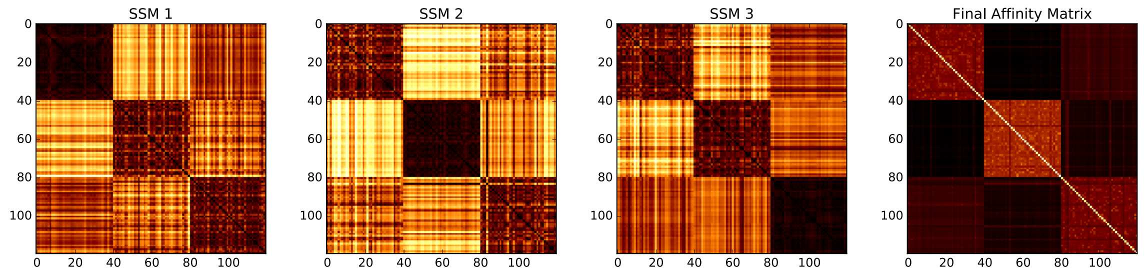


FIGURE 2.17: An example of similarity network fusion on 3 different SSMs. There are 3 clusters in this synthetic dataset. Each SSM measures one of the clusters with reliability, while the other two are noisy. The final affinity matrix from similarity network fusion, on the other hand, sees all three clusters clearly and distinctly separated from the other clusters.

in Chapter 3, we find 3 iterations is enough to substantially improve results with certain feature choices.

2.4 Sequence Alignment

Non-uniform time warping is a major theme in this work. In music, what happens when two people are singing the same passage, but one singer accentuates the ritardando more than another, dragging one section along? Similarly, what happens when two different people make the same complex gesture, but one person does a part of it faster than the other person? Naive algorithms that match sequences point by point with a p-norm, such as Euclidean distance, are no longer appropriate, as the sequences are misaligned in these cases. In this section, we review some algorithms for dealing with this problem in practice. We first start by reviewing basic edit distance algorithms for string matching under deletion/addition/substitution, and we show that a similar dynamic programming trick of "optimal subsequence alignment" has been used to align any time series metric spaces which have real-valued distances. Note that "dynamic" is an overloaded word in this thesis; it either refers to dynamical systems or in this case (and in the case of Chapter 5) to a particular strategy in algorithm design that breaks problems down intelligently into sub-problems whose results can be re-used.

2.4.1 Levenshtein Distance And Variants

We first start by considering the problem of transforming one character string into another with an optimal sequence of insertions, deletions, or letter swapping. More formally, define the following notation and operations on strings

Definition 30. *Let a* string *s be an ordered sequence of characters $s_1 s_2 s_3 ... s_N$ from an alphabet Σ. Let $s_{i:j}$ be the* substring *of s starting at character i and ending* *at character $j-1$, or the empty string if $j \leqslant i$. Let plus (+) represent string concatenation. Define the following operations on a string*

- Insertion*:* $I_k(s,a) = s_{1:k} + a + s_{k:n}$
- Deletion*:* $D_k(s) = s_{1:k} + s_{k+1:n}$
- Substitution*:* $S_k(s,a) = s_{1:k} + a + s_{k+1:n}$

Given the above operations, the Levenshtein Distance (Levenshtein (1966)) is defined as follows

Definition 31. *Given two strings s_1 and s_2, the* Levenshtein Distance *between s_1 and s_2, denoted $LD(s_1, s_2)$, is the minimum number of operations needed to transform s_1 and s_2 into the same string*

For instance, to transform “school” into “fools,” an optimal sequence of cost 4 of operations on the first string would be

1. _chool [Deletion]
2. _hool [Deletion]
3. *f*ool [Substitution]
4. fool*s* [Addition]

A non-optimal sequence of cost 5 would be

1. _chool [Deletion]
2. _hool [Deletion]
3. _ool [Deletion]
4. *f*ool [Addition]
5. fool*s* [Addition]

The Levenshtein Distance is a type of “edit distance” between strings. The symmetry is inherent, since to transform s_2 into s_1, simply reverse the operations. But the Levenshtein Distance also satisfies all of the properties of a metric space, as shown by Waterman (1995).

Now that the distance has been defined, the question remains of how to calculate it. Naively trying all possible alignments would be quite costly, since, as shown by Waterman (1995), the number of possible alignments between two strings of length n approaches $(1+\sqrt{2})^{2n+1}/\sqrt{n}$ as n approaches infinity. To design a tractable algorithm, first one must make a very clever observation, which leads to a dynamic programming algorithm. This is well-known and surveyed, for example, by Waterman (1995), but we repeat it here in our own style.

Theorem 4. *Given a string a of length M and a string b of length N, define $LD_{i,j}(a,b)$, $i \leqslant M$ and $j \leqslant N$, as the Levenshtein Distance between the substrings $a_{1:i+1}$ and $b_{1:j+1}$. Also define the following boundary conditions:*

1. $LD_{0,0}(a,b) = 0$

2. $LD_{i,0}(a,b) = i$ *(matching a blank string to string a)*

3. $LD_{0,j}(a,b) = j$ *(matching a blank string to string b)*

then the following recurrence holds:

$$LD_{i,j}(a,b) = \min \left\{ \begin{array}{ccc} LD_{i-1,j}(a,b) & +1 & \textit{Deleting character } a_i \\ LD_{i-1,j-1}(a,b) & +(1-\delta(a_i,b_j)) & \textit{Matching } a_i \textit{ and } b_j \\ LD_{i,j-1}(a,b) & +1 & \textit{Deleting character } b_j \end{array} \right\} \tag{2.56}$$

for $1 \leqslant i \leqslant M$*,* $1 \leqslant j \leqslant N$*, where* δ *is the Kronecker delta function*

To prove this, note that all characters have to be matched somewhere in the alignment or deleted. This means that the last step of an optimal alignment between $a_{1:i}$ and $b_{1:j}$ can do one of three things

1. Delete a_i from a (case 1 in Theorem 4). This operation has a cost of 1, and the rest of the optimal alignment comes from aligning $a_{1:i-1}$ to $b_{1:j}$ at a cost of $LD_{i-1,j}$
2. Delete b_j from b (case 3 in Theorem 4). This operation has a cost of 1, and the rest of the optimal alignment comes from aligning $a_{1:i}$ to $b_{1:j-1}$ at a cost of $LD_{i,j-1}$
3. Matching a_i and b_j. If $a_i = b_j$, then this incurs no cost. Otherwise, it has a cost of 1 for a swapping operation of either $S_i(a, b_j)$ or $S_j(b, a_i)$. These two cases are encapsulated by the Kronecker delta in case 2 of Theorem 4. Then the rest of the optimal alignment comes from aligning $a_{1:i-1}$ to $b_{1:j-1}$ at a cost of $LD_{i-1,j-1}$

By induction, this leads to a simple and efficient dynamic programming algorithm with time complexity $O(MN)$ for computing the Levenshtein Distance between the entire sequences a and b, as shown in Algorithm 2. The base cases are aligning a blank to a blank $D[0,0]$, aligning a blank a to substrings in b (the top row of D), and aligning a blank b to substrings in a (the leftmost column of D). Then, the inductive step has to be applied in an order so that the items to the left, above, and left/above are filled in first. A simple "raster scan" algorithm can be used to fill in this matrix, going row by row from left to right, starting at the topmost row. This order ensures that $D[i-1,j]$, $D[i,j-1]$, and $D[i-1,j-1]$ are always computed before $D[i,j]$. There are other valid orders which are more convenient for parallel processing, as will be explained more in Section 5.3.5.

Once the algorithm has completed, the optimal distance is always in the bottom right most entry of D: $D[M,N]$, since this represents the optimal cost of aligning

Algorithm 2 Levenshtein Distance dynamic programming Algorithm

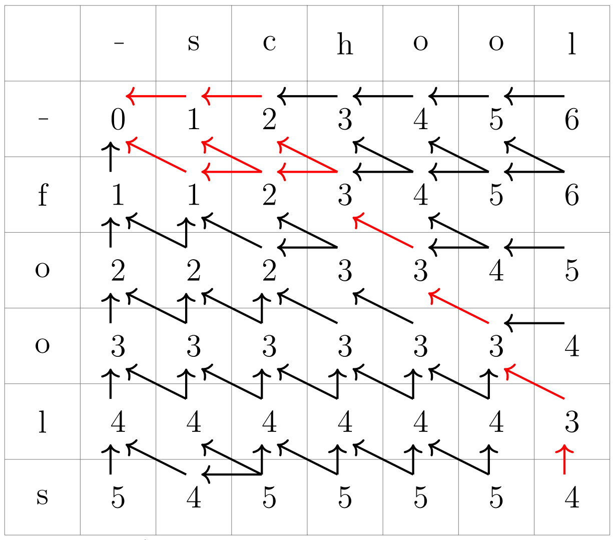
```
1: procedure LEVENSHTEINDISTANCE($a, b$)    ▷ Strings $a$ and $b$, $|a| = M, |b| = N$
2:     D ← $\underbrace{\left.\begin{array}{|c|c|c|c|c|}\hline 0 & 1 & 2 & \dots & N \\ \hline 1 & 0 & 0 & \dots & 0 \\ \hline 2 & 0 & 0 & \dots & 0 \\ \hline \vdots & \vdots & \vdots & \dots & \vdots \\ \hline M & 0 & 0 & \dots & 0 \\ \hline \end{array}\right\}M+1}_{N+1}$    ▷ Zero-Indexed 2D array
3:     for $i = 1 : M$ do
4:         for $j = 1 : N$ do
5:             $D[i, j] \leftarrow \min\{D[i-1, j-1] + \delta(a_i, b_j), D[i-1, j] + 1, D[i, j-1] + 1\}$
6:         end for
7:     end for
8: return D[M, N]
9: end procedure
```

both full strings, by definition. A sequence of operations yielding the optimal cost can also be recovered by keeping track of pointers from node (i, j) to all nodes $(i-1, j)$, $(i, j-1)$, and $(i-1, j-1)$ which tie for the optimal score $D[i, j]$ when their respective operation is taken. These pointers yield a DAG over all of the nodes of D, and this DAG can be traced backwards from $D[M, N]$ to $D[0, 0]$. Note that it is not actually necessary to use $O(MN)$ storage, as only the previously processed row ever needs to be stored at any given time during the raster scan, and while back-tracing the pointers can be re-computed on the fly. But for clarity in our examples, we visualize the entire table.

An example of this algorithm having filled in D and the DAG arrows for the traceback is shown in Figure 2.18. As this example shows, there is not always a unique set of operations that can transform the strings into each other with optimal cost. In other words, there are multiple paths through the DAG from node (M, N) to node $(0, 0)$. When we first presented this example, we showed one alignment sequence which started by deleting “s” and “c” from “school” and then swapping “h” with “f.” But as Figure 2.18 shows, it is also possible to first swap “s” with “f”, then delete “c” and “h” and end up at the same place with optimal cost. And there is yet a third optimal alignment deleting “s,” swapping “c” with “f,” and deleting “h.”

Needleman-Wunsch

There is an easy way to generalize the dynamic programming algorithm used to solve the Levenshtein Distance to accommodate non-unit costs and costs which depend on the symbols being added/deleted/substituted. This leads to what’s known as the *Needleman-Wunsch Distance* (Needleman and Wunsch (1970)). The goal is to again



	-	s	c	h	o	o	l
-	0	1	2	3	4	5	6
f	1	1	2	3	4	5	6
o	2	2	2	3	3	4	5
o	3	3	3	3	3	3	4
l	4	4	4	4	4	4	3
s	5	4	5	5	5	5	4

FIGURE 2.18: An example of the dynamic programming table that results by computing the Levenshtein Distance between “school” and “fools.” Back-pointers are shown to depict which alignments (addition/deletion/insertion) yield the best local alignment. Red arrows are drawn along optimal alignment paths in the resulting DAG.

find an optimal sequence of additions/deletions/substitutions/matches, but the costs change as follows:

- There is an addition/deletion cost which is negative (usually -1)
- There is a substitution cost $c(a, b)$ between characters a and b, which is always negative, but which can vary depending on a and b. Herein lies the strength of this algorithm; it can charge more for certain swaps than others, which is useful in DNA processing where certain transcription errors between base pairs are more likely.
- There is a new matching “cost” which is positive (usually +1) for characters which match

The algorithm for solving for the Needleman-Wunsch distance is exactly the same as the dynamic programming algorithm for Levenshtein Distance, except we are now trying to find the sequence of operations that *maximizes* the alignment cost. In this way, we think of it as a “score” rather than a cost. We could keep the weights like they were with the Levenshtein Distance, but then the optimal cost may be negative, which is aesthetically strange compared to a high score. This also gives a natural way to implement local alignment, as shown in the next section.

2.4.2 Smith Waterman Sub-Sequence Alignment

The Levenshtein Distance and the Needleman-Wunsch distance both require matching entire sequences to each other, but this may not be appropriate if the two signals have beginnings or endings which are very different. This is particularly important in cover songs, where one song could have an intro or an outro not present in the other, but they otherwise match very well. To tackle this, there is an algorithm for computing local alignments known as *Smith Waterman* (Smith and Waterman (1981)). Like Needleman-Wunsch, it seems to maximize an alignment *score*, but the alignment does not have to start on the first character and end on the last character. To solve this, the exact same dynamic programming algorithm can be used, except there is one extra "restart" condition if a local alignment has become sufficiently poor. The recurrence is as follows

$$SW_{ij} = \max \left\{ \begin{array}{c} SW_{i-1,j-1} + m(a,b) \\ SW_{i-1,j} + g \\ SW_{i,j-1} + g \\ 0 \end{array} \right\}, SW_{0,j} = SW_{i,0} = 0 \tag{2.57}$$

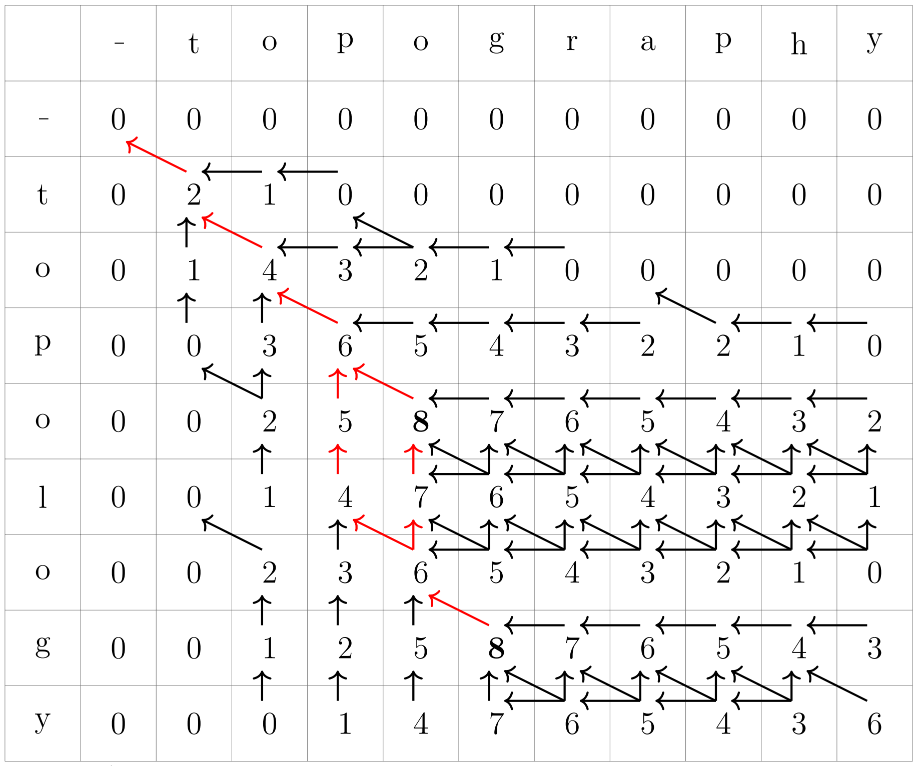
where $m(a,b)$ is a score of matching or mismatching, and g is a gap penalty. This would be exactly the same as Needleman-Wunsch were it not for the 0 in the fourth row. This is precisely what allows the alignment to restart locally.

Figure 2.19 shows the result of aligning the word "topology" to the word "topograph"[8] with alignment score +2, misalignment cost −2, and gap penalty −1. One optimal local alignment is between the first four characters "topo." Another local alignment with equal cost, obtained two different ways, is "topog."

2.4.3 Dynamic Time Warping

We now extend alignment concepts beyond strings with fixed addition/deletion/swapping costs to time-ordered point clouds in real-valued metric spaces, with an algorithm called "Dynamic Time Warping (DTW)." This is a fancy sounding name for "aligning time-ordered point clouds in the same metric space." It is quite popular in the audio processing and music information retrieval communities, as it was invented to align spoken word patterns (Sakoe and Chiba (1970), Sakoe and Chiba (1978)). It has also made its way into many other applications, including general time series (Berndt and Clifford (1994)), gesture recognition (ten Holt et al. (2007)), touch screen authentication (De Luca et al. (2012)), and video contour shape sequence alignment (Maurel and Sapiro (2003)), to name a few of the thousands of works that use it. Before we define it, we need some preliminary definitions. To start, we define something like a correspondence between two time series, but for convenience of computation, and for a more direct analogy with string matching algorithms, we don't want to allow backtracking along either sequence. That is, once a point x in the first

[8] These two words are often confused with each other by lay people



	-	t	o	p	o	g	r	a	p	h	y
-	0	0	0	0	0	0	0	0	0	0	0
t	0	2	1	0	0	0	0	0	0	0	0
o	0	1	4	3	2	1	0	0	0	0	0
p	0	0	3	6	5	4	3	2	2	1	0
o	0	0	2	5	**8**	7	6	5	4	3	2
l	0	0	1	4	7	6	5	4	3	2	1
o	0	0	2	3	6	5	4	3	2	1	0
g	0	0	1	2	5	**8**	7	6	5	4	3
y	0	0	0	1	4	7	6	5	4	3	6

FIGURE 2.19: An example of the dynamic programming table that results by computing Smith Waterman between "topology" and "topography," using a matching cost of +2, a non-matching cost of −2, and a gap cost of −1. Back-pointers are shown to depict which alignments (addition/deletion/insertion) yield the best local alignment, and the cells corresponding to the ends of these alignments are bolded. Red arrows are drawn along optimal alignment paths in the resulting DAG.

sequence matches to a point y in the second sequence, a point after x must match to a point after y. More formally, we want a special type of "ordered correspondence," called a *warping path* in the DTW literature, between two sequences.

Definition 32. *Let X and Y be two sets whose elements are adorned with a time order: $X = \{x_1, x_2, ..., x_M\}$ and $Y = \{y_1, y_2, ..., y_N\}$. A* warping path *between X and Y is a correspondence $\mathcal{W}$ satisfying the following two properties*

- Monotonicity*: If $(x_i, y_j) \in \mathcal{W}$, then $(x_k, y_l) \notin \mathcal{W}$ for $k < i$, $l > j$*
- Boundary Conditions*: $(x_1, y_1), (x_M, y_N) \in \mathcal{W}$*

This ends up leading to the same allowable alignments as in Levenshtein Distance, with the additional constraint that the first points and last points of the sequences must be in correspondence. Given this definition, DTW is defined as

Definition 33. *Let X and Y be two time-ordered point clouds indexed by the corresponding time-ordered sets I and J, respectively, in a metric space with distances d.*

The DTW Dissimilarity *between X and Y is*

$$DTW(X, Y) = \min_{\mathcal{W} \in \Omega} \sum_{(i,j) \in \mathcal{W}} d(x_i, y_j)$$

where Ω is the set of all valid warping paths between I and J.

Although DTW has a very similar flavor to Levenshtein Distance, it does not induce a metric on sequences. For a counter-example inspired by Müller (2007), consider a metric space on 3 elements a, b, c, where $d(x, y) = \delta(x, y)$. Now consider the following element sequences

- $X = abb$
- $Y = abc$
- $Z = accc$

Then DTW(X, Y) = 1, DTW(Y, Z) = 1, and DTW(X, Z) = 3. Thus, DTW(X, Z) = $3 > 2 =$ DTW(X, Y) + DTW(Y, Z), violating the triangle inequality.

In spite of this, DTW is still useful as a measure of dissimilarity. One can use an algorithm very similar to that of Levenshtein Distance to compute it, which follows from a similar inductive observation:

Theorem 5. *Given two time-ordered point clouds X and Y with M and N points, respectively, in a metric space with distances d, define $DTW_{i,j}(X, Y)$ as the DTW Distance between the first i points of X and the first j points of Y. Also define the following boundary conditions:*

1. $DTW_{0,0}(X, Y) = 0$
2. $DTW_{i,0} = DTW_{0,j}(X, Y) = \infty$

then the following recurrence holds

$$DTW_{i,j} = d(X_i, Y_j) + \min \left\{ \begin{array}{c} DTW_{i-1,j}(X, Y) \\ DTW_{i,j-1}(X, Y) \\ DTW_{i-1,j-1}(X, Y) \end{array} \right\}, 1 \leqslant i \leqslant M, 1 \leqslant j \leqslant N \quad (2.58)$$

DTW must match the final points of X and Y, which explains the presence of $d(X_i, Y_i)$ in Equation 2.58. Also, defining the boundary conditions to be ∞ everywhere except $(0, 0)$ forces the first points in each sequence to be in correspondence $((1, 1))$.

As Algorithm 3 shows, DTW is computed in a manner extremely similar to Levenshtein Distance, and as with the Levenshtein Distance, it is possible to backtrace to find the optimal warping path. The subtle difference in this algorithm is in the

Algorithm 3 Dynamic Time Warping Algorithm

```
1: procedure DTW($X, Y, d$)  ▷ TOPCs $X$ and $Y$ with $M$ and $N$ points, metric $d$
2:     D ← $\underbrace{\begin{array}{|c|c|c|c|c|}\hline 0 & \infty & \infty & \dots & \infty \\ \hline \infty & 0 & 0 & \dots & 0 \\ \hline \infty & 0 & 0 & \dots & 0 \\ \hline \vdots & \vdots & \vdots & \dots & \vdots \\ \hline \infty & 0 & 0 & \dots & 0 \\ \hline \end{array}}_{N+1} \left.\vphantom{\begin{array}{c}0\\0\\0\\0\\0\end{array}}\right\} M+1$    ▷ Zero-Indexed 2D array
3:     for $i = 1 : M$ do
4:         for $j = 1 : N$ do
5:             $D[i, j] \leftarrow d(X_i, Y_i) + \min\{D[i-1, j-1], D[i-1, j], D[i, j-1]\}$
6:         end for
7:     end for
8: return D[M, N]
9: end procedure
```

interpretation of backpointer arrows in the diagram. With Levenshtein Distance, the arrows represented operations, which can include additions/deletions. But with DTW, there are only matchings, and the arrows simply connect matched pairs which occur in sequence. Also, there is the added condition that the first elements and last elements of each sequence, respectively, must be matched.

Figure 2.20 shows an example of DTW between two 2D time-ordered point clouds using the Euclidean metric in $\mathbb{R}^2$. Figure 2.21 shows an example of DTW between two 1D time series using the absolute distance on $\mathbb{R}$, which is a common application of DTW to 1D time series. From these examples, it is clear that what is really happening is a shortest path is being computed from the upper left to the lower right of the CSM between two time-ordered point clouds. Using this observation, some choose to implement *fast marching* techniques, a series of techniques for computing solutions to discretized partial differential equations (in this case the inhomogeous wave equation), as an alternative algorithm to find the minimum cost path from lower left to upper right (Sprechmann et al. (2013)). This has the advantage of leveraging highly optimized algorithms for fast marching that have been developed in the applied PDEs community.

Efficient Approximations

Though the basic dynamic programming algorithm for DTW is quite simple, it has been notoriously difficult to beat the quadratic time complexity of that algorithm. Intuitively, a cross-similarity matrix between two time-ordered point clouds of length M and N takes up $O(MN)$ space, so a scheme which breaks the quadratic barrier has to intelligently rule out ever computing some of the pairwise distances for general time-ordered point clouds. A few months ago to the time of publication of

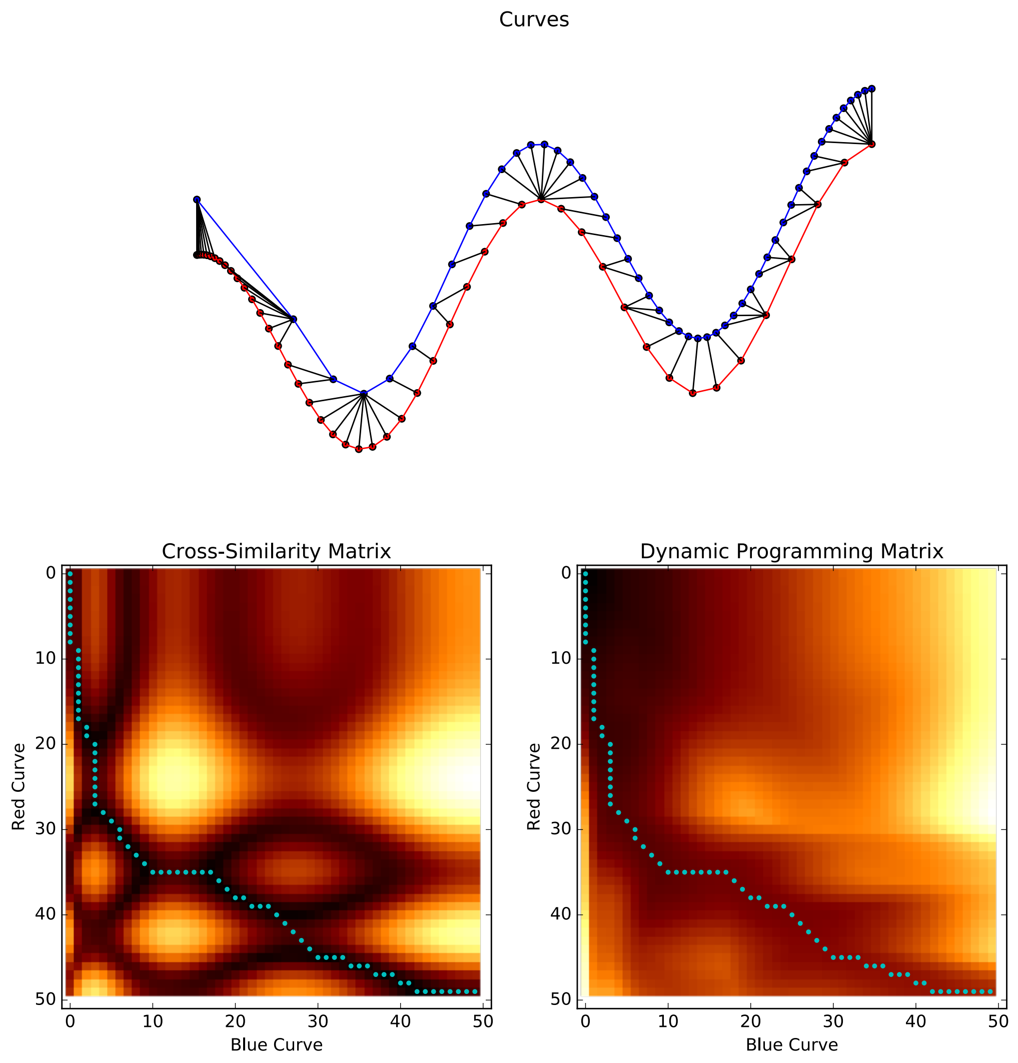


FIGURE 2.20: An example of DTW between two time-ordered point clouds in $\mathbb{R}^2$ using the Euclidean distance. The optimal warping path is sketched in cyan in the cross-similarity matrix and the dynamic programming matrix.

this thesis, a group of researchers finally broke this bound theoretically with an $O(N^2 \log\log\log(N)/\log\log(N))$ time algorithm for sequences in $\mathbb{R}$ (Gold and Sharir (2016)). However, most of the practical work has come in approximation algorithms, which we focus on for the rest of this section.

One strategy for reducing the computational cost is to use global constraints on allowed warping paths. For instance, one can limit the maximum and minimum slope of allowed warping paths to be T and $1/T$, respectively, for some constant T. This narrows down the search region to a parallelogram, since regions outside will have portions with slopes that exceed this bound. This is known as the *Itakura Parallelogram* (Itakura (1975)). Minimizing the slope is a natural idea in beat-

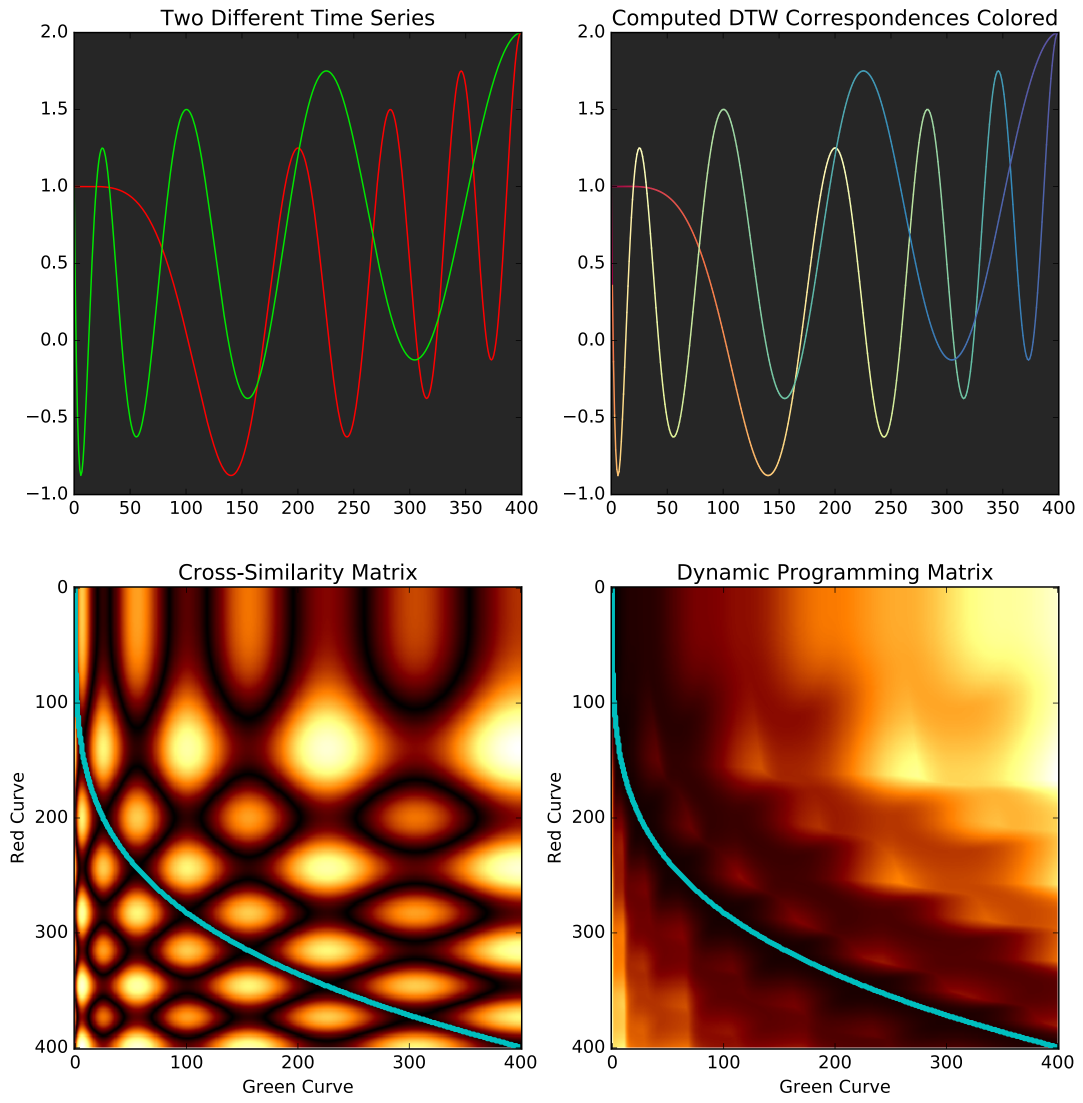


FIGURE 2.21: An example of DTW between two 1D time series, using the absolute height difference as the distance between two points. Even though these time series look quite different, they are exactly the same (up to discretization) after a re-parameterization. The optimal warping path is sketched in cyan in the cross-similarity matrix and the dynamic programming matrix in the bottom two plots. In the top right, points which are in correspondence based on this path are colored the same.

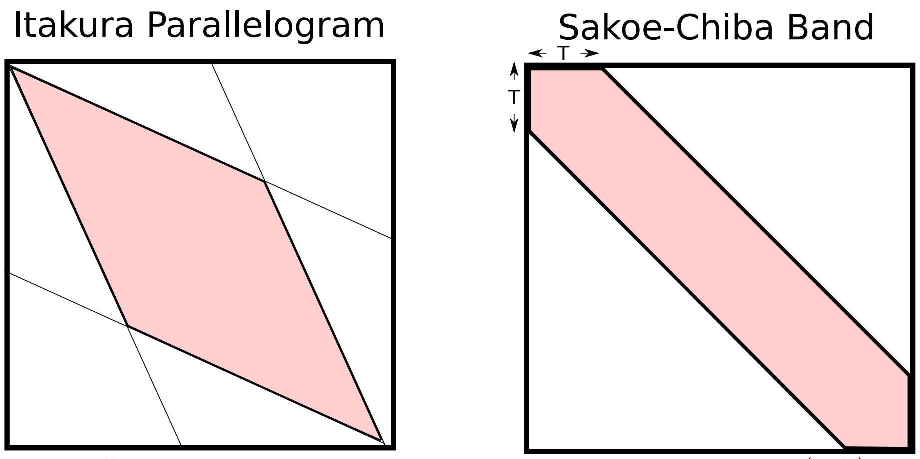


FIGURE 2.22: Global constraints on DTW: Itakura parallelogram (left) and Sakoe-Chiba Band (right). The narrowed down search region is highlighted in a salmon color in each case.

synchronous music processing, since we don't want too many beats to be skipped in a row in one or the other song, and we use this idea in our cover songs application (Section 3.2.3). A similar idea is to assume that a point x_i maps to a point y_j if and only if $|i - j| < T$; that is, the sequences are only allowed to drift T indices away from each other. In this case, the number of cells needed to check is narrowed down to $O(NT)$, so it can lead to a linear algorithm for small enough T. This is known as the *Sakoe-Chiba Band* (Sakoe and Chiba (1978)). Figure 2.22 shows both of these constraints pictorially. Unfortunately, it is quite likely for the optimal path to leave these regions, which can render the results useless in practice, so care should be taken when applying global constraints.

Recently, Ying et al. (2016) devised a clever ϵ-approximation scheme for global unconstrained DTW that exploits the geometric configuration of natural curves to approximate certain pairwise distances in the CSM, thereby avoiding all pairwise distance computations. The idea is to partition the CSM into disjoint rectangles with weights, so that for each rectangle $R = [i_1, i_2] \times [j_1, j_2]$ with weight w_R

$$|d(X_i, Y_j) - w_R| \leqslant \frac{\epsilon}{2} w_R, \forall (i, j) \in R \tag{2.59}$$

If we then create a new approximate CSM where all of the entries in R are replaced with w_R, then this is like saying the section of X from X_{i1} to X_{i2} is approximately the same distance from the section of Y from Y_{j1} to Y_{j2} (up to a proportion of the distance), so they can be approximated by some constant distance w_R. This makes it so that the DTW values only need to be computed on the boundary of the rectangles, since optimal cost paths which cut through constant regions can be done in constant time. Note that similar tricks are used in fast multipole solutions of the N-body problem (Coifman et al. (1993)).

Figure 2.23 shows an example of this algorithm. Ying et al. (2016) show that if the

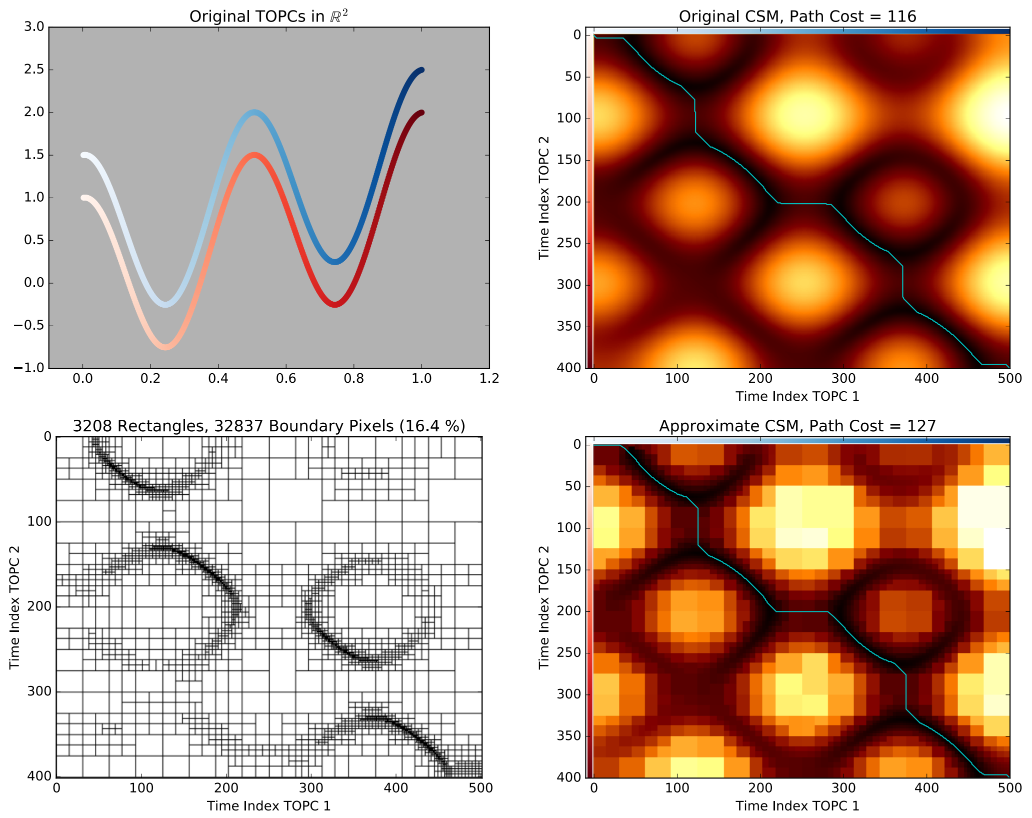


FIGURE 2.23: An example of approximate DTW between two time-ordered point clouds in $\mathbb{R}^2$, as computed by Ying et al. (2016), for $\epsilon = 0.5$. The optimal warping paths are highlighted in cyan for both the full and approximate cases. Aside from a few subtle deviations, the paths are nearly identical, and the computed cost is certainly within $\epsilon = 0.5$. Also note how for further distances, there is more ϵ slack, so there tend to be larger rectangles covering these regions.

two time-ordered point clouds are sampled from κ-packed curves; that is, curves with arc length at most κr in every ball of radius r. For a κ-packed curve with spread σ (ratio of maximum to minimum value in the CSM), the running time of the algorithm is $O(\frac{\kappa^2}{\epsilon} n \log \sigma)$. The authors also devised an $O(\frac{\kappa}{\epsilon} n \log n)$ algorithm (Agarwal et al. (2016)) that uses the well-separated pairs decomposition (Callahan and Kosaraju (1995)) with KD-trees to come up with (possibly overlapping) rectangles, though they note this scheme has a larger runtime constant that makes it slower than the former scheme for moderately sized inputs ($\approx 10^5$ to 10^6 points).

Finally, Zhou and De la Torre (2016) devised a rather exotic technique that expresses a warping path as a positive sum of dictionary warping paths which are themselves valid warping paths (e.g. log, exponential, hyperbolic tangent), and they solve for the coefficients of the dictionary jointly with the cost of the path. This is a nonconvex problem, but they approximate it with quadratic programming and show fast convergence in practice.

2.4.4 Fréchet Distance

There is another distance which is worth mentioning known as the *Fréchet Distance*, which is actually a metric between curves

Definition 34. *Given two space curves $\gamma_1 : [0,1] \to (\mathcal{M}, d), \gamma_2 : [0,1] \to (\mathcal{M}, d)$, where $\mathcal{M}$ is some metric space with a distance d, the* Fréchet Distance *$\mathcal{F}$ between γ_1 and γ_2 is*

$$F = \inf_{f,g} \sup_{x \in [0,1]} d(\gamma_1(f(x)), \gamma_2(g(x))) \tag{2.60}$$

where f and g range over all orientation-preserving homeomorphisms from the unit interval to itself. Alternatively, depending on convenience, it can be defined with just a single orientation-preserving homeomorphism h (i.e. only warping a single curve to match the other):

$$\mathcal{F} = \inf_{h} \sup_{x \in [0,1]} d(\gamma_1(x), \gamma_2(h(x))) \tag{2.61}$$

The Fréchet Distance is often referred to colloquially as the “dog-walking distance.” It can be thought of as the minimum length of a leash that’s needed for a human owner to remain attached to his/her dog as the human and dog walk forward from start to finish along pre-determined paths at different rates. Homeomorphisms from the unit interval to itself are like the continuous version of time-ordered correspondences. An infimum is taken over homeomorphisms to allow in the limit for maps which don’t inject, or in plain English, “the human and the dog can remain still at points along their paths.” It was actually used to initialize the approximate DTW algorithm described in the previous section (Agarwal et al. (2016)).

Alt and Godau (1995) discovered an algorithm which computes the Fréchet Distance between two piecewise linear curves of length M and N respectively in time $O(NM \log(NM))$. What’s interesting about this algorithm is that it is able to compute this distance over continuously parameterized piecewise linear curves using a continuous version of a CSM known as a “free space diagram.” This is in contrast to DTW, which only works at discrete vertices. There is also a variant of Fréchet Distance more similar to DTW called *Discrete Fréchet Distance* (Eiter and Mannila (1994)). This algorithm is an approximation of the full Fréchet Distance between two piecewise linear curves, hopping from vertex to vertex and ignoring the segments in between, which makes it computationally more similar to DTW. The algorithm for Discrete Fréchet Distance and DTW are nearly identical, and Discrete Fréchet Distance can also be computed in $O(NM)$ time.

2.5 Music Signal Processing

This section will provide a brief overview of features from audio analysis which aggressively reduce the dimensionality of raw audio while still retaining important perceptual features.

2.5.1 Timbral Features

We first describe what are known as "timbral features," which are essentially shape features on top of the STFT meant to summarize relative information across the spectrum (Tzanetakis and Cook (2002)). Before proceeding, a slight modification to the STFT for dealing with real signals, known as the "spectrogram," is needed

Definition 35. *Given a discrete signal $x[n]$ of length N and its Discrete Fourier Transform $X[k]$, the* spectrogram *$S[k]$ contains the first M coefficients of $X[k]$, where*

$$M = \left\{ \begin{array}{cc} N/2+1 & N even \\ (N-1)/2+1 & N odd \end{array} \right\} \tag{2.62}$$

The spectrogram subset contains all of the information needed to reconstruct a real signal, since the upper half of the spectrum is redundant for real signals. Let $\mathcal{S}(k,t)$ be a $M \times F$ matrix storing the magnitude of the coefficients of the Short-Time Spectrogram, where t is the frame number, k is the spectral sample number, M is the number of spectrogram coefficients, and F is the number of STFT frames. Then the following features describe timbre at a snapshot in time

1. Spectral Centroid: The center of mass of the Short Time Fourier Transform at a particular frame t:

   $$C_t = \frac{\sum_{k=1}^{M} k\mathcal{S}(k,t)}{\sum_{k=1}^{M} \mathcal{S}(k,t)}$$

   High-pitched electronic songs, for instance, would be expected to have a higher spectral centroid than low-frequency trombone jazz songs.

2. Spectral Roloff: A kind of spectral shape descriptor which is the frequency index to the left of which 85% of the spectrum's magnitude is concentrated

   $$\sum_{k=1}^{R_t} \mathcal{S}(k,t) = 0.85 \sum_{k=1}^{M} \mathcal{S}(k,t)$$

   solving for the roloff R_t

3. Spectral Flux: The L2 norm of the difference between successive spectrogram frames.

$$F_t = \sum_{k=1}^{M} (\mathcal{S}(k,t) - \mathcal{S}(k,t-1))^2$$

   This measures how the spectrum is changing in time (electronic music with dynamic instrumentation would be expected to change a lot more in a short amount of time than low-key smooth jazz or classical music)

4. Low Energy Feature: The percentage of STFT frames which have an RMS energy less than the average RMS energy over all frames.

5. Zero Crossings: This is the only timbral feature not based on the STFT, but which has been found to perform very well in practice. It is a time domain measure of high-frequency content of a sound sample.

$$Z_t = \frac{1}{2}|\text{sign}(x[n]) - \text{sign}(x[n-1])|$$

   This feature is a very good discriminator between vowels (small Z_t) and consonants (large Z_t), for example.

To ensure that the statistics are stationary, the spectrogram frames are usually taken over very small intervals of about 30 milliseconds. For longer blocks, some authors simply compute all of the 30 millisecond features within that interval and replace them with the mean/average over the entire interval (Tzanetakis and Cook (2002)), though in Chapter 3, we show that increasing the window size has its own advantages.

2.5.2 Mel-Frequency Cepstral Coefficients (MFCCs)

Timbral features are extremely simple, but slightly more sophisticated techniques are needed for truly good results in genre classification (Tzanetakis and Cook (2002)) and artist classification (Ellis (2007b)), for instance. The first group of these features relies on a classical transformation known as "Mel-Frequency Cepstral Coefficients" (MFCC) (Bogert et al. (1963)), which are also designed to pick up on aspects of timbre using the STFT. Given a spectrogram for a sound sequence $\mathcal{S}(k,t)$, as defined in the previous section, the MFCC is calculated as follows

1. Multiply each frame of $\mathcal{S}(:,t)$ by the "Mel-Filterbank" to yield a new matrix $\mathcal{M}(n,t)$, where $\mathcal{M}$ is a $T \times M$ matrix; T is the number of filters in the filterbank and M is the number of frames in the spectrogram. The mel-filterbank is a series of T overlapping triangular windows which are exponentially spaced in frequency to model the fact that humans perceive pitch logarithmically. The shape of the filterbank is shown in Figure 2.24.

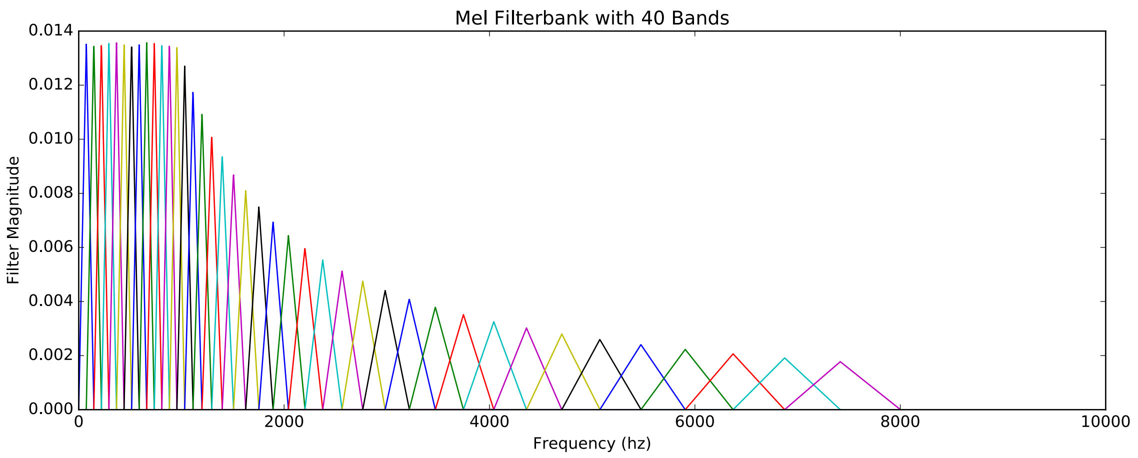


FIGURE 2.24: The shape of the logarithmically-spaced Mel Filterbank

2. Take the log of the magnitude of each element $\widetilde{\mathcal{M}(n,t)} \leftarrow \log(|\mathcal{M}(n,t)|)$. This models that fact that humans perceive loudness logarithmically as well as pitch.

3. Perform a DCT on each window of $\widetilde{\mathcal{M}}$: $\widetilde{\mathcal{M}(:,t)} \xrightarrow{\text{DCT}} MFCC(:,t)$. The resulting DCT coefficients are referred to as *MFCC coefficients*.

Figure 2.25 shows an example of the spectrogram of a song before MFCC is applied and after MFCC is applied and inverted to bring it back. The spectral envelope of the signal is clearly preserved, but the exact notes and their harmonic overtones are lost and blurred together within an octave.

2.5.3 Automatic Beat Tracking

Synchronizing features to beats is a common preprocessing task, particular for cover songs, as it is a natural way to control for tempo variations between recordings. This is usually much more difficult than one would think due to the cacophony of different instruments and sounds in most natural music which could be confused as indicating rhythmic events. Most techniques start with what is known as an *audio novelty function* (Bello et al. (2005)), which is a 1D surrogate signal at a much lower sample rate than the audio (by a factor of the STFT hop size, usually around 512) whose peaks indicate possible beat onsets (e.g. percussive strikes). Figure 2.26 shows an example of an audio novelty function which is defined as the sum of the positive differences in spectrogram bins spaced on a Mel-scale (Ellis (2007a)). Looking at the spectrogram in that figure, many vertical lines are apparent. These vertical lines indicate possible beat onsets, as percussive strikes tend to be broadband events. Therefore, a moment at which all of the frequency bins suddenly change in magnitude is a candidate for a percussive beat onset. In geometric terms, this can be viewed as the "jump" of a sliding window embedding using Mel-spaced magnitude STFT coefficients.

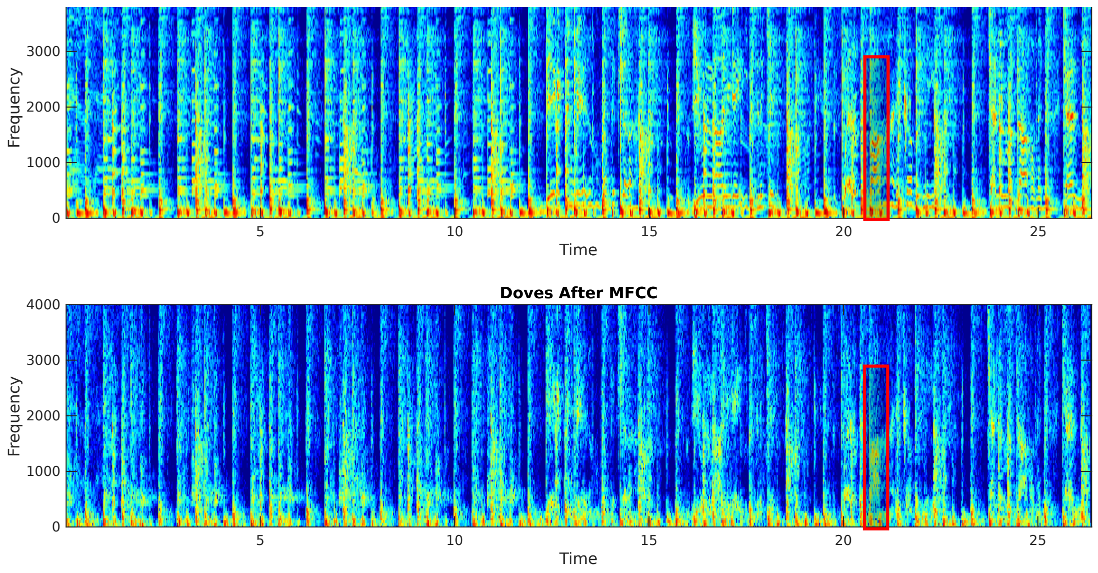


FIGURE 2.25: An example of MFCC features on a 30 second clip from Prince's "When Doves Cry." The top figure shows the spectrogram of the original clip, and the bottom shows the spectrogram of the clip after it has been summarized by the first 20 MFCCs per frame and brought back. The red box highlights a region that shows the overal spectral shape is preserved, but the exact harmonics of notes are blurred together.

There is an absolute zoo of possible audio novelty functions. For instance, Gouyon et al. (2007) review 172 different functions in a single paper. The success of the methods is highly variable and depends largely on genre, so it is difficult to say which one is the best in general, though the current state of the art appears to be a method by Böck and Widmer (2013). In general, orchestral music is quite challenging, as stringed instruments do not signal onsets with broadband as much as phase changes due to switching bowing directions, so some works use the complex STFT magnitude differences instead of the spectrogram absolute value differences.

Once audio novelty functions have been extracted, we would like to turn them into beat onsets so that we can figure out where a computer should "tap its foot" to the music. Most techniques which extract beat onsets use dynamic programming in some way. For instance, Ellis (2007a) use a technique which tries to find an optimal set of beat onsets that maximize the audio novelty function at each declared onset, while minimizing the deviation from a tempo within onsets. A more effective technique uses a hidden markov model that takes into consideration low values of the audio novelty function as indicating non-beat events, thereby incorporating more information than simply the peaks (Degara et al. (2012)). By most metrics, Krebs et al. (2015) appear to have improved on this slightly, with the current state of the art.

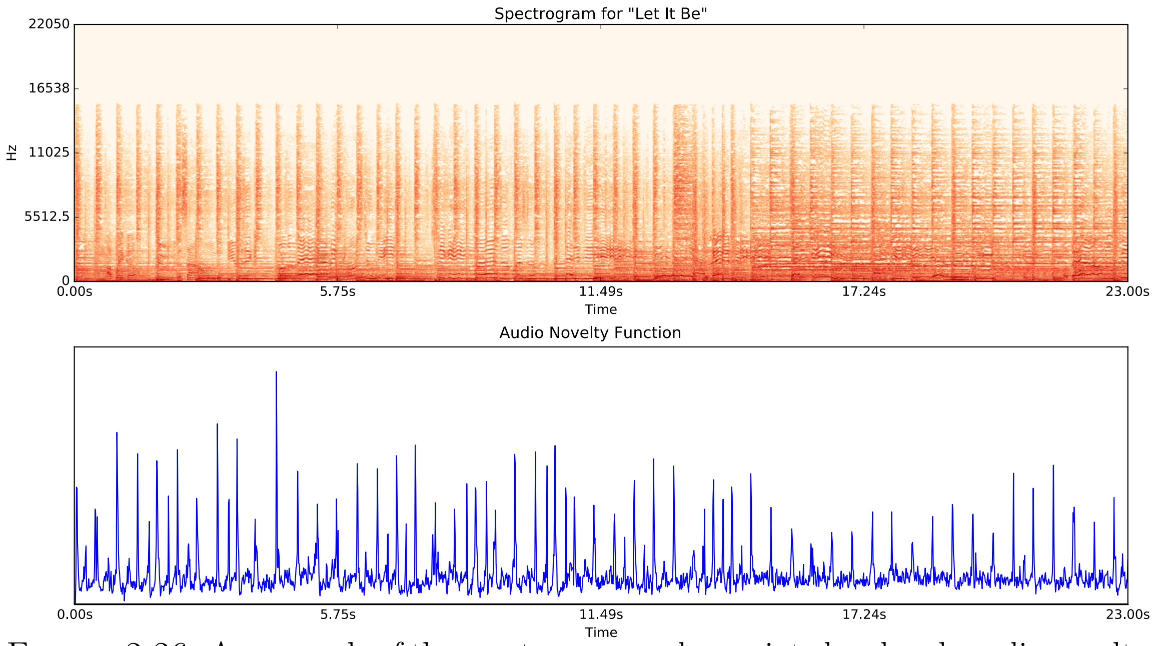


FIGURE 2.26: An example of the spectrogram and associated mel scale audio novelty function on "Let It Be" by The Beatles, using the method of Ellis (2007a). Peaks occur when there are vertical lines in the spectrogram corresponding to broadband percussive events.

2.5.4 Chroma Features

While the MFCC picks up on the overall shape of the spectrum, which means voices and instruments are well-preserved, the exact pitch content gets blurred out. Pitches are quite important for determining what key a song is in and for determining chord progressions. A feature set known as "Chroma" has been developed to fill this gap (Bartsch and Wakefield (2001) Fujishima (1999)). The idea is completely complementary to MFCC; rather than perserving the spectral shape, an absolute spectral reference of notes is disregarded, and frequencies corresponding to notes that are the same pitch are collapsed into a single equivalence class. For instance, a note that is heard as an "A" corresponds to 440hz, 880hz, 1760hz, etc (note the doubling of frequency per octave). There are 12 equivalence classes (halfstep notes) per octave, so the feature will report 12 different pitch strengths per spectrogram frame. More specifically, given an $M \times F$ spectrogram $\mathcal{S}(k,t)$, design a $12 \times M$ matrix W known as the "Chroma matrix," so that

$$C = WS \tag{2.63}$$

C is a $12 \times F$ matrix known as the *chromagram* which contains the strength of each note for each frame of the spectrogram. Figure 2.27 shows what this matrix looks like. Each row of the matrix W corresponds to a halfstep, and upon multiplication by the matrix F, the row numerically integrates energy around all frequencies that

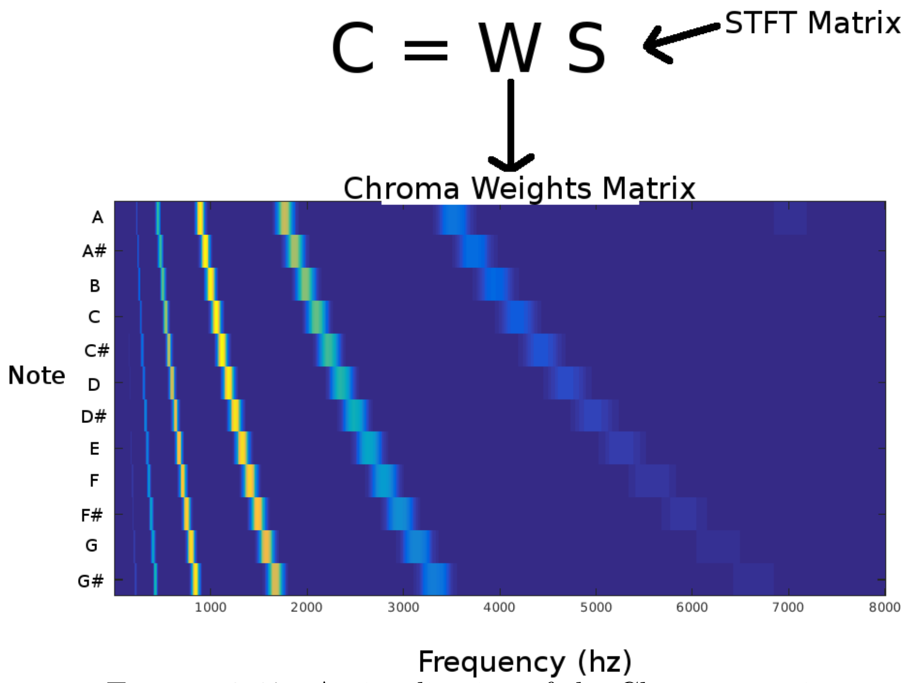


FIGURE 2.27: A visualization of the Chroma matrix

correspond to a note at that halfstep. A Gaussian bump is placed over the center frequency of each pitch in the halfstep equivalence class to count notes that are slight deviations away from a perfect pitch to still be counted properly. Note how the widths of the Gaussians become larger as the center frequency goes up, to model the fact that humans perceive pitch logarithmically.

Figure 2.28 shows the chromagram of two octaves of a chromatic scale (all halfsteps in sequence) played on the piano. This feature actually did quite well at picking up on the scale, which started on a D sharp. Notice, however, that in addition to the ground truth notes, the fifths of the notes show up strongly as well but slightly weaker, since the piano has overtones due to the physical structure of the vibrating string. To control for this in applications where the specific notes are important, Gómez (2006) designed a feature set known as “harmonic pitch class profiles” (HPCP). These have also been shown to work well in cover songs applications compared to other types of chroma features (Serra et al. (2008a)), which we have also verified experimentally, so these are the features we will use in our work. Because there are often many tricks and optimizations that accumulate over years of work, we use the Essentia library to compute these features (Bogdanov et al. (2013)), rather than relying on our own naive implementation.

2.5.5 Audio Fingerprinting

We now review a technique for audio retrieval known as “audio fingerprinting,” which is the technique used in the popular “Shazam” app (Wang (2006)), which can rec-

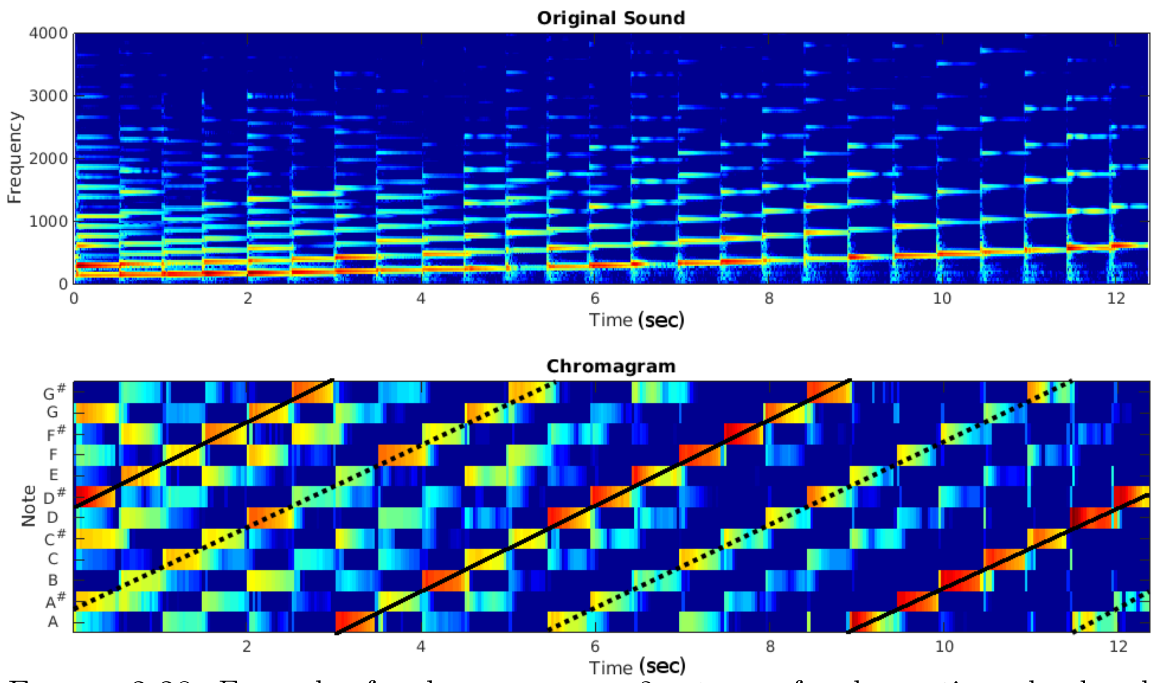


FIGURE 2.28: Example of a chromagram on 2 octaves of a chromatic scale played on a piano, which starts on a D sharp. The top plot shows the original spectrogram, while the bottom plot shows the chromagram. Solid black lines are drawn over the ground truth notes that are being played at each frame, and dashed black lines are drawn over fifths of the ground truth notes. Heat corresponds to strength of the note, where red means strong and blue means weak

ognize songs after only a few seconds of audio, provided that the query song is in a database. The effectiveness of this technique is quite remarkable and surprised many in the music information retrieval community. Picking up a song on a phone's microphone across a room introduces lots of noise and distortion, and the algorithm is supposed to work after only a few seconds, which seems like it should be impossible given how difficult most basic music information retrieval tasks are (we can't even perfectly track beats yet!). However, there is an advantage to looking for *exact copies of recordings*, which is what makes audio fingerprinting possible. Even if there is distortion, enough patterns will be in common between the spectrograms of the original and the distorted version over time, even if some features are missing. This is certainly not the case for cover songs, however, which is why that problem remains challenging (Chapter 3).

Haitsma / Kalker Technique

One of the earliest successful approaches for audio fingerprinting was by Haitsma and Kalker (2002). They turn their spectrogram images into binary images by recording whether the gradient at a particular spectrogram bin is going up or down as time

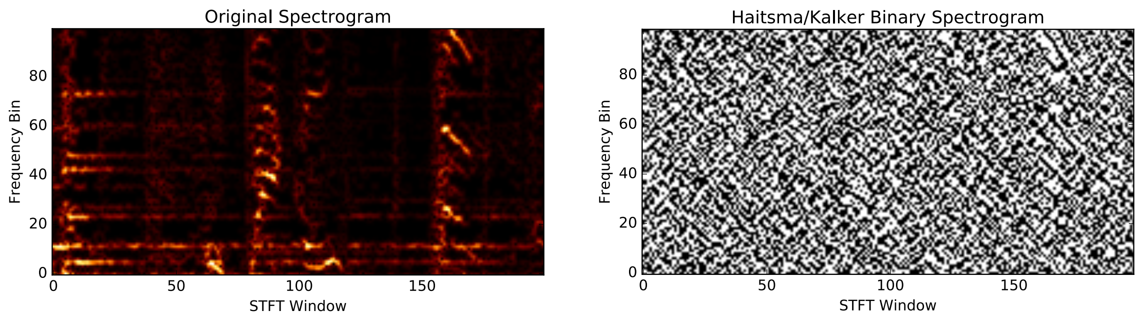


FIGURE 2.29: The bitmask has generated by Haitsma and Kalker (2002) on a spectrogram patch. Note, for example, how right after window 150 the descending pitch trajectories are discernible in the binary mask.

progresses to the right. In particular, given a spectrogram S, create a binary image B so that

$$B(t, f) = \left\{ \begin{array}{ll} 1 & X(t, f) + X(t+1, f+1) > X(t, f+1) + X(t+1, f) \\ 0 & \text{otherwise} \end{array} \right\} \quad (2.64)$$

Figure 2.29 shows such a mask. To reduce the effect of windows which may be offset between a database and query version, they choose a much larger window size then hop size so that there is a 95% overlap in the STFT windows. We find a similar use of long window sizes in regularizing the SSMs in MFCCs in our cover songs work (see e.g. Figure 3.2). Once they have these masks, they are able to hash blocks of these binary regions and compare them to a database, and this works very well for audio fingerprinting, even in the presence of some errors.

The Shazam Technique

The technique used in Shazam was developed by Wang et al. (2003), and it has similar performance to Haitsma and Kalker (2002), but its representation is much more sparse. The technique works by picking local maxes in the spectrogram and coming up with fingerprint primitives which are pairs of maxes. Over time, even for degraded recordings, the chances that some fraction of these primitives is contained in two songs which are not the exact same recording becomes vanishingly small. Figure 2.30 shows an example of a song query that we recorded from an album playing on our laptop speakers, which severely degraded the audio. When queried against all songs in the album, however, the correct song is returned by this technique.

Other Applications of Audio Fingerprinting

Aside from the Shazam application, there are some other very neat applications of audio fingerprinting. In the era of big data, there is a lot of code to automatically

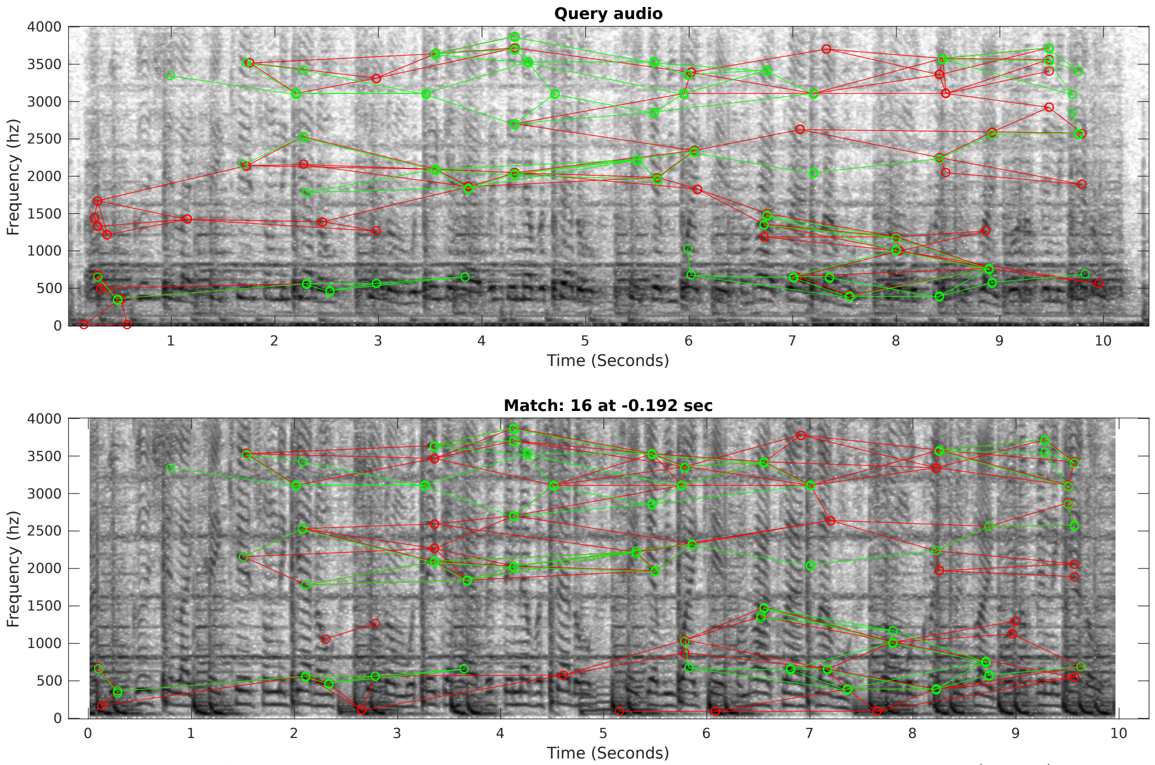
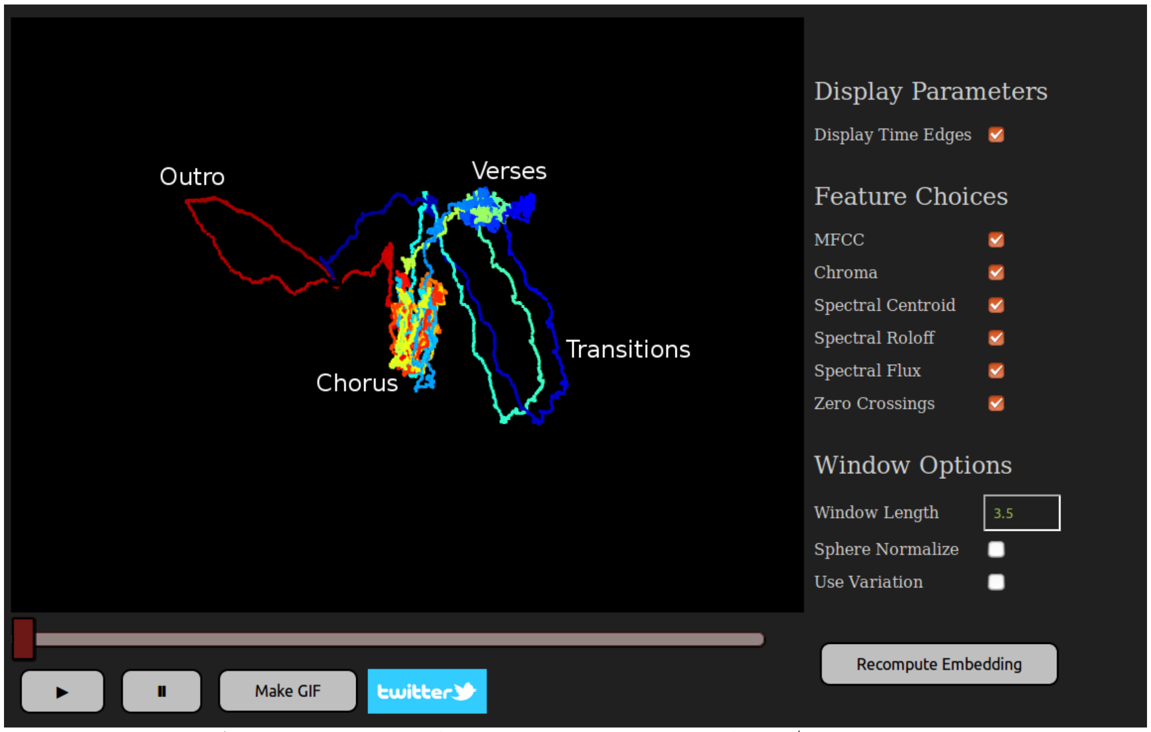


FIGURE 2.30: An example of the "Shazam technique" of Wang et al. (2003), using code by Ellis (2009). The query audio shown on top is a degraded version of Ma$e's "24 Hours To Live." This clip was compared to all tracks in his album Harlem World, and track 16 (the correct track) was returned. Matching fingerprint segments are drawn in green, while mismatched segments are drawn in red.

scrape music files from the web to make large corpora of music to advance research in music information retrieval, but it is likely that there are duplicate tracks that occur without manual checking. Audio Fingerprinting can detect and remove these duplicate tracks, but it will avoid removing cover songs or other similar but different enough songs that we want to keep. Additionally, in some cases, music producers will use the exact same clip at different parts of the same song. Audio Fingerprinting can be used to tell the difference between such mixes and unique renditions of the same musical expression, which may be difficult to distinguish by ear. This is a way to automatically reverse engineer part of the production of a song.

2.5.6 Loop Ditty

To help understand the features better, we created an interface that synchronizes PCA on sliding window embeddings summarized by the different features to the music, which we call "Loop Ditty." The effect is a time-ordered point cloud which "waves to the music" as the song progresses. Beats show up as back and forth motion, choruses and verses tend to cluster together, and transitions between different



Display Parameters

Display Time Edges ✓

Feature Choices

MFCC ✓
Chroma ✓
Spectral Centroid ✓
Spectral Roloff ✓
Spectral Flux ✓
Zero Crossings ✓

Window Options

Window Length 3.5
Sphere Normalize ☐
Use Variation ☐

FIGURE 2.31: An screenshot of the LoopDitty Interface (`http://www.loopditty.net`) on the song "Work Hard Play Hard" by Wiz Khalifa. Different colors indicate different times, so regions with multiple colors indicate recurrence in the music. We have annotated where the verse, chorus, transitions, and outro occur spatially.

segments are path-like. There are better ways to quantify music structure (McFee and Ellis (2014)), but we are not aware of a comparable interface. The closest work we have found is by Schwarz et al. (2008), who do 2D PCA on MFCC features to help users navigate through a collection of audio grains for concatenative synthesis.

Figure 2.31 shows a screenshot from our interface. Loop Ditty is currently live at the URL `http://www.loopditty.net`. It is powered by a web server which computes features on the server side, and by Javascript and WebGL to draw the 3D projected curve synchronized to the music on the client side. Users can paste any valid URL from SoundCloud, and the features will be computed for that song. They can also upload their own music. Once the music has been loaded in, users can check and un-check different feature sets they want to include, they can vary the window length, and they can even share their discoveries on Twitter, with an optional animated GIF of the curve in 3D. These updates are all done client side.

We have found that this interface is extremely fun to play with, and it was largely the motivation for using shape as a way to identify cover songs.

3

Cover Song Identification Fusing MFCC Shape Sequences And Chroma

A "cover song" is a different version of the same song, usually performed by a different artist, and often with different instruments, recording setting, mixing/balance, tempo, and key. It is not well-defined mathematically, but more of a "you know it when you hear it" scenario. For completeness, we provide two more definitions of cover songs below from two community-based online projects that chronicle cover songs

" A cover is a song that is performed by a performer different from the original performer. " [1]

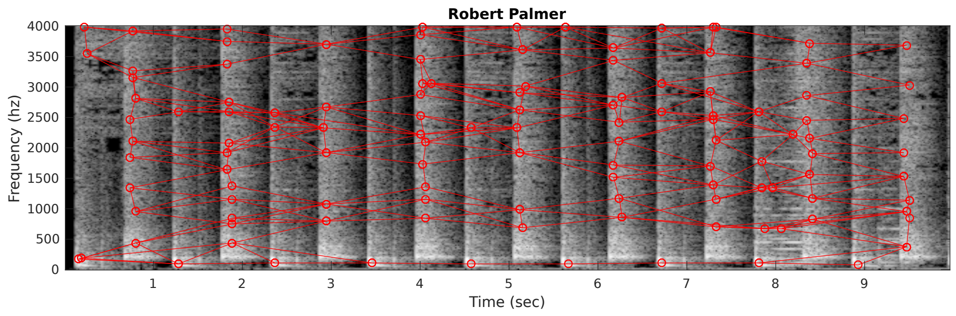
" A cover song, also known as a remake or cover version, is a new release of a song originally recorded or released by a different artist. For instance, The Rolling Stones released 'Satisfaction' in 1965 and Otis Redding and Devo later released their own versions of this classic song, putting their own musical spin on the well-known hit. " [2]

In this work[3], we will be evaluating algorithms on a set of songs that have been labeled as covers of each other, so we sidestep the issue of a rigorous general definition and instead declare success when our algorithm recognizes the correct song clusters. In this way, our work can be viewed as designing unsupervised features for assessing "high level music similarity" beyond exact recording retrieval. In fact, in most of the datasets, even live versions by the original artist are considered to be "cover songs," since the acoustic scenarios are so different. By then end of this chapter, we will

[1] `https://secondhandsongs.com/wiki/Guidelines/Cover`

[2] `https://www.coversproject.com/`

[3] About a third of this chapter overlaps with the paper Tralie and Bendich (2015), which is currently in print as part of the proceedings of ISMIR 2015.



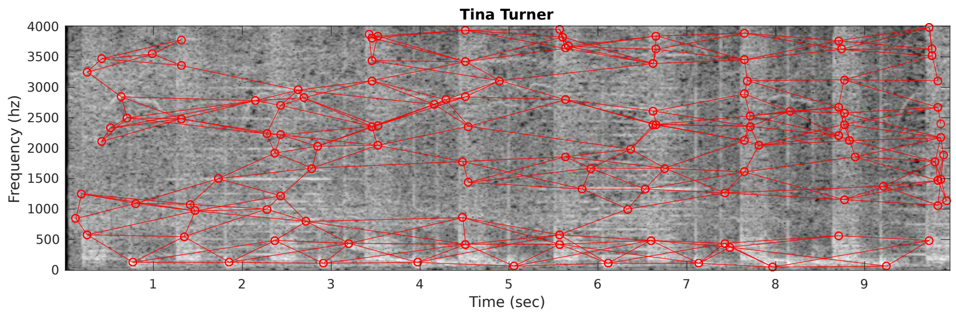


FIGURE 3.1: Audio audio fingerprints with the "Shazam" technique (Wang et al. (2003)) on Robert Palmer's "Addicted To Love" covered by Tina Turner. The fingerprints are shown in red. Although patterns in the spectrograms are qualitatively similar, none of their fingerprints match.

develop an unsupervised low level fusion technique which achieves state of the art results by combining our new geometric features with more traditional pitch-based features. In particular, state of the art supervised techniques on the popular "covers 80" benchmark dataset yield a mean reciprocal rank (MRR) of 0.625 (Chen et al. (2017)), while our completely unsupervised technique achieves a MRR of 0.845.

3.1 Automatic Cover Song Identification

Machine identification of cover songs is a surprisingly difficult classical problem that has long been of interest to the music information retrieval community (Ellis (2006)). This problem is significantly more challenging than traditional audio fingerprinting, because a combination of tempo changes, musical key transpositions, embellishments in time and expression, and changes in vocals and instrumentation can all occur simultaneously between the original version of a song and its cover. Figure 3.1 shows a motivating example. Hence, low level features used in this task need to be robust to all of these phenomena, ruling out raw forms of popular features such as MFCC, Constant-Q Transform (CQT), and Chroma.

One representative prior approach, as reviewed in Section 3.1.2, is to compare beat-synchronous sequences of Chroma vectors between candidate covers. The beat-syncing helps this be invariant to tempo, but it is still not invariant to key. However, many schemes have been proposed to deal with this, up to and including a brute force check over all key transpositions.

3.1.1 Our Contributions

Chroma representations factor out some timbral information by folding together all octaves, which is sensible given the effect that different instruments and recording environments have on timbre. However, valuable non-pitch information which is preserved between cover versions, such as spectral fingerprints from drum patterns, is obscured in Chroma representation. This motivated us to take another look at whether timbral-based features could be used at all for this problem. Our idea is that even if absolute timbral information is vastly different between two versions of the same song, the *relative evolution* of timbre over time should be comparable.

With careful centering and normalization within small windows to combat differences in global timbral drift between the two songs, we design MFCC-based features which are approximately invariant to cover. These features, which are based on self-similarity matrices (SSMs) of MFCC coefficients, can be used on their own to effectively score cover songs. This, in turn, demonstrates that even if absolute pitch is obscured and blurred, cover song identification is still possible. Furthermore, since the feature sets are so different from traditional pitch-based HPCP features used to do cover song identification, they provide complementary information. We devise various fusion schemes to exploit this in Section 3.3, which allows us to obtain better results than either feature set alone.

Section 3.1.2 reviews prior work in cover song identification. Our method using MFCC-based techniques is described in detail by Sections 3.2 and 3.2.3. We then provide three different fusion schemes in Section 3.3 to combine our features with traditional features for this task. Finally, we report results on the "Covers 80" benchmark dataset (Ellis (2007c)) in Section 3.4, and we apply our algorithm to the recent "Blurred Lines" copyright controversy.

3.1.2 Prior Work

To the best of our knowledge, all prior low level feature design for cover song identification has focused on Chroma-based representations alone. The cover songs problem statement began with the work of Ellis (2006), which used FFT-based cross-correlation of all key transpositions of beat-synchronous Chroma between two songs. A follow-up work by Ellis and Cotton (2007) showed that high passing such cross-correlation can lead to better results. In general, however, cross-correlation is not robust to changes in timing, and it is also a global alignment technique. Serra (2007) extended this initial work by considering dynamic programming local alignment of Chroma sequences, with follow-up work and rigorous parameter testing and an "optimal key transposition index" estimation (Serra et al. (2008a)). The same authors also showed that a delay embedding of statistics spanning multiple beats before local alignment improves classification accuracy (Serra et al. (2009)). A recent technique by Silva et al. (2016) approximates this technique by reporting median distances over all nearest neighbors of delay sequences from one track to another, so that the expensive quadratic local alignment is avoided. In a different approach, Kim et al. (2008) compare modeled covariance statistics of all Chroma bins, as well as covariance statistics for all pairwise differences of beat-level Chroma features, which is not unlike the "bag of words" and bigram representations, respectively, in text analysis. Other work by Bello (2007) modeled sequences of chords as a slightly higher level feature than Chroma. Slightly later work concentrated on fusing the results of music separated into melody and accompaniment (Foucard et al. (2010)) and melody, bass line, and harmony (Salamon et al. (2012)), showing improvements over matching Chroma on the raw audio.

Some of the more recent work on cover song identification has focused on fast techniques for large scale pitch-based cover song identification, using a sparse set of approximate nearest neighbors (Tavenard et al. (2012)) and low dimensional projections (Humphrey et al. (2013)). Ellis and Thierry (2012) and Nieto and Bello (2014) also use the *magnitude* of the 2D Fourier Transform of a sequences of Chroma vectors treated as an image, so the resulting coefficients will be automatically invariant to key and time shifting without any extra computation, at the cost of some discriminative power.

There have also been a number of recent works on feature fusion for cover song identification. Osmalsky et al. (2015) use standard rank aggregation techniques to fuse different feature sets that are based on beat-synchronous Chroma. They also show, rather surprisingly, that performance is boosted when incorporating features which, by themselves, would lead to poor performance, such as tempo and song duration. Followup work by Osmalsky et al. (2016) shows that adding our SSM-based features further improves results. Finally, Chen et al. (2017) show that using similarity network fusion on the pairwise scores between all different cover songs using two different constrained versions of Smith Waterman (one of which the authors call "DMax") gives better results than either alone. We draw particular inspiration from this work and use a similar approach in Section 3.3.2, though we also put our own twist on it in Section 3.3.4 which gives much better results.

Outside of cover song identification, there are other works which examine gappy sequences of MFCC in music, such as Casey and Slaney (2006). However, these works look at matched sequences of MFCC-like features in their original feature space. By contrast, in our work, we examine the *relative* shape of such features. Finally, we are not the first to consider shape in an applied musical context. For instance, Urbano et al. (2011) turns sequences of notes in sheet music into plane curves, whose curvature is then examined. To our knowledge, however, we are the first to explicitly model shape in musical audio for version identification.

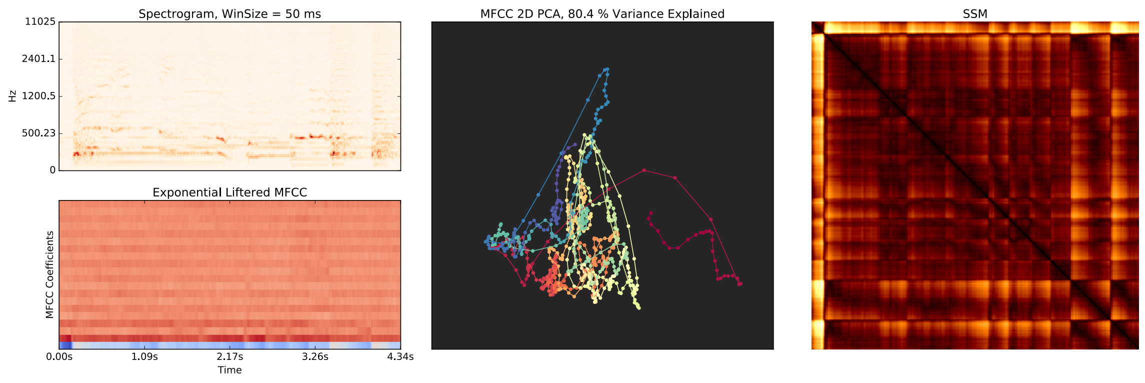
3.2 MFCC-Based Time-Ordered Point Clouds from Blocks of Audio

Before we describe how to fuse our features with other features, we first develop a full pipeline with only MFCC-based features. Surprisingly, even restricting solely to this feature set, which significantly obscures notes (Figure 2.25), we are able to achieve reasonable classification performance.

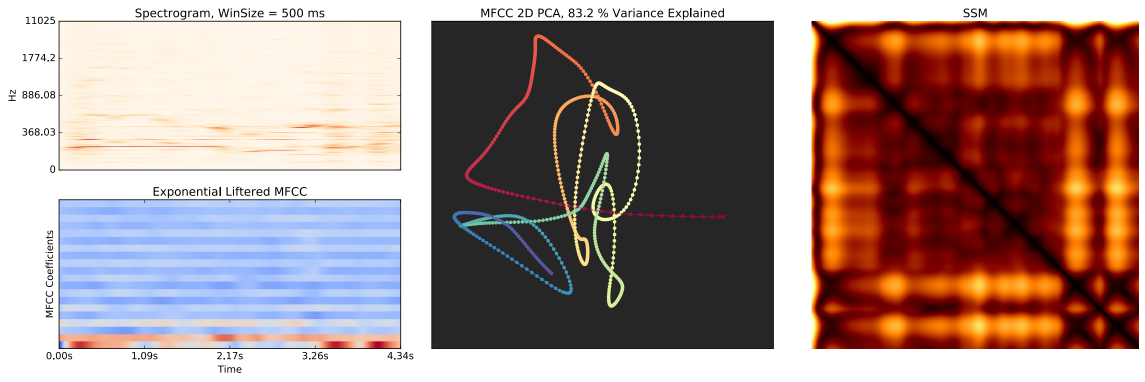
3.2.1 Beat-Synchronous Block/Windowing

As many others in the MIR community have done, including Ellis (2006) and Ellis and Cotton (2007) for the cover songs application, we compute our features synchronized within beat intervals. We use a simple dynamic programming beat tracker developed by Ellis (2007a). Similarly to Ellis and Cotton (2007), we bias the beat tracker with three initial tempo levels: 60BPM, 120BPM, and 180BPM, and we compare the embeddings from all three levels against each other when comparing two songs, taking the best score out of the 9 combinations. This is to mitigate the tendency of the beat tracker to double or halve the true beat intervals of different versions of the same song when there are tempo changes between the two. The trade-off is of course additional computation. We should note that other cover song works, such as Serra et al. (2008a), avoid beat tracking step altogether, hence bypassing these problems. However, it is important for us to align our sequences as well as possible in time so that the SSMs are in correspondence, and this is a straightforward way to do so.

Given a set of beat intervals, the union of which makes up the entire song, we take blocks to be all contiguous groups of B beat intervals. In other words, we create a sequence of overlapping blocks $X_1, X_2, ...$ such that X_i is made up of B time-contiguous beat intervals, and X_i and X_{i+1} differ only by the starting beat of X_i and the finishing beat of X_{i+1}. Hence, given N beat intervals, there are $N-B+1$ blocks total. Note that computing an embedding over more than one beat is similar in spirit to the Chroma delay embedding approach in Serra et al. (2009). Intuitively, examining patterns over a group of beats gives more information than one beat alone, the effect of which is empirically evaluated in Section 3.4. For all blocks, we take the window size W to be the length of the average tempo period, and we advance the window intervals evenly from the beginning of the block to the end of a block with a *hop size* of 512 samples. We would like to match beat-level periodicities and fluctuations therein, so it is sensible to choose a window size corresponding to the tempo. In Chapter 4, we will show some theory behind period matching the window size (see also Figure 4.16). For a typical tempo of 120bpm at a sample rate of $44100hz$, this leads to a window size of 22050 and an overlap of roughly 97.5% between windows. This is in contrast to most other applications that use MFCC sliding window embeddings, which use a much smaller window size on the order of 10s of milliseconds, generally with a 50% overlap, to ensure that the frequency statistics are stationary in each window. In our application, however, a longer window size makes our self similarity matrices (Section 3.2.2) smoother, allowing for more



(a) Window size 0.05 seconds



(b) Window size 0.5 seconds

FIGURE 3.2: PCA on the MFCC-summarized sliding window representation of an 8-beat block from the hook of Robert Palmer's "Addicted To Love" with two different window sizes. We prefer longer windows because they lead to a smoother embedding, which makes a pixel by pixel SSM comparison more robust.

reliable matches of beat-level musical trajectories, while having more windows per beat (high overlap) leads to more robust matching of SSMs using L_2 (Section 3.2.3). Figure 3.2 shows the first three principal components of an MFCC embedding with a traditional small window size versus our longer window embedding to show the smoothing effect. Similar tricks have been used to increase the robustness of audio fingerprinting (Haitsma and Kalker (2002)).

3.2.2 Euclidean Self-Similarity Matrices

We now describe how SSMs are used as a geometric feature in our pipeline. Unlike Bello (2009), who compares SSMs on an entire song, we compare SSMs summarizing blocks of audio summarizing on the order of tens of beats. For each beat-synchronous block X_l spanning B beats, we have a 13-dimensional point cloud extracted from the

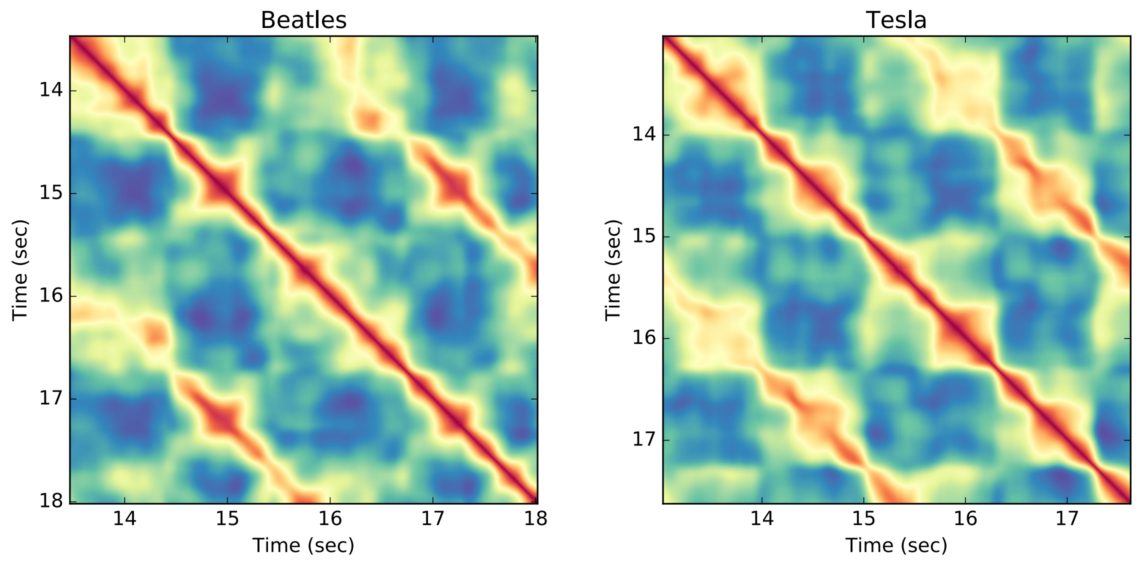


FIGURE 3.3: An 8 beat block from "We Can Work It Out" by The Beatles with a live cover by Tesla. In both clips, the singers sing the words "we can work it out" twice, and so there is one strong secondary diagonal in each one.

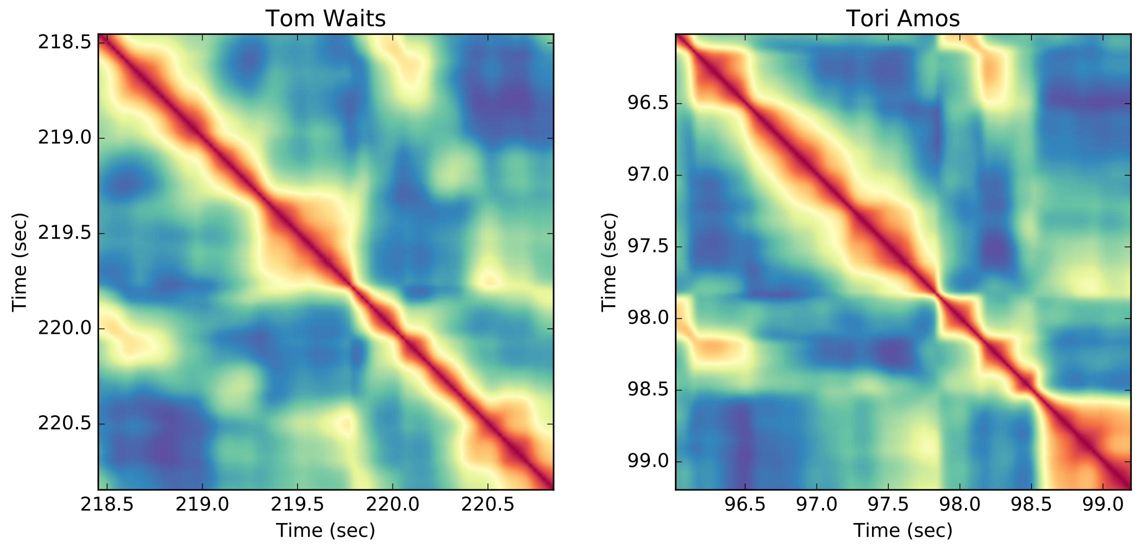


FIGURE 3.4: An 8 beat block from "Time" by Tori Amos and Tom Waits. In this block, the pattern is "time" + instrumentals + "time" + different instrumentals. The recurrence for the "time" is visible in both about 2/3 of the way through.

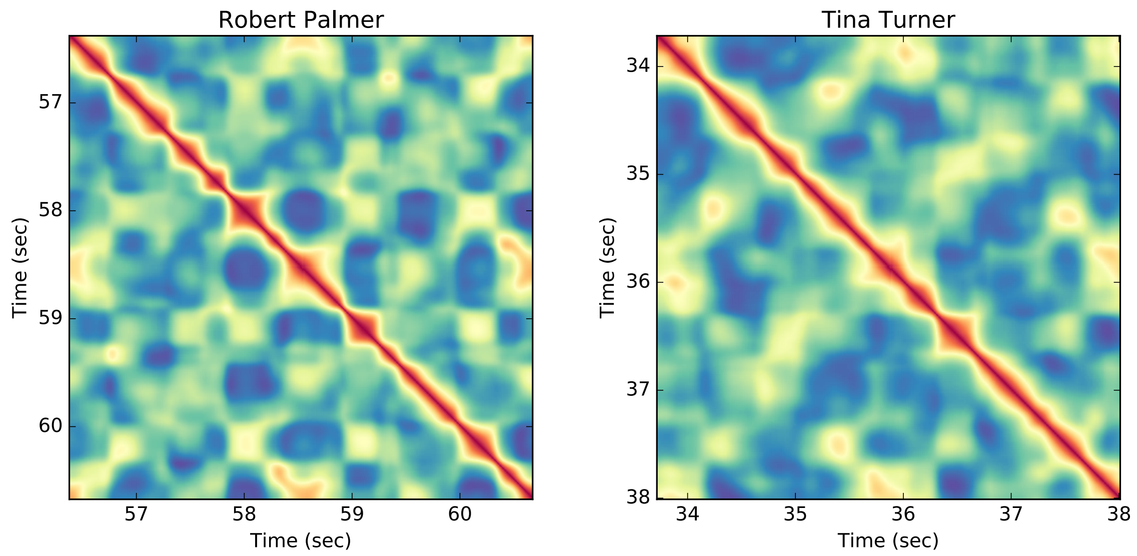


FIGURE 3.5: An 8 beat block from "Addicted To Love" by Robert Palmer covered by Tina Turner. This song has stronger rhythmic events which give rise to a grid-like structure in the SSMs.

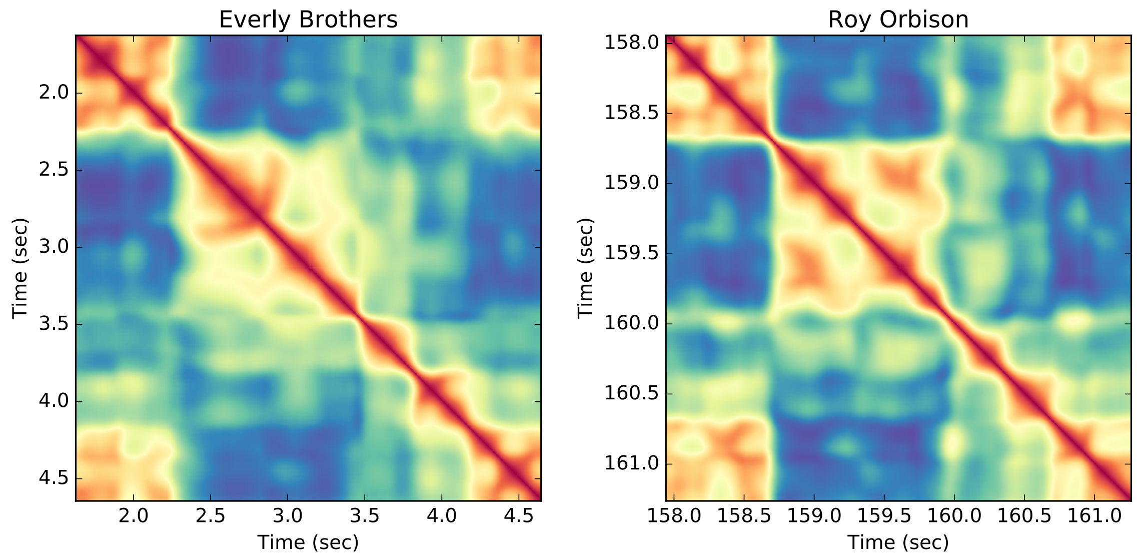


FIGURE 3.6: "Claudette" by the Everly Brothers and Roy Orbison. The pattern in this block is Guitar + "Oooh ooh Claudette" + Guitar.

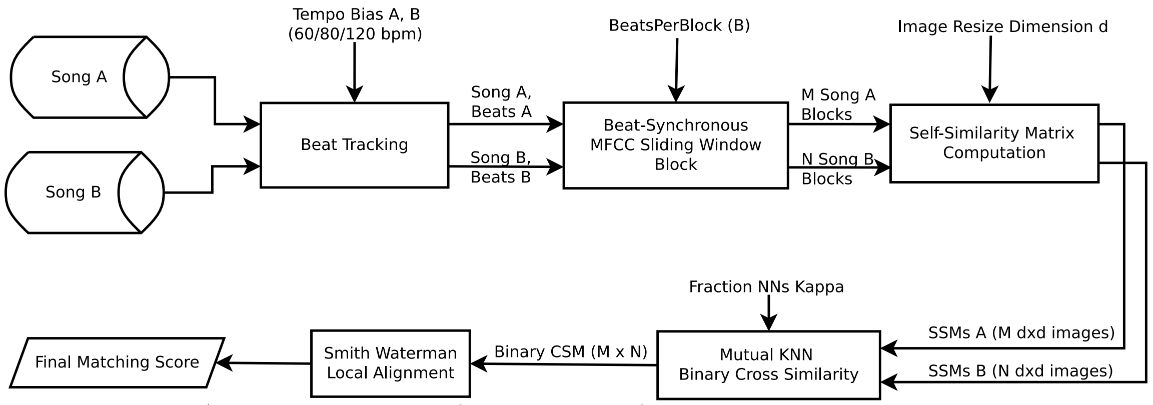


FIGURE 3.7: A block diagram of our system for computing a cover song similarity score of two songs using MFCC SSMs.

sliding window MFCC representation, from which we create a Euclidean SSM[4]. To help normalize for loudness and other changes in relationships between instruments, we perform z-normalization in each block (Definition 4) before computing the SSM. Note that not every beat block has the same number of samples due to natural variations of tempo in real songs. Thus, to allow comparisons between all blocks, we resize each SSM to a common image dimension $d \times d$, which is a parameter chosen in advance, the effects of which are explored empirically in Section 3.4.

Figure 3.3, Figure 3.4, Figure 3.5, Figure 3.6 show examples of MFCC SSM blocks with 8 beats and 500 windows per block which were matched between a song and its cover in the covers80 dataset. A divergent colormap was used to help visualize structural similarities, where red indicates similarity and magenta indicates dissimilarity. Of course, in all examples, there are subtle differences due to embellishments, local time stretching, and imperfect normalization between the different versions, but as we show in Section 3.4, there are often enough similarities to match up blocks correctly in practice.

3.2.3 Global Comparison of Two Songs

Once all of the beat-synchronous SSMs have been extracted from two songs, we do a global comparison between all SSMs from two songs to score them as cover matches. Figure 3.7 shows a block diagram of our system. After extracting beat-synchronous timbral shape features on SSMs, we then extract a binary cross-similarity matrix based on the L_2 distance between all pairs of self-similarity matrices between two songs. We subsequently apply the Smith Waterman algorithm on the binary cross-similarity matrix to score a match between the two songs.

[4] Note also that we exponentially lifter our MFCCs

Binary Cross-Similarity And Local Alignment

Given a set of N beat-synchronous block SSMs for a song A and a set of M beat-synchronous block SSMs for a song B, we compute a song-level matching between song A and B by comparing all pairs of SSMs between the two songs. For this we create an $N \times M$ cross-similarity matrix (CSM), where

$$\text{CSM}_{ij} = ||\text{SSMA}_i - \text{SSMB}_j||_F \tag{3.1}$$

is the Frobenius norm (L_2 image norm) between the SSM for the i^{th} beat block from song A and the SSM for j^{th} beat block for song B. In other words, we compute a pixel by pixel comparison between each pixel in each $d \times d$ resized similarity matrix, for every pair of similarity matrices. This process is made reasonable computationally with code in Listing 1.1, which is possible to speed up specifically because we are using an L_2 distance.

Given this cross-similarity information, we want to find the best local alignment between songs. Local alignment is a more appropriate choice than global alignment for the cover songs problem, since it is possible that different versions of the same song may have intros, outros, or bridge sections that were not present in the original song, but otherwise there are many sections in common. Good local alignments are possible when the cross-similarity matrix contains long diagonals with low values, indicating that many blocks match in sequence between the two songs. However, even in well-matching songs, it is possible that these diagonals are interrupted due to local errors in the beat tracking or for embellishments that add small sections. Therefore, we choose to use the Smith Waterman algorithm, since it can connect two long sections with interruptions by modeling gap costs.

To apply Smith Waterman, we turn the CSM into a binary matrix B^M. A binary matrix is necessary, since Smith Waterman only works on a discrete, quantized alphabet, not real values (Serra et al. (2008a)). It also has the added advantage of discarding possibly noisy real-valued metric information. To compute B^M, we take the mutual fraction κ nearest neighbors between song A and song B, as in Serra et al. (2009). That is, $B^M_{ij} = 1$ if CSM_{ij} is within the κM^{th} smallest values in row i of the CSM and if CSM_{ij} is within the κN^{th} smallest values in column j of the CSM, and 0 otherwise (Equation 1.10). Like Serra et al. (2009), we found that a dynamic distance threshold for mutual nearest neighbors per element worked significantly better than a fixed distance threshold for the entire matrix (Equation 1.9).

Once we have the B^M matrix, we can feed it to the Smith Waterman algorithm. We choose a version of Smith Waterman with diagonal constraints, which was shown to work well for aligning binary cross-similarity matrices for Chroma in cover song identification Serra et al. (2008a). In particular, we recursively compute a matrix D

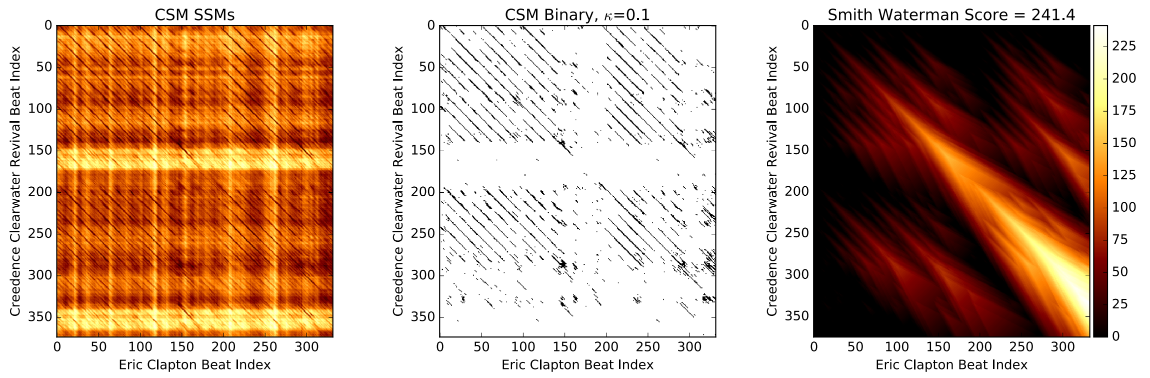


FIGURE 3.8: An example of the CSM of all pairwise SSMs of normalized MFCC blocks for the song "Before You Accuse Me" with versions by Creedence Clearwater Revival and Eric Clapton, with the nearest neighbors threshold $\kappa = 1$ and 20 beats per block. Long diagonal regions indicate many blocks match in sequence, which is a good indicator for a potential cover song, and this works even in the presence of a matching interruption around beat 150 in each song. As a result Smith Waterman returns a high fairly high score of 241.4.

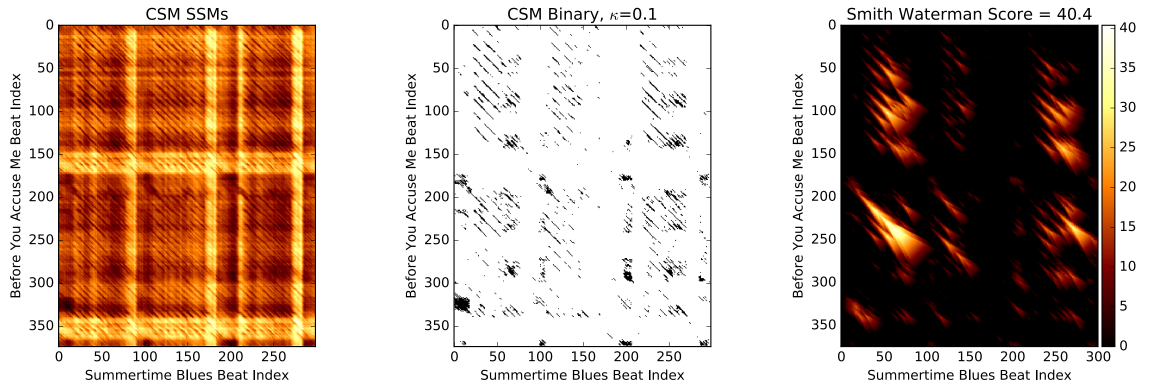


FIGURE 3.9: An example of the CSM of all pairwise SSMs of normalized MFCC blocks for the song "Before You Accuse Me" by Creedence Clearwater Revival versus the song "Summertime Blues" by the Beach Boys, with the nearest neighbors threshold $\kappa = 1$ and 20 beats per block. Since these songs are not true covers of each other, there are no long diagonal regions in common, so Smith Waterman returns a relatively low score of 40.4.

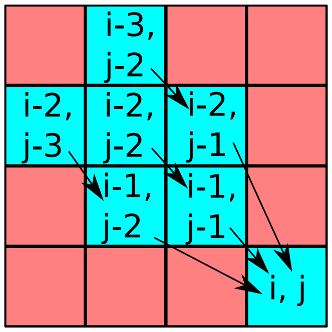


FIGURE 3.10: Constrained local matching paths considered in Smith Waterman, as prescribed by Serra et al. (2008a).

so that

$$D_{ij} = max \left\{ \begin{array}{c} D_{i-1,j-1} + (2B_{i-1,j-1} - 1) + \\ \Delta(B_{i-2,j-2}, B_{i-1,j-1}), \\ \\ D_{i-2,j-1} + (2B_{i-1,j-1} - 1) + \\ \Delta(B_{i-3,j-2}, B_{i-1,j-1}), \\ \\ D_{i-1,j-2} + (2B_{i-1,j-1} - 1) + \\ \Delta(B_{i-2,j-3}, B_{i-1,j-1}), \\ \\ 0 \end{array} \right\} \tag{3.2}$$

where

$$\Delta(a, b) = \left\{ \begin{array}{cc} 0 & \text{b} = 1 \\ \text{-0.5} & \text{b} = 0\text{,a} = 1 \\ \text{-0.7} & \text{b} = 0\text{,a} = 0 \end{array} \right\} \tag{3.3}$$

The $(2B_{i-1,j-1} - 1)$ term in each line is such that there will be a +1 score for a match and a -1 score for a mismatch. The Δ function is the so-called "affine gap penalty" which gives a score of $-0.5 - 0.7(g - 1)$ for a gap of length g. The local constraints are to bias Smith Waterman to choose paths along near-diagonals of B^M. This is important since in musical applications, we do not expect large gaps in time in one song that are not in the other, which would show up as horizontal or vertical paths through the B^M matrix. Rather, we prefer gaps that occur nearly simultaneously in time for a poorly matched beat or set of beats in an otherwise well-matching section. Figure 3.10 shows a visual representation of the paths considered through B^M. This is similar to other schemes for restricting warping paths in the DTW algorithm, as described by Müller (2007) (Figure 4.5 in that book), one of which was used in Smith Waterman for the cover songs application (Chen et al. (2017)), which the authors called "DMax." Another way of understanding these constraints is that they limit the slope of the warping path, so they are similar in spirit to the Itakura

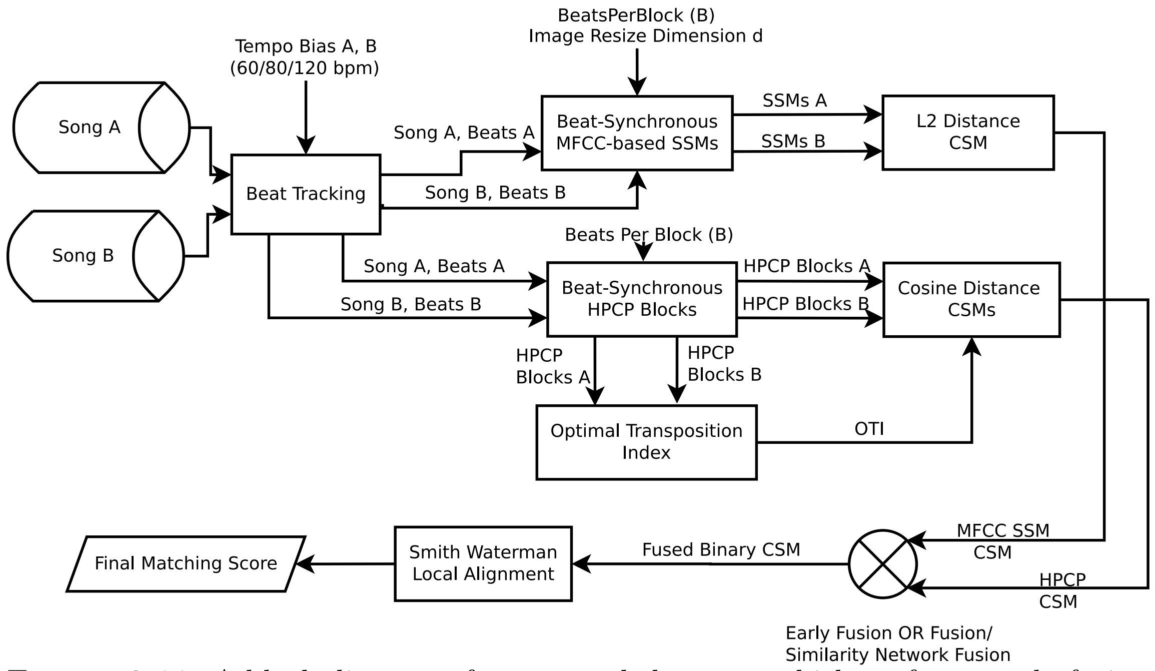


FIGURE 3.11: A block diagram of our extended system which performs early fusion of MFCC SSMs and HPCP before scoring with Smith Waterman. This pipeline is the same for OR fusion and similarity network fusion before the fusion happens.

Parallelogram in DTW (Section 2.4.3). However, unlike DTW, since we are looking for *local alignments*, we cannot rule out any portion of the CSM, so the algorithm still takes quadratic time even with these constraints.

Figure 3.8 shows an example of a CSM, B^M, and resulting Smith Waterman for a true cover song pair. Several long diagonals are visible [5], indicating large chunks of the two songs are in correspondence, and this gives rise to a large score of 241.4 between the two songs. Figure 3.9 shows the CSM, B^M, and Smith Waterman for two songs which are not versions of each other. By contrast, there are no long diagonals, and this pair only receives a score of 40.4.

3.3 Feature Fusion Incorporating Pitch

So far, we have focused entirely on a feature based on MFCC, but we have ignored a long line of work showing the utility of Chroma-based features. To tie our scheme into past work and to leverage information that has been lost in an MFCC representation, we also compute a pitch-based binary cross-similarity matrix B^C based on 12-bin HPCP features. First, we explain the details of how we use these features in block/window framework, and then we explain three different techniques to fuse

[5] Note that in songs which match, there is often more than one long diagonal, since there are multiscale repetitions in each song (measures, verse/chorus, etc)

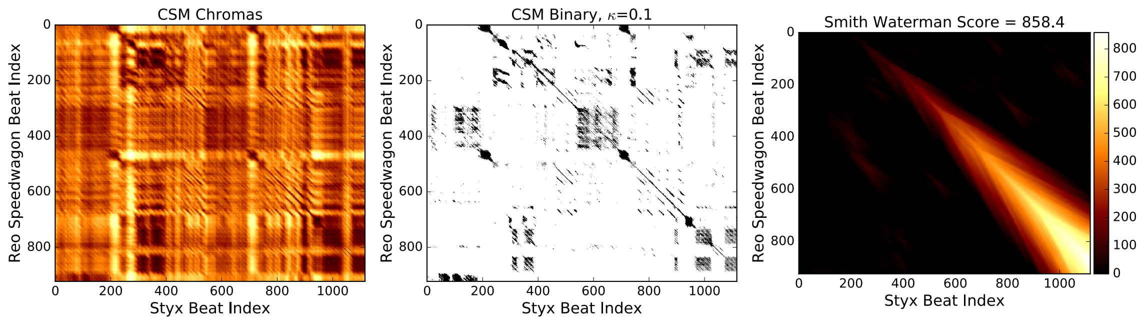


FIGURE 3.12: HPCP-based CSM between two cover versions of the song "Grand Illusion" using a block size of 20 beats and two HPCP windows per beat. There is one very long diagonal which leads to a high Smith Waterman score. Note that the Styx version is live, and there is a long intro where they are hyping up the crowd, so Smith Waterman doesn't start matching until around 25% of the way through that song.

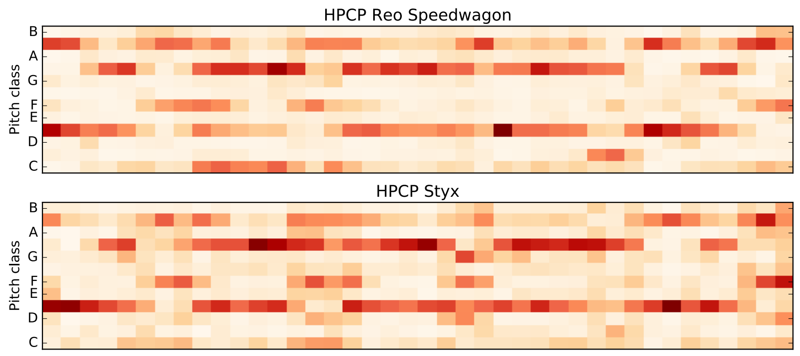


FIGURE 3.13: An example 20-beat block of HPCP features matching between the two covers in Figure 3.12. The matching notes can be seen visually.

these features with ours. Figure 3.11 shows an overall pipeline for the fusion process.

3.3.1 Blocked HPCP Features

Similarly to how we computed MFCC sliding windows over a block of B beats, and following Serra et al. (2009), we compute a stacked delay embedding of the HPCP features within B beats, with two HPCP windows per beat, for a total of $2B$ windows per block. This has an advantage over other works which do not use a delay, as it gives more context in time, and it is consistent with the overall block/windowing ideas in this thesis. To normalize for key transpositions, we need to determine an "optimal transposition index" (OTI) between two songs (Serra et al. (2008b)). Given the average HPCP vector $X_i \in \mathbb{R}^{+12}$ from song A and the average HPCP vector $Y_j \in \mathbb{R}^{+12}$ from song B, we compute the correlation $X_i^T Y_j$ over all 12 half-step transpositions of the original HPCP features in the block, and we use the transposition that leads to the maximum correlation. Then, we compute cosine distance between all pairs of HPCP blocks between the two songs. Finally, we can do the same binary K-nearest neighbor thresholding as before, followed by Smith Waterman. Figure 3.12 shows an example of two songs that match well with this technique, and Figure 3.13 shows one of the matching HPCP blocks for this example.

3.3.2 Late Similarity Network Fusion

Taking inspiration from Serrà et al. (2012) and Chen et al. (2017), we perform similarity network fusion (Section 2.3.5) on the Smith Waterman scores we get from MFCC SSMs and HPCP blocks. In particular, for a collection of N songs, we compute two or more score matrices that hold the pairwise scores between all songs. For instance, we may compute an $N \times N$ score matrix S_M for MFCC-based SSMs and an $N \times N$ score matrix S_H for HPCP blocks. Given a score matrix S, we compute the kernel matrix W as $W = 1/S$. Since Smith Waterman gives a higher score for better matching songs, this ensures that the kernel is close to 0 in this case. At this point, we perform similarity network fusion, as described in Section 2.3.5 (given a nearest neighbor threshold κ and some number of iterations), and we obtain a final $N \times N$ transition probability matrix P. From this, we can look along each row to find the neighboring songs with maximum fused probability.

3.3.3 Early OR Fusion

In addition to similarity network fusion after Smith Waterman has given scores, we can fuse features before running Smith Waterman. One advantage of doing fusion before scores are computed is that we don't need a network of songs to compute a score; we can obtain an improved score between two songs without any other context. [6]

One simple way of doing this is with *OR Fusion*. Given a binary CSM for MFCC SSMs and another CSM for HPCP, we compute a final binary cross-similarity matrix which is the union of the two: $B^F = B^M + B^C$, where + is binary OR. This is similar to the "early fusion" proposed in Foucard et al. (2010) for combining Chroma-based similarity matrices for melody and accompaniment. With this scheme, if there is a section of beats that one feature missed but the other matched, then their union will pick up on it. This way, sections where timbre matches well and sections where pitch match well will both be matched. The downside is that noise will be incorporated from both, increasing the chance of false positives, and this problem is compounded the more features that are fused this way.

[6] Note that Chen et al. (2017) refer to similarity network fusion after Smith Waterman as "early fusion" with respect to rank aggregation, which they call "late fusion," but we call their technique "late fusion" because we fuse before Smith Waterman with OR fusion and similarity network fusion, which is even earlier in the pipeline.

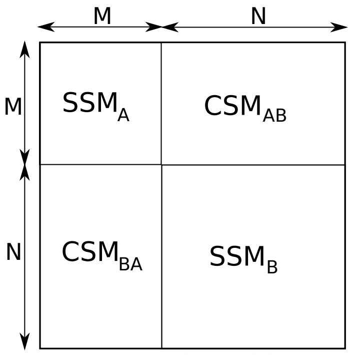


FIGURE 3.14: A pictorial representation of the SSM that results when concatenating song B to song A, which we feed to similarity network fusion for early fusion of low level features.

3.3.4 Early Similarity Network Fusion

A more sophisticated technique for early fusion is to apply similarity network fusion on the cross-similarity matrices obtained from two or more different feature sets before creating a binary CSM and sending it off to Smith Waterman. This can be thought of as a type of metric learning between all of the blocks which incorporates information from different modalities. The problem with a direct application of similarity network fusion is that it operates on SSMs. To make it so that CSMs fit into the framework, we create a "parent SSM" for each feature set that holds both SSMs and the CSM for that feature set. In particular, given song A with M blocks in a particular feature set and song B with N blocks in that feature set, form the SSM D_{AB} which is the SSM that results after concatenating song B to the end of song A. Let the SSM for song A be D_A, the SSM for song B be D_B, and the CSM between them be C_{AB}. Then D_{AB} can be split into four parts as follows (assuming zero-indexing)

$$D_{AB}(i,j) = \left\{ \begin{array}{cc} D_A(i,j) & i < M, j < M \\ D_B(i-M, j-M) & i >= M, j >= M \\ C_{AB}(i, j-M) & i < M, j >= M \\ C_{BA}(i-M, j) = C_{AB}^T(j, i-M) & i >= M, j < M \end{array} \right\} \tag{3.4}$$

Figure 3.14 shows this pictorially. Given such a matrix for each feature set, we could then run similarity network fusion and extract the cross-similarity sub-matrix at the end. The issue with this is the dynamic range of the SSM may be quite different from the dynamic range of the CSM, as it is likely that blocks in song A are much more similar to other blocks in song A than they are to blocks in B. To mitigate this, given a nearest neighbor threshold κ for the CSM, we compute the adaptive Gaussian kernel thresholds σ_{ij} (Equation 2.43) individually for D_A, D_B, and CSM_{AB}, and we put them together in the final kernel matrix W_{AB} according to Figure 3.14. Once we have such a matrix W_{AB} for each feature set, we can

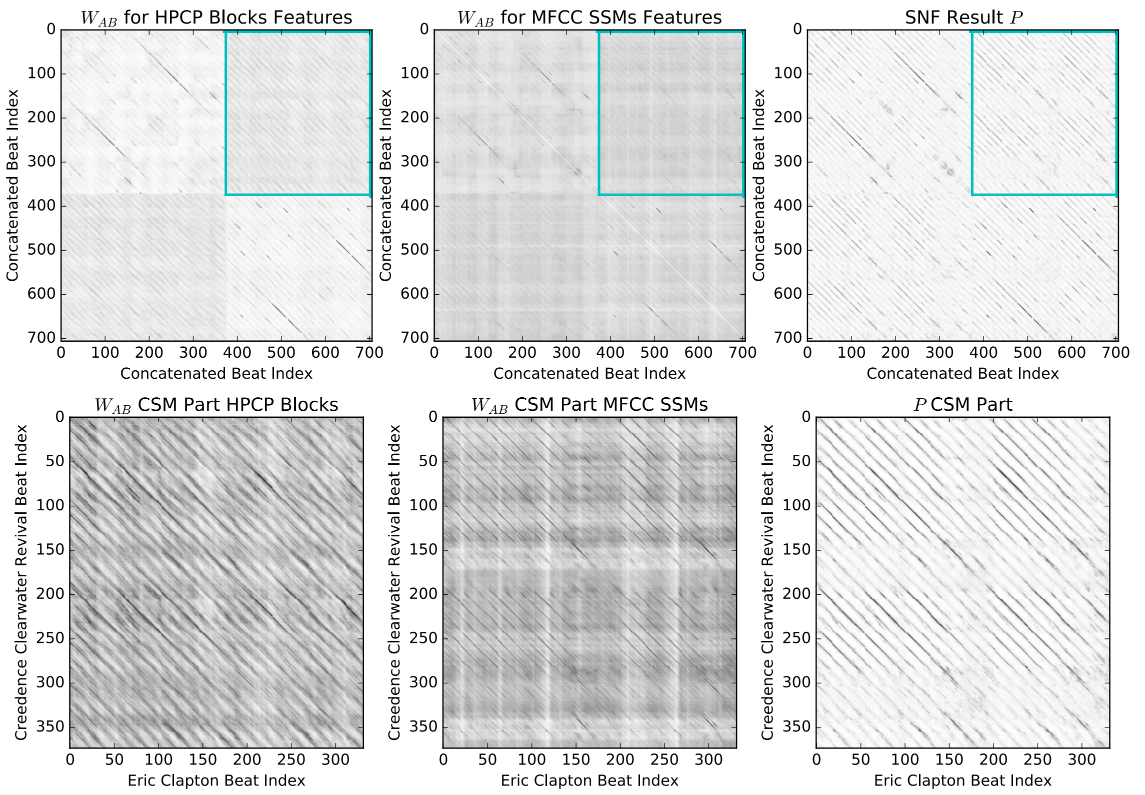


FIGURE 3.15: An example of early similarity network fusion on blocks of MFCC SSMs and blocks of HPCP features on the song "Before You Accuse Me" with versions by Eric Clapton and Creedence Clearwater Revival. The block size is 20, and there are three iterations of similarity network fusion with $\kappa = 0.1$. The kernels W_{AB} are shown for each, and the CSM portion is highlighted with a blue box. The final fused probability matrix P returned from similarity network fusion is shown in the upper right. The corresponding CSM portions for all three matrices shown for each on the bottom. Notice how in the fused probability matrix, the diagonal regions are much crisper and more distinct from the background than they are in the original two.

finally perform similarity network fusion. At the end, we will end up with a fused probability matrix P, from which we can extract the cross-probability $P_{C_{AB}}$. If we take the matrix $C'_{AB} = e^{-P_{C_{AB}}}$ (or any other monotonically decreasing function), we can then perform mutual nearest neighbors to turn C'_{AB} into a binary matrix and perform Smith Waterman as normal. Figure 3.15 shows an example of constructed matrices W_{AB} and the resulting fused probabilities P.

One advantage of this technique is that since the CSM and SSMs are treated together, any recurrent structure which exists in the SSMs can reinforce otherwise weaker structure in the CSMs during the diffusion process. This can potentially help to strengthen weaker beat matches in an otherwise well-matching section, leading to longer uninterrupted diagonals in the resulting binary CSM.

3.3.5 Early Fusion Examples

Before we launch into a more comprehensive experiment, we show a few examples of OR Fusion and early similarity network fusion to illustrate the value added. In addition to MFCC SSMs and HPCP blocks, we also compute features for the raw MFCC which has been Z-normalized per block, so we do a 3 way fusion. In each example, we used 20-beat blocks, $\kappa = 0.1$ for both similarity fusion and binary nearest neighbors, and 3 iterations of similarity network fusion. Figure 3.16 shows an example where the three individual features are rather poor by themselves, but where they happen to all pick up on similarities in complementary regions. As a result, both OR Fusion and early similarity network fusion do a fantastic job fusing the features. Figure 3.17 and Figure 3.18 both show examples where MFCC SSMs happen to do better than HPCP, but where the results fusing both are still better than each individually. As we will show, by itself, HPCP generally works better than any MFCC-based feature, but we wanted to highlight here a couple of examples where the reverse is true.

3.4 Results

3.4.1 Covers 80 Dataset

To benchmark our algorithm, we apply it to the standard "Covers 80" dataset Ellis (2007c), which consists of 80 sets of two versions of the same song, most of which are pop songs from the past three decades. The songs are split into two disjoint subsets A and B, each with exactly one version of every pair. To benchmark our algorithm on this dataset, we follow the scheme in Ellis (2006) and Ellis and Cotton (2007). That is, given a song from set A, compute the Smith Waterman score from all songs from set B and declare the cover song to be the one with the maximum score. Note that a random classifier would only get 1/80 in this scheme.

To get a better idea of the performance, we also compute the mean rank (MR), median rank (MDR), mean reciprocal rank (MRR), and the number of songs correctly identified past a certain number. All of these statistics are computed on the full set of 160 songs, which is more difficult and more prone to confusion than simply looking in set A or set B.

In general, we found that the nearest neighbor fraction κ and the dimension of the SSM image have very little effect, which is encouraging from the standpoint of robustness. Increasing the number of beats per block has a positive effect on the performance until the mid 20s. There is an inherent tradeoff in choosing a longer window size. As we will show in Section 4.4.2 (Figure 4.14), a longer delay has the effect of promoting diagonals, which will lead to a higher Smith Waterman score,

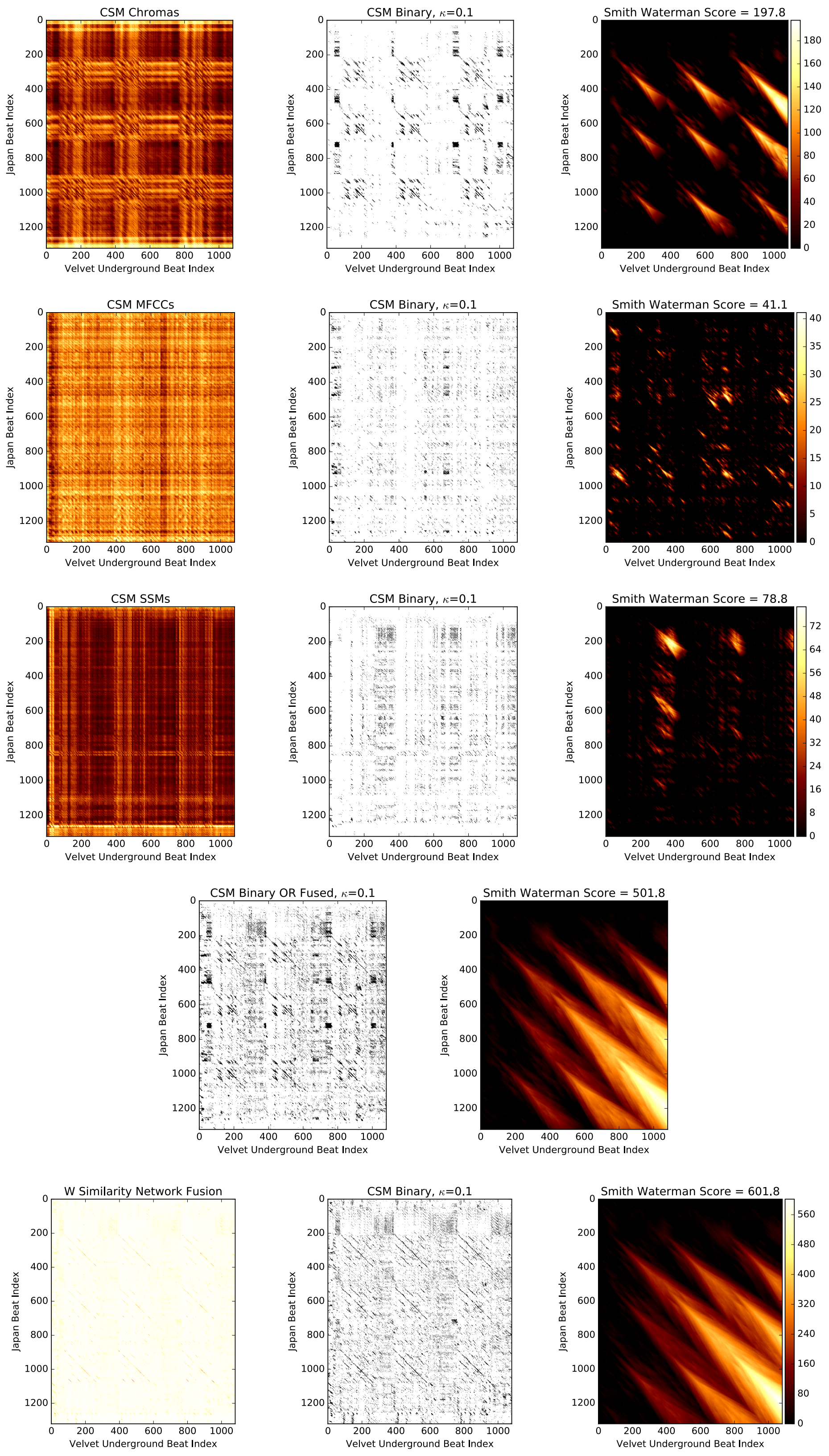


FIGURE 3.16: "All Tomorrow's Parties" by Japan and Velvet Underground.

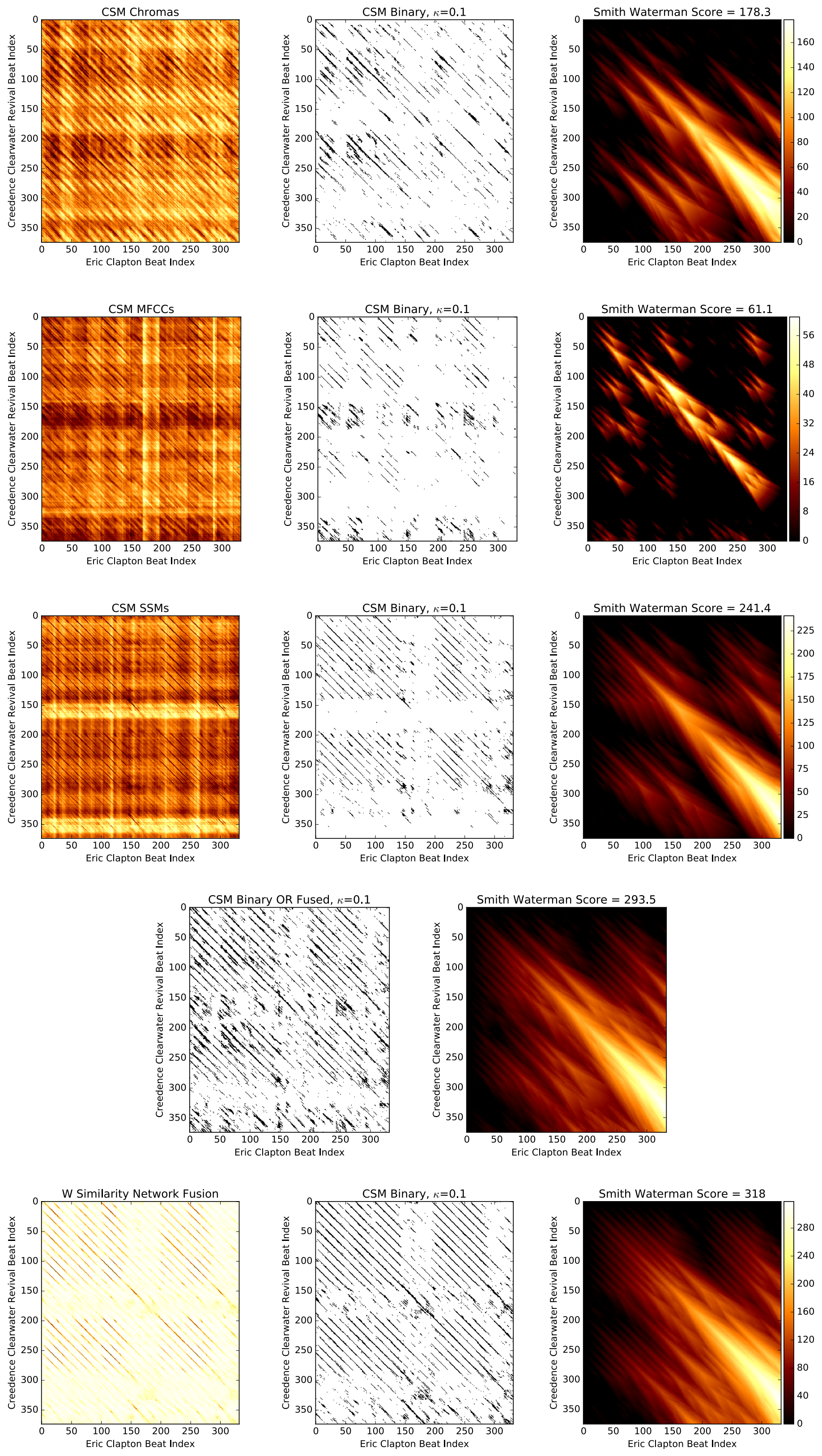


FIGURE 3.17: “Before You Accuse Me” by Eric Clapton and Creedence Clearwater Revival

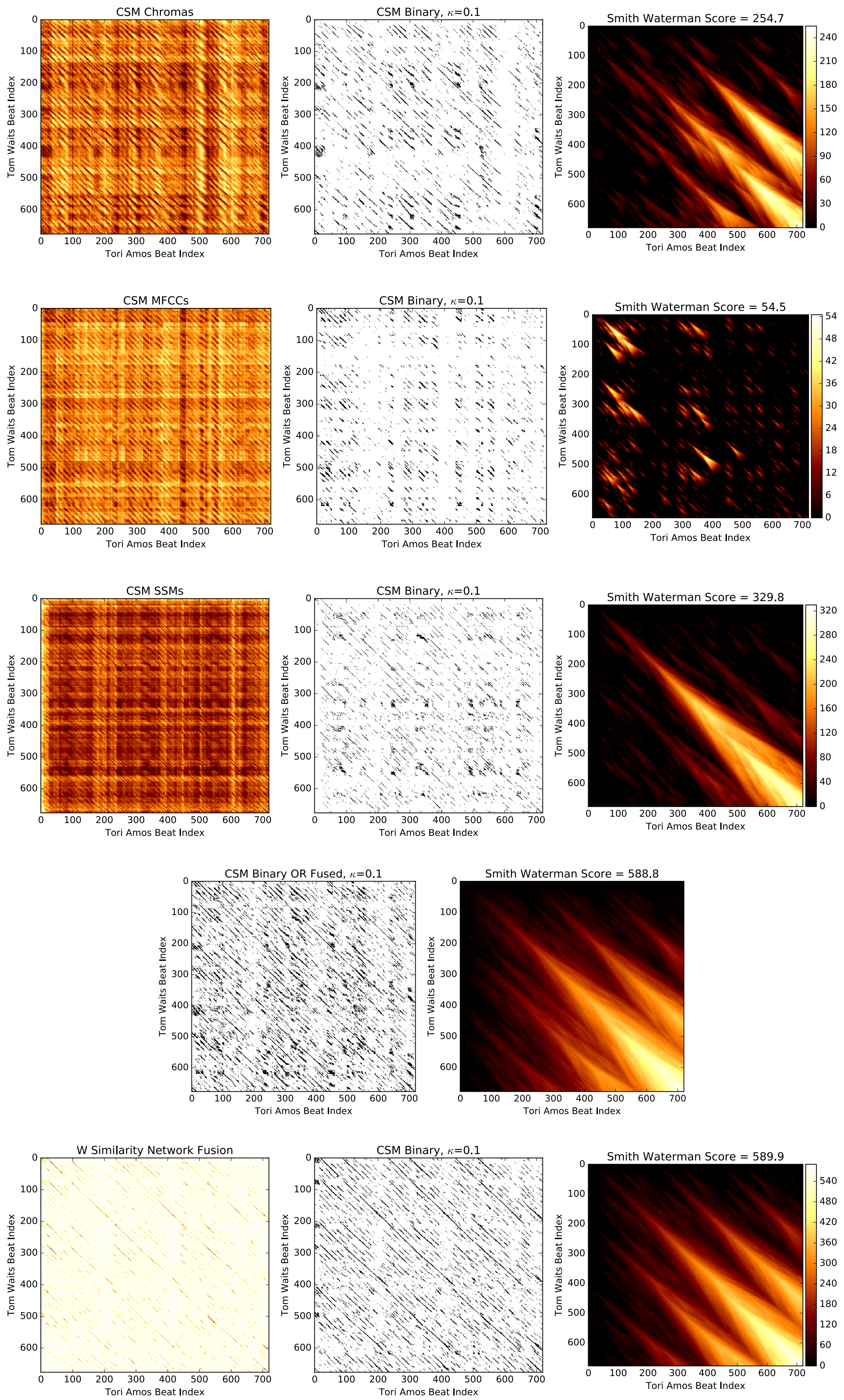


FIGURE 3.18: "Time" by Tom Waits and Tori Amos

but this could also increase false positives. In the end, we settle on $\kappa = 0.1$, $B = 20$ beats per block, and an SSM dimension of 50×50 pixels.

MFCC-Based Scores

Table 3.1: The result of blocked MFCC features and block MFCC SSM features on the Covers 80 dataset. We show here the importance of a longer window length and Z-normalization.

	MR	MRR	MDR	Top-1	Top-25	Top-50	Top-100	Score
SSM	15.14	0.615	1	91	130	144	155	48/80
SSM Win 1024	33.75	0.387	14.5	53	95	117	142	29/80
MFCC	29.71	0.538	2	79	108	122	142	42/80
MFCC Win 1024	33.86	0.44	7	62	103	116	135	33/80
MFCC No Norm	47.53	0.232	37.5	30	67	96	133	16/80
MFCC No Norm Win 1024	56.29	0.146	49.5	15	54	81	128	11/80

Table 3.1 shows the results of MFCC-based features on the Covers 80 dataset. To show the added value of using SSMs and normalization, we also report results with MFCC only (in blocks), with and without normalization. We also show the positive effect of having an MFCC window size that's equal to the beat length, rather than a more standard window size of 1024 samples (23 ms at 44100hz). Surprisingly, we obtain a score of 42/80 just by blocking and normalizing the MFCCs. This shows the power of having stacked delay MFCCs and of normalizing within each block to cut down on drift. Even without normalization and a suboptimal window size, the MFCCs still get 11/80. Overall, though, the SSM-based technique is the clear winner.

Chroma-Based Scores

Table 3.2: The result of blocked Chroma features on the Covers 80 dataset. Our results agree with the consensus in the literature that HPCP is the best variant of Chroma for cover songs, but we are able to get away with only 12 bins.

	MR	MRR	MDR	Top-01	Top-25	Top-50	Top-100	Score
CENS	37.04	0.411	10.5	59	94	110	137	31/80
HPCP 12	16.14	0.669	1	100	130	140	150	52/80
HPCP 36	17.58	0.665	1	99	128	138	149	51/80

To put MFCC-based approaches in context, we also report results for Chroma-based features by themselves. Also, to verify that HPCP is the right choice, we compare to another Chroma feature known as "Chroma-Energy Normalized Features" (CENS) (Müller and Ewert (2011)). Table 3.2 shows the results. Each test was run with 20 beats per block and 2 chroma windows per beat. The results are slightly better than SSMs + MFCCs, but not appreciably.

Fusion Results

Table 3.3: The result of late similarity network fusion on different feature sets on the Covers 80 dataset. MFCC SSMs + HPCP are better than either one alone. Also, MFCCs + SSMs fused with raw MFCCs improves results appreciably over either alone, suggesting that SSMs are picking up on useful relative information even when the absolute timbre is changing.

	MR	MRR	MDR	Top-01	Top-25	Top-50	Top-100	Score
SSMs/HPCP	8.53	0.793	1	122	146	151	156	61/80
SSMs/MFCC	13.96	0.7	1	107	132	142	155	55/80
HPCP to Self	16.71	0.7	1	107	131	138	150	54/80
SSMs to Self	15.28	0.632	1	93	133	139	153	47/80

Table 3.4: The result of Early OR Fusion on different feature sets on the Covers 80 dataset. Results are improvements over the individual feature sets.

	MR	MRR	MDR	Top-01	Top-25	Top-50	Top-100	Score
HPCP/MFCC/ SSMs, $\kappa = 0.1$	9.75	0.739	1	111	141	152	156	59/80
HPCP/SSMs $\kappa = 0.1$	11.82	0.738	1	112	142	145	154	58/80
HPCP/SSMs $\kappa = 0.05$	9.67	0.758	1	115	139	148	158	59/80

We have now approached the grand finale of this work: feature fusion. Table 3.3 shows the results of using late similarity network fusion after computing Smith Waterman scores. This improves the results over all of the features taken alone, which is encouraging since it is such a simple approach. In particular, MFCC-based features and HPCP are clearly complementary, as are raw MFCC features and SSM-based MFCC features, which shows that the SSMs are adding value. As a sanity check, we also did late similarity network fusion fusion on features to themselves (i.e. ordinary diffusion), which improved HPCP slightly but not appreciably, and which left MFCC the same, making it clear that the real value in these techniques is combining features with complementary information.

Table 3.4 shows the result of applying early OR Fusion. The results are improved, but overall they are similar to the late similarity network fusion results.

Table 3.5: The result of Early Similarity Network Fusion on different feature sets for the Covers 80 dataset. These are by far the best result, and in fact they appear to reach state of the art. When only two features are involved, similarity network fusion appears to only need a few iterations until converging to a good result. When three feature sets are involved, more iterations are likely needed, though the mean rank is the lowest out of all of the experiments in this case.

	MR	MRR	MDR	Top-01	Top-25	Top-50	Top-100	Covers80
HPCP/SSMs 10 Iters	7.74	0.845	1	131	147	152	155	68/80
HPCP/SSMs 5 Iters	7.76	0.846	1	131	147	152	155	68/80
HPCP/SSMs 3 Iters	7.52	0.849	1	131	150	152	155	68/80
HPCP/SSMs /MFCCs, 3 Iters	6	0.805	1	120	149	153	159	63/80
HPCP/SSMs /MFCCs, 3 Iters	6	0.805	1	120	149	153	159	63/80

Finally, Table 3.5 shows the early similarity network fusion results. These are by far the best results, and they appear to be state of the art, as a very recent work by Chen et al. (2017) using late similarity network fusion on Chroma-based features reported a Top10 score of 114 and a MRR of 0.625. By contrast, we report 131 songs in the **Top 01** (nearly all songs were correctly identified) and a MRR of 0.845 with our early similarity network fusion technique on HPCP + MFCC SSMs.

3.4.2 "Blurred Lines" Music Copyright Controversy

In addition to the Covers 80 benchmark, we apply our cover songs score to a recent popular music controversy, the "Blurred Lines" controversy Miao and Grimm (2013). Marvin Gaye's estate argues that Robin Thicke's recent pop song "Blurred Lines" is a copyright infringement of Gaye's "Got To Give It Up." Though the note sequences differ between the two songs, ruling out any chance of a high Chroma-based score, Robin Thicke has said that his song was meant to "evoke an era" (Marvin Gaye's era) and that he derived significant inspiration from "Got To Give It Up" specifically (Miao and Grimm (2013)). Without making a statement about any legal implications, we note that our timbral shape-based score between "Blurred Lines" and "Got To Give It Up" is in the 99.9$^{\text{th}}$ percentile of all scores between songs in group A and group B in the Covers 80 dataset, for $\kappa = 0.1$, $B = 14$, and $d = 200$. Unsurprisingly, when comparing "Blurred Lines" with all other songs in the Covers 80 database plus "Got To Give It Up," "Got To Give It Up" was the highest ranked. For reference, binary cross similarity matrices are shown in Figure 3.19, both for our timbre shape based technique and the delay embedding Chroma technique in Serra et al. (2009).

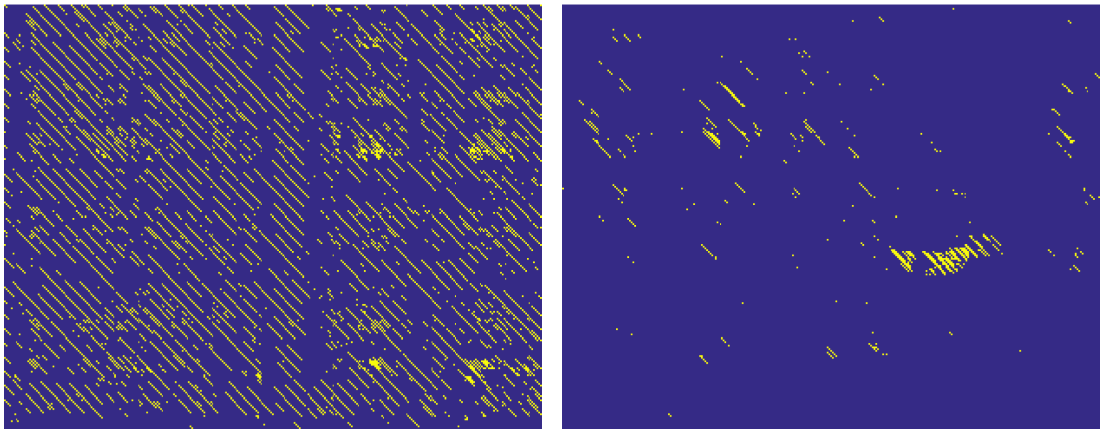

(a) Shape-based timbre (b) Chroma delay embedding

FIGURE 3.19: Corresponding portions of the binary cross-similarity matrix between Marvin Gaye's "Got To Give It Up" and Robin Thicke's "Blurred Lines" for both shape-based timbre (our technique) and Chroma delay embedding

The timbre-based cross-similarity matrix is densely populated with diagonals, while the pitch-based one is not.

3.5 Other Geometric Features

Self-similarity matrices describe all of the metric information of a time-ordered point cloud up to an isometry, but they take up quadratic space in the number of points. In this section, we explore some one dimensional isometry blind techniques for quantifying the geometry.

3.5.1 Space Curve Curvature/Torsion Scale Space

Velocity, curvature and torsion are time-ordered isometry invariants which can be stored more compactly. For the case of MFCC descriptors, they are like delta and delta/delta coefficients, respectively (Furui (1986)), though we have more control with the use of scale space. As shown in Appendix A, the derivations have assumed that the given curves are parameterized over a continuous interval. However, in real data, as with sliding window MFCCs, curves are discretely sampled. By convention, we index discrete signals with square braces and n

$$\boldsymbol{\gamma}[n] = (\gamma_1[n], \gamma_2[n], ..., \gamma_d[n]) \in \mathbb{R}^d \quad (3.5)$$

So that each sample is a vector and each vector component can be viewed as a 1D discrete time series. If we connect adjacent time samples with line segments to form piecewise linear curves in $\mathbb{R}^d$, that presents some issues for the above definitions, because curvature is either zero in the interor of the line segments or undefined at the vertices (since the curve is not $\mathbb{C}^2$ there in general). One very simple way to estimate the curvature vector is to take the negative of the "Discrete Laplacian" (Section 2.3.1, Taubin (1995)):

$$\overline{\boldsymbol{e_2}}[n] \approx \frac{1}{2}(\boldsymbol{\gamma}[n-1] + \boldsymbol{\gamma}[n+1]) - \boldsymbol{\gamma}[n] \tag{3.6}$$

For this application, though, it isn't sufficient, because it is too local and far too susceptible to noise. Instead, we will generalize the "curvature scale space" (Mokhtarian and Mackworth (1986), Mokhtarian et al. (1996)) from 2D curves to space curves and derive a similar formula for torsion. The goal is to smooth the components of the curve with successively wider Gaussians, yielding a multiscale estimation of curvature. Curvature scale space uses a clever trick that convolving a discontinuous function with a continuous one yields a continuous function, whose derivative can then be taken. In particular, define the Gaussian with scale σ as

$$g_\sigma(t) = \frac{1}{\sigma\sqrt{2\pi}} \exp^{-t^2/2\sigma^2} \tag{3.7}$$

then convolving the continuous $g_\sigma(t)$ with components of $\gamma[n]$, which are discontinuous pulse trains, yields the following

$$\gamma_{i\sigma}(t) = \gamma_i[n] * g_\sigma(t) = \sum_{n=0}^{N} \gamma_i[n] g_\sigma(t-n) \tag{3.8}$$

which is continuous, as promised. Then, by linearity of the derivative:

$$\gamma'_{i\sigma}(t) = \sum_{n=0}^{N} \gamma_i[n] g'_\sigma(t-n) = \gamma_i[n] * g'_\sigma(t) \tag{3.9}$$

and in general

$$\gamma^n_{i\sigma}(t) = \gamma_i[n] * g^n_\sigma(t) \tag{3.10}$$

Hence, we no longer estimate the derivatives of $\boldsymbol{\gamma}(t)$ directly by pairwise differences; we instead get smoothed estimates by convolving with derivatives of Gaussians of different widths. In other words, the derivative and convolution by a Gaussian commute, even for functions which aren't differentiable. This is a common trick used in scale space theory, and in this context the Gaussian is referred to as a "mollifying function." To obtain the final expressions for curvature, we plug these Gaussian derivatives into the curvature formulas obtained in Appendix A.

It is worth noting that from the perspective of persistence, there isn't much information in the coarser levels of a scale space (Chen and Edelsbrunner (2011)). However, for the purposes of this work, it is useful to have very slowly varying curves at the coarse levels. Also, as discussed by Yuille and Poggio (1985), it is possible to reconstruct functions up to translations from the zero crossings of the scales spaces of their derivatives, which justifies using all levels.

3.5.2 Velocity And Curvature Results

Table 3.6: The results of various combinations of velocity and curvature with each other and with other features on the Covers 80 dataset.

	MR	MRR	MDR	Top-01	Top-25	Top-50	Top-100	Score
Vel $\sigma = 10$	30.53	0.447	7	64	101	121	143	34/80
Vel $\sigma = 60$	37.74	0.346	16	47	90	111	142	25/80
Vel $\sigma = 100$	45.26	0.304	22	39	84	105	130	23/80
Curv $\sigma = 60$	50.96	0.196	44	23	62	87	135	11/80
Vel $\sigma = 10$, Curvs $\sigma = 60$	28.39	0.452	5.5	62	106	121	148	36/80
Vel $\sigma = 10$, Vel $\sigma = 60$	28.37	0.498	3.5	72	111	118	144	37/80
Vel $\sigma = 10$, Vel $\sigma = 60$, Curvs $\sigma = 60$	28.84	0.49	3	69	109	117	142	38/80
Vel $\sigma = 10$, Vel $\sigma = 60$ Early SNF	26.64	0.521	2	74	114	126	142	39/80
SSMs, Vel $\sigma = 10$	15.32	0.614	1	93	128	144	156	48/80
SSMs, MFCCs Vel $\sigma = 10$	13.19	0.674	1	101	130	145	156	53/80

As with all of our features, we compute curvature and velocity in blocks, and we create CSMs by comparing all pairs of blocks, which we then feed to Smith Waterman. We take 400 curvature samples per block, and we try out various levels of smoothing σ. Table 3.6 shows the results of various combinations of velocity and curvature with each other and with other features. It is clear that combining different scales helps, but if we have to choose a scale, smoother is better. Also, it appears that velocity is more useful than curvature overall[7]. However, none of these results beat the SSM results, and combining velocity with SSMs and MFCCs does worse than the combination of just the two. In spite of our efforts, we take this as a positive sign, as SSMs are incredibly simple to explain and compute compared to most other isometry blind features.

[7] We also tried experiments with torsion and saw even further diminishing returns

3.6 Conclusions And Future Work

In this chapter, we showed that timbral information in the form of MFCC can indeed be used for cover song identification. Most prior approaches have used Chroma-based features averaged over intervals. By contrast, we show that an analysis of the fine relative shape of MFCC features over intervals is another way to achieve good performance. This opens up the possibility for MFCC to be used in much more flexible music information retrieval scenarios than traditional audio fingerprinting. More generally, it is possible that our features are more appropriate for more complex music than US pop music, where evolution of timbre plays at least as an important of a role as pitch. The low level fusion described in this chapter may also be of independent interest for genre independent music structure analysis.

The main drawback of our technique is a strong reliance on beat trackers, since we do a direct L_2 distance on SSMs, meaning they need to be in good alignment. In practice, beat trackers may not return correct onsets. Our current remedy for this of using different tempo levels blows up computation by a factor of 9. Also, coming up with a single beat level is ill-posed, since usually there are metrical hierarchies (Quinton et al. (2015)). There does seem to be a recent convergence of techniques for rhythm analysis, however, so hopefully our system could benefit from them as a black box.

There are also computational drawbacks that still exist in these techniques. We are reliant on Smith Waterman, which is a quadratic algorithm, and early similarity network fusion adds another quadratic time complexity algorithm even with sparse nearest neighbor sets. To address this, we are in the process of implementing GPU algorithms for ever step of our pipeline, and we hope to apply it to the "Second Hand Songs Dataset," which is a subset of the Million Songs Dataset (Bertin-Mahieux et al. (2011)) with cliques of cover songs.

Finally, we note that there has been similar work in video analysis for detecting copyright infringement in videos. On Youtube, for instance, some may try to rotate/scale/time warp a video so that it is not detected. The problem of identifying such copies could be thought of as "cover videos." Wu et al. (2009) show that computing and analyzing different SSM for all frames in a video clip works to this end, and Yeh and Cheng (2009) and Bronstein et al. (2010) use Smith Waterman on other features which are approximately invariant under these transformations. This work makes us think that perhaps our technique of block windowed SSMs is applicable beyond music to videos and other sequences. We are particularly interested in 3D motion sequences.

4

Sliding Windows of Periodic Videos

We have now demonstrated the use of geometric descriptors of time series with the application of cover song identification in 1D time series of audio. In this chapter, we look at an example of a multivariate time series: video data. We devise a novel sliding window embedding scheme for videos, and we use it to quantify periodicity and quasiperiodicity of motion. In the process, a variety of geometric structures emerge. Periodic time series show up as topological loops, and the roundness of these loops can be used to score how periodic the video is. Surprisingly, specific cases of periodic videos with harmonics sometimes show up as the boundary of a Möbius strip (Section 4.3.3). Finally, quasiperiodic videos fill out a torus, which has been known for some time, but topological data analysis gives us a leg up on quantifying that geometry, and we incorporate one of the first known uses of persistent H_2 to this end.

To apply our technique, we show that our ranking of periodic videos agrees with humans on the Amazon Mechanical Turk (Section 4.5). Finally, in the grand finale of this chapter, we use the loop, Möbius strip, and torus to automatically tell apart different types of anomalies in videos of vibrating vocal folds, known as *high speed glottography* in Section 4.6.

4.1 Basic Scheme

4.1.1 Sliding Window Videos

Define a continuous video as a function $\boldsymbol{X}(t) : \mathbb{R}^+ \rightarrow \mathbb{R}^W \times \mathbb{R}^H$. That is, a video time series is a curve in the space of image frames, each of resolution $W \times H$ (W is the width and H is the height). This is a special case of Definition 1. Figure 4.4 shows an example synthetic periodic video of a pendulum, which we will use as a prototypical periodic video throughout.

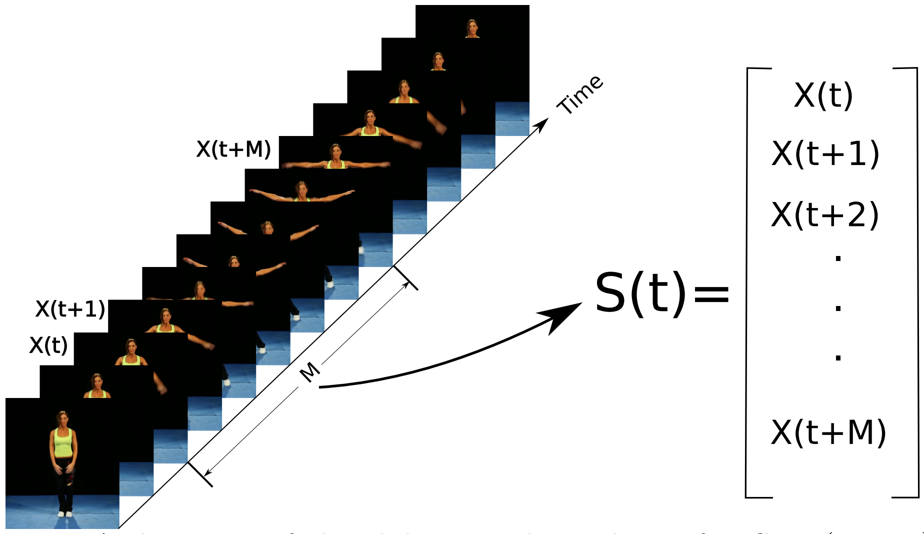


FIGURE 4.1: A depiction of the sliding window scheme for $S_{M,1}$ ($\tau = 1$). Each window holds a stack of frames, and each pixel within each frame is a dimension in a high dimensional Euclidean space. Sliding the window forward in time traces out a trajectory in that space.

We now look at the sliding window of a video time series, following (Definition 3):

$$\boldsymbol{S}_{M,\tau}(\boldsymbol{X}(t)) = \begin{bmatrix} \boldsymbol{X}(t) \\ \boldsymbol{X}(t+\tau) \\ \vdots \\ \boldsymbol{X}(t+M\tau) \end{bmatrix} \in \mathbb{R}^{W \times H \times (M+1)} \tag{4.1}$$

This can be viewed as the concatenation of the delay embeddings of each individual pixel in the video into one large vector. Figure 4.1 shows a visual depiction of this scheme for $\tau = 1$. This fits within our general definition of sliding windows for time series (Definition 3), but to our knowledge, this is a unique approach to studying video dynamics. Note that sliding windows are traditionally applied to 1D time series, which can be viewed as 1-pixel videos ($W = H = 1$) in this framework. Finally, since the pixel measurement locations are fixed, it is an "Eulerian" view into the dynamics of the video.

Derivative Filtering

To attempt to mitigate nonlinear drift, we implement a simple pixel-wise convolution by the derivative of a Gaussian for each pixel $X_i(t)$ in the original video before applying the delay embedding:

$$X_i(t) = X_i(t) * -at \exp^{-t^2/(2\sigma^2)} \tag{4.2}$$

This is a pixel-wise bandpass filter which could be replaced with any other bandpass filter leveraging application specific knowledge of expected frequency bounds.

4.1.2 Geometry of Sliding Window Videos

Though it may seem daunting compared to the 1D case, the geometry of the sliding window embedding shares many similarities for periodic videos, as shown in Tralie (2016). To see this, consider an example video that contains a set of K frequencies $\omega_1, \omega_2, ..., \omega_K$. Let the amplitude of the k^{th} frequency and i^{th} pixel be a_{ik}. For simplicity, assume that each is a cosine with a zero phase offset (this simplifies the math without loss of generality). Then the time series at pixel i can be written as

$$X_i(t) = \sum_{k=1}^{K} a_{ik}\cos(\omega_k t) \tag{4.3}$$

Grouping all of the coefficients together into a $(W \times H) \times N$ matrix A, the time series for the whole video can be written as

$$\boldsymbol{X}(t) = \sum_{k=1}^{K} \boldsymbol{A}^k cos(\omega_k t) \tag{4.4}$$

where $\boldsymbol{A}^k$ stands for the k^{th} column of A. Now, we construct a sliding window embedding as in Equation 4.1 for $\boldsymbol{S}(t)_{M,\tau}$:

$$\boldsymbol{S}(t)_{M,\tau} = \sum_{k=1}^{K} \begin{bmatrix} \boldsymbol{A}^k\cos(\omega_k t) \\ \boldsymbol{A}^k\cos(\omega_k(t+\tau)) \\ \vdots \\ \boldsymbol{A}^k\cos(\omega_k(t+M\tau) \end{bmatrix} \tag{4.5}$$

applying the cosine sum identity, we get

$$\begin{aligned} \boldsymbol{S}(t)_{M,\tau} = \sum_{k=1}^{K} &\begin{bmatrix} \boldsymbol{A}^k \\ \boldsymbol{A}^k\cos(\omega_k\tau) \\ \vdots \\ \boldsymbol{A}^k\cos(\omega_k M\tau) \end{bmatrix}\cos(\omega_k t) - \\ &\begin{bmatrix} 0^k \\ \boldsymbol{A}^k\sin(\omega_k\tau) \\ \vdots \\ \boldsymbol{A}^k\sin(\omega_k M\tau) \end{bmatrix}\sin(\omega_k t) \end{aligned} \tag{4.6}$$

Or written more succinctly

$$\boldsymbol{S}(t)_{M,\tau} = \sum_{k=1}^{K} \boldsymbol{u_k}\cos(\omega_k t) - \boldsymbol{v_k}\sin(\omega_k t) \tag{4.7}$$

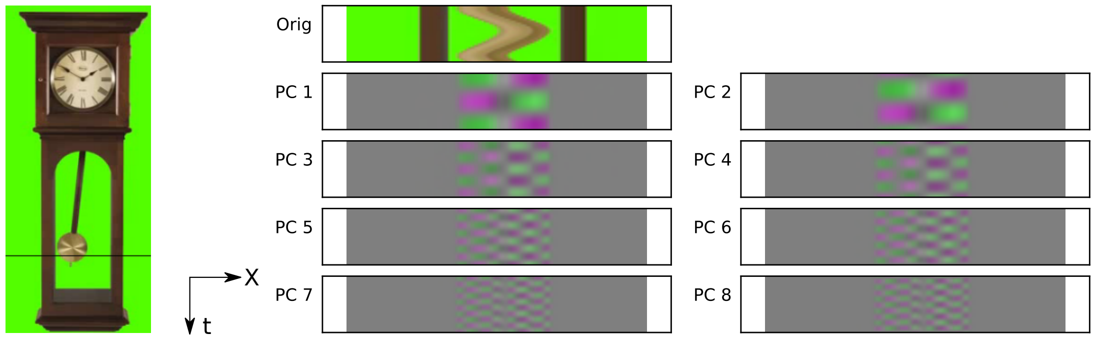


FIGURE 4.2: XT slices of the principal components on $\boldsymbol{S}_{M,1}$ for the synthetic video of an oscillating pendulum. The black line shows the x slice which is taken. The window size is chosen just under the period length, so the sine and cosine components of each harmonic are perpendicular. Hence, the principal components in each row correspond to the two axes of an independent ellipse in the delay embedding.

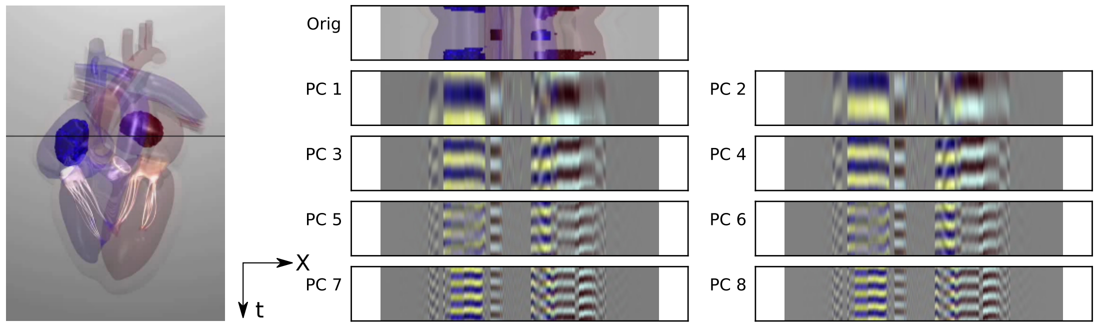


FIGURE 4.3: XT slices of the principal components of $\boldsymbol{S}_{M,1}$ on a video of a beating heart. Parameters are chosen as in Figure 4.2.

where $\boldsymbol{u_k}, \boldsymbol{v_k} \in \mathbb{R}^{(M+1)\times W \times H}$. In other words, the sliding window embedding is the sum of linearly independent ellipses, but this time in the space of $M+1$ frame videos at resolution $W \times H$. This is another way to see that the sliding window embedding of periodic signals lives on a hypertorus without using Takens' Delay Theorem directly, as in Section 2.2.2. The presence of the linearly independent sine and cosine terms also explains why two dimensions are needed for each periodic component. As shown in Perea and Harer (2015), when the window length is just under the length of the period, all of the $\boldsymbol{u_k}$ and $\boldsymbol{v_k}$ vectors become orthogonal, and so they can be recovered by doing PCA on $\boldsymbol{S}(t)_{M,\tau}$. Figure 4.2 shows the components of the first 8 PCA vectors for a horizontal line of pixels in a video of the synthetic pendulum with this setup. Note how the oscillations are present in time in the frames of the principcal components, as suggested by Equation 4.6. Similarly, Figure 4.3 shows the same for a video animation of a beating heart.

4.2 Prior Work in Periodic Videos

We now review some of the prior art on detecting and quantifying periodicities in videos, primarily in the computer vision literature. We hope both to create a taxonomy of techniques and to show the diverse array of applications that motivate the study of periodic videos. Later in Section 4.6, we will look more at prior work on quasiperiodic videos.

4.2.1 1D Surrogate Signals

One common strategy to quantify periodicity in videos is to derive a 1D function to act as a surrogate for the dynamics of the video, and then to use either frequency domain (Fourier) or time domain (autocorrelation, peak finding) techniques for quantifying periodicity of the resulting 1D signal.

One of the earliest works in this genre finds level set surfaces in a spatiotemporal "XYT" volume of video (all frames stacked on top of each other), and then uses curvature scale space on curves that live on these "spatiotemporal surfaces" as the 1D function (Allmen and Dyer (1990)). Polana and Nelson (1997) use Fourier Transforms on pixels which exhibit motion, and define a measure of periodicity based on the energy around the Fourier peak and its harmonics. Goldenberg et al. (2005) extract contours and find eigenshapes from the contours to classify and parameterize motion within a period. Frequency estimation is done by using Fourier analysis and peak finding on top of different 1D statistics derived from the contours, such as area and center of mass. In a similar approach, Ghaderian et al. (2011) perform foreground object segmentation and boundary contour finding, and create 4 1D signals which are the average distances of the points on the contour boundary to the 4 edges of a bounding box around the tracked blob. They then do autocorrelation on these signals to determine whether the underlying motion is periodic. Wehbe et al. (2015) turn the video into a 1D time series by simply averaging the intensity over all pixels in each frame, and they then apply the "YIN" autocorrelation method for fundamental frequency estimation, which came out of a long line of work in the audio analysis community (De Cheveigné and Kawahara (2002)). Finally, Yang et al. (2016) derive a 1D surrogate function based on mutual information between the first frame and subsequent frames, and then they look for peaks in the similarity function, with the help of a watershed method.

4.2.2 Self-Similarity Matrices

Another class of techniques relies on analyzing self-similarity matrices (SSMs) between frames, where similarity can be defined in a variety of ways. Seitz and Dyer (1997) track a set of points on a foreground object and compare them with an affine invariant similarity. They then extract so-called "period traces" of different orders k, which are functions that estimate the instantaneous "k-period" (length of the next k cycles) at frames in the video, and which can be extracted with snakes along

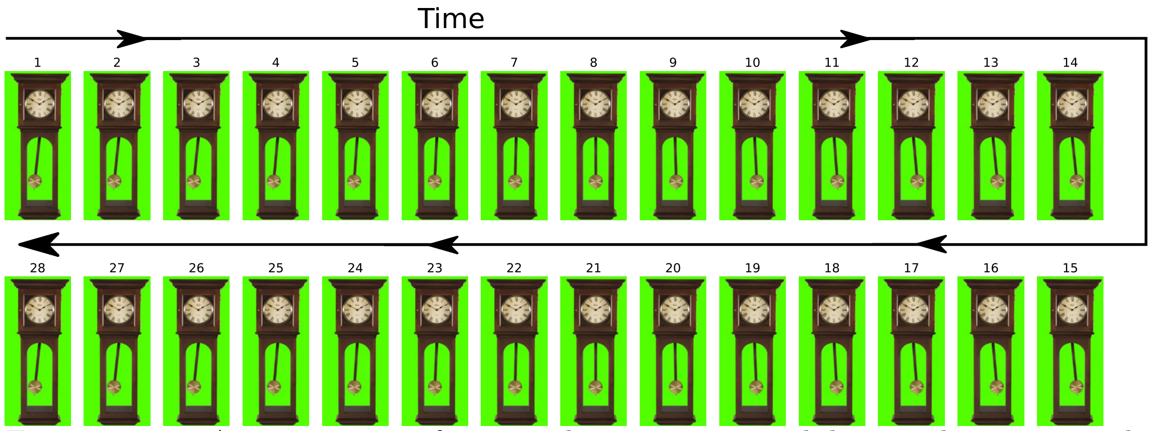


FIGURE 4.4: An animation of a periodic swinging pendulum. This is a simple example with mirror symmetry in its period which we will use throughout this work.

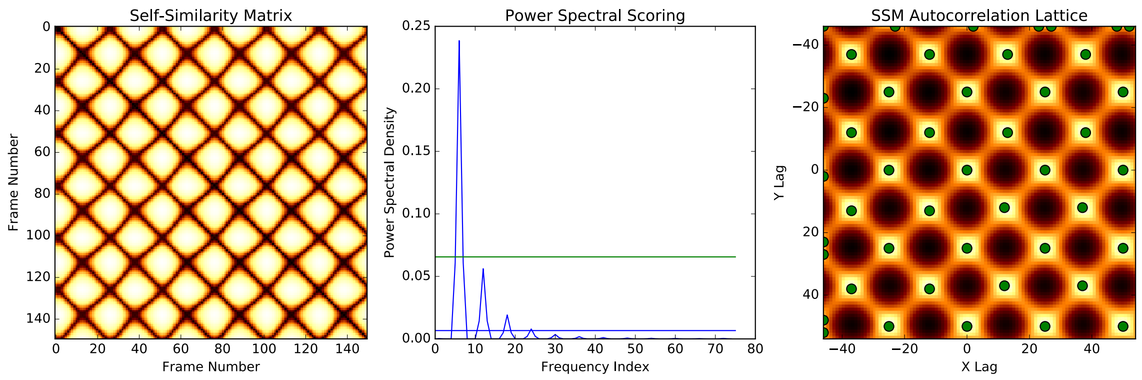


FIGURE 4.5: The method of Cutler and Davis (2000) on the pendulum video. The peak PSD frequency is above two times the standard deviation above the mean (green line, center figure). The maxes (green points) on the 2D autocorrelation of the SSM lie on a diamond shape pattern.

near-diagonal paths in a SSM. They then score these functions corresponding to a true cycle or not using the Kolmogorov-Smirinov test.

Another widely recognized technique for periodicity quantification by Cutler and Davis (2000) derives periodicity measures based on self-similarity matrices of L_1 pixel differences between all pairs of frames in a foreground video which have been lowpassed and resized to be in correspondence with each other. This technique has inspired a diverse array of applications, including analyzing the cycles of expanding/-contracting jellyfish (Plotnik and Rock (2002)), analyzing bat wings (Atanbori et al. (2013)), and analyzing videos of autistic spectrum children performing characteristic repetitive motions such as "hand flapping" (Kumdee and Ritthipravat (2015)). We now go into detail on this technique, since we will be comparing ours to it in Section 4.5. We use two videos as examples. Figure 4.4 shows a video of an oscillating pendulum, which has mirror symmetry in the second half of its period. That is,

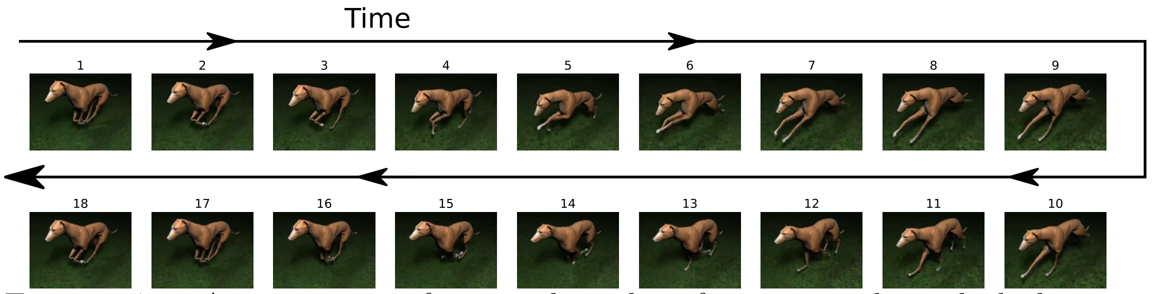


FIGURE 4.6: An animation of a periodic video of a running dog, which does not have mirror symmetry in the second half of its period.

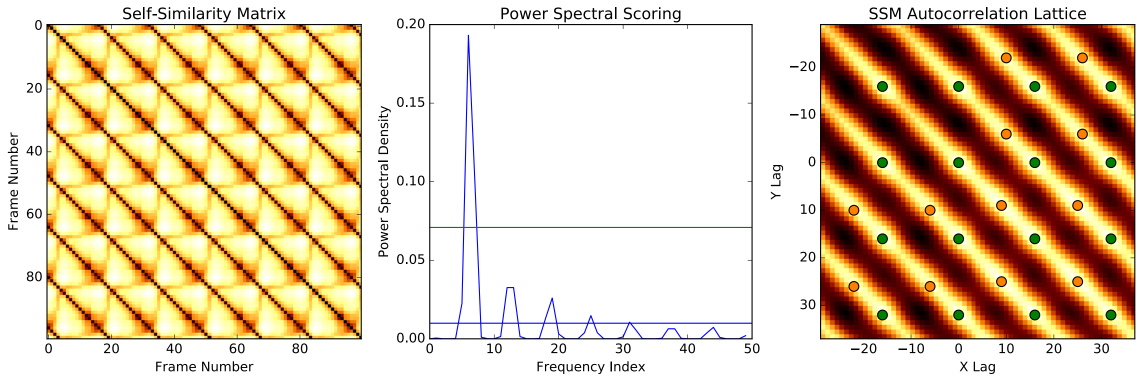


FIGURE 4.7: The method of Cutler and Davis (2000) on the video of a running dog. The peak PSD frequency is above two times the standard deviation above the mean. The maxes (green points) of the 2D autocorrelation of the SSM lie on a square pattern, since there is no symmetry. However, there are also spurious points along the diagonal (orange points), which can happen due to numerical instability for finding maxes along diagonal lines with a near constant max value (i.e. non-isolated critical points).

$f(t) = f(T - t)$, where T is the period. Based on the way the frames are arranged in Figure 4.4, this is equivalent to saying that the frames in the top row match the frames in the bottom row, or the video does the same thing "going there" as "on the way back." By contrast, as shown in Figure 4.6, a running dog does not have mirror symmetry in its period. These observations were central to the work in Cutler and Davis (2000), whose primary application domain was surveillance videos of residential areas, in which a walking person has approximate mirror symmetry a running dog does not.

Figure 4.5 shows the technique of Cutler and Davis (2000) on the mirror symmetric pendulum, and Figure 4.7 shows their technique on the running dog. There are two different ways they quantify periodicity from a self-similarity matrix. The first is a frequency domain technique, where they compute the power spectral density of each column (row) of the SSM after linearly de-trending and applying a Hann window. They then take the average power spectral density and compute its peak.

If the peak is more than twice the standard deviation above the mean across the power spectrum, then they deem the video periodic. As the authors warn, and as we show in Section 4.5, this method has a high susceptibility to false positives. This motivated the design of a more robust technique in Cutler and Davis (2000), which is actually a time domain technique. It works by finding peaks in the 2D normalized autocorrelation of the SSMs, which have been Gaussian smoothed. For videos with mirror symmetry, the peaks will lie on a diamond lattice (Figure 4.4), while for videos without mirror symmetry, they will lie on a square lattice (Figure 4.6). After peak finding within neighborhoods, one simply searches over all possible lattices at all possible widths to find the best match with the peaks. Since each lattice is centered at the autocorrelation point $(0, 0)$, no translational checks are necessary. We will give more details on an objective function for ranking periodicity with this technique in Section 4.5.

4.2.3 Miscellaneous Techniques for Periodic Video Quantification

There are also a number of works that don't fall into the two categories above. Some works focus solely on walking humans. Niyogi et al. (1994) look at the "braiding patterns" that occur in XYT slices of videos of walking people. Huang et al. (2016) perform blob tracking on foreground of walking person, and use ratio of second eigenvalue to first eigenvalue of PCA on that blob. A smoothed version of this has maxes twice during each period during what they call "lambda frames," when either the left or right foot is sticking out in front. Assuming the person is walking in a straight line, they show how to use walking trajectory estimates to do camera calibration.

For more general periodic videos, Wang et al. (2009) make a codebook of visual words and look for repetitions within the string that they get from their visual alphabet. Levy and Wolf (2015) take a deep learning approach to counting the number of periods that occur in a video segment. They use a 3D convolutional neural network on spatially downsampled, non-sequential regions of interest, which are uniformly spaced in time (sliding window with $\tau > 1$), to estimate the length of the cycle. Interestingly, they are able to train these networks on fully synthetic videos of low frequency patterns moving against a white noise background. Finally, perhaps the most philosophically similar work to ours is the work of Vejdemo-Johansson et al. (2015), who use cohomology to find maps of MOCAP data to the circle for parameterizing periodic motions, though this work does not use delay embeddings or provide a way to quantify how periodic something is.

4.2.4 Our Work

While most of the works we surveyed so far address the problem of period finding in videos, we are not aware of any works which take an explicitly *geometric* perspective to this problem like ours. Geometry gives us a nice way to quantify periodicity by measuring the roundness of sliding window embeddings, as shown in Section 4.5. Crucially, it also gives us a way to measure *quasiperiodicity* using topology, which is not at all obvious with standard techniques. As we show in Section 4.6, this simplifies the pipeline for detecting anomalies in vocal cord videos.

We also note that, unlike Cutler and Davis (2000), in our sliding window scheme, there is no need to distinguish between videos with and without mirror symmetry. This is because, as shown in Figure 4.5, adding the delay frames destroys the symmetry and turns a path-like embedding into a loop embedding, if the window length is close to the length of the period. In other words, the state space is able to uniquely represent all phases in the period with an adequate delay. This is a crucial preprocessing step for us, since we use measures of roundness of our embedding to measure periodicity, and videos with mirror symmetry do not form a loop. Of course, we can no longer differentiate between videos with and without mirror symmetry, but we have a unified framework for quantifying periodicity, and we can also address quasiperiodicity. In this way, our work is complementary to Cutler and Davis (2000).

Finally, in contrast to both frequency and time domain techniques, our method does not care if the period is an integer number of samples.

4.3 Theoretical Analysis of Eulerian Periodic Videos

In this section, we explore theoretically the geometry generated from the type of sliding window embeddings we have defined. The fact that we are using *Eulerian coordinates*, or measurement locations which don't change and which witness phenomena passing back and forth, rather than tracking the motion, has a crucial impact on the resulting shapes.

4.3.1 Basic Model with Mirror Symmetry

We now introduce a basic model of periodic videos. First, define the following terms.

- Define a fixed radian frequency $\omega \in \mathbb{R}^+$ and a fixed phase shift $\phi \in \mathbb{R}$
- Let I be some indexing set into the N pixels of a video frame $\boldsymbol{X}(t)$, and let $X_i(t)$, $i \in I$, be a 1-parameter function describing how a pixel at location i evolves in time
- Let $g_i(t)$ be an arbitrary 1D function associated with pixel i
- Let A be a constant associated with amplitude of oscillation.

Given all of this, the model for motion is that $X_i(t)$ take the form

$$X_i(t) = g_i\left(A\cos\left(\frac{2\pi}{T}n + \phi\right)\right) \quad (4.8)$$

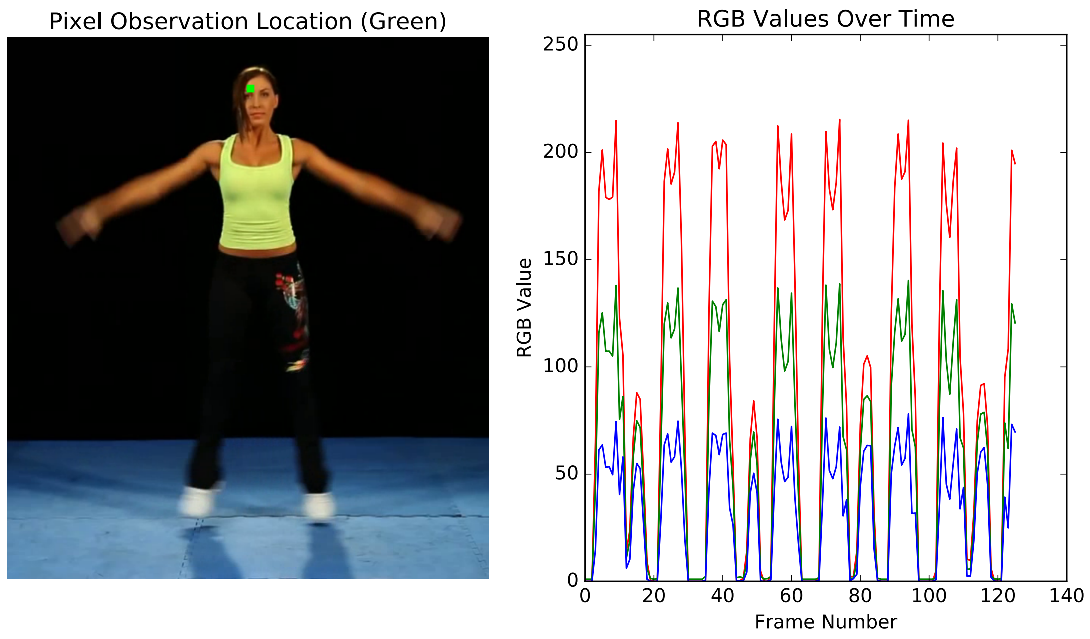


FIGURE 4.8: An example of an Eulerian pixel witnessing a foreground/background transition in a video of a woman doing jumping jacks. Red, green, and blue channels are plotted over time. These transitions induce a per pixel periodic signal with sharp transitions, which leads to lots of harmonic axes in the sliding window embedding.

In other words, each pixel is an arbitrary function composed with a scale of the same cosine. This allows our model to encompass non-sinusoidal repetitions at each pixel. In general, though the shape of the function at each pixel may be different, the functions across all pixels are globally *in phase* under this model. To see that this is a reasonable model, consider the example of a 1D video of length $A+B$ with amplitude of oscillation A. In terms of the model above, define the following:

$$g_i(t) = \text{rect}_B\left(t - i + \frac{A+B}{2}\right) \tag{4.9}$$

where

$$\text{rect}_B(t) = \left\{ \begin{array}{cc} 1 & |t| < \frac{B}{2} \\ 0 & \text{otherwise} \end{array} \right\} \tag{4.10}$$

Now let $X_i(t) = g_i(A\cos(t))$. This simulates a 1D solid line segment of length B oscillating left and right over a range of A in tune with a cosine, which is a simplified model of the kind of motion present in the pendulum video and many other natural videos where a solid object moves back and forth, but at the pixel level a pixel i measuring the foreground can suddenly jump to the background and then back again (Figure 4.8 shows an example in a real video). Figure 4.9 shows an example of the oscillating bar with $A = 100$, $B = 60$, and $T = 50$. Each column of the image

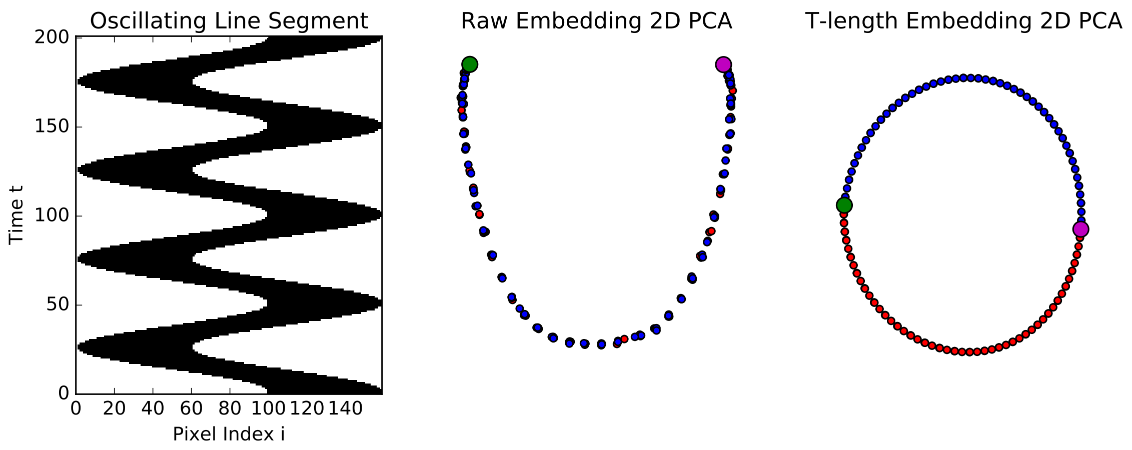


FIGURE 4.9: A synthetic 1D video of an oscillating line segment of length B oscillating over a range A. This can be described by the model put forth in this paper as $g_i(t) = \text{rect}_B\left(t - i + \frac{A+B}{2}\right)$. In this example $A = 100$, $B = 60$, $T = 50$, and $\phi = 0$. The video is shown going through 4 periods. The center plot shows PCA of the raw embedding, and the right plot shows PCA of a sliding window embedding. Points that correspond to the first half of a period are drawn in red, while points that belong to the second half of a period are drawn in blue (noise was intentionally added to the center plot because some points directly coincide in the sampling). A green circle is drawn over the point corresponding to the beginning of the period, and a magenta circle is drawn over the point halfway through the period.

on the left shows $X_i(t)$ for a fixed i. Note the similarity of this simulated example to the pendulum example in Figure 4.14.

One immediate observation is that there is mirror symmetry built into this model; that is

$$X_i(t) = X_i\left(T - \left(t + \phi\frac{T}{\pi}\right)\right) \tag{4.11}$$

In other words, each pixel repeats itself during the second half of its period, but in reverse. This means that a raw embedding (sliding window size of 1) traces out a topological path between two states X_a and X_b. During the first half of the period, the path goes between X_a and X_b, and during the second half of the period it follows the *exact same* path but in reverse from X_b to X_a. The middle column in Figure 4.9 shows PCA on this path, where X_a is drawn in green and X_b is drawn in magenta, points on the path from X_a to X_b are drawn in red, and points on the path from X_b to X_a are drawn in blue. They overlapped each other, so noise had to be added to see both red and blue points in some regions.

On the other hand, a sliding window size of appropriate length can turn this path into a loop by taking a different trajectory from X_b to X_a than was taken from X_a to X_b, as can be seen in column 3 of Figure 4.9. A similar observation was made and exploited in early work on video textures to properly capture the dynamics of videos being used as templates (Schödl et al. (2000)). Another way of saying this is, even though there are orders of magnitude more measurements than there would be in a 1D time series, they don't always jointly disambiguate states, so a frame by frame embedding without delay can still collapse parts of the state space.

4.3.2 The High Dimensional Geometry of Repeated Pulses

Even though our model is a composed function of a pure cosine, the loop that's formed is a *topological* loop whose geometry is much more complicated than a 2D ellipse that appears from a single cosine, as hinted at in Figure 4.2 and Figure 4.3. To see why, examine the Fourier transform of one of the pixels. First extract one pulse

$$f_i(t) = \left\{ \begin{array}{cc} X_i(t) & 0 \leqslant t \leqslant T \\ 0 & \text{otherwise} \end{array} \right\} \tag{4.12}$$

Then $X_i(t)$ can be rewritten in terms of the pulse as

$$X_i(t) = \sum_{m=-\infty}^{\infty} f_i(t - mT) \tag{4.13}$$

Since $X_i(t)$ repeats itself, regardless of what $f_i(t)$ looks like [1], *periodic summation* discretizes the frequency domain (Pinsky (2002))

$$\mathcal{F}\left\{X_i(t)\right\}(k) \propto \sum_{m=-\infty}^{\infty} \mathcal{F}(f_i(t))\left(\frac{m}{T}\right)\delta\left(\frac{m}{T} - k\right) \tag{4.14}$$

Switching back to the time domain, we can write $X_i(t)$ as

$$X_i(t) \propto \sum_{m=-\infty}^{\infty} \mathcal{F}\left(f_i(t)\right)\left(\frac{m}{T}\right) e^{i\frac{2\pi m}{T}t} \tag{4.15}$$

In other words, each pixel is the sum of some DC offset plus a (possibly infinite) set of harmonics at integer multiples of $\frac{1}{T}$. For instance, applying Equation 4.15 to a square wave of period T centered at the origin is a roundabout way of deriving the Fourier series

$$\sin\left(\frac{2\pi}{T}t\right) + \frac{1}{3}\sin\left(\frac{6\pi}{T}t\right) + \frac{1}{5}\sin\left(\frac{10\pi}{T}t\right) + \ldots \tag{4.16}$$

by sampling the sinc function $\sin(\pi T f)/(\pi f)$ at intervals of $m/2T$ (every odd m coincides with $\pi/2 + k\pi$, proportional to $1/k$, and every even harmonic is zero conciding with πk). In general, the sharper the transitions are in $X_i(t)$, the longer

[1] Note that periodic summation is more general than pulses which have mirror symmetry

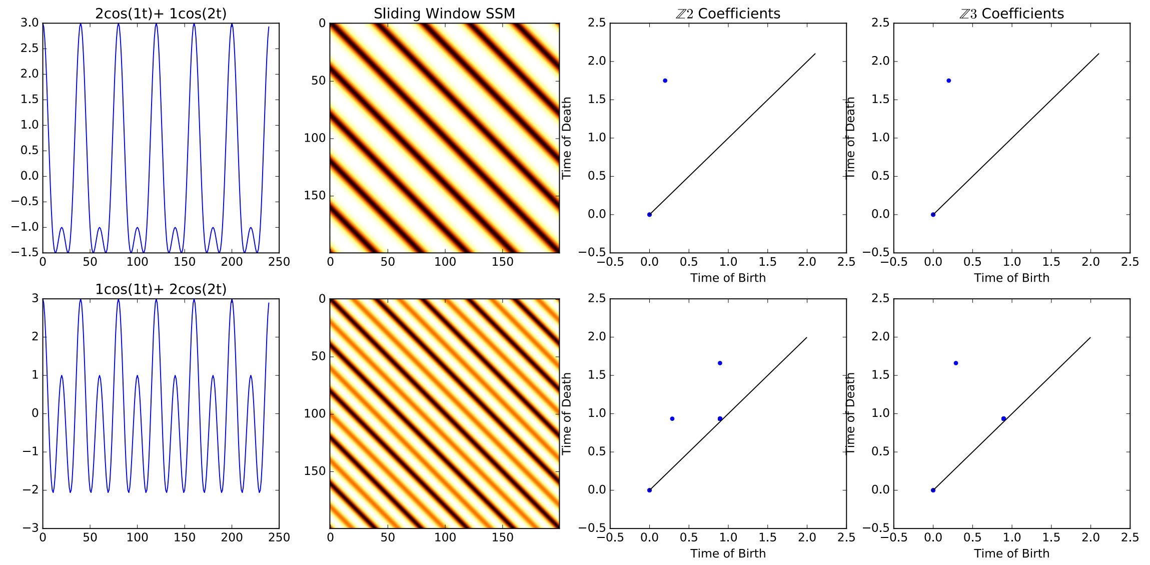


FIGURE 4.10: The persistence diagrams of sliding window embeddings of $g(t) = \cos(t) + 2\cos(2t)$ differ for different field coefficients, while they are the same for $f(t) = 2\cos(t) + \cos(2t)$, as shown by Perea and Harer (2015). The former has an SSM with alternating magnitude stripes, while the latter has an SSM with constant valued stripes.

the tail of $\mathcal{F}\{f_i(t)\}$ will be, and the more high frequency harmonics will exist in the embedding, calling for a higher delay dimension to fully capture the geometry, since every harmonic lives on a linearly independent ellipse. Similar observations about harmonics have been made in images for collections of patches around sharp edges (Yu et al. (2012), Figure 2).

4.3.3 The Möbius Strip Geometry of Harmonic Repeated Pulses

In addition to the harmonics that occur for every periodic signal with sharp transitions in Eulerian sliding window videos, there is also a special type of twisted geometry that occurs when there are harmonics of motion in the video itself. Perea and Harer (2015) first discovered the following fact about persistence diagrams of sliding window embeddings of signals with a harmonic at twice the base frequency:

Lemma 6. *The persistence diagrams of sliding window embeddings of dimension $\geqslant 4$ of $f(t) = a\cos(t) + b\cos(2t)$ differ for $\mathbb{Z}_2$ and $\mathbb{Z}_3$ coefficients if and only if $b > a$.*

Figure 4.10 shows an example with $f_2(t) = \cos(t) + 2\cos(2t)$. As with the sliding window embedding of any periodic signal, the sliding window embedding $f_2(t)$ lives on a hypertorus. In fact, it is isometric to the trajectory $e^{it} + 2e^{2it} \in \mathbb{C}^2$ on the flat torus in $\mathbb{R}^4$ with the right sliding window parameters. A similar phenomenon occurs for the following trajectory on the 1-2 torus knot embedded in $\mathbb{R}^3$

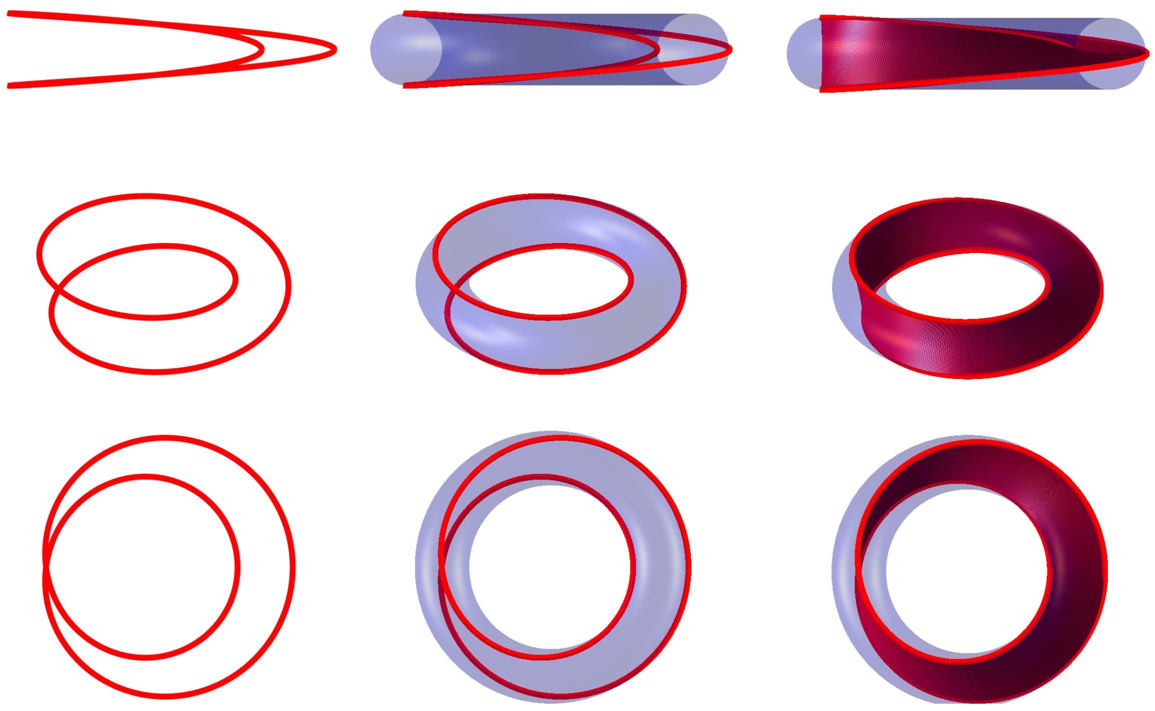


FIGURE 4.11: The sliding window embedding of $g(t) = \cos(t) + 2\cos(2t)$ can be identified with a 1-2 torus knot, which bounds a Möbius strip.

$$\boldsymbol{T}_{12}(t) = \begin{bmatrix} (2+\cos(t))\cos(2t) \\ (2+\cos(t))\sin(2t) \\ \sin(t) \end{bmatrix} \tag{4.17}$$

As shown in Figure 4.11, this curve is the boundary of a Möbius strip that lives inside of the torus. There is one homology class on the filled in Möbius strip which is homologous to its boundary using $\mathbb{Z}_3$ coefficients, since there is a homotopy between the Möbius strip and its boundary (Hatcher (2002)). This means that there is only ever one strongly persistent homology class with $\mathbb{Z}_3$ coefficients for a Möbius strip boundary. With $\mathbb{Z}_2$ coefficients, on the other hand, these two classes are not homologous. This means that the first class dies early and there is a second class that is born later. As can be seen in Figure 4.10, the one class that remains with $\mathbb{Z}_3$ coefficients has the same birth time as the class that's born first with $\mathbb{Z}_2$ coefficients and the same death time as the class that dies last. It is for this reason that Perea and Harer (2015) always use the $\mathbb{Z}_3$ persistence diagram to score periodicity.

In our sliding window videos, we have a similar phenomenon that can happen, but the amplitude ratios of the harmonics giving rise to the Möbius strip boundary differ. To see how this works, we first note one key property of a Möbius strip boundary $\boldsymbol{T}_{12}$ parameterized on an interval $t = [0, T]$ by Equation 4.17

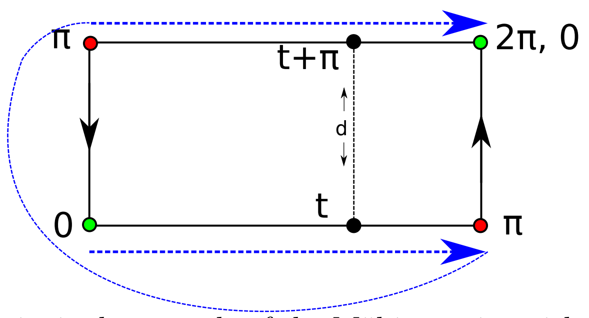


FIGURE 4.12: The principal rectangle of the Möbius strip, with the boundary drawn in blue (it jumps from the lower right corner to the upper right corner of this diagram because of the twist). A uniform Möbius strip has the property that $||\boldsymbol{X}(t) - \boldsymbol{X}(t + \pi)||_2$ is a constant d.

$$||\boldsymbol{T}_{12}(t) - \boldsymbol{T}_{12}(t - T/2)||_2 = ||\boldsymbol{T}_{12}(s) - \boldsymbol{T}_{12}(s - T/2)||_2 = d, \forall s, t \in \mathbb{R} \tag{4.18}$$

In other words, a parameterized loop $[0, 2\pi] \rightarrow \gamma(t)$ having the property that every point on the loop has a constant distance d to a point π away is a candidate for a loop that bounds a Möbius strip, and d is the width of said Möbius strip, as shown in Figure 4.12 [2]. For the sliding window videos, the geometry is in much higher dimensions and is difficult to visualize, but we simply check that the distances satisfy this property. To do this, we consider a simple model of harmonics at an Eulerian pixel time series that consists of thin Gaussian pulses, which model the smoothed derivative magnitudes of sharp transitions in videos (Equation 4.2)

$$h_\sigma(t) = \sum_{k=-\infty}^{\infty} ae^{-(t-kT)^2/(2\sigma^2)} + be^{-(t-kT/2)^2/(2\sigma^2)} \tag{4.19}$$

Figure 4.13 shows three different examples of these signals for the ratios $b/a = 1/2, 1, 2$. It is indeed the case that the sliding window embeddings line up at a local min distance of $T/2$ when the smaller pulses are aligned with the larger pulses. This leads to diagonal stripes of alternating magnitude, where the second stripe, corresponding to this local min distance, is always larger than the primary diagonal. Unlike the pure sinusoidal case, however, this geometry can show up at a variety of ratios b/a, not just $b/a > 1$, which is useful if we want to simply answer the question of whether or not there is a harmonic present.

[2] It can be readily verified using trig identities that $d = 2$ is the width of the Möbius strip whose boundary is parameterized by Equation 4.17

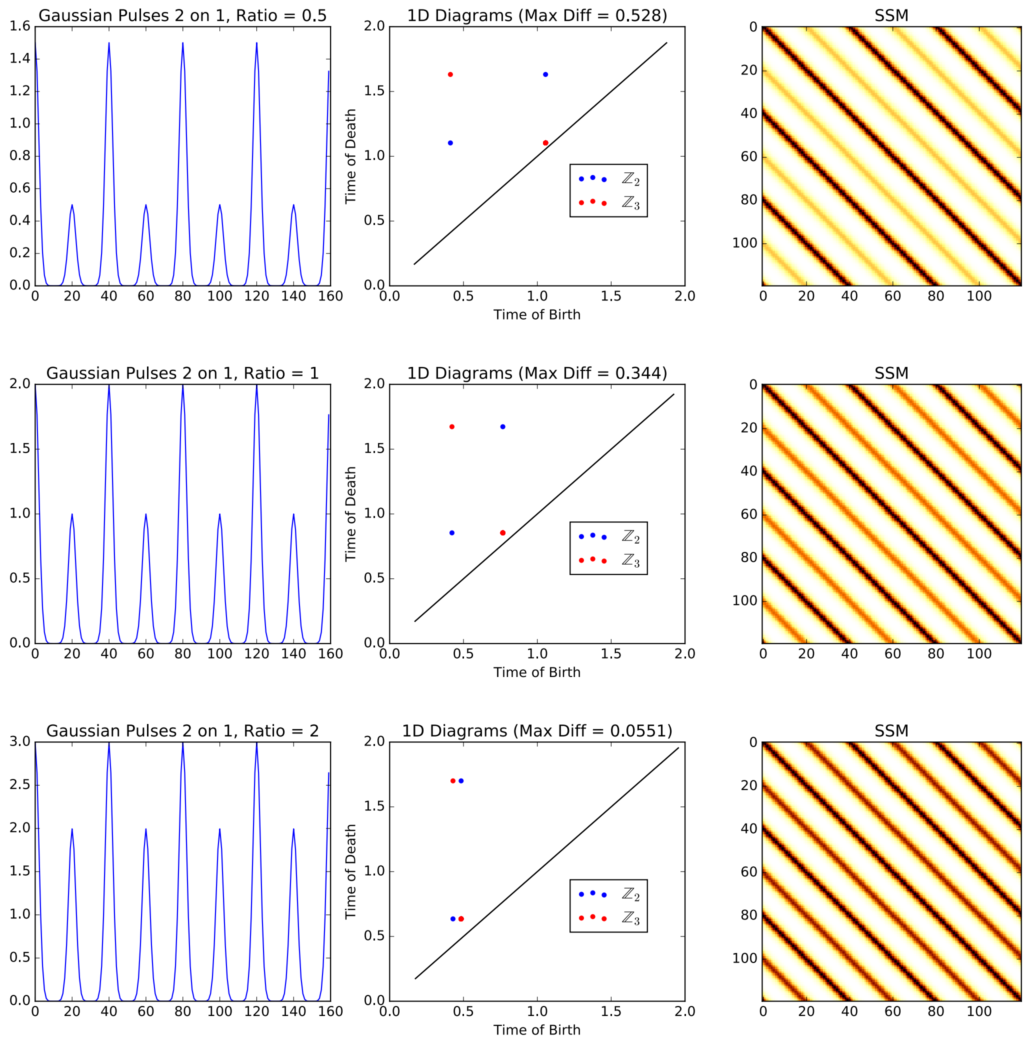


FIGURE 4.13: Normalized sliding window embeddings of Gaussian harmonic pulses $h_2(t)$ and $T = 40$, for different ratios between the harmonic and base frequency magnitude. Alternating diagonal stripes are visible in this SSMs in all cases.

4.4 Practical Issues in Sliding Window Videos

Now that we have set the stage for our work, we are ready to discuss some of the technical aspects of sliding window videos in more detail.

4.4.1 Reducing Memory Requirements with SVD

One potential drawback of the scheme that we have devised is the memory requirement and ensuing computational burden to construct and access the memory. Suppose we have a video which has been discretely sampled at N different frames at a resolution of $W \times H$, and we do a delay embedding with $M + 1$ frames. Assuming 32 bit floats per grayscale value, and $\tau = 1$, this will take up $4WHN(M+1)$ bytes. For instance, for a low resolution 200×200 video that is only 10 seconds long at 30fps, using a dimension of 30, this already exceeds 1GB of memory.

In our applications, however, we only need the pairwise distances between different delay vectors, which enables a few optimizations. First of all, for N points in $\mathbb{R}^{WH}$, where $N \ll WH$, the N points span a $N-1$ flat. This means that we can come up with a Euclidean isometry that only has $N \ll WH$ coordinates. In particular, let A be a $(W \times H) \times N$ matrix with each video frame along a column of A. Then perform the singular value decomposition $A = USV^T$, where U holds a set of orthogonal coordinate directions spanning the flat in question. Orthogonally project all video frames onto those directions:

$$U^T A = U^T USV = SV \tag{4.20}$$

and use the coordinates of the columns of SV instead of the pixels when comparing distances. This turns the delay embedding into

$$\hat{\boldsymbol{S}}_{M,\tau}(t) = \begin{bmatrix} U^T\boldsymbol{X}(t) \\ U^T\boldsymbol{X}(t+\tau) \\ \vdots \\ U^T\boldsymbol{X}(t+M\tau) \end{bmatrix} \in \mathbb{R}^{N \times N} \tag{4.21}$$

Note that SV can be computed by finding the eigenvectors of A^TA, which has a cost of $O(W^2H^2 + N^3)$, which is dominated by W^2H^2 if $WH \gg N$. In our example above, this alone reduces the memory requirements from 1GB to 10MB. Of course, this procedure is the most effective for short videos where there are actually many fewer frames than pixels, but this encompasses most of the examples in this work. In fact, the break even point for a 200x200 30fps video is 22 minutes. A similar trick was used in the classical work on "eigenfaces" (Turk and Pentland (1991)) when computing the principal components over a set of face images.

4.4.2 Delay Independent Memory And Computation with Diagonal Convolution

In addition to changing coordinate systems, we can exploit another fact if $\tau = 1$; that is, if delays are taken exactly on frames and no interpolation is needed. In this case, the squared Euclidean distance between two vectors $\boldsymbol{S}_{M,1}(i)$ and $\boldsymbol{S}_{M,1}(j)$ is

$$||\boldsymbol{S}_{M,1}(i) - \boldsymbol{S}_{M,1}(j)||_2^2 = \sum_{m=0}^{M} ||\boldsymbol{X}(i+m) - \boldsymbol{X}(j+m)||_2^2 \tag{4.22}$$

Let D_X^2 be an $N \times N$ matrix of all pairwise squared Euclidean distances between frames (possibly computed with the memory optimization in Section 4.4.1), and let D_Y^2 be an $(N-M) \times (N-M)$ matrix of all pairwise distances between delay frames. Then Equation 4.22 implies that D_Y^2 can be obtained from D_X^2 by a convolution by

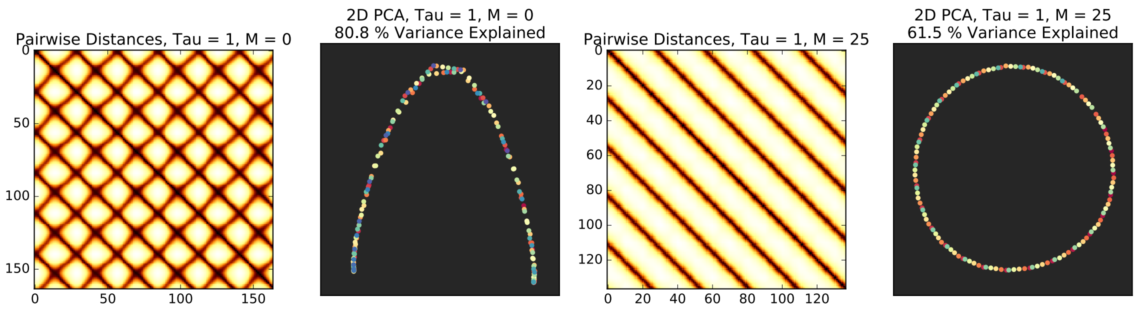


FIGURE 4.14: 2D PCA and pairwise distance matrices $D_{\boldsymbol{S}_{0,1}}$ and $D_{\boldsymbol{S}_{28,1}}$ for a video of a the oscillating pendulum. Bright colors indicate far distances and dark colors indicate near distances. This example clearly shows how adding a delay embedding is like performing block averaging along all diagonals of the pairwise distance matrices, and it gets rid of the mirror symmetry.

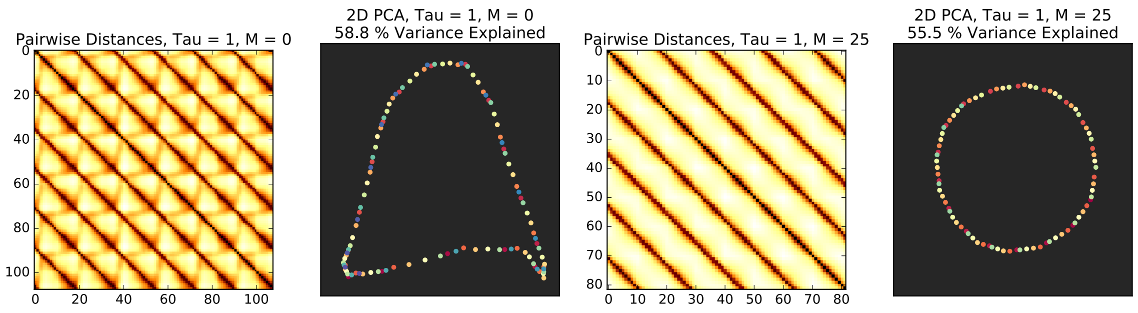


FIGURE 4.15: 2D PCA and pairwise distance matrices $D_{\boldsymbol{S}_{0,1}}$ and $D_{\boldsymbol{S}_{18,1}}$ for a video of a running dog. Even without the delay embedding ($M = 0$), the video frames still form a topological loop. But the period matched delay embedding cleans up the geometry and leads to a rounder loop, as seen in the resulting self-similarity matrix.

a "rect function," or a vector of 1s of length $M + 1$, over all diagonals in D_X^2 (i.e. a moving average). Since it is a vector of all ones, this can be implemented in time $O(N^2)$ with cumulative sums. This means that regardless of how many delays are chosen, the computation and memory requirements for computing D_Y^2 depend only on the number of frames in the video. Note also that D_Y can simply be computed by taking the entry wise square root of D_Y^2, another $O(N^2)$ computation. We note that a very similar scheme was used by Huang et al. (2010) when comparing distances of 3D shape descriptors in videos of 3D meshes.

Figure 4.14 shows the pendulum video with no delay and with a delay approximately matching the period. The effect of a moving average along diagonals with the delay is clearly visible. Even in Figure 4.15, where the diagonals dominate and the raw frames already form a loop embedding, this diagonal averaging cleans up the geometry.

4.4.3 Normalization And Scoring

We also need to perform a few normalization steps to enable fair comparisons between videos that may be at different resolutions or which have a different range in periodic motion either spatially or in intensity. First, we perform a "point-center and sphere normalize" vector normalization which was shown by Perea and Harer (2015) to have nice theoretical properties (Definition 4). Re-iterating here for convenience in the context of sliding window videos:

$$\boldsymbol{S}_{M,\tau}(t) = \frac{\boldsymbol{S}_{M,\tau}(t) - (\boldsymbol{S}_{M,\tau}(t)^T\mathbf{1})\mathbf{1}}{||\boldsymbol{S}_{M,\tau}(t) - (\boldsymbol{S}_{M,\tau}(t)^T\mathbf{1})\mathbf{1}||_2} \tag{4.23}$$

where $\mathbf{1}$ is a $WH(M+1) \times 1$ vector of all ones. In other words, subtract off the mean of each component of each vector, and scale each vector so that it has unit norm (i.e. lives on the unit hypersphere in $\mathbb{R}^{WH(M+1)}$). Subtracting off the mean of each component will cancel out additive linear drift on top of the periodic motion, while scaling takes care of resolution / magnitude differences. Note that we can still use the memory optimization in Section 4.4.1, but we can no longer use the optimizations in Section 4.4.2 since each window is normalized independently.

Once we have normalized in this way, we can score the videos for periodicity, quasiperiodicity, and harmonics based on the geometry. Let $mp_i(\text{dgm}_{p,k})$ be the i^{th} largest persistence in the persistence diagram of homology dimension p with field coefficients $\mathbb{Z}_k$. We come up with the following scores

1. *Periodicity Score (PS)*

$$PS = \frac{1}{\sqrt{3}} \max\big(mp_1(\text{dgm}_{1,2}), mp_1(\text{dgm}_{1,3})\big) \tag{4.24}$$

Like Perea and Harer (2015), we exploit the fact that a unit circle with infinitely many points has a death time of $\sqrt{3}$, since this is the limit shape of a normalized perfectly periodic sliding window video with infinitely many frames.

2. *Quasiperiodicity Score (QPS)*

$$QPS = \sqrt{\frac{mp_2(\text{dgm}_{1,3})mp_1(\text{dgm}_{2,3})}{3}} \tag{4.25}$$

This score is designed with the torus in mind. We score based on the second largest 1D persistent dot *times* the largest 2D persistent dot, since we want a shape that has two core circles and encloses a void to get a large score. For a Čech Complex, based on the K*ü*nneth theorem of homology, the 2-cycle (void) should die the moment the smallest 1-cycle dies.

3. *Modified Periodicity Score (MPS)*
$$MPS = \frac{1}{\sqrt{3}} \max\left(mp_1(\mathrm{dgm}_{1,2} - mp_2(\mathrm{dgm}_{1,2}), mp_1(\mathrm{dgm}_{1,3}) - mp_2(\mathrm{dgm}_{1,3})\right) \tag{4.26}$$
We design a periodicity score which should be lower for quasiperiodic videos, to help differentiate them from periodic

4. *Harmonic Score (HS)* If $mp_1(\mathrm{dgm}_{1,3}) > 0$
$$HS = 1 - mp_1(\mathrm{dgm}_{1,2})/mp_1(\mathrm{dgm}_{1,3}) \tag{4.27}$$
Otherwise, if $HS = 0$. We design this score to indicate the presence of a harmonic. If the maximum persistence 1-cycle goes up when changing from $\mathbb{Z}_2$ to $\mathbb{Z}_3$ coefficients, this is indicative of a harmonic, as shown in Section 4.3.3

We use these scores to quantify videos of oscillating vocal folds in Section 4.6, and we use the periodicity score to rank videos in Section 4.5.

4.4.4 Window Size

As we have shown, in video, the pixels may or may not give unique information to reconstruct the state space, so we may still need to use delay (Figure 4.14). But what should M actually be in the worst case, if the dimensions of the state space are unknown, or in our case, if the number of independent sinusoids in the video is unknown? One common strategy is the so-called "false nearest-neighbors" scheme of Kennel et al. (1992). The idea is to keep track of the K nearest neighbors of each delay point, and if they change as M is increased, then the prior estimates for M were too low. This algorithm was used in recent work on video dynamics by Venkataraman and Turaga (2016), for instance.

Even if we can estimate M, however, how do we choose the step size τ? As shown by Perea and Harer (2015), the sliding window embedding of periodic signals is roundest when the window size, $M\tau$, satisfies the following relation:
$$M\tau = \frac{\pi k}{L}\left(\frac{M}{M+1}\right) \tag{4.28}$$
where L is the length of the period and k is some positive integer. Noting that the period length is $2\pi/L$, this peaks when the window size is just under half the period, just under the period, just under 1.5x the period, etc. To verify this experimentally, we analyzed the maximum persistence versus window size in the pendulum video (Figure 4.16). We fixed some sufficiently large M and chose τ to be L/M for a given period L, so that the dimension was fixed. Fixing the dimension helps to control for effects that could arise from the curse of dimensionality, such as points spreading apart, which would lead to a later birth time of the maximum cycle class. The peaks in roundness occur where expected, so when we have a rough estimate of the period we are looking for, we fix the window size just under that period.

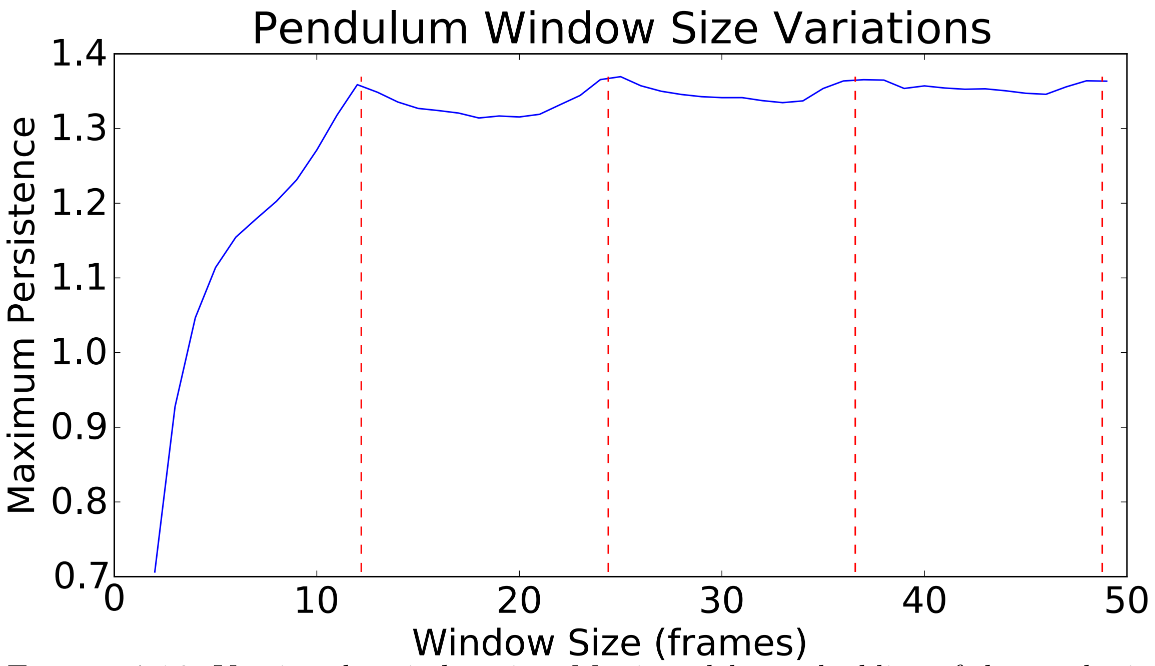


FIGURE 4.16: Varying the window size, $M\tau$, in a delay embedding of the synthetic pendulum video, which has a period length of 25 frames. Red dashed lines are drawn at the window lengths that would be expected to maximize roundness of the embedding for that period length based on theory in Perea and Harer (2015).

4.4.5 Fundamental Frequency Estimation

Though Figure 4.16 suggests robustness to window size as long as the window is more than half of the period, we may not know what that is in practice. To automate window size choices, we do a coarse estimate using fundamental frequency estimation techniques on a 1D surrogate signal. To get a 1D signal, we extract the first coordinate of autotuned diffusion maps with $\kappa = 0.1$ (Section 2.3.4) on the raw video frames (no delay) after taking a smoothed time derivative (Equation 4.2). Note that a similar diffusion-based method was also used in recent work by Yair et al. (2016) to analyze the frequency spectrum of a video of an oscillating 2 pendulum + spring system in a quasiperiodic state. Once we have the diffusion time series, we then apply the simple and elegant normalized autocorrelation method of Mcleod and Wyvill (2005) to estimate the fundamental frequency. In particular, given a discrete signal x of length N, define the autocorrelation as

$$r_t(\tau) = \sum_{j=t}^{t+N-1-\tau} x_j x_{j+\tau} \tag{4.29}$$

However, as observed by De Cheveigné and Kawahara (2002), a more robust function for detecting periodicities is the *squared difference function*

$$d_t(\tau) = \sum_{j=t}^{t+N-1-\tau} (x_j - x_{j+\tau})^2 \tag{4.30}$$

this can be rewritten as

$$d_t(\tau) = m_t(\tau) - 2r_t(\tau) \tag{4.31}$$

where

$$m_t(\tau) = \sum_{j=t}^{t+N-1-\tau} (x_j^2 + x_{j+\tau}^2) \tag{4.32}$$

This is in fact the squared Euclidean distance between two sliding windows of length $(N-1-\tau)$ offset by τ, so it is very similar philosophically to our time-ordered point cloud processing pipeline, except in this case it's a time-ordered point cloud on top of a function derived from another time-ordered point cloud (a sliding window on sliding windows). Finally, Mcleod and Wyvill (2005) suggest normalizing this function to the range $[-1, 1]$ to control for window size and to have an interpretation more like a Pearson correlation coefficient:

$$n_t(\tau) = 1 - \frac{m_t(\tau) - 2r_t(\tau)}{m_t(\tau)} = \frac{2r_t(\tau)}{m_t(\tau)} \tag{4.33}$$

The fundamental frequency is then the inverse period of the largest peak in n_t which is to the right of a zero crossing. The zero crossing condition helps prevent an offset of 0 from being the largest peak. Defining the normalized autocorrelation as in Equation 4.33 has the added advantage that the value of $n_t(\tau)$ at the peak can be used as a way of scoring periodicity, which the authors call "clarity." Values closer to 1 indicate more perfect periodicities. Note that this technique will sometimes pick integer multiples of the period. To mitigate this, we multiply $n_t(\tau)$ by a slowly decaying envelope which is 1 for 0 lag and 0.9 for the maximum lag to emphasize smaller periods. Figure 4.17 shows the result of this algorithm on a periodic video, and Figure 4.18 shows the algorithm on an irregular video.

4.5 Ranking Videos by Periodicity

Now that we have established our full pipeline, we would like to verify that rankings obtained from our periodicity score (Equation 4.24) agree with how humans rank videos by periodicity. We created a small dataset of 20 different creative commons videos, which were each 5 seconds at 30 frames per second. Some of them seem periodic, such as a person waving hands, a beating heart, and spinning carnival rides. Some of them seem nonperiodic, such as explosions, a traffic cam, and drone view of a boat sailing. And some of them are in between, such as the pendulum

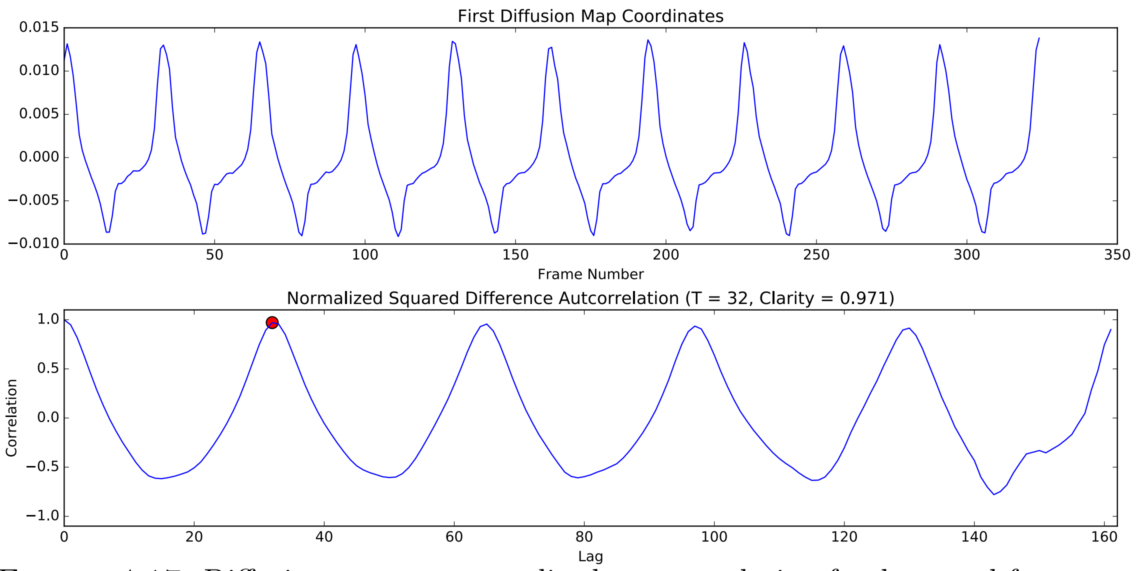


FIGURE 4.17: Diffusion maps + normalized autocorrelation fundamental frequency estimation in a video of vibrating vocal folds (Section 4.6). The chosen period length is 32, as indicated by the red dot over the peak. This matches with the visually inspected period length.

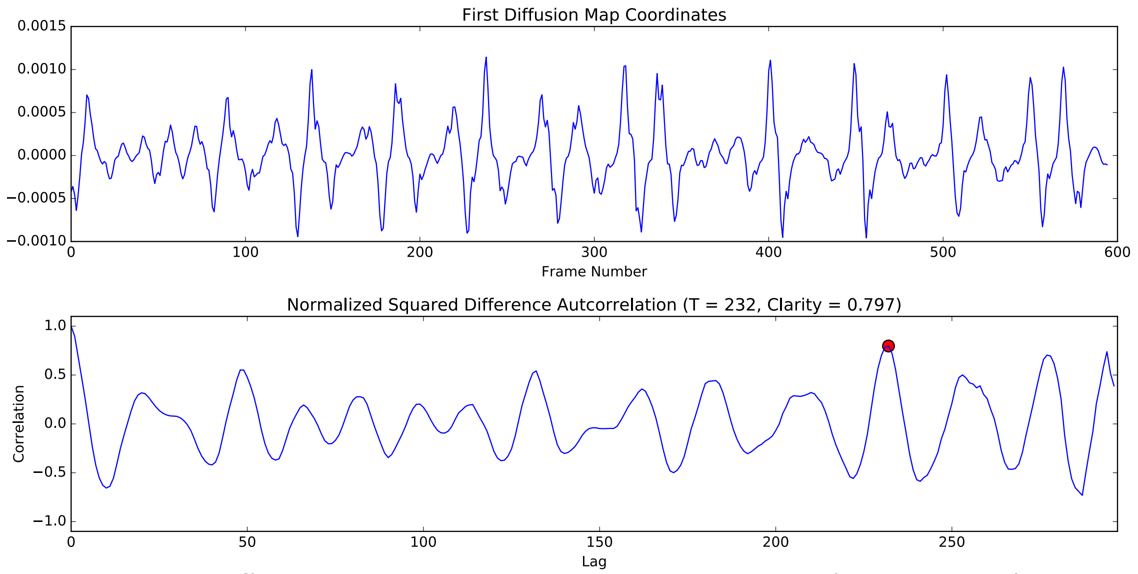


FIGURE 4.18: Diffusion maps + normalized autocorrelation fundamental frequency estimation on a video of vibrating vocal folds with irregular oscillations (Section 4.6).

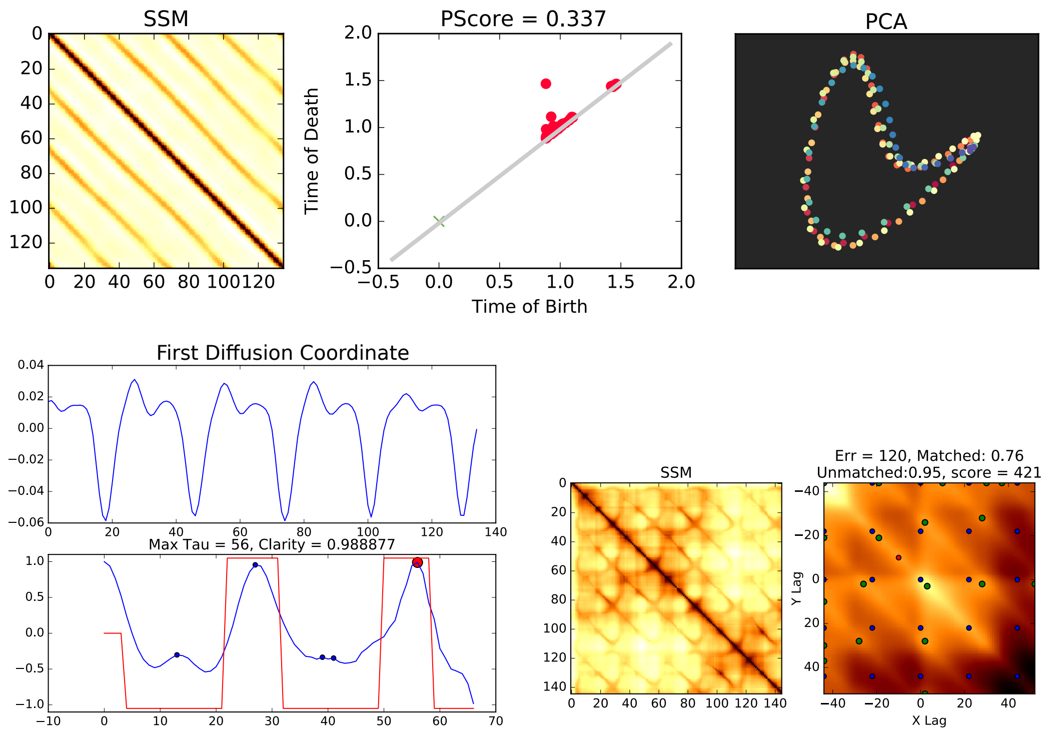


FIGURE 4.19: An example of the $\mathbb{Z}_3$ TDA score (top), the Cutler and Davis (2000) score (bottom left, matched peaks in green and lattice in blue), and the clarity score (bottom right) on a periodic video of a man waving his arms from the KTH dataset (Schuldt et al. (2004)).

video with simulated camera shake by convolving with per-frame directed random walks (see Delbracio and Sapiro (2015)).

4.5.1 Automated Techniques for Ranking

We use three different classes of techniques for machine ranking of periodicity.

TDA-Based Technique

We sort the videos in decreasing order of *periodicity score* (Equation 4.24). We fix the window size at 20 frames and the embedding dimension at 20 frames (which is enough to capture 10 strong harmonics). We also apply a time derivative of width 10 to every frame. We report two rankings, one for the periodicity score on $\mathbb{Z}_2$ coefficients, and one for the periodicity score on $\mathbb{Z}_3$ coefficients.

Cutler-Davis Technique

We use the techniques of Cutler and Davis (2000), as described in Section 4.2.2. We report two sets of rankings. The first is based on the number of standard deviations

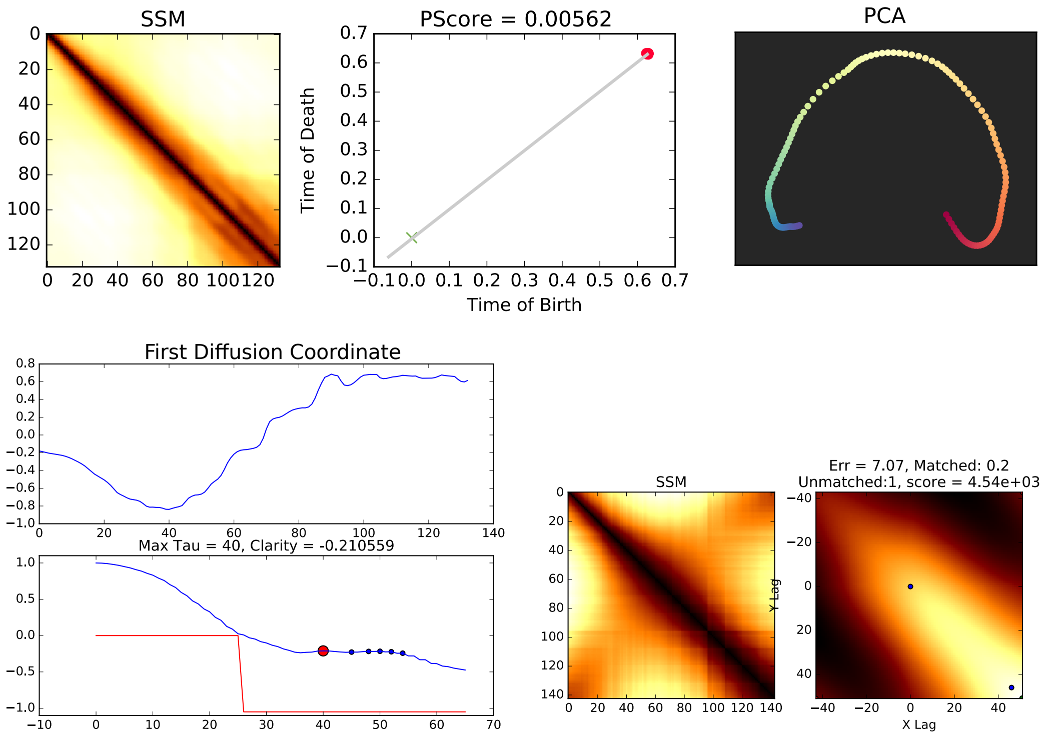


FIGURE 4.20: An example of the $\mathbb{Z}_3$ TDA score (top), the Cutler and Davis (2000) score (bottom left, matched peaks in green and lattice in blue), and the clarity score (bottom right) on a video of an explosion, which is nonperiodic.

above the mean that the largest peak is in the average power spectral density of the video SSM (Figure 4.5, center). Higher values indicate stronger frequency peaks, so we sort in descending order to get a ranking with most periodic videos first. The second score is based on the autocorrelation lattice (Figure 4.5, right). We modify the binary periodic/non-periodic score provided in Cutler and Davis (2000) as follows. Let E be sum of the Euclidean distances of the matched peaks in the autocorrelation image to the best fit lattice, let r_1 be the proportion of lattice points that have been matched, and let r_2 be the proportion of peaks which have been matched to a lattice point. Then the final periodicity score is given as

$$\frac{(1 + E/r_1)}{(r_1 r_2)^3} \tag{4.34}$$

A lattice which fits the peaks perfectly ($r_1 = 1$) with no error ($E = 0$) and no false positive peaks ($r_2 = 1$) will have a score of 1, and any video which fails to have a perfectly matched lattice will have a score greater than 1. Hence, we sort in increasing order of the score to get a ranking.

As we will show, this technique agrees the second best with humans after the TDA ranking. One of the main drawbacks is numerical stability of finding maxes in non-isolated critical points around nearly diagonal regions (Figure 4.7), which will erroneously inflate the score. Also, the lattice searching only occurs over an integer grid, but there may be periods that aren't integer number of frames, so there will always be a nonzero E for such videos. By contrast, our sliding window scheme can work for any real valued period length.

Diffusion Maps + Normalized Autocorrelation "Clarity"

Finally, we apply the technique from Section 4.4.5 to get an autocorrelation function, and we report the value of the maximum peak of the normalized autocorrelation to the right of a zero crossing, referred to as "clarity" by Mcleod and Wyvill (2005). Values closer to 1 indicate more perfect repetitions, so we sort in descending order of clarity to get a ranking.

Examples

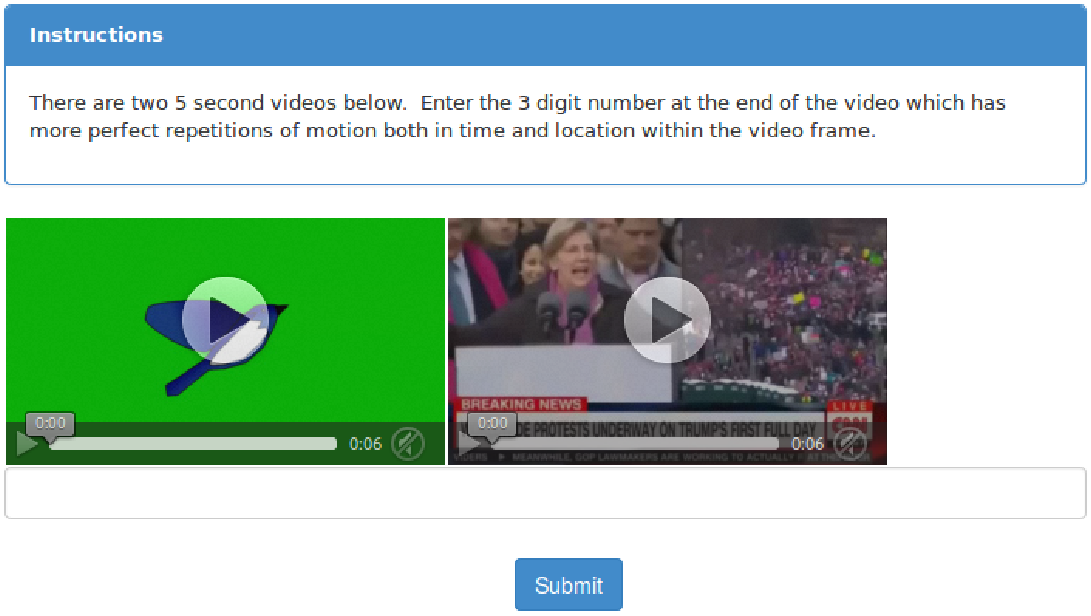
Figure 4.19 shows an example of these three different techniques on a periodic video. There is a dot which rises above the diagonal in the persistence diagram, a lattice is found which nearly matches the critical points in the autocorrelation image, and autocorrelation function on diffusion maps has a nice peak. By contrast, for a non-periodic video (Figure 4.19), there is hardly any persistent homology, there is no well matching lattice, and the first diffusion coordinate has no apparent periodicities.

4.5.2 Human Hodge Rank Aggregation

We now describe how we aggregate the opinions of many human users to get a human ranking, which we can compare to the machine rankings. Humans are notoriously bad at coming up with global rankings of sets larger than about 5 to 7 (Miller (1956)), so we opt for an experiment where humans are only presented with two videos at a time, and they simply make a binary decision about which one is more periodic. This has the added benefit of enabling a very simple interface, as show in in Figure 4.21. In our experiment, we present each pair of videos in the set of 20, $\binom{20}{2} = 190$, each to three different users, for a total of 570 pairwise rankings. We ended up with 15 unique workers doing this in our experiment[3].

To aggregate this information into a global ranking which is "as consistent as possible" with the pairwise rankings, we implement a technique known as *Hodge rank aggregation* (Jiang et al. (2011)). This technique expresses consistency of rankings in topological language, using *cohomology*, the dual of homology. Describing fully the mathematics is beyond our scope, but the idea is that a set of inconsistent rankings, such as $a > b, b > c, c > a$, when considered as *1-forms)* (signed functions on directed edges between a, b, and c) lead to a loop, or "co-cycle." In terms which may be more familiar to engineers, a set of rankings can be thought of as a vector field over a

[3] We gratefully acknowledge the time and effort put in by anonymous people on the Internet to helps us to create an unbiased global ranking!


Instructions

There are two 5 second videos below. Enter the 3 digit number at the end of the video which has more perfect repetitions of motion both in time and location within the video frame.

FIGURE 4.21: The interface that humans are given on the Amazon Mechanical Turk for pairwise ranking videos by periodicity. We use a seeded random number generator to come up with 3 digit numbers that flash at the end of the videos to ensure users actually spend 5 seconds watching them.

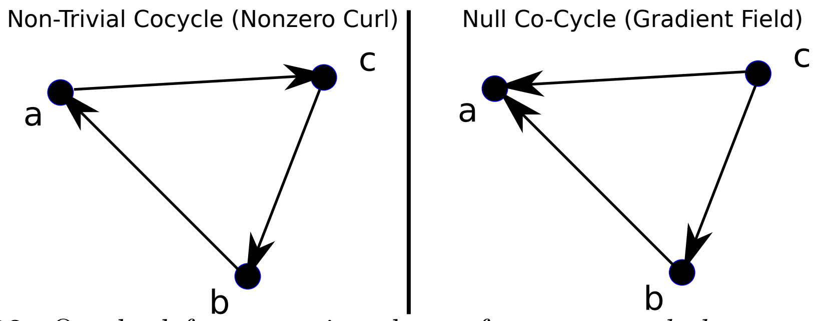

Non-Trivial Cocycle (Nonzero Curl)
a
b
c
Null Co-Cycle (Gradient Field)
a
b
c

FIGURE 4.22: On the left, expressing the preferences $a > b$, $b > c$, and $c > a$ as 1-forms (vectors) leads to a vector field which has curl, which is one way of expressing the Condorcet paradox, or the fact that it is impossible to come up with a scalar function on a, b, and c that gives rise to those preferences when considering uphill flows. On the right, swapping the third preference to $a > c$ leads to a globally consistent set of rankings, from which it is possible to find a scalar field whose gradient gives rise to those arrows (e.g. $a = 3, b = 2, c = 1$)

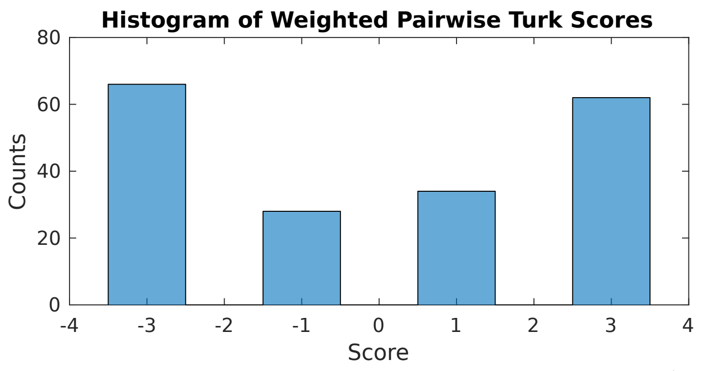


FIGURE 4.23: The histogram of scores that the workers on the Amazon Mechanical Turk gave to all pairwise videos.

set of objects, and an inconsistent ranking has nonzero *curl* (i.e. nontrivial cocycle class), as illustrated in Figure 4.22. This is also known as the *Condorcet paradox* in the context of rankings. Hodge rank aggregation is able to decompose all of the human rankings into the direct sum of three 1-forms:

$$\mathcal{C}^1 = -\nabla(s) \bigoplus H \bigoplus I \tag{4.35}$$

where s is a scalar field, whose gradient is curl-free (fully consistent), H expresses "harmonic cocycles," and I expresses local cocycles. The scalar function s on all of the vertices can be used to give a global ranking, as it is curl-free. It is, in the least squares sense, the scalar function whose gradient best matches the set of preferences given. For the leftover terms, the norm of H is a measure of global inconsistencies in the ranking, and the norm of I is a measure of local inconsistencies. All of these quantities can be computed with a sparse linear least squares system.

Note that the 1-forms that we feed to the algorithm are weights based on the pairwise rankings returned from the Turk. If video a is greater than video b, then the edge ab gets a weight of +1, or -1 otherwise. Since we have 3 rankings for each video, we actually assign weights of +3, +1, -1, or -3. The +/- 3 are if all rankings agree in one direction, and the +/- 1 are if one of the rankings disagrees with the other two. Figure 4.23 shows a histogram of all of the weighted scores from users on the Amazon Mechanical Turk. They are mostly in agreement, though there are a few +/- 1 scores.

4.5.3 Results

Once we have the global human rankings and the global machine rankings, we can compare them using the *Kendall-Tau score* (Kendall (1938)), which is defined as follows:

Definition 36. *Given a set of objects N objects X and two total orders $>_1$ and $>_2$, where $> (x_a, x_b) = 1$ if $x_a > x_b$ and $> (x_a, x_b) = -1$ if $x_a < x_b$, the* Kendall-Tau

Table 4.1: The Kendall-Tau scores between all of the machine rankings and the Hodge aggregated human rankings.

	Human	TDA Z_2	TDA Z_3	CD Freq	CD Lattice	Clarity
Human	1	0.642	0.663	-0.295	0.347	0.284
TDA Z_2	0.642	1	0.979	-0.295	0.2	0.537
TDA Z_3	0.663	0.979	1	-0.316	0.221	0.516
CD Freq	-0.295	-0.295	-0.316	1	-0.0842	-0.189
CD Lattice	0.347	0.2	0.221	-0.0842	1	0.411
Clarity	0.284	0.537	0.516	-0.189	0.411	1

score *is defined as*

$$\tau = \frac{1}{N(N-1)/2} \sum_{i<j} (>_1 (x_i, x_j))(>_2 (x_i, x_j)) \qquad (4.36)$$

For two rankings which agree exactly, the Kendall-Tau score will be 1. For two rankings which are exactly the reverse of each other, the Kendall-Tau score will be -1. In this way, it analogous to a Pearson correlation between rankings.

Table 4.1 shows the Kendall-Tau scores between all of the different machine rankings and the human rankings. Encouragingly, our sliding window video TDA techniques agree with the human rankings more than any other pair of ranking types. Z_3 coefficients agree slightly more with the human ranking, but not in any statistically significant way (in the dataset, the difference was two pairs of adjacent videos flipping, leading to a τ of 0.98 between $\mathbb{Z}_2$ and $\mathbb{Z}_3$). The second most similar are the TDA and the diffusion clarity, which is interesting since they are both geometric techniques.

4.6 Dynamics in Vocal Fold Videos

Now that we have carefully explored the basic capabilities of our methodology, we apply it to a real world problem of interest in medicine. We show that our method can automatically detect certain types of voice pathologies from "high-speed glottography," or high speed videos (4000 frames per second) of the glottis and left and right vocal folds in the human vocal tract (Wittenberg et al. (1995), Wilden et al. (1998), Deliyski et al. (2007)). In particular, we are able to detect and differentiate quasiperiodicity from periodicity by using our geometric sliding window pipeline. Quasiperiodicity is a special case of what is referred to as "biphonation" in the biological literature, where nonlinear phenomena case a physical process to bifurcate into two different periodic modes, often during a transition to chaotic behavior (Herzel et al. (1996)). The torus structure we sketched in Figure 2.11 has long been recognized in this context (Herzel et al. (1996),Herzel et al. (1994)) but we provide a novel way of quantifying it using TDA. We are also sometimes able differentiate

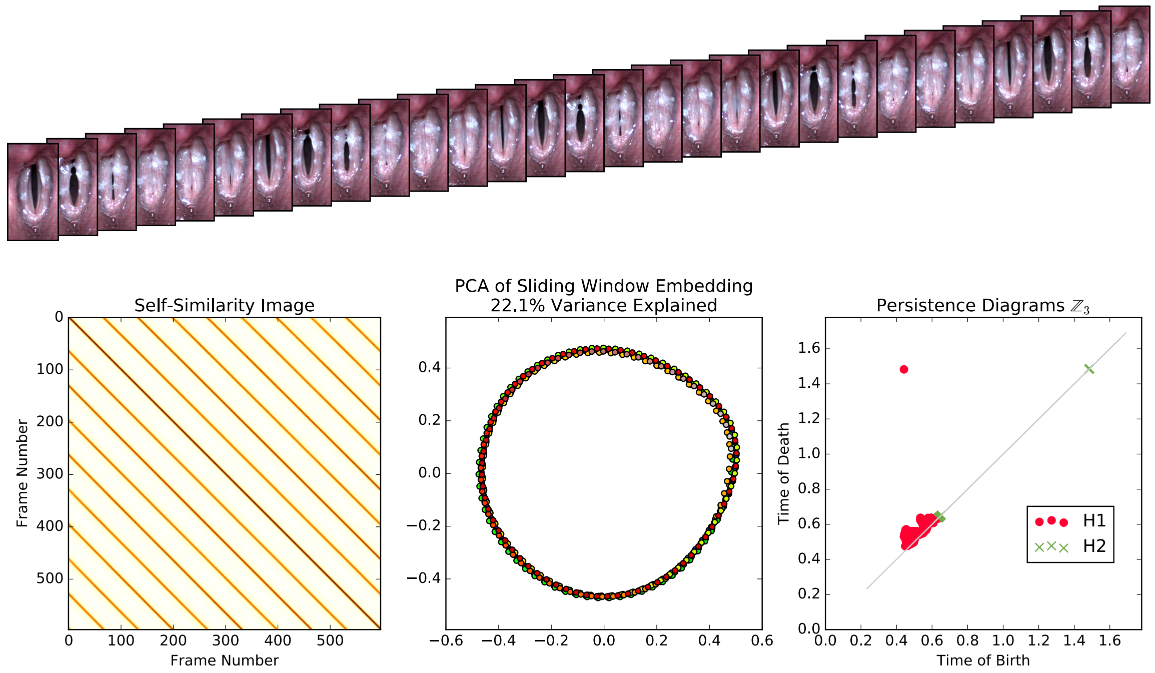


FIGURE 4.24: Video frames and sliding window statistics on a video of vocal folds undergoing normal periodic vibrations (Mehta et al. (2011)). One strong loop is visible in PCA and in the persistence diagrams

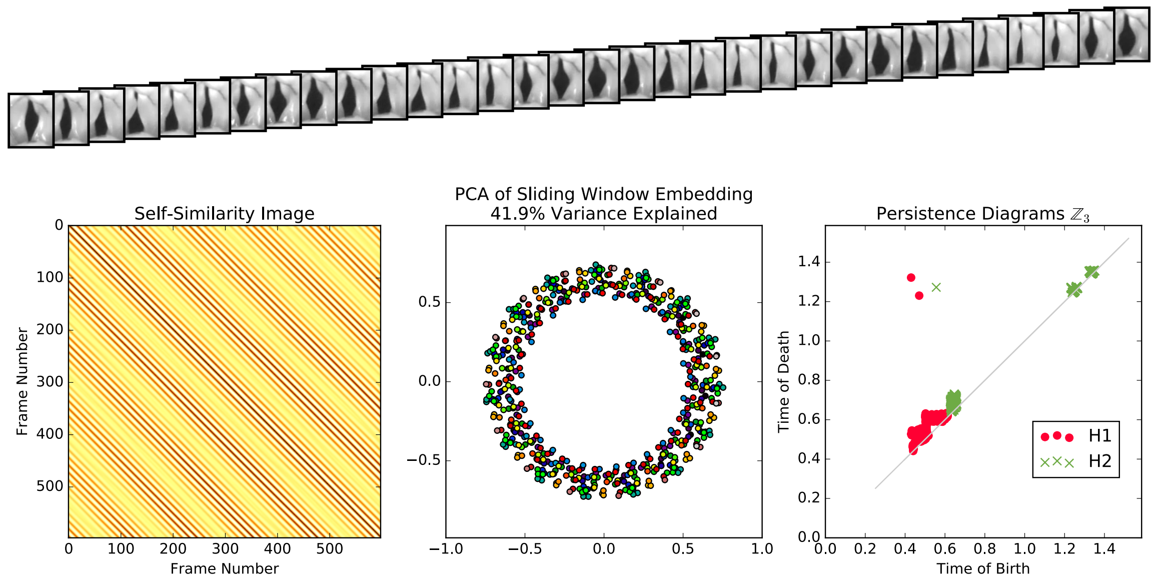


FIGURE 4.25: Video frames and sliding window statistics on a video of vocal folds undergoing biphonation, courtesy of Juergen Neubauer (Neubauer et al. (2001)). PCA suggests a possible torus, and the persistence diagram indeed has the signature of a torus (two strong independent 1-cycles and one 2-cycle)

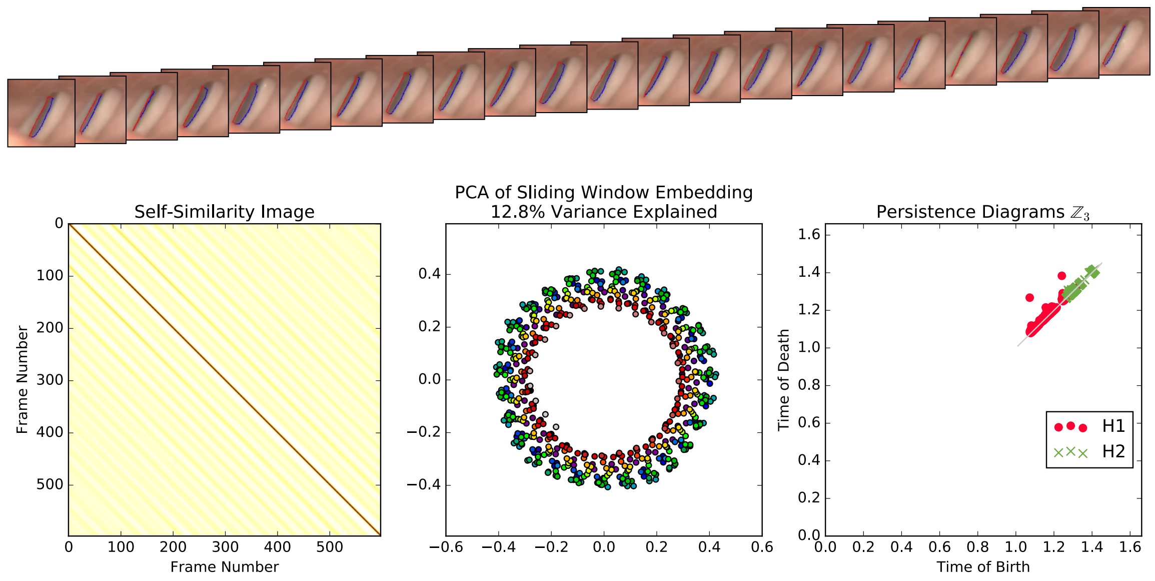
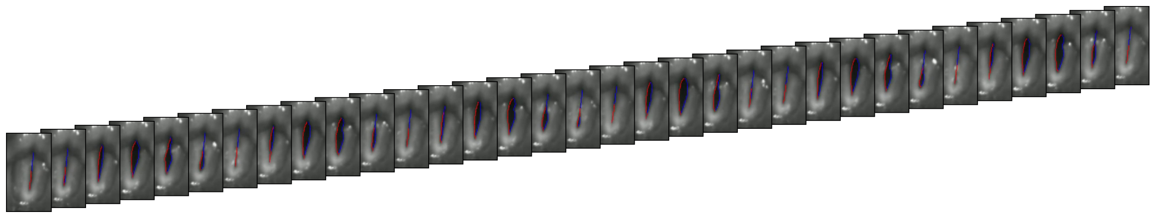


FIGURE 4.26: Video frames and sliding window statistics of "irregular" vocal cord vibrations, courtesy of Herbst et al. (2016). Though 2D PCA looks similar to Figure 4.25, no apparent 1D or 2D topological features are apparent in the high dimensional state space.

subharmonic periodicity from ordinary periodicity by using the observation in Section 4.3.3. Subharmonicity is a phenomenon that can occur in nonlinear systems where there is an additional harmonic introduced at a positive integer multiple of the period of the main oscillation (Wilden et al. (1998)).

We have a collection of 9 videos that we analyze, which we draw from a variety of different sources (Zacharias et al. (2016), Mehta et al. (2011), Neubauer et al. (2001), Herbst et al. (2016)). There are two videos which correspond to "normal" periodic vocal folds, three which correspond to biphonation (Neubauer et al. (2001)), two of which correspond to subharmonic phenomena, and two of which correspond to irregular motion. We manually extracted 400 frames per video (100 milliseconds) and autotuned the window size based on autocorrelation of 1D diffusion maps (Section 4.4.5). We then chose an appropriate τ so that the window size was 70, and we chose a time spacing so that each point cloud would have 600 points. As shown in Table 4.2, our technique is able to differentiate between the four classes. We also show PCA and persistence diagrams for one example for each class. In Figure 4.24, we see what appears to be a loop in PCA, and one strong 1D persistent dot confirms this. In Figure 4.25, we see a prominent torus in the persistence diagram. In Figure 4.26, we don't see any prominent structures in the persistence diagram, even though PCA looks like it could be a loop or a torus. Note, however, that PCA only preserves 13.7% of the variance in the signal, which is why high dimensional techniques are important to draw quantitative conclusions. Finally, in Figure 4.27, we see what appears to be a periodic video, but the maximum periodicity changes when switch-



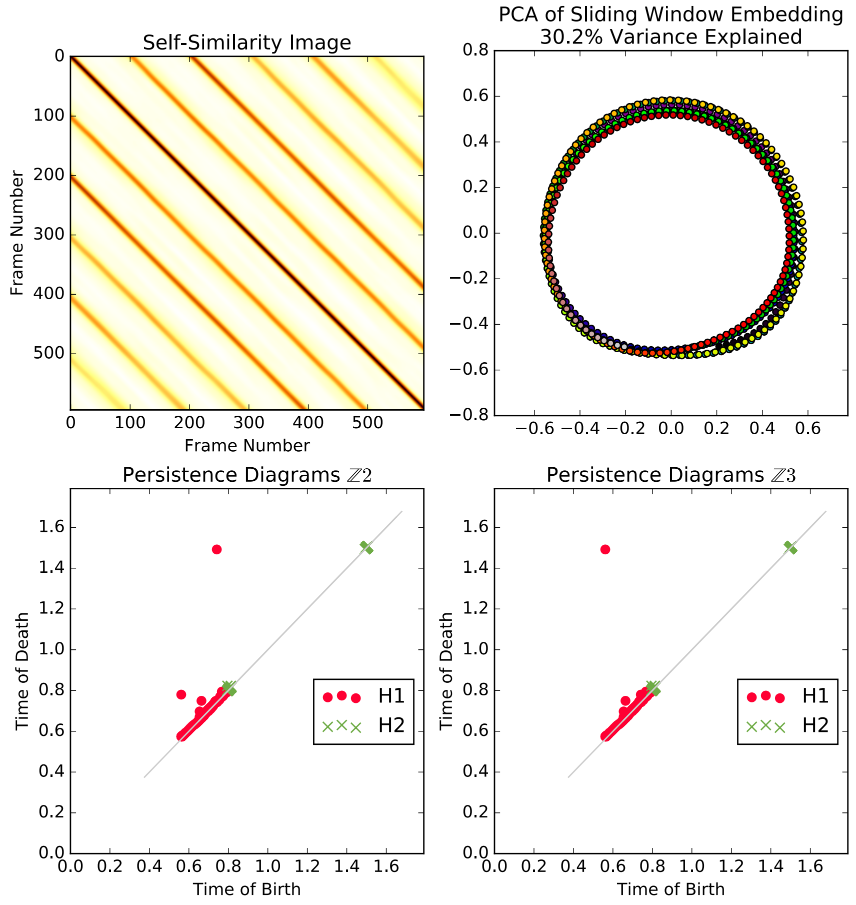


FIGURE 4.27: Video frames and sliding window statistics of subharmonic vocal cord vibrations, courtesy of Herbst et al. (2016). Although 2D PCA would suggest an ordinary periodic loop, changing field coefficients from $\mathbb{Z}_2$ to $\mathbb{Z}_3$ reveals that there is potentially a Möbius strip structure, which is consistent with the ground truth labeling of subharmonic.

ing field coefficients, since it actually corresponds to a subharmonic video. The one anomaly in Table 4.2 is that one of the periodic videos got a relatively high harmonic score. Overall, the harmonic score is the weakest indicator we have, though the fact that both of the subharmonic videos are in the top 3 is encouraging.

Table 4.2: Results of our sliding window pipeline videos of periodic vocal folds, biphonation, and irregularities. We give the max persistence periodicity score (PS), the modified periodicity score (MPS), the harmonic score (HS), and quasiperiodic score (QPS) presented in Section 4.4.3. We also show the window size (Win) that the autocorrelation technique in Section 4.4.5 gives. We have bolded the top three MPS, HS, and QPS scores across all videos. The max modified periodic scores include the two periodic videos and one of the biphonation videos. The max quasiperiodic scores are all of the biphonation videos, which means the one with a high periodicity score could be ruled out of the periodicity category. The subharmonic videos and one of the periodic videos receive the highest harmonic score.

Video Name	Win	PS	MPS	HS	QPS
Periodic 1 (Herbst et al. (2016))	16	0.816	**0.789**	0.035	0.011
Periodic 2 (Mehta et al. (2011), Figure 4.24)	32	0.601	**0.533**	**0.131**	0.009
Biphonation 1 (Neubauer et al. (2001))	53	0.638	0.294	0.012	**0.292**
Biphonation 2 (Neubauer et al. (2001))	42	0.703	**0.583**	0	**0.116**
Biphonation 3 (Neubauer et al. (2001), Figure 4.25)	67	0.515	0.076	0	**0.426**
Subharmonic Mucus (Zacharias et al. (2016))	39	0.519	0.442	**0.093**	0.021
Subharmonic Dynamics (Herbst et al. (2016))	21	0.537	0.488	**0.194**	0.028
Mucus Perturbed Periodic (Zacharias et al. (2016))	94	0.028	0.019	0	0.004
Irregular (Herbst et al. (2016), Figure 4.26)	232	0.18	0.097	-0.6	0.04

4.6.1 Comparisons To Standard Techniques

Our work takes quite a different approach from prior works on machine analysis of these videos. Usually, an inherently *Lagrangian* approach is applied, where different points on the left and right vocal folds are tracked, and coordinates of these points are analyzed as 1D time series (e.g. Neubauer et al. (2001); Qiu et al. (2003); Lohscheller et al. (2007); Herbst et al. (2016), Mehta et al. (2011)). This is a natural approach, since those are the pixels that measure the signal of interest, and using a 1D surrogate function enables well-understood 1D signal processing techniques. However, one of the issues of off-the-shelf image processing techniques that look for edges is that the edge detectors often require tuning, and they can suddenly fail when the vocal folds close (Lohscheller et al. (2007)). In our technique, we give up the ability to localize the anomalies (left/right, anterior/posterior) since we are not tracking them, but in return we do virtually no preprocessing, and our technique is domain independent.

There are also differences in how we quantify the signals. Neubauer et al. (2001) perform PCA on all of the time series from the tracked 1D points on the vocal folds, and they examine the Fourier spectrum of the principal components (which they refer to as "empirical orthogonal functions") to qualitatively point out differences in the modes present between normal phonation and biphonation. Many works also use standard techniques from nonlinear time series analysis on these signals, such as phase portraits, Poincaré Sections, and next amplitude maps (Herzel et al. (1994)). A very clever recent work uses the entropy of histograms of 1D Poincare Sections to detect bifurcations and the onset of irregularities (Herbst et al. (2016)). However, even though bifurcations can be effectively detected, they only use a 1-lag time delay embedding, which is inadequate to properly reconstruct the quasiperiodic phase space. In our work, we are able to specifically reconstruct and detect quasiperiodic phase spaces with a joint multi-lag delay embedding of all pixels using tools which can work in high dimensional ambient spaces. This allows us, for example, to differentiate between Figure 4.25 and Figure 4.26.

4.7 Conclusions / Future Work

We believe we have made the case for a geometric / dynamical systems approach in video, with two cursory applications matching human rankings of periodicity and funding normal and chaotic regimes in videos of vibrating vocal folds.

We made a naive attempt in this work to correct for drift, but in future work we would like to tackle more complicated camera shake / alignment scenarios. The challenge in that case will be to blindly decouple drift from legitimate periodic motion. We remark that other Eulerian approaches to analyzing periodic videos suffer from similar problems when drift is present (Wadhwa et al. (2013)). If enough of the signal is in color changes in the video as opposed to motion, a simple solution is to use optical flow to estimate motion and to warp all frames to a canonical pose with a piecewise affine map. This has been done automatic heartrate estimation algorithms in consumer video (Kumar et al. (2015); Tulyakov et al. (2016)), and there are perhaps applications where we can apply our technique.

Additionally, we would like to derive more specific bounds for when to expect a change in field coefficients to make a difference for harmonics, and we would like to extend beyond period doubling to any integer harmonic.

Finally, we would like to combine our vocal cord anomaly classification techniques with those in Herbst et al. (2016) on much larger databases to see if a large number of different types of anomalies can be automatically detected and classified.

5

Isometry Blind Time Warping

5.1 Introduction

We have now seen how different geometric descriptors of time-ordered point clouds are useful for describing the "shape of time series" in audio and video applications. Now, we will hone in more specifically on one of the descriptors in its own right: the self-similarity matrix (SSM). Using the SSM as a guide, we construct invariants to parameterizations of time-ordered point clouds, so that time warping of data is properly modeled. Moreover, since these structures are built on the SSM, they are also automatically *isometry blind*. The result is time-ordered point cloud comparison schemes which simultaneously factor out both isometries and parameterization.

We create two classes of isometry/parameterization blind matchings in this chapter. The first is a generalization of DTW for full metric alignment up to an isometry, which we discuss in Section 5.3. The second is a higher level descriptor based on critical points of the SSM, which we discuss in Section 5.4. Though this chapter is mostly theoretical, we hope that it will eventually address problems which are able to synchronize time series across modalities. For instance, one may like to synchronize audio of one person saying a phrase and video of another person saying the same thing, possibly at different rates (with non-uniform time warps). A simple example in this is the "CUAVE" dataset (Patterson et al. (2002)), which consists of video and audio of different subjects pronouncing the digits 0 through 9.

5.2 Prior Work on Time-Ordered Point Cloud Alignment / Re-Parameterization

There has been a lot of work on space curve matching for curves in the same ambient space, up to re-parameterizations. Among these are DTW and Fréchet Distance, which we reviewed in Section 2.4.

Recently there has also been some work attacking the more difficult problem of matching curves which live in different ambient spaces or which live in the same space but which may differ by a rigid rotation or translation, in addition to re-parameterization. Zhou and Torre (2009) and Zhou and De la Torre (2016) attack this with an expectation maximization approach, switching back and forth between a generalized eigenvalue problem to find the best linear projection to put curves in correspondence with each other and DTW to come up with the best parameterization map once the curves are aligned. This is similar to the earlier work of Hsu et al. (2005), which does the same basic alternating algorithm, but restricted to the same space ($\mathbb{R}^3$ with applications to MOCAP synchronization). Vejdemo-Johansson et al. (2015) attack a special case where curves form closed loops, using cohomology to find maps from the curves to the circle, where they can be synchronized.

Below we discuss two related techniques in slightly more detail to show the difficulties that can arise when trying to perform a spatial alignment before time warping, further motivating SSM-based techniques which avoid an explicit alignment.

5.2.1 Procrustes Alignment / Iterative Closest Points

One objective for spatial alignment could be to find the optimal *rigid transformation* taking one set of points to another. This leads to what's known as the *Procrustes distance* (or "Whaba's problem" Whaba (1965))

Definition 37. *Given two Euclidean point clouds $X, Y \in \mathbb{R}^d$, each with N points which are assumed to be in correspondence, the* Procrustes distance *is defined as* [1]

$$d_P(X, Y) = \min_{R_x, R_y, t_x, t_y} \sum_{i=1}^{N} ||R_x(x_i - t_x) - R_y(y_i - t_y)||_2^2 \tag{5.1}$$

where R_x and R_y are $d \times d$ rotation matrices, and t_x and t_y are d-dimensional translational offsets

The Procrustes can be solved with the Kabsch algorithm (Kabsch (1976)). The optimal translation simply aligns the points by their center of mass, so $t_x = \overline{X}$ and $t_y = \overline{Y}$. Let $\hat{Y} = Y - t_y$ and $\hat{X} = X - t_x$, and let $\hat{X}$ and $\hat{Y}$ be $d \times N$ matrices holding the coordinates of each point in the corresponding columns. Then the optimal rotations are found by taking a singular value decomposition.

$$\hat{X}\hat{Y}^T = USV^T \tag{5.2}$$

The optimal rotations are then $R_x = U^T$ and $R_y = V^T$.

One huge issue that's been swept under the rug is that not only do X and Y have to have the same number of points, but the correspondences must be known a

[1] Most authors will define only one rotation or translation, usually with respect to Y, which is convenient in practice if we want to keep one point cloud fixed, but we prefer this definition because it is more symmetric.

priori. Often in practice, neither of these assumptions are true. To deal with this, there has been a long line of research on an algorithm known as *Iterative Closest Points (ICP)*[2] (Besl and McKay (1992), Chen and Medioni (1992)), usually in the context of 3D shape registration. If a subset of correspondences is known, they are used. For the rest, the algorithm proceeds as follows

1. Center the point clouds on their centers of mass
2. Find the nearest neighbors in Y to all points in X
3. Solve the Procrustes alignment based on these correspondences (possibly with duplication if two points in X have the same nearest neighbor in Y)
4. Rotate the points so that they are in alignment, and repeat steps 2-4 until convergence

This algorithm works well in practice if there is a good initial guess of alignment, but it can converge to a poor local minimum if not. It also has high sensitivity to outliers and missing data, which occur frequently in applications of 3D scanning (for example), though there has been some very promising recent work addressing this with norms other than L_2 which deal with outliers better (Bouaziz et al. (2013)). In spite of these problems, Ying et al. (2016) use a modified version of ICP, replacing step 3 with dynamic time warping instead of nearest neighbors. This ensures that the time order will be respected, which is not guaranteed with nearest neighbors only. This analogous to the difference between Fréchet Distance and Hausdorff distance, for readers who are familiar.

We also remark that as an alternative to ICP and Procrustes, it is possible to rasterize point clouds to a grid and exhaustively search over all rotations using a modified version of the FFT (Kazhdan (2007)). This works well and is efficient for reasonable grid sizes of low dimensions, but it suffers from the curse of dimensionality since the number of grid cells explodes in higher dimensions.

5.2.2 Canonical Correlation Analysis

There are also techniques which use *Canonical Correlation Analysis*. These are very similar in spirit to Procrustes-based techniques, except spatial transformations are not restricted to rigid rotations, and the point clouds can be mapped into Euclidean spaces of a lower dimension. More precisely,

Definition 38. *Given two point clouds represented by matrices $X \in \mathbb{R}^{d_1 \times N}$ and $Y \in \mathbb{R}^{d_2 \times N}$, each with N points arranged along columns assumed to be in correspondence between X and Y,* Canonical Correlation Analysis *is defined as*

[2] Not to be confused with the Insane Clown Posse

$$d_{CCA} = \min_{V_x \in \mathbb{R}^{d_x \times b}, V_y} ||V_x^T X - V_y^T Y||_F^2 \tag{5.3}$$

for some chosen constant $b \leqslant \min(d_1, d_2)$*, s.t.*

$$V_x^T X X^T V_x = V_y^T Y Y^T V_y = I_b \tag{5.4}$$

In other words, we seek to find linear projections for each point cloud that project into the same space $\mathbb{R}^b$, where the sum of the squares is minimized in that space. This can also be solved with a singular value decomposition. Like Procrustes, this assumes that the correspondences are known a priori. To find the correspondences, Zhou and Torre (2009) take the same iterative approach as Ying et al. (2016) did with Procrustes, but they alternate back and forth between DTW and CCA instead of DTW and Procrustes. The same caveats apply to getting stuck in local mins if the initial correspondences are bad.

Finally, note that a recent work in Trigeorgis et al. (2016) takes a similar approach, but it replaces CCA by learning features in the projection stage with a deep neural network. This of course requires training data with known correspondences.

5.3 Self-Similarity Images And Metric Alignment

Most of the approaches we reviewed to align time series which have undergone linear transformations try to explicitly factor out those transformations before doing an alignment, but this is not necessary if we build our algorithm on top an *SSM* between two point clouds, which is already isometry blind, so that is the strategy we take. We note an additional advantage that we no longer need to restrict ourselves to Euclidean spaces either.

5.3.1 Induced 2D Warping Functions

To set the stage for our algorithms, we first study the maps that are induced between self-similarity images by re-parameterization functions, which will help in the algorithm design. For example, take the figure 8 curve, which we designate γ_8 (Figure 5.1a)

$$\gamma_8(t) : [0,1] \rightarrow \left[\begin{array}{c} \cos(2\pi t) \\ \sin(4\pi t) \end{array} \right] \tag{5.5}$$

Figure 5.1c shows the SSM of a linearly parameterized sampled version of this curve, while Figure 5.1d shows the SSM corresponding to a re-parameterized sampled version. Note that the maps between the domains of the SSMs shown are always rectangles, and they are independent of underlying curve being parameterized (they only depend on the relationship between two parameterizations). This can be seen by starting with a space curve $\gamma : [0,1] \rightarrow \mathbb{R}^d$ and its resulting self-similarity image

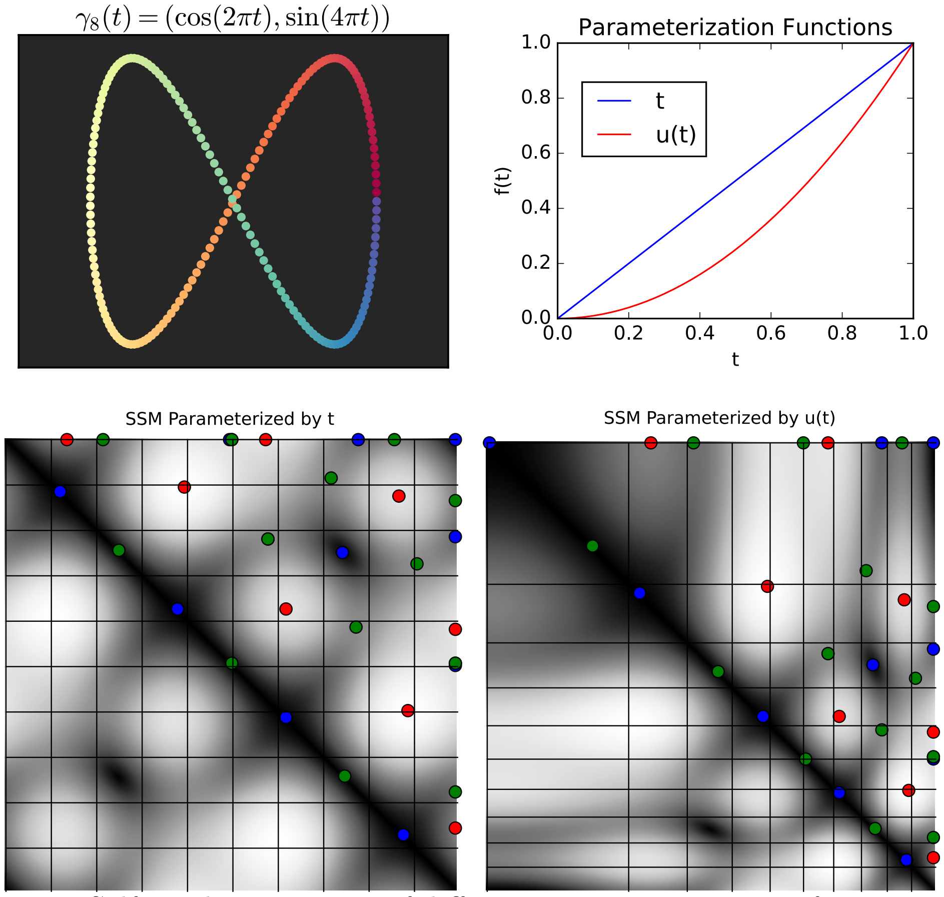


FIGURE 5.1: Self-similarity images of different parameterizations of a Figure 8 with, annotated with the critical points. Local mins are shown in blue, local maxes are shown in red, and saddle points are shown in green. Corresponding rectangles of the 2D map $h \times h$ are drawn with lines. Note that the critical points are unchanged, and that rectangles in one image map to rectangles in the other image

D_γ (Definition 6). Given a homeomorphism $h : [0,1] \to [0,1]$, which gives rise to a space curve $\gamma_h : [0,1] \to \mathbb{R}^d$ and a corresponding self-similarity image D_{γ_h}, there is an induced homeomorphism, $h \times h$ from the square to itself between the two domains of D_γ and D_{γ_h}. The commutative diagram below shows all of the maps that are involved

$$\begin{tikzcd}
{[u,v]} \arrow[r, "D_\gamma"] & \mathbb{R} \\
{[s,t]} \arrow[u, "h \times h"] \arrow[ur, "D_{\gamma_h}"'] &
\end{tikzcd}$$

In other words

$$D_{\gamma_h} = D_\gamma(h(s), h(t)) \tag{5.6}$$

For a discrete version of these maps between time-ordered point clouds, replace the homeomorphism h with a warping path, and the relationships are otherwise the same.

5.3.2 Gromov-Hausdorff Distance

All of our work in this section can be put into the *Gromov-Hausdorff Distance* framework, which describes how to "embed" one metric space into another. More formally,

Definition 39. *Given two discrete metric spaces (X, d_X) and (Y, d_Y), and a correspondence $\mathcal{C}$ (Definition 9) between X and Y, the* p-Stress *is defined as*

$$\mathcal{S}_p(\mathcal{C}) = \left(\sum_{(x,y),(x',y') \in \mathcal{C}} (d_X(x,x') - d_Y(y,y'))^p \right)^{1/p} \tag{5.7}$$

L^∞ stress $\mathcal{S}_\infty(\mathcal{C})$, or the maximum stretching induced by a correspondence, is often referred to as the "distortion" of a correspondence.

Intuitively, the p-Stress measures how much one has to stretch one metric space when moving it to another. The Gromov-Hausdorff Distance is based off of the L_∞ stress specifically (Gromov (2007)):

Definition 40. *Given two discrete metric spaces (X, d_X) and (Y, d_Y), the* Gromov-Hausdorff Distance *$d_H(X, Y)$ between X and Y is*

$$d_{GH}(X,Y) = \frac{1}{2} \inf_{\mathcal{C} \in \Pi} \mathcal{S}_\infty(\mathcal{C}) \tag{5.8}$$

where Π is the set of all correspondences between X and Y.

In other words, the Gromov-Hausdorff Distance measures the smallest possible *distortion* between a pair of points over all possible embeddings of one metric space into another. Unfortunately, the Gromov-Hausdorff Distance turns out to be NP-complete, and there is no known algorithm to even approximate it within a constant factor (Agarwal et al. (2015)). Bronstein et al. (2006) develop a practical algorithm to approximate the 2-stress between two polygon mesh surfaces. But even in this special case, they end up posing a highly nonconvex quadratic optimization problem, which is only guaranteed to converge to a local minimum and hence which needs a good initial guess, though they showed success for applications of 3D face matching.

5.3.3 IBDTW Distance Definition

The hardness of the Gromov-Hausdorff Distance is inherent in the definition, as the sheer number of correspondences to consider is huge. We design a new distance similar to the Gromov-Hausdorff Distance which can be lower bounded in polynomial time by restricting to warping path correspondences:

Definition 41. *Given two time-ordered point clouds X and Y with metrics d_X and d_Y, respectively, the* IBDTW Distance *is defined as*

$$IBDTW(X,Y) = \min_{\mathcal{W} \in \Omega} \mathcal{S}_1(\mathcal{W}) \tag{5.9}$$

where $\mathcal{W} \in \Omega$ is a valid warping path (Definition 32).

In other words, it is the 1-stress restricted to warping path correspondences. Note that for a general correspondence, there can be up to $\binom{MN}{2}$ or $O(M^2N^2)$ terms in each correspondence in Equation 5.8. However, in a warping path has at most $(M+N-1)$ correspondences, so there are only $O((M+N)^2)$ terms in Equation 5.9.

5.3.4 First Last Distance

As a warmup exercise to designing an algorithm to find the IBDTW, we first define a dissimilarity measure which is weaker than Definition 41 but which is easier to solve

Definition 42. *Given two time-ordered point clouds X and Y with M and N points and metrics d_X and d_Y, respectively, the* First-Last Dynamic Time Warping (FLDTW) *Distance is defined as*

$$FLDTW(X,Y) = \min_{\mathcal{W} \in \Omega} \sum_{(x_i, y_j) \in \mathcal{W}} |d_X(x_1, x_i) - d_Y(y_1, y_j)| + |d_X(x_M, x_i) - d_Y(y_N, y_j)| \tag{5.10}$$

where $\mathcal{W} \in \Omega$ is a valid warping path (Definition 32).

In other words, this is like the IBDTW, except we only consider a subset of the possible terms in the L_1-distortion; namely, the stretching between the first point and all of the rest of the points and the last point and all of the points. The reason this is so convenient is that by the definition of a warping path, the first point of each sequence and the last point of each sequence must be matched to each other, so the L_1-distortion terms in Equation 5.10 are guaranteed to be a subset of those in Equation 5.9. This problem can reduced to ordinary DTW, and can hence be solved in $O(MN)$ time. In particular, it is the DTW Distance of the following two time-ordered point clouds under the L_1 metric:

$$X = \begin{bmatrix} d_X(x_1, x_1) & d_X(x_M, x_1) \\ d_X(x_1, x_2) & d_X(x_M, x_2) \\ d_X(x_1, x_3) & d_X(x_M, x_3) \\ \vdots & \vdots \\ d_X(x_1, x_M) & d_X(x_M, x_M) \end{bmatrix}, Y = \begin{bmatrix} d_Y(y_1, y_1) & d_Y(y_N, y_1) \\ d_Y(y_1, y_2) & d_Y(y_N, y_2) \\ d_Y(y_1, y_3) & d_Y(y_N, y_3) \\ \vdots & \vdots \\ d_Y(y_1, y_N) & d_Y(y_N, y_N) \end{bmatrix} \quad (5.11)$$

where the first coordinate of each point cloud is the "first point distance" and the second coordinate is the "last point distance." Figure 5.2 shows an example with parts of the Figure 8 curve which have been parameterized differently and which have been rotated and translated with respect to each other. Note how the first and last point distances are warped versions of each other, and when they are plotted jointly in $\mathbb{R}^2$ they trace the same plane curve.

5.3.5 A Greedy Algorithm Lower Bounding The IBDTW

The first-last distance works well if the first and last point happen to be similar metrically between the two time-ordered point clouds, but this is not guaranteed, as the geometry could be distorted at the beginning or end of otherwise similar metric time-ordered point clouds. To extend this idea so that it works matching any row from one SSM to any row of another SSM, we make a simple modification to the DTW algorithm which can handle an imposed restriction on the warping path, defining a subroutine in Algorithm 4. The idea is to run the original DTW algorithm twice, once for point clouds $X_{1:i}, Y_{1:j}$ and once on $X_{i:M}$ and $Y_{j:N}$, and to add the costs. This is again exploiting the fact that the DTW algorithm must start on the first point and end on the last point, so we split it into two optimal sub-paths, one which ends on X_i, Y_j and one which starts on X_i, Y_j. Figure 5.3 shows an example.

Now, we can do the same thing as in the first-last distance, but this time we can match any pair of rows. In particular, we match the i^{th} row of SSM A to the j^{th} row of SSM B under the L_1 distance, enforcing the constraint that $(i, j) \in \mathcal{W}$. Since we don't know ahead of time which rows should be in correspondence, we try every row i of SSM A against every row j in SSM B, and we create a *Cross-Similarity Time-Warp Matrix (CSTWM)* C so that $C_{ij} = \text{constrainedDTW}(\text{SSMA}_i, \text{SSMB}_j)$. Then, we can apply the ordinary DTW algorithm to C. Algorithm 5 summarizes this process,

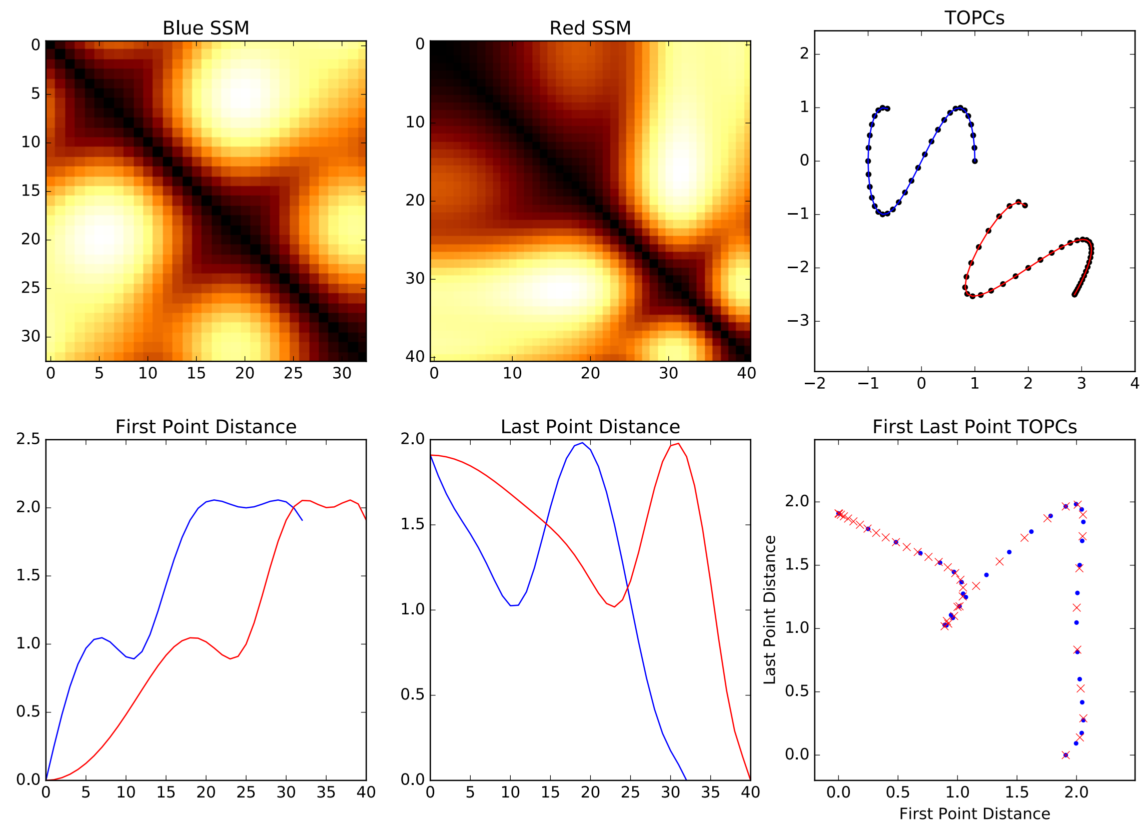


FIGURE 5.2: An example of the first and last point distances on parts of the Figure 8 which have been sampled differently from each other. The bottom left two plots show the first and last point distances, while the bottom right plot shows them plotted jointly with one distance along each axis.

which is simply a wrapper around the ordinary constrained DTW algorithm, with the outer loop itself as a DTW algorithm (so DTW on lots of constrained DTWs).

Figure 5.4 shows an example of running this algorithm on two rotated/translated/re-parameterized time-ordered point clouds in $\mathbb{R}^2$. As can be seen by the colors indicating correspondences, the algorithm was able to put the points into correspondence correctly even without first spatially aligning the curves. Figure 5.5 shows an example of running the algorithm between two curves which are metrically distorted in addition to being rotated/translated/re-parameterized. The returned warping seems reasonable.

Lower Bound Proof

We now connect Algorithm 5 to the IBDTW distance, and thereby show a relationship to the Gromov-Hausdorff Distance:

Lemma 7. *This cost returned by Algorithm 5 lower bounds the IBDTW Distance.*

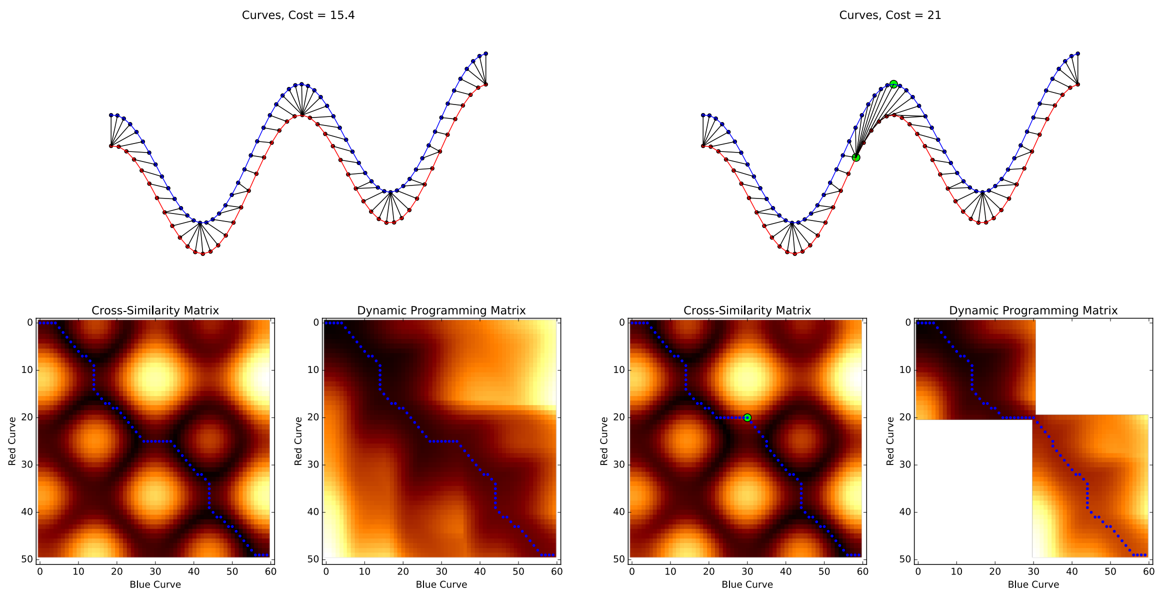


FIGURE 5.3: An example of unconstrained DTW (left) compared to constrained DTW (right). The two points which are designated to be in correspondence are highlighted with a green dot in the figure on the right. This forces a slightly suboptimal path with respect to the global alignment, but it is the optimal path with respect to the constraint.

Algorithm 4 Dynamic Time Warping with Constraints

```
1: procedure ConstrainedDTW($X$, $Y$, $d$, $i$, $j$)
2:     ▷ TOPCs $X$ and $Y$ with $M$ and $N$ points, compared with metric $d$. Return optimal path with the constraint that $x_i$ matches to $x_j$
3:     $D1 \leftarrow \text{DTW}([X_0, X_1, ...X_i], [Y_0, Y_1, ..., Y_j], d)$
4:     $D2 \leftarrow \text{DTW}([X_i, X_{i+1}, ...X_M], [Y_j, Y_{j+1}, ..., Y_N], d)$
5: return $D1 + D2 - d(X_i, Y_j)$                    ▷ $d(X_i, Y_j)$ was double counted
6: end procedure
```

Proof: To see this, note that the optimal IBDTW warping path $\mathcal{W}^*$ has the following cost $c(\mathcal{W}^*)$

$$c(\mathcal{W}^*) = \sum_{(x_i,y_j),(x',y')\in\mathcal{W}^*} |d_X(x_i, x') - d_Y(y_j, y')| \tag{5.12}$$

which can be rewritten as

$$c(\mathcal{W}^*) = \frac{1}{2} \sum_{(x_i,y_j)\in\mathcal{W}^*} \sum_{(x',y')\in\mathcal{W}^*} |d_X(x_i, x'), -d_Y(y_j, y')| \tag{5.13}$$

since if $(x', y') = (x_i, y_j)$ then the cost is zero, all other terms counted twice. Now

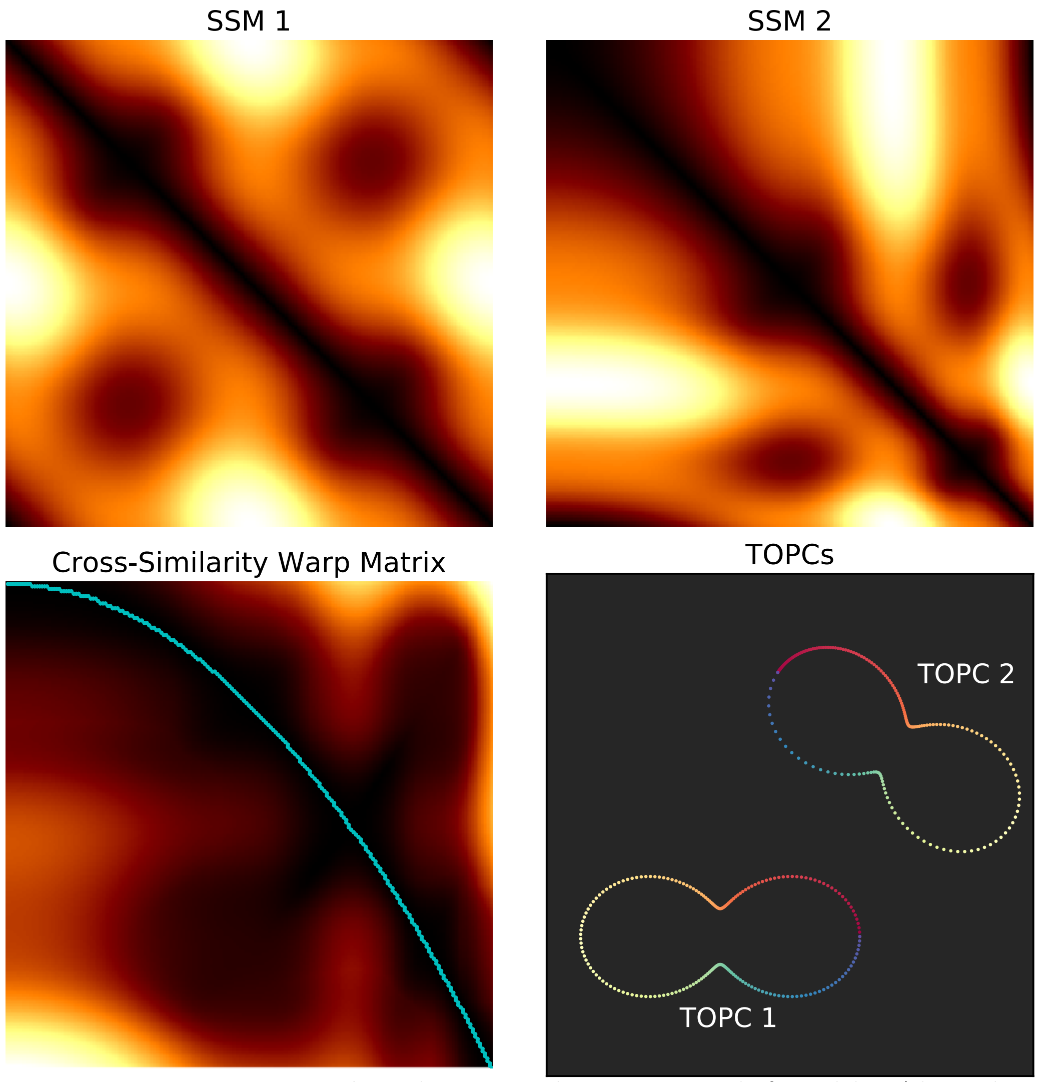


FIGURE 5.4: IBDTW example. The optimal warping path found by Algorithm 5 is drawn in cyan on top of the CSTWM. Based on this, points which are in correspondence are drawn with the same color in the lower right figure. Though time-ordered point cloud 2 has more points towards the beginning and fewer points towards the end than time-ordered point cloud 1, correct regions are put into correspondence with each other.

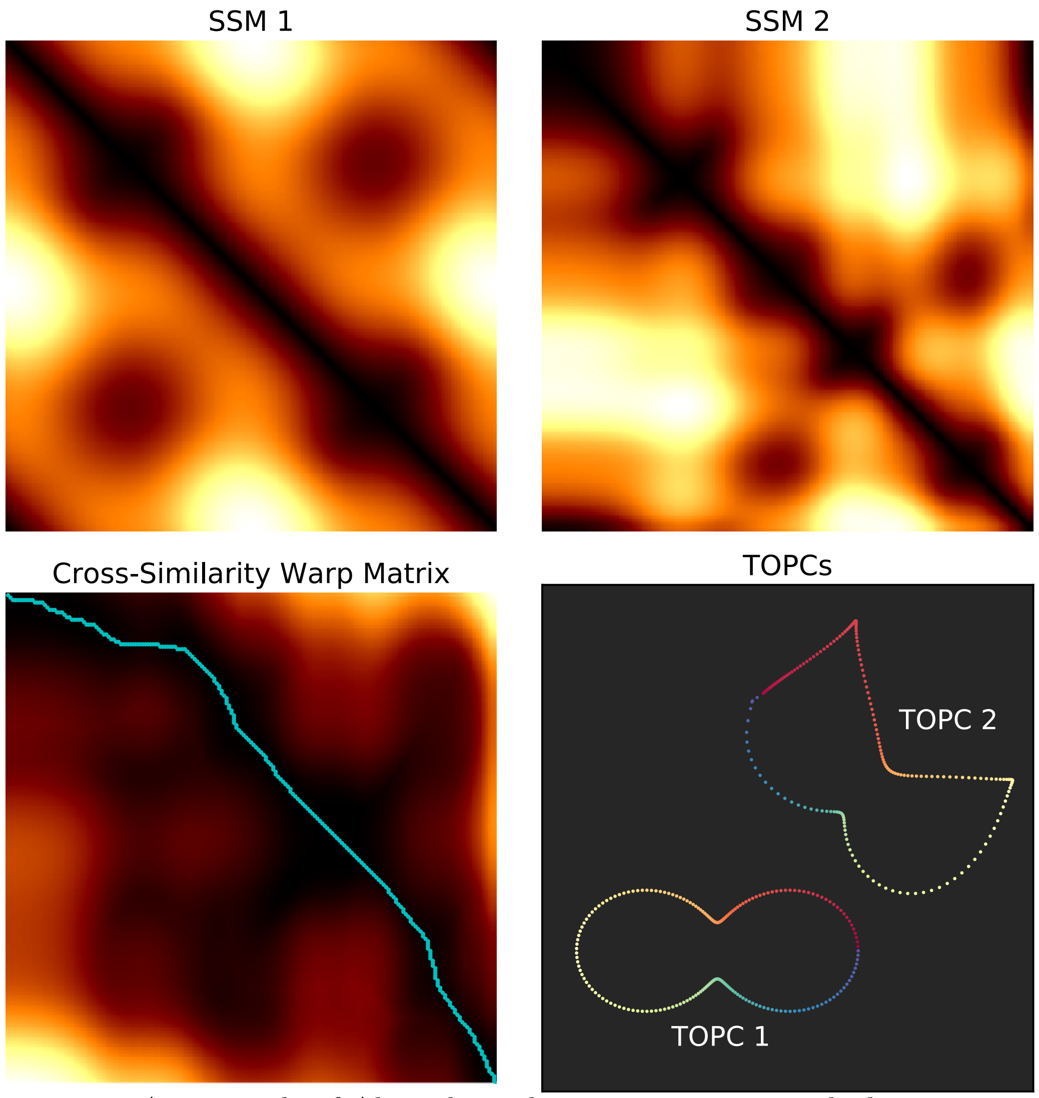


FIGURE 5.5: An example of Algorithm 5 between two curves which are not isometries. The result degrades gracefully.

fix an x_i and y_j. Then the sum of the terms of the form $|d_X(x_i, x') - d_Y(y_j, y')|$ is simply the L_1 warping distance between 1D time series which are the i^{th} row of d_X, $d_X[i,:]$ and the j^{th} row of d_Y, $d_Y[j,:]$ under the warping $\mathcal{W}^*$. Note that the DTW Distance between $d_X[i,:]$ and $d_Y[j,:]$ is at most the L_1 warping distance under $\mathcal{W}^*$, and is potentially lower since we are computing them greedily only between x_i and y_j, ignoring all other constraints. Hence, the sum of the terms $|d_X(x_i, x') - d_Y(y_j, y')|$ is lower bounded by Line 8 in Algorithm 5. ■

One subtlety is that although we have shown that the cost $D[M, N]$ returned by Algorithm 5 lower bounds $c(\mathcal{W}^*)$, the cost of the warping path $\hat{\mathcal{W}}$ obtained by backtracing through D has a cost which is greater than or equal to the optimal warping path: $c(\hat{\mathcal{W}}) \geqslant c(\mathcal{W}^*)$. This is because D was computed in a greedy manner

Algorithm 5 Isometry Blind Dynamic Time Warping Greedy Lower Bound

```
1: procedure IBDTW($X$, $Y$, $d_X$, $d_Y$)      ▷ TOPCs $X$ and $Y$ with $M$ and $N$ points, metrics $d_X$ and $d_Y$
2:     D ← $\underbrace{\left.\begin{array}{|c|c|c|c|}\hline 0 & \infty & \dots & \infty \\ \hline \infty & 0 & \dots & 0 \\ \hline \vdots & \vdots & \dots & \vdots \\ \hline \infty & 0 & \dots & 0 \\ \hline \end{array}\right\}_{M+1}}_{N+1}$      ▷ dynamic programming Matrix
3:     C ← $\underbrace{\left.\begin{array}{|c|c|c|c|}\hline 0 & 0 & \dots & 0 \\ \hline 0 & 0 & \dots & 0 \\ \hline \vdots & 0 & \dots & 0 \\ \hline 0 & 0 & \dots & 0 \\ \hline \end{array}\right\}_{M}}_{N}$      ▷ Cross-similarity time-warp matrix (CSTWM)
4:     for $i = 1 : M$ do
5:         for $j = 1 : N$ do
6:             $A \leftarrow [d_X(x_i, x_1), d_X(x_i, x_2), \dots, d_X(x_i, x_M)]$      ▷ $i^{\text{th}}$ row of $d_X$
7:             $B \leftarrow [d_Y(y_j, y_1), d_Y(y_j, y_2), \dots, d_Y(y_j, y_N)]$      ▷ $j^{\text{th}}$ row of $d_Y$
8:             $C_{ij} \leftarrow$ ConstrainedDTW(A, B, $L_1$, $i$, $j$)      ▷ (Algorithm 4)
9:             $D[i, j] \leftarrow C_{ij} + \min\{D[i-1, j-1], D[i-1, j], D[i, j-1]\}$
10:        end for
11:    end for
12: return $(\frac{1}{2}D[M, N], C)$      ▷ Return the cost and the CSTWM
13: end procedure
```

only considering pairs of rows at any step of the algorithm, which could lead to an overall suboptimal warping path with respect to all constraints.

Approximating the Gromov-Hausdorff Distance

For a more direct analogy with DTW, Algorithm 5 was designed to lower bound the 1-stress restricted to warping paths, but a very similar technique could be used to lower bound the Gromov-Hausdorff Distance restricted to warping paths. The constrained DTW in the inner loop in Line 8 can be replaced instead with a constrained version of the Discrete Fréchet Distance to find the maximum distortion induced by putting two points in correspondence, and the Discrete Fréchet Distance can be run on the final cross-similarity time-warp matrix. In this work, however, we stick to the 1-stress, since it gives a more informative overall picture of the full metric space.

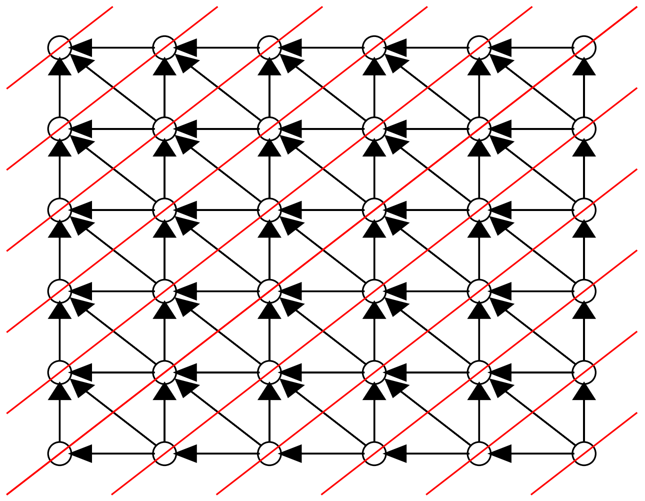


Figure 5.6: A schematic of the linear systolic array used to compute DTW in parallel. Elements of the dynamic programming matrix are drawn as circles. Black arrows are drawn to show dependencies between elements in the dynamic programming matrix, and red arrows show elements which can be computed in parallel if the processing occurs from the upper left $(0,0)$ to the lower right (M,N).

Computational Issues

The time complexity of this algorithm is $O(N^2M^2)$, since an $O(MN)$ dynamic programming algorithm is called for each pair $(i,j) \in (1,2,\dots M) \times (1,2,\dots,N)$. This quartic time complexity can be prohibitively expensive in practice. To mitigate this, we use a GPU algorithm that was designed by Yu et al. (2005) for Smith Waterman on gene sequences. It uses what's known as a *linear systolic array* to parallelize computation. In particular, instead of processing the elements of the dynamic programming matrix in "raster order" (row by row from left to right across each row), it is possible to process them along diagonal lines that propagate from the upper left to the lower right. The dependencies are satisfied simultaneously for all points along a diagonal line, so these elements can be computed in parallel. This takes $O(M+N)$ computation time instead of $O(MN)$ computation time, since there are $N+M-1$ red lines that need to be computed in sequence, but with enough parallel units the processing of each red line is $O(1)$. Furthermore, in the case of Algorithm 5, there are NM dynamic time warping problems that need to be computed in to obtain the CSTWM C (all pairs of SSM rows), but these can also be computed in parallel. In CUDA, for instance, each pixel of C can be computed in its own block, and each element along a red line can be computed by a thread within the corresponding block. Therefore, with enough parallel units, Algorithm 5 is itself reduced to $O(M+N)$

from $O(M^2N^2)$, although in practice we usually don't have that many parallel units. On our machine, we see a speedup of between 20-30x of a CUDA implementation of the linear systolic array over a C implementation of serial DTW.

5.4 Critical Point Topological Time Warping

In addition to the quartic time complexity IBDTW algorithms which work on the full metric alignment, we can design coarser isometry and parameterization blind similarity measures by focusing solely on the critical points of SSMs. Recall that a self-similarity matrix is a discretized version of a self-similarity image, which is a function defined over the unit square that returns all pairwise distances between points along a curve (Definition 6). Since a self-similarity image is defined over a 2D domain, there are three types of critical points that can occur: min (index 0), saddle (index 1), and max (index 2). Figure 5.1 shows a motivating example suggesting that critical points in SSMs are preserved under re-parameterizations. Though this is easier to prove in the continuous setting with self-similarity images, so we move away momentarily from time ordered point clouds to continuously parameterized space curves.

5.4.1 SSM Critical Points Are Preserved under Time Warps

Lemma 8. *Start with a space curve $\gamma : [0,1] \rightarrow \mathbb{R}^d$ and its resulting self-similarity image D_γ. Given a homeomorphism $h : [0,1] \rightarrow [0,1]$, which gives rise to a space curve $\gamma_h : [0,1] \rightarrow \mathbb{R}^d$ and a corresponding self-similarity image D_{γ_h}, there is an induced homeomorphism, $h \times h$ from the square to itself from the domain of D_γ to the domain of D_{γ_h} which preserves the critical points of D_γ.*

Proof:

We need to show both that a critical point (s^*, t^*) of D_γ maps to a critical point $(h(s^*), h(t^*))$ in D_{γ_h} and that the critical point remains the same index (min, saddle max). First, note that by the chain rule,

$$\frac{\partial D_{\gamma_h}(s,t)}{\partial s} = \frac{\partial D_\gamma(h(s),h(t))}{\partial s}\frac{\partial h(s)}{\partial s} + \cancel{\frac{\partial D_\gamma(h(s),h(t))}{\partial t}\frac{\partial t}{\partial s}} \tag{5.14}$$

since s and t are independent coordinates, the second term goes to zero. By a similar argument

$$\frac{\partial D_{\gamma_h}(s,t)}{\partial t} = \frac{\partial D_\gamma(h(s),h(t))}{\partial t}\frac{\partial h(t)}{\partial t} \tag{5.15}$$

Since h a homeomorphism, it is strictly monotonic, and $\frac{\partial h(t)}{\partial t}, \frac{\partial h(t)}{\partial s} > 0$. Therefore,

$$\frac{\partial D_{\gamma_h}(s,t)}{\partial s} = 0 \iff \frac{\partial D_\gamma(h(s),h(t))}{\partial s} = 0 \tag{5.16}$$

$$\frac{\partial D_{\gamma_h}(s,t)}{\partial t} = 0 \iff \frac{\partial D_\gamma(h(s),h(t))}{\partial t} = 0 \tag{5.17}$$

Or in other words, a critical point (s,t) remains a critical point after the map $(h(s),h(t))$.

What remains to be shown is that the index of the critical point remains the same. To do this, we need to look at the second partial derivatives of the warped image with respect to its coordinates at a critical point. By the product rule and chain rule

$$\frac{\partial^2 D_{\gamma_h}(s,t)}{\partial s^2} = \frac{\partial^2 D_\gamma(h(s),h(t))}{\partial s^2}\left(\frac{\partial h(s)}{\partial s}\right)^2 + \cancel{\frac{\partial D_\gamma(h(s),h(t))}{\partial s}\frac{\partial^2 h(s)}{\partial s^2}} \tag{5.18}$$

The second term goes to zero since we are assumed to be at a critical point where $\frac{\partial D_\gamma(h(s),h(t))}{\partial s} = 0$. By a similar argument

$$\frac{\partial^2 D_{\gamma_h}(s,t)}{\partial t^2} = \frac{\partial^2 D_\gamma(h(s),h(t))}{\partial t^2}\left(\frac{\partial h(t)}{\partial t}\right)^2 \tag{5.19}$$

and

$$\frac{\partial^2 D_{\gamma_h}(s,t)}{\partial s \partial t} = \frac{\partial^2 D_{\gamma_h}(s,t)}{\partial t \partial s} = \frac{\partial^2 D_\gamma(h(s),h(t))}{\partial s \partial t}\left(\frac{\partial h(s)}{\partial s}\right)\left(\frac{\partial h(t)}{\partial t}\right) \tag{5.20}$$

Now, let

$$a = \frac{\partial^2 D_{\gamma_h}(h(s),h(t))}{\partial s^2}, b = \frac{\partial^2 D_{\gamma_h}(h(s),h(t))}{\partial t^2}, c = \frac{\partial^2 D_{\gamma_h}(h(s),h(t))}{\partial s \partial t} \tag{5.21}$$

and

$$j = \frac{\partial h(s)}{\partial s} > 0, k = \frac{\partial h(t)}{\partial t} > 0 \tag{5.22}$$

Then showing that the two critical points have the same index amounts to showing that the sign of the determinant

$$\begin{vmatrix} a & c \\ c & b \end{vmatrix} \tag{5.23}$$

is the same as the sign of the determinant of

$$\begin{vmatrix} j^2a & jkc \\ jkc & k^2b \end{vmatrix} \tag{5.24}$$

and that the sign of a is the same as the sign of j^2a (the latter which is trivially true). To check that the signs of the determinants agree, note that the constant term

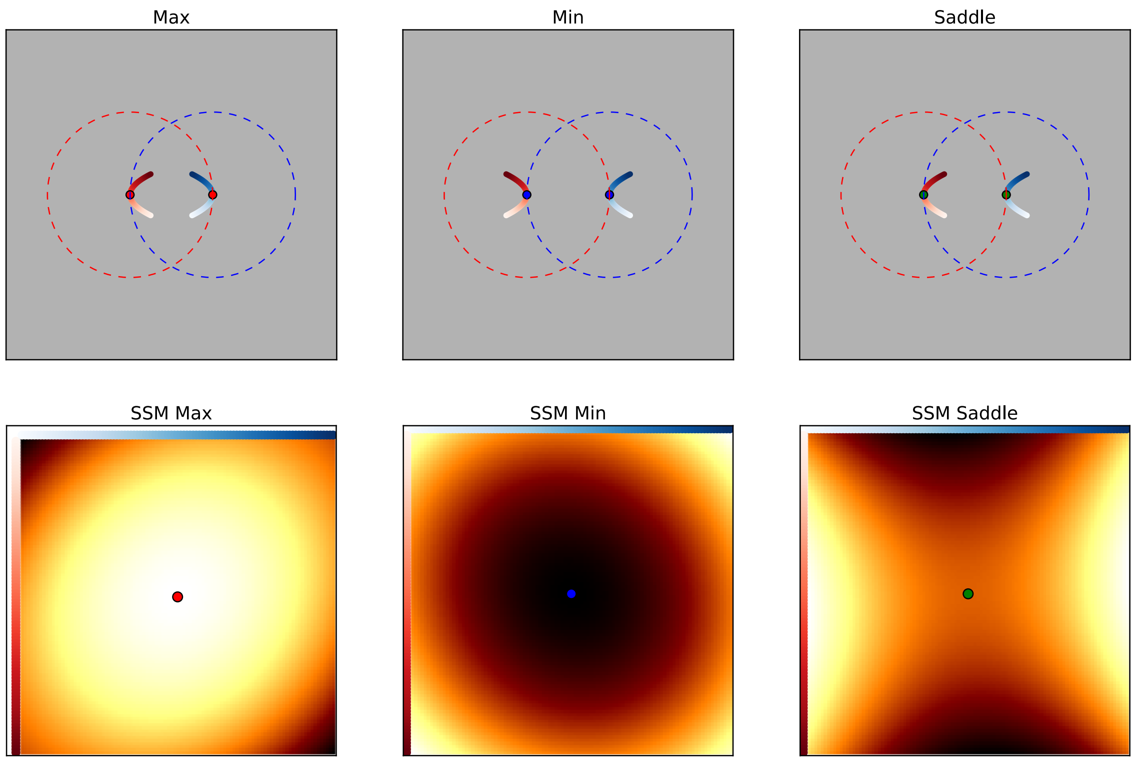


FIGURE 5.7: Examples of mins, maxes, and saddles in sub-sections of an SSM. Circles are drawn around the points in each segment where the critical point occurs to show that tangency is satisfied in both directions.

in the characteristic polynomial of a 2×2 matrix is the determinant of the matrix, which is also the product of the eigenvalues of the matrix $\lambda_1\lambda_2$. For the matrix in Equation 5.23, this determinant is $ab - c^2$, and for the corresponding matrix in Equation 5.24, it is $j^2k^2(ab - c^2)$. Since $j^2k^2 > 0$, they have the same sign, which completes the proof. ■

5.4.2 The Geometry behind Critical Points in SSMs

A lot of metric information is disregarded when only looking at critical points of SSMs, so it is useful to consider the geometric features that critical points retain. Given a self-similarity image D constructed from a Euclidean space curve $\gamma \in \mathbb{R}^d$, if (s, t) is a critical point of D, this means both that γ at s is tangent to the $(d-1)$ sphere of radius $D(s, t)$ centered at $\gamma(t)$, and that γ at t is tangent to the $(d-1)$ sphere of radius $D(s, t)$ centered at $\gamma(s)$. Figure 5.7 shows an example of a max, a min, and a saddle occurring from the interaction of two sub-segments of a time-ordered point cloud in $\mathbb{R}^2$. There is also a degenerate min that occurs if the curve actually crosses itself, though in general position in at least $\mathbb{R}^3$, we can eliminate such crossings and reduce it to the case shown in the center of Figure 5.7. The value $D(s,t)$ at the critical point indicates how close or far the curve gets at this critical time. It can be verified that all of the critical points in the Figure 8 example (Figure 5.1) behave like the examples in Figure 5.7.

The min and max scenarios are rather intuitive, but the saddle point is interesting. As Figure 5.7 shows, moving away from the critical point along the red segment gets closer to the critical point on the blue segment, while moving away from the critical point on the blue segment actually gets *further* away from the critical point on the red segment. If enough mins and maxes are present in the SSM, the Poincaré-Hopf theorem guarantees that a certain number of these features will also exist on the curve (Milnor (1997)). In particular, since a self-similarity image is defined on a topological disc, the Euler characteristic of its domain is 1, so this implies that $\chi_0 - \chi_1 + \chi_2 = 1$ [3], or that $\chi_1 = \chi_0 + \chi_2 - 1$. This could be of independent theoretical interest.

5.4.3 Quantifying with Persistence of Sublevelset Filtrations

We can use the persistence algorithm (Algorithm 1) to quantify the critical points in an SSM using persistent H_0 (connected components). To do this, we need to turn the SSM into a simplicial complex, and we need to come up with a valid filtration over it. To turn it into a simplicial complex, we triangulate the domain. Each element of an SSM has up to eight neighbors. We always include edges between the neighbors above, below, left, and right. To determine which diagonal neighbors are included, we add the diagonal edges which are the closest to aligning with the image gradient, which can help when using discrete algorithms to estimate flows between critical points (Edelsbrunner et al. (2001)). Triangles are included between three points whose pairwise edges are all included. Figure 5.8 shows an example triangulation of the SSM from 20 point sampled from a figure-8 curve γ_8. Note that we only include one point on the diagonal, as the diagonal is a degenerate, non-isolated minimum. We connect this point to all of the points on the diagonal directly next to the main diagonal. Also, we only triangulate the upper triangular part of the matrix, since an SSM is symmetric and the lower triangular part is redundant.

To build a filtration over the above triangulation, we use the *sublevelset filtration*, sometimes referred to as a "watershed method." Intuitively, the number of isolated pools of water are tracked as the water level rises. New pools form when the water passes over minima, and pools merge together when the water passes over a saddle point. More formally, the algorithm proceeds as follows:

1. Sort all of the points of the SSM D in ascending order. This takes time $O(N^2 \log(N))$ for an $N \times N$ SSM

2. Add vertices to the filtration in increasing order of value in D. The moment that two vertices of an edge are added, add that edge, and the moment that

[3] Where χ_0 is the number of mins, χ_1 is the number of saddle points, and χ_2 is the number of maxes

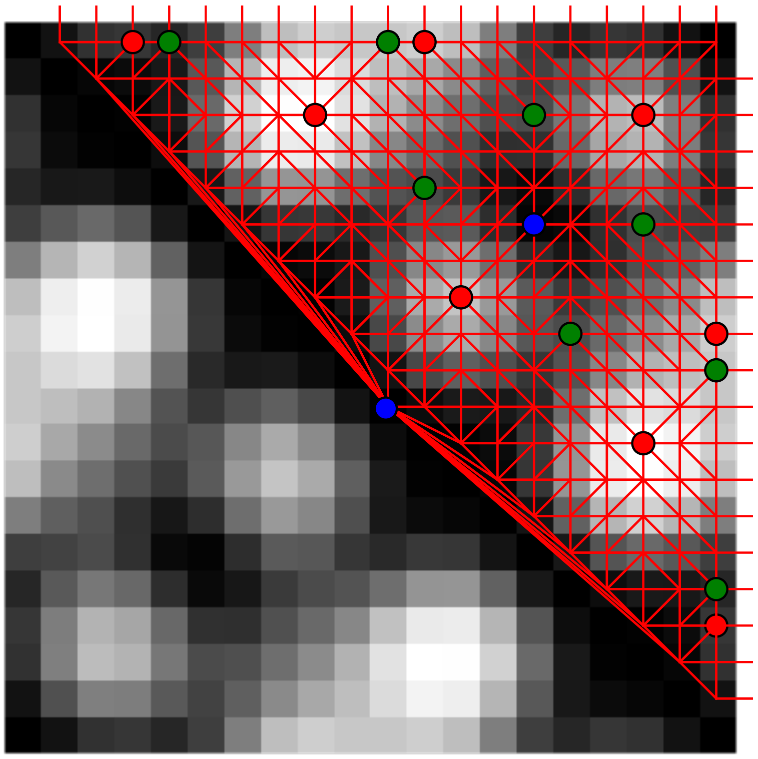


FIGURE 5.8: An example of building a triangle mesh on top of an SSM. Local maxes are drawn in red, local mins are drawn in blue, and saddle points are drawn in green. Only one point is used for the entire diagonal, as it is a degenerate, non-isolated minimum.

all three edges of a triangle are added, add that triangle. Each persistent H_0 class is associated with a unique local minimum in the image. A tree structure that records the merge events for this filtration is known as a "join tree" (Carr et al. (2003)).

3. Repeat steps 1 and 2, but sort the points in descending order, so that the pixels in D are sorted from largest to smallest value. In this case, each persistent H_0 class is associated with a local maximum. This is referred to as a "superlevelset filtration." A tree structure which records the merge events for this filtration is known as a "split tree" (Carr et al. (2003)).

This can be done much faster than higher homology, since all that's needed are connected components (the "pools" of water), which can be tracked by union find at a cost of $O(N^2\alpha(N^2))$. Hence, the dominant cost is for a global sort (though there is a recent output sensitive algorithm by Raichel and Seshadhri (2016) that can avoid the global sort). Note that the death of a connected component is always

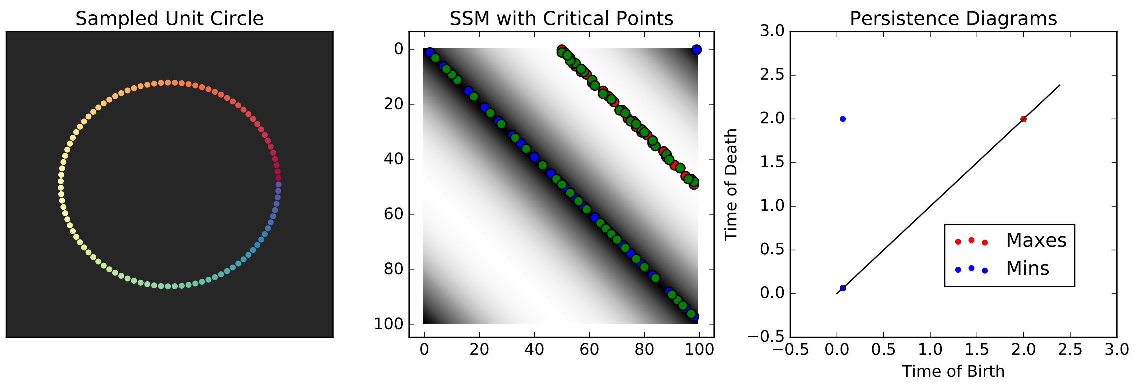


FIGURE 5.9: An example of sub and superlevelset filtrations on the SSM of a point cloud sampled from a circle. There is a degenerate non-isolated maximum in the self-similarity image of a circle at a diagonal offset of π, and there is also a non-isolated minimum near the diagonal. These both cause a lot of spurious critical points when computed numerically on a discrete SSM, but these spurious points have low persistence. There is one max and one min with infinite persistence (not drawn), and there is one min with persistence 2 which is born either in the upper right corner or somewhere on the main diagonal, depending on numerics, and which dies at a saddle point on the middle maximum line at a height of 2 (the diameter of the circle).

paired with the most recent birth (Line 24, Algorithm 1). If two critical points exchange height values, then birth/death pairings could change in the associated tree structures, but the persistence measures are stable. Furthermore, even if the critical points change locations in the SSM due to an instability, such as degenerate critical points along a near constant diagonal, which can occur for circular regions (Figure 5.9), the persistence measure is still stable. Hence, this is a natural choice for quantifying critical points.

Figure 5.10 and Figure 5.11 show two examples of persistence diagrams built off of SSMs. Note that the birth times of maxes from the superlevelset filtrations (red points) occur at larger heights than the death times, so the superlevelset diagram resides below the diagonal.

Matching Diagrams

To come up with a similarity score between the persistence diagrams sub/superlevelset filtrations of SSMs between two point clouds, we use the L_2 Wasserstein Distance (Definition 23). In particular, we report the sum of d_W^2 between the min diagrams and the max diagrams for a total cost. Figure 5.12 shows an example of the optimal bipartite matching found between a perturbed version of a Figure 8 and an unperturbed version. Note that a few of the points in the persistence diagrams of the unperturbed version have multiplicity greater than 1, so they get matched to

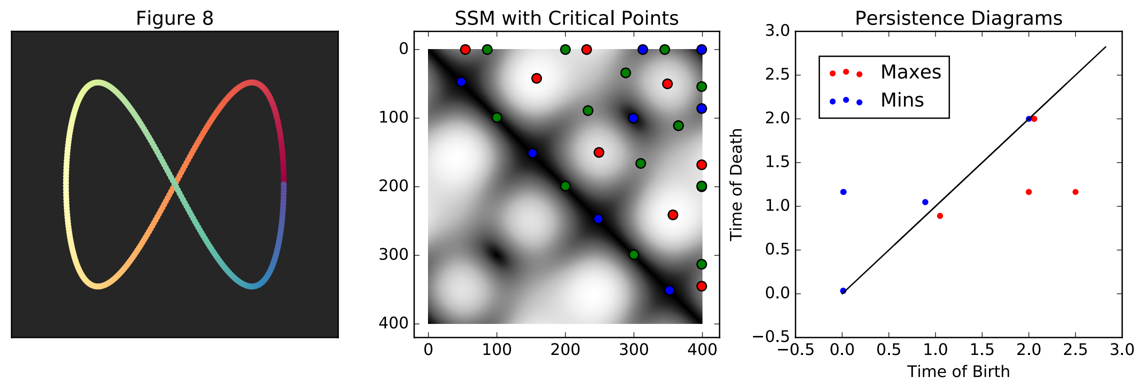


FIGURE 5.10: An example of sub and superlevelset filtrations on the SSM of a point cloud sampled from a Figure 8. Note that some of the persistence diagram points have multiplicity 2 due to uniform sampling and symmetries in the Figure 8.

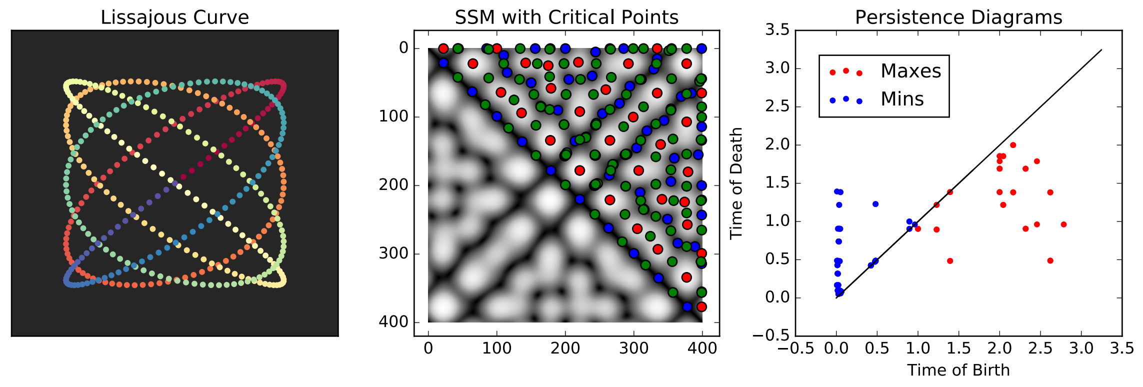


FIGURE 5.11: An example of sub and superlevelset filtrations on the SSM of a point cloud sampled from a Lissajous curve.

more than one point in the perturbed version.

5.5 Synthetic Experiments

So far, we have provided a number of illustrations of our algorithms, but a more comprehensive assessment of their performance is needed. To test the IBDTW approximation algorithm and the SSM critical point algorithms, we devise a classification experiment with 9 families of time-ordered point clouds, as shown in Figure 5.13. For each type of curve, we sample under 60 different parameterizations, for a total of 540 time-ordered point clouds. To create different parameterizations, we create a dictionary of basic warping paths as in Zhou and De la Torre (2016), and we use random convex combinations of the dictionary elements to create new warping paths (Figure 5.14). We then randomly translate/rotate the resulting time-ordered point clouds.

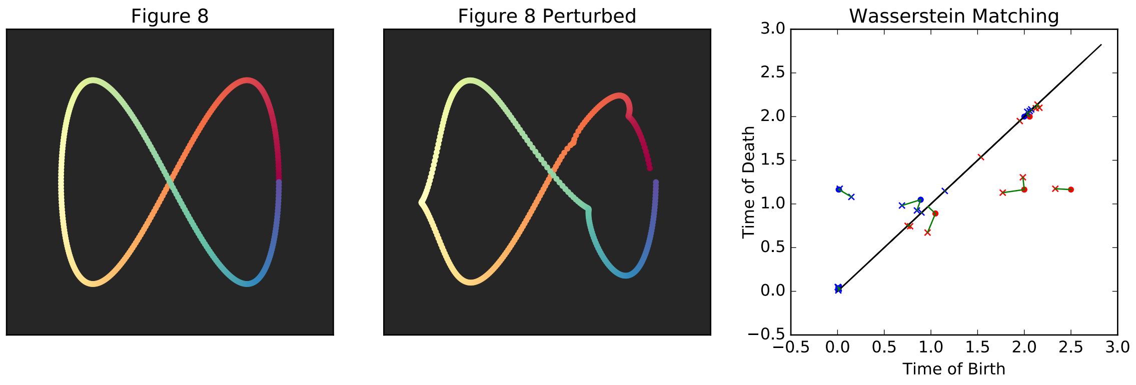


FIGURE 5.12: An example of Wasserstein matchings of persistence diagrams of the sub/superlevelset filtrations on SSMs between a Figure 8 and a warped Figure 8. Points for the original Figure 8 are drawn as dots, and points for the warped version are drawn as Xs.

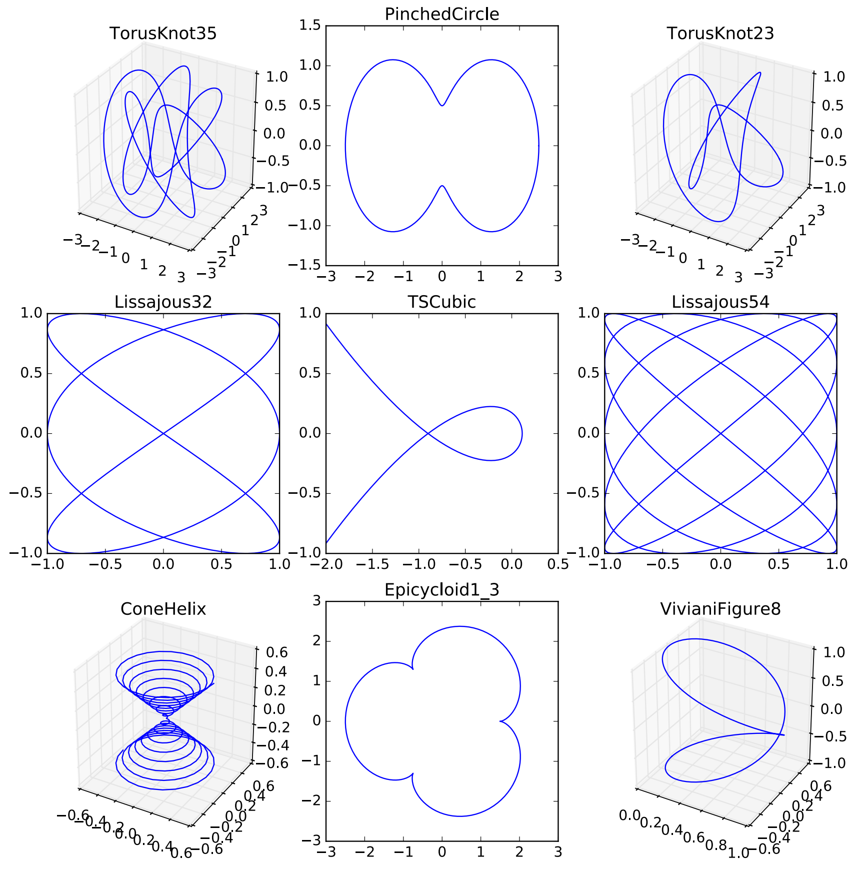
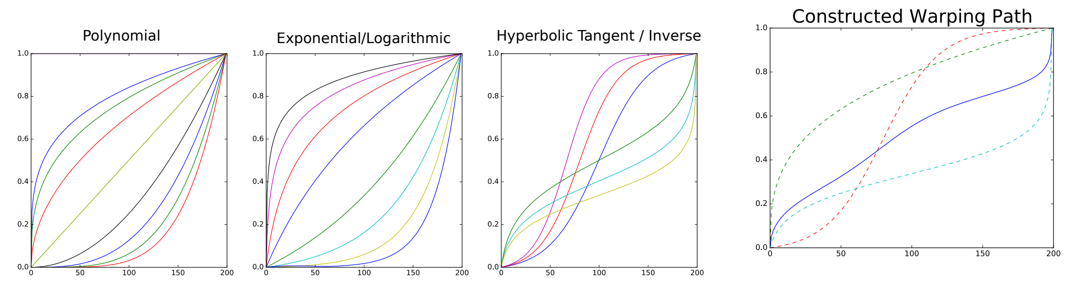


FIGURE 5.13: The 9 different families of synthetic curves we use in our synthetic isometry blind time warping experiment.



(a) Warping path dictionary (b) An example warping path

FIGURE 5.14: Basis functions for random synthetic warping paths, and an example random synthetic warping path constructed from a convex combination of 3 elements in this basis.

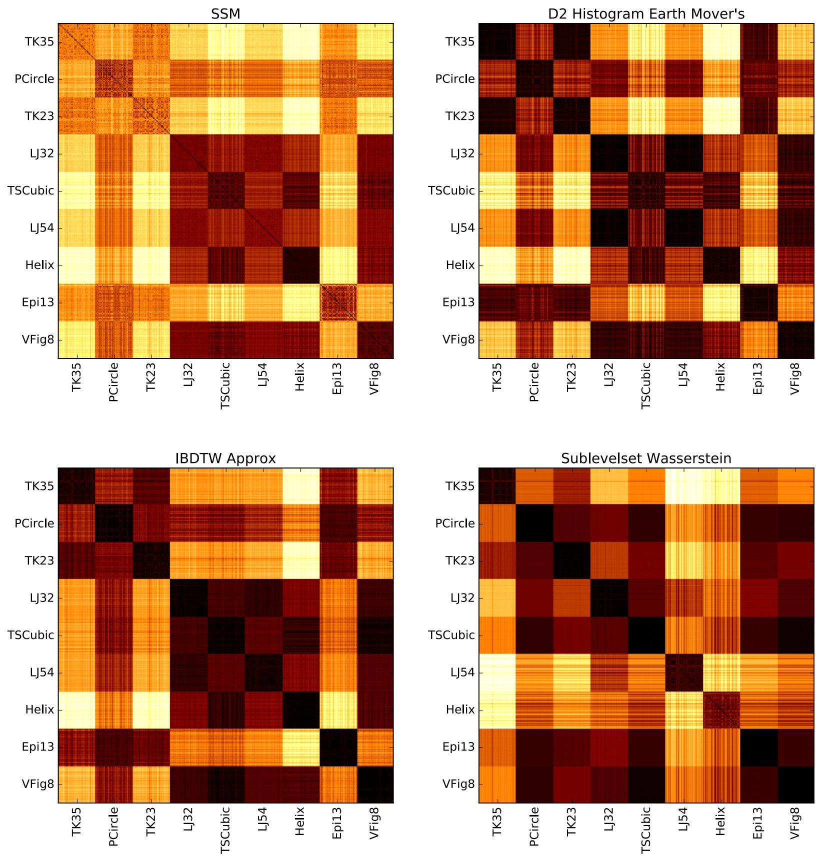


FIGURE 5.15: Confusion matrices for the classification problem of synthetic rotated/translated/re-parameterized curves.

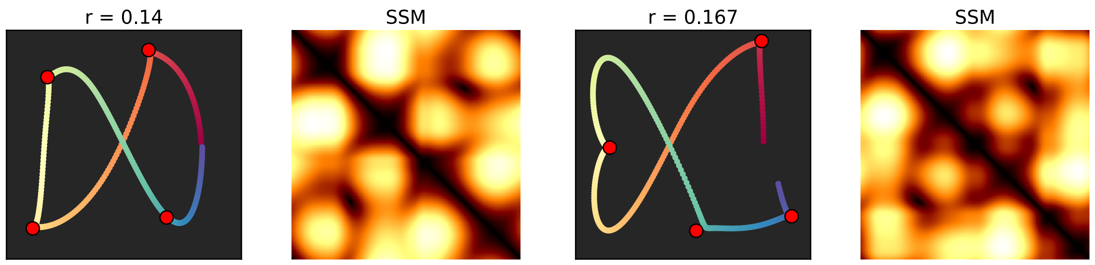


FIGURE 5.16: Examples of distorted Figure 8 time-ordered point clouds. Control points for the distortion are drawn as red dots. The ratio of the maximum displacement to the diameter, $r = |X - X'|_\infty / d(X)$, is reported.

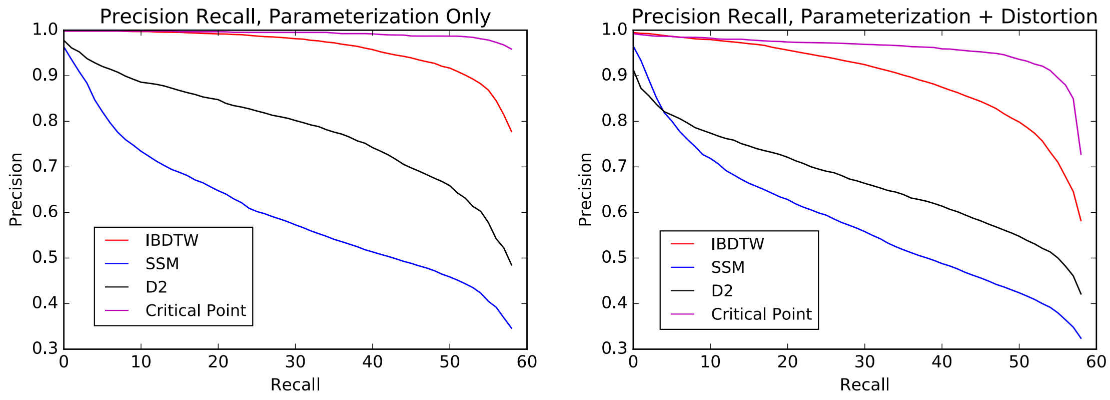


FIGURE 5.17: Precision-recall curves on the classification problem of synthetic rotated/translated/re-parameterized time-ordered point clouds. The left figure shows the results for re-parameterization only, while the right figure shows the results with displacements on top of re-parameterizations.

To classify the time-ordered point clouds, we compute the alignment cost from Algorithm 5, as well as the Wasserstein distance of the critical point diagrams in Section 5.4.3. We also compare to a naive L_2 distance between SSMs, mirroring what we did with cover songs in Chapter 3, and we compare to the Earth Mover's Distance (Rubner et al. (2000)) between the D2 histograms of pairwise distances. Figure 5.15 shows the confusion matrices obtained in this experiment, and Figure 5.17a shows the associated precision-recall curves. Encouragingly, both of our algorithms do better than naive approaches. In theory, the performance of the sublevelset critical point technique should be perfect if no two persistence diagrams in the original family of curves are the same. In practice, because the time-ordered point clouds are discretely sampled, this induces a discretization on the SSM which may change the location and sampling of the critical points. Also, although the critical point algorithm performs slightly better than the IBDTW, the critical point algorithm does not provide a warping path, so it is not possible to align the time series. In this way, the two techniques are complementary to each other.

Finally, to assess the robustness of our algorithms to noise, we simulate random displacements on top of the re-parameterization/rotation/translation. Figure 5.16 shows an example, where the mean ratio between the maximum displacement of a point and the diameter of the time-ordered point cloud over all 540 time-ordered point clouds is 0.24. The results certainly degrade, but not appreciably.

5.6 Future Work

Unlike the previous two chapters, this chapter does not yet have a real world application. However, it is our belief that these techniques can be applied to real datasets on multimodal time warping, such as those shown by Trigeorgis et al. (2016). Because the scales and extrinsic geometry of the time-ordered point clouds across modalities may be different, however, we may need to do some more preprocessing. One approach could be to try diffusion maps as a preprocessing step. We would also like to examine whether the IBDTW algorithm in particular can be applied to SSMs if they are turned into binary matrices, where each pixel in each row is a +1 if and only if the point directly to the right is greater, and 0 otherwise. In other words, turn each row into an indicator function of whether the time-ordered point cloud is moving away from or towards the corresponding point. This is obviously now scale invariant, but some experimentation would be needed to determine whether it is still discriminating enough.

Appendix A

Curvature And Torsion of Space Curves

We found it necessary to put all of these definitions in one place, and to prove the formulas we used in the cover songs chapter

A.0.1 Basic Definitions

Given a space curve $\boldsymbol{\gamma}(t) : [0, a] \rightarrow \mathbb{R}^d$ (the curve is parameterized from $t = 0$ to $t = a$ and has d coordinates), the *velocity vector* of the curve can be calculated as

$$\overline{\boldsymbol{e_1}}(t) = \boldsymbol{\gamma}'(t) \tag{A.1}$$

Let us assume, for the moment, that $\boldsymbol{\gamma}(s)$ is *parameterized by arc-length*; that is, the s corresponds to the arc length traveled over the interval $[0, s]$ (as a convention we will switch the parameter from t to s under this assumption). Another way of saying this is that $|\overline{\boldsymbol{e_1}}(s)|^2 = 1$, or the curve moves along at unit speed as the parameter is increased. In this case, the *curvature vector* is defined as

$$\overline{\boldsymbol{e_2}}(s) = \boldsymbol{\gamma}''(s) \tag{A.2}$$

The curvature vector is always orthogonal to the velocity vector in the case of arc-length parameterization. Intuitively, if the curve is moving along at a unit speed with respect to the parameter, then there is no tangential acceleration, so the second derivative along the tangent is zero. To see this mathematically, use the fact that arc-length parameterized curves have $|\boldsymbol{\gamma}'(s)|^2 = 1$, which can also be written as

$$\boldsymbol{\gamma}'(s) \cdot \boldsymbol{\gamma}'(s) = 1 \tag{A.3}$$

Using implicit differentiation with respect to s on both sides of the equation yields

$$2\boldsymbol{\gamma}''(s) \cdot \boldsymbol{\gamma}'(s) = 0 \tag{A.4}$$

Hence proving they are orthogonal. Additionally, the magnitude of the curvature vector

$$\kappa = |\boldsymbol{\gamma}''(s)| \tag{A.5}$$

is equal to the the inverse radius of the best fit circle, also known as the *osculating circle*, at a point. To see this, construct a circle of radius $\frac{1}{\kappa}$ which is parameterized by arc length:

$$\boldsymbol{\gamma_c}(t) = \frac{1}{\kappa}\left(\cos(2\pi\kappa t), \sin(2\pi\kappa t)\right) \tag{A.6}$$

It is then easily verified that $|\boldsymbol{\gamma_c}'(t)| = |\boldsymbol{\gamma}'(t)| = 1$ and $|\boldsymbol{\gamma_c}''(t)| = \kappa = |\boldsymbol{\gamma}''(s)|$, so the circle matches the curve up to a second order at the point s. κ is referred to as the *unsigned curvature* of the curve, and as shown by the discussion above, it measures how much a curve bends away from making a straight line.

If an arc-length parameterized curve is derived a third time, then what results is known as the *torsion vector*

$$\overline{\boldsymbol{e_3}}(s) = \boldsymbol{\gamma}'''(s) \tag{A.7}$$

As with curvature, *unsigned torsion* τ, is the magnitude of this vector

$$\tau = |\boldsymbol{\gamma}'''(s)| \tag{A.8}$$

and it measures the degree to which a curve is nonplanar. As such, this vector is always zero for plane curves. For space curves, we can continue this process and get higher and higher order numbers, which are all invariant to isometries, and hence make good candidates of numbers that could be used to match cover songs. In fact, by the fundamental theorem of curve analysis, all such numbers are sufficient to reconstruct a curve up to isometries (Toponogov (2006)).

A.0.2 Generalizing Beyond Arc-Length Parameterizations

Most curves are not parameterized by arc-length automatically, and it is useful to derive expressions for curvature and torsion which do not require explicit re-parameterization, as numerical re-parameterization may be difficult and susceptible to noise in practice. To see what these more general expressions might be, start with an arc-length parameterized curve $\boldsymbol{\gamma}(s)$, and re-parameterize it with a function $s = h(t)$ to get a curve $\boldsymbol{\gamma_h}(t)$ so that

$$\boldsymbol{\gamma_h}(t) = \boldsymbol{\gamma}(h(t)) \tag{A.9}$$

For simplicity of notation, let $\boldsymbol{\gamma}'$, $\boldsymbol{\gamma}''$, and $\boldsymbol{\gamma}'''$ denote the first three vector derivatives of $\boldsymbol{\gamma}$ with respect to s and $\dot{\boldsymbol{\gamma}}_{\boldsymbol{h}}$, $\ddot{\boldsymbol{\gamma}}_{\boldsymbol{h}}$, and $\dddot{\boldsymbol{\gamma}}_{\boldsymbol{h}}$ denote the first three vector derivatives of $\boldsymbol{\gamma_h}$ with respect to t

$$\dot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t) = \boldsymbol{\gamma}'(h(t))h'(t) \tag{A.10}$$

and

$$\ddot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t) = \boldsymbol{\gamma}''(h(t))h'(t)^2 + \boldsymbol{\gamma}'(h(t))h''(t) \tag{A.11}$$

Since γ is arc-length parameterized, $|\boldsymbol{\gamma}'(h(t))| = 1$, so $|\dot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t)| = h'(t)$. Thus, the above equations can be written as

$$\dot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t) = \boldsymbol{\gamma}'(h(t))|\dot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t)| \tag{A.12}$$

and

$$\ddot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t) = \boldsymbol{\gamma}''(h(t))|\dot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t)|^2 + \boldsymbol{\gamma}'(h(t))h''(t) \tag{A.13}$$

The unit-speed velocity vector $\overline{\boldsymbol{e_1}}(h(t)) = \boldsymbol{\gamma}'(h(t))$, is therefore

$$\overline{\boldsymbol{e_1}}(h(t)) = \frac{\dot{\gamma}(t)}{|\dot{\gamma}(t)|} \tag{A.14}$$

Now, to compute $\overline{\boldsymbol{e_2}}(h(t))$, start by subtract from both sides of Equation A.13 the projection onto the velocity component $\dot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t)$, so that only an acceleration component is left, and solve for the arc-length parameterized curvature vector $\boldsymbol{\gamma}''(h(t))$:

$$\ddot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t) - \frac{\ddot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t) \cdot \dot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t)}{|\dot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t)|^2}\dot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t) = \boldsymbol{\gamma}''(h(t))|\dot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t)|^2 \tag{A.15}$$

the $\boldsymbol{\gamma}'$ term on the right hand side drops out since $\boldsymbol{\gamma}'(h(t)) \cdot \boldsymbol{\gamma}''(h(t)) = 0$, and so $\boldsymbol{\gamma}'(h(t)) \cdot \ddot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t) = 0$. Re-arranging and making common denominators, we get

$$\overline{\boldsymbol{e_2}}(h(t)) = \boldsymbol{\gamma}''(h(t)) = \frac{|\dot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t)|^2\ddot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t) - (\ddot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t) \cdot \dot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t))\dot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t)}{|\dot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t)|^4} \tag{A.16}$$

As a sanity check, if $h(t) = t$ so γ_h is arc-length parameterized, then this entire expression reduces to $\ddot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t)$.

Written to avoid square roots (which can save computation), the expression is

$$\overline{\boldsymbol{e_2}}(h(t)) = \boldsymbol{\gamma}''(h(t)) = \frac{(\dot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t) \cdot \dot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t))\ddot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t) - (\ddot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t) \cdot \dot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t))\dot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t)}{(\dot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t) \cdot \dot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t))^2} \tag{A.17}$$

To derive the magnitude only (unsigned curvature) of this vector, compute

$$|\boldsymbol{\gamma}''(h(t))| = \sqrt{\boldsymbol{\gamma}''(h(t)) \cdot \boldsymbol{\gamma}''(h(t))} \tag{A.18}$$

which is

$$\sqrt{\frac{|\dot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t)|^4(\ddot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t)\cdot\ddot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t)) - 2|\dot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t)|^2(\dot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t)\cdot\ddot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t)) + (\dot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t)\cdot\ddot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t))(\dot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t)\cdot\dot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t))}{|\dot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t)|^8}} \tag{A.19}$$

after some simplification, this turns into

$$\kappa = |\boldsymbol{\gamma}''(h(t))| = \frac{\sqrt{|\dot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t)|^2|\ddot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t)|^2 - (\dot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t)\cdot\ddot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t))^2}}{|\dot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t)|^3} \tag{A.20}$$

which may be a more familiar expression for some readers. In particular, the widely cited work of Mokhtarian and Mackworth (1986) on curvature scale space gives the equation for *signed curvature* of a 2D curve $\gamma(t) = (x(t), y(t))$

$$\frac{x'y'' - y'x''}{(x'^2 + y'^2)^{3/2}} \tag{A.21}$$

where the convention is that counter-clockwise curvature has a positive sign and clockwise curvature has a negative sign. Plugging in $\gamma(t) = (x(t), y(t))$ into Equation A.20, we get

$$\kappa = \frac{\sqrt{(x'^2 + y'^2)(x''^2 + y''^2) - (x'x'' + y'y'')^2}}{(x'^2 + y'^2)^{3/2}} = \frac{\sqrt{(x'y'' - y'x'')^2}}{(x'^2 + y'^2)^{3/2}} = \frac{|x'y'' - y'x''|}{(x'^2 + y'^2)^{3/2}} \tag{A.22}$$

So Equation A.21 agrees with Equation A.20 up to a sign. To see that positive signs correspond to counter-clockwise turns upon removing the absolute value from the numerator of Equation A.22, note that the determinant of the matrix $|\gamma'(t)\gamma''(t)|$ is positive for counter-clockwise turns. Indeed,

$$\begin{vmatrix} x' & x'' \\ y' & y'' \end{vmatrix} = x'y'' - x''y' \tag{A.23}$$

which checks out.

Let's now continue this line of thought for torsion, and then a pattern will emerge. First, differentiate Equation A.11 to get

$$\dddot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t) = \boldsymbol{\gamma}'''(h(t))h'(t)^3 + \boldsymbol{\gamma}''(h(t))(2h''(t) + h''(t)h'(t)) + \boldsymbol{\gamma}'(h(t))h'''(t) \tag{A.24}$$

Now, since $\overline{\boldsymbol{e_1}}(h(t)) \perp \overline{\boldsymbol{e_2}}(h(t))$ $(\boldsymbol{\gamma}'(h(t)) \perp \boldsymbol{\gamma}''(h(t)))$, we can re-arrange Equation A.24 and project out $\overline{\boldsymbol{e_1}}$ and $\overline{\boldsymbol{e_2}}$ to get

$$\overline{\boldsymbol{e_3}}(h(t)) = \boldsymbol{\gamma}'''(h(t)) = \frac{1}{|\boldsymbol{\gamma}'(h(t))|^3}\left(\begin{matrix}\dddot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t) - \dfrac{\dddot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t)\cdot\overline{\boldsymbol{e_1}}(h(t))}{|\overline{\boldsymbol{e_1}}(h(t))|^2}\overline{\boldsymbol{e_1}}(h(t)) \\ - \dfrac{\dddot{\boldsymbol{\gamma}}_{\boldsymbol{h}}(t)\cdot\overline{\boldsymbol{e_2}}(h(t))}{|\overline{\boldsymbol{e_2}}(h(t))|^2}\overline{\boldsymbol{e_2}}(h(t))\end{matrix}\right) \tag{A.25}$$

And in general

$$\overline{\boldsymbol{e_k}}(h(t)) = \frac{1}{|\boldsymbol{\gamma}'(h(t))|^k}\left(\boldsymbol{\gamma}^k_{\boldsymbol{h}}(t) - \sum_{j=1}^{k-1}\frac{\boldsymbol{\gamma}^k_{\boldsymbol{h}}(t)\cdot\overline{\boldsymbol{e_j}}(h(t))}{|\overline{\boldsymbol{e_j}}(h(t))|^2}\overline{\boldsymbol{e_j}}(h(t))\right) \tag{A.26}$$

This lends itself to a nice recursive computation

Bibliography

Adachi, M. (2012), *Embeddings and immersions*, American Mathematical Soc.

Agarwal, P. K., Fox, K., Nath, A., Sidiropoulos, A., and Wang, Y. (2015), *Computing the Gromov-Hausdorff Distance for Metric Trees*, pp. 529–540, Springer Berlin Heidelberg, Berlin, Heidelberg.

Agarwal, P. K., Fox, K., Pan, J., and Ying, R. (2016), "Approximating Dynamic Time Warping and Edit Distance for a Pair of Point Sequences," in *32nd International Symposium on Computational Geometry (SoCG 2016)*, eds. S. Fekete and A. Lubiw, vol. 51 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 6:1–6:16, Dagstuhl, Germany, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

Allmen, M. and Dyer, C. R. (1990), "Cyclic motion detection using spatiotemporal surfaces and curves," in *Pattern Recognition, 1990. Proceedings., 10th International Conference on*, vol. 1, pp. 365–370, IEEE.

Alt, H. and Godau, M. (1995), "Computing the Fréchet distance between two polygonal curves," *International Journal of Computational Geometry & Applications*, 5, 75–91.

Atanbori, J., Cowling, P., Murray, J., Colston, B., Eady, P., Hughes, D., Nixon, I., and Dickinson, P. (2013), "Analysis of bat wing beat frequency using Fourier transform," in *International Conference on Computer Analysis of Images and Patterns*, pp. 370–377, Springer.

Averbuch-Elor, H. and Cohen-Or, D. (2015), "RingIt: Ring-Ordering Casual Photos of a Temporal Event." *ACM Trans. Graph.*, 34, 33.

Barnes, C., Shechtman, E., Finkelstein, A., and Goldman, D. (2009), "PatchMatch: a randomized correspondence algorithm for structural image editing," *ACM Transactions on Graphics-TOG*, 28, 24.

Bartsch, M. A. and Wakefield, G. H. (2001), "To catch a chorus: Using chroma-based representations for audio thumbnailing," in *Applications of Signal Processing to Audio and Acoustics, 2001 IEEE Workshop on the*, pp. 15–18, IEEE.

Bauer, U. (2015–2017), "Ripser: a lean C++ code for the computation of Vietoris–Rips persistence barcodes," `http://ripser.org`.

Belkin, M. and Niyogi, P. (2003), "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural computation*, 15, 1373–1396.

Bello, J. P. (2007), "Audio-Based Cover Song Retrieval Using Approximate Chord Sequences: Testing Shifts, Gaps, Swaps and Beats." in *ISMIR*, vol. 7, pp. 239–244.

Bello, J. P. (2009), "Grouping recorded music by structural similarity," *Int. Conf. Music Inf. Retrieval (ISMIR-09)*.

Bello, J. P. (2011), "Measuring structural similarity in music," *IEEE Transactions on Audio, Speech, and Language Processing*, 19, 2013–2025.

Bello, J. P., Daudet, L., Abdallah, S., Duxbury, C., Davies, M., and Sandler, M. B. (2005), "A tutorial on onset detection in music signals," *IEEE Transactions on speech and audio processing*, 13, 1035–1047.

Bendich, P., Galkovskyi, T., and Harer, J. (2011), "Improving homology estimates with random walks," *Inverse Problems*, 27, 124002.

Bendich, P., Gasparovic, E., Harer, J., and Tralie, C. (2016), "Geometric Models for Musical Audio Data," in *Proceedings of the 32st International Symposium on Computational Geometry (SOCG)*.

Berger, M., Nonato, L. G., Pascucci, V., and Silva, C. T. (2010), "Fiedler trees for multiscale surface analysis," *Computers & Graphics*, 34, 272–281.

Berndt, D. J. and Clifford, J. (1994), "Using Dynamic Time Warping to Find Patterns in Time Series." in *KDD workshop*, vol. 10, pp. 359–370, Seattle, WA.

Bertin-Mahieux, T., Ellis, D. P., Whitman, B., and Lamere, P. (2011), "The Million Song Dataset." in *ISMIR*, vol. 2, p. 10.

Besl, P. J. and McKay, N. D. (1992), "Method for registration of 3-D shapes," in *Robotics-DL tentative*, pp. 586–606, International Society for Optics and Photonics.

Beygelzimer, A., Kakade, S., and Langford, J. (2006), "Cover trees for nearest neighbor," in *Proceedings of the 23rd international conference on Machine learning*, pp. 97–104, ACM.

Böck, S. and Widmer, G. (2013), "Maximum filter vibrato suppression for onset detection," in *Proc. of the 16th Int. Conf. on Digital Audio Effects (DAFx). Maynooth, Ireland (Sept 2013)*.

Bogdanov, D., Wack, N., Gómez, E., Gulati, S., Herrera, P., Mayor, O., Roma, G., Salamon, J., Zapata, J. R., and Serra, X. (2013), "Essentia: An Audio Analysis Library for Music Information Retrieval." in *ISMIR*, pp. 493–498, Citeseer.

Bogert, B. P., Healy, M. J., and Tukey, J. W. (1963), "The quefrency alanysis of time series for echoes: Cepstrum, pseudo-autocovariance, cross-cepstrum and saphe cracking," in *Proceedings of the symposium on time series analysis*, vol. 15, pp. 209–243, chapter.

Boissonnat, J.-D., Karthik, C., and Tavenas, S. (2016), "Building efficient and compact data structures for simplicial complexes," *Algorithmica*, pp. 1–38.

Bouaziz, S., Tagliasacchi, A., and Pauly, M. (2013), "Sparse iterative closest point," in *Computer graphics forum*, vol. 32, pp. 113–123, Wiley Online Library.

Bronstein, A. M., Bronstein, M. M., and Kimmel, R. (2006), "Generalized multidimensional scaling: a framework for isometry-invariant partial surface matching," *Proceedings of the National Academy of Sciences*, 103, 1168–1172.

Bronstein, A. M., Bronstein, M. M., and Kimmel, R. (2009), "Numerical Geometry of Non-Rigid Shapes," *Monographs in Computer Science (*.

Bronstein, A. M., Bronstein, M. M., and Kimmel, R. (2010), "The video genome," *arXiv preprint arXiv:1003.5320*.

Broomhead, D. S. and King, G. P. (1986), "Extracting qualitative dynamics from experimental data," *Physica D: Nonlinear Phenomena*, 20, 217–236.

Buades, A., Coll, B., and Morel, J.-M. (2005), "A non-local algorithm for image denoising," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 2, pp. 60–65, IEEE.

Bubenik, P. (2015), "Statistical topological data analysis using persistence landscapes." *Journal of Machine Learning Research*, 16, 77–102.

Callahan, P. B. and Kosaraju, S. R. (1995), "A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields," *Journal of the ACM (JACM)*, 42, 67–90.

Carr, H., Snoeyink, J., and Axen, U. (2003), "Computing contour trees in all dimensions," *Computational Geometry*, 24, 75–94.

Carrière, M., Oudot, S. Y., and Ovsjanikov, M. (2015), "Stable topological signatures for points on 3d shapes," in *Computer Graphics Forum*, vol. 34, pp. 1–12, Wiley Online Library.

Casey, M. and Slaney, M. (2006), "The importance of sequences in musical similarity," in *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, vol. 5, pp. V–V, IEEE.

Cavanna, N. J., Jahanseir, M., and Sheehy, D. R. (2015), "A Geometric Perspective on Sparse Filtrations," in *Proceedings of the Canadian Conference on Computational Geometry*.

Chen, C. and Edelsbrunner, H. (2011), "Diffusion runs low on persistence fast," in *2011 International Conference on Computer Vision*, pp. 423–430, IEEE.

Chen, C. and Freedman, D. (2011), "Hardness results for homology localization," *Discrete & Computational Geometry*, 45, 425–448.

Chen, N., Li, W., and Xiao, H. (2017), "Fusing similarity functions for cover song identification," *Multimedia Tools and Applications*, pp. 1–24.

Chen, Y. and Medioni, G. (1992), "Object modelling by registration of multiple range images," *Image and vision computing*, 10, 145–155.

Chung, F. R. (1997), *Spectral graph theory*, vol. 92, American Mathematical Soc.

Cohen-Steiner, D., Edelsbrunner, H., and Harer, J. (2007), "Stability of persistence diagrams," *Discrete & Computational Geometry*, 37, 103–120.

Cohen-Steiner, D., Edelsbrunner, H., Harer, J., and Mileyko, Y. (2010), "Lipschitz functions have L p-stable persistence," *Foundations of computational mathematics*, 10, 127–139.

Coifman, R., Rokhlin, V., and Wandzura, S. (1993), "The fast multipole method for electromagnetic scattering calculations," in *Antennas and Propagation Society International Symposium, 1993. AP-S. Digest*, pp. 48–51, IEEE.

Coifman, R. R. and Lafon, S. (2006), "Diffusion maps," *Applied and computational harmonic analysis*, 21, 5–30.

Cutler, R. and Davis, L. S. (2000), "Robust real-time periodic motion detection, analysis, and applications," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22, 781–796.

De Cheveigné, A. and Kawahara, H. (2002), "YIN, a fundamental frequency estimator for speech and music," *The Journal of the Acoustical Society of America*, 111, 1917–1930.

De la Porte, J., Herbst, B., Hereman, W., and Van Der Walt, S. (2008), "An introduction to diffusion maps," in *The 19th Symposium of the Pattern Recognition Association of South Africa*, Citeseer.

De Luca, A., Hang, A., Brudy, F., Lindner, C., and Hussmann, H. (2012), "Touch me once and i know it's you!: implicit authentication based on touch screen patterns," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 987–996, ACM.

de Silva, V., Skraba, P., and Vejdemo-Johansson, M. (2012), "Topological analysis of recurrent systems," in *Workshop on Algebraic Topology and Machine Learning, NIPS*.

Degara, N., Rúa, E. A., Pena, A., Torres-Guijarro, S., Davies, M. E., and Plumbley, M. D. (2012), "Reliability-informed beat tracking of musical signals," *IEEE Transactions on Audio, Speech, and Language Processing*, 20, 290–301.

Delbracio, M. and Sapiro, G. (2015), "Removing camera shake via weighted fourier burst accumulation," *IEEE Transactions on Image Processing*, 24, 3293–3307.

Deliyski, D. D., Petrushev, P. P., Bonilha, H. S., Gerlach, T. T., Martin-Harris, B., and Hillman, R. E. (2007), "Clinical implementation of laryngeal high-speed videoendoscopy: challenges and evolution," *Folia Phoniatrica et Logopaedica*, 60, 33–44.

Dey, T. K., Hirani, A. N., and Krishnamoorthy, B. (2011), "Optimal homologous cycles, total unimodularity, and linear programming," *SIAM Journal on Computing*, 40, 1026–1044.

Deyle, T., Tralie, C. J., Reynolds, M. S., and Kemp, C. C. (2013), "In-hand radio frequency identification (RFID) for robotic manipulation," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pp. 1234–1241, IEEE.

Dsilva, C. J., Lim, B., Lu, H., Singer, A., Kevrekidis, I. G., and Shvartsman, S. Y. (2015), "Temporal ordering and registration of images in studies of developmental dynamics," *Development*, 142, 1717–1724.

Eckmann, J.-P., Kamphorst, S. O., and Ruelle, D. (1987), "Recurrence plots of dynamical systems," *EPL (Europhysics Letters)*, 4, 973.

Edelsbrunner, H. (1993), "The union of balls and its dual shape," in *Proceedings of the ninth annual symposium on Computational geometry*, pp. 218–231, ACM.

Edelsbrunner, H. and Harer, J. (2008), "Persistent homology-a survey," *Contemporary mathematics*, 453, 257–282.

Edelsbrunner, H. and Harer, J. (2010), *Computational topology: an introduction*, American Mathematical Soc.

Edelsbrunner, H., Kirkpatrick, D., and Seidel, R. (1983), "On the shape of a set of points in the plane," *IEEE Transactions on information theory*, 29, 551–559.

Edelsbrunner, H., Letscher, D., and Zomorodian, A. (2000), "Topological persistence and simplification," in *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pp. 454–463, IEEE.

Edelsbrunner, H., Harer, J., and Zomorodian, A. (2001), "Hierarchical Morse complexes for piecewise linear 2-manifolds," in *Proceedings of the seventeenth annual symposium on Computational geometry*, pp. 70–79, ACM.

Eiter, T. and Mannila, H. (1994), "Computing discrete Fréchet distance," Tech. rep., Citeseer.

Eldar, Y. C., Sidorenko, P., Mixon, D. G., Barel, S., and Cohen, O. (2015), "Sparse phase retrieval from short-time Fourier measurements," *IEEE Signal Processing Letters*, 22, 638–642.

Ellis, D. (2009), "Robust Landmark-Based Audio Fingerprinting," `http://labrosa.ee.columbia.edu/matlab/fingerprint/`.

Ellis, D. P. (2006), "Identifying'cover songs' with beat-synchronous chroma features," *MIREX 2006*, pp. 1–4.

Ellis, D. P. (2007a), "Beat tracking by dynamic programming," *Journal of New Music Research*, 36, 51–60.

Ellis, D. P. (2007b), "Classifying music audio with timbral and chroma features," in *ISMIR 2007: Proceedings of the 8th International Conference on Music Information Retrieval: September 23-27, 2007, Vienna, Austria*, pp. 339–340, Austrian Computer Society.

Ellis, D. P. (2007c), "The "covers80" cover song data set," *URL: http://labrosa. ee. columbia. edu/projects/coversongs/covers80.*

Ellis, D. P. and Cotton, C. V. (2007), "The 2007 LabROSA cover song detection system," *MIREX 2007*.

Ellis, D. P. and Thierry, B.-M. (2012), "Large-scale cover song recognition using the 2d fourier transform magnitude," in *The 13th international society for music information retrieval conference*, pp. 241–246.

Erickson, J. and Whittlesey, K. (2005), "Greedy optimal homotopy and homology generators," in *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 1038–1046, Society for Industrial and Applied Mathematics.

Foote, J. (2000), "Automatic audio segmentation using a measure of audio novelty," in *Multimedia and Expo, 2000. ICME 2000. 2000 IEEE International Conference on*, vol. 1, pp. 452–455, IEEE.

Foucard, R., Durrieu, J.-L., Lagrange, M., and Richard, G. (2010), “Multimodal similarity between musical streams for cover version detection,” in *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, pp. 5514–5517, IEEE.

Frank, J., Mannor, S., and Precup, D. (2010), “Activity and Gait Recognition with Time-Delay Embeddings.” in *AAAI*, Citeseer.

Fujishima, T. (1999), “Realtime chord recognition of musical sound: A system using common lisp music,” in *Proc. ICMC*, vol. 1999, pp. 464–467.

Furui, S. (1986), “Speaker-independent isolated word recognition based on emphasized spectral dynamics,” in *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP’86.*, vol. 11, pp. 1991–1994, IEEE.

Ghaderian, M., Behrad, A., and Kaboodi, S. A. D. (2011), “Recognition of periodic motions using one-dimensional contour based features,” in *Machine Vision and Image Processing (MVIP), 2011 7th Iranian*, pp. 1–5, IEEE.

Gold, O. and Sharir, M. (2016), “Dynamic Time Warping and Geometric Edit Distance: Breaking the Quadratic Barrier,” *arXiv preprint arXiv:1607.05994*.

Goldenberg, R., Kimmel, R., Rivlin, E., and Rudzsky, M. (2005), “Behavior classification by eigendecomposition of periodic motions,” *Pattern Recognition*, 38, 1033–1043.

Gómez, E. (2006), “Tonal description of polyphonic audio for music content processing,” *INFORMS Journal on Computing*, 18, 294–304.

Gouyon, F., Dixon, S., and Widmer, G. (2007), “Evaluating low-level features for beat classification and tracking,” in *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, vol. 4, pp. IV–1309, IEEE.

Griffin, D. and Lim, J. (1984), “Signal estimation from modified short-time Fourier transform,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 32, 236–243.

Gromov, M. (2007), *Metric structures for Riemannian and non-Riemannian spaces*, Springer Science & Business Media.

Haitsma, J. and Kalker, T. (2002), “A highly robust audio fingerprinting system.” in *Ismir*, vol. 2002, pp. 107–115.

Hatcher, A. (2002), *Algebraic Topology*, Cambridge University Press.

Herbst, C. T., Unger, J., Herzel, H., Švec, J. G., and Lohscheller, J. (2016), “Phasegram analysis of vocal fold vibration documented with laryngeal high-speed video endoscopy,” *Journal of Voice*, 30, 771–e1.

Herzel, H., Berry, D., Titze, I. R., and Saleh, M. (1994), "Analysis of vocal disorders with methods from nonlinear dynamics," *Journal of Speech, Language, and Hearing Research*, 37, 1008–1019.

Herzel, H., Reuter, R., and Katz, R. A. (1996), "Biphonation in voice signals," in *AIP Conference Proceedings*, vol. 375, pp. 644–657, AIP.

Hsu, E., Pulli, K., and Popović, J. (2005), "Style translation for human motion," in *ACM Transactions on Graphics (TOG)*, vol. 24, pp. 1082–1089, ACM.

Huang, P., Hilton, A., and Starck, J. (2010), "Shape similarity for 3D video sequences of people," *International Journal of Computer Vision*, 89, 362–381.

Huang, S., Ying, X., Rong, J., Shang, Z., and Zha, H. (2016), "Camera calibration from periodic motion of a pedestrian," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3025–3033.

Humphrey, E. J., Nieto, O., and Bello, J. P. (2013), "Data Driven and Discriminative Projections for Large-Scale Cover Song Identification." in *ISMIR*, pp. 149–154.

Itakura, F. (1975), "Minimum prediction residual principle applied to speech recognition," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 23, 67–72.

Iwanski, J. S. and Bradley, E. (1998), "Recurrence plots of experimental data: To embed or not to embed?" *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 8, 861–871.

Jiang, X., Lim, L.-H., Yao, Y., and Ye, Y. (2011), "Statistical ranking and combinatorial Hodge theory," *Mathematical Programming*, 127, 203–244.

Junejo, I. N., Dexter, E., Laptev, I., and Pérez, P. (2008), "Cross-View Action Recognition from Temporal Self-similarities," in *Proceedings of the 10th European Conference on Computer Vision: Part II*, pp. 293–306, Springer-Verlag.

Junejo, I. N., Dexter, E., Laptev, I., and Perez, P. (2011), "View-independent action recognition from temporal self-similarities," *IEEE transactions on pattern analysis and machine intelligence*, 33, 172–185.

Kabsch, W. (1976), "A solution for the best rotation to relate two sets of vectors," *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography*, 32, 922–923.

Kaiser, F. and Sikora, T. (2010), "Music Structure Discovery in Popular Music using Non-negative Matrix Factorization." in *ISMIR*, pp. 429–434.

Kantz, H. and Schreiber, T. (2004), *Nonlinear time series analysis*, vol. 7, Cambridge university press.

Karni, Z. and Gotsman, C. (2000), “Spectral compression of mesh geometry,” in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pp. 279–286, ACM Press/Addison-Wesley Publishing Co.

Kazhdan, M. (2007), “An approximate and efficient method for optimal rotation alignment of 3D models,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29.

Kendall, M. G. (1938), “A new measure of rank correlation,” *Biometrika*, 30, 81–93.

Kennel, M. B., Brown, R., and Abarbanel, H. D. (1992), “Determining embedding dimension for phase-space reconstruction using a geometrical construction,” *Physical review A*, 45, 3403.

Kerber, M., Morozov, D., and Nigmetov, A. (2016), “Geometry helps to compare persistence diagrams,” in *2016 Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pp. 103–112, Society for Industrial and Applied Mathematics.

Kim, S., Unal, E., and Narayanan, S. (2008), “Music fingerprint extraction for classical music cover song identification,” in *Multimedia and Expo, 2008 IEEE International Conference on*, pp. 1261–1264, IEEE.

Krebs, F., Böck, S., and Widmer, G. (2015), “An Efficient State-Space Model for Joint Tempo and Meter Tracking.” in *ISMIR*, pp. 72–78.

Kronecker, L. (1884), *Näherungsweise ganzzahlige Auflösung linearer Gleichungen.*

Kumar, M., Veeraraghavan, A., and Sabharwal, A. (2015), “DistancePPG: Robust non-contact vital signs monitoring using a camera,” *Biomedical optics express*, 6, 1565–1588.

Kumdee, O. and Ritthipravat, P. (2015), “Repetitive motion detection for human behavior understanding from video images,” in *Signal Processing and Information Technology (ISSPIT), 2015 IEEE International Symposium on*, pp. 484–489, IEEE.

Levenshtein, V. I. (1966), “Binary codes capable of correcting deletions, insertions and reversals,” in *Soviet physics doklady*, vol. 10, p. 707.

Levy, O. and Wolf, L. (2015), “Live repetition counting,” in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3020–3028.

Lohscheller, J., Toy, H., Rosanowski, F., Eysholdt, U., and Döllinger, M. (2007), “Clinically evaluated procedure for the reconstruction of vocal fold vibrations from endoscopic digital high-speed videos,” *Medical image analysis*, 11, 400–413.

Mahadevan, S. (2008), "Representation discovery using harmonic analysis," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 2, 1–147.

Mahmoudi, M. and Sapiro, G. (2009), "Three-dimensional point cloud recognition via distributions of geometric distances," *Graphical Models*, 71, 22–31.

Marwan, N., Thiel, M., Nowaczyk, N., and Marwan, N. (2000), "Cross Recurrence Plot Based Synchronization of Time Series," *Nonlinear Processes in Geophysics*, 20, 101–107.

Marwan, N., Romano, M. C., Thiel, M., and Kurths, J. (2007), "Recurrence plots for the analysis of complex systems," *Physics reports*, 438, 237–329.

Maurel, P. and Sapiro, G. (2003), "Dynamic shapes average," .

McFee, B. and Ellis, D. P. (2014), "Analyzing song structure with spectral clustering," in *15th International Society for Music Information Retrieval (ISMIR) Conference*.

McGuire, G., Azar, N. B., and Shelhamer, M. (1997), "Recurrence matrices and the preservation of dynamical properties," *Physics Letters A*, 237, 43–47.

Mcleod, P. and Wyvill, G. (2005), "A smarter way to find pitch," in *In Proceedings of the International Computer Music Conference (ICMC'05*, pp. 138–141.

Mees, A., Rapp, P., and Jennings, L. (1987), "Singular-value decomposition and embedding dimension," *Physical Review A*, 36, 340.

Mehta, D. D., Deliyski, D. D., Quatieri, T. F., and Hillman, R. E. (2011), "Automated measurement of vocal fold vibratory asymmetry from high-speed videoendoscopy recordings," *Journal of Speech, Language, and Hearing Research*, 54, 47–54.

Miao, E. and Grimm, N. E. (2013), "The Blurred Lines of What Constitutes Copyright Infringement of Music: Robin Thicke v. Marvin Gaye's Estate," *WESTLAW J. INTELLECTUAL PROP.*, 20, 1.

Miller, G. A. (1956), "The magical number seven, plus or minus two: some limits on our capacity for processing information." *Psychological review*, 63, 81.

Milnor, J. W. (1997), *Topology from the differentiable viewpoint*, Princeton University Press.

Mokhtarian, F. and Mackworth, A. (1986), "Scale-based description and recognition of planar curves and two-dimensional shapes," *IEEE transactions on pattern analysis and machine intelligence*, pp. 34–43.

Mokhtarian, F., Abbasi, S., and Kittler, J. (1996), "Robust and Efficient Shape Indexing through Curvature Scale Space," in *Proceedings of the 1996 British Machine and Vision Conference BMVC*, vol. 96.

Müller, M. (2007), *Information retrieval for music and motion*, vol. 2, Springer.

Müller, M. and Ewert, S. (2011), "Chroma Toolbox: MATLAB implementations for extracting variants of chroma-based audio features," in *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR), 2011. hal-00727791, version 2-22 Oct 2012*, Citeseer.

Munkres, J. R. (1975), *Topology.*

Needleman, S. B. and Wunsch, C. D. (1970), "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of molecular biology*, 48, 443–453.

Neubauer, J., Mergell, P., Eysholdt, U., and Herzel, H. (2001), "Spatio-temporal analysis of irregular vocal fold oscillations: Biphonation due to desynchronization of spatial modes," *The Journal of the Acoustical Society of America*, 110, 3179–3192.

Nieto, O. and Bello, J. P. (2014), "Music Segment Similarity Using 2D-Fourier Magnitude Coefficients," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pp. 664–668, IEEE.

Niyogi, S. A., Adelson, E. H., et al. (1994), "Analyzing and recognizing walking figures in XYT," in *CVPR*, vol. 94, pp. 469–474.

Nolte, D. D. (2010), "The tangled tale of phase space," *Physics today*, 63, 33–38.

Osada, R., Funkhouser, T., Chazelle, B., and Dobkin, D. (2002), "Shape distributions," *ACM Transactions on Graphics (TOG)*, 21, 807–832.

Osmalsky, J., Embrechts, J.-J., Foster, P., and Dixon, S. (2015), "Combining features for cover song identification," in *16th International Society for Music Information Retrieval Conference.*

Osmalsky, J., Van Droogenbroeck, M., and Embrechts, J.-J. (2016), "Enhancing Cover Song Identification with Hierarchical Rank Aggregation," in *Proceedings of the 17th International for Music Information Retrieval Conference*, pp. 136–142.

Patterson, E. K., Gurbuz, S., Tufekci, Z., and Gowdy, J. N. (2002), "CUAVE: A new audio-visual database for multimodal human-computer interface research," in *Acoustics, Speech, and Signal Processing (ICASSP), 2002 IEEE International Conference on*, vol. 2, pp. II–2017, IEEE.

Perea, J. A. (2016), "Persistent homology of toroidal sliding window embeddings," in *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pp. 6435–6439, IEEE.

Perea, J. A. and Carlsson, G. (2014), "A klein-bottle-based dictionary for texture representation," *International journal of computer vision*, 107, 75–97.

Perea, J. A. and Harer, J. (2015), "Sliding windows and persistence: An application of topological methods to signal analysis," *Foundations of Computational Mathematics*, 15, 799–838.

Perea, J. A., Deckard, A., Haase, S. B., and Harer, J. (2015), "Sw1pers: Sliding windows and 1-persistence scoring; discovering periodicity in gene expression time series data," *BMC bioinformatics*, 16, 257.

Pinkall, U. and Polthier, K. (1993), "Computing discrete minimal surfaces and their conjugates," *Experimental mathematics*, 2, 15–36.

Pinsky, M. A. (2002), *Introduction to Fourier analysis and wavelets*, vol. 102, American Mathematical Soc.

Plesnik, E., Malgina, O., Tasic, J. F., Tomazic, S., and Zajc, M. (2014), "Detection and Delineation of the Electrocardiogram Qrs-complexes from Phase Portraits," .

Plotnik, A. M. and Rock, S. M. (2002), "Quantification of cyclic motion of marine animals from computer vision," in *OCEANS'02 MTS/IEEE*, vol. 3, pp. 1575–1581, IEEE.

Polana, R. and Nelson, R. C. (1997), "Detection and recognition of periodic, nonrigid motion," *International Journal of Computer Vision*, 23, 261–282.

Qiu, Q., Schutte, H., Gu, L., and Yu, Q. (2003), "An automatic method to quantify the vibration properties of human vocal folds via videokymography," *Folia Phoniatrica et Logopaedica*, 55, 128–136.

Quinton, E., Harte, C., and Sandler, M. (2015), "Extraction of Metrical Structure from Music Recordings," in *Proc. of the 18th Int. Conference on Digital Audio Effects (DAFx). Trondheim.*

Raichel, B. and Seshadhri, C. (2016), "Avoiding the Global Sort: A Faster Contour Tree Algorithm," in *LIPIcs-Leibniz International Proceedings in Informatics*, vol. 51, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Reininghaus, J., Huber, S., Bauer, U., and Kwitt, R. (2015), "A stable multi-scale kernel for topological machine learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4741–4748.

Reuter, M., Wolter, F.-E., and Peinecke, N. (2006), "Laplace–Beltrami spectra as 'Shape-DNA'of surfaces and solids," *Computer-Aided Design*, 38, 342–366.

Rubner, Y., Tomasi, C., and Guibas, L. J. (2000), "The earth mover's distance as a metric for image retrieval," *International journal of computer vision*, 40, 99–121.

Sakoe, H. and Chiba, S. (1970), "A similarity evaluation of speech patterns by dynamic programming," in *Nat. Meeting of Institute of Electronic Communications Engineers of Japan*, p. 136.

Sakoe, H. and Chiba, S. (1978), "Dynamic programming algorithm optimization for spoken word recognition," *IEEE transactions on acoustics, speech, and signal processing*, 26, 43–49.

Salamon, J., Serrà, J., and Gómez, E. (2012), "Melody, bass line, and harmony representations for music version identification," in *Proceedings of the 21st international conference companion on World Wide Web*, pp. 887–894, ACM.

Schödl, A., Szeliski, R., Salesin, D. H., and Essa, I. (2000), "Video textures," in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pp. 489–498, ACM Press/Addison-Wesley Publishing Co.

Schuldt, C., Laptev, I., and Caputo, B. (2004), "Recognizing human actions: a local SVM approach," in *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, vol. 3, pp. 32–36, IEEE.

Schwarz, D., Cahen, R., and Britton, S. (2008), "Principles and applications of interactive corpus-based concatenative synthesis," in *Journées d'Informatique Musicale (JIM)*, pp. 1–1.

Seitz, S. M. and Dyer, C. R. (1997), "View-invariant analysis of cyclic motion," *International Journal of Computer Vision*, 25, 231–251.

Serra, J. (2007), "Music similarity based on sequences of descriptors: tonal features applied to audio cover song identification," *Department of Information and Communication Technologies, Universitat Pompeu Fabra, Barcelona, Spain.*

Serra, J., Gómez, E., Herrera, P., and Serra, X. (2008a), "Chroma binary similarity and local alignment applied to cover song identification," *Audio, Speech, and Language Processing, IEEE Transactions on*, 16, 1138–1151.

Serra, J., Gómez, E., and Herrera, P. (2008b), "Transposing chroma representations to a common key," in *IEEE CS Conference on The Use of Symbols to Represent Music and Multimedia Objects*, pp. 45–48.

Serra, J., Serra, X., and Andrzejak, R. G. (2009), "Cross recurrence quantification for cover song identification," *New Journal of Physics*, 11, 093017.

Serrà, J., Zanin, M., Herrera, P., and Serra, X. (2012), "Characterization and exploitation of community structure in cover song networks," *Pattern Recognition Letters*, 33, 1032–1041.

Serra, J., Müller, M., Grosche, P., and Arcos, J. L. (2012), "Unsupervised detection of music boundaries by time series structure features," in *Twenty-Sixth AAAI Conference on Artificial Intelligence*.

Sheehy, D. R. (2013), "Linear-Size Approximations to the Vietoris-Rips Filtration," *Discrete & Computational Geometry*, 49, 778–796.

Silva, D. F., Yeh, C.-C. M., Batista, G. E. d. A. P. A., Keogh, E., et al. (2016), "SiMPle: assessing music similarity using subsequences joins," in *International Society for Music Information Retrieval Conference, XVII*, International Society for Music Information Retrieval-ISMIR.

Smith, T. F. and Waterman, M. S. (1981), "Identification of common molecular subsequences," *Journal of molecular biology*, 147, 195–197.

Sprechmann, P., Bronstein, A., Morel, J.-M., and Sapiro, G. (2013), "Audio restoration from multiple copies," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pp. 878–882, IEEE.

Stam, C. J. (2005), "Nonlinear dynamical analysis of EEG and MEG: review of an emerging field," *Clinical Neurophysiology*, 116, 2266–2301.

Steenrod, S. E.-N. and Eilenberg, S. (1952), "Foundations of algebraic topology," .

Takens, F. (1981), "Detecting strange attractors in turbulence," in *Dynamical systems and turbulence, Warwick 1980*, pp. 366–381, Springer.

Taubin, G. (1995), "A signal processing approach to fair surface design," in *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pp. 351–358, ACM.

Tavenard, R., Jégou, H., and Lagrange, M. (2012), "Efficient Cover Song Identification using approximate nearest neighbors," .

ten Holt, G. A., Reinders, M. J., and Hendriks, E. (2007), "Multi-dimensional dynamic time warping for gesture recognition," in *Thirteenth annual conference of the Advanced School for Computing and Imaging*, vol. 300.

Toponogov, V. A. (2006), *Differential geometry of curves and surfaces*, Springer.

Tralie, C. (2016), "High-Dimensional Geometry of Sliding Window Embeddings of Periodic Videos," in *LIPIcs-Leibniz International Proceedings in Informatics*, vol. 51, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Tralie, C. J. and Bendich, P. (2015), "Cover Song Identification with Timbral Shape Sequences," in *16th International Society for Music Information Retrieval (ISMIR)*, pp. 38–44.

Trigeorgis, G., Nicolaou, M. A., Zafeiriou, S., and Schuller, B. W. (2016), "Deep Canonical Time Warping," .

Tulyakov, S., Alameda-Pineda, X., Ricci, E., Yin, L., Cohn, J. F., and Sebe, N. (2016), "Self-adaptive matrix completion for heart rate estimation from face videos under realistic conditions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2396–2404.

Turk, M. and Pentland, A. (1991), "Eigenfaces for recognition," *Journal of cognitive neuroscience*, 3, 71–86.

Tzanetakis, G. and Cook, P. (2002), "Musical genre classification of audio signals," *IEEE Transactions on speech and audio processing*, 10, 293–302.

Urbano, J., Lloréns, J., Morato, J., and Sánchez-Cuadrado, S. (2011), "Melodic similarity through shape similarity," in *Exploring music contents*, pp. 338–355, Springer.

Vejdemo-Johansson, M., Pokorny, F. T., Skraba, P., and Kragic, D. (2015), "Cohomological learning of periodic motion," *Applicable Algebra in Engineering, Communication and Computing*, 26, 5–26.

Venkataraman, V. and Turaga, P. (2016), "Shape Descriptions of Nonlinear Dynamical Systems for Video-based Inference." *IEEE transactions on pattern analysis and machine intelligence.*

Wadhwa, N., Rubinstein, M., Durand, F., and Freeman, W. T. (2013), "Phase-based video motion processing," *ACM Transactions on Graphics (TOG)*, 32, 80.

Wang, A. (2006), "The Shazam music recognition service," *Communications of the ACM*, 49, 44–48.

Wang, A. et al. (2003), "An Industrial Strength Audio Search Algorithm." in *ISMIR*, pp. 7–13, Washington, DC.

Wang, B., Jiang, J., Wang, W., Zhou, Z.-H., and Tu, Z. (2012), "Unsupervised metric fusion by cross diffusion," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pp. 2997–3004, IEEE.

Wang, B., Mezlini, A. M., Demir, F., Fiume, M., Tu, Z., Brudno, M., Haibe-Kains, B., and Goldenberg, A. (2014), "Similarity network fusion for aggregating data types on a genomic scale," *Nature methods*, 11, 333–337.

Wang, P., Abowd, G. D., and Rehg, J. M. (2009), "Quasi-periodic event analysis for social game retrieval," in *Computer Vision, 2009 IEEE 12th International Conference on*, pp. 112–119, IEEE.

Waterman, M. S. (1995), *Introduction to computational biology: maps, sequences and genomes*, CRC Press.

Wehbe, H., Joly, P., and Haidar, B. (2015), "Automatic detection of repetitive actions in a video," in *Content-Based Multimedia Indexing (CBMI), 2015 13th International Workshop on*, pp. 1–6, IEEE.

Whaba, G. (1965), "A least squares estimate of spacecraft attitude," *SIAM Review*, 7, 409.

Wilden, I., Herzel, H., Peters, G., and Tembrock, G. (1998), "Subharmonics, biphonation, and deterministic chaos in mammal vocalization," *Bioacoustics*, 9, 171–196.

Wittenberg, T., Moser, M., Tigges, M., and Eysholdt, U. (1995), "Recording, processing, and analysis of digital high-speed sequences in glottography," *Machine vision and applications*, 8, 399–404.

Wu, Z., Jiang, S., and Huang, Q. (2009), "Near-duplicate video matching with transformation recognition," in *Proceedings of the 17th ACM international conference on Multimedia*, pp. 549–552, ACM.

Yair, O., Talmon, R., Coifman, R. R., and Kevrekidis, I. G. (2016), "No equations, no parameters, no variables: data, and the reconstruction of normal forms by learning informed observation geometries," *arXiv preprint arXiv:1612.03195.*

Yang, J., Zhang, H., and Peng, G. (2016), "Time-domain period detection in short-duration videos," *Signal, Image and Video Processing*, 10, 695–702.

Yeh, M.-C. and Cheng, K.-T. (2009), "Video copy detection by fast sequence matching," in *Proceedings of the ACM International Conference on Image and Video Retrieval*, p. 45, ACM.

Ying, R., Pan, J., Fox, K., and Agarwal, P. K. (2016), "A Simple Efficient Approximation Algorithm for Dynamic Time Warping," in *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '16, pp. 21:1–21:10, New York, NY, USA, ACM.

Yu, C. W., Kwong, K., Lee, K.-H., and Leong, P. H. W. (2005), "A Smith-Waterman systolic cell," in *New Algorithms, Architectures and Applications for Reconfigurable Computing*, pp. 291–300, Springer.

Yu, G., Sapiro, G., and Mallat, S. (2012), "Solving inverse problems with piecewise linear estimators: From Gaussian mixture models to structured sparsity," *IEEE Transactions on Image Processing*, 21, 2481–2499.

Yuille, A. L. and Poggio, T. (1985), "Fingerprints theorems for zero crossings," *JOSA A*, 2, 683–692.

Zacharias, S. R., Myer, C. M., Meinzen-Derr, J., Kelchner, L., Deliyski, D. D., and de Alarcón, A. (2016), "Comparison of Videostroboscopy and High-speed Videoendoscopy in Evaluation of Supraglottic Phonation," *Annals of Otology, Rhinology & Laryngology*, p. 0003489416656205.

Zhao, W., Gao, S., and Lin, H. (2007), "A robust hole-filling algorithm for triangular mesh," *The Visual Computer*, 23, 987–997.

Zhou, F. and De la Torre, F. (2016), "Generalized canonical time warping," *IEEE transactions on pattern analysis and machine intelligence*, 38, 279–294.

Zhou, F. and Torre, F. (2009), "Canonical time warping for alignment of human behavior," in *Advances in neural information processing systems*, pp. 2286–2294.

Alphabetical Index

Biography

Christopher Tralie was born in Philadelphia, Pennsylvania in 1989, and he spent most of his childhood in Fort Washington, PA. He was an avid coder throughout middle school and high school, and he was a counselor at the summer camp "Fun with Math, Science, And Computers" sponsored by Temple University. He attended the PA Governor's School for the Sciences Summer 2006, and he graduated from Upper Dublin High School in 2007. He got his bachelor's degree from Princeton University in 2011 in Electrical Engineering (cum laude) with a certificate in Computer Science, where he focused on signal processing and computer graphics, and he was awarded the G. David Forney prize in Signals And Systems. He also regularly tutored multivariable calculus to engineers, he did a robotics REU at Duke University in 2009, and he was a member of Terrace F. Club (Food = Love).

In 2011, he was awarded the NSF Graduate Fellowship, and he began his Ph.D. studies at Duke University in Electrical And Computer Engineering. His initial research was on fusing electro-optical sensors and radar sensors using geometry, and he also did some work on robotics in healthcare (Deyle et al. (2013)). Halfway through the third year of his studies, his adviser left Duke. Christopher then received his master's degree in ECE and promptly switched his research focus to multimedia time series analysis, as one of the first graduate students in the Information Initiative at Duke (IID). He has since published works on automatic cover song identification (Tralie and Bendich (2015)), periodic videos (Tralie (2016)), and geometric music structure analysis (Bendich et al. (2016)). Outside of his own research, he supervised five undergraduate student research projects, and he was awarded the Bass Family Undergraduate Teaching Fellowship, with which he designed and taught a course from scratch called "Digital 3D Geometry" to 31 students, which was cross-listed between CS and math. This course was rated in the top 5% of all spring 2016 courses university-wide in "quality of course/intellectual stimulation" in student evaluations.

Christopher is part of a very large family with 32 first cousins, most of whom are in the Philadelphia area. He was the concert master of his high school orchestra, and he played 2nd violin in the Duke Symphony Orchestra his second year in grad school. During grad school, he also volunteered with Habitat for Humanity, and he was the president of the Buddhist Meditation Community At Duke (BMCD) for a semester. He also skateboarded very poorly at the Durham skatepark for a while until one day he fell 7 feet in a concrete pool and John Harer encouraged him to quit.