Carnegie Mellon University
Dietrich College of Humanities and Social Sciences
Dissertation

Submitted in Partial Fulfillment of the Requirements
For the Degree of Doctor of Philosophy

Title: The Sliding Window Discrete Fourier Transform

Presented by: Lee F. Richardson

Accepted by: Department of Statistics and Data Science

Readers:

William F. Eddy, Advisor

Max G'Sell

John Lehoczky

Chad Schafer

Samuel L. Ventura

Approved by the Committee on Graduate Degrees:

Richard Scheines, Dean Date

Carnegie Mellon University

The Sliding Window Discrete Fourier Transform

A Dissertation Submitted to the Graduate School
in Partial Fulfillment of the Requirements for the degree

Doctor of Philosophy
in
Statistics and Data Science

by

Lee F. Richardson
Department of Statistics and Data Science
Carnegie Mellon University
Pittsburgh, PA 15213

April 2019

Acknowledgments

To start, I would like to thank my advisor Bill Eddy. Bill's patience helped me get started as a researcher, and I am grateful to him for suggesting the topic of this thesis, which turned out to have many open research problems.

I would also like to thank my committee- Max G'Sell, John Lehoczky, Chad Schafer, and Sam Ventura. A special thanks goes to John, who edited my entire thesis document, and wrote several letters on my behalf. Another special thank you goes to Sam, who helped me on just about every project I worked on at CMU. Sam even helped me before I got here, when I randomly e-mailed him about admission to CMU.

I also want to thank other members of the CMU faculty and staff. Thanks to Peter Freeman and Rebecca Nugent for everything they've done. Thanks to Valerie Ventura, Cosma Shalizi, and Peter Adams for advising my ADA project. Thanks to Ryan Tibshirani for organizing some epic basketball games. Thanks to David Creswell and Howard Seltman for including me on their Life@CMU project. Finally, thanks to Margie, Jessica, Heidi, Kira, Laura, Mari, Beth and Carloz for all of the help.

Before CMU, I had many great math teachers. Mr. O'Connor, Mr. Nutting, Mrs. Bowers, and Mr. Schmidt all helped my decision to pursue math and statistics in college. In college, I certainly wouldn't have done a PhD without the encouragement of Peter Guttorp, June Morita, and Don Percival. Finally, thanks to Dorit Hammerling, Doug Nychka, and everyone at the Institude of Health Metrics and Evaluation (IHME) who helped me along the way.

It would be difficult to get through a PhD without great classmates, so I want to thank some of them here. Thanks to Jining, Taylor, Collin, Shannon, Zongge, and Francesca for teaming up with me on various *hack-a-thons*. Thanks to Yotam, Purvasha, and Natalie for being great friends. Thanks to Peter and Alex for the fun discussions in the *history of stats* reading group, thanks to Ron for organizing the *statistics in sports* research group, thanks to Sangwon and Jerzy for their sage advice and Goodreads friendship, and thanks to the Random Walkers for five great years of intramural sports, which includes three championships. There's many more people who helped me along the way, and I only listed a few here, so apologies to anyone I missed.

I am also very lucky to have great family and friends. Thanks to Mitch, Matt, Korey, Karson, and Tobia for always making me feel back at home in Seattle. Perhaps the biggest accomplishment of my PhD was making it to our fantasy football draft each year. Thanks to the Pittsburgh New Church for their hospitality. And of course, thanks to my parents, Suzanne and Peter, and my brother Harris, who were unconditionally supportive, and always trusted my intuition on what to do next.

Most importantly, I would like to thank my wife Francesca Matano. My life really turned around when I met Francesca, and I couldn't imagine doing a PhD without her. I love you, and I'm so excited to be California Dreamin' with you next year!

“*The river runs deep*”

- Knowmads

Abstract

The discrete Fourier transform (DFT) is a widely used tool across science and engineering. Nevertheless, the DFT assumes that the frequency characteristics of a signal remain constant over time, and is unable to detect local changes. Researchers beginning with Gabor (1946) addressed this shortcoming by inventing methods to obtain time-frequency representations, and this thesis focuses on one such method: the Sliding Window Discrete Fourier Transform (SWDFT). Whereas the DFT operates on an entire signal, the SWDFT takes an ordered sequence of smaller DFTs on contiguous subsets of a signal. The SWDFT is a fundamental tool in time-frequency analysis, and is used in a variety of applications, such as spectrogram estimation, image enhancement, neural networks, and more.

This thesis studies the SWDFT from three perspectives: algorithmic, statistical, and applied. Algorithmically, we introduce the Tree SWDFT algorithm, and extend it to arbitrary dimensions. Statistically, we derive the marginal distribution and covariance structure of SWDFT coefficients for white noise signals, which allows us to characterize the SWDFT coefficients as a Gaussian process with a known covariance. We also propose a localized version of cosine regression, and show that the approximate maximum likelihood estimate of the frequency parameter in this model is the maximum SWDFT coefficient over all possible window sizes. From an applied perspective, we introduce a new algorithm to decompose signals with multiple non-stationary periodic components, called matching demodulation. We demonstrate the utility of matching demodulation in an analysis of local field potential recordings from a neuroscience experiment.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Time-series methods are often partitioned into two groups: the **time domain** and the **frequency domain**. Time domain methods focus on the temporal dependence between observations. For example, a time domain method would use the fact that the temperature in Los Angeles is similar from Monday to Tuesday. Alternatively, frequency domain methods imagine that signals are composed of oscillations, such as the daily rise and fall of the sun, and study these oscillations using tools based on the discrete Fourier transform (DFT). But some signals have both time *and* frequency components; see, for example, the top panel of Figure 1.1. With signals like this, practitioners resort to **time-frequency** representations.

Time-frequency representations are self explanatory; they provide a 2D representation of a signal, where the first dimension is frequency, and the second dimension is time. Time-frequency representations are complex-valued, so they have both amplitude and phase components. For example, if $x(t)$ is a signal, then the time-frequency representation of $x(t)$, denoted $S(f, t)$ is given by

$$S(f, t) = |S(f, t)|e^{i\phi(f,t)}. \tag{1.1}$$

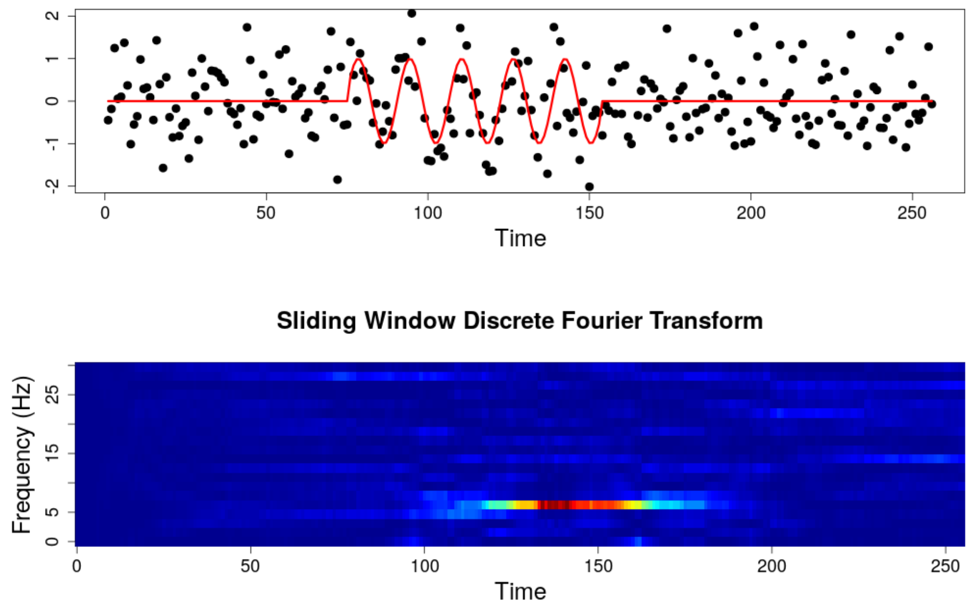

Figure 1.1: Top: A local cosine signal plus noise, where the red line is the signal, and the black dots are the signal plus noise. Bottom: The sliding window DFT of the the top panel. The y-axis represents frequency, and the x-axis represents time.

In (1.1), $|S(f,t)|$ represents amplitude at frequency f and time t, and $\phi(f,t)$ represents phase. Time-frequency representations are typically visualized using *spectrograms*, which display the squared amplitude of a signal over time and frequency. An example spectrogram is shown on the bottom panel of Figure 1.1.

The first time-frequency representation is due to Gabor (1946), who summarized his motivation with the following quote:

Hitherto communication theory was based on two alternative methods of signal analysis. One is the description of the signal as a function of time; the other is Fourier analysis. Both are idealizations, as the first method operates with sharply defined instants of time, the second with infinite wave trains of rigorously defined frequencies. But our everyday experiences- especially our auditory sensations- insist on a description in terms of both time and frequency.

In this paper, Gabor invents the first time-frequency representation, now called the *Gabor transform*. The Gabor transform multiplies a signal by a Gaussian window function, then obtains a time-frequency representation by taking a Fourier transform of the resulting signal. More than seventy years later, the Gabor transform still finds applications, and the underlying ideas are closely related to this thesis.

Ever since Gabor (1946), various methods to obtain time-frequency representations have been proposed. One such method, which we call the Sliding Window Discrete Fourier Transform (SWDFT), is the focus of this thesis. The SWDFT goes by other names, such as the windowed Fourier transform, and most commonly, the short-time Fourier transform. This thesis uses *sliding window* instead or *short-time*, because when a signals dimension is larger than one (e.g. a 2D image), the second dimension no longer represents time.

But other time-frequency transforms exist, such as the wavelet and Hilbert transforms, so why focus on the SWDFT? The main reason is that, just as the DFT is widely used in the frequency domain, the SWDFT is one of, if not the, most widely used time-frequency transformations. In fact, most textbooks start with the SWDFT before moving to other transforms (e.g. Cohen (1995) and Mallat (1999)). Another reason is that, from the eyes of the practitioner, no time-frequency transformation is the best choice in all situations, a point well summarized in the introduction of Vetterli et al. (2014):

The local Fourier and wavelet methods are presented side-by-side, without favoring any one in particular; the truth is that each representation is a tool in the toolbox of the practitioner, and the problem or application at hand ultimately determines the appropriate one to use.

The final reason we give is that, since the SWDFT is a straightforward extension of the DFT, it is the easiest time-frequency transform to understand. And just as it's smart to understand logistic regression before neural networks, it makes sense to understand the SWDFT before moving to more complicated transforms.

This thesis provides an in-depth study of the SWDFT, which is important for several reasons. First, despite the ubiquity of spectrograms across science and engineering, there have not been many statistical studies of the underlying computations. In fact, during the flurry of frequency domain statistics research in the 1980's (e.g. Brillinger and Krishnaiah (1983)), there was more emphasis on purely frequency domain methods. This gap opens up many opportunities to study how frequency domain methods carry over to the time-frequency case, which we pursue in this thesis. Finally, from an algorithmic perspective, the Tree SWDFT algorithm we propose is a novel contribution to the literature (Richardson and Eddy (2019)).

1.1 Outline

This thesis is organized in four parts: algorithms, statistical properties, local cosine regression, and an application. Before delving into these topics, Chapter 2 defines the SWDFT and sets the notation and terminology used throughout the thesis.

The purpose of Chapters 3 and 4 is introducing the Tree SWDFT algorithm. To start, Chapter 3 provides a detailed synthesis of the literature on algorithms to compute the SWDFT. This helps, because the literature on SWDFT algorithms is fairly scattered. After this review, we introduce four different SWDFT algorithms, including our Tree SWDFT algorithm, and compare the properties of these algorithms. Chapter 4 moves from the 1D SWDFT to the multidimensional case. We derive the three algorithms for the multidimensional SWDFT, then generalize the Tree SWDFT algorithm to arbitrary dimensions, where our multidimensional Tree SWDFT derivation is more general than the derivation given in Richardson and Eddy (2019).

The focus shifts from algorithms to statistics in Chapter 5. This chapter derives the marginal distribution and covariance structure of the SWDFT coefficients for white noise signals. This is crucial, because an important statistical difference between DFT and SWDFT coefficients is that DFT coefficients are independent. This difference matters because many statistical tests based on the DFT coefficients, such as Fisher's periodicity test (Fisher (1929)), rely on the independence property. Therefore, to construct reliable tests for the SWDFT coefficients, a first step is understanding their dependence.

Chapter 6 studies the widely used method of *cosine regression*. It has been known since Whittle (1952) that the maximum of the periodogram (i.e. the largest DFT coefficient) is the least squares solution for the frequency component in the cosine regression model. We extend this result to the time-frequency case, by showing that the maximum of a spectrogram function over window size, which is a continuous analog of the SWDFT, is the maximum likelihood estimate for frequency, signal length, and signal starting position in our proposed local cosine regression model.

In Chapters 7 and 8, we demonstrate how the SWDFT can be used in a data analysis. Chapter 7 shows that when a signal has a time-varying amplitude or phase, the local cosine regression model we proposed Chapter 6 breaks down. To overcome this, we introduce the method of complex demodulation, which extracts a time-varying amplitude and phase from a signal. We show that, under certain parameterizations, complex demodulation is mathematically equivalent to the SWDFT. This equivalence is important, because it tells us that the modulus of SWDFT coefficients is proportional to the time-varying amplitude extracted by complex demodulation.

Chapter 8 uses ideas related to complex demodulation to analyze a neuroscience dataset. The goal of this analysis is characterizing local oscillations, called *bursts*, that occur when a monkey holds objects in its working memory. We propose a new statistical model to analyze the bursting behavior, and introduce the *matching demodulation* algorithm to fit this model. Matching demodulation repeatedly applies complex demodulation to different frequency bands, where the frequency band is automatically selected by the maximum SWDFT coefficient. We demonstrate that matching demodulation produces flexible oscillating functions that fit this dataset quite well. Importantly, and different than non-parametric regression methods such as smoothing splines, matching demodulation allows us to interpret the results in terms of amplitude, frequency, and phase parameters.

Chapter 9 summarizes the thesis, articulates several future research directions, and provides some concluding remarks.

1.2 Summary of Contributions

This thesis makes four primary contributions:

1. **The Tree SWDFT algorithm**. Although the underlying idea has existed since at least Covell and Richardson (1991), no one has explicitly defined the underlying data-structure, nor showed how this data-structure leads to a fast algorithm. In contrast, we derive the mathematics underlying the data-structure and algorithm, and instantiate the algorithm with a publicly available software implementation (Richardson and Eddy (2019)). We also generalized the algorithm to work in arbitrary dimensions, which has never been done.

2. **The asymptotic joint distribution of the SWDFT for white noise signals is a Gaussian process**. We derive the asymptotic marginal distribution of SWDFT coefficients, and the covariance between SWDFT coefficients, which is enough to make this characterization.

3. **Local cosine regression**. To the best of our knowledge, we are the first to localize the cosine regression model, and show that the maximum SWDFT coefficient over all possible window sizes is the approximate maximum likelihood estimate of our proposed model.

4. **The matching demodulation algorithm**. Our algorithm to decompose signals with multiple non-stationary periodic components, and the correspondence of this algorithm with a statistical model, is new. We also demonstrate the utility of the algorithm on a neuroscience dataset, which shows that the method can work for complicated signals.

Chapter 2

The Sliding Window Discrete Fourier Transform

This chapter defines the Sliding Window Discrete Fourier Transform (SWDFT) and sets the notation and terminology used in this thesis. Briefly described, the SWDFT takes an ordered sequence of discrete Fourier transforms (DFTs) on contiguous subsets of a signal. The SWDFT outputs a 2D array of coefficients, where the first dimension represents frequency, and the second dimension represents window position. When signals are one dimensional, the second dimension typically represents time. Since the SWDFT is composed of DFTs, we start by defining the DFT, then move to the SWDFT.

2.1 The Discrete Fourier Transform

The discrete Fourier transform is a formula used to calculate N complex-valued coefficients from a length N real or complex-valued signal. This thesis focuses on real-valued signals, since real-valued signals are more common in practice.

Let $\mathbf{x} = [x_0, x_1, \ldots, x_{N-1}]$ be a length N real-valued signal. The DFT of $\mathbf{x}$ is

$$\begin{aligned} a_k &= \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j \omega_N^{-jk}, \\ k &= 0, 1, \ldots N-1, \end{aligned} \tag{2.1}$$

where

$$\omega_N = \exp(\frac{2\pi i}{N}) = \cos(\frac{2\pi}{N}) + i\sin(\frac{2\pi}{N}). \tag{2.2}$$

An example length eight DFT is shown in Figure 2.1.

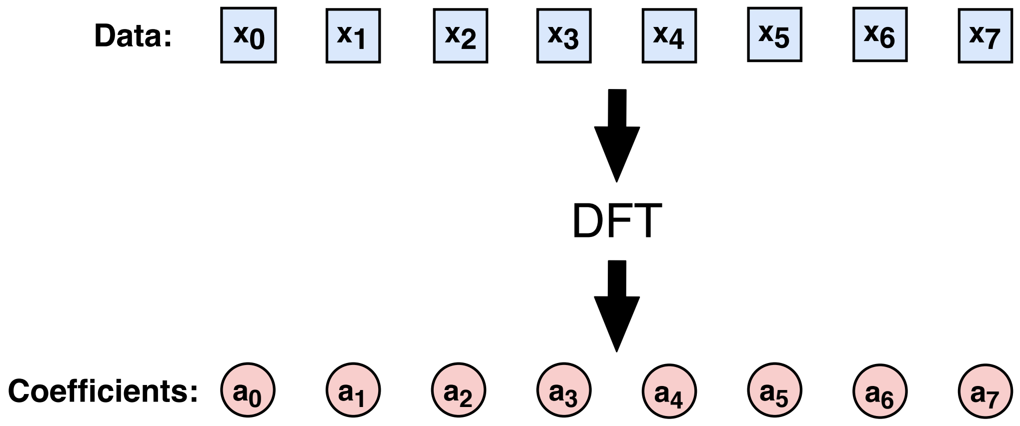

Figure 2.1: The discrete Fourier transform for a length eight signal $\mathbf{x} = [x_0, x_1, \ldots, x_7]$ computes eight complex-valued coefficients $\mathbf{a} = [a_0, a_1, \ldots, a_7]$.

DFT coefficients measure the correlation between a signal ($\mathbf{x}$) and sine and cosine functions with particular frequencies. To be explicit, DFT coefficients are complex-valued; the real part measures the correlation with a cosine function, the imaginary part measures the correlation with a sine function, and both the sine and cosine functions have the same frequency. For example, the real and imaginary parts of the k^{th} DFT coefficient (a_k) are:

$$
\begin{aligned}
Re(a_k) &= \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j \cos(2\pi \frac{k}{N} j), \\
Im(a_k) &= \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j \sin(2\pi \frac{k}{N} j).
\end{aligned}
$$

This explains why researchers are interested in large DFT coefficients; a large DFT coefficient indicates high correlation between our signal ($\mathbf{x}$) and sine and cosine functions with a particular frequency.

The input signal ($\mathbf{x}$) can be recovered by an inverse DFT

$$
\begin{aligned}
x_j &= \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} a_k \omega_N^{jk}, \\
j &= 0, 1, \ldots, N-1. \qquad (2.3)
\end{aligned}
$$

The normalization factor used in (2.1) and (2.3) is not strictly required. For (2.1) and (2.3) to be correct, the only requirement is that the product of the normalization factors equals $\frac{1}{N}$. For example, we could have used a normalization of $\frac{1}{N}$ in (2.1), and no normalization factor in (2.3), or vice-versa.

You can also define the DFT with matrix-vector notation

$$
\mathbf{a} = \mathbf{F}\mathbf{x}, \qquad (2.4)
$$

where $\mathbf{a} \in \mathbb{C}^N, \mathbf{x} \in \mathbb{R}^N$ and $\mathbf{F} \in \mathbb{C}^{N \times N}$.

The matrix $\mathbf{F}$ is known as the Fourier matrix, and is a special case of a Vandermonde matrix. Vandermonde matrices have terms of a geometric series in each row, and the entries of $\mathbf{F}$ are

$$
F_{k,j} = \frac{1}{\sqrt{N}} \omega_N^{-(k-1)(j-1)}. \qquad (2.5)
$$

The inverse DFT in matrix-vector notation is

$$\mathbf{F^{-1}a} = \mathbf{x}. \tag{2.6}$$

Matrix-vector notation illustrates one way to think about the DFT: as a model for data. This model assumes that our signal is a linear combination of sine and cosine functions with different frequencies. The next section defines the exact frequencies used by the DFT, called the *Fourier frequencies.*

2.1.1 Fourier Frequencies

The set of frequencies used by the DFT is called the Fourier frequencies, given by

$$\begin{aligned} f_k &= \frac{2\pi k}{n}, \\ k &= 0, 1, \ldots, n-1. \end{aligned} \tag{2.7}$$

The important fact regarding the Fourier frequencies is that they are orthogonal, and they form an orthogonal basis for $\mathbb{R}^n$ (see Chapter 2 of Bloomfield (2004) for a derivation).

2.1.2 Output Types

Becuase DFT coefficients are complex numbers, there are multiple ways to view them, which we call *output types.* The five most common output types are:

1. a_k: Complex number.
2. $Re(a_k)$: Real part.
3. $Im(a_k)$: Imaginary part.
4. $|a_k|^2 = Re(a_k)^2 + Im(a_k)^2$: Squared modulus.

5. $\arg(a_k) = \tan^{-1}(\frac{Im(a_k)}{Re(a_k)})$: Argument (or, phase).

We use each output type in different parts of this thesis. Following signal processing conventions, we sometimes refer to the squared modulus of DFT coefficients as the *energy* at frequency k. For a more detailed introduction to complex numbers, which we reference throughout the thesis, see Appendix B.

Finally, we often reference the largest, or maximum SWDFT coefficient. When we say this, we implicitly refer to the squared modulus ($|a_{k,p}|^2$) output type.

2.1.3 The Multidimensional DFT

The multidimensional DFT is a straightforward extension of the 1D DFT. For example, let $\mathbf{x}$ be a 2D, $N_0 \times N_1$ real-valued signal. The 2D DFT of $\mathbf{x}$ is

$$\begin{aligned} a_{k_0,k_1} &= \frac{1}{\sqrt{N_0 N_1}} \sum_{j_0=0}^{N_0-1} \sum_{j_1=0}^{N_1-1} x_{j_0,j_1} \omega_{N_0}^{-j_0 k_0} \omega_{N_1}^{-j_1 k_1}, \\ k_0 &= 0, 1, \ldots, N_0 - 1, \\ k_1 &= 0, 1, \ldots, N_1 - 1. \end{aligned} \tag{2.8}$$

This definition extends to any dimension. For example, the d-dimensional DFT is given by

$$\begin{aligned} a_{k_0,k_1,\ldots,k_{d-1}} &= \frac{1}{\sqrt{N_0 N_1 \ldots N_{d-1}}} \sum_{j_0=0}^{N_0-1} \sum_{j_1=0}^{N_1-1} \cdots \sum_{j_{d-1}=0}^{N_{d-1}-1} x_{j_0,j_1,\ldots,j_{d-1}} \omega_{N_0}^{-j_0 k_0} \omega_{N_1}^{-j_1 k_1} \ldots \omega_{N_{d-1}}^{-j_{d-1} k_{d-1}}, \\ k_0 &= 0, 1, \ldots, N_0 - 1, \\ k_1 &= 0, 1, \ldots, N_1 - 1, \\ \ldots & \quad , \\ k_{d-1} &= 0, 1, \ldots N_{d-1} - 1. \end{aligned} \tag{2.9}$$

2.2 The Sliding Window Discrete Fourier Transform

Now that we've defined the DFT, we can define the SWDFT as a sequence of DFTs multiplied by a contiguous sliding window function, where the window function is only nonzero for $n \leq N$ data points. In our terminology, we refer to n as the *window size*. Formally, the SWDFT is given by

$$\begin{aligned} a_{k,p} &= \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} c_j x_{p-n+1+j} \omega_n^{-jk}, \\ k &= 0, 1, \ldots, n-1, \\ p &= n-1, n, \ldots, N-1. \end{aligned} \tag{2.10}$$

In (2.10), the nonzero coefficients of the sliding window function are $\mathbf{c} = [c_0, c_1, \ldots, c_{n-1}]$. In order to keep our definition aligned with the DFT, this thesis primarily uses a rectangular sliding window function

$$\mathbb{1}_{p-n+1,p}(j) = \begin{cases} 1 & p-n+1 \leq j \leq p, \\ 0 & \text{otherwise.} \end{cases} \tag{2.11}$$

In some situations, for example to taper the ends of a signal, we use a different sliding window function. But unless otherwise stated, you can assume that a rectangular sliding window function is used. Applying (2.11) to (2.10) gives the following definition SWDFT definition

$$\begin{aligned} a_{k,p} &= \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} x_{p-n+1+j} \omega_n^{-jk}, \\ k &= 0, 1, \ldots, n-1, \\ p &= n-1, n, \ldots, N-1. \end{aligned} \tag{2.12}$$

(2.12) outputs an $n \times P$ array of SWDFT coefficients, where $P = N - n + 1$ is the number of window positions. In the SWDFT output array, the n dimension corresponds to frequency, and the P dimension corresponds to window position, which typically represents time. In (2.12), the index p is the rightmost data point in each window position. An example of how (2.12) works is given in Figure 2.2.

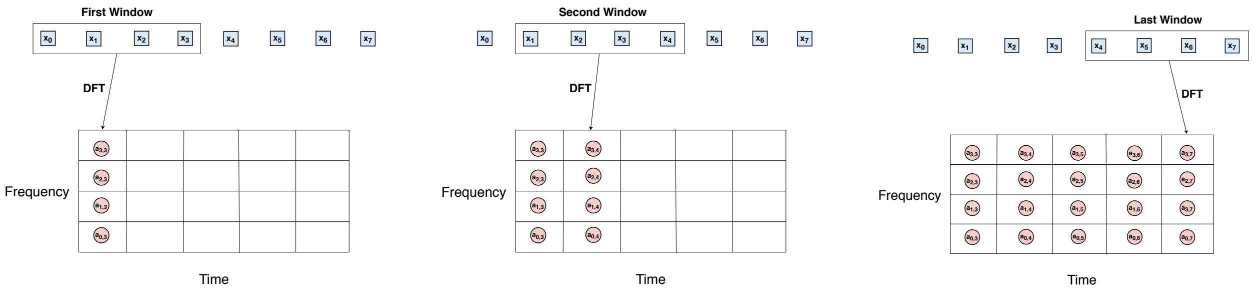
First Window
Second Window
Last Window
x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7
DFT
Frequency
Time
$a_{3,3}$ $a_{3,4}$ $a_{3,5}$ $a_{3,6}$ $a_{3,7}$
$a_{2,3}$ $a_{2,4}$ $a_{2,5}$ $a_{2,6}$ $a_{2,7}$
$a_{1,3}$ $a_{1,4}$ $a_{1,5}$ $a_{1,6}$ $a_{3,7}$
$a_{0,3}$ $a_{0,4}$ $a_{0,5}$ $a_{0,6}$ $a_{0,7}$

Figure 2.2: When a rectangular sliding window function is used, the SWDFT takes a sequence of length $n \leq N$ DFTs. In this example, the length of each DFT, called this window size, is four. The first window position takes a length four DFT of the first points, the second window position takes a DFT of the second through fifth points, and this continues until the window position reaches the final four data points.

It can help to think about the 2D array of SWDFT coefficients as a multivariate time-series. Under this perspective, each series corresponds to the DFT coefficients at each window position for a particular frequency. For example, $a_{k,.}$ is the length P time-series corresponding to frequency $\frac{2\pi k}{n}$

$$a_{k,.} \quad = \quad [a_{k,n-1}, a_{k,n}, \ldots, a_{k,N-1}]. \tag{2.13}$$

Sometimes, we want the number of SWDFT window positions to equal the length of the input signal $\mathbf{x}$. An easy way to accomplish this is by inserting n - 1 zeros to the beginning of $\mathbf{x}$ before taking the SWDFT, which is called *zero padding*. Practitioners also zero pad signals for computational efficiency. Specifically, when practitioners compute the DFT with the fast Fourier transform algorithm, the signal length is often padded to be a power of two. In this thesis, and in the software that accompanies this thesis (see Appendix A), we always zero pad signals with n - 1 zeros. This is something to keep in mind when viewing the figures in this thesis.

2.2.1 The Multidimensional SWDFT

Like the DFT, the SWDFT easily extends to the multidimensional case. As an example, let $\mathbf{x}$ be an $N_0 \times N_1$ real-valued signal. Then the 2D SWDFT of $\mathbf{x}$ for length $n_0 \times n_1$ windows is given by

$$\begin{aligned}
a_{(k_0,k_1),(p_0,p_1)} &= \frac{1}{\sqrt{n_0 n_1}} \sum_{j_0=0}^{n_0-1} \sum_{j_1=0}^{n_1-1} x_{p_0-n_0+1+j_0,p_1-n_1+1+j_1} \omega_{n_0}^{-j_0 k_0} \omega_{n_1}^{-j_1 k_1}, \\
k_0 &= 0, 1, \ldots, n_0 - 1; k_1 = 0, 1, \ldots, n_1 - 1, \\
p_0 &= n_0 - 1, n_0, \ldots, N_0 - 1; p_1 = n_1 - 1, n_1, \ldots, N_1 - 1. \qquad (2.14)
\end{aligned}$$

In (2.14), (k_0, k_1) indexes the 2D frequency, and (p_0, p_1) indexes the 2D window position, where (p_0, p_1) is the bottom-right point of each window.

The 2D SWDFT outputs a 4D array, with dimensions $n_0 \times n_1 \times P_0 \times P_1$. There are $P_0 = N_0 - n_0 + 1$ window positions in the row-direction, and $P_1 = N_1 - n_1 + 1$ window positions in the column direction. The SWDFT extends to the arbitrary dimensions in the same way. For example, the d-dimensional SWDFT is given by

$$\begin{aligned}
a_{(k_0,k_1,\ldots k_{d-1}),(p_0,p_1,\ldots,p_{d-1})} &= \frac{1}{\sqrt{n_0 \cdot n_1 \cdot \ldots \cdot n_d}} \sum_{j_0=0}^{n_0-1} \sum_{j_1=0}^{n_1-1} \cdots \sum_{j_{d-1}=0}^{n_d-1} x_{(p_0-n_0+1+j_0),\ldots,(p_d-n_d+1+j_{d-1})} \omega_{n_0}^{-j_0 k_0} \ldots \omega_{n_{d-1}}^{-j_{d-1} k_{d-1}}, \\
k_0 &= 0, 1, \ldots, n_0 - 1, \ldots, k_{d-1} = 0, 1, \ldots, n_{d-1} - 1, \\
p_0 &= n_0 - 1, n_0, \ldots, N_0 - 1, \ldots, p_{d-1} = n_{d-1} - 1, n_{d-1}, \ldots, N_{d-1} - 1.
\end{aligned}$$

Chapter 3

SWDFT Algorithms

Ever since Cooley and Tukey (1965) introduced the fast Fourier transform (FFT) algorithm, computation has played a central role in Fourier analysis. Thus, a study of the SWDFT wouldn't be complete without understanding how to to compute it. In the next two chapters, we present a detailed analysis of algorithms to compute the SWDFT; Chapter 3 covers 1D SWDFT algorithms, and Chapter 4 covers the multidimensional case. We have two primary goals:

1. A detailed introduction to SWDFT algorithms.
2. To introduce my contribution, the Tree SWDFT algorithm (Richardson and Eddy (2019)).

As it turns out, fast SWDFT algorithms have been around for a long time, so we start by covering the history.

3.1 History of SWDFT Algorithms

Shortly after Gabor (1946) proposed the first time-frequency transform, researchers began discovering fast algorithms to compute the SWDFT. Broadly speaking, SWDFT algorithms can be grouped into two classes: *recursive* and *non-recursive*. Recursive algorithms use the Fourier shift theorem (see Appendix C.3) to update

SWDFT coefficients in consecutive window positions. Non-recursive algorithms compute a fast Fourier transform (FFT) in each window position, and remove repeated calculations in overlapping windows.

The literature on SWDFT algorithms is scattered. In fact, both the recursive and non-recursive algorithms were independently discovered at least four times. In response to the scattered literature, this section synthesizes all of the algorithms we could find. This synthesis helps us organize the large volume of previous work, and illustrates the major themes in a coherent narrative.

The first recursive algorithm was proposed by Halberstein (1966). Since then, many author's have independently proposed the same algorithm (e.g. Hostetter (1980), Bitmead (1982), Goldberg (1980), Amin (1987), Aravena (1990), Lilly (1991), and Bongiovanni et al. (1976)). Unser (1983) first pulled together the different recursive algorithms, and generalized the recursive relationship to compute *features* other than DFT coefficients. Unser (1983) also proposed extending the recursive algorithm to 2D in a section titled *Bidimensional recursion*.

After the initial surge of recursive algorithms, several researchers proposed improvements. Both Sherlock and Monro (1992) and Sherlock (1999) extended the recursive algorithm to 2D. These authors also showed how to incorporate non-rectangular windows, such as the Hamming and Blackman windows. Sorensen and Burrus (1988) extended the recursive algorithm to the *hopping DFT* case (see Section 3.3.1), which means that the window position shifts by more than one data-point. Park and Ko (2014) then proposed the *The Hopping DFT* algorithm, which improves on Sorensen and Burrus (1988). Lo and Lee (1999), Macias and Exposito (1998), Albrecht et al. (1997), and Albrecht and Cumming (1999) all proposed extensions and improvements to the recursive algorithm. Jacobsen and Lyons (2003) wrote the most cited article on the recursive algorithm, naming it *The Sliding DFT*, which popularized the algorithm to a wider audience.

The primary issue with recursive algorithms is numerical stability. The recursive algorithm is numerically unstable because complex exponential coefficients (ω_n) can only be represented with finite precision, and since computing new coefficients relies previously computed coefficients, numerical error accumulates as the number of window positions increases. In response, Douglas and Soh (1997) proposed the first numerically stable adaptation of the recursive algorithm. Since then, Duda (2010), Park (2015b), and Park (2017) have all proposed different numerically stable recursive algorithms, all of which sacrifice computational complexity. van der Byl and Inggs (2016) compares these algorithms in terms of speed and numerical accuracy.

A completely different numerically stable approach can be found in non-recursive algorithms. Although the paper is not well known in the literature, the first non-recursive algorithm dates back to Bongiovanni et al. (1975, 1976), who called it the *Triangular Fourier Transform.* Covell and Richardson (1991) proposed the next non-recursive algorithm, and proved that their algorithm is numerically stable. Farhang and Lim (1992) independently proposed the same non-recursive algorithm as Covell and Richardson (1991), which they called the *Sliding Fast Fourier Transform.* Farhang-Boroujeny and Gazor (1994) generalized this algorithm to the hopping DFT case, calling this the *Generalized SFFT*. Farhang-Boroujeny (1995) further extended the algorithm to a large class of orthogonal transforms, called *Generalized Transforms* (Elliott and Rao (1983)). This class includes the discrete cosine transform and the Walsh-Hadamard transform, among others. Exposito and Macias (1999, 2000) adapted the algorithm of Farhang and Lim (1992) to compute a single coefficient. In addition, these authors were the first to connect the binary tree structure of the fast Fourier transform (FFT) with the non-recursive algorithm, which is very important in this thesis. To reiterate, the binary tree structure of the FFT was already known (e.g Chapter 1 of Van Loan (1992)), but these authors were the first to connect this structure with the SWDFT. Montoya et al. (2012) and Montoya et al. (2014) further improved existing non-recursive algorithms, including an extension to the Radix-4 case. Wang et al. (2009) independently discovered the non-recursive algorithm while conducting an MEG experiment, and proposed a triangular prism data-structure to store calculations. Wang and Eddy (2010, 2012) gave parallel versions of the Wang et al. (2009) algorithm.

In recent years, several papers have extended SWDFT algorithms to 2D, although the idea dates back to (at least) Unser (1983)). Park (2015b) gave the first explicit 2D SWDFT algorithm, which extends the 1D recursive algorithm to 2D. Byun et al. (2016) proposed an algorithm based on the 2D Vector-Radix algorithm (Rivard (1977), Harris et al. (1977)), and Richardson and Eddy (2019) proposed the 2D Tree SWDFT algorithm, which generalizes to any dimension.

3.2 Baseline Algorithms

Before we delve into the fastest SWDFT algorithms, we define two algorithms used for comparison, which we call *baseline algorithms*. Baseline algorithms help us precisely quantify the gains due to faster algorithms. To start, recall the definition of the SWDFT with length n windows

$$\begin{aligned} a_{k,p} &= \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} x_{p-n+1+j} \omega_n^{-jk}, \\ k &= 0, 1, \ldots, n-1, \\ p &= n-1, n, \ldots, N-1. \end{aligned} \tag{3.1}$$

Following Cooley and Tukey (1965), we define an *operation* as a complex multiplication followed by a complex addition. Because there are $P = N - n + 1$ window positions, a straightforward calculation of (3.1) takes exactly Pn^2 operations. In this and the next chapter, we refer to this algorithm as the sliding window DFT (SWDFT). Since (3.1) takes a DFT in each window position, a simple improvement computes each DFT with a fast Fourier transform (FFT). We call this the sliding window fast Fourier transform (SWFFT), and this algorithm takes $O(Pn \log(n))$ operations. In comparison, both the recursive and non-recursive algorithms reduce the computational complexity to $O(Pn)$.

It is important to specify a few details regarding the FFT used in this thesis. First, while a variety of FFT algorithms exist, for example Rabiner et al. (1969) and Good (1958), we focus on the original Cooley-Tukey algorithm (Cooley and Tukey (1965)). So whenever we say FFT, we mean the Cooley-Tukey FFT. Second, different FFT algorithms exist for different sized signals, since the FFT works by diving a length N signal into smaller DFTs. The most common FFT algorithms are designed for signals with lengths that are powers of two (e.g. $N = 2^m$), which is called the *radix-2 case*. We focus on the radix-2 case in this thesis, but all the concepts extend to signals of any size.

3.3 Recursive Algorithms

Recursive algorithms use the Fourier shift theorem (derived in Appendix C.3) to establish a recurrence relation between SWDFT coefficients consecutive window positions. The recurrence relation is

$$a_{k,p} \quad = \quad \omega_n^k(a_{k,p-1} + x_p - x_{p-n}). \tag{3.2}$$

This equation implies that if we compute the SWDFT coefficients in the first window position, we can compute the remaining SWDFT coefficients with (3.2). The pseudocode for implementing the recursive algorithm is given in Algorithm 1.

Algorithm 1: The 1D Recursive SWDFT

```
input  : $x$: length $N$ signal, $n$: window size
- Compute the DFT coefficients in the first window position ($p = n - 1$)
for Window position $p$ from $n : (N - 1)$ do
    for Frequency $k$ from $0 : (n - 1)$ do
        - Compute the SWDFT coefficient $a_{k,p}$ using the recursive formula (3.2)
    end
end
output: $n \times P$ array of SWDFT coefficients
```

The calculation inside the two loops over window position and frequency is a complex addition, subtraction, and multiplication. Thus, Algorithm 1 takes $O(Pn)$ operations. In this thesis, we refer to this algorithm as the *recursive SWDFT*.

3.3.1 The Hopping SWDFT

A popular extension to the SWDFT is called *hopping DFT*. Whereas the SWDFT (3.1) computes a DFT in every window position ($p = n-1, n, n+1, \ldots$), the hopping SWDFT calculates a DFT once every L window positions. The recursive SWDFT easily extends to the hopping SWDFT case by adjusting the recurrence relation

$$a_{k,p} \quad = \quad \omega_n^{Lk}(a_{k,p-L} + \sum_{b=0}^{L-1}(x_{p-L+1+b} - x_{p-n-L+1+b})\omega_n^{-bk}). \tag{3.3}$$

3.4 Non-Recursive Algorithms

A different approach to computing the SWDFT comes from non-recursive algorithms. Non-recursive algorithms compute a fast Fourier transform (FFT) in each window position, and remove repeated calculations in window positions with overlapping input data. Since the FFT is so fundamental to non-recursive algorithms, we start by showing how the FFT works. Then, we show that visualizing the FFT algorithm with either a butterfly or tree diagram illustrates the repeated SWDFT calculations in overlapping windows. We leverage this insight in Section 3.4.3 to derive the non-recursive Tree SWDFT algorithm.

3.4.1 The Fast Fourier Transform (FFT)

A landmark moment in science and engineering occurred when Cooley and Tukey (1965) published the FFT algorithm. The FFT reduces the number of operations to compute a length N DFT from $O(N^2)$ to $O(N\log(N)$. This drastic speedup led the Society of Industrial and Applied Mathematics (SIAM) to name the FFT as a top-ten algorithm of the 20^{th} century (Cipra (2000)).

The main idea underlying the FFT is that we can compute larger DFTs by combining smaller DFTs. We demonstrate this idea by showing how a length $N = R \cdot C$ DFT can be computed with R length C DFTs, followed by C length R DFTs. To start, recall the DFT definition

$$\begin{aligned} a_k &= \frac{1}{\sqrt{N}}\sum_{j=0}^{N-1} x_j\omega_N^{-jk}, \\ k &= 0, 1, \ldots, N-1. \end{aligned} \tag{3.4}$$

We can change the subscripts with the following substitutions

$$\begin{aligned}
N &= R \cdot C, \\
k &= aR + b, \\
j &= sC + t \\
a, t &= 0, 1, \ldots, C-1, \\
b, s &= 0, 1, \ldots, R-1. \qquad (3.5)
\end{aligned}$$

Plugging (3.5 into (3.4) gives

$$a_{a,b} = \frac{1}{\sqrt{N}} \sum_{s=0}^{R-1} \sum_{t=0}^{C-1} x_{s,t} \omega_N^{-(sC+t)(aR+b)}. \qquad (3.6)$$

Since $(\omega_N^k)^N = 1 \ \forall k$, and $N = R \cdot C$, we can simplify the complex coefficient (ω_N) as

$$\begin{aligned}
\omega_N^{-(sC+t)(aR+b)} &= \omega_N^{-(sCaR+sCb+taR+tb)}, \\
&= (\omega_N^{-sc})^N \cdot \omega_N^{-(sCb+taR+tb)}, \\
&= \omega_N^{-(sCb+taR+tb)}. \qquad (3.7)
\end{aligned}$$

Plugging (3.7) back into (3.6)

$$a_{a,b} = \frac{1}{\sqrt{N}} \sum_{s=0}^{R-1} \sum_{t=0}^{C-1} x_{s,t} \omega_N^{-(sCb+taR+tb)}.$$

Next, note that $\omega_N^C = e^{\frac{i2\pi C}{N}} = e^{\frac{i2\pi}{R}} = \omega_R$, and in the same way $\omega_N^R = \omega_C$. Using this, and interchanging the order of the sums results in

$$a_{a,b} = \frac{1}{\sqrt{N}} \sum_{t=0}^{C-1} \omega_C^{-ta} \omega_N^{-bt} \sum_{s=0}^{R-1} x_{s,t} \omega_R^{-sb}. \tag{3.8}$$

The inner sum in (3.8) is a length R DFT, which we denote as

$$z_{b,t} = \sum_{s=0}^{R-1} x_{s,t} \omega_R^{-sb}. \tag{3.9}$$

Plugging (3.9) into (3.8) gives

$$a_{a,b} = \frac{1}{\sqrt{N}} \sum_{t=0}^{C-1} (z_{b,t} \omega_C^{-ta}) \omega_N^{-bt}. \tag{3.10}$$

The factor inside the parenthesis is a length C DFT, and the factor outside the parenthesis is called the *twiddle factor* (Gentleman and Sande (1966)). Twiddle factors are the complex coefficients (see Section B.4) used to combine smaller DFTs during the FFT algorithm.

In this derivation, the required computations are C length R DFTs, followed by R length C DFTs, which requires $CR^2 + RC^2 = N(R+C)$ total operations, compared with N^2 operations used by applying the DFT formula (3.4). The FFT algorithm applies this idea recursively, which leads to large speedups when N can be decomposed into a large number of factors (e.g. $N = N_1 \cdot N_2 \cdots N_m$).

3.4.2 Butterflies and Trees

Although the algebraic derivation of the FFT is important, visualizing the FFT clarifies the structure. Perhaps the most popular visualization of the FFT algorithm is the butterfly diagram, shown in Figure 3.1. The blue squares on the left hand side of Figure 3.1 are the input data, the red circles on the right hand side are the DFT coefficients, and the lines in the middle are the intermediate calculations. Both Farhang and

Lim (1992) and Covell and Richardson (1991) derived their non-recursive SWDFT algorithms by visualizing the butterfly diagram in consecutive window positions, and specifying which computations were duplicated.

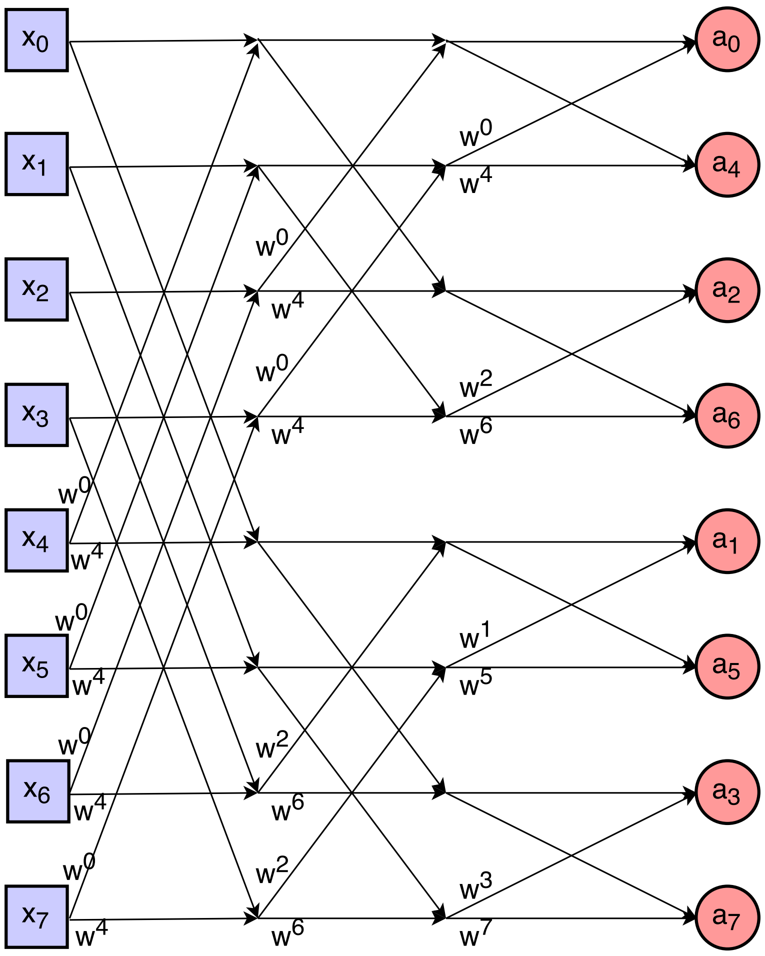

Figure 3.1: The butterfly diagram for a length $N = 8$ signal. The squares on the left hand side are the input data, and circles on the right hand side are the DFT coefficients. Multiplication occurs at the beginning of each arrow, and addition occurs at the end of each arrow. For example, the first calculation on the bottom left implies that x_7 is multiplied by ω_8^4, then added to x_3.

An alternative visualization to the butterfly diagram is the tree diagram, which is shown in Figure 3.2. We emphasize that the calculations in the tree diagram and the butterfly diagram are identical. In our view, the tree diagram is a better visualization for the SWDFT for two primary reasons. First, both the input data and output DFT coefficients are ordered, which is not true for the butterfly diagram. Second, each data-point has a binary tree with intermediate FFT calculations underneath it, and the DFT coefficients for each window position are at the bottom of the binary tree indexed by the corresponding data-point. We will see why this is important in the next section.

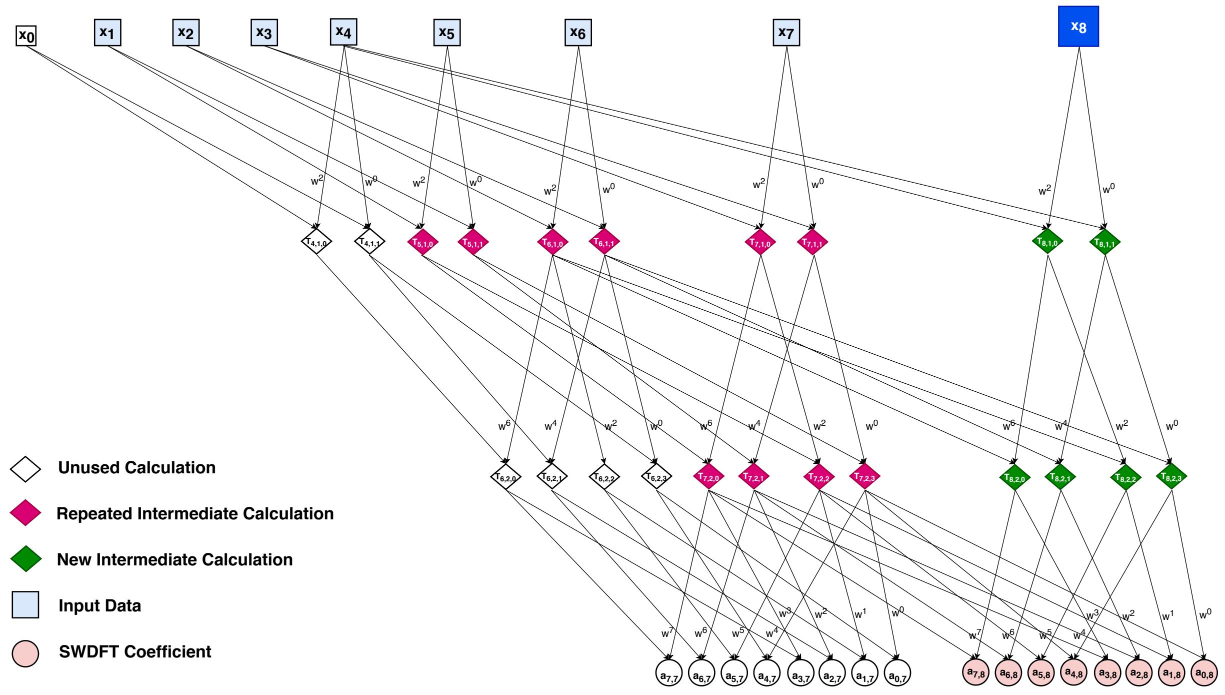

Figure 3.2: Calculations of the Tree SWDFT algorithm with window size $n = 8$ at window positions $p = 7$ and $p = 8$. The white circles at the bottom of the binary tree for window position $p = 7$ are the DFT coefficients corresponding to input data $[x_0, x_1, \ldots, x_7]$. Like the butterfly diagram (Figure 3.1), the arrows are the intermediate calculations, and the intermediate calculations are saved in a tree data-structure, which we indicate with the diamond shapes at the end of each intermediate calculation. For the next window position $p = 8$, the large square (x_8) enters the window, and the small square (x_0) exits the window, and the DFT coefficients are the red circles at the bottom of the right most binary tree. The pink diamonds in the tree data-structure are the repeated calculations, and the green diamonds are the new calculations to compute the SWDFT coefficients for this window position.

3.4.3 The Tree SWDFT Algorithm

While Figure 3.2 provides intuition, this section formally derives the 1D Tree SWDFT algorithm. We name our algorithm the Tree SWDFT because we crystallize the tree diagram into a formal data-structure. Our derivation explicitly shows how to use the tree data-structure to store and lookup calculations. We derive the Tree SWDFT algorithm for the radix-2 case, but the ideas generalize to arbitrary window sizes.

To set notation, let $\mathbf{x} = [x_0, x_1, \ldots, x_{N-1}]$ be a real or complex-valued signal, and let $n = 2^m \leq N$ be the window size, which implies that our binary trees have m levels. Let $\Omega = [\omega_n^0, \omega_n^{-1}, \ldots, \omega_n^{-(n-1)}]$ be a length n vector of twiddle factors, and let $\mathbf{T}$ be the tree structure that stores intermediate FFT calculations for each window position. We index $\mathbf{T}$ with $T_{p,l,i}$, which represents node i on level l of the binary tree at window position p. Finally, we need to specify which binary tree holds the repeated intermediate calculation required to compute the current node. For this task, let $s_l = 2^{m-l}$ be the *shift* at level l of $\mathbf{T}$. With this notation, the calculation at an arbitrary node of $\mathbf{T}$ is

$$T_{p,l,i} \quad = \quad T_{p-s_l,l-1,i \bmod 2^{l-1}} + (\Omega_{i \cdot s_l} \cdot T_{p,l-l,i \bmod 2^{l-1}}). \tag{3.11}$$

Implementation of the Tree SWDFT algorithm requires three nested loops:

1. Over N trees.
2. Over $m = \log_2(n)$ levels.
3. Over 2^l nodes at level l of the tree.

The only restriction on loop order is that loop 2 (over levels) must precede the loop 3 (over nodes), since the loop over nodes depends on calculations at previous levels of the tree. An important implementation detail is whether loop 2 (over levels) or loop 1 (over trees) is innermost. When loop 2 is innermost, we don't need to store the entire tree data-structure $\mathbf{T}$, since we only need the previous level of trees to compute the current level. When loop 1 comes first, the Tree SWDFT algorithm is an *online* algorithm, because if $\mathbf{T}$ is available for all previous window positions, the current window position can be computed in $O(n)$ time. An example implementation, with loop order 1, 2, 3, is given in Algorithm 2.

3.5 Comparisons

This chapter proposed four algorithms to compute the 1D SWDFT:

- **Sliding Window DFT**: Takes a DFT in each window position.

Algorithm 2: 1D Tree SWDFT Algorithm

```
input  : $x$: length $N$ signal, $n = 2^m$: window size
- Allocate the tree data-structure ($\mathbf{T}$)
- Create the twiddle factor vector ($\Omega$)
- Compute FFT in first window position, store in tree data-structure ($\mathbf{T}$)
for Trees ($p$ in $(n-1):(N-1)$) do
    for Levels ($l$ in $1:\log_2(n)$) do
        for Nodes ($i$ in $0:(2^l-1)$) do
            - Compute node $T_{p,l,i}$ using (3.11)
        end
    end
end
- Subset the final level of each binary tree with the DFT coefficients
output: $n \times P$ array of SWDFT coefficients
```

- **Sliding Window FFT**: Takes an FFT for each window position.
- **Recursive SWDFT**: Recursively updates SWDFT coefficients with the Fourier shift theorem.
- **Tree SWDFT**: Reuses repeated calculations with a tree data-structure.

We'll compare these algorithms across four criteria:

1. Speed
2. Memory
3. Stability
4. Window Size Flexibility

The comparisons are summarized in Table 3.1. The recursive and Tree SWDFT algorithms are the fastest, with a computational complexity of $O(Pn)$. The Tree SWDFT uses more memory than the recursive algorithm, since it stores intermediate calculations in a tree data-structure. The precise amount of memory depends on the loop order, and Table 3.1 assumes that loop 2 (over levels) is innermost.

The only algorithm that isn't numerically stable is the recursive SWDFT. Recursive algorithms are unstable because the complex exponentials (e.g. ω_n) are represented with finite precision, and numerical error accumulates across window positions, which implies that numerical error is unbounded as the number

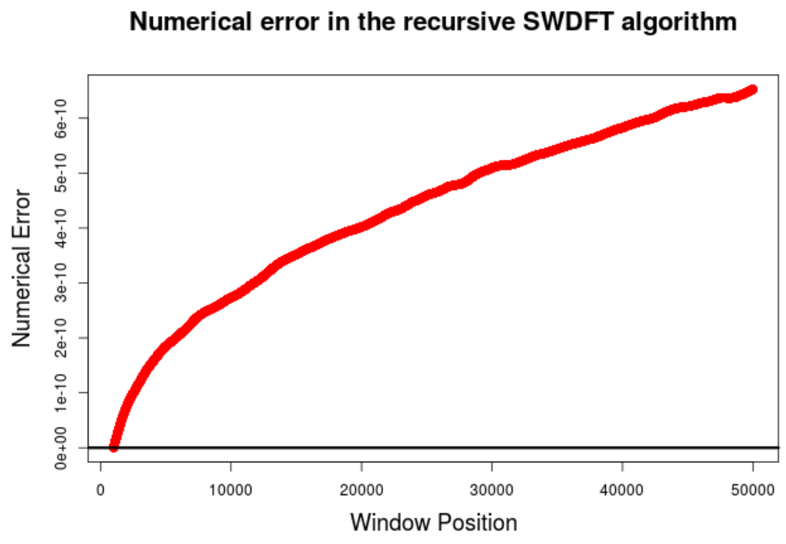

Figure 3.3: The average numerical error for each window position when the SWDFT is computed with the recursive algorithm. This shows that numerical error grows as the number of window positions increases.

of window positions increases. A demonstration of this is shown in Figure 3.3. To be fair, it is possible to derive numerically stable recursive algorithms, and Section 3.1 lists several such approaches.

The final criteria is based on whether the algorithm works for any window size. All algorithms except the Tree SWDFT work here, because our derivation was tailored to the radix-2 case. However, since the Tree SWDFT builds on the Cooley-Tukey FFT, and the Cooley-Tukey FFT works for arbitrary window sizes, the Tree SWDFT can clearly be extended to work for arbitrary window sizes. The downside is that these extensions require extra programming effort (e.g. Singleton (1966, 1969)).

Algorithm	Speed	Memory	Stability	Window Size
Sliding Window DFT	Pn^2	Pn	Yes	Yes
Sliding Window FFT	$O(Pn\log(n))$	Pn	Yes	Yes
Recursive SWDFT	$O(Pn)$	Pn	Stable adaptations available	Yes
Tree SWDFT	$O(Pn)$	$2Nn$	Yes	Possible

Table 3.1: Comparison of the four 1D SWDFT algorithms proposed in Chapter 3.

Chapter 4

Multidimensional SWDFT Algorithms

The rise of image and video processing has lead to an explosion of multidimensional signals (e.g. Roth et al. (2015), Tai and Eddy (2018), Mathieu et al. (2013), Chen (1998), Yao et al. (2019)). Therefore, it's important that algorithms work not just in 1D, but in 2D, 3D, and beyond. In response, this chapter covers algorithms for the multidimensional SWDFT. After describing existing algorithms, we give a detailed derivation of the multidimensional Tree SWDFT algorithm. This derivation is more general than the derivation given in our earlier paper on the 2D Tree SWDFT (Richardson and Eddy (2019)).

4.1 2D Baseline Algorithms

Like the 1D case, we briefly describe the *baseline algorithms* used for comparison. Let $\mathbf{x}$ be an $N_0 \times N_1$ real or complex valued signal, and recall that the 2D SWDFT for length $n_0 \times n_1$ windows is

$$\begin{aligned} a_{(k_0,k_1),(p_0,p_1)} &= \frac{1}{\sqrt{n_0 n_1}} \sum_{j_0=0}^{n_0-1} \sum_{j_1=0}^{n_1-1} x_{p_0-n_0+1+j_0, p_1-n_1+1+j_1} \omega_{n_0}^{-j_0 k_0} \omega_{n_1}^{-j_1 k_1}, \\ k_0 &= 0, 1, \ldots, n_0 - 1, \\ k_1 &= 0, 1, \ldots, n_1 - 1, \end{aligned}$$

$$\begin{aligned} p_0 &= n_0 - 1, n_0, \ldots, N_0 - 1, \\ p_1 &= n_1 - 1, n_1, \ldots, N_1 - 1. \end{aligned} \tag{4.1}$$

A straightforward calculation of (4.1) takes a 2D DFT for each window position, which requires exactly $P_0P_1n_0^2n_1^2$ operations. Like the 1D case, we can replace 2D DFTs with 2D FFTs (see Section 9 of Duhamel and Vetterli (1990)), which reduces the computational complexity to $O(P_0P_1n_0n_1\log(n_0n_1))$. The remainder of this chapter covers algorithms with computational complexity $O(P_0P_1n_0n_1)$.

4.2 2D Recursive Algorithms

Both Sherlock (1999) and Park (2015a) extended the 1D recursive algorithm to 2D. Like the 1D recursive algorithm, the 2D recursive SWDFT uses a 2D version of the Fourier shift theorem. To be explicit, we derive the 2D recursive algorithm in this section.

To be concise, let $\hat{p}_0 = p_0 - n_0 + 1$ and let $\hat{p}_1 = p_1 - n_1 + 1$. We will derive the recursive algorithm for a shift in the horizontal direction. Let's start by re-writing the 2D SWDFT

$$\begin{aligned} a_{(k_0,k_1),(p_0,p_1)} &= \sum_{j_0=0}^{n_0-1}\sum_{j_1=0}^{n_1-1} x_{\hat{p}_0+j_0,\hat{p}_1+j_1}\omega_{n_0}^{-j_0k_0}\omega_{n_1}^{-j_1k_1} \\ &= \sum_{j_0=0}^{n_0-1}\sum_{j_1=0}^{n_1-1} x_{\hat{p}_0+j_0-1,\hat{p}_1+j_1}\omega_{n_0}^{-(j_0-1)k_0}\omega_{n_1}^{-j_1k_1} \\ &+ \sum_{j_1=0}^{n_1-1} x_{p_0,\hat{p}_1+j_1}\omega_{n_0}^{-(n_0-1)k_0}\omega_{n_1}^{-j_1k_0} - \sum_{j_1=0}^{n_1-1} x_{p_0-n_0,\hat{p}_1+j_1}\omega_{n_0}^{k_0}\omega_{n_1}^{-j_1k_1} \\ &= \omega_{n_0}^{k_0}(a_{(k_0,k_1),(p_0-1,p_1)} + \sum_{j_1=0}^{n_1-1} x_{p_0,\hat{p}_1+j_1}\omega_{n_1}^{-j_1k_1} - \sum_{j_1=0}^{n_1-1} x_{p_0-n_0,\hat{p}_1+j_1}\omega_{n_1}^{-j_1k_1}). \end{aligned} \tag{4.2}$$

Now define z_{p_0,p_1} as the following 1D DFT

$$z_{k_1,(p_0,p_1)} = \sum_{j_1=0}^{n_1-1} x_{p_0,\hat{p}_1+j_1}\omega_{n_1}^{-j_1k_1}. \tag{4.3}$$

Plugging (4.3) into (4.2) gives the recurrence relation

$$a_{(k_0,k_1),(p_0,p_1)} = \omega_{n_0}^{k_0}(a_{(k_0,k_1),(p_0-1,p_1)} + z_{k_1,(p_0,p_1)} - z_{p_0-n_0,p_1}). \tag{4.4}$$

And that's it. The main difference between the 1D and 2D recursive algorithm is that the 2D algorithm requires 1D DFTs instead of a single data point. This makes implementation more complicated.

4.3 2D Non-Recursive Algorithms

Like the 1D case, multidimensional non-recursive SWDFT algorithms provide a numerically stable alternative to recursive algorithms. Also like the 1D case, the important idea is reusing FFT calculations in overlapping windows. Thus, we need a precise understanding of how the multidimensional FFT works, which we cover in Section 4.3.1. After we understand multidimensional FFTs, Sections 4.3.2 and 4.3.3 cover the two different 2D non-recursive SWDFT algorithms proposed in the literature, and we show that the fundamental distinction between the two algorithms is a different type of 2D FFT.

4.3.1 The Multidimensional FFT

According to Duhamel and Vetterli (1990), there are four classes of multidimensional FFT algorithms:

1. Row-column
2. Vector-radix
3. Nested algorithms

4. Polynomial transforms

Of these four, we focus on the row-column and vector-radix algorithms, since they both generalize the 1D Cooley-Tukey FFT (Mersereau and Speake (1981)). The row-column FFT exploits the fact that the 2D DFT is separable. Separability implies that a 2D DFT can be computed by taking 1D FFTs of each row, followed by 1D FFTs of each resulting column. In contrast, the vector-radix algorithm decomposes both indices simultaneously. For example, the 2D 2×2 vector-radix FFT combines 2×2 DFTs into larger 2D DFTs. We explicitly derive both of these concepts below.

4.3.2 Vector-Radix 2×2 Sliding FFT

The first non-recursive 2D SWDFT algorithm proposed in the literature was Byun et al. (2016), who called it the *Vector-Radix* 2×2 *Sliding FFT* (VR-SFFT). This algorithm uses a multidimensional FFT algorithm called the *vector-radix FFT* in each window position, and removes repeated calculations in overlapping window positions.

The main difference between the VR-SFFT and our 2D Tree SWDFT algorithm is the type of 2D FFT; the VR-SFFT uses a 2D vector-radix FFT in each window position, and the 2D Tree SWDFT uses a 2D row-column FFT in each window position. We briefly derive the vector-radix algorithm for arbitrary radices here, following the original derivation in Harris et al. (1977). Recall the 2D DFT

$$a_{k_0,k_1} \quad = \quad \sum_{j_0=0}^{N_0-1} \sum_{j_1=0}^{N_1-1} x_{j_0,j_1} \omega_{N_0}^{-j_0 k_0} \omega_{N_1}^{j_1 k_1}. \tag{4.5}$$

We start by re-writing both indices

$$N_0 = r_0 P_0 \qquad N_1 = r_1 P_1. \tag{4.6}$$

Next, replace the indices in x with the following change of variables

$$j_0 = r_0 \cdot p_0 + u_0 \qquad j_1 = r_1 \cdot p_1 + u_1,$$
$$p_0 = 0, \dots, P_0 - 1 \qquad p_1 = 0, \dots, P_1 - 1,$$
$$u_0 = 0, \dots, r_0 - 1 \qquad u_1 = 0, \dots, r_1 - 1. \tag{4.7}$$

Then plug (4.7) into (4.5)

$$a_{k_0,k_1} = \sum_{p_0=0}^{P_0-1} \sum_{u_0=0}^{r_0-1} \sum_{p_1=0}^{P_1-1} \sum_{u_1=0}^{r_1-1} x_{r_0p_0+u_0, r_1p_1+u_1} \omega_{N_0}^{-(r_0p_0+u_0)k_0} \omega_{N_1}^{-(r_1p_1+u_1)k_1}. \tag{4.8}$$

Re-write the complex exponentials as

$$\omega_{N_0}^{-(r_0p_0+u_0)k_0} = \omega_{P_0}^{-p_0k_0} \omega_{N_0}^{-u_0k_0} \quad = \quad \omega_{N_1}^{-(r_1p_1+u_1)k_1} = \omega_{P_1}^{-p_1k_1} \omega_{N_1}^{-u_1k_1}. \tag{4.9}$$

And similar to the row-column FFT, we can break this into a smaller 2D DFTs, with $r_0 \times r_1$ total $P_0 \times P_1$ DFTs

$$a_{k_0,k_1} = \sum_{u_0=0}^{r_0-1} \sum_{u_1=0}^{r_1-1} \omega_{N_0}^{-u_0k_0} \omega_{N_1}^{-u_1k_1} z_{k_0,k_1}(u_0, u_1), \tag{4.10}$$

where

$$z_{k_0,k_1}(u_0, u_1) = \sum_{p_0=0}^{P_0-1} \sum_{p_1=0}^{P_1-1} x_{r_0p_0+u_0, r_1p_1+u_1} \omega_{P_0}^{-p_0k_0} \omega_{P_1}^{-p_1k_1}. \tag{4.11}$$

The Harris et al. (1977) method makes one more change of variables

$$\begin{aligned} k_0 &= a_0 + b_0 \cdot P_0 \qquad & k_1 &= a_1 + b_1 \cdot P_1, \\ a_0 &= 0, \ldots, P_0 - 1 \qquad & a_1 &= 0, \ldots, P_1 - 1, \\ b_0 &= 0, \ldots, r_0 - 1 \qquad & a_1 &= 0, \ldots, r_1 - 1, \end{aligned} \tag{4.12}$$

which gives

$$a_{a_0+b_0P_0,a_1+b_1P_1} = \sum_{u_0=0}^{r_0-1} \sum_{u_1=0}^{r_1-1} \omega_{N_0}^{-u_0(a_0+b_0 \cdot P_0)} \omega_{N_1}^{-u_1(a_1+b_1 \cdot P_1)} z_{a_0+b_0P_0,a_1+b_1P_1}(u_0, u_1). \tag{4.13}$$

And since the complex exponential is periodic, we can further simplify

$$z_{a_0+b_0P_0,a_1+b_1P_1}(u_0, u_1) = z_{a_0,a_1}(u_0, u_1). \tag{4.14}$$

Putting everything together gives

$$a_{a_0+b_0P_0,a_1+b_1P_1} = \sum_{u_0=0}^{r_0-1} \sum_{u_1=0}^{r_1-1} \omega_{r_0}^{-u_0b_0} \omega_{r_1}^{-u_1b_1} [\omega_{N_0}^{-u_0a_0} \omega_{N_1}^{-u_1a_1} z_{a_0,a_1}(u_0, u_1)]. \tag{4.15}$$

(4.15) is the $r_0 \times r_1$ butterfly used in the vector-radix $r_0 \times r_1$ derivation. This derivation can be applied recursively.

Why use the vector-radix FFT instead of the row-column FFT? The answer is that it requires fewer complex multiplications, explained by Harris et al. (1977):

One very important observation is that the product of the twiddle factors $\omega_{N_0}^{-u_0 a_0}\omega_{N_1}^{-u_1 a_1}$" may be precomputed and stored in a table. The computational advantage enjoyed by the vector-radix algorithm arises from the fact that such products are not computed, whereas they are computed implicitly in the row-column approach.

That said, the downside of Byun et al. (2016) is that no explicit implementation was provided. While Figure 6 of Byun et al. (2016) points out which butterflies are duplicated in overlapping windows, the authors do not provide either an example implementation or an explicit data-structure to store/look up repeated calculations. In contrast, (Richardson and Eddy (2019)) instantiates the 2D Tree SWDFT algorithm with a C program, derives an explicit data structure, and details how the data-structure leads to the exact calculations.

4.3.3 The 2D Tree SWDFT Algorithm

The 2D Tree SWDFT computes a 2D FFT in each window position, and avoids repeating calculations by storing intermediate calculations in a tree data-structure. We formally derive the 2D Tree SWDFT in this section, under the following assumptions:

- We use the 2D row-column FFT in each window position.
- Window sizes are powers of two: $n_0 = 2^{m_0}$, $n_1 = 2^{m_1}$.

However, the ideas underlying the Tree SWDFT extend to arbitrary window sizes *and* different types of 2D FFTs. For example, the vector-radix sliding FFT derived in the previous section fits into our framework.

Let's start by briefly deriving a 2D row-column FFT. The important idea is that the 2D DFT is *separable*, which implies that a 2D DFT can be computed with 1D DFTs. For example

$$a_{(k_0,k_1)} = \sum_{j_0=0}^{n_0-1}\sum_{j_1=0}^{n_1-1} x_{j_0,j_1}\omega_{n_0}^{-j_0 k_0}\omega_{n_1}^{-j_1 k_1},$$

$$= \sum_{j_0=0}^{n_0-1} z_{j_0,k_1} \omega_{n_0}^{-j_0 k_0}, \tag{4.16}$$

where

$$z_{j_0,k_1} = \sum_{j_1=0}^{n_1-1} x_{j_0,j_1} \omega_{n_1}^{-j_1 k_1}. \tag{4.17}$$

Thus, the 2D row-column FFT first computes the 1D DFTs of each row (z_{j_0,k_1}), then computes the 1D DFTs of the corresponding columns ($a_{(k_0,k_1)}$), which is why it's called the *row-column* FFT. Since the 2D row-column FFT is composed of 1D DFTs, we can compute each 1D DFT with the FFT algorithm, which makes implementation trivial.

An illustration of the 2D Tree SWDFT algorithm is given in Figure 4.1. This figure shows the calculations for the second window position, after the calculations from the first window position have been completed. The second window position slides the first window position one column to the right; the large dark blue squares in column four are the data points entering the window, and the small white squares in column zero are the data points exiting the window. There's a 2D binary tree under each data-point, and the nodes in each binary tree are elements of the tree data-structure, which are represented as diamonds in Figure 4.1. The pink diamonds are the nodes already computed in window position one, and the green diamonds are the nodes that need to be computed for window position two. The important idea is that, by saving the nodes from the first window position, we don't need to recompute these nodes in the second window position, which is why the 2D Tree SWDFT is fast.

Let's set notation for the 2D Tree SWDFT. Let $\mathbf{x}$ be an $N_0 \times N_1$ array. Our 2D binary trees have $m_0 + m_1$ levels, since the 2D row-column FFT requires $m_0 + m_1$ stages. The first m_0 levels correspond to row FFTs, and the remaining m_1 levels correspond to column FFTs. Let $\Omega_i = [\omega_{n_i}^0, \omega_{n_i}^{-1}, \ldots, \omega_{n_i}^{-(n_i-1)}]$ be the length n_i twiddle factor vector for dimension $i; i = 0, 1$. Let $\mathbf{T}$ be the tree data-structure used to store the 2D FFT calculations for each window position. We access $\mathbf{T}$ by window position, level, and node.

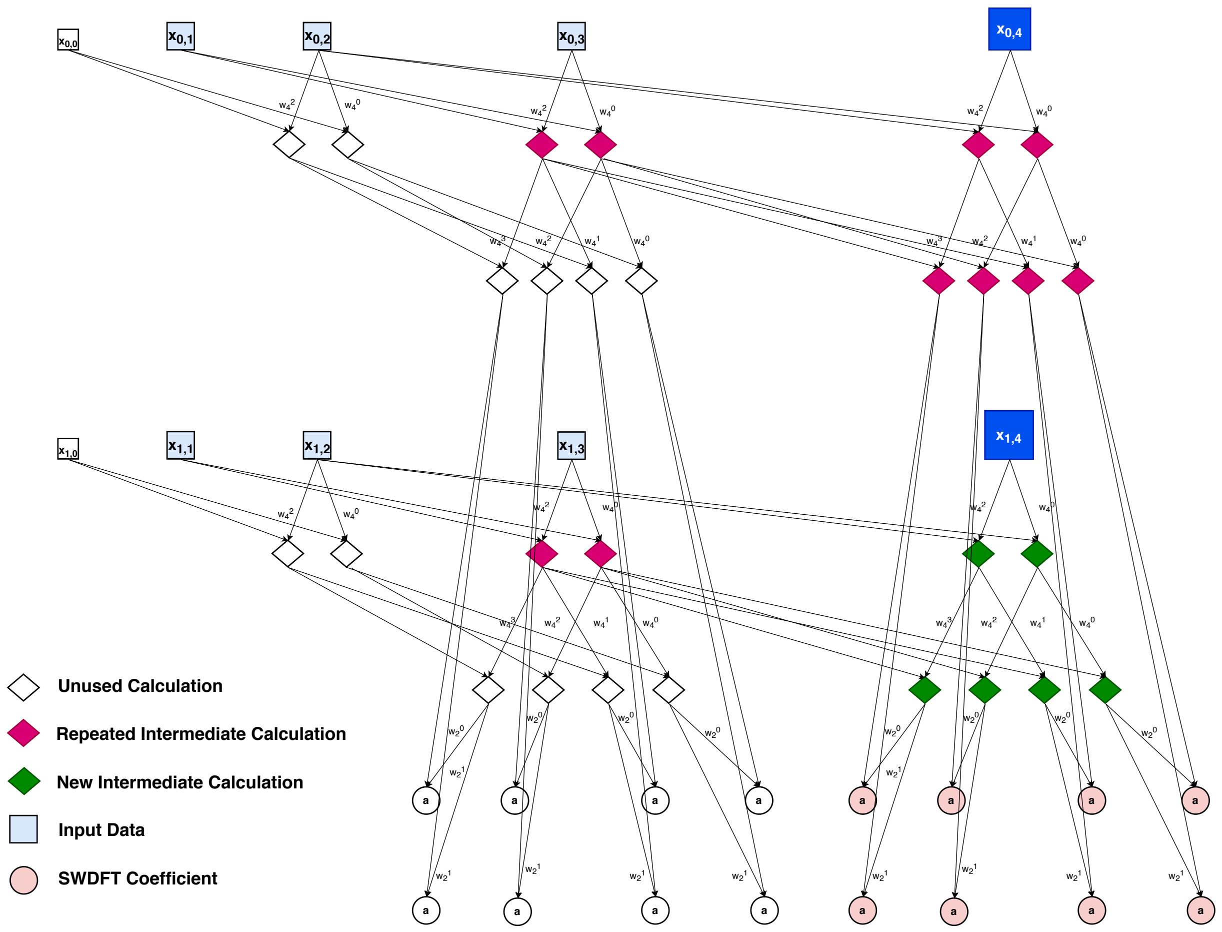

Figure 4.1: The 2D Tree SWDFT for a 2×5 array in the in two window positions: $(p_0, p_1) = (1, 3), (1, 4)$. For the first window position $(1, 3)$, the white circles underneath the 2D binary tree are the 2D DFT coefficients, and for the second window position $(1, 4)$, the red circles underneath the binary tree are the 2D DFT coefficients. In the second window position, the two large blue squares are the new data points entering the window, and the two small white squares are the data points exiting the window. The diamonds are nodes in the tree data-structure; the white diamonds are only used in the first window position, the pink diamonds are used in both window positions, and the green diamonds are only used in the second window position.

For example, $T_{(p_0,p_1),(l_0,l_1),(i_0,i_1)}$ corresponds to node (i_0, i_1) on level $l_0 + l_1$ of the tree located at window position (p_0, p_1). Level 0 of $\mathbf{T}$ is the input data $(\mathbf{x})$. Finally, we need to know which binary tree contains the repeated calculation required to calculate the current node. Let $s_d(l_0, l_1); d = 0, 1$ be the *shift function* for dimension d at level $l_0 + l_1$. The shift function determines how many 2D binary trees across dimension d the repeated calculation is. Specifically, define the shift function as

$$s_0(l_0, l_1) = \begin{cases} 0, & l_0 + l_1 \in [1, m_1] \\ 2^{m_0 - l_0}, & l_0 + l_1 \in [m_1 + 1, m_1 + m_0]. \end{cases}$$

$$s_1(l_0, l_1) = \begin{cases} 2^{m_1 - l_1}, & l_0 + l_1 \in [1, m_1] \\ 0, & l_0 + l_1 \in [m_1 + 1, m_1 + m_0]. \end{cases}$$

The 2D Tree SWDFT works by computing each node of $\mathbf{T}$. The structure of each calculation has the following form

$$T_A = T_B + \Omega_C \cdot T_D, \tag{4.18}$$

where A, B, and D are indices into $\mathbf{T}$, and C indexes Ω_i. This implies that computing each node of $\mathbf{T}$ requires one complex multiplication and one complex addition, which is defined as an operation.

The rest of this section derives the exact indices for (4.18) at an arbitrary window position (p_0, p_1). First, we define several *helper* functions, which will also help us extend the algorithm to k-dimensions. Let $g_{l_d}(l_0, l_1); d = 0, 1$ be a function that determines the level of the tree:

$$g_{l_0}(l_0, l_1) = \begin{cases} 0, & l_0 + l_1 \in [1, m_1] \\ l_0 - 1, & l_0 + l_1 \in [m_1 + 1, m_1 + m_0]. \end{cases}$$

$$g_{l_1}(l_0, l_1) = \begin{cases} l_1 - 1, & l_0 + l_1 \in [1, m_1] \\ m_1, & l_0 + l_1 \in [m_1 + 1, m_1 + m_0]. \end{cases}$$

Similarly, let $h_{i_d}(l_0, l_1); d = 0, 1$ be a function that determines the node index for each dimension:

$$h_{i_0}(l_0,l_1) = \begin{cases} 0, & l_0+l_1 \in [1, m_1] \\ \\ i_0 \bmod 2^{l_0-1}, & l_0+l_1 \in [m_1+1, m_1+m_0] \end{cases}$$

$$h_{i_1}(l_0,l_1) = \begin{cases} i_1 \bmod 2^{l_1-1}, & l_0+l_1 \in [1, m_1] \\ \\ i_1, & l_0+l_1 \in [m_1+1, m_1+m_0]. \end{cases}$$

Finally, let f_d be our twiddle-factor indexing function:

$$f_d(l_0,l_1,i_0,i_1) = \begin{cases} \Omega_1(i_1 \cdot s_1(l_0,l_1)), & l_0+l_1 \in [1, m_1] \\ \\ \Omega_0(i_0 \cdot s_0(l_0,l_1)), & l_0+l_1 \in [m_1+1, m_1+m_0]. \end{cases}$$

To be concise, we write $g_{l_d}(l_0,l_1) = g_{l_d}$, and similarly for s_d, f_d and h_{i_d}. With this notation, the calculation for each node in $\mathbf{T}$ is

$$T_{(p_0,p_1),(l_0,l_1),(i_0,i_1)} \quad = \quad T_{(p_0-s_0,p_1-s_1),(g_{l_0},g_{l_1}),(h_{i_0},h_{i_1})} + f_d \cdot T_{(p_0,p_1),(g_{l_0},g_{l_1}),(h_{i_0},h_{i_1})}. \tag{4.19}$$

For example, the calculation for levels $l_0+l_1 \in [1, m_1]$, which corresponds to 1D row FFTs, is

$$T_{(p_0,p_1),(0,l_1),(0,i_1)} \quad = \quad T_{(p_0,p_1-2^{m_1-l_1}),(0,l_1-1),(0,i_1 \bmod 2^{l_1-1})} + \Omega_1(i_1 \cdot 2^{l_1-1}) \cdot T_{(p_0,p_1),(0,l_1-1),(0,i_1 \bmod 2^{l_1-1}))}. \tag{4.20}$$

The calculation for levels $l_0+l_1 \in [m_1+1, m_1+m_0]$, which corresponds to 1D column FFTs, is

$$T_{(p_0,p_1),(l_0,l_1),(i_0,i_1)} \quad = \quad T_{(p_0-,p_1),(l_0-1,m_1),(i_0 \bmod 2^{l_0-1},i_1)} + \Omega_0(i_0 \cdot 2^{l_0-1}) \cdot T_{(p_0,p_1),(l_0-1,l_1),(i_0 \cdot 2^{l_0-1},i_1)}.$$

(4.21)

The 2D Tree SWDFT calculates each node of $\mathbf{T}$ using (4.19), and the 2D SWDFT coefficients are located at level m_0+m_1 of the binary trees. All that remains is sub-setting the coefficients from $\mathbf{T}$, and the algorithm is complete.

Implementation

The 2D Tree SWDFT can be implemented with six nested loops, over:

1. $m_0 = \log_2(n_0)$ levels (row FFTs)
2. $m_1 = \log_2(n_1)$ levels (column FFTs)
3. N_0 trees in row-direction
4. N_1 trees in column-direction
5. 2^{l_0} nodes in row-direction
6. 2^{l_1} nodes in column-direction

Inside the six nested loops is (4.19).

Like the 1D case, we can rearrange the loop order, subject to a few restrictions. The first is that the loop over l_0 must precede the loop over l_1, since row FFTs are calculated before column FFTs. The second is that the loops 5 and 6 (over nodes) must come after loops 1 and 2 (over levels), since the calculation at each node depends on calculations at higher levels of $\mathbf{T}$. From an algorithmic perspective, the main distinction is whether the loop over trees, or the loop over levels, is innermost.

When the loops over levels are innermost, there are two primary advantages. First, since each node only depends on nodes on the previous level of $\mathbf{T}$, we don't need to allocate the entire tree data-structure $\mathbf{T}$ in memory. Instead, we can allocate $2N_0N_1n_0n_1$ complex numbers: half for the previous level, and half for the current level of the tree. The second advantage is that the algorithm is suitable for parallel computing, following Wang and Eddy (2012).

When the loops over trees are innermost, the algorithm does not require that all the data be available at the outset, which makes the 2D Tree SWDFT an *online* algorithm that can be tailored to real-time tasks. For example, when a new data point enters the window, and we have computed the nodes at all previous window positions, we can compute the next 2D DFT coefficients in $O(n_0 n_1)$ time.

Finally, a key consideration when moving from 1D to 2D is memory. The output of a 2D SWDFT is a 4D ($n_0 \times n_1 \times P_0 \times P_1$) array. If we use 16-byte complex numbers, $P_0 = P_1 = 400$, and $n_0 = n_1 = 32$, then storing the output alone takes $16 \cdot 163840000 = 2621440000$ bytes, or approximate 2.6 GB.

Computational Complexity and Comparisons

Recall that we defined an operation as one complex multiplication and one complex addition., which implies that calculating each node of $\mathbf{T}$ requires a single operation. To calculate the overall computational complexity, let $l = l_0 + l_1$ be the level of the tree. Since each level of the 2D binary trees has 2^l total nodes, the number of operations for each window position, denoted C_{window}, is

$$\begin{aligned} C_{\text{window}} &= \sum_{l=1}^{m_1+m_0} 2^l, \\ &= (2 + 4 + \ldots + 2^{m_1+m_0}), \\ &= 2(2^{m_1+m_0} - 1), \\ &= 2(n_0 n_1 - 1), \end{aligned} \tag{4.22}$$

which implies the number of operations grows linearly ($O(n_0 n_1)$) with window size. The 2D Tree SWDFT also grows linearly with window position, so the overall computational complexity is $O(P_0 P_1 n_0 n_1)$. In terms of memory, Section 4.3.3 showed that when the loops over levels are innermost, the algorithm requires $2N_0 N_1 n_0 n_1$ complex numbers of memory, When the loops over trees are innermost, we need to store the entire tree data structure ($\mathbf{T}$), which requires slightly less than one binary tree per data-point. In either case, the required memory is bounded by

$$\begin{aligned} C_{\mathbf{T}} &\leq N_0 \cdot N_1 \cdot C_{window}, \\ &= N_0 N_1 2(n_0 n_1 - 1), \end{aligned}$$

and the memory complexity is $O(N_0 N_1 n_0 n_1)$.

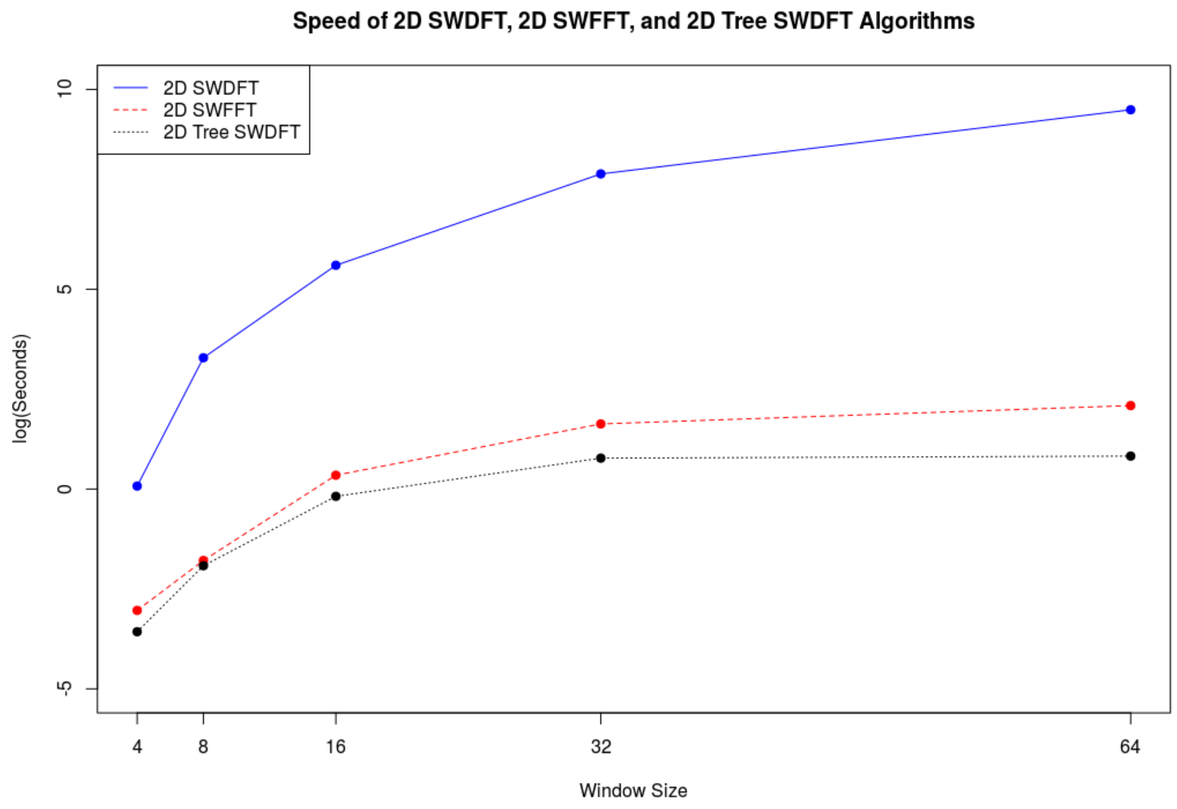

Figure 4.2: A comparison of the Tree SWDFT algorithm with the baseline algorithms, reproduced from Richardson and Eddy (2019). Each algorithm was programmed in the C language, and uses the same computer, so this figure isolates differences in computational complexity. We see that going from the SWDFT to the SWFFT provides substantial gains, and going from the SWFFT to the Tree SWDFT also provides a clear speedup.

Figure 4.2 demonstrates the computational gains from the 2D Tree SWDFT algorithm compared with the baseline algorithms. The takeaway from this figure is that the computational gains from the Tree SWDFT aren't as large as the computational gains from the fast Fourier transform (FFT), but it still represents a sizable improvement.

Table 4.1 compares the 2D SWDFT algorithms we proposed in this chapter. Out of all algorithms with $O(P_0 P_1 n_0 n_1)$ speed, the 2D Tree SWDFT is the only one that is numerically stable, works for non-square window sizes, and has a publicly available software implementation. As we will see in the next section, the 2D Tree SWDFT also generalizes to arbitrary dimensions.

Algorithm	**Speed**	**Memory**	**Numerically Stable**	**Non-Square Windows**	**Software**
2D SWDFT	$O(P_0 P_1 n_0^2 n_1^2)$	$P_0 P_1 n_0 n_1$	Yes	Yes	Yes
2D SWFFT	$O(P_0 P_1 n_1 n_0 \log(n_0 n_1))$	$P_0 P_1 n_0 n_1$	Yes	Yes	Yes
Park (2015a)	$O(P_0 P_1 n_0 n_1)$	$O(N_0 N_1 n_0 n_1)$	No	Yes	No
Byun et al. (2016)	$O(P_0 P_1 n_0 n_1)$	$O(N_0 N_1 n_0 n_1)$	Yes	No	No
Richardson and Eddy (2019)	$O(P_0 P_1 n_0 n_1)$	$O(N_0 N_1 n_0 n_1)$	Yes	Yes	Yes

Table 4.1: Speed, memory, and properties of existing 2D SWDFT algorithms. The 2D SWDFT and SWFFTs take a 2D DFT or 2D FFT in each window. Park (2015a) extends the 1D recursive algorithm to 2D (Section 3.3, and Byun et al. (2016) gives a vector-radix 2×2 Sliding FFT algorithm described in Section 4.3.2. The three properties we consider are: 1.) Numerical stability 2.) Non-square windows 3.) Software implementation. We see that the recursive algorithm of Park (2015a) is numerically unstable, and the vector-radix algorithm of Byun et al. (2016) does not work for non-square windows. In addition, neither of these algorithms has a publicly available software implementation. The 2D Tree SWDFT improves on all of these properties.

4.4 The kD Tree SWDFT Algorithm

In our view, the most important property in extending the 1D Tree SWDFT the 2D was that the 2D DFT is separable. For higher dimensions ($k > 2$), the separability property still holds, and this means we can compute a kD DFT using 1D FFTs. Thus, we can use the separability property to extend the Tree SWDFT algorithm to arbitrary dimensions.

In kD, our tree data-structure $\mathbf{T}$ now stores k-dimensional binary trees underneath each data-point. Like the 2D case, we can derive the kD algorithm by defining *helper* functions that help us compute each node of $\mathbf{T}$. To be concise, let $l = \sum_{i=0}^{k-1} l_i$ be the level of the tree. Our kD derivation uses four indexing functions:

1. $s_d(l_0, \ldots, l_{k-1}); d = 0, \ldots, k-1$: Shift across dimension d at level l that gives the window position of the repeated calculation.

2. $g_{i_d}(l_0, \ldots, l_{k-1}); d = 0, \ldots, k-1$: Index for level l_d at level l

3. $h_{i_d}(l_0, \ldots, l_{k-1}); d = 0, \ldots, k-1$: Index for node i_d at level l

4. $f_d(l_0, \ldots, l_{k-1}, i_0, \ldots, i_{k-1}; d = 0, \ldots, k-1$: Determines the twiddle-factor to use at level l for node $(i_0, i_1, \ldots, i_{k-1})$.

Also like the 2D case, we abbreviate these functions as s_d, g_d, h_d, and f_d. From a programming perspective, it helps to think of the indexing functions as `switch` statements, which are determined by the dimension the kD DFT operates on at level l.

The shift function is:

$$s_d(l_0, \ldots l_{k-1}) = \begin{cases} 2^{m_d - l_d} & l \in [\sum_{j=d+1}^{k-1} m_j + 1, \sum_{j=d}^{k-1} m_j] \\ 0 & \text{otherwise .} \end{cases}$$

The levels function is:

$$g_{l_d}(l_0, \ldots l_{k-1}) = \begin{cases} 0 & l \leq \sum_{j=d+1}^{k-1} m_j \\ l_d - 1 & l \in [\sum_{j=d+1}^{k-1} m_j + 1, \sum_{j=d}^{k-1} m_j] \\ m_d & \text{otherwise.} \end{cases}$$

The node function is:

$$h_{i_d}(l_0, \ldots l_{k-1}) = \begin{cases} 0 & l \leq \sum_{j=d+1}^{k-1} m_j \\ i_d \bmod 2^{l_d - 1} & l \in [\sum_{j=d+1}^{k-1} m_j + 1, \sum_{j=d}^{k-1} m_j] \\ i_d & \text{otherwise.} \end{cases}$$

And finally, the twiddle-factor function is:

$$f_d(l_0, \dots, l_{k-1}, i_0, \dots, i_{k-1}) \quad = \quad \Omega_d(i_d \cdot s_d(l_0, \dots, l_{k-1})). \tag{4.23}$$

Using these four functions, the calculation at an arbitrary node of $\mathbf{T}$ becomes

$$\begin{aligned} T_{(p_0,\dots,p_{k-1}),(l_0,\dots,l_{k-1}),(i_0,\dots,i_{k-1})} &= T_{(p_0-s_0,\dots,p_{k-1}-s_{k-1}),(g_{l_0},\dots,g_{l_{k-1}}),(h_{i_0},\dots,h_{i_{k-1}})} \\ &+ f_d \cdot T_{(p_0,\dots,p_{k-1}),(g_{l_0},\dots,g_{l_{k-1}}),(h_{i_0},\dots,h_{i_{k-1}})}. \end{aligned} \tag{4.24}$$

As an example, let's derive the calculations for the 3D Tree SWDFT algorithm. Let $\mathbf{x}$ be an $N_0 \times N_1 \times N_2$ array, and say we want the SWDFT of all $n_0 \times n_1 \times n_2$ windows, where $n_0 = 2^{m_0}$, $n_1 = 2^{m_1}$, and $n_2 = 2^{m_2}$. The calculation for each node from level 1 to m_2, corresponding to 1D FFTs across dimension 2, is

$$\begin{aligned} T_{(p_0,p_1,p_2),(0,0,l_2),(0,0,i_2)} &= T_{(p_0,p_1,p_2-(2^{m_2-l_2})),(0,0,l_2-1),(0,0,i_2 \bmod 2^{l_2-1})} \\ &+ \Omega_2(i_2 \cdot (2^{m_2-l_2})) \cdot T_{(p_0,p_1,p_2),(0,0,l_2-1),(0,0,i_2 \bmod 2^{l_2-1})}. \end{aligned} \tag{4.25}$$

The calculation for levels $m_2 + 1$ to $m_2 + m_1$ is given by

$$\begin{aligned} T_{(p_0,p_1,p_2),(0,l_1,m_2),(0,i_1,i_2)} &= T_{(p_0,p_1-(2^{m_1-l_1}),p_2),(0,l_1-1,m_2),(0,i_1 \bmod 2^{l_1-1},i_2)} \\ &+ \Omega_1(i_1 \cdot (2^{m_1-l_1})) \cdot T_{(p_0,p_1,p_2),(0,l_1-1,m_2),(0,i_1 \bmod 2^{l_1-1},i_2)}. \end{aligned} \tag{4.26}$$

Finally, the calculations for levels $m_1 + m_2 + 1$ to $m_1 + m_2 + m_3$ is

$$\begin{aligned} T_{(p_0,p_1,p_2),(l_0,m_1,m_2),(i_0,i_1,i_2)} &= T_{(p_0-(2^{m_0-l_0}),p_1,p_2),(l_0-1,m_1,m_2),(i_0 \bmod 2^{l_0-1},i_1,i_2)} \\ &+ \Omega_0(i_0 \cdot (2^{m_0-l_0})) \cdot T_{(p_0,p_1,p_2),(l_0-1,m_1,m_2),(i_0 \bmod 2^{l_0-1},i_1,i_2)}. \end{aligned} \tag{4.27}$$

4.5 Summary

The primary algorithmic contribution of this thesis is the Tree SWDFT algorithm. We explicitly defined a tree data-structure to store and look up repeated fast Fourier transform (FFT) calculations in overlapping windows, extended the algorithm to arbitrary dimensions, and instantiated our algorithm with a C program (Richardson and Eddy (2019)). In addition, Section 3.1 provided a coherent literature review of SWDFT algorithms and synthesized the many existing algorithms into a coherent narrative.

Chapter 5

Statistical Properties of the SWDFT

So far, this thesis has focused on algorithmic aspects of the SWDFT. This chapter shifts the focus, by studying statistical properties of the SWDFT.

When one considers statistical properties of the SWDFT, there are essentially two approaches. The first approach follows the seminal work of Fisher (1929), who showed that, when the input signal is Gaussian, the joint distribution of discrete Fourier transform (DFT) coefficients is given by independent chi-square random variables with two degrees of freedom. The purpose of Fisher's inquiry was understanding whether large DFT coefficients are *statistically significant*, which he formalized with a hypothesis test.

A second approach is based on the classic idea that smooth DFT coefficients provide a consistent estimator of the spectral density. The first extension of this idea to the time-frequency case came from Priestley (1965), who defined the *evolutionary spectra* as a spectral density that slowly changes over time. Many authors have built on this idea, such as Dahlhaus et al. (1997), Adak (1998), and Rosen et al. (2012), and this approach has been successfully applied to many scientific investigations.

Because this thesis studies the SWDFT explicitly, we focus on the first approach. Thus, a natural goal would be extending Fisher's test from the DFT coefficients to the SWDFT coefficients. The primary obstacle in this extension is that DFT coefficients are independent, and SWDFT coefficients are dependent, since SWDFT coefficients in overlapping window positions use the same input data. Therefore, a reasonable first step would be understanding the dependence between SWDFT coefficients, and that's the goal of this chapter.

Specficially, this chapter derives both the asymptotic marginal distribution of, and the covariance between, SWDFT coefficients for white noise signals. And because the asymptotic marginal distribution is normal, knowing the covariance is enough to characterize the joint distribution of SWDFT coefficients. Let's start with a formal definition of white noise signals.

5.1 White Noise Signals

To derive the statistical properties of SWDFT coefficients, we need to make assumptions on the distribution of the input signal. In this chapter, we assume that the input signal is real-valued white noise, for the following reasons:

1. It is a simple and standard base model.
2. Most statistical tests based on the DFT coefficients (for example Fisher (1929), Siegel (1980), Chiu (1989), Juditsky et al. (2015) and Cai et al. (2016) use white noise errors (typically Gaussian) as their null distribution.

Our white noise definition differs from Okamura (2011), who considers complex-valued instead of real-valued white noise. We choose real-valued signals because they're more common in practice.

Setting notation, let $\mathbf{x} = [x_0, x_1, \ldots, x_{N-1}]$ be a 1D length N real-valued signal. We call $\mathbf{x}$ white noise if it has the following properties

- $\mathbb{E}(x_j) = 0$, $\forall j$
- $\mathbf{Var}(x_j) = \sigma^2$, $\forall j$
- x_i, x_j, $i \neq j$ are independent
- $Im(x_j) = 0$, $\forall j$.

We include $Im(x_j) = 0$ because the SWDFT coefficients are complex-valued. The 2D white noise derivation is the same as 1D, except that we replace x_j with x_{j_0,j_1}. With this, we are ready to derive the marginal distribution of SWDFT coefficients for white noise signals.

5.2 The Marginal Distribution of SWDFT Coefficients

The fundamental discovery of Fisher (1929) was that, for an independent Gaussian signal, the real and imaginary parts of DFT coefficients are independent Gaussians. This discovery implies that the squared modulus of a DFT coefficient follows a chi-square distribution with two degrees of freedom. Fisher used these properties to derive the distribution of the maximum DFT coefficient, and this idea still finds applications today (e.g. Ahdesmaki et al. (2007)). In this chapter, we show that Fisher's idea holds white noise signals, since DFT coefficients for white noise signals are asymptotically Gaussian.

5.2.1 1D SWDFT Coefficients

Let $\mathbf{x}$ be a length N white noise signal, as defined in Section 5.1. Let $a_{k,p,n}$ be the SWDFT coefficient for frequency k, window position p, and window size $n \leq N$. Since $\mathbf{x}$ is independent and identically distributed, and this derivation focuses on the marginal distribution, window position doesn't matter, so we remove the subscript p for the remainder of this section. To be concise, let $c_{k,n} = Re(a_{k,n})$ and let $s_{k,n} = Im(a_{k,n})$, where

$$\begin{aligned}
a_{k,n} &= [c_{k,n}, s_{k,n}], \\
c_{k,n} &= \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} x_j \cos(\frac{-2\pi jk}{n}), \\
s_{k,n} &= \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} x_j \sin(\frac{-2\pi jk}{n}).
\end{aligned} \tag{5.1}$$

The expected value of $c_{k,n}$ is

$$\begin{aligned}\mathbb{E}\,(c_{k,n}) &= \mathbb{E}\,(\frac{1}{\sqrt{n}}\sum_{j=0}^{n-1} x_j \cos(\frac{-2\pi jk}{n})),\\ &= 0. \end{aligned} \tag{5.2}$$

The same derivation implies $\mathbb{E}[s_{k,n}] = 0$. For the variance, we use property (C.6)

$$\sum_{j=0}^{n-1} \cos(\frac{2\pi j}{n})^2 = \sum_{j=0}^{n-1} \sin(\frac{2\pi j}{n})^2 = \frac{n}{2}, \tag{5.3}$$

which implies that the variance of $c_{k,n}$ is

$$\begin{aligned}\mathbf{Var}\,(c_{k,p}) &= \mathbf{Var}\,(\frac{1}{\sqrt{n}}\sum_{j=0}^{n-1} x_j \cos(\frac{-2\pi jk}{n}),\\ &= \frac{1}{n}\sum_{j=0}^{n-1}\cos(\frac{-2\pi jk}{n})^2\,\mathbf{Var}\,(x_j),\\ &= \frac{\sigma^2}{n}\cdot\frac{n}{2} = \frac{\sigma^2}{2}.\end{aligned} \tag{5.4}$$

The same derivation gives $\mathbf{Var}\,(s_{k,p}) = \frac{\sigma^2}{2}$. Using (C.6), the covariance between $c_{k,n}$ and $s_{k,n}$ is given by

$$\begin{aligned}\mathbf{Cov}\,(c_{k,n}, s_{k,n}) &= \mathbb{E}\,[c_{k,n}\cdot s_{k,n}] - \mathbb{E}\,[c_{k,n}]\,\mathbb{E}\,[s_{k,n}],\\ &= \mathbb{E}\,[(\frac{1}{\sqrt{n}}\sum_{j=0}^{n-1} x_j\cos(\frac{-2\pi jk}{n}))\cdot(\frac{1}{\sqrt{n}}\sum_{l=0}^{n-1} x_l\sin(\frac{-2\pi lk}{n}))],\\ &= \frac{1}{n}\sum_{j=0}^{n-1}\sum_{l=0}^{n-1}\mathbb{E}\,[x_j x_l]\cos(\frac{-2\pi jk}{n}))\sin(\frac{-2\pi lk}{n}),\\ &= \frac{1}{n}\sum_{m=0}^{n-1}\mathbb{E}\,[x_m^2]\cos(\frac{-2\pi mk}{n})\sin(\frac{-2\pi mk}{n}),\end{aligned}$$

$$\begin{aligned}
&= \frac{\sigma^2}{n}\sum_{m=0}^{n-1}\cos(\frac{-2\pi mk}{n})\sin(\frac{-2\pi mk}{n}), \\
&= \frac{\sigma^2}{2n}\sum_{m=0}^{n-1}\sin(\frac{-4\pi mk}{n}), \\
&= 0. \qquad (5.5)
\end{aligned}$$

Thus, when $\mathbf{x}$ is a white noise signal, the covariance between $c_{k,n}$ and $s_{k,n}$ is exactly zero. Next, we show that the distribution of the SWDFT coefficients is asymptotically normal.

Theorem 5.1 (Asymptotic Normality of SWDFT coefficients)**.** *Let* $x_0, x_1, \ldots \overset{i.i.d}{\sim} F_i$*, where* $\mathbb{E}[x_i] = 0$ *and* $\mathbf{Var}[x_i] = \sigma^2$*. Then the following sums:*

$$\begin{aligned}
c_{k,n} &= \frac{1}{\sqrt{n}}\sum_{j=0}^{n-1} x_j \cos(\frac{-2\pi jk}{n}), \\
s_{k,n} &= \frac{1}{\sqrt{n}}\sum_{j=0}^{n-1} x_j \sin(\frac{-2\pi jk}{n}), \qquad (5.6)
\end{aligned}$$

when scaled by their standard deviations, converge weakly to standard normals, i.e

$$\begin{aligned}
\frac{c_{k,n}}{\sqrt{\mathbf{Var}[c_{k,n}]}} &\rightsquigarrow N(0,1), \\
\frac{s_{k,n}}{\sqrt{\mathbf{Var}[s_{k,n}]}} &\rightsquigarrow N(0,1), \qquad (5.7)
\end{aligned}$$

as $n \to \infty$*.*

Proof. We prove asymptotic normality using Lindeberg's CLT condition (Feller (2008)). The condition states that if random variables $x_0, x_1, \ldots$ are mutually independent with distributions $F_0, F_1, \ldots$, $\mathbb{E}[x_i] = 0$, $\mathbf{Var}[x_i] = \sigma_i^2$, and Lindeberg's condition

$$\lim_{n\to\infty} \frac{1}{s_n^2} \sum_{j=0}^{n-1} \mathbb{E}\left[x_j^2 \cdot \mathbb{1}(|x_j| > \epsilon s_n)\right] \quad \to \quad 0, \tag{5.8}$$

holds for fixed $\epsilon > 0$, where

$$s_n^2 \quad = \quad \sigma_0^2 + \sigma_1^2 + \ldots + \sigma_{n-1}^2, \tag{5.9}$$

then the normalized sums $(\frac{\sum_{j=0}^{n-1} x_j}{s_n})$ converge weakly to a standard normal distribution. To prove that Lindeberg's condition holds, define the following sequence

$$y_{j,k,n} \quad = \quad \frac{1}{\sqrt{n}} x_j \cos(\frac{-2\pi jk}{n}), \tag{5.10}$$

and note that $c_{k,n} = \sum_{j=0}^{n-1} y_{j,k,n}$. We know that $\mathbb{E}\left[y_{j,k,n}\right] = 0$ and $\mathbf{Var}\left[y_{j,k,n}\right] = \cos^2(\frac{-2\pi jk}{n})\sigma^2$, which implies

$$\begin{aligned} s_n^2 &= \sum_{j=0}^{n-1} \mathbf{Var}\left[y_{j,k,n}\right] \\ &= \frac{\sigma^2}{n} \sum_{j=0}^{n-1} \cos^2(\frac{-2\pi jk}{n}), \\ &= \frac{\sigma^2}{n} \cdot \frac{n}{2} = \frac{\sigma^2}{2}. \end{aligned} \tag{5.11}$$

Since the same identity holds for the imaginary part of the SWDFT coefficient $(s_{k,n})$, the same proof works for $s_{k,n}$. With our newly defined sequence (5.10), Lindeberg's condition becomes

$$\begin{aligned}
\lim_{n\to\infty}\frac{1}{s_n^2}\sum_{j=0}^{n-1}\mathbb{E}\left[|(y_{j,k,n})^2|\cdot\mathbb{1}(|y_{j,k,n}|>\epsilon s_n)\right] &= \lim_{n\to\infty}\frac{2}{\sigma^2}\sum_{j=0}^{n-1}\mathbb{E}[(\frac{1}{\sqrt{n}}\cos(\frac{-2\pi jk}{n}x_j))]\mathbb{1}(|\frac{1}{\sqrt{n}}x_j\cos(\frac{-2\pi jk}{n})>\epsilon\frac{\sigma}{\sqrt{2}}),\\
&= \lim_{n\to\infty}\frac{2}{\sigma^2 n}\sum_{j=0}^{n-1}\cos^2(\frac{-2\pi jk}{n})\,\mathbb{E}\left[x_j^2\cdot\mathbb{1}(|x_j|>\frac{\epsilon\sigma\sqrt{n}}{\sqrt{2}|\cos(\frac{-2\pi jk}{n})|})\right],\\
&\leq \lim_{n\to\infty}\frac{2}{\sigma^2 n}\sum_{j=0}^{n-1}\cos^2(\frac{-2\pi jk}{n})\,\mathbb{E}\left[x_1^2\cdot\mathbb{1}(|x_1|>\frac{\epsilon\sigma\sqrt{n}}{\sqrt{2}})\right],\\
&= \lim_{n\to\infty}\frac{2}{\sigma^2 n}\cdot\frac{n}{2}\cdot\mathbb{E}\left[x_1^2\cdot\mathbb{1}(|x_1|>\frac{\epsilon\sigma\sqrt{n}}{\sqrt{2}})\right],\\
&= \lim_{n\to\infty}\frac{1}{\sigma^2}\,\mathbb{E}\left[x_1^2\cdot\mathbb{1}(|x_1|>\frac{\epsilon\sigma\sqrt{n}}{\sqrt{2}})\right],\\
&= 0, \qquad (5.12)
\end{aligned}$$

where the second to third line follows because $\max(\cos(x)) = 1$. So Lindeberg's condition holds, which proves

$$\begin{aligned}
\frac{\sqrt{2}c_{k,n}}{\sigma} &\to N(0,1),\\
\frac{\sqrt{2}s_{k,n}}{\sigma} &\to N(0,1), \qquad (5.13)
\end{aligned}$$

as $n \to \infty$.

□

Theorem 5.1 says that if the input signal is white noise, then the asymptotic distribution of the real part and imaginary part of a SWDFT coefficient is Gaussian. In addition, it may be shown by the Cramer-Wold theorem that the real and imaginary parts of SWDFT coefficients are jointly normally distributed. And if a distribution is jointly normal, then zero covariance implies independence, so $c_{k,n}$ and $s_{k,n}$ are asymptotically independent. nd since the squared modulus of a SWDFT coefficient is given by

$$|a_{k,p}|^2 = Re(a_{k,p})^2 + Im(a_{k,p})^2, \tag{5.14}$$

we know that the squared modulus is asymptotically distributed as chi-square with two degrees of freedom. Symbolically, this means $|a_{k,p}|^2 \to \chi^2_2$.

Finally, although this thesis focuses on white noise signals, some authors have extended these central limit theorem results to stationary sequences, which are more common in the time-series literature. For example, see Peligrad et al. (2010), Section 18.3.2 of Kass et al. (2014), Chapter 10 of Brockwell and Davis (2013), and Lahiri et al. (2003)). Nevertheless, we use white noise asymptotics since it makes our presentation self contained.

5.2.2 2D SWDFT Coefficients

This section extends the marginal distribution of 1D coefficients to 2D. Recall that the 2D SWDFT is

$$a_{k_0,k_1,p_0,p_1} = \frac{1}{\sqrt{n_0 n_1}} \sum_{j_0=0}^{n_0-1} \sum_{j_1=0}^{n_1-1} x_{j_0,j_1} \omega_{n_0}^{-j_0 k_0} \omega_{n_0}^{-j_1 k_1}. \tag{5.15}$$

Like the 1D case, the marginal distribution doesn't depend on window size, so we remove the p subscripts. The real and imaginary parts of the SWDFT coefficients are given by

$$a_{k_0,k_1,n_0,n_1} = [c_{k_0,k_1,n_0,n_1}, s_{k_0,k_1,n_0,n_1}]. \tag{5.16}$$

Using complex multiplication (B.1.1), we can rewrite the real and imaginary parts of the 2D coefficients as

$$
\begin{aligned}
c_{k_0,k_1,n_0,n_1} &= \frac{1}{\sqrt{n_0 n_1}} \sum_{j_0=0}^{n_0-1} \sum_{j_1=0}^{n_1-1} x_{j_0,j_1} [\cos(\frac{-2\pi j_0 k_0}{n_0}) \cos(\frac{-2\pi j_1 k_1}{n_1}) - \sin(\frac{-2\pi j_0 k_0}{n_0}) \sin(\frac{-2\pi j_1 k_1}{n_1})], \\
s_{k_0,k_1,n_0,n_1} &= \frac{1}{\sqrt{n_0 n_1}} \sum_{j_0=0}^{n_0-1} \sum_{j_1=0}^{n_1-1} x_{j_0,j_1} [\cos(\frac{-2\pi j_0 k_0}{n_0}) \sin(\frac{-2\pi j_1 k_1}{n_1}) + \sin(\frac{-2\pi j_0 k_0}{n_0}) \cos(\frac{-2\pi j_1 k_1}{n_1})].
\end{aligned}
\tag{5.17}
$$

Fortunately, we can use the angle addition and difference identities (C.5) to further simplify

$$
\begin{aligned}
c_{k_0,k_1,n_0,n_1} &= \frac{1}{\sqrt{n_0 n_1}} \sum_{j_0=0}^{n_0-1} \sum_{j_1=0}^{n_1-1} x_{j_0,j_1} \cos(-2\pi(\frac{j_0 k_0}{n_0} + \frac{j_1 k_1}{n_1})), \\
s_{k_0,k_1,n_0,n_1} &= \frac{1}{\sqrt{n_0 n_1}} \sum_{j_0=0}^{n_0-1} \sum_{j_1=0}^{n_1-1} x_{j_0,j_1} \sin(-2\pi(\frac{j_0 k_0}{n_0} + \frac{j_1 k_1}{n_1})).
\end{aligned}
\tag{5.18}
$$

The 1D identities (C.6) extend to 2D, namely for integers $q_0, q_1 \in \mathbb{Z}$,

$$
\begin{aligned}
\sum_{j_0=0}^{n_0-1} \sum_{j_1=0}^{n_1-1} \cos(-2\pi((\frac{j_0 q_0}{n_0}) + (\frac{j_1 q_1}{n_1}))) &= 0, \\
\sum_{j_0=0}^{n_0-1} \sum_{j_1=0}^{n_1-1} \sin(-2\pi((\frac{j_0 q_0}{n_0}) + (\frac{j_1 q_1}{n_1}))) &= 0, \\
\sum_{j_0=0}^{n_0-1} \sum_{j_1=0}^{n_1-1} \cos^2(-2\pi((\frac{j_0 q_0}{n_0}) + (\frac{j_1 q_1}{n_1}))) &= \frac{n_0 n_1}{2}, \\
\sum_{j_0=0}^{n_0-1} \sum_{j_1=0}^{n_1-1} \sin^2(-2\pi((\frac{j_0 q_0}{n_0}) + (\frac{j_1 q_1}{n_1}))) &= \frac{n_0 n_1}{2}.
\end{aligned}
\tag{5.19}
$$

This means the same 1D techniques give the following 2D properties

$$
\mathbb{E}[c_{k_0,k_1,n_0 n_1}] = 0,
$$

$$\begin{aligned}
\mathbb{E}\left[s_{k_0,k_1,n_0n_1}\right] &= 0, \\
\mathbf{Var}\left[c_{k_0,k_1,n_0n_1}\right] &= \frac{\sigma^2}{2}, \\
\mathbf{Var}\left[s_{k_0,k_1,n_0n_1}\right] &= \frac{\sigma^2}{2}, \\
\mathbf{Cov}\left[c_{k_0,k_1,n_0n_1}, s_{k_0,k_1,n_0n_1}\right] &= 0. \qquad (5.20)
\end{aligned}$$

A proof similar to Theorem 5.1 also works for 2D,

$$\begin{aligned}
\frac{\sqrt{2}c_{k_0,k_1,n_0,n_1}}{\sigma} &\rightsquigarrow N(0,1), \\
\frac{\sqrt{2}s_{k_0,k_1,n_0,n_1}}{\sigma} &\rightsquigarrow N(0,1), \qquad (5.21)
\end{aligned}$$

as $n_0, n_1 \to \infty$. All of these properties hold for dimensions $k > 2$ as well.

5.3 Covariance of 1D SWDFT Coefficients

In the beginning of this chapter, we mentioned that the main complication in extending results based on DFT coefficients to SWDFT coefficients is that SWDFT coefficients aren't independent. The reason SWDFT coefficients aren't independent is because some window positions use the same data points. This is an issue because most statistical methods that use the DFT coefficients, such as the frequency domain bootstrap (Dahlhaus et al. (1996)), rely on the independence of DFT coefficients. Therefore, in this and the next section we derive the covariance between the complex-valued, real, and imaginary parts of the SWDFT coefficients:

- $\mathbf{Cov}(a_{k,p}, a_{l,p+\delta})$
- $\mathbf{Cov}(Re(a_{k,p}), Re(a_{l,p+\delta}))$
- $\mathbf{Cov}(Re(a_{k,p}), Im(a_{l,p+\delta}))$
- $\mathbf{Cov}(Im(a_{k,p}), Re(a_{l,p+\delta}))$

- $\mathbf{Cov}(Im(a_{k,p}), Im(a_{l,p+\delta}))$,

for $0 < \delta < n$.

Since the input signal ($\mathbf{x}$) is white noise, the only case where SWDFT coefficients are not independent is when window positions use the same data points. Since the DFT is orthogonal, SWDFT coefficients in the same window position are independent. Summarizing these insights gives

$$\begin{aligned} \mathbf{Cov}\left(f(a_{k,p}), f(a_{l,p})\right) &= 0,\ \forall j \neq l, \\ \mathbf{Cov}\left(f(a_{k,p}), f(a_{l,p+\delta})\right) &= 0,\ \forall \delta \geq n. \end{aligned} \tag{5.22}$$

What remains is deriving the covariance between SWDFT coefficients in window positions with overlapping input data.

Our derivations augment Chapter 3 of Okamura (2011) with various trigonometric identities, and extend Okamura's derivations to work in any dimension.

5.3.1 Complex-Number

We start with the covariance between complex-valued SWDFT coefficients. Using the definition of the covariance between complex-valued coefficients in Appendix B, and the fact that $\mathbb{E}[a_{k,p}] = [0, 0]$, we have

$$\begin{aligned} \mathbf{Cov}(a_{k,p}, a_{l,p+\delta}) &= \mathbb{E}[(a_{k,p} - \mathbb{E}[a_{k,p}]) \cdot (a_{l,p+\delta} - \mathbb{E}[a_{l,p+\delta}])^*], \\ &= \mathbb{E}[(a_{k,p}) \cdot (a_{l,p+\delta})^*], \\ &= \mathbb{E}[(\frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} x_{p-n+1+j} \omega_n^{-jk}) \cdot (\frac{1}{\sqrt{n}} \sum_{q=0}^{n-1} x_{p+\delta-n+1+q} \omega_n^{ql})], \\ &= \frac{1}{n} \sum_{j=0}^{n-1} \sum_{q=0}^{n-1} \omega_n^{-jk+ql}\, \mathbb{E}[x_{p-n+1+j} \cdot x_{p+\delta-n+1+q}]. \end{aligned} \tag{5.23}$$

Next, we use $(xy)^* = x^*y^*$ and $(\omega_n^{-j})^* = \omega_n^j$ from Appendix B. Since the input data is independent, we have $\mathbb{E}[x_i x_j] = \sigma^2$ if $i = j$, and 0 otherwise. These facts allow further simplification

$$\begin{aligned}\mathbf{Cov}(a_{k,p}, a_{l,p+\delta}) &= \frac{\sigma^2}{n} \sum_{m=\delta}^{n-1} \omega_n^{-mk+(m-\delta)l}, \\ &= \frac{\sigma^2}{n} \sum_{m=\delta}^{n-1} \omega_n^{m(l-k)-\delta l}. \end{aligned} \tag{5.24}$$

Finally, the summation identities based on the Dirichlet kernel (C.13) give

$$\mathbf{Cov}(a_{k,p}, a_{l,p+\delta}) = \frac{\sigma^2}{n} DW_{\delta,n-1}(\frac{2\pi}{n}((l-k) - \delta l)). \tag{5.25}$$

5.3.2 Real and Imaginary Parts

Next, we derive the covariance between the real and imaginary parts of the SWDFT coefficients. Starting with the real-real covariance, we have

$$\begin{aligned}\mathbf{Cov}(Re(a_{k,p}), Re(a_{l,p+\delta})) &= \mathbb{E}[Re(a_{k,p})Re(a_{l,p+\delta})] - \mathbb{E}[Re(a_{k,p})]\,\mathbb{E}[Re(a_{l,p+\delta})] \\ &= \mathbb{E}[Re(a_{k,p})Re(a_{l,p+\delta})], \\ &= \mathbb{E}[Re(\frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} x_{\hat{p}+j}\omega_n^{-jk}) \cdot Re(\frac{1}{\sqrt{n}} \sum_{q=0}^{n-1} x_{\hat{p}+\delta+q}\omega_n^{-ql})], \\ &= \frac{1}{n} \sum_{j=0}^{n-1} \sum_{q=0}^{n-1} \cos(\frac{-2\pi jk}{n}) \cos(\frac{-2\pi ql}{n})\,\mathbb{E}[x_{\hat{p}+j} x_{\hat{p}+\delta+q}], \\ &= \frac{\sigma^2}{n} \sum_{m=\delta}^{n-1} \cos(\frac{-2\pi mk}{n}) \cos(\frac{-2\pi(m-\delta)l}{n}). \end{aligned} \tag{5.26}$$

This can be further simplified using the product-to-sum identity (C.4) from Appendix C

$$\cos(x)\cos(y) = \frac{1}{2}[\cos(x-y)+\cos(x+y)]. \tag{5.27}$$

Plugging (5.27) into (5.26) results in

$$\begin{aligned} &= \frac{\sigma^2}{2n}\sum_{m=\delta}^{n-1}\cos(\frac{-2\pi mk}{n} - \frac{-2\pi(m-\delta)l}{n}) + \cos(\frac{-2\pi mk}{n} + \frac{-2\pi(m-\delta)l}{n}), \\ &= \frac{\sigma^2}{2n}\sum_{m=\delta}^{n-1}\cos(\frac{-2\pi}{n}(m(k-l)+\delta l)) + \cos(\frac{-2\pi}{n}(m(k+l)-\delta l)). \end{aligned} \tag{5.28}$$

We can also substitute the arbitrary expressions for trigonometric sums (C.14)

$$\begin{aligned} \sum_{j=a}^{b}\cos(jx+\phi) &= \cos((\frac{a+b}{2})x+\phi)D_{b-a+1}(x) = Re(DW_{a,b}(x+\phi)), \\ \sum_{j=a}^{b}\sin(jx+\phi) &= \sin((\frac{a+b}{2})x+\phi)D_{b-a+1}(x) = Im(DW_{a,b}(x+\phi)). \end{aligned} \tag{5.29}$$

Plugging (5.29) identities into (5.28) gives the final expression for the covariance

$$\mathbf{Cov}(Re(a_{k,p}), Re(a_{l,p+\delta})) = \frac{\sigma^2}{2n}[Re(DW_{\delta,n-1}(\frac{-2\pi}{n}((k-l)+\delta l))) + Re(DW_{\delta,n-1}(\frac{-2\pi}{n}((k+l)-\delta l)))]. \tag{5.30}$$

The same derivation works for $\mathbf{Cov}(Re(a_{k,p}), Im(a_{l,p+\delta}))$, $\mathbf{Cov}(Im(a_{k,p}), Re(a_{l,p+\delta}))$, and $\mathbf{Cov}(Im(a_{k,p}), Im(a_{l,p+\delta}))$. Specifically

$$\begin{aligned}
\mathbf{Cov}(Re(a_{k,p}), Im(a_{l,p+\delta})) &= \frac{\sigma^2}{2n}[Im(DW_{\delta,n-1}(\frac{-2\pi}{n}((k+l)-\delta l))) - Im(DW_{\delta,n-1}(\frac{-2\pi}{n}((k-l)+\delta l)))],\\
\mathbf{Cov}(Im(a_{k,p}), Re(a_{l,p+\delta})) &= \frac{\sigma^2}{2n}[Im(DW_{\delta,n-1}(\frac{-2\pi}{n}((k+l)-\delta l))) + Im(DW_{\delta,n-1}(\frac{-2\pi}{n}((k-l)+\delta l)))],\\
\mathbf{Cov}(Im(a_{k,p}), Im(a_{l,p+\delta})) &= \frac{\sigma^2}{2n}[Re(DW_{\delta,n-1}(\frac{-2\pi}{n}((k-l)+\delta l))) + Re(DW_{\delta,n-1}(\frac{-2\pi}{n}((k+l)-\delta l)))].
\end{aligned} \tag{5.31}$$

A visual example that shows the covariance between two coefficients for six different output types is shown in Figure 5.1.

5.4 Covariance of 2D SWDFT Coefficients

This section extends the 1D covariance derivations to 2D. We also show how these derivations extend to higher dimensions.

Setting notation, let $\mathbf{x}$ be an $N_0 \times N_1$ white noise array, and let the window size be $n_0 \times n_1$. Let $a_{(k_0,k_1),(p_0,p_1)}$ be the 2D SWDFT coefficient at frequency $k_0 \times k_1$ and window position $p_0 \times p_1$. To simplify notation, we use the following matrix-vector notation

$$\begin{aligned}
\mathbf{k} &= [k_0, k_1],\\
\mathbf{p} &= [p_0, p_1],\\
\mathbf{n} &= [n_0, n_1],\\
\mathbf{j} &= [j_0, j_1],\\
\delta &= [\delta_0, \delta_1].
\end{aligned} \tag{5.32}$$

For example, the 2D coefficient in matrix-vector notation is $a_{\mathbf{k},\mathbf{p}} = a_{(k_0,k_1),(p_0,p_1)}$.

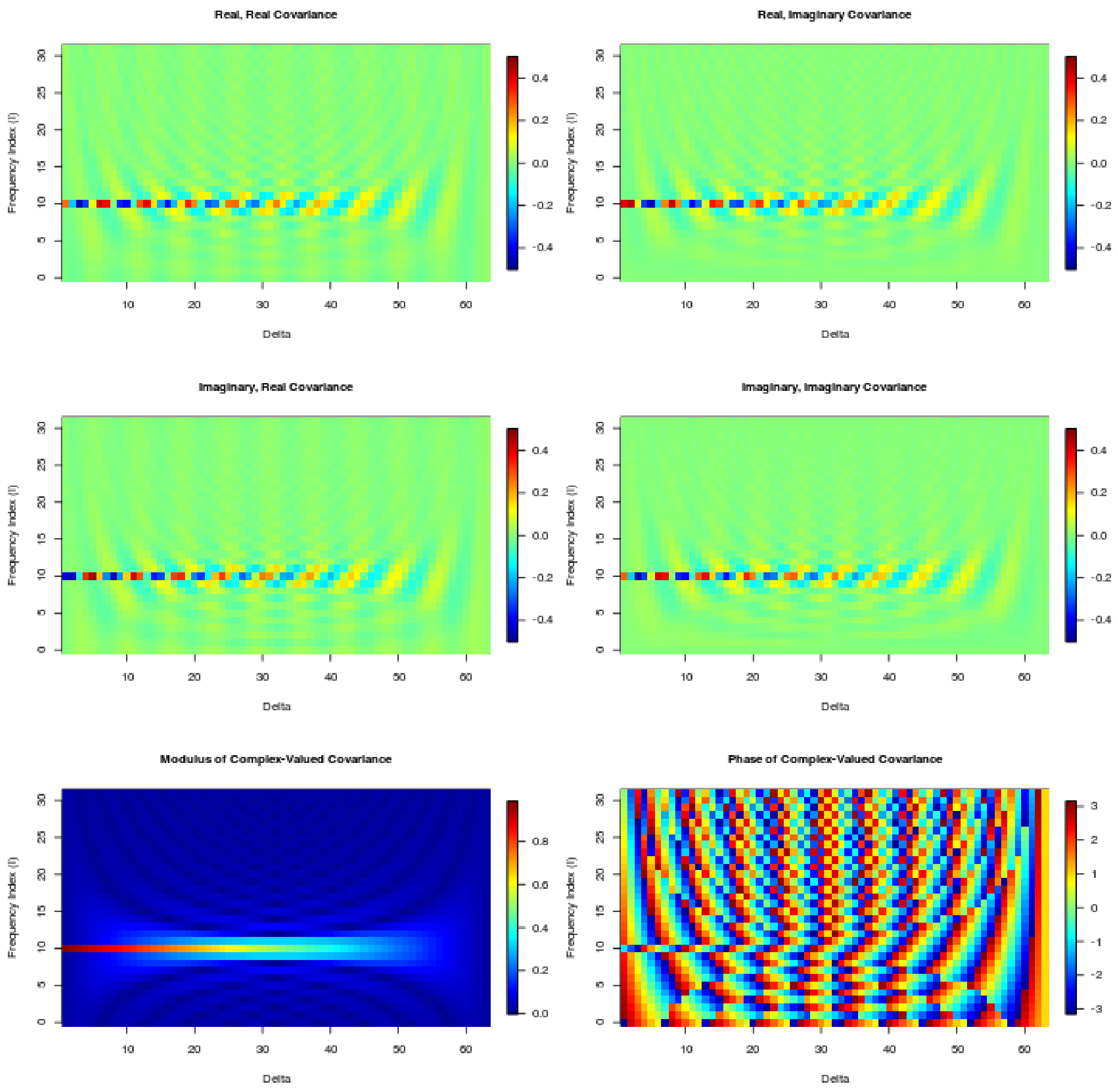

Figure 5.1: Shows the covariance between coefficients $a_{k,p}$ and $a_{l,p+\delta}$ for six different output types. The x-axis varies the shift parameter δ, and the y-axis varies the frequency position l. The other parameters used are $n = 64$ and $\sigma = 1$. The top four plots show the covariance between the real and imaginary parts, and the bottom two plots show the Modulus and Phase of the covariance between the complex-numbers. We see that the covariance is relatively large when the frequencies are the same $k = l = 10$, and for the remainder of the frequencies, the covariance is negligible.

5.4.1 Complex Number

We start with the covariance between the complex-valued output of 2D SWDFT coefficients

$$\begin{aligned}
\mathbf{Cov}(a_{\mathbf{k},\mathbf{p}}, a_{\mathbf{l},\mathbf{p}+\delta}) &= \mathbb{E}[(a_{\mathbf{k},\mathbf{p}})(a_{\mathbf{l},\mathbf{p}+\delta})^*], \\
&= \mathbb{E}[(\frac{1}{\sqrt{n_0 n_1}} \sum_{j_0=0}^{n_0-1} \sum_{j_1=0}^{n_1-1} x_{\mathbf{p}-\mathbf{n}+1+\mathbf{j}} \omega_{n_0}^{-j_0 k_0} \omega_{n_1}^{-j_1 k_1}) \cdot (\frac{1}{\sqrt{n_0 n_1}} \sum_{q_0=0}^{n_0-1} \sum_{q_1=0}^{n_1-1} x_{\mathbf{p}+\delta-\mathbf{n}+1+\mathbf{q}} \omega_{n_0}^{-q_0 l_0} \omega_{n_1}^{-q_1 l_1})^*], \\
&= \frac{1}{n_0 n_1} \sum_{j_0=0}^{n_0-1} \sum_{j_1=0}^{n_1-1} \sum_{q_0=0}^{n_0-1} \sum_{q_1=0}^{n_1-1} \omega_{n_0}^{-j_0 k_0 + q_0 l_0} \omega_{n_1}^{-j_1 k_1 + q_1 l_1} \, \mathbb{E}[x_{\mathbf{p}-\mathbf{n}+1+\mathbf{j}} x_{\mathbf{p}+\delta-\mathbf{n}+1+\mathbf{q}}]. \qquad (5.33)
\end{aligned}$$

Like the 1D case, the expectation inside the sums is only nonzero when $\mathbf{p} - \mathbf{n} + \mathbf{1} + \mathbf{j} = \mathbf{p} + \delta - \mathbf{n} + \mathbf{1} + \mathbf{q}$

$$\begin{aligned}
&= \frac{\sigma^2}{n_0 n_1} \sum_{m_0=\delta_0}^{n_0-1} \sum_{m_1=\delta_1}^{n_1-1} \omega_{n_0}^{-m_0 k_0 + (m_0-\delta_0) l_0} \omega_{n_1}^{-m_1 k_1 + (m_1 - \delta_1) l_1}, \\
&= \frac{\sigma^2}{n_0 n_1} \sum_{m_0=\delta_0}^{n_0-1} \sum_{m_1=\delta_1}^{n_1-1} \omega_{n_0}^{m_0(l_0-k_0) - \delta_0 l_0} \omega_{n_1}^{m_1(l_1-k_1) - \delta_1 l_1}, \\
&= \frac{\sigma^2}{n_0 n_1} \sum_{m_0=\delta_0}^{n_0-1} \omega_{n_0}^{m_0(l_0-k_0) - \delta_0 l_0} \sum_{m_1=\delta_1}^{n_1-1} \omega_{n_1}^{m_1(l_1-k_1) - \delta_1 l_1}. \qquad (5.34)
\end{aligned}$$

We can plug in the same Dirichlet weight identities (C.13) we used in the 1D derivation

$$\mathbf{Cov}(a_{\mathbf{k},\mathbf{p}}, a_{\mathbf{l},\mathbf{p}+\delta}) = \frac{\sigma^2}{n_0 n_1} [DW_{\delta_0, n_0-1}(\frac{2\pi}{n_0}((l_0 - k_0) - \delta_0 l_0) \cdot DW_{\delta_1, n_1-1}(\frac{2\pi}{n_1}((l_1 - k_1) - \delta_1 l_1))]. \qquad (5.35)$$

(5.35) easily generalizes to higher dimensions. For example, the D-dimensional case is

$$\mathbf{Cov}(a_{\mathbf{k},\mathbf{p}}, a_{\mathbf{l},\mathbf{p}+\delta}) = \frac{\sigma^2}{\prod_{d=0}^{D-1} n_d} \cdot [\prod_{d=0}^{D-1} DW_{\delta_d, n_d - 1} \frac{2\pi}{n_d}((l_d - k_d) - \delta_d l_d)]. \qquad (5.36)$$

5.4.2 Real and Imaginary Parts

We conclude by deriving the covariance of the real and imaginary parts of the 2D SWDFT coefficients. Starting with the real parts

$$\begin{aligned}
\mathbf{Cov}(Re(a_{\mathbf{k},\mathbf{p}}), Re(a_{\mathbf{l},\mathbf{p}+\delta})) &= \mathbb{E}[Re(a_{\mathbf{k},\mathbf{p}}) \cdot Re(a_{\mathbf{l},\mathbf{p}+\delta})], \\
&= \frac{1}{n_0 n_1} \mathbb{E}[Re(\sum_{j_0=0}^{n_0-1}\sum_{j_1=0}^{n_1-1} x_{\mathbf{p}-\mathbf{n}+\mathbf{1}+\mathbf{j}} \omega_{n_0}^{-j_0 k_0}\omega_{n_1}^{-j_1 k_1}) \cdot Re(\sum_{q_0=0}^{n_0-1}\sum_{q_1=0}^{n_1-1} x_{\mathbf{p}+\delta-\mathbf{n}+\mathbf{1}+\mathbf{j}} \omega_{n_0}^{-q_0 l_0}\omega_{n_1}^{-q_1 l_1})], \\
&= \frac{1}{n_0 n_1} \sum_{j_0=0}^{n_0-1}\sum_{j_1=0}^{n_1-1}\sum_{q_0=0}^{n_0-1}\sum_{q_1=0}^{n_1-1} Re(\omega_{n_0}^{-j_0 k_0}\omega_{n_1}^{-j_1 k_1}) Re(\omega_{n_0}^{-q_0 l_0}\omega_{n_1}^{-q_1 l_1})\, \mathbb{E}[x_{\mathbf{p}+\delta-\mathbf{n}+\mathbf{1}+\mathbf{j}}], \\
&= \frac{\sigma^2}{n_0 n_1} \sum_{m_0=\delta_0}^{n_0-1}\sum_{m_1=\delta_1}^{n_1-1} Re(\omega_{n_0}^{-m_0 k_0}\omega_{n_1}^{-m_1 k_1}) Re(\omega_{n_0}^{-(m_0-\delta_0)l_0}\omega_{n_1}^{-(m_1-\delta_1)l_1}). \qquad (5.37)
\end{aligned}$$

Unfortunately, due to the multiplication property of complex numbers

$$(x_1, x_2) \cdot (y_1, y_2) = (x_1 y_1 - x_2 y_2, x_1 y_2 + x_2 y_1), \qquad (5.38)$$

(5.37) doesn't simplify as nicely as the 2D complex-valued covariance, so we leave the final expression like this. The same derivation works in any dimensions, and for all combinations of real and imaginary SWDFT coefficients, thus

$$\begin{aligned}
\mathbf{Cov}(Re(a_{\mathbf{k},\mathbf{p}}), Re(a_{\mathbf{l},\mathbf{p}+\delta})) &= \frac{\sigma^2}{\prod_{d=0}^{D-1} n_d} \sum_{m_0=\delta_0}^{n_0-1} \cdots \sum_{m_{D-1}=\delta_{D-1}}^{n_{D-1}-1} Re(\omega_{n_0}^{-m_0 k_0} \cdots \omega_{n_{D-1}}^{-m_{D-1}k_{D-1}}) \cdot Re(\omega_{n_0}^{(m-\delta_0)l_0} \cdots \omega_{n_{D-1}}^{(m_{D-1}-\delta_{D-1})l_{D-1}}), \\
\mathbf{Cov}(Re(a_{\mathbf{k},\mathbf{p}}), Im(a_{\mathbf{l},\mathbf{p}+\delta})) &= \frac{\sigma^2}{\prod_{d=0}^{D-1} n_d} \sum_{m_0=\delta_0}^{n_0-1} \cdots \sum_{m_{D-1}=\delta_{D-1}}^{n_{D-1}-1} Re(\omega_{n_0}^{-m_0 k_0} \cdots \omega_{n_{D-1}}^{-m_{D-1}k_{D-1}}) \cdot Im(\omega_{n_0}^{(m-\delta_0)l_0} \cdots \omega_{n_{D-1}}^{(m_{D-1}-\delta_{D-1})l_{D-1}}), \\
\mathbf{Cov}(Im(a_{\mathbf{k},\mathbf{p}}), Re(a_{\mathbf{l},\mathbf{p}+\delta})) &= \frac{\sigma^2}{\prod_{d=0}^{D-1} n_d} \sum_{m_0=\delta_0}^{n_0-1} \cdots \sum_{m_{D-1}=\delta_{D-1}}^{n_{D-1}-1} Re(\omega_{n_0}^{-m_0 k_0} \cdots \omega_{n_{D-1}}^{-m_{D-1}k_{D-1}}) \cdot Im(\omega_{n_0}^{(m-\delta_0)l_0} \cdots \omega_{n_{D-1}}^{(m_{D-1}-\delta_{D-1})l_{D-1}}), \\
\mathbf{Cov}(Im(a_{\mathbf{k},\mathbf{p}}), Im(a_{\mathbf{l},\mathbf{p}+\delta})) &= \frac{\sigma^2}{\prod_{d=0}^{D-1} n_d} \sum_{m_0=\delta_0}^{n_0-1} \cdots \sum_{m_{D-1}=\delta_{D-1}}^{n_{D-1}-1} Im(\omega_{n_0}^{-m_0 k_0} \cdots \omega_{n_{D-1}}^{-m_{D-1}k_{D-1}}) \cdot Im(\omega_{n_0}^{(m-\delta_0)l_0} \cdots \omega_{n_{D-1}}^{(m_{D-1}-\delta_{D-1})l_{D-1}}).
\end{aligned}$$

(5.39)

5.5 The Joint Distribution

Between the marginal distribution and the covariance, we have enough information to characterize the asymptotic joint distribution of SWDFT coefficients for white noise signals. Since the asymptotic marginal distribution of the real and imaginary parts is Gaussian with a known covariance, and the covariance completely determines a Gaussian distribution, we have everything we need. For example, the bivariate distribution of $a_{k,p}$ and $a_{l,p+\delta}$ is

$$[a_{k,p}, a_{l,p+\delta}] = [Re(a_{k,p}), Im(a_{k,p}), Re(a_{l,p+\delta}), Im(a_{l,p+\delta})] \quad \rightsquigarrow \quad MVN(0, \Sigma), \tag{5.40}$$

where

$$\Sigma = \begin{bmatrix} \frac{\sigma^2}{2} & 0 & \mathbf{Cov}(Re(a_{k,p}), Re(a_{l,p+\delta})) & \mathbf{Cov}(Re(a_{k,p}), Im(a_{l,p+\delta})) \\ 0 & \frac{\sigma^2}{2} & \mathbf{Cov}(Im(a_{k,p}), Re(a_{l,p+\delta})) & \mathbf{Cov}(Im(a_{k,p}), Im(a_{l,p+\delta})) \\ \mathbf{Cov}(Re(a_{l,p+\delta}), Re(a_{k,p})) & \mathbf{Cov}(Re(a_{l,p+\delta}), Im(a_{k,p})) & \frac{\sigma^2}{2} & 0 \\ \mathbf{Cov}(Im(a_{l,p+\delta}), Re(a_{k,p})) & \mathbf{Cov}(Im(a_{l,p+\delta}), Im(a_{k,p})) & 0 & \frac{\sigma^2}{2}. \end{bmatrix}. \tag{5.41}$$

The covariance terms in (5.41) were all derived in this chapter.

5.6 Summary

This chapter derived the statistical properties required to characterize the asymptotic joint distribution of SWDFT coefficients for white noise signals. We showed that the marginal distribution of the real and imaginary parts of SWDFT coefficients converge weakly to a Gaussian distribution, and we derived the analytic covariance between the real, imaginary, and complex-valued parts of SWDFT coefficients. Some ideas on how to further develop these results are given in Section 9.1.2.

Chapter 6

Local Cosine Regression

When statisticians model signals with oscillations, they often fit a cosine function to data, which we call *cosine regression.* Cosine regression assumes that oscillations persist for the entire duration of a signal, such as the example shown in the top panel of Figure 6.1. But some oscillations only occur locally, like the signal in the bottom panel of Figure 6.1. This chapter extends cosine regression to handle local signals, and we call this *local cosine regression.* We show that, just as cosine regression uses the maximum DFT coefficient for estimation, local cosine regression uses the maximum SWDFT coefficient over all possible window sizes.

Cosine regression is widely used across statistics and signal processing, so we start by deriving cosine regression and summarizing the vast literature on the topic. Next, we show how to localize the cosine regression model, and how estimation for the frequency component in this model uses the maximum SWDFT coefficient over all possible window sizes. We conclude by describing our implementation of local cosine regression, and illustrate how it works with an example.

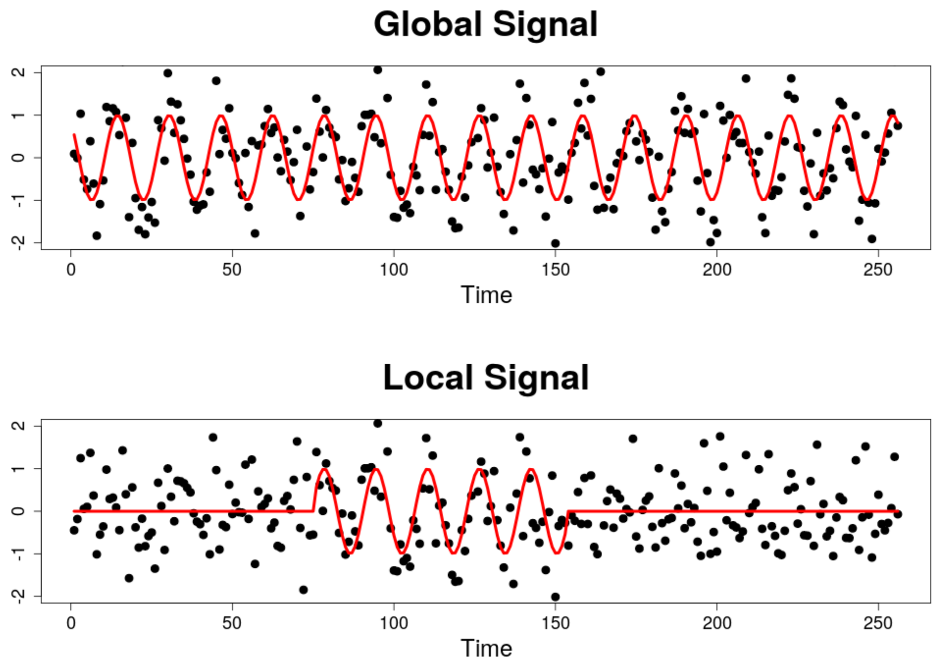

Figure 6.1: The difference between a global signal and a local signal. The global signal oscillates for the entire duration, and the local signal only oscillates for a short period in the middle of the signal.

6.1 Cosine Regression

Perhaps the most well studied frequency domain statistical method fits a cosine function to data, which we call *cosine regression.* Cosine regression is sometimes called *harmonic regression*, or simply *fitting sinusoids* (see Chapter 2 of Bloomfield (2004)). Whatever you call it, the cosine regression model is

$$\begin{aligned} y_t &= A\cos(\frac{2\pi tF}{N} + \phi) + \epsilon_t \\ t &= 0, 1, \ldots, N-1, \end{aligned} \tag{6.1}$$

where ϵ_t is noise, typically assumed to be either independent or stationary. Cosine regression requires estimation of three parameters:

- A: Amplitude. $A \in [0, \infty]$. Indicates how large the peaks of the oscillation are.

- $f = \frac{2\pi F}{N}$: Frequency. Interpreted as F cycles in a length N signal, with $\frac{F}{N} \in [0, \frac{1}{2}]$.

- ϕ: Phase. $\phi \in [0, 2\pi]$. Shifts the cosine function in time.

To be concise, let $f = \frac{2\pi F}{N}$ for the rest of the chapter.

The cosine regression model is non-linear, and this fact complicates estimation. To deal with non-linearity, there is a standard trick that *linearizes* (6.1) in parameters A and ϕ. The trick uses the following trigonometric identity (C.5)

$$\cos(x+y) = \cos(x)\cos(y) - \sin(x)\sin(y). \tag{6.2}$$

Plugging (6.2) identity into (6.1) gives

$$y_t = A\cos(\phi)\cos(ft) - A\sin(\phi)\sin(ft) + \epsilon_t. \tag{6.3}$$

Then using the following change of variables

$$\begin{aligned} \beta_1 &= A\cos(\phi), \\ \beta_2 &= -A\sin(\phi), \end{aligned}$$

the cosine regression model becomes

$$y_t = \beta_1\cos(ft) + \beta_2\sin(ft) + \epsilon_t. \tag{6.4}$$

If we assume that frequency (f) is known, then $\hat{\beta}_1$ and $\hat{\beta}_2$ can be easily estimated with least squares, and we can derive $\hat{A}$ and $\hat{\phi}$ with the following one-to-one transformation

$$\begin{aligned} \hat{A} &= \sqrt{\hat{\beta}_1^2 + \hat{\beta}_2^2}, \\ \hat{\phi} &= \arctan(\frac{-\hat{\beta}_2}{\hat{\beta}_1}). \end{aligned} \tag{6.5}$$

But in many situations, we don't know the frequency (f), and we need to estimate it. A vast literature on the estimation of frequency for the cosine regression model exists, so we give a brief synthesis in the next paragraph.

To the best of my knowledge, the first solution for frequency estimation in cosine regression comes from Whittle (1952). Whittle showed that the least squares estimate for f in (6.4) is equivalent to the maximum DFT coefficient, when f is restricted the be a Fourier frequency. *This is the most important fact connecting cosine regression with Fourier analysis.* The second wave of solutions came in the early 1970's, when Walker (1971) derived the asymptotic distribution of the Whittle estimator for iid errors, then extended the error model to include stationary linear processes (Walker (1973)). Around the same time, Hannan (1971, 1973, 1974) extended the results to include a strictly stationary error term. Since then, various applications and improvements have been proposed in the literature. Brillinger (1987) summarized the uses of cosine regression and similar models in scientific applications. Brown (1990) corrected the asymptotic distributions derived by Hannan. Greenhouse et al. (1987) fitted a multiple sinusoid model with ARMA errors using maximum likelihood. Rice and Rosenblatt (1988) discuss computational issues, including upward bias in the amplitude estimate when the true amplitude is small, and the nonlinear optimization surface having many local minima. Irizarry (1998, 2001, 2002) extended estimation from least squares to weighted least squares, and proposed *local harmonic regression*, which fits multiple sinusoids at each time point. Kay (1993) discusses estimation of sinusoids from a signal processing perspective. Nadler and Kontorovich (2011) gave a model selection approach to determine the number of sinusoids embedded in noise, using a likelihood ratio test and penalty term. Quinn and Hannan (2001) wrote an influential book on frequency estimation. And there is much, much, more. An excellent recent synthesis of the literature is given in the book by Kundu and Nandi (2012).

6.1.1 The Periodogram Estimator

The most important fact that connects cosine regression with Fourier analysis is that the least squares estimate for the frequency parameter (f) is the maximum of the *periodogram function*, which is a continuous analog of the DFT. Since this is so important, we introduce the periodogram function and its connection with cosine regression in this section. Setting notation, for a length N real or complex-valued signal, the periodogram function is

$$I(f) = \frac{1}{N}|\sum_{j=0}^{N-1} x_j \omega_n^{-jf}|^2. \tag{6.6}$$

The maximum of (6.6)

$$\hat{f} = \arg\max_f I(f), \tag{6.7}$$

is the least squares estimator for f in the cosine regression model. Since the periodogram function is non-convex, maximizing it directly can lead to local minima. To account for this, the standard approach uses the maximum DFT coefficient as a starting value for non-linear optimization.

To be more precise, let $\theta = [A, \phi, F]$ be the parameters in the cosine regression model, and let $\mathrm{T}(\theta)$ be the least squares criteria we want to minimize, namely

$$T(\theta) = \sum_{t=0}^{N-1}(x_t - A\cos(ft + \phi))^2. \tag{6.8}$$

Kundu and Nandi (2012) show that if we use the *linearization* trick to estimate A and ϕ, and estimate f by maximizing the periodogram function, then θ is a strongly consistent estimator. Kundu and Nandi (2012) also show that θ is asymptotically normal, and derive the limiting covariance matrix of θ under different noise processes.

6.2 Local Cosine Regression

Cosine regression only works for oscillations that last for the entire signal, and is not appropriate for local oscillations. Figure 6.1 demonstrates the difference; the top panel shows a global signal, where cosine regression is appropriate, and the bottom panel shows a local signal, where cosine regression is not appropriate. For local signals, we propose local cosine regression, and to the best of my knowledge, we are the first to propose localizing the cosine regression model.

Just like the DFT is used for frequency estimation in cosine regression, the SWDFT is used for frequency estimation in local cosine regression. But in addition to frequency estimation, the SWDFT also estimates the length and starting position of a local signal. In my opinion, the most interesting aspect of local cosine regression is that both the least squares and maximum likelihood estimators are found by maximizing the SWDFT over all possible window sizes.

The local cosine regression model applies a window function to the cosine regression model, and to start simple, we use a rectangular sliding window function. The rectangular sliding window function is given by

$$\mathbb{1}_{[S,S+L-1]}(t) = \begin{cases} 1 & S \leq j \leq S+L-1 \\ 0 & \text{otherwise.} \end{cases} \tag{6.9}$$

Using (6.9), the local cosine regression model is

$$
\begin{aligned}
y_t &= A\cos(ft+\phi)\cdot \mathbb{1}_{[S,S+L-1]}(t)+\epsilon_t, \\
t &= 0,1,\ldots,N-1. \qquad (6.10)
\end{aligned}
$$

For now, assume that the errors (ϵ_t) are either iid or stationary, and we will be more precise about the error structure in Section 6.3. The local cosine regression model adds two additional parameters to the cosine regression model:

- S: Start of oscillation. $S \in [0, 1, \ldots, N-2]$.
- L: Length of oscillation. $L \in [1, 2, \ldots, N-2]$.

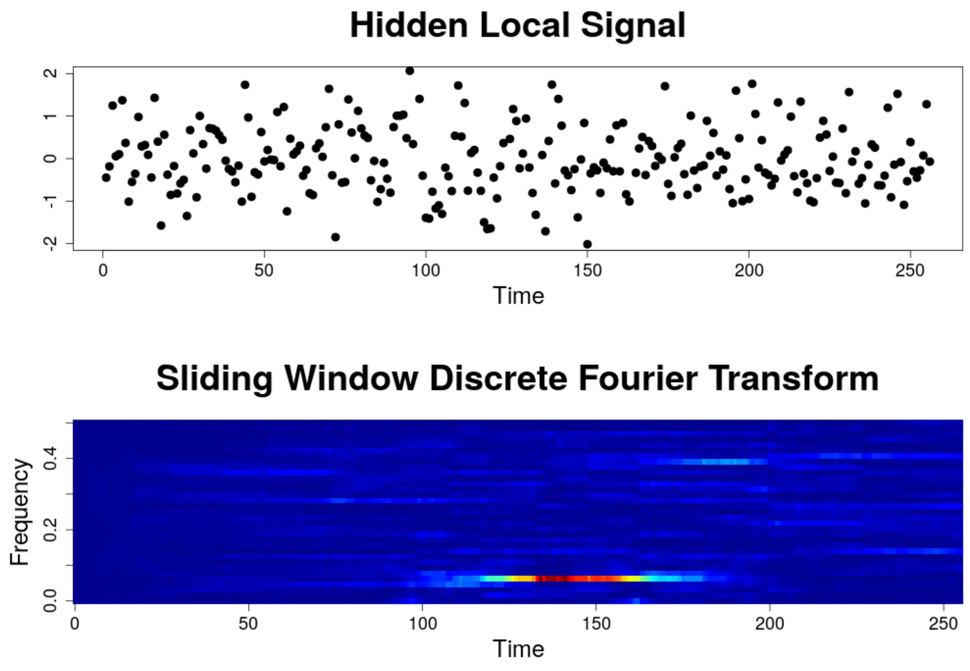

Figure 6.2: Top: Realization of the local cosine regression model with noise. Bottom: The SWDFT of the top panel.

A realization of the local cosine regression model with Gaussian noise and its corresponding SWDFT is shown in Figure 6.3. This figure shows that hidden local signals are difficult to see in the time domain, but are very distinct in the time-frequency domain. In particular, the SWDFT shows *when* the local cosine signal starts, *how long* it lasts, and at *which frequency* it occurs. Intuitively, this is why the SWDFT is a good estimator for f, S, and L in our local cosine regression model.

6.3 Estimation

Earlier, we hinted that the maximum likelihood estimator of the local cosine regression model is approximately the maximum SWDFT coefficient over window size. This section formally derives this fact. To keep things simple, we follow Rice and Rosenblatt (1988) by assuming that the errors are independent and identically distributed as white Gaussian noise $\epsilon_t \sim N(0, \sigma^2)$. However, our results should be similar for stationary errors. Our derivation closely follows a similar derivation for cosine regression in Chapter 8 of Kay (1993), where the main difference is the additional parameters S and L.

Setting notation, let $\mathbf{x} = [x_0, x_1, \ldots, x_{N-1}]$ be a length N signal, then the distribution of each data point is

$$x_t \sim N(A\cos(ft + \phi)\mathbb{1}_{[S,S+L-1]}(t), \sigma^2). \tag{6.11}$$

Let $\theta = [A, f, \phi, S, L, \sigma]$ be our parameter vector. The joint pdf of $\mathbf{x}$ is given by

$$\mathbb{P}(\mathbf{x}|\theta) = \prod_{t=0}^{N-1} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(x_t - A\cos(ft + \phi)\mathbb{1}_{[S,S+L-1]}(t))^2}{2\sigma^2}\right). \tag{6.12}$$

To be concise, let $g_t = A\cos(ft + \phi)\mathbb{1}_{[S,S+L-1]}(t)$ henceforth. The likelihood function is given by

$$L(\theta|\mathbf{x}) = \prod_{t=0}^{N-1} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(x_t - g_t)^2}{2\sigma^2}\right), \tag{6.13}$$

and the log likelihood is proportional to

$$\ell(\theta) \quad \propto \quad -N\log(\sigma) - \frac{1}{2\sigma^2}\sum_{t=0}^{N-1}(x_t - g_t)^2. \tag{6.14}$$

Define the summation term inside the log likelihood, which is equivalent to the least squares criteria

$$T(A,\phi,f,S,L) \quad = \quad \sum_{t=0}^{N-1}(x_t - g_t)^2, \tag{6.15}$$

and notice that the maximum likelihood estimate for A, ϕ, f, S, and L corresponds to the (scaled) least squares estimate

$$(\hat{A},\hat{\phi},\hat{f},\hat{S},\hat{L}) \quad = \quad \underset{A,\phi,f,S,L}{\arg\min}\,\frac{1}{N}T(A,\phi,f,S,L).$$

If we linearize the model using the same trick from cosine regression ((6.2) and (6.3)), then g_t becomes

$$g_t \quad = \quad \beta_1\cos(ft)\mathbb{1}_{[S,S+L-1]} + \beta_2\sin(ft)\mathbb{1}_{[S,S+L-1]}. \tag{6.16}$$

If we assume that f, S, and L are known, (6.16) is just multiple linear regression. Because multiple linear regression is easier to understand in matrix-vector notation, we define

$$\begin{aligned}\mathbf{c} \quad &= \quad [\cos(f\cdot 0)\mathbb{1}_{S,S+L-1}(0),\ldots,\cos(f\cdot N-1)\mathbb{1}_{[S,S+L-1]}(N-1)],\\ \mathbf{s} \quad &= \quad [\sin(f\cdot 0)\mathbb{1}_{S,S+L-1}(0),\ldots,\sin(f\cdot N-1)\mathbb{1}_{[S,S+L-1]}(N-1)],\end{aligned}$$

$$\begin{aligned}\mathbf{U} &= [\mathbf{c}, \mathbf{s}], \\ \beta &= [\beta_1, \beta_2]. \end{aligned} \tag{6.17}$$

Under this notation, the solution for β is given by

$$\hat{\beta} = (\mathbf{U}^T\mathbf{U})^{-1}\mathbf{U}^T\mathbf{x}. \tag{6.18}$$

The next question is, how to estimate f, S, and L? As a first step, we plug (6.18) back into the least-squares criteria (6.16)

$$\begin{aligned} T(\hat{\beta}_1, \hat{\beta}_2, f, S, L) &= \frac{1}{N}(\mathbf{x} - \mathbf{U}\hat{\beta})^T(\mathbf{x} - \mathbf{U}\hat{\beta}), \\ &= \frac{1}{N}(\mathbf{x} - \mathbf{U}(\mathbf{U}^T\mathbf{U})^{-1}\mathbf{U}\mathbf{x})^T(\mathbf{x} - \mathbf{U}(\mathbf{U}^T\mathbf{U})^{-1}\mathbf{U}\mathbf{x}), \\ &= \frac{1}{N}\mathbf{x}^T(\mathbf{I} - \mathbf{U}(\mathbf{U}^T\mathbf{U})^{-1}\mathbf{U}^T)\mathbf{x}, \\ &= \frac{1}{N}(\mathbf{x}^T\mathbf{x} - \mathbf{x}^T\mathbf{U}(\mathbf{U}^T\mathbf{U})^{-1}\mathbf{U}^T\mathbf{x}). \end{aligned} \tag{6.19}$$

(6.19) implies we want the values of f, S, and L that solve the following maximization problem

$$\begin{aligned} \arg\max_{f,S,L} \quad & \frac{1}{N}\mathbf{x}^T\mathbf{U}(\mathbf{U}^T\mathbf{U})^{-1}\mathbf{U}^T\mathbf{x}, \\ = \quad & \frac{1}{N}\begin{bmatrix}\mathbf{c}^T\mathbf{x} \\ \mathbf{s}^T\mathbf{x}\end{bmatrix}^T \begin{bmatrix}\mathbf{c}^T\mathbf{c} & \mathbf{c}^T\mathbf{s} \\ \mathbf{s}^T\mathbf{c} & \mathbf{s}^T\mathbf{s}\end{bmatrix}^{-1} \begin{bmatrix}\mathbf{c}^T\mathbf{x} \\ \mathbf{s}^T\mathbf{x}\end{bmatrix}. \end{aligned} \tag{6.20}$$

The solution to (6.20) is where estimation for cosine regression and local cosine regression diverge. In cosine regression, the 2×2 matrix in (6.20) simplifies to a diagonal matrix with $\frac{N}{2}$ on both diagonal entires. In local cosine regression, this doesn't work out as nicely, because the diagonal elements aren't exactly $\frac{N}{2}$, and the non-diagonal elements aren't exactly 0. In this derivation, we show that as the signal length gets large ($L < N \to \infty$), the 2×2 matrices in both cosine and local cosine regression are approximately the same. To see this, we can write the 2×2 matrix as

$$\begin{bmatrix} \mathbf{c}^T\mathbf{c} & \mathbf{c}^T\mathbf{s} \\ \mathbf{s}^T\mathbf{c} & \mathbf{s}^T\mathbf{s} \end{bmatrix} = \begin{bmatrix} \sum_{t=0}^{N-1} \cos(ft)^2 \mathbb{1}_{[S,S+L-1]} & \sum_{t=0}^{N-1} \cos(ft)\sin(ft)\mathbb{1}_{S,S+L-1} \\ \sum_{t=0}^{N-1} \cos(ft)\sin(ft)\mathbb{1}_{[S,S+L-1]} & \sum_{t=0}^{N-1} \sin(ft)^2 \mathbb{1}_{S,S+L-1} \end{bmatrix}. \tag{6.21}$$

To approximate the diagonal entries of (6.21), we use (2.5) and (2.6) on page 8 of Kundu and Nandi (2012)

$$\begin{aligned} \frac{1}{N}\sum_{t=1}^{N} \cos^2(ft) &= \frac{1}{2} + o(\frac{1}{N}), \\ \frac{1}{N}\sum_{t=1}^{N} \sin^2(ft) &= \frac{1}{2} + o(\frac{1}{N}). \end{aligned} \tag{6.22}$$

(6.22) implies that the diagonal elements can be approximated as

$$\begin{aligned} \sum_{t=0}^{N-1} \cos(ft)^2 \mathbb{1}_{[S,S+L-1]} = \sum_{t=S}^{S+L-1} \cos^2(ft) &= \frac{L}{2} + o(\frac{1}{L}), \\ \sum_{t=0}^{N-1} \sin(ft)^2 \mathbb{1}_{[S,S+L-1]} = \sum_{t=S}^{S+L-1} \sin^2(ft) &= \frac{L}{2} + o(\frac{1}{L}). \end{aligned} \tag{6.23}$$

The next step is approximating the non-diagonal elements. Both non-diagonal elements are equivalent, and can be written as

$$\sum_{t=0}^{N-1} \cos(ft)\sin(ft)\mathbb{1}_{[S,S+L-1]} \quad = \quad \sum_{t=S}^{S+L-1} \cos(ft)\sin(ft). \tag{6.24}$$

The exact solution to (6.24) can be found by first applying the product-to-sum trigonometric identity (C.4), followed by the generalized complex exponential sum identity (C.14)

$$\begin{aligned}\sum_{t=S}^{S+L-1} \cos(ft)\sin(ft) &= \frac{1}{2}\sum_{t=S}^{S+L-1} \sin(2ft), \\ &= \frac{1}{2}\sin((2S+L-1)f)D_L(2f). \end{aligned} \tag{6.25}$$

But we can also approximate (6.24) with (2.7) on page 8 of Kundu and Nandi (2012))

$$\frac{1}{n^{k+1}}\sum_{t=1}^{n} t^k \cos(ft)\sin(ft) \quad = \quad o(\frac{1}{N}), \tag{6.26}$$

and plugging $k = 0$ into (6.26) implies that as $L \to \infty$

$$\frac{1}{L}\sum_{t=S}^{S+L-1} \cos(ft)\sin(ft) \quad = \quad o(\frac{1}{L}). \tag{6.27}$$

The inverse of the 2×2 matrix as $L \to \infty$ is given by

$$\begin{bmatrix} \mathbf{c}^T\mathbf{c} & \mathbf{c}^T\mathbf{s} \\ \mathbf{s}^T\mathbf{c} & \mathbf{s}^T\mathbf{s} \end{bmatrix}^{-1} \quad = \quad \frac{1}{(\mathbf{c}^T\mathbf{c}\cdot\mathbf{s}^T\mathbf{s}) - (\mathbf{c}^T\mathbf{s}\cdot\mathbf{s}^T\mathbf{c})} \begin{bmatrix} \mathbf{s}^T\mathbf{s} & -\mathbf{c}^T\mathbf{s} \\ -\mathbf{s}^T\mathbf{c} & \mathbf{c}^T\mathbf{c} \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} \frac{2}{L} & 0 \\ 0 & \frac{2}{L} \end{bmatrix}. \tag{6.28}$$

We can plug (6.28) back into (6.20)

$$\begin{aligned}
\underset{f,S,L}{\arg\max} \quad & \frac{1}{N}\mathbf{x}^T\mathbf{U}(\mathbf{U}^T\mathbf{U})^{-1}\mathbf{U}^T\mathbf{x}, \\
\approx \quad & \frac{1}{N}\begin{bmatrix} \mathbf{c}^T\mathbf{x} \\ \mathbf{s}^T\mathbf{x} \end{bmatrix}^T \begin{bmatrix} \frac{2}{L} & 0 \\ 0 & \frac{2}{L} \end{bmatrix} \begin{bmatrix} \mathbf{c}^T\mathbf{x} \\ \mathbf{s}^T\mathbf{x} \end{bmatrix}, \\
= \quad & \frac{2}{L}[(\mathbf{c}^T\mathbf{x})^2 + (\mathbf{s}^T\mathbf{x})^2]. \qquad (6.29)
\end{aligned}$$

This brings us to the most important observation; we can rewrite (6.29) in the same form as the SWDFT. That is

$$\begin{aligned}
\frac{2}{L}[(\mathbf{c}^T\mathbf{x})^2 + (\mathbf{s}^T\mathbf{x})^2] &= \frac{2}{L}[(\sum_{t=0}^{N-1} x_t \cos(ft)\mathbb{1}_{S,S+L-1}(t))^2 + (\sum_{t=0}^{N-1} x_t \sin(ft)\mathbb{1}_{S,S+L-1}(t))^2], \\
&= \frac{2}{L}|\sum_{t=0}^{N-1} x_t e^{-ift)}\mathbb{1}_{S,S+L-1}(t)|^2, \\
&= \frac{2}{L}|\sum_{t=S}^{S+L-1} x_t e^{-ift}|^2. \qquad (6.30)
\end{aligned}$$

If we define the *spectrogram function* by

$$I(f,S,L) = \frac{2}{L}|\sum_{t=S}^{S+L-1} x_t e^{-ift}|^2, \tag{6.31}$$

then the final maximization problem is

$$\underset{f,S,L}{\arg\max} \qquad I(f,S,L). \tag{6.32}$$

(6.32) is essentially the periodogram estimator for cosine regression, except we now have additional parameters for signal starting position (S) and signal length (L).

One can solve (6.32) using standard optimization algorithms, such as Newton-Raphson. But for large signals, searching the entire space is computationally intensive, and the search is almost guaranteed to end up at a local minimum (Quinn et al. (2008)). This is where we use the SWDFT. Just as non-linear optimization algorithms for the periodogram function use the maximum of the DFT as a starting value, we can use the maximum SWDFT coefficient over window size as a starting value for (6.32).

To demonstrate how this works, recall that the SWDFT with length n windows is given by

$$\begin{aligned}
a_{k,p} &= \sum_{j=0}^{n-1} x_{p-n+1+j}\omega_n^{-jk}, \\
k &= 0,1,\dots,n-1, \\
p &= n-1,n,\dots,N-1. \qquad (6.33)
\end{aligned}$$

If we maximize the (scaled) SWDFT coefficients over window size, we have

$$(\hat{k},\hat{p},\hat{n}) = \underset{k,p,n}{\arg\max} \frac{2}{n}\Big|\sum_{j=0}^{n-1} x_{p-n+j+1}\omega_n^{-jk}\Big|^2. \tag{6.34}$$

(6.34) is easily translated into estimates for f, S, and L by

$$\hat{L} = \hat{n},$$

$$\begin{aligned}\hat{S} &= \hat{p} - \hat{n} + 1,\\ \hat{f} &= \frac{\hat{k}}{\hat{n}}.\end{aligned} \tag{6.35}$$

So we can use the maximum SWDFT coefficient over window size as a starting value in a non-linear optimization of the spectrogram function (6.31), and this solution is the least squares estimate for the local cosine regression model.

The final parameter required for maximum likelihood estimation is σ. Let $\hat{g}_t$ be the fitted values for $\hat{A}, \hat{\phi}, \hat{S}, \hat{L}$, and $\hat{f}$. The estimate of σ is just the standard maximum likelihood estimate for Gaussian errors, that is

$$\frac{\partial \ell}{\partial \sigma} = \frac{-N}{\sigma} + \frac{1}{\sigma^3} \sum_{t=0}^{N-1} (x_t - \hat{g}_t)^2. \tag{6.36}$$

Setting (6.36) to 0 and solving for σ gives

$$\hat{\sigma} = \sqrt{\frac{1}{N} \sum_{t=0}^{N-1} (x_t - g_t)^2}. \tag{6.37}$$

And that's it. To summarize, the main idea is that the maximum likelihood estimate for f, S, and L in local cosine regression is approximately equivalent to maximizing the spectrogram function over frequency, window position, and window size.

6.4 Implementation

Because neither the likelihood function nor the spectrogram function is convex, the implementation of local cosine regression is crucial (Rice and Rosenblatt (1988)). For example, applying Newton's method is not guaranteed to work well (Quinn et al. (2008)). Our implementation of the maximum likelihood estimator is given in Algorithm 3.

Algorithm 3: Fitting Local Cosine Regression

```
input  : x: length $N$ signal
         l: minimum length of the signal (parameter L in the model)
         pwidth: range of window positions around the maximum to search
         kwidth: range of frequencies around the maximum to search
for n in l:N do
    $(\hat{k}, \hat{p}) = \arg\max_{k,p} |\sum_{j=0}^{n-1} x_{p-n+1+j}\omega_n^{-jk}|^2$
    krange = $\frac{\hat{k} - \text{kwidth}}{n} : \frac{\hat{k} + \text{kwidth}}{n}$
    prange = $\hat{p} - \text{pwidth} : \hat{p} + \text{pwidth}$
    for p in prange do
        $(\hat{S}, \hat{L}) = \text{get\_SL}(n, p)$
        $(\hat{S}, \hat{L}, \hat{f}, \hat{A}, \hat{\phi}, \hat{\sigma}) = \arg\max_{f \in \text{krange}} \ell_{S,L,k}(A, \phi, \sigma)$
    end
end
output: $\theta = (\hat{S}, \hat{L}, \hat{f}, \hat{A}, \hat{\phi}, \hat{\sigma})$ that maximizes the likelihood
```

An example is shown in Figure 6.3. This figure displays a signal generated from the local cosine regression model. The top panel shows that the fitted values from our model are reasonable. The middle panel shows the SWDFT for the true window size 78, which we use to find starting values. The SWDFT clearly shows large spectral power around the true frequency and window position. Finally, the bottom panel compares the log likelihood of the signal against window position, which shows that the maximum likelihood estimate for L is close to the true window size.

How did we address the non-convexity of the likelihood function? Our solution restricts the range of frequencies and window positions to search around a neighborhood of the maximum SWDFT coefficient. Specifically, we use a locally biased version of the Dividing Rectangles (DIRECT) algorithm (Jones et al. (1993); Gablonsky and Kelley (2001)), implemented in the `nloptr` `R`-package (Ypma (2014)), which is a wrapper for the *non-linear optimization* (NLopt) library (Johnson (2014)). Through many simulations and experiments, I found that this approach works well. The success is partly due to restricting the frequency range (specified by the **krange** parameter in Algorithm 3) to $\frac{1}{2}$, using theoretical arguments made in Section 2 of Quinn et al. (2008). The idea is that if we maximize the spectrogram function at the Fourier frequencies

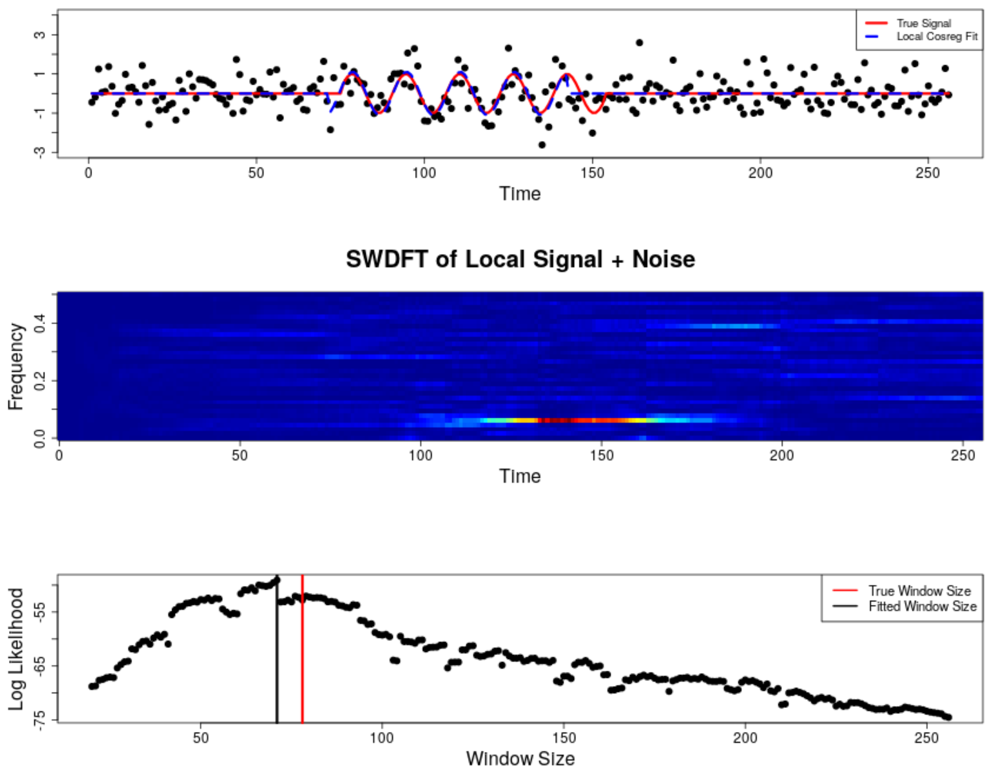

Figure 6.3: Top: The black dots are a realization of the Local Cosine Regression model with parameters $N = 256, L = 78, S = 75, f = .0625, A = 1, \phi = 1$, and $\sigma = .8$. The red line is the signal component of the model, without adding noise. The blue line is the fitted values after running the local cosine regression estimation procedure. Middle: The SWDFT of the realization of the local cosine regression model from the top panel. Bottom: Log likelihood of the estimation procedure for each window size. We see that the maximum likelihood estimate is close to the true window size.

(which is what the SWDFT does), we are asymptotically guaranteed to be less than one Fourier frequency away from the true maximum. An illustration of this concept is shown in Figure 6.4.

You may wonder why Algorithm 3 maximizes the likelihood function instead of the spectrogram function. We choose this approach for two reasons. First, because the asymptotic approximations from the derivation section aren't exact, this leads to small biases in frequency estimation. Second, both criteria can be evaluated in constant time, so we don't lose computational speed by maximizing the likelihood function directly.

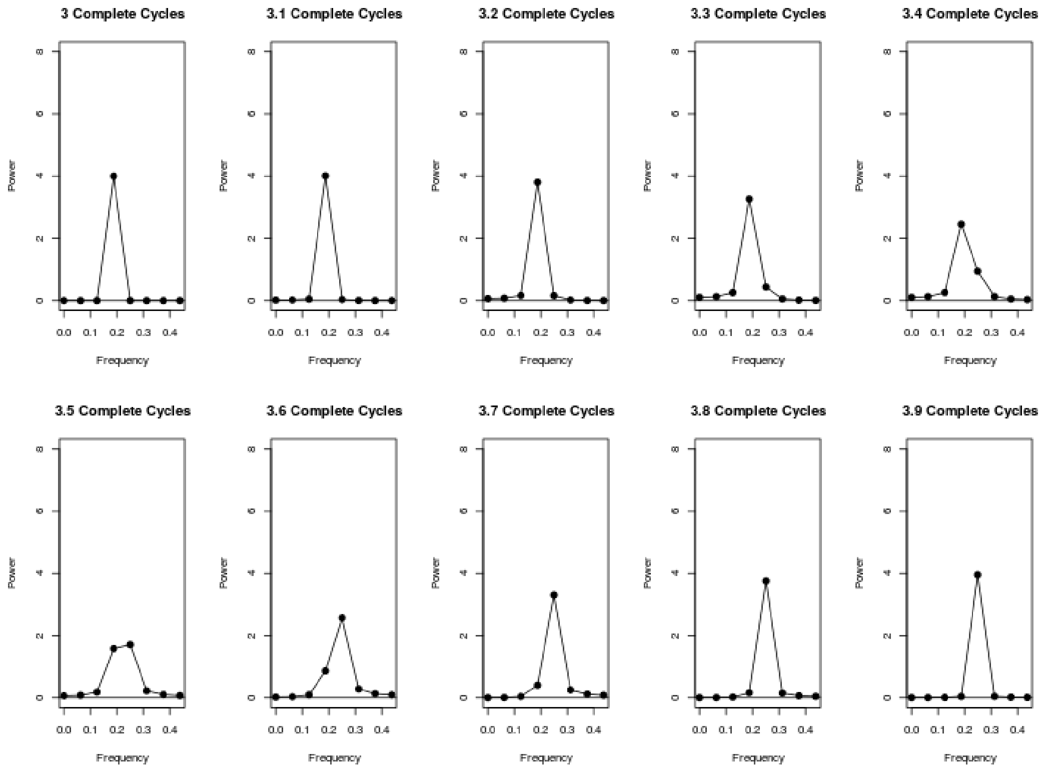

Figure 6.4: The periodogram of the signal evaluated at the Fourier frequencies for the signal $y_t = \cos(\frac{2\pi F t}{16})$ where $F = 3, 3.1, 3.2, \ldots, 3.9$. Illustrates that taking the maximum of the Fourier frequencies will be *close* to the true frequency, and that searching for the true frequency around the adjacent frequencies is sufficient.

6.5 Summary

This chapter introduces local cosine regression, which is a localized version of cosine regression. Just as the DFT is used as an estimator for cosine regression, the SWDFT can be used as an estimator for local cosine regression. In particular, we show that the maximum SWDFT coefficient over all possible window sizes is the approximate maximum likelihood estimator for the frequency, start, and length parameters in the local cosine regression model.

Chapter 7

Complex Demodulation and the SWDFT

Many signals have oscillations that aren't perfect sinusoids. The peak of each cycle may vary, the length of each cycle may vary, and so on. From a statistical perspective, imperfect sinusoids mean that we need flexible models to capture these oscillations, and one popular approach models oscillations with time-varying parameters. For example, a time-varying amplitude parameter would account the peak of each cycle varying, and a time-varying phase parameter would account for the length of each cycle varying. A flexible method to estimate these time-varying parameters is called *complex demodulation*.

To demonstrate when complex demodulation is useful, consider the time-series of the annual sunspot numbers shown in Figure 7.1. In this figure, observe that the annual number of sunspots goes up and down, but both the peaks and lengths of each cycle change over time. We applied cosine regression to this data, and the fitted values are shown in blue. The fitted values show that cosine regression isn't an accurate model. In contrast, the red line shows the fitted values from a model fit by complex demodulation, which substantially improves the fit.

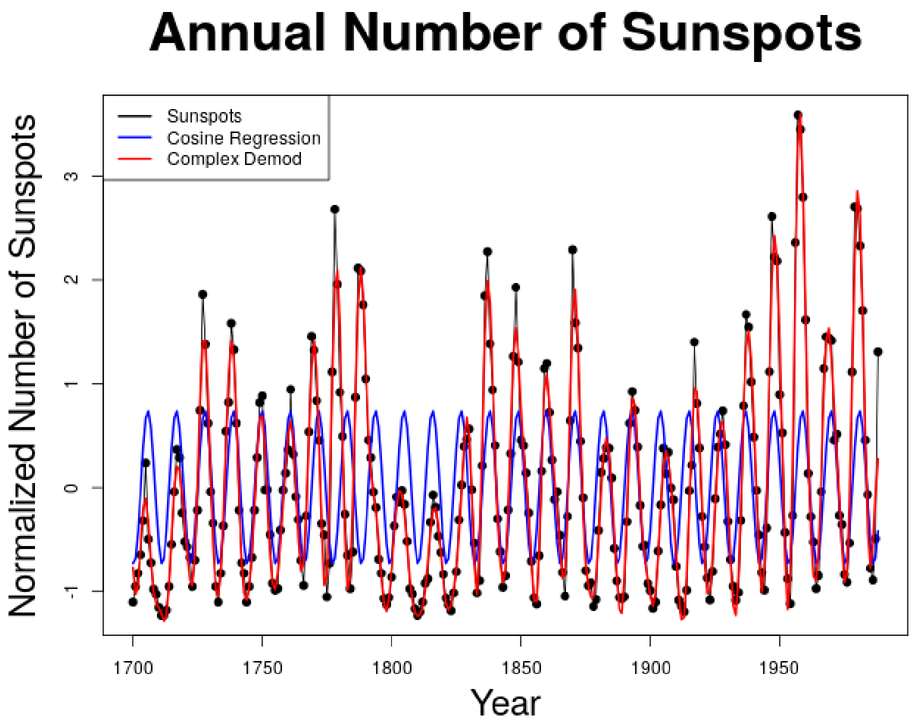

Figure 7.1: The annual number of sunspots from 1700-1988 observed at the Swiss Federal Observatory. The blue line shows the fit from cosine regression at frequency $f = \frac{1}{11}$, and the red line shows the fit from complex demodulation at the same frequency, using a Butterworth filter.

We make heavy use of complex demodulation in Chapter 8, and it's a fairly complicated method, so we dedicate this chapter to explaining how complex demodulation works. We start by deriving the method, both mathematically and with simple illustrations. Then, we focus on how to use complex demodulation, and in particular, the choices a practitioner must make when applying the method.

7.1 Complex Demodulation

Complex demodulation extracts a time-varying amplitude and a time-varying phase from a signal. From a statistical perspective, complex demodulation can be seen as an estimator for A_t and ϕ_t in the following statistical model

$$\begin{aligned} x_t &= A_t \cos(\frac{2\pi F t}{N} + \phi_t) + \epsilon_t, \\ t &= 0, 1, \ldots, N-1, \end{aligned} \tag{7.1}$$

where ϵ_t is assumed to be either independent or stationary.

The main difference between (7.1) and the cosine regression model (6.1) is that we replace A with A_t, and ϕ with ϕ_t. To be consistent with Chapter 6, let $f = \frac{2\pi F}{N}$ for the rest of this section.

If we assume the frequency parameter (f) in (7.1) is known, which we denote as f_0, then complex demodulation is implemented in three steps:

- **Step One**: Demodulate the original series $y_t = x_t \cdot e^{-2\pi i f_0 t}$.
- **Step Two**: Smooth the demodulated series ($z_t = \mathbf{smooth}(y_t)$), typically using a low pass filter.
- **Step Three**: Extract A_t, ϕ_t from smoothed series (z_t).

It's not obvious why this works, so we give a derivation that closely follows both Chapter 7 of Bloomfield (2004) and Hasan (1983). Our derivation starts by plugging Euler's inverse relation (C.2)

$$\cos(x) = \frac{1}{2}(e^{ix} + e^{-ix}), \tag{7.2}$$

into (7.1)

$$x_t = \frac{1}{2} A_t e^{2\pi i (f_0 t + \phi_t)} + \frac{1}{2} A_t e^{-2\pi i (f_0 t + \phi_t)} + \epsilon_t. \tag{7.3}$$

Applying step one of complex demodulation to (7.3) gives

$$\begin{aligned}
y_t &= x_t \cdot e^{-2\pi i f_0 t}, \\
&= \frac{1}{2}A_t e^{2\pi i(f_0 t+\phi_t)} \cdot e^{-2\pi i f_0 t} + \frac{1}{2}A_t e^{-2\pi i(f_0 t+\phi_t)} \cdot e^{-2\pi i f_0 t} + \epsilon_t \cdot e^{-2\pi i f_0 t}, \\
&= \frac{1}{2}A_t e^{2\pi i\phi_t} + \frac{1}{2}A_t e^{-2\pi i(2f_0 t+\phi_t)} + \epsilon_t e^{-2\pi i f_0 t}. \qquad (7.4)
\end{aligned}$$

(7.4) shows that step one of complex demodulation decomposes the signal into three components. We can extract the time-varying amplitude (A_t) and phase (ϕ_t) from the first component, which is $\frac{1}{2}A_t e^{2\pi i\phi_t}$. To demonstrate how this extraction works, let $c_t = \frac{1}{2}A_t e^{2\pi i\phi_t}$ be the first component of (7.4), then we have

$$\begin{aligned}
|c_t| &= |\frac{1}{2}A_t e^{2\pi i\phi_t}|, \\
&= |\frac{1}{2}||A_t||e^{2\pi i\phi_t}|, \\
&= \frac{1}{2}A_t, \qquad (7.5)
\end{aligned}$$

where the second to third line follows since $|e^{2\pi i\phi_t}| = 1$. (7.5) implies that $A_t = 2 \cdot |c_t|$, and since c_t may be viewed as a complex number with phase $2\pi\phi_t$, we extract the phase with $\phi_t = \text{Arg}(c_t)$. Thus, if we could extract first component from (7.4), which is c_t in our notation, then we could obtain the time-varying amplitude and phase with the following equations

$$\begin{aligned}
A_t &= 2 \cdot |c_t| \\
\phi_t &= \text{Arg}(c_t). \qquad (7.6)
\end{aligned}$$

The crux of the problem, then, is obtaining the first component (c_t) from the decomposition (7.4). The standard approach, which is step two of complex demodulation, applies a smoother to the signal obtained

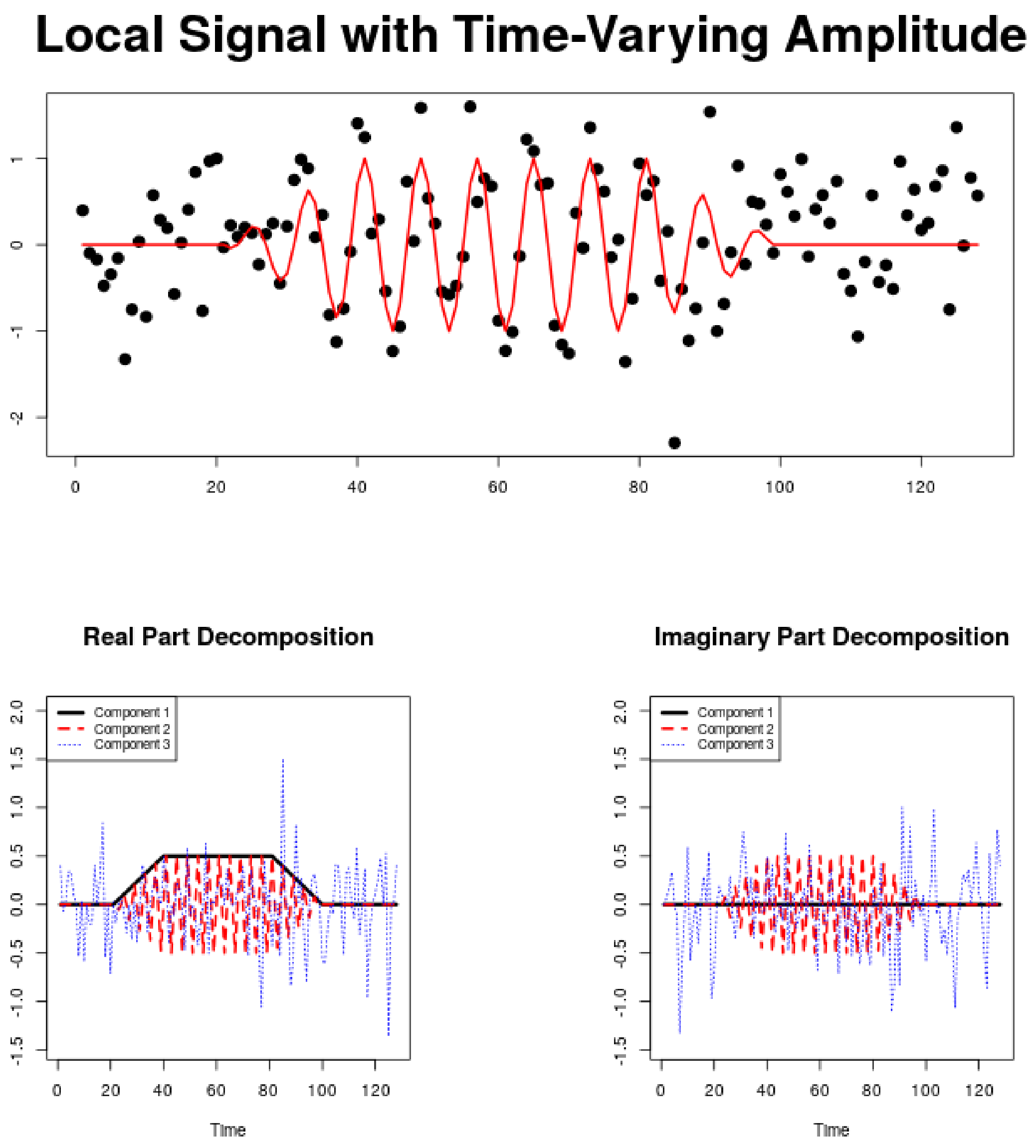

Figure 7.2: Step one of complex demodulation decomposed into three components, given by (7.4). The top panel shows the signal $x_t = A_t \cos(\frac{2\pi 16}{128}) + \epsilon_t$, where $\epsilon_t \sim N(0, .5)$ and A_t is the black line shown in the bottom left panel. The bottom two panels show the decomposition of the signal plus noise into the three components of (7.4).

by step one: $z_t = \textbf{smooth}(y_t)$. The smoothed signal (z_t) is then used as an estimate for the first component of the (7.4), which is the quantity that we are interested in. Why does this work? The argument, given in Section 7.1 of Bloomfield (2004)), is that the first component of (7.4) is the only smooth part of the decomposition, since the second component oscillates around frequency $-2f_0$, and the third component is noise. A visual demonstration of this argument is presented in Figure 7.2.

7.2 How to Use Complex Demodulation

To use complex demodulation, practitioners need to make two choices:

- **Step One**: Which frequency (f_0) to use for demodulation.
- **Step Two**: The type of smoother to use non the demodulated signal.

Section 7.2.1 covers step one, and Section 7.2.2 covers step two.

7.2.1 Frequency Selection with the SWDFT

Just like cosine regression, complex demodulation requires users to specify the frequency they want to demodulate, which we denote as f_0. In some applications, such as daily, weekly, or annual cycles, it's easy to select f_0. But in other applications, we don't know the frequency of interest (f_0) beforehand, and we want to select this frequency using the data.

How should we select f_0? Just as cosine regression selected the frequency of the maximum DFT coefficient, a straightforward approach for complex demodulation is selecting the frequency of the maximum SWDFT coefficient. And as it turns out, there is a mathematical connection between complex demodulation and the SWDFT, which we derive next.

Let's start with step one of complex demodulation

$$y_t \quad = \quad x_t e^{-2\pi i f_0 t}. \tag{7.7}$$

Because moving average filters are relatively simple, this derivation uses a simple moving average filter for smoothing, although the concepts carry over to more complicated filters. Let z_t be the demodulated signal with an order m simple moving average filter (assume m is odd)

$$\begin{aligned} z_t &= \frac{1}{m} \sum_{j=t-l}^{t+l} x_t e^{-2\pi i f_0 t}, \\ l &= \frac{m-1}{2}. \end{aligned} \tag{7.8}$$

Recall the definition of the SWDFT normalized by $\frac{1}{n}$ is

$$a_{k,p} = \frac{1}{n} \sum_{j=0}^{n-1} x_{p-n+1+j} e^{-2\pi i \frac{k}{n} j}, \tag{7.9}$$

which shows that (7.8) and (7.9) are nearly identical. In fact, if we select the order of the moving average filter (m) to equal the window size (n) of the SWDFT, and if we set the frequency (f_0) equal to $\frac{k}{n}$, then (7.8) becomes

$$z_t = \frac{1}{n} \sum_{j=t-\frac{n-1}{2}}^{t+\frac{n-1}{2}} x_t e^{-2\pi i \frac{k}{n} j}. \tag{7.10}$$

The only difference between (7.9) and (7.10) is that the index j in (7.9) always starts at 0, and the index j in (7.10) always starts at $t - \frac{n-1}{2}$. To set these equal, we can multiply z_t by a shift factor

$$a_{k,t+l} = e^{2\pi i (\frac{k}{n}(t-l))} z_t. \tag{7.11}$$

(7.11) shows that the t^{th} value of the demodulated series at frequency $\frac{k}{n}$, using an order n filter, equals the SWDFT with length n windows at frequency $\frac{k}{n}$ at window position $p = t + l$.

From a practical perspective, (7.11) implies that the amplitude of a demodulated signal is proportional to the SWDFT at the same frequency. Thus, it makes sense to demodulate at the frequency of the maximum SWDFT coefficient.

For an alternative perspective, Section 3.3 of Hasan (1983) gives another derivation of the connection between complex demodulation and spectrum estimation. Specifically, Hasan defines a *running periodogram*, then shows that this running periodogram is an approximately unbiased estimate of the spectral density at the demodulated frequency (f_0), when a simple moving average filter is used.

7.2.2 Filters for Complex Demodulation

Step two of complex demodulation involves smoothing, also known as filtering. The most common smoother is a simple moving average (Section 2.6 of Hasan (1983)). It is also common to apply a second moving average filter, since this removes peaks at nearby frequencies. Hasan (1983) proposes the following rule of thumb for choosing the order of moving average filters

If we wish to demodulate the frequency band $w_0 \pm \delta$, the length of the moving average should be such that $\frac{2\pi}{2L-1} - \delta$ is small. Since shifting the frequency of interest w_0 down to zero has caused what was at the zero-frequency point to be shifted to w_0, the authors also suggest that the second moving average (of length $2M+1$) be chosen such that $w_0 - \frac{2\pi}{2M+1}$ is as small as possible.

In this quote, w_0 corresponds to f_0 in our notation, L is the order of the first moving average, and M is the order of the second moving average. Bloomfield (2004) discusses several filters that aren't moving averages, as does Childers and Pao (1972) and Bolt and Brillinger (1979).

In signal processing, demodulation refers to extraction of information from a modulated *carrier wave*, a concept having its origins in AM and FM radio; AM refers to *amplitude modulation*, and FM refers to *frequency modulation*. Generally speaking, this tells us that modulation and demodulation are much more important in signal processing than statistics, so it makes sense to understand the type of filters used by this community. In signal processing, the standard approach to smoothing uses a low-pass filter, and in particular, the default filter in Matlab's `demod` function is a Butterworth filter (which is a certain type of low-pass filter).

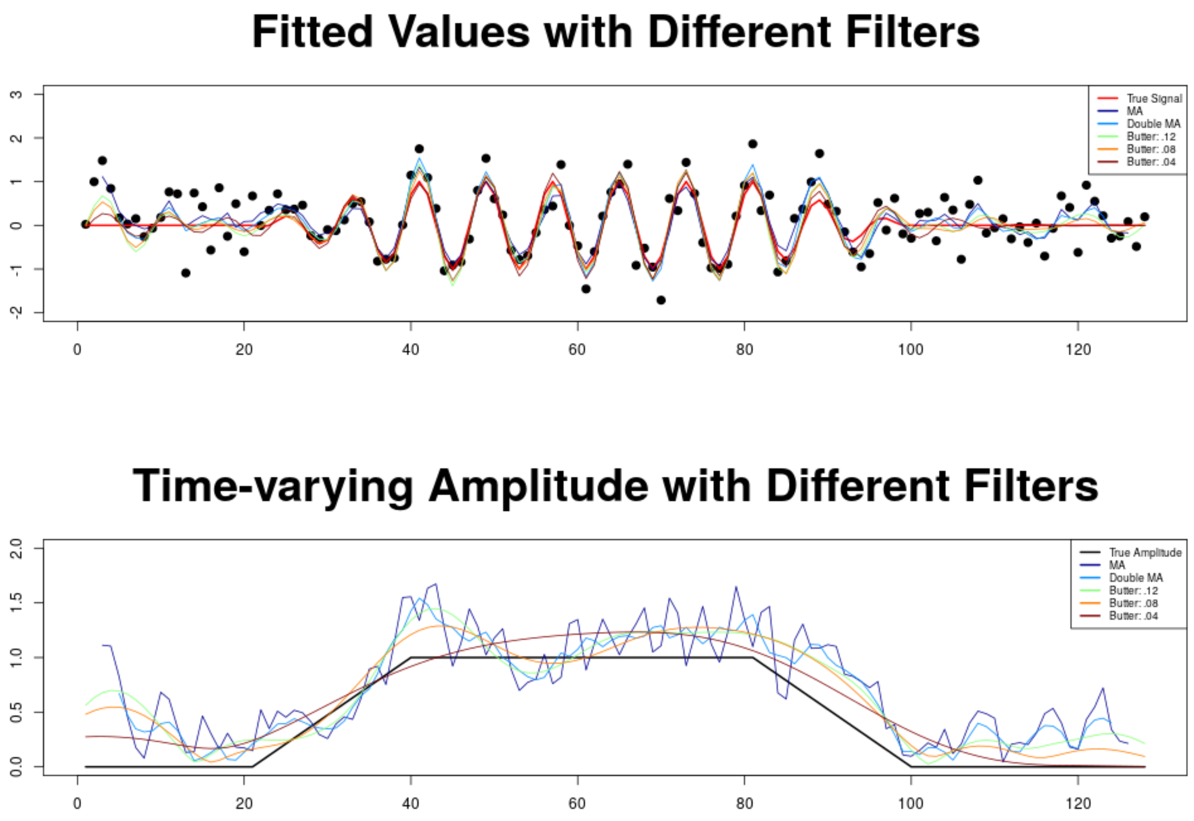

Figure 7.3: A comparison of different filters used for complex demodulation. The five filters used are a a simple moving average, a double simple moving average, and Butterworth filters with pass frequencies .12, .08, and .04. The top panel shows the fitted values for the true signal and five different filters, and the bottom panel shows the true and amplitude and the estimated amplitudes.

A comparison of different filters is shown in Figure 7.3. This figure shows that the moving-average filters are somewhat rough, and the Butterworth filters provide a smoother fit. While the fitted values don't vary much in the center of the signal, which is where the oscillation occurs, we see that filter choice matters at the ends of the signal. In particular, as the pass frequency of the Butterworth filter gets smaller, both the fitted values and time-varying amplitude become smoother. In this way, we can control the time and frequency resolution by changing the pass frequency of a low-pass filter.

7.3 Summary

Complex demodulation can be used to estimate the time-varying amplitude and phase of a signal. To use complex demodulation, practitioners must make two decisions: the type of smoother, and the frequency to demodulate. To study the choice of smoother, we compared several different filters, including a moving average, double moving average, and Butterworth low-pass filter with several different pass frequencies. Our comparison showed that Butterworth filters provide smoother estimates, and that the most important consideration is the choice of pass frequency. In terms of frequency selection, we recommended selecting the frequency of the maximum SWDFT coefficient. We justified that by showing the mathematical connection between SWDFT and complex demodulation, which demonstrated that the time-varying amplitude parameter is proportional to the modulus of the SWDFT.

Chapter 8

Matching Demodulation for Burst Detection

Imagine you're a statistician modeling the weather, and you notice two trends: the earth completes a rotation every 24 hours, and the earth orbits around the sun once a year. Both trends have have a profound effect on the weather, and you want to incorporate both of these trends into your model. In other words, you want a model that can handle multiple oscillations.

So far in this thesis, our statistical models have only included single oscillations, meaning that you would need to select between the earth's daily rotation, and the earth's annual orbit. But in this chapter, we analyze a neuroscience dataset with multiple oscillations, and we adjust our methodology accordingly. We introduce a new algorithm, called *matching demodulation*, to handle an arbitrary number of oscillations, and we demonstrate the utility of this algorithm on our neuroscience dataset.

8.1 Background

In recent years, neuroscience debates have emerged on whether oscillations occur in short *bursts*, or if the oscillations persist for longer time periods. A prominent instance of this debate occurs in the field of working memory, where the scientific understanding depends on how long oscillations last. This section introduces the background of this debate, and the statistical methods used to address it.

The primary paper we build on is Lundqvist et al. (2016). In this paper, the authors observe that oscillations occur in short *bursts*, which contradicts classic theory, which says that oscillations sustain for longer time periods. The major empirical discovery was that, if one averages spectrograms of individual trials, the averaged spectrogram shows sustained oscillations. But when Lundqvist et al. (2016) looked at individual trials, instead of averaging, he noticed that oscillations occur in short bursts. This observation led to a new theory of working memory that accounts for the bursting nature of oscillations.

In just two years, Lundqvist et al. (2016) has been cited over 150 times, and the paper has generated substantial scientific discussion. Lundqvist et al. (2018b) further developed the analysis, using similar techniques to explore the dynamics of volitional control. Lundqvist et al. (2018a) and Constantinidis et al. (2018) co-wrote a *Dual Perspectives* piece for the *Journal of Neuroscience*, in which they debate the nature of the oscillations. Finally, Miller et al. (2018) synthesizes the observations from Lundqvist et al. (2016) into the broader research paradigm surrounding working memory, and proposes a new model that incorporates these recent developments.

From a statistical perspective, an interesting aspect of the debate is the methodology used to characterize the bursts. The accuracy of this methodology is crucial, since much of the scientific debate hinges on the nature of the oscillations. Given this, we summarize the literature on burst characterization methods in the next few paragraphs.

The burst extraction method used by Lundqvist et al. (2016) is intuitive. The method says that a burst occurs when the power on a frequency band exceeds a threshold, and this threshold is set empirically, using previous trials. The detailed steps are:

1. Compute a Spectrogram using either the *multitaper method* (Percival et al. (1993)) or a *Morlet wavelet* (Wikipedia contributors (2019)).

2. Set the frequency band threshold as two standard deviations above the mean over a 10-trial reference period.

3. Fit a two-dimensional Gaussian distribution to the time-frequency neighborhood where the power of the frequency band exceeds the threshold.

4. Classify the burst frequency as the peak of the Gaussian fit, and the burst length as the time interval where *band average power has higher than half the local maximum*.

Other researchers have developed different burst extraction methods. A good overview of previous work comes from Chandran KS et al. (2017), who covers several neuroscience studies with bursts. Chandran KS et al. (2017) summarizes the structure of all previous burst extraction methods as

first computing the phase and/or power of the rhythm as a function of time, using a spectral estimator, finding 'seeds' of bursts when the power exceeded a predetermined threshold, and finally measuring the time around the seed for which the phase progression remained constant or power remained high.

Chandran KS et al. (2017) then compares several burst extraction methods, and argues that the best method is *matching pursuit* (KS et al. (2016)). These author's prefer matching pursuit because other burst extraction methods rely on spectral estimation, and spectral estimation has high variance. The authors argue that this variance leads to the appearance of bursts, which can instead be attributed to noisy estimation procedure, and this would imply that the bursts have no biological significance.

But variability in spectral estimation is well studied in statistics, dating back to at least Blackman and Tukey (1958). In their discussion, Chandran KS et al. (2017) admit that the use of tapering (and smoothing) reduces both the bias and variance of spectral estimation. So, this argument for matching pursuit loses weight if standard spectral estimation techniques, such as *multitaper methods*, are used.

This point aside, Chandran KS et al. (2017) make a compelling case for matching pursuit, since matching pursuit estimates burst length without setting an arbitrary threshold. Matching pursuit is also not restricted to orthogonal bases, and can test for more shapes than standard Fourier or wavelet techniques. Overall, we agree that matching pursuit improves the analysis, and we take this into account when developing our methodology.

8.2 Data

We analyze a dataset of Local Field Potential (LFP) recordings from an experiment where a monkey performs a working memory task. The task shows a monkey an object, then after a two-second delay, presents the monkey with two objects, one of which was shown earlier in the trial. At this point, the monkey is tasked with selecting the object shown earlier in the trial. The steps of the experiment can be broken into the following sequence of events:

1. Pre-trial: $\approx$ 1300 ms

2. Monkey fixates on the middle of the screen: $\approx$ 700 ms

3. Monkey is presented a sample object: $\approx$ 500 ms

4. Delay: $\approx$ 2000 ms

5. Two objects appear, and the monkey attempts to identify the object presented in step 3: $\approx$ 100 ms

6. After trial: $\approx$ 2500 ms.

We focus on the first 4.5 seconds of each trial, since this is when the monkey is supposed to be holding information about the object in working memory, which is when bursting is hypothesized to occur.

8.3 Matching Demodulation

This section introduces an algorithm to extract bursts, which we call *matching demodulation*. Matching demodulation combines the structure of matching pursuit with the observation that bursts have a complicated structure. A few aspects of our dataset led to the algorithm, so we start by explaining these aspects.

One motivation for our method is shown in Figure 8.1, which shows one second of a trial where bursting occurs. This figure shows that the signal can be described by oscillations, but these oscillations can't be described by simple sinusoids. In particular, two characteristics of the oscillations make them difficult to model. The first is that the peak of each cycle varies. For example, the cycle peaks from -.5 to -.4 seconds

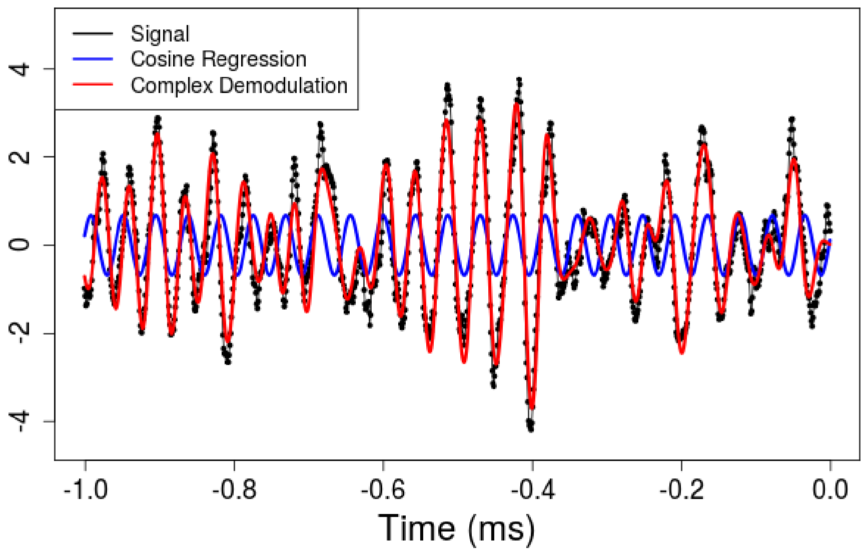

Figure 8.1: One second of trial 16 for electrode 6 in the dataset described in Section 8.2. The black dots and line show the measured LFP signal, the blue line shows the fit from cosine regression, and the red line shows the fit from complex demodulation.

are all large, and the surrounding peaks are relatively small. The second complication is that, while each cycle has a similar length, the cycle lengths are slowly drifting. These observations hint that we should model bursts with a time-varying amplitude and phase, and luckily, we can use complex demodulation from Chapter 7 to accomplish this. The blue line in Figure 8.1 shows the fitted values from cosine regression, and the red line shows the fitted values from a model fit with complex demodulation. Complex demodulation clearly improves the fit, which tells us that complex demodulation is a good option for modeling a single burst.

But as we mentioned earlier, the individual trials don't show single bursts, but multiple. This can be seen in Figure 8.2. This figure shows that bursts occur at different times and different frequencies throughout the trial, so our methodology must account for this. We also know there are hundreds of trials and electrodes in a single study, which means our methodology must be computationally efficient. To summarize, we want our methodology to be fast, work for multiple bursts, and work at multiple frequency ranges.

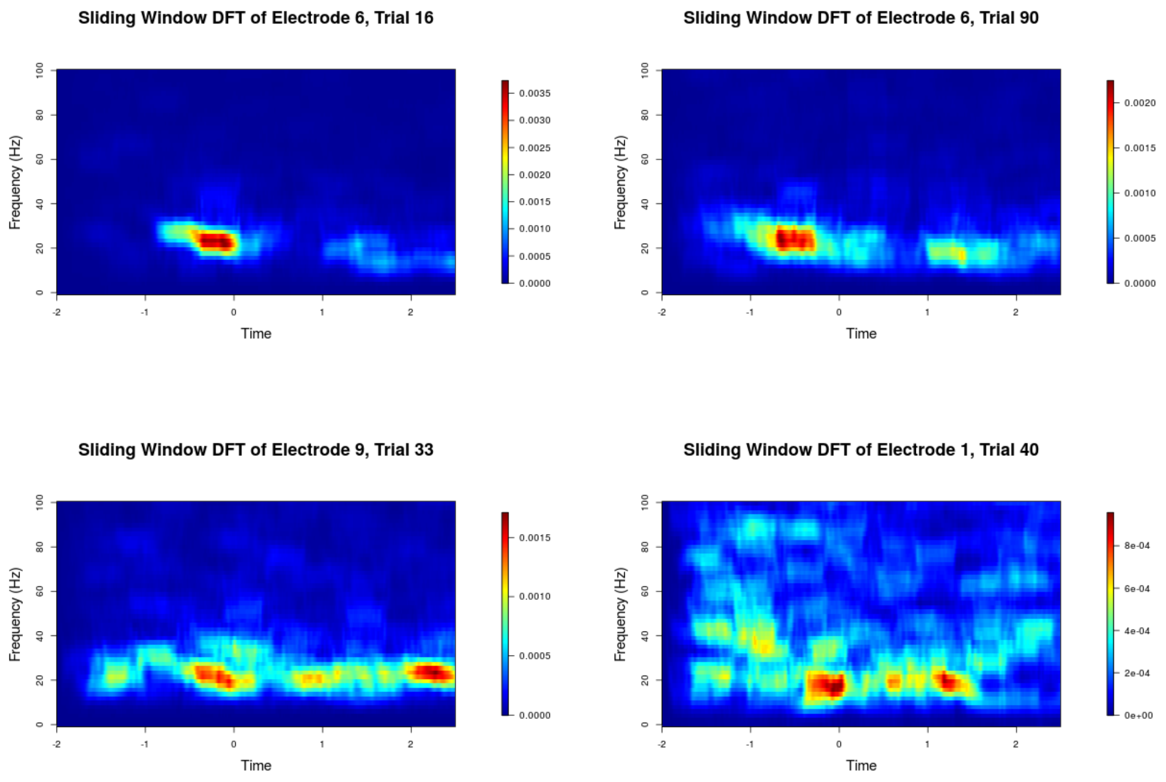

Figure 8.2: The SWDFT for four trials recorded at different electrodes in the working memory experiment. Each SWDFT is computed using a 10% cosine bell taper, and smoothed with a Daniell kernel. In order to highlight higher frequencies, we multiplied the SWDFTs by frequency, since these types of signals are well known to have a $\frac{1}{f}$ (e.g. pink noise) structure. This bias correction is common in the literature.

The matching demodulation algorithm incorporates all of these observations. After selecting the frequency of the maximum SWDFT coefficient, the algorithm applies complex demodulation at this frequency, removes the fitted values, and repeats this process on the residuals. These steps are repeated until no large SWDFT coefficients remain, which is determined by a threshold on the maximum SWDFT coefficient. The pseudo-code for matching demodulation is shown in Algorithm 4.

From a statistical perspective, matching demodulation can be seen as an estimator for the following statistical model

Algorithm 4: Matching Demodulation

```
input  : $x$: length $N$ signal, $n$: window size, $t$: threshold, $o$: order
$\hat{f} = [0, 0, \ldots, 0]$
while $\max(|a_{k,p}|^2) < t$ do
    $a = \text{swdft}(x - \hat{f}, n)$
    $\hat{k} = \max_k(|a_{k,p}|^2)$
    $y = \text{demod}(x, \hat{k}, o)$
    $\hat{f} = \hat{f} + y$
end
output: $\hat{f}$, the fitted signal
```

$$\begin{aligned} y_t &= \sum_{r=1}^{R} A_{t,r}\cos(2\pi f_r t + \phi_{t,r}) + \epsilon_t \\ t &= 0, 1, \ldots, N-1. \end{aligned} \tag{8.1}$$

(8.1) is simply the sum of R cosine functions with time-varying amplitudes and phases. In other words, it's the complex demodulation model (7.1) extended to R total signals.

The name *matching demodulation* comes from the *matching pursuit* algorithm introduced by Mallat and Zhang (1993). Both algorithms are greedy, both algorithms iteratively operate on the residuals, and both algorithms continue until a stopping criteria is reached. The main difference is that matching pursuit selects the next signal as the *atom* in a *redundant dictionary* that has the largest inner product with the signal. Here, atom is a signal processing term for basis function, and redundant dictionary means a non-orthogonal set of basis functions. In contrast, matching demodulation uses complex demodulation instead of selecting the atom with the largest inner product. One reason for this is that the time-varying amplitudes and phases are difficult to parameterize, so it's better to learn their shapes from the data, which is conceptually similar to the idea of dictionary learning (Tosic and Frossard (2011)). Another advantage is that we can interpret matching demodulation from an oscillation centric perspective, using standard terms like frequency, amplitude, and phase for the parameters.

Finally, matching demodulation is fast: each iteration is a dot-product followed by a low pass filter, which means it's an $O(N)$ algorithm.

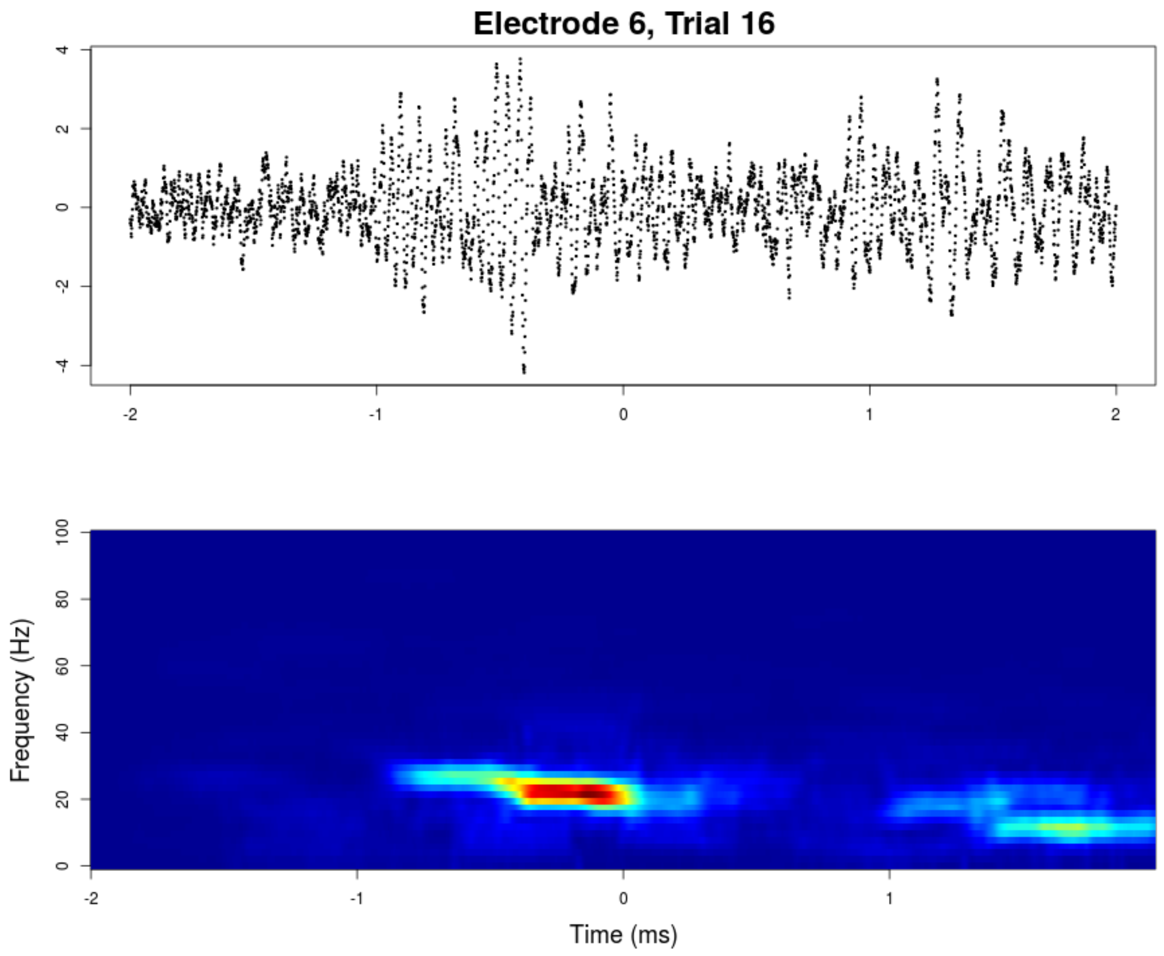

Figure 8.3: The top panel shows the signal recorded on trial 16 in electrode 6. The red line shows the fitted values from matching demodulation, and the bottom panel shows the smoothed and tapered SWDFT of of the signal in the top panel.

8.4 An Example

The best way to understand matching demodulation is through an example. Thus, this section walks through an example of matching demodulation applied to an individual trial in our dataset, and the individual trial we use is shown in Figure 8.3. This figure shows that there are two *bursts* in our trial; there's a burst around 20 Hertz starting at -.5 seconds, and there's a burst around 10 Hertz starting around 1.5 seconds. We want matching demodulation to pick up both of these bursts.

The iterations from matching demodulation are displayed in Figure 8.4. The top panels in this figure show the signal in black, and the fitted values from complex demodulation in red. The bottom panels show

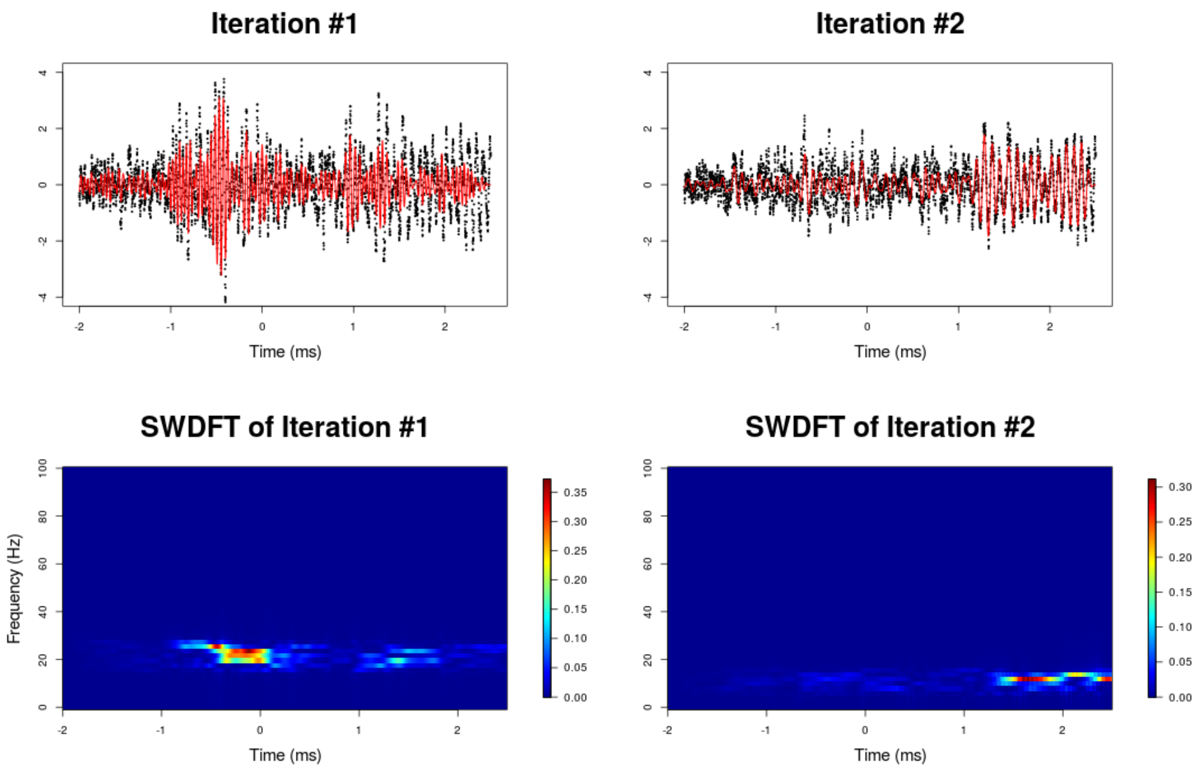

Figure 8.4: The two iterations of matching demodulation applied to the signal shown in the top panel of Figure 8.3. The top panels show the residuals of the current signal in black, and the fitted values during this iteration in red. The bottom panels show the SWDFTs of the fitted values (red lines) of the top panels. The SWDFTs show that each iteration picks up a burst.

the SWDFTs of the fitted values from the top panels, and this tells us where in the time-frequency plane each iteration picks up. The SWDFTs tell us that the first iteration picks up the first burst, and the second iteration picks up the second burst, which is exactly what we wanted.

Another important point from Figure 8.4 is that the SWDFTs in the bottom panel only have power in *narrow* frequency bands, which is why complex demodulation is often referred to as a narrow-band filter. The width of the narrow frequency band depends on the low-pass filter, and in particular, the pass frequency used by the low-pass filter: The lower the pass frequency, the narrower the frequency band.

The fitted values from matching demodulation are obtained by summing the fitted values in each iteration, and the overall fitted values are shown in Figure 8.5. This figure shows that matching demodulation provides

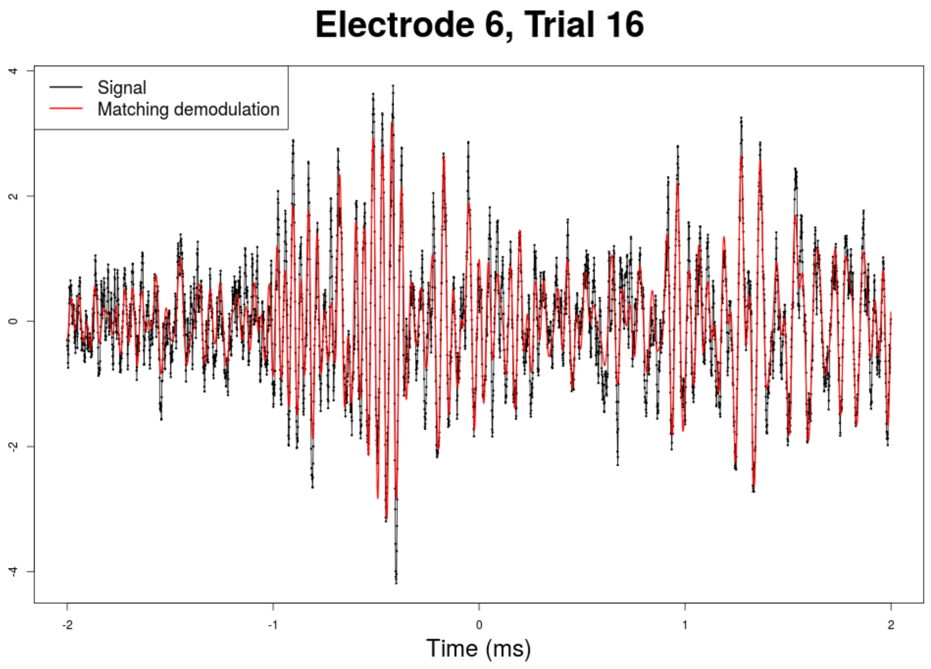

Figure 8.5: The fitted values from matching demodulation

a good fit for a fairly complicated signal. But couldn't you do this with a spline? Sure, but matching demodulation provides a straightforward interpretation of the oscillations, which we describe next.

Matching demodulation allows us to see how the time-varying amplitudes and phases change over time by simply viewing the estimated parameters from our statistical model (8.1). We show the time-varying amplitude parameters from our example in Figure 8.6. This figure shows that the amplitude of the frequency band centered around 23 Hertz has large amplitudes between both -1 to 0 seconds, and 1 to 1.5 seconds. We also see that the time-varying amplitude around 14 Hertz is large at the end of the trial. These interpretations can help us characterize when a burst occurs, for example, when the time-varying amplitude parameter exceeds a threshold.

8.5 Summary and Discussion

This chapter introduced the matching demodulation algorithm, and showed that matching demodulation can successfully decompose a signal with non-stationary periodic components.

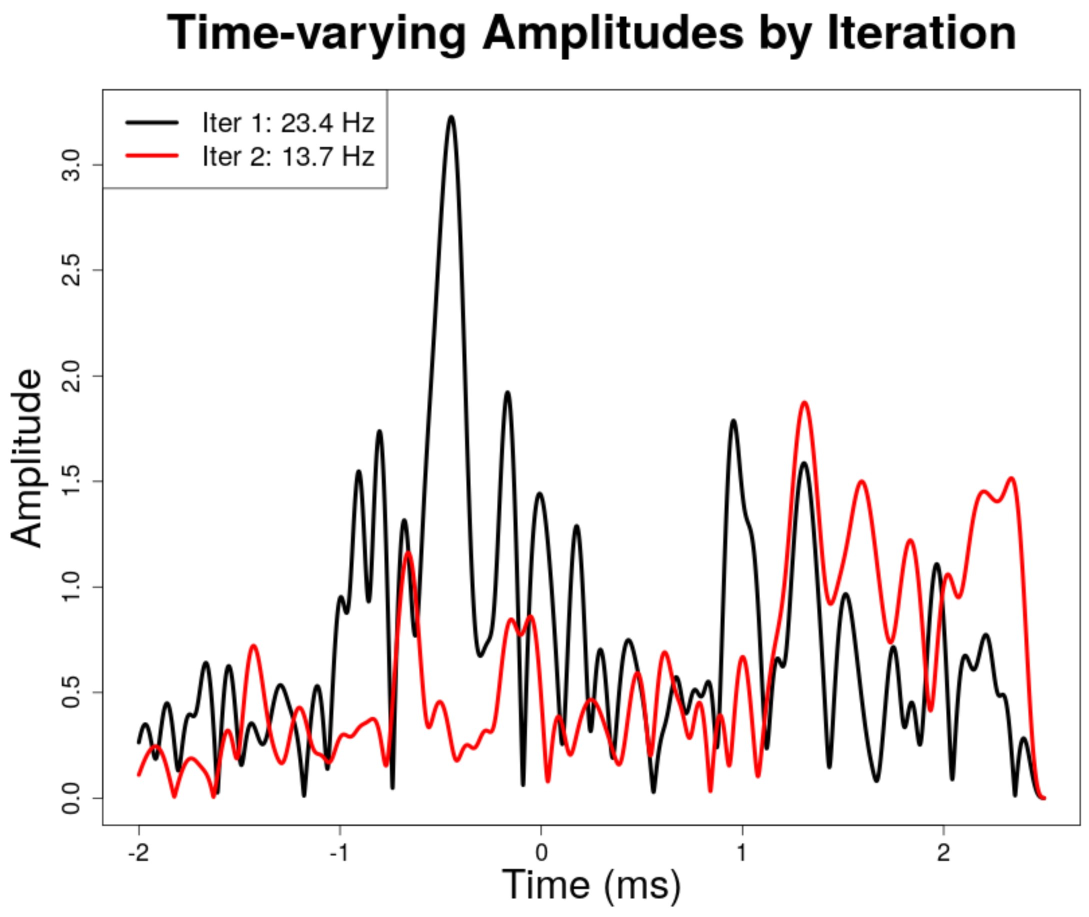

Figure 8.6: The instantaneous amplitude of each iteration of matching demodulation.

As for the working memory controversy, after analyzing many individual trials, we agree that averaging trials throws away useful information. At a minimum, our analysis shows that the dynamics of working memory are more complicated than the long, sustained oscillations assumed under the classic theory. This can be seen by simply visualizing single trials, as we show in Figure 8.2. More formally, we can fit individual trials with matching demodulation, and then use the estimated parameters to characterize bursting behavior, which would give us a similar qualitative interpretation. At the end of the day, Lundqvist et al. (2016) made a great observation on the downside of averaging trials, which we believe to be true after analyzing some of the trials ourselves.

Chapter 9

Conclusion

This thesis studies the Sliding Window Discrete Fourier Transform (SWDFT) from an algorithmic, statistical, and applied perspective. In Chapters 3 and 4, we derived the Tree SWDFT algorithm, and extended the algorithm to arbitrary dimensions. Chapter 5 derived both the asymptotic marginal distribution and covariance between SWDFT coefficients for white noise signals, which allowed us to characterize the joint distribution in two dimensions and higher. Chapter 6 localized the widely used method of cosine regression, which we call *local cosine regression*. For the local cosine regression model, we showed that the maximum SWDFT coefficient over all possible window sizes is the approximate maximum likelihood estimate of the frequency, signal length, and signal position parameters. Chapters 7 and 8 introduced the *matching demodulation* algorithm to decompose signals with non-stationary periodic components. We demonstrated the utility of matching demodulation by showing how it works on a neuroscience dataset.

9.1 Future Work

Since the SWDFT is such a general tool, there are many future research directions closely related to this thesis. This section describes what we consider to be the most promising future research directions.

9.1.1 SWDFT Algorithms

There are many ways to improve SWDFT algorithms. First, it is possible to extend the Tree SWDFT to different orthogonal transforms, such as the discrete cosine transform, the Walsh-Hadamard transform, and more. In fact, Farhang-Boroujeny (1995) gave conditions on the set of coefficients that allows the Tree SWDFT structure to be extended to the the class of *generalized transforms* (Chapter 9 of Elliott and Rao (1983)). Second, as discussed by Sherlock (1999), it is important to understand if the Tree SWDFT algorithm retains its theoretical speedup when different smoothing functions and tapers are used. Finally, a major advance in SWDFT algorithms would be to convert the tree data-structure into a multi-resolution SWDFT, which is described further in Section 9.1.4.

From a programming perspective, and interesting extension would be writing a single program that works in arbitrary dimensions, following the general formula (4.24). In this thesis, I wrote three separate programs for the 1D, 2D, and 3D Tree SWDFT algorithms. The only difference between these three programs is that the 1D algorithm used three loops, the 2D algorithm used six loops, and the 3D algorithm used nine loops. A more general program would generate the number of loops *on the fly*, alleviating the need for a separate program in each dimension.

From a software engineering perspective, a valuable contribution would be a Tree SWDFT program that works for arbitrary window sizes. This would be an improvement over our current implementation, which only works for the radix-2 case. Because the Tree SWDFT uses the Cooley-Tukey FFT algorithm, and the Cooley-Tukey algorithm can be implemented for arbitrary radices, we know that extending the algorithm to arbitrary window sizes is theoretically possible. However, the programming details are more complicated (Singleton (1966, 1969)). An easy solution to this problem would simply replace the Tree SWDFT with a sliding window FFT whenever the window size is not a power of two. This approach suffices for practical purposes, but it is not the most elegant solution.

Finally, one of the most eye-opening things I learned is that, despite the theoretical speedup the Tree SWDFT has over the sliding window FFT, I wasn't able to write a program that out performs the highly optimized *Fastest Fourier Transform in the West* (FFTW) library (Frigo and Johnson (2005)). For the Tree

SWDFT to go mainstream, it is necessary to build a highly optimized software library that focuses more on memory, parallelization, and other concepts related to software engineering. The Tree SWDFT will only become widely adopted when it becomes the fastest SWDFT *in practice*.

9.1.2 Time-Frequency Statistical Tests

An archetypal statistical question applied to the SWDFT is: how large would a SWDFT coefficient need to be to distinguish it from chance? In fact, the main reason we derived the joint distribution of SWDFT coefficients in Chapter 5 was answering this question. The first formal statistical test based on the DFT coefficients was Fisher's Periodicity test (Fisher (1929)), which has been extended many times (e.g. Siegel (1980), Chiu (1989), Juditsky et al. (2015), Cai et al. (2016)). One approach towards constructing an analogous test for the SWDFT coefficients would be either an analytic solution or approximation of the distribution of the maximum of the SWDFT, i.e.

$$\mathbb{P}(\max_{k,p} |a_{k,p}|^2 \leq u). \tag{9.1}$$

Some attempts to solve for this distribution are provided in Appendix D. In particular, Section D.2 uses extreme value theory to show that the limiting distribution of (9.1) follows a Gumbel distribution. While this thesis does not provide a complete solution, it is a promising area with many potential applications. For example, a solution would give practitioners a more principled approach to setting thresholds for large SWDFT coefficients.

An even more valuable extension would build on Chapter 7 of Okamura (2011), who observed that for SWDFT coefficients, we aren't only interested in the *size*, but also *how long* a SWDFT coefficient remains large across time. Specifically, we may be interested in calculating the probability a particular sequence of SWDFT coefficient exceeds a threshold for R consecutive window positions, i.e.

$$\mathbb{P}(|a_{k,p}|^2 > u, |a_{k,p+1}|^2 > u, \ldots |a_{k,p+R-1}|^2 > u). \tag{9.2}$$

Intuitively, the larger we make R, the less likely it is that the large SWDFT coefficients occurred by chance.

9.1.3 Matching Demodulation and Working Memory

Chapter 8 proposed the matching demodulation algorithm to analyze a neuroscience dataset. The goal of Chapter 8 was introducing the matching demodulation algorithm, and showing that the algorithm is useful on a real-world signal. But there's more work to be done here, both on the data-analysis, and the matching demodulation algorithm.

For the neuroscience dataset, there are many opportunities for further analysis. For instance, we only used data from ten randomly selected electrodes, so an obvious first step would be obtaining data from the remaining electrodes. A next step would determine which values of the time-varying amplitude parameter correspond to a burst. The easiest approach to do this would be setting an empirical threshold based on previous trials, but a more elegant solution could use some of the ideas developed on Section 9.1.2. Once we have a good burst definition, it would be interesting to correlate bursts with behavioral variables, such as whether the monkey was successful in its task. Finally, it's important to compare matching demodulation against other burst extraction methods. For this, we could simulate bursts with known parameters, and see how well each method recovers the parameters. I am currently working with two of the authors from Lundqvist et al. (2016) to produce this analysis.

For matching demodulation, we demonstrated that the method works well for signals with non-stationary oscillations. But there are many ways to improve the algorithm. First and foremost, matching demodulation is a greedy algorithm, so there may be downsides to working with the residuals directly. However, since each demodulated signal is shifted to zero, then goes through a low-pass filter, the different frequency bands should be approximately independent, which implies that the greedy nature of the algorithm shouldn't have drastic consequences. A larger concern is that, in order to get the method working well, I specified both the threshold for the maximum of the SWDFT and the pass frequency used by the Butterworth filter through trial and error. Automatic selection of these parameters would greatly enhance the algorithm, although it's not immediately clear how to do this.

9.1.4 The Multi-resolution SWDFT and Local Cosine Regression

This thesis partitioned chapters into three broad categories: algorithms, statistics, and applications. So a natural question is, how are these topics related? There are some obvious connections. We need an algorithm to compute the SWDFT, and we need statistical methods for applications, but what about the connection between the SWDFT algorithms and the statistical methods?

In my opinion, the most interesting connection to explore is between the Tree SWDFT algorithm and local cosine regression. The idea is that the tree data-structure used in the Tree SWDFT algorithm can become the maximum likelihood estimator for the local cosine regression model. Let's briefly sketch this argument.

The data-structure used for the Tree SWDFT algorithm has multiple levels, and each level contains DFTs with the size determined by the level of the tree. For example, level one of the tree has length two DFTs, level two of the tree has length four DFTs, level three has length eight DFTs, and so on. So in a sense, the tree data-structure is a multi-resolution SWDFT. The problem, unfortunately, is that the DFTs at lower levels of the tree data-structure don't use adjacent points of the input signal. For example, in the first window position of a SWDFT with length four windows, level one as two length two DFTs with of inputs (x_0, x_2) and (x_1, x_3), respectively. We would prefer these DFTs to use adjacent data-points, such (x_0, x_1) and (x_2, x_3), since this would make the tree data-structure multi-resolution. Thus, a first step for this research direction would be adapting the tree data-structure to use DFTs with adjacent data points, although it isn't obvious how to do this.

Next, we saw in Chapter 6 that the maximum likelihood estimate of frequency, signal length, and signal starting position can be obtained by maximizing the spectrogram function (6.31) over window size. This implies that to calculate the MLE, we need to find the maximum SWDFT coefficient over all possible window sizes. This is where the tree data-structure comes in. If we could adapt the tree data-structure to the multi-resolution case, then the maximum likelihood estimate would be the maximum of the tree data-structure.

9.2 Discussion

The SWDFT serves the same role in time-frequency analysis that linear regression serves in regression analysis, in that there are often extensions better suited to the task at hand (e.g. wavelets), but most of these extensions rely on the fundamental ideas of the SWDFT. And just as linear regression is still widely used, many of today's applications rely on the SWDFT.

So the SWDFT is widely used, and has been around since Gabor (1946), which leads to the question: is there anything left to figure out?

To my surprise, this thesis showed that there are still fundamental properties of the SWDFT to be derived, and I was able to make algorithmic, statistical and applied contributions in this thesis. What's even more surprising is that Section 9.1 only scratched the surface on what remains to be discovered. In the coming years, I expect that the SWDFT will remain widely used, and maybe this thesis will spark investigations into unexplored areas.

Bibliography

Adak, S. (1998). Time-dependent spectral analysis of nonstationary time series. *Journal of the American Statistical Association*, 93(444):1488–1501. 47

Ahdesmaki, M., Lahdesmaki, H., and Yli-Harja, O. (2007). Robust fisher's test for periodicity detection in noisy biological time series. In *Genomic Signal Processing and Statistics, 2007. GENSIPS 2007. IEEE International Workshop on*, pages 1–4. IEEE. 49

Albrecht, S. and Cumming, I. (1999). Application of momentary Fourier transform to SAR processing. *IEE Proceedings-Radar, Sonar and Navigation*, 146(6):285–297. 16

Albrecht, S., Cumming, I., and Dudas, J. (1997). The momentary Fourier transformation derived from recursive matrix transformations. In *Digital Signal Processing Proceedings, 1997. DSP 97., 1997 13th International Conference on*, volume 1, pages 337–340. IEEE. 16

Amin, M. G. (1987). A new approach to recursive Fourier transform. *Proceedings of the IEEE*, 75(11):1537–1538. 16

Aravena, J. L. (1990). Recursive moving window dft algorithm. *IEEE Transactions on Computers*, 39(1):145–148. 16

Arratia, R., Goldstein, L., Gordon, L., et al. (1989). Two moments suffice for poisson approximations: the chen-stein method. *The Annals of Probability*, 17(1):9–25. 148

Bitmead, R. (1982). On recursive discrete Fourier transformation. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 30(2):319–322. 16

Blackman, R. B. and Tukey, J. W. (1958). The measurement of power spectra from the point of view of communications engineering—part i. *Bell System Technical Journal*, 37(1):185–282. 97

Bloomfield, P. (2004). *Fourier analysis of time series: an introduction.* John Wiley & Sons. 10, 68, 87, 89, 92

Bolt, B. A. and Brillinger, D. R. (1979). Estimation of uncertainties in eigenspectral estimates from decaying geophysical time series. *Geophysical Journal International*, 59(3):593–603. 92

Bongiovanni, G., Corsini, P., and Frosini, G. (1975). *Non-recursive real time Fourier analysis.* Consiglio Nazionale delle Ricerche. Istituto di Elaborazione della Informazione Pisa. 17

Bongiovanni, G., Corsini, P., and Frosini, G. (1976). Procedures for computing the discrete Fourier transform on staggered blocks. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 24(2):132–137. 16, 17

Brillinger, D. and Krishnaiah, P. (1983). Time series in the frequency domain. handbook of statistics 3. 4

Brillinger, D. R. (1987). Fitting cosines: some procedures and some physical examples. In *Advances in the Statistical Sciences: Applied Probability, Stochastic Processes, and Sampling Theory*, pages 75–100. Springer. 70

Brockwell, P. J. and Davis, R. A. (2013). *Time series: theory and methods.* Springer Science & Business Media. 54

Brown, E. (1990). A note on the asymptotic distribution of the parameter estimates for the harmonic regression model. *Biometrika*, 77(3):653–656. 70

Byun, K.-Y., Park, C.-S., Sun, J.-Y., and Ko, S.-J. (2016). Vector Radix 2× 2 Sliding Fast Fourier Transform. *Mathematical Problems in Engineering*, 2016. xiii, 17, 32, 35, 43

Cai, T. T., Eldar, Y. C., Li, X., et al. (2016). Global testing against sparse alternatives in time-frequency analysis. *The Annals of Statistics*, 44(4):1438–1466. 48, 109

Chandran KS, S., Seelamantula, C. S., and Ray, S. (2017). Duration analysis using matching pursuit algorithm reveals longer bouts of gamma rhythm. *Journal of neurophysiology*, 119(3):808–821. 97

Chen, T. (1998). The past, present, and future of image and multidimensional signal processing. *IEEE Signal Processing Magazine*, 15(2):21–58. 29

Childers, D. and Pao, M.-T. (1972). Complex demodulation for transient wavelet detection and extraction. *IEEE Transactions on Audio and Electroacoustics*, 20(4):295–308. 92

Chiu, S.-T. (1989). Detecting periodic components in a white gaussian time series. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 249–259. 48, 109

Cipra, B. A. (2000). The best of the 20th century: editors name top 10 algorithms. *SIAM news*, 33(4):1–2. 20

Cohen, L. (1995). *Time-frequency analysis*, volume 778. Prentice hall. 3

Coles, S., Bawa, J., Trenner, L., and Dorazio, P. (2001). *An introduction to statistical modeling of extreme values*, volume 208. Springer. 145

Constantinidis, C., Funahashi, S., Lee, D., Murray, J. D., Qi, X.-L., Wang, M., and Arnsten, A. F. (2018). Persistent spiking activity underlies working memory. *Journal of Neuroscience*, 38(32):7020–7028. 96

Cooley, J. W. and Tukey, J. W. (1965). An algorithm for the machine calculation of complex Fourier series. *Mathematics of computation*, 19(90):297–301. 15, 18, 20

Covell, M. and Richardson, J. (1991). A new, efficient structure for the short-time Fourier transform, with an application in code-division sonar imaging. In *Acoustics, Speech, and Signal Processing, 1991. ICASSP-91., 1991 International Conference on*, pages 2041–2044. IEEE. 6, 17, 23

Dahlhaus, R. et al. (1997). Fitting time series models to nonstationary processes. *The annals of Statistics*, 25(1):1–37. 47

Dahlhaus, R., Janas, D., et al. (1996). A frequency domain bootstrap for ratio statistics in time series analysis. *The Annals of Statistics*, 24(5):1934–1963. 56

Dawkins, P. (2019). Paul's online math notes: Complex numbers primer. `http://tutorial.math.lamar.edu/Extras/ComplexPrimer/ComplexNumbers.aspx`. 129

Douglas, S. and Soh, J. (1997). A numerically-stable sliding-window estimator and its application to adaptive filters. In *Signals, Systems & Computers, 1997. Conference Record of the Thirty-First Asilomar Conference on*, volume 1, pages 111–115. IEEE. 16

Duda, K. (2010). Accurate, guaranteed stable, sliding discrete Fourier transform [DSP tips & tricks]. *IEEE Signal Processing Magazine*, 27(6):124–127. 16

Duhamel, P. and Vetterli, M. (1990). Fast fourier transforms: a tutorial review and a state of the art. *Signal processing*, 19(4):259–299. 30, 31

Elliott, D. F. and Rao, K. R. (1983). *Fast transforms algorithms, analyses, applications.* Elsevier. 17, 108

Eriksson, J., Ollila, E., and Koivunen, V. (2009). Statistics for complex random variables revisited. In *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*, pages 3565–3568. IEEE. 135

Exposito, A. G. and Macias, J. (2000). Fast non-recursive computation of individual running harmonics. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 47(8):779–782. 17

Exposito, A. G. and Macias, J. A. R. (1999). Fast harmonic computation for digital relaying. *IEEE Transactions on Power Delivery*, 14(4):1263–1268. 17

Farhang, B. and Lim, Y. (1992). Comment on the computational complexity of sliding fft. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 39(12):875–876. 17, 22

Farhang-Boroujeny, B. (1995). Order of n complexity transform domain adaptive filters. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 42(7):478–480. 17, 108

Farhang-Boroujeny, B. and Gazor, S. (1994). Generalized sliding fft and its application to implementation of block lms adaptive filters. *IEEE Transactions on Signal Processing*, 42(3):532–538. 17

Feller, W. (2008). *An introduction to probability theory and its applications*, volume 2. John Wiley & Sons. 51

Fisher, R. A. (1929). Tests of significance in harmonic analysis. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 125(796):54–59. 4, 47, 48, 49, 109, 149

Frigo, M. and Johnson, S. G. (2005). The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231. Special issue on “Program Generation, Optimization, and Platform Adaptation”. 108

Gablonsky, J. M. and Kelley, C. T. (2001). A locally-biased form of the direct algorithm. *Journal of Global Optimization*, 21(1):27–37. 82

Gabor, D. (1946). Theory of communication. part 1: The analysis of information. *Electrical Engineers-Part III: Radio and Communication Engineering, Journal of the Institution of*, 93(26):429–441. vii, 2, 3, 15, 112

Gentleman, W. M. and Sande, G. (1966). Fast Fourier Transforms: for fun and profit. In *Proceedings of the November 7-10, 1966, fall joint computer conference*, pages 563–578. ACM. 22

Goldberg, B. (1980). A continuous recursive dft analyzer–the discrete coherent memory filter. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(6):760–762. 16

Good, I. J. (1958). The interaction algorithm and practical Fourier analysis. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 361–372. 18

Greenhouse, J. B., Kass, R. E., and Tsay, R. S. (1987). Fitting nonlinear models with arma errors to biological rhythm data. *Statistics in medicine*, 6(2):167–183. 70

Halberstein, J. (1966). Recursive, complex Fourier analysis for real-time applications. *Proceedings of the IEEE*, 54(6):903–903. 16

Hannan, E. (1971). Non-linear time series regression. *Journal of Applied Probability*, 8(4):767–780. 70

Hannan, E. (1974). Time series analysis. *IEEE Transactions on Automatic Control*, 19(6):706–715. 70

Hannan, E. J. (1973). The estimation of frequency. *Journal of Applied probability*, 10(3):510–519. 70

Harris, D., McClellan, J., Chan, D., and Schuessler, H. (1977). Vector radix fast Fourier transform. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'77.*, volume 2, pages 548–551. IEEE. 17, 32, 34

Hasan, T. (1983). Complex demodulation: Some theory and applications. *Handbook of Statistics*, 3:125–156. 87, 92

Hostetter, G. (1980). Recursive discrete Fourier transformation. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(2):184–190. 16

Irizarry, R. A. (1998). *Statistics and music: fitting a local harmonic model to musical sound signals.* PhD thesis, University of California, Berkeley. 70

Irizarry, R. A. (2001). Local harmonic estimation in musical sound signals. *Journal of the American Statistical Association*, 96(454):357–367. 70

Irizarry, R. A. (2002). Weighted estimation of harmonic components in a musical sound signal. *Journal of Time Series Analysis*, 23(1):29–48. 70

Jacobsen, E. and Lyons, R. (2003). The sliding dft. *IEEE Signal Processing Magazine*, 20(2):74–80. 16

Johnson, S. G. (2014). The nlopt nonlinear-optimization package. 82

Jones, D. R., Perttunen, C. D., and Stuckman, B. E. (1993). Lipschitzian optimization without the lipschitz constant. *Journal of optimization Theory and Applications*, 79(1):157–181. 82

Juditsky, A., Nemirovski, A., et al. (2015). On detecting harmonic oscillations. *Bernoulli*, 21(2):1134–1165. 48, 109

Kass, R. E., Eden, U. T., and Brown, E. N. (2014). *Analysis of neural data*, volume 491. Springer. 54, 129

Kay, S. M. (1993). Fundamentals of statistical signal processing, volume i: Estimation theory (v. 1). *PTR Prentice-Hall, Englewood Cliffs*. 70, 74

Kim, I. ("2019"). personal communication. 147

KS, S. C., Mishra, A., Shirhatti, V., and Ray, S. (2016). Comparison of matching pursuit algorithm with other signal processing techniques for computation of the time-frequency power spectrum of brain signals. *Journal of Neuroscience*, 36(12):3399–3408. 97

Kundu, D. and Nandi, S. (2012). *Statistical signal processing: frequency estimation*. Springer Science & Business Media. 71, 72, 77, 78

Lahiri, S. et al. (2003). A necessary and sufficient condition for asymptotic independence of discrete fourier transforms under short-and long-range dependence. *The Annals of Statistics*, 31(2):613–641. 54

Lapidoth, A. (2017). *A foundation in digital communication*. Cambridge University Press. 135

Leadbetter, M. R., Lindgren, G., and Rootzén, H. (2012). *Extremes and related properties of random sequences and processes*. Springer Science & Business Media. 145

Lilly, J. (1991). Efficient dft-based model reduction for continuous systems. *IEEE transactions on automatic control*, 36(10):1188–1193. 16

Lo, P.-C. and Lee, Y.-Y. (1999). Real-time implementation of the moving fft algorithm. *Signal Processing*, 79(3):251–259. 16

Lundqvist, M., Herman, P., and Miller, E. K. (2018a). Working memory: delay activity, yes! persistent activity? maybe not. *Journal of Neuroscience*, 38(32):7013–7019. 96

Lundqvist, M., Herman, P., Warden, M. R., Brincat, S. L., and Miller, E. K. (2018b). Gamma and beta bursts during working memory readout suggest roles in its volitional control. *Nature communications*, 9(1):394. 96

Lundqvist, M., Rose, J., Herman, P., Brincat, S. L., Buschman, T. J., and Miller, E. K. (2016). Gamma and beta bursts underlie working memory. *Neuron*, 90(1):152–164. 96, 105, 110

Macias, J. R. and Exposito, A. G. (1998). Efficient moving-window DFT algorithms. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 45(2):256–260. 16

Mallat, S. (1999). A wavelet tour of signal processing. *The Sparse Way.* 3

Mallat, S. and Zhang, Z. (1993). Matching pursuit with time-frequency dictionaries. Technical report, Courant Institute of Mathematical Sciences New York United States. 101

Mathieu, M., Henaff, M., and LeCun, Y. (2013). Fast training of convolutional networks through ffts. *arXiv preprint arXiv:1312.5851.* 29

Mersereau, R. and Speake, T. (1981). A unified treatment of cooley-tukey algorithms for the evaluation of the multidimensional dft. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 29(5):1011–1018. 32

Miller, E. K., Lundqvist, M., and Bastos, A. M. (2018). Working memory 2.0. *Neuron*, 100(2):463–475. 96

Montoya, D.-E. A., Macias, J. R., and Gomez-Exposito, A. (2012). Efficient computation of the short-time dft based on a modified radix-2 decimation-in-frequency algorithm. *Signal Processing*, 92(10):2525–2531. 17

Montoya, D.-E. A., Macias, J. R., and Gomez-Exposito, A. (2014). Short-time dft computation by a modified radix-4 decimation-in-frequency algorithm. *Signal Processing*, 94:81–89. 17

Nadler, B. and Kontorovich, A. (2011). Model selection for sinusoids in noise: Statistical analysis and a new penalty term. *IEEE Transactions on Signal Processing*, 59(4):1333–1345. 70

Okamura, S. (2011). *The short time Fourier transform and local signals*. PhD thesis, Carnegie Mellon University. 48, 57, 109

Park, C.-S. (2015a). 2D discrete Fourier transform on sliding windows. *IEEE Transactions on Image Processing*, 24(3):901–907. xiii, 30, 43

Park, C.-S. (2015b). Fast, Accurate, and Guaranteed Stable Sliding Discrete Fourier Transform [sp Tips&Tricks]. *IEEE Signal Processing Magazine*, 32(4):145–156. 16, 17

Park, C.-S. (2017). Guaranteed-stable sliding dft algorithm with minimal computational requirements. *IEEE Transactions on Signal Processing*, 65(20):5281–5288. 16, 142

Park, C.-S. and Ko, S.-J. (2014). The hopping discrete Fourier transform [sp tips&tricks]. *IEEE Signal Processing Magazine*, 31(2):135–139. 16

Peligrad, M., Wu, W. B., et al. (2010). Central limit theorem for fourier transforms of stationary processes. *The Annals of Probability*, 38(5):2009–2022. 54

Percival, D. B., Walden, A. T., et al. (1993). *Spectral analysis for physical applications*. cambridge university press. 96

Priestley, M. B. (1965). Evolutionary spectra and non-stationary processes. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 204–237. 47

Quinn, B. G. and Hannan, E. J. (2001). *The estimation and tracking of frequency*, volume 9. Cambridge University Press. 71

Quinn, B. G., McKilliam, R. G., and Clarkson, I. V. L. (2008). Maximizing the periodogram. In *IEEE GLOBECOM 2008-2008 IEEE Global Telecommunications Conference*, pages 1–5. IEEE. 80, 82

R Core Team (2013). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. 127

Rabiner, L., Schafer, R., and Rader, C. (1969). The chirp z-transform algorithm. *IEEE transactions on audio and electroacoustics*, 17(2):86–92. 18

Rice, J. A. and Rosenblatt, M. (1988). On frequency estimation. *Biometrika*, 75(3):477–484. 70, 74, 81

Richardson, L. F. and Eddy, W. F. (2019). Algorithm 991: The 2d tree sliding window discrete fourier transform. *ACM Transactions on Mathematical Software (TOMS)*, 45(1):12. xvi, 4, 6, 15, 17, 29, 35, 42, 43, 46

Rivard, G. (1977). Direct fast Fourier transform of bivariate functions. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 25(3):250–252. 17

Rosen, O., Wood, S., and Stoffer, D. S. (2012). Adaptspec: Adaptive spectral estimation for nonstationary time series. *Journal of the American Statistical Association*, 107(500):1575–1589. 47

Roth, J., Carriveau, A., Liu, X., and Jain, A. K. (2015). Learning-based ballistic breech face impression image matching. In *Biometrics Theory, Applications and Systems (BTAS), 2015 IEEE 7th International Conference on*, pages 1–8. IEEE. 29

Sherlock, B. and Monro, D. (1992). Moving discrete Fourier transform. In *IEE Proceedings F-Radar and Signal Processing*, volume 139, pages 279–282. IET. 16

Sherlock, B. G. (1999). Windowed discrete Fourier transform for shifting data. *Signal Processing*, 74(2):169–177. 16, 30, 108

Siegel, A. F. (1980). Testing for periodicity in a time series. *Journal of the American Statistical Association*, 75(370):345–348. 48, 109

Singleton, R. (1969). An algorithm for computing the mixed radix fast fourier transform. *IEEE Transactions on audio and electroacoustics*, 17(2):93–103. 27, 108

Singleton, R. C. (1966). An algol procedure for the fast fourier transform with arbitrary factors. Technical report, Stanford Research Inst, Menlo Park, CA. 27, 108

Sorensen, H. V. and Burrus, C. S. (1988). Efficient computation of the short-time fast Fourier transform. In *Acoustics, Speech, and Signal Processing, 1988. ICASSP-88., 1988 International Conference on*, pages 1894–1897. IEEE. 16

Tai, X. H. and Eddy, W. F. (2018). A fully automatic method for comparing cartridge case images. *Journal of forensic sciences*, 63(2):440–448. 29

Tosic, I. and Frossard, P. (2011). Dictionary learning. *IEEE Signal Processing Magazine*, 28(2):27–38. 101

Unser, M. (1983). Recursion in short-time signal analysis. *Signal Processing*, 5(3):229–240. 16, 17

van der Byl, A. and Inggs, M. R. (2016). Constraining error—A sliding discrete Fourier transform investigation. *Digital Signal Processing*, 51:54–61. 17

Van Loan, C. (1992). *Computational frameworks for the fast Fourier transform*, volume 10. Siam. 17

Vetterli, M., Kovačević, J., and Goyal, V. K. (2014). *Foundations of signal processing*. Cambridge University Press. 3

Walker, A. (1971). On the estimation of a harmonic component in a time series with stationary independent residuals. *Biometrika*, 58(1):21–36. 70

Walker, A. (1973). On the estimation of a harmonic component in a time series with stationary dependent residuals. *Advances in Applied Probability*, 5(2):217–241. 70

Wang, J. and Eddy, W. F. (2010). Prfft: A fast algorithm of sliding-window dft with parallel structure. In *Signal Processing Systems (ICSPS), 2010 2nd International Conference on*, volume 2, pages V2–355. IEEE. 17

Wang, J., Woods, B., and Eddy, W. F. (2009). Meg, rffts, and the hunt for high frequency oscillations. In *Image and Signal Processing, 2009. CISP'09. 2nd International Congress on*, pages 1–5. IEEE. 17

Wang, J.-M. and Eddy, W. F. (2012). Parallel algorithm for swfft using 3d data structure. *Circuits, Systems, and Signal Processing*, 31(2):711–726. 17, 40

Whittle, P. (1952). The simultaneous estimation of a time series harmonic components and covariance structure. *Trabajos de estadística*, 3(1-2):43–57. 4, 70

Wikipedia contributors (2019). Morlet wavelet — Wikipedia, the free encyclopedia. [Online; accessed 1-May-2019]. 96

Yao, S., Piao, A., Jiang, W., Zhao, Y., Shao, H., Liu, S., Liu, D., Li, J., Wang, T., Hu, S., et al. (2019). Stfnets: Learning sensing signals from the time-frequency perspective with short-time fourier neural networks. *arXiv preprint arXiv:1902.07849*. 29

Ypma, J. (2014). Introduction to nloptr: an r interface to nlopt. Technical report, Tech. rep. 82

Appendix

Appendix A

Software

All of the methods developed in this thesis have been implemented in the `swdft` `R`-package, which is available on CRAN (R Core Team (2013)). You can install the package with:

```
install.packages("swdft")
library(swdft)
```

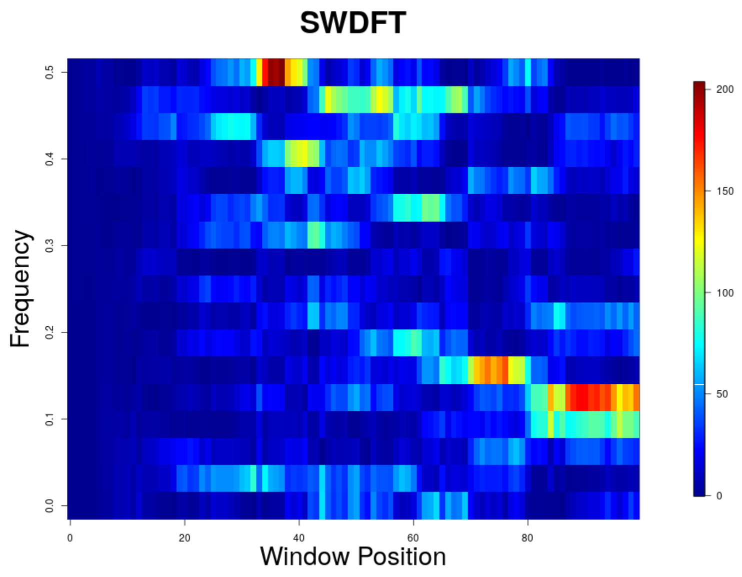

Figure A.1: Visualization of the SWDFT using the `swdft` `R`-package.

You can compute a SWDFT and visualize the output (see Figure A.1) with:

```
x <- rnorm(100)
window_size <- 32
a <- swdft(x, n=window_size)
plot(a, col="tim.colors", freq_type="fraction")
```

You can run cosine regression and local cosine regression with:

```
freq <- 3 / 32
cosreg(x=x, f=freq)
local_cosreg(x=x)
```

And you can run complex demodulation and matching demodulation with:

```
complex_demod(x=x, f0=freq)
matching_demod(x=x, n=70)
```

For a detailed introduction, see the `swdft` tutorial, available online at:

`https://cran.r-project.org/web/packages/swdft/vignettes/swdft-intro.html`

Appendix B

Complex Numbers

Since SWDFT coefficients are complex numbers, it is important to develop a basic understanding of complex numbers and their properties. This section provides a brief introduction to complex numbers, emphasizing what is most useful for this thesis. Other good introductions are given by Dawkins (2019) and Appendix A.10 of Kass et al. (2014).

B.1 Properties of Complex Numbers

We start with the definition of a complex number

Definition B.1.1 (Complex Number)**.** A complex number x is an ordered pair of real numbers $x = (x_1, x_2)$, where $x_1, x_2 \in \mathbb{R}$. The first element is the real part ($x_1 = Re(x)$), and the second element is the imaginary part ($x_2 = Im(x)$). We often write complex numbers as $x = x_1 + ix_2$, where $i = \sqrt{-1}$.

For $x = (x_1, x_2)$, $y = (y_1, y_2)$, and $z = (z_1, z_2)$, some basic properties are:

- **Equality:** $x = y$ if and only if $x_1 = y_1$ and $x_2 = y_2$,
- **Commutative:** $x + y = y + x$, $xy = yx$,
- **Associative:** $x + (y + z) = (x + y) + z$,

- **Distributive:** $x(y+z) = xy + xz$,
- **Addition:** $x + y = (x_1 + y_1, x_2 + y_2)$,
- **Multiplication:** $x \cdot y = (x_1y_1 - x_2y_2, x_1y_2 + x_2y_1)$,
- **Exponentiation:** If $x = (x_1, x_2)$: $x^n = x \cdot x \cdots x$ (n times),
- **Division:** If $x \neq (0,0)$, $\frac{y}{x} = (\frac{y_1x_1+y_2x_2}{x_1^2+x_2^2}, \frac{x_1y_2-y_1x_2}{x_1^2+x_2^2})$.

This thesis frequently uses complex addition and multiplication.

B.2 Complex Conjugate and Modulus

Two fundamental operations for complex numbers are the complex conjugate and modulus. Thus, this section defines both, lists some of their important properties. We start with the definition of the complex conjugate operation

Definition B.2.1 (Complex Conjugate)**.** The complex conjugate of a complex number $x = (x_1, x_2)$ is

$$x^* = (x_1, -x_2). \tag{B.1}$$

The complex conjugate operation has the following properties:

- $(x \pm y)^* = x^* \pm y^*$,
- $(xy)^* = x^*y^*$,
- $(\frac{x}{y})^* = \frac{x^*}{y^*}$,
- $Re(x) = \frac{x_1+x_2^*}{2}$,
- $Im(x) = \frac{x_1-x_2^*}{2i}$.

The modulus operation is defined as

Definition B.2.2 (Modulus)**.** The modulus of a complex number x is the non-negative real-valued number:

$$|x| = \sqrt{x_1^2 + x_2^2}. \tag{B.2}$$

The modulus operation has the following properties:

- $|xy| = |x| \cdot |y|$,
- $|\frac{x}{y}| = \frac{|x|}{|y|}$,
- $|x+y|^2 = |x|^2 + |y|^2 + 2(Re(x) \cdot Re(y) + Im(x) \cdot Im(y))$,
- $|x+y| \leq |x| + |y|$.

A helpful formula that connects the complex conjugate and modulus is

$$\begin{aligned} xx^* &= (x_1, x_2) \cdot (x_1, -x_2), \\ &= (x_1^2 + x_2^2, 0), \\ &= |x|^2. \end{aligned} \tag{B.3}$$

B.3 Polar and Exponential Representations

Although we typically represent a complex number x by $x = (x_1, x_2) = x_1 + ix_2$, this thesis also relies on two other representations: polar and exponential.

Polar representation of complex numbers is easy to describe using the geometric representation shown in Figure B.1. The coordinate system in this figure is known as the complex plane, and the complex number x is the point (x_1, x_2) on the complex plane. We can also represent x in polar coordinates, namely $x = (r, \theta)$, where r is the distance from the origin to (x_1, x_2), and θ is the angle of the vector from the origin to x. We can go back and forth between (x_1, x_2) and (r, θ) with the following equations

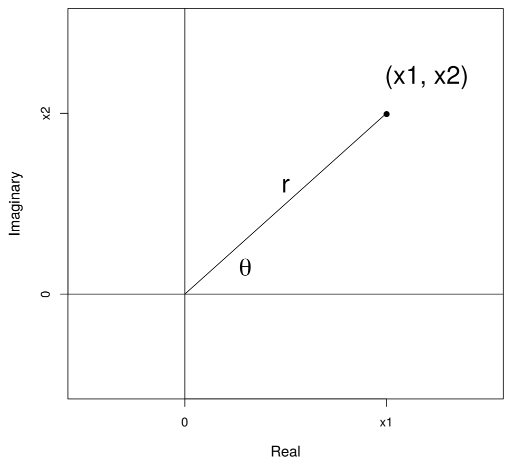

Figure B.1: Geometric representation of complex number x on the complex plane. The x-axis corresponds to the real part of the imaginary number, and the y-axis corresponds to the imaginary part.

$$\begin{aligned} x_1 &= r\cos(\theta), \\ x_2 &= r\sin(\theta), \\ r &= \sqrt{x_1^2 + x_2^2} = |x|, \\ \theta &= \arg(\frac{x_2}{x_1}). \end{aligned} \tag{B.4}$$

This leads to the following definition of polar coordinates

Definition B.3.1 (Polar Form)**.** A complex number x in polar form equals $x = r(\cos(\theta) + i\sin(\theta))$.

The third representation we use in this thesis is called exponential form, which used Euler's formula, and is defined as

Definition B.3.2 (Exponential Form)**.** A complex number x has exponential form:

$$x = re^{i(\theta+2\pi n)}, \forall n \in \mathbb{Z}. \tag{B.5}$$

Exponential form converts complex multiplication and division into addition and subtraction of exponents. For example, let $x = (x_1, x_2) = r_x e^{i\theta_x}$ and $y = (y_1, y_2) = r_y e^{i\theta_y}$, then complex multiplication and division are

$$\begin{aligned} xy &= r_x r_y e^{i(\theta_x + \theta_y)} \\ \frac{x}{y} &= \frac{r_x}{r_y} e^{i(\theta_x - \theta_y)} \end{aligned} \tag{B.6}$$

The SWDFT makes extensive use of complex numbers in exponential form.

B.4 Roots of Unity

The principal roots of unity for complex numbers are the complex coefficients used by the discrete Fourier transform (DFT). To derive them, recall that the n^{th} roots of unity for complex numbers are the unique solutions to $x^n = 1$. Since two complex numbers $x = (r_x, \theta_x)$ and $y = (r_y, \theta_y)$ are equal when $r_x = r_y$ and $\theta_x = \theta_y + 2\pi k$, $\forall k \in \mathbb{Z}$, the solution for the roots of unity is given by

$$\begin{aligned} x^n &= 1, \\ (re^{i\theta})^n &= 1e^{\theta 0}, \\ r^n e^{in\theta} &= 1e^{\theta 0}. \end{aligned} \tag{B.7}$$

(B.7) is true when $r^n = 1$ and $n\theta = 0 + 2\pi k$, $\forall k \in \mathbb{Z}$, which leads to the solutions $r = 1$ and $\theta = \frac{2\pi k}{n}$. $\forall k \in \mathbb{Z}$. The solutions are evenly spaced around the unit circle, and the unique values are known as the principal roots of unity

Definition B.4.1 (Principal Roots of Unity)**.** Let $\omega_n = e^{\frac{i2\pi}{n}}$, then the principal n^{th} roots of unity for complex numbers are

$$\begin{aligned}
\omega_n^k &= e^{\frac{i2\pi k}{n}}, \\
&= \cos(\frac{2\pi k}{n}) + i\sin(\frac{2\pi k}{n}), \\
k &= 0, 1, \ldots, n-1. \qquad \text{(B.8)}
\end{aligned}$$

ω_n has the following useful properties:

$$\begin{aligned}
|\omega_n| &= 1, \ \forall n \in \mathbb{R} \\
\omega_n^n &= 1, \\
(\omega_n^k)^n &= 1, \ \forall k \in \mathbb{Z}, \\
(\omega_n^{-m})^* &= \omega_n^m. \qquad \text{(B.9)}
\end{aligned}$$

To reiterate, the the principal n^{th} roots are important because they are the complex coefficients used by the DFT.

B.5 Complex Random Variables

In Chapter 5, we derive the statistical properties of SWDFT coefficients. And since the SWDFT coefficients are complex-valued random variables, we define the expected value, variance, and covariance of complex-valued random variables in this section. The expected value is

Definition B.5.1 (Expected Value)**.** The expected value of a complex valued random variable Z is:

$$\mathbb{E}[Z] = \mathbb{E}[Re(Z)] + i\mathbb{E}[Im(Z)]. \qquad \text{(B.10)}$$

The expected value has the following properties:

- $\mathbb{E}(Z^*) = (\mathbb{E}(Z))^*$,
- $Re(\mathbb{E}(Z)) = \mathbb{E}(Re(Z))$,
- $Im(\mathbb{E}(Z)) = \mathbb{E}(Im(Z))$.

Definition B.5.2 (Variance)**.** The variance of a complex-valued random variable is the nonnegative, real valued sum of the variances of the real and imaginary components

$$\mathbf{Var}(Z) = \mathbf{Var}(Re(Z)) + \mathbf{Var}(Im(Z)). \tag{B.11}$$

Proof. Following Eriksson et al. (2009) and Lapidoth (2017), the variance of a complex-valued random variable is

$$\mathbf{Var}(Z) \quad = \quad \mathbb{E}(|Z - \mathbb{E}(Z)|^2)$$

Using properties defined in this chapter, we can re-write the variance as:

$$\begin{aligned}
\mathbf{Var}(Z) &= \mathbb{E}(|Z - \mathbb{E}(Z)|^2), \\
&= \mathbb{E}((Z - \mathbb{E}(Z)) \cdot (Z - \mathbb{E}(Z))^*), \\
&= \mathbb{E}((Z - \mathbb{E}(Z)) \cdot (Z^* - \mathbb{E}(Z)^*)), \\
&= \mathbb{E}((Z - \mathbb{E}(Z))Z^*) - \mathbb{E}((Z - \mathbb{E}(Z))\mathbb{E}(Z^*)), \\
&= \mathbb{E}((Z - \mathbb{E}(Z))Z^*), \\
&= \mathbb{E}(ZZ^*) - \mathbb{E}(Z) \cdot \mathbb{E}(Z^*), \\
&= \mathbb{E}(|Z|^2) - |\mathbb{E}(Z)|^2, \\
&= \mathbb{E}(Re(Z)^2 + Im(Z)^2) - ((\mathbb{E}(Re(Z))^2 + (\mathbb{E}(Im(Z))^2),
\end{aligned}$$

$$= \mathbf{Var}(Re(Z)) + \mathbf{Var}(Im(Z)).$$

□

Definition B.5.3 (Covariance)**.** The covariance of a complex-valued random variable Z is:

$$\mathbf{Cov}(Z) = \begin{bmatrix} \mathbf{Var}(Re(Z)) & \mathbf{Cov}(Re(Z), Im(Z)) \\ \mathbf{Cov}(Im(Z), Re(Z)) & \mathbf{Var}(Im(Z)) \end{bmatrix}. \tag{B.12}$$

Definition B.5.4 (Covariance of Two Random Variables)**.** The covariance between two complex-valued random variables X, Y is:

$$\mathbf{Cov}(X, Y) = \mathbb{E}[(X - \mathbb{E}[X]) \cdot (Y - \mathbb{E}[Y])^*]. \tag{B.13}$$

Appendix C

Useful Formulas and Identities

Throughout this thesis, we use trigonometric identities and other formulas to simplify our results, and this section collects all of these facts into a single location.

C.1 Trigonometric Identities

This thesis often moves between exponential and polar forms, and the Euler's formula is the fundamental expression that allows this

$$e^{ix} = \cos(x) + i\sin(x). \tag{C.1}$$

(C.1) translates from exponential to polar form, and Euler's inverse relation goes from polar to exponential

$$\begin{aligned} \cos(x) &= \frac{1}{2}(e^{ix} + e^{-ix}), \\ \sin(x) &= \frac{1}{2i}(e^{ix} - e^{-ix}). \end{aligned} \tag{C.2}$$

The sum-to-product, product-to-sum, and angle addition identities are all used in this thesis to simplify trigonometric expressions. The sum-to-product identity is

$$\begin{aligned}
\sin(x) + \sin(y) &= 2\sin(\frac{x+y}{2})\cos(\frac{x-y}{2}), \\
\sin(x) - \sin(y) &= 2\cos(\frac{x+y}{2})\sin(\frac{x-y}{2}), \\
\cos(x) + \cos(y) &= 2\cos(\frac{x+y}{2})\cos(\frac{x-y}{2}), \\
\cos(x) - \cos(y) &= -2\sin(\frac{x+y}{2})\sin(\frac{x-y}{2}).
\end{aligned} \tag{C.3}$$

The product-to-sum identity is

$$\begin{aligned}
\sin(x)\cos(y) &= \frac{1}{2}[\sin(x+y) + \sin(x-y)], \\
\cos(x)\sin(y) &= \frac{1}{2}[\sin(x+y) - \sin(x-y)], \\
\cos(x)\cos(y) &= \frac{1}{2}[\cos(x-y) + \cos(x+y)], \\
\sin(x)\sin(y) &= \frac{1}{2}[\cos(x-y) - \cos(x+y)].
\end{aligned} \tag{C.4}$$

The angle sum and difference identities are

$$\begin{aligned}
\sin(x+y) &= \sin(x)\cos(y) + \cos(x)\sin(y), \\
\sin(x-y) &= \sin(x)\cos(y) - \cos(x)\sin(y), \\
\cos(x+y) &= \cos(x)\cos(y) - \sin(x)\sin(y), \\
\cos(x-y) &= \cos(x)\cos(y) + \sin(x)\sin(y).
\end{aligned} \tag{C.5}$$

We also use the following identities in Chapter 5 to simplify calculations of the marginal distribution of SWDFT coefficients for white noise signals

$$\begin{aligned}\sum_{j=0}^{n-1}\cos(\frac{2\pi j}{n}) &= \sum_{j=0}^{n-1}\sin(\frac{2\pi j}{n}) = 0,\\ \sum_{j=0}^{n-1}\cos(\frac{2\pi j}{n})^2 &= \sum_{j=0}^{n-1}\sin(\frac{2\pi j}{n})^2 = \frac{n}{2}. \end{aligned} \tag{C.6}$$

These identities hold for all integers, e.g. $\frac{2\pi mj}{n}$, where $m \in \mathbb{Z}$.

C.2 Dirichlet Kernel

The Dirichlet kernel is a fundamental concept in Fourier analysis, and we use it in Chapters 5 and 6. Our primary use of the Dirichlet kernel is to simplify sums of complex exponentials. The Dirichlet kernel is based on the following identity

$$\begin{aligned}\sum_{j=-n}^{n} e^{ijk} &= 1 + 2\sum_{j=1}^{n}\cos(kx),\\ &= \frac{\sin((n+\frac{1}{2})x)}{\sin(\frac{x}{2})}. \end{aligned} \tag{C.7}$$

Variants of (C.7) exist, depending on the summation range. Because this thesis defines the DFT using a summation index from 0 to n - 1, we use use the following variant

$$\sum_{j=0}^{n-1} e^{ijx} = e^{i\frac{n-1}{2}x}\frac{\sin(nx/2)}{\sin(x/2)}. \tag{C.8}$$

Because $\sin(j\cdot\pi) = 0$ when j is an integer, the identity is undefined when x is an integer multiple of π, so the following equation is the complete definition of the Dirichlet kernel we use in this thesis

$$D_n(x) = \begin{cases} n & x = \pi m,\ m = 0, \pm 4, \pm 8, \ldots \\ -n & x = \pi m,\ m = \pm 2, \pm 6, \pm 10, \ldots \\ 0 & x = \pi m,\ m = \pm 1, \pm 3, \pm 5, \ldots \\ \frac{\sin(\frac{nx}{2})}{\sin(\frac{x}{2})} & \text{otherwise.} \end{cases} \tag{C.9}$$

Using (C.9), we can re write (C.8) as

$$\sum_{j=0}^{n-1} e^{ijx} = e^{i\frac{n-1}{2}x} D_n(x). \tag{C.10}$$

$D_n(x)$ is real-valued, so complex multiplication gives the following identities for the real and imaginary parts of the (C.10)

$$\begin{aligned} \sum_{j=0}^{n-1} \cos(jx) &= \cos(\frac{n-1}{2}x) D_n(x), \\ \sum_{j=0}^{n-1} \sin(jx) &= \sin(\frac{n-1}{2}x) D_n(x). \end{aligned} \tag{C.11}$$

In terms of notation, the complex constant factor in (C.10) can be a nuisance, so we sometimes refer to the entire identity as the *Dirichlet weight*

$$\sum_{j=0}^{n-1} e^{ijx} = DW_n(x). \tag{C.12}$$

Finally, because this thesis deals with sliding windows, there are many cases where the summation range is slightly different that 0 to n - 1. We also often use a phase term. Therefore, it's useful to have a generalized complex exponential summation identity, which we derive as

$$\begin{aligned}\sum_{j=a}^{b} e^{i(jx+\phi)} &= e^{i\phi}[e^{iax} + e^{i(a+1)x} + \ldots + e^{i(a+b-a)x}], \\ &= e^{i(ax+\phi)} \sum_{j=0}^{b-a} e^{ijx}, \\ &= e^{i(ax+\phi)} e^{i(\frac{b-a}{2})x} D_{b-a+1}(x), \\ &= e^{i((\frac{a+b}{2}x)+\phi)} D_{b-a+1}(x), \\ &= DW_{a,b}(x+\phi). \end{aligned} \tag{C.13}$$

(C.13) also holds for the real and imaginary parts

$$\begin{aligned}\sum_{j=a}^{b} \cos(jx+\phi) &= \cos((\frac{a+b}{2})x+\phi)D_{b-a+1}(x) = Re(DW_{a,b}(x+\phi)), \\ \sum_{j=a}^{b} \sin(jx+\phi) &= \sin((\frac{a+b}{2})x+\phi)D_{b-a+1}(x) = Im(DW_{a,b}(x+\phi)). \end{aligned} \tag{C.14}$$

C.3 Fourier Shift Theorem

The Fourier shift theorem is a recursive formula we use to relate SWDFT coefficients in window positions with overlapping input data. Recall the the SWDFT for window position p and frequency k is

$$\begin{aligned} a_{k,p} &= \sum_{j=0}^{n-1} x_{p-n+1+j} \omega_n^{-jk}, \\ k &= 0, 1, \ldots, n-1, \end{aligned}$$

$$p = n-1, n, \ldots, N-1. \tag{C.15}$$

To start, we derive the Fourier shift when the window position shifts by one position. In this case, we can re-write the SWDFT coefficient in terms of the coefficient at the same frequency in the previous window position, given by

$$\begin{aligned}
a_{k,p} &= \sum_{j=0}^{n-1} x_{p-n+j}\omega_n^{-(j-1)k} + x_p\omega_n^{-(n-1)k} - x_{p-n}\omega_n^k, \\
&= \sum_{j=0}^{n-1} x_{p-n+j}\omega_n^{-jk}\omega_n^k + x_p\omega_n^{-nk}\omega_n^k - x_{p-n}\omega_n^k, \\
&= \omega_n^k(\sum_{j=0}^{n-1} x_{p-n+j}\omega_n^{-jk} + x_p - x_{p-n}), \\
&= \omega_n^k(a_{k,p-1} + x_p - x_{p-n}),
\end{aligned}$$

where the third to fourth line follows from line 3 of (B.9). When the window position shifts by more than one position, denoted $\delta > 1$, the Fourier shift theorem is given by

$$a_{k,p} = \omega_n^{\delta k}(a_{k,p-\delta} + \sum_{b=0}^{\delta-1}(x_{p-\delta+1+b} - x_{p-n-\delta+1+b})\omega_n^{-bk}). \tag{C.16}$$

A similar derivation is given in Section 3 of Park (2017). This thesis uses the Fourier shift theorem to derive recursive algorithms for the SWDFT in Chapters 3 and 4.

Appendix D

The Maximum of the SWDFT

As we mentioned in Section 9.1.2, deriving the distribution of the maximum SWDFT coefficient has many potential applications. Although we did not find a complete solution in this thesis, this section presents two of our most promising attempts. The goal of this section is formally stating the problem, and showing some ways of approaching the problem, so that future researchers can pick up where we left off.

D.1 Problem Statement

Let $\mathbf{x} = [x_0, x_1, \ldots, x_{N-1}]$ be a length N independent identically distributed signal, where $x_i \sim N(0,1)$. The SWDFT of $\mathbf{x}$ with length n windows is

$$\begin{aligned} a_{k,p,n,N} &= \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} x_{p-n+1+j} \omega_n^{-jk}, \\ k &= 0, 1, \ldots, n-1, \\ p &= n-1, n, \ldots, N-1. \end{aligned} \tag{D.1}$$

We assume that n is fixed, and are interested in the limiting distribution of the maximum SWDFT coefficient as the signal length $N \to \infty$. Denote this as

$$\mathbb{P}(\max_{\substack{0<k\leq n-1 \\ n-1\leq p\leq N-1}} |a_{k,p,n,N}|^2 \leq u_N). \tag{D.2}$$

If all the SWDFT coefficients were independent, (D.2) would simply be

$$\mathbb{P}(|a_{k,p,n,N}|^2 \leq u_N)^{nP}, \tag{D.3}$$

where

$$P \quad = \quad N - n + 1 \approx N, \tag{D.4}$$

and since Chapter 5 showed that $|\sqrt{2} \cdot a_{k,p}|^2$ is marginally distributed as chi square with two degrees of freedom, $|a_{k,p}|^2$ follows an exponential distribution with rate parameter $\lambda = 1$, so the exact distribution of (D.3) is given by

$$\mathbb{P}(|a_{k,p,n,N}|^2 \leq u_N)^{nP} \quad = \quad (1 - e^{-x})^{nP}. \tag{D.5}$$

Unfortunately, SWDFT coefficients in window positions with overlapping input data are not independent, so our solution must account for the dependence structure. In other words, (D.5) provides a very conservative upper bound.

In some cases, it helps to convert the SWDFT from an $n \times P$ array to a 1D length P sequence of maximums for each window position. Let M_p be the random variable corresponding to the maximum SWDFT coefficient at window position p

$$M_p \quad = \quad \max_{k \in \{0, n-1\}} |a_{k,p}|^2. \tag{D.6}$$

Using (D.6), we can define the stationary sequence of maximums in each window position as

$$\mathbf{M} \quad = \quad [M_{n-1}, M_n, \ldots, M_{N-1}]. \tag{D.7}$$

Studying the limiting distribution of $\mathbf{M}$ as $N \to \infty$ provides an equivalent solution.

D.2 Extreme Value Theory Approach

Since we are concerned with the the distribution of the maximum, a natural approach to solving (D.2) applies the tools of extreme value theory. Specifically, because (D.7) is a stationary m-dependent sequence, we can use results from the extremes of stationary sequences (see Chapter 3 of Leadbetter et al. (2012) and Chapter 5 of Coles et al. (2001) for excellent references). The essential idea is that the limiting distribution of a stationary sequence is *close* to the limiting distribution of an independent copy of the sequence with an identical marginal distribution.

The main result is Theorem 5.2 on page 96 of Coles et al. (2001), which is

Theorem D.1 (Theorem 5.2 from Coles et al. (2001))**.** *Let* $\mathbf{x} = [x_0, x_1, \ldots]$ *be a stationary process and let* $\mathbf{x}^* = [x_0^*, x_1^*, \ldots]$ *be a sequence of independent variables with the same marginal distribution. Define* $M_n = \max(\mathbf{x})$ *and* $M_n^* = \max(\mathbf{x}^*)$*. Under suitable regularity conditions,*

$$\mathbb{P}(\frac{M_n^* - b_n}{a_n} \leq z) \quad \rightarrow \quad G_1(z) \tag{D.8}$$

as $n \to \infty$ for normalizing sequences $a_n > 0$ and b_n, where G_1 is a non-degenerate distribution function, if and only if

$$\mathbb{P}(\frac{M_n - b_n}{a_n} \leq z) \quad \rightarrow \quad G_2(z), \tag{D.9}$$

where $G_2(z) = G_1^{\theta}(z)$ for $0 < \theta \leq 1$.

Theorem D.1 implies that if $G_1(x)$ is the limiting distribution of an independent copy of (D.7), then the limiting distribution of (D.7), denoted $G_2(x)$, is related to $G_1(x)$ through an *extremal index* θ by $G_2(x) = G_1^{\theta}(x)$. This means that if we find the limiting distribution of an independent copy of our sequence of maximum (D.6), then we know the limiting distribution of (D.6).

Let $S_N = \max([M_0, M_1, \ldots, M_{N-1}]$, and let $S_N^* = \max([M_0^*, M_1^*, \ldots, M_{N-1}^*]$. The marginal distribution of M_p and M_p^*, $\forall n-1 \leq p \leq N-1$ is

$$\begin{aligned}
\mathbb{P}(M_p \leq y) &= \mathbb{P}(|a_k, p|^2 \leq y)^n, \\
&= (1 - e^{-y})^n. && \text{(D.10)}
\end{aligned}$$

The limiting distribution of S_N^* is

$$\begin{aligned}
\mathbb{P}(\frac{S_N^* - b_N}{a_N} \leq x) &= \mathbb{P}(S_N^* \leq x a_N + b_N), \\
&\leq \mathbb{P}(M_p^* \leq x a_N + b_N)^N,
\end{aligned}$$

$$= \quad (1 - e^{-(xa_N + b_N)})^{nN}. \tag{D.11}$$

The first to second line follows because $P = N - n + 1 \leq N$, and the second to third line follows from (D.10). Define the constants

$$\begin{aligned} a_N &= 1, \\ b_N &= \log(N). \end{aligned} \tag{D.12}$$

Plugging the constants (D.12) back into (D.11) gives

$$\begin{aligned} &= (1 - e^{-(x + \log(N))})^{nN}, \\ &= (1 - e^{-x} e^{-log(N)})^{nN}, \\ &= (1 - \frac{e^{-x}}{N})^{nN}, \\ &\to e^{e^{-x}}, \end{aligned} \tag{D.13}$$

as $N \to \infty$. (D.13) is a Gumbel distribution, so Theorem D.1 says that the limiting distribution of S_N is also a Gumbel distribution raised to the power $0 < \theta \leq 1$. Unfortunately, specifying θ is a difficult problem, and we did not obtain a result in this thesis.

D.3 Chen-Stein Approach

Another approach to deriving the limiting distribution uses a Chen-Stein approximation. The Chen-Stein approach we derive in this section was formulated by Kim (2019). The main idea is that we can rewrite the distribution of the maximum as the sum of dependent indicator variables

$$\mathbb{P}(\max_{k,p} |a_{k,p}|^2 \leq u_N) \quad = \quad \mathbb{P}(\sum_{k=0}^{n-1} \sum_{p=n-1}^{N-1} \mathbb{1}(|a_{k,p}|^2 > u_N) = 0). \tag{D.14}$$

With this insight, we can adapt Theorem 1 from Arratia et al. (1989) to our notation

Theorem D.2 (Theorem 1 from Arratia et al. (1989))**.** *Let I be an index set for window position and frequency, and $\forall$ $(k,p) \in I$, let $X_{(k,p)} \sim Bern(p_{(k,p)} > 0)$. Define*

$$\begin{aligned} W &= \sum_{(k,p)\in I} X_{(k,p)}, \\ \lambda &= \mathbb{E}[W] = \sum_{(k,p)\in I} p_{(k,p)}. \end{aligned} \tag{D.15}$$

Let $B_{(k,p)} \in I$ be all the indices where $X_{(k,p)}$ and $X_{(l,q)}$ are dependent, where $(l,q) \in B_{(k,p)}$. Then define the following

$$\begin{aligned} b_1 &= \sum_{(k,p)\in I} \sum_{(l,q)\in B_\alpha} p_{(k,p)} p_{(l,q)}, \\ b_2 &= \sum_{(k,p)\in I} \sum_{(k,p)\neq(l,q)\in B_{(k,p)}} p_{(k,p)(l,q)}, \text{ where } p_{(k,p)(l,q)} = \mathbb{E}[X_{(k,p)} X_{(l,q)}], \\ b_3 &= \sum_{(k,p)\in I} \mathbb{E}[\mathbb{E}[X_{(k,p)} - p_{(k,p)}| \sum_{(l,q)\in I - B_{(k,p)}} X_{(l,q)}]]. \end{aligned} \tag{D.16}$$

Then the following holds

$$|\mathbb{P}(W = 0) - e^{-\lambda}| \quad \leq \quad \min(1, \lambda^{-1})(b_1 + b_2 + b_3). \tag{D.17}$$

Theorem D.2 says that if we can show that $b_1 + b_2 + b_3 \to 0$ as $N \to \infty$, then we obtain the limiting distribution, which we showed in Section D.2 is a Gumbel. It is not difficult to determine b_1 and b_3, but determining b_2 is trickier, since it requires an explicit expression for the bivariate distribution. However, with more research effort, a Chen-Stein approximation should be possible.

D.4 Summary

Understanding the limiting distribution of the maximum SWDFT coefficient has many potential applications, and would be a natural extension to Fisher's periodicity test (Fisher (1929)). The primary obstacle in obtaining the distribution is the dependence between SWDFT coefficients in window positions with overlapping input data.

This section formally states the problem, and describes two approaches to solve for the limiting distribution. The first approach uses extreme value theory, which shows that the limiting distribution is a Gumbel, but is not able to obtain the exact parameterization due to an unknown *extremal index* parameter. The second approach uses a Chen-Stein approximation, which we believe with more work could yield a correct approximation. Future research efforts along these lines would be valuable, and could provide a statistically principled approach to threshold selection for spectrograms.