A Sparse Random Feature Model for Signal Decomposition

by

Nicholas Joseph Emile Richardson

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Applied Mathematics

Waterloo, Ontario, Canada, 2022

Author's Declaration

Statement of Contributions

Nicholas Richardson was the sole author of Chapters 1, 2, 3, 5 (excl. Section 5.2), and 6 which were written under the supervision of Dr. Giang Tran and not written for publication.

This thesis consists in part of a manuscript written for publication. Exceptions to sole authorship of material are as follows:

Research Presented in Chapter 4 and Section 5.2:

This research was conducted at the University of Waterloo by Nicholas Richardson under the supervision of Dr. Giang Tran. Nicholas Richardson designed the method presented with Dr. Giang Tran, and Dr. Hayden Schaeffer. The numerical experiments, codes, and figures were primarily developed by Nicholas Richardson with guidance and feedback from Dr. Giang Tran and Dr. Hayden Schaeffer. Nicholas Richardson created the data for the mathematical examples in Python, and played and recorded the music used in the musical example.

Nicholas Richardson drafted the manuscript, and all co-authors contributed to the writing and intellectual input of the final manuscript. The submitted paper "SRMD: Sparse Random Mode Decomposition" can be found on arXiv [62].

Abstract

Signal decomposition and multiscale signal analysis provide useful tools for time-frequency analysis. In this thesis, an overview of the signal decomposition problem is given and popular methods are discussed. A novel signal decomposition algorithm is presented: Sparse Random Mode Decomposition (SRMD). This method sparsely represents a signal as a sum of random windowed-sinusoidal features before clustering the time-frequency localized features into the constituent modes. SRMD outperforms state-of-the-art methods on a variety of mathematical signals, and is applied to real-world astronomical and musical examples. Finally, we discuss a neural network approach to tackle challenging musical signals.

Acknowledgements

I would like to thank everyone who made this thesis possible. This includes Prof. Rachel Ward for her feedback refining the novel decomposition algorithm, and Prof. Hayden Schaeffer for his constant support and humour. I also want to acknowledge Prof. Kirsten Morris and Prof. Stephen Vavasis for their time spent on my thesis examining committee and valuable revisions. Finally, I want to give special thanks to my supervisor Prof. Giang Tran for her constant encouragement and critical feedback throughout my degree.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

BP basis pursuit, 5

BPDN basis pursuit denoising, 5

CEEMDAN Complete EEMD with Adaptive Noise, 20

CWT continuous wavelet transform, 13

DBSCAN Density Based Spatial Clustering of Applications with Noise, 6

EEMD Ensemble EMD, 18

EMD Empirical Mode Decomposition, 17

EWT Empirical Wavelet Transform, 20

IF instantaneous frequency, 10

IMF intrinsic mode function, 9

LASSO least absolute shrinkage and selection operator, 5

RFM random feature model, 14

SPGL1 spectral projected-gradient for sparse least squares, 5

SRMD Sparse Random Mode Decomposition, 24

SST synchrosqueezing transform, 22

STFT short-time Fourier transform, 12

VDM Variational Mode Decomposition, 21

Chapter 1

Introduction

Signals can be found across a variety of academic and industrial domains such as seismology, medicine, and telecommunication, and in everyday life from music to photography. It is therefore imperative we study ways to analyze and process signals for these applications. One such analytical tool is signal decomposition which separates an input into the sum of constituent modes. Signal decomposition may be used to identify important sources with a signal, or as a preprocessing step before further analytical tools are applied.

In this thesis, we propose a novel decomposition algorithm: Sparse Random Mode Decomposition (SRMD). This two stage algorithm starts with a random feature method for analyzing time-series data by constructing a sparse approximation to the short-time Fourier transform spectrogram. The randomization is both in the time window locations and the frequency sampling, which lowers the overall sampling and computational cost. The sparsification of the spectrogram leads to a sharp separation between time-frequency clusters which makes it easier to identify intrinsic modes, and thus leads to a new data-driven mode decomposition. The applications of this method include signal representation, visualization, and outlier removal in addition to mode decomposition.

Background and notation on required topics including linear algebra, sparse optimization, signal processing, and random feature models are provided in Chapter 2. This chapter is followed by an overview of the signal decomposition problem and popular methods for tackling the problem in Chapter 3. The proposed SRMD algorithm is detailed in Chapter 4 along with comparisons to state-of-the-art methods. Chapter 5 is dedicated to signal decomposition in the context of musical signals where the problem is approached using SRMD and an adapted neural network architecture. Finally, the thesis is concluded in Chapter 6 along with related future work that may be of interest.

Chapter 2

Background & Notation

2.1 Linear Algebra

For clarity when discussing mathematical objects, scalar functions and elements are kept italicized and unbolded (ex. f, x) where as vectors and matricies bolded (ex. $\boldsymbol{x}, \boldsymbol{A}$). Vectors generally use lower-cased symbols and matricies use upper-cased symbols.

If we wish to identify the i^{th} element of a vector $\boldsymbol{x}$, we subscript the unbolded letter x_i or subscript parenthesis $(\boldsymbol{x})_i$. We may then write a vector $\boldsymbol{x} \in \mathbb{R}^m$, thought of as a column vector $\boldsymbol{x} \in \mathbb{R}^{m \times 1}$ by convention, in full as

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} = \begin{bmatrix} x_1 & \dots & x_m \end{bmatrix}^\top .$$

Here we use the transpose symbol $\top$ to swap the two dimensions of a matrix, or convert a column vector to a row vector. We also use a bolded zero for a vector of all zeros $\mathbf{0} \in \mathbb{R}^m$.

We similarly use double subscripts a_{ij}, A_{ij} or subscript parenthesis $(\boldsymbol{A})_{ij}$ to denote the element in the i^{th} row and j^{th} column of a matrix $\boldsymbol{A}$. For $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, we may write this in full as

$$\boldsymbol{A} = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{bmatrix} = [a_{ij}].$$

When we wish to discuss a group of vectors, we maintain the bolded vector convention while indexing $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_j, \ldots, \boldsymbol{a}_n \in \mathbb{R}^m$ or $\{\boldsymbol{a}_j\}_{j=1}^n \subset \mathbb{R}^m$ to avoid confusion with their entries $(\boldsymbol{a}_j)_i \in \mathbb{R}$. Using this convention we define the following.

Definition 1 (Linearly Dependent)**.** A set of vectors $\{\boldsymbol{a}_j\}_{j=1}^n \subset \mathbb{R}^m$ is *linearly dependent* if there exists $x_1, \ldots, x_n \in \mathbb{R}$ not all zero such that

$$\sum_{j=1}^{n} x_j \boldsymbol{a}_j = \mathbf{0}.$$

We say a set of vectors is *linearly independent* if they are not linearly dependent.

We also use the standard norms on $\mathbb{R}^n$ defined as follows.

Definition 2 (ℓ^2-norm)**.** The *Euclidean* or *ℓ^2-norm* $\|\cdot\|_2 : \mathbb{R}^n \to \mathbb{R}$ of a vector $\boldsymbol{x} \in \mathbb{R}^n$ is defined as

$$\|\boldsymbol{x}\|_2 = \left(\sum_{i=1}^{n} x_i^2\right)^{\frac{1}{2}}.$$

Definition 3 (ℓ^1-norm)**.** The *ℓ^1-norm* $\|\cdot\|_1 : \mathbb{R}^n \to \mathbb{R}$ of a vector $\boldsymbol{x} \in \mathbb{R}^n$ is defined as

$$\|\boldsymbol{x}\|_1 = \sum_{i=1}^{n} |x_i|.$$

Definition 4 (ℓ^0-norm)**.** The *ℓ^0-norm* $\|\cdot\|_0 : \mathbb{R}^n \to \mathbb{R}$ of a vector $\boldsymbol{x} \in \mathbb{R}^n$ is defined as

$$\|\boldsymbol{x}\|_0 = |\{i \in [n] \mid x_i \neq 0\}|.$$

Here, we use $[n]$ to denote the set of positive integers $\{1, \ldots, n\}$. In other words, $\|\boldsymbol{x}\|_0$ is the number of nonzero entries of $\boldsymbol{x}$ [23].

Remark. Although we used the word "norm" in the definition of the ℓ^0-norm, it is *not* formally a mathematical norm.

2.2 Numerical Algebra & Sparse Optimization

Consider the inverse problem of solving $\boldsymbol{Ax} = \boldsymbol{y}$ for $\boldsymbol{x}$ where $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{y} \in \mathbb{R}^m$ are known. This could represent some linear model where we wish to write $\boldsymbol{y}$ as a linear combination of column vectors $\boldsymbol{a}_j$ of $\boldsymbol{A}$ with coefficients x_j. There are two cases to consider.

Case 1: $m \geq n$

If $\boldsymbol{A}$ has linearly independent columns, the unique solution is given by

$$\boldsymbol{x} = (\boldsymbol{A}^\top \boldsymbol{A})^{-1} \boldsymbol{A}^\top \boldsymbol{y}$$

and can be solved with techniques such as QR Factorization [27]. When $m = n$ ($\boldsymbol{A}$ is square), this reduces to $\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{y}$ which can also be solved using a variety of methods such as LU Decomposition, Cholesky Method, or Conjugate Gradient Method depending on the structure of $\boldsymbol{A}$ [27].

When $\boldsymbol{A}$ has linearly dependent columns, there are either infinitely many solutions when $\boldsymbol{y}$ is in the column space of $\boldsymbol{A}$, or no solutions otherwise. In the former case, all solutions are given by $\boldsymbol{x} = \hat{\boldsymbol{x}} + \boldsymbol{z}$ where $\hat{\boldsymbol{x}}$ is any one solution to $\boldsymbol{A}\hat{\boldsymbol{x}} = \boldsymbol{y}$ and $\boldsymbol{z}$ is in the kernel of $\boldsymbol{A}$ ($\boldsymbol{A}\boldsymbol{z} = \boldsymbol{0}$).

Case 2: $m < n$

In this case, $\boldsymbol{A}$ has more columns than rows and is *underdetermined*. This means the columns of $\boldsymbol{A}$ are always linearly dependent and we have infinitely many solutions or no solutions similar to the later part of Case 1. Rather than conclude the discussion, we may impose additional constraints on the solution $\boldsymbol{x}$ to reduce the case of infinitely many solutions to a unique solution.

One such constraint may be to minimize the ℓ^2-norm of the solution

$$\min_{\boldsymbol{x} \in \mathbb{R}^n} \|\boldsymbol{x}\|_2^2 \quad \text{s.t.} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}.$$

This may be appropriate if we wish to find a solution that has entries x_j with small magnitude or the solution closest to $\boldsymbol{0} \in \mathbb{R}^n$. When the rows of $\boldsymbol{A}$ are linearly independent, this has the closed-form solution

$$\boldsymbol{x} = \boldsymbol{A}^\top (\boldsymbol{A}\boldsymbol{A}^\top)^{-1} \boldsymbol{y},$$

which can again be solved with techniques such as QR Factorization [27].

If we instead wish to find the sparsest solution for $\boldsymbol{x}$, we can define the problem

$$\min_{\boldsymbol{x} \in \mathbb{R}^n} \|\boldsymbol{x}\|_0 \quad \text{s.t.} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}. \tag{P_0}$$

This may be desired if we would like the solutions with the most zero entries for data compression, easier visualization, or require the least number of columns of $\boldsymbol{A}$ to represent $\boldsymbol{y}$. This last approach is the most relevant to Chapter 4.

The P_0 problem is hard to solve in practice (specifically *NP-hard*) [23], so the convex relaxation of this problem, called the *basis pursuit* (BP) problem, is often more practical:

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} \|\boldsymbol{x}\|_1 \quad \text{s.t.} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}. \tag{BP}$$

This recovers the solution to P_0 under some assumptions which are detailed in [23]. Another issue can now be raised. If the matrix $\boldsymbol{A}$ or vector $\boldsymbol{y}$ is subject to perturbations or noise, the solution to the basis pursuit problem may not be stable. We adjust our original $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}$ problem to

$$\boldsymbol{A}\boldsymbol{x} + \boldsymbol{\epsilon} = \boldsymbol{y},$$

where $\boldsymbol{\epsilon} \in \mathbb{R}^m$ represents some unknown noise in our measurement of $\boldsymbol{y}$. This can be implemented into our optimization problem by considering the (ℓ^1) *basis pursuit denoising* (BPDN) problem:

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} \|\boldsymbol{x}\|_1 \quad \text{s.t.} \quad \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2 \leq \sigma. \tag{BPDN}$$

This can be equivalently stated, for $\tau = \|\boldsymbol{x}_\sigma\|_1$ when $\boldsymbol{x}_\sigma$ is a unique solution to the BPDN problem, as the *least absolute shrinkage and selection operator* (LASSO) problem

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 \quad \text{s.t.} \quad \|\boldsymbol{x}\|_1 \leq \tau, \tag{LASSO}$$

or the *penalized least-squares* (QP) problem

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{x}\|_1 \tag{QP}$$

for an appropriate choice of λ [74].

Remark. We use the naming convention of these problems according to [74] rather than [23] to match the convention used by the Python package we import to solve these problems.

The QP problem can be solved with a variety of methods like the *fast iterative shrinkage-thresholding algorithm* [3] or *Nesterov's method* [52]. However, the BPDN problem can be a more natural way of expressing the problem. We often do not know a reasonable bound τ on $\|\boldsymbol{x}\|_1$ or a good choice of λ *a priori*, but can give some upper bound σ on the desired reconstruction error $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2$.

This thesis focuses on SPGL1: a spectral projected-gradient solver for sparse least squares, designed to solve the BP, BPDN, and LASSO problems [74]. The BPDN problem can be challenging to solve directly, so the SPGL1 method solves a sequence of LASSO problems $\mathcal{P}_{\tau_k}$ with hyperparameter τ_k using a spectral projected-gradient algorithm similar to [4]. The τ_k are updated by Newton's method $\tau_{k+1} = \tau_k - f(\tau_k)/f'(\tau_k)$ in order to find the root of $f(\tau) = \|\boldsymbol{A}\boldsymbol{x}_\tau - \boldsymbol{y}\|_2 - \sigma$. Here, $\boldsymbol{x}_\tau$ is the solution to $\mathcal{P}_\tau$. The solution for the BPDN problem $\boldsymbol{x}_\sigma$ coincides with the solution to the LASSO problem $\boldsymbol{x}_{\tau_k}$ once $f(\tau_k) = 0$. The SPGL1 algorithm is efficiently implemented in MATLAB [75] and Python [15].

2.3 Clustering

Clustering is the concept of finding structure within data. In density based clustering, this typically means grouping together data points that are "close" under some notion of distance [65]. There are many algorithms that exist such as k-means, spectral clustering, and BIRCH, to name a few, each with their best use cases [55]. Nonnegative matrix factorization has also been used as a clustering tool since it is shown to be equivalent to k-means and spectral clustering under some settings [14], and another method called Density Based Spatial Clustering of Applications with Noise (DBSCAN) also has a relationship to matrix factorization [66].

In this thesis, we focus on DBSCAN [19]. In the context of the proposed decomposition algorithm in Chapter 4, this method is sufficient in terms of computational efficiency and ability to accurately identify the clusters in this setting. The main advantages of DBSCAN—over other clustering methods like k-means for example—is its ability to cluster arbitrary shaped regions, cluster by density of points, and identify the number of clusters.

We consider a data set of m points $X = \{\boldsymbol{x}_i\}_{i=1}^m \subset \mathbb{R}^n$, with the distance function $d(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\|_2$. We require the following definitions.

Definition 5 (ε-neighbourhood)**.** Given a set $X = \{\boldsymbol{x}_i\}_{i=1}^m \subset \mathbb{R}^n$, and $\varepsilon \in \mathbb{R}_{>0}$, the *$\varepsilon$-neighbourhood* of a point $\boldsymbol{x} \in X$ is the set given by

$$N_\varepsilon(\boldsymbol{x}) = \{\boldsymbol{y} \in X \mid \|\boldsymbol{x} - \boldsymbol{y}\|_2 \leq \varepsilon\}.$$

Definition 6 (Directly Density-Reachable)**.** Given a set $X = \{\boldsymbol{x}_i\}_{i=1}^m \subset \mathbb{R}^n$, a point $\boldsymbol{x} \in X$ is *directly density-reachable* from $\boldsymbol{y} \in X$ with respect to $\varepsilon \in \mathbb{R}_{>0}$ and $\texttt{min\_points} \in \mathbb{Z}_{>0}$ if the following conditions hold:

1. $\boldsymbol{x} \in N_\varepsilon(\boldsymbol{y})$
2. $|N_\varepsilon(\boldsymbol{y})| \geq \texttt{min\_points}$.

Here, $|N_\varepsilon(\boldsymbol{y})|$ denotes the cardinally or the number of points in $N_\varepsilon(\boldsymbol{y})$, and examples are given in Figure 2.1.

Definition 7 (Density-Reachable)**.** Given a set $X = \{\boldsymbol{x}_i\}_{i=1}^m \subset \mathbb{R}^n$, a point $\boldsymbol{x} \in X$ is *density-reachable* from $\boldsymbol{y} \in X$ with respect to $\varepsilon \in \mathbb{R}_{>0}$ and `min_points` $\in \mathbb{Z}_{>0}$ if there exists a chain of points $\boldsymbol{z}_1, \dots, \boldsymbol{z}_p \in X$ such that $\boldsymbol{z}_1 = \boldsymbol{y}$, $\boldsymbol{z}_p = \boldsymbol{x}$, and $\boldsymbol{z}_{i+1}$ is directly density-reachable from $\boldsymbol{z}_i$.

Definition 8 (Cluster)**.** The *clusters* $C_1, \dots, C_K \subseteq X$ with respect to ε and `min_points` are the sets of points

$$C_i = \{\boldsymbol{x} \in X \mid \boldsymbol{x} \text{ density-reachable from } \boldsymbol{y}_i\}$$

where $|N_\varepsilon(\boldsymbol{y}_i)| \geq$ `min_points` and $\boldsymbol{y}_i \in X$ are not density-reachable to each other.

The DBSCAN algorithm identifies the points belonging to the clusters $C_1, \dots, C_k \subseteq X$ defined in Definition 8, where any leftover points are labelled as noise. The algorithm is summarized in Algorithm 1.

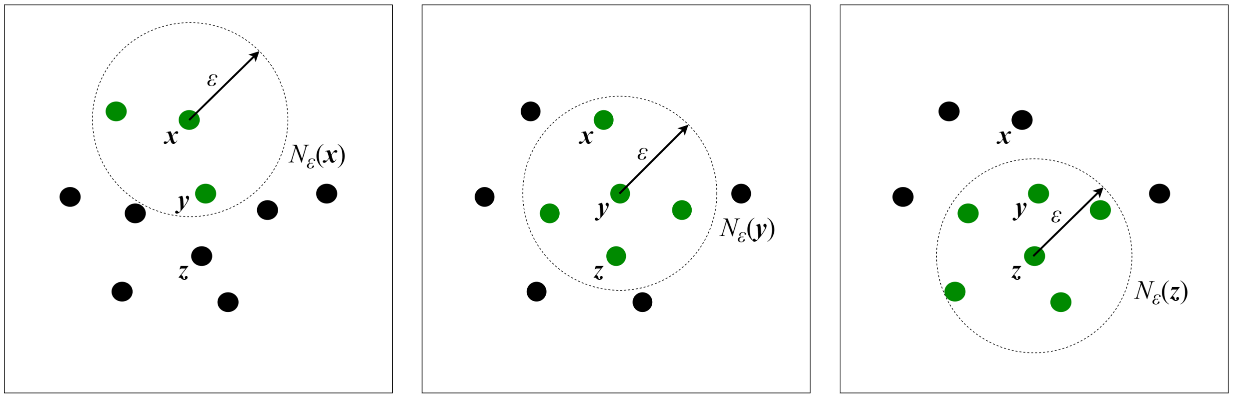


Figure 2.1: **Example ε-neighbourhoods:** Left to Right: ε-neighbourhoods around $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in X$ shown in green. Suppose `min_points` $= 4$. We can now say that $\boldsymbol{x}$ is directly density-reachable from $\boldsymbol{y}$ (see middle plot), and $\boldsymbol{y}$ is directly density-reachable from $\boldsymbol{z}$ (see right plot). This makes $\boldsymbol{x}$ density-reachable from $\boldsymbol{z}$ by the chain $\boldsymbol{z}, \boldsymbol{y}, \boldsymbol{x}$. Note $\boldsymbol{y}$ is *not* directly density-reachable from $\boldsymbol{x}$ since $|N_\varepsilon(\boldsymbol{x})| = 3 <$ `min_points` (see left plot).

Algorithm 1 Density Based Spatial Clustering of Applications with Noise (DBSCAN)

Input: Data points $X = \{\boldsymbol{x}_i\}_{i=1}^m$, ε, and `min_points`.

Method:

```
    let $k = 1$
    for $\boldsymbol{x} \in X$:
        if $\boldsymbol{x}$ is not labelled and $|N_\varepsilon(\boldsymbol{x})| \geq$ min_points:
            label all $\boldsymbol{y} \in N_\varepsilon(\boldsymbol{x})$ to be in cluster $C_k$
            let $Y = N_\varepsilon(\boldsymbol{x}) \setminus \{\boldsymbol{x}\}$
            while $Y$ is not empty:
                let $\boldsymbol{y}$ be the first point in $Y$
                if $|N_\varepsilon(\boldsymbol{y})| \geq$ min_points:
                    add unlabelled $\boldsymbol{z} \in N_\varepsilon(\boldsymbol{y})$ to $Y$
                    label all $\boldsymbol{z} \in N_\varepsilon(\boldsymbol{y})$ to be in cluster $C_k$
                update $Y \leftarrow Y \setminus \{\boldsymbol{y}\}$
            update $k \leftarrow k + 1$
    Output: Clusters $C_1, \ldots, C_K$
```

2.4 Signal Processing

2.4.1 Sampling

In the field of signal processing, we look at ways to analysis and manipulate signals. Signals can be thought of as a continuous function in time (ex. an audio signal $f : \mathbb{R} \to \mathbb{R}$ where the input t is time and the output y is the amplitude), in space (ex. a grayscale image $f : \mathbb{R}^2 \to \mathbb{R}$ where the input (x, y) is a point in space and the output z is the brightness of the pixel), or both (ex. a colour video $f : \mathbb{R}^3 \to \mathbb{R}^4$ with input (t, x, y) and output (r, b, g, a) corresponding the brightness of the red, green, and blue channels, and the audio channel). Many signal processing techniques can be applied directly on these signals on their continuous domain, but many modern techniques are applied on a discretized or *sampled* version of the signal. In this thesis, we focus our attention to finite 1-dimensional signals which take the form $f : \mathbb{R} \to \mathbb{R}$ where the support set of the signal is some finite interval $\mathcal{D} = [a, b] \subset \mathbb{R}$.

Definition 9 (Equally-Spaced Sampling)**.** The *sampled signal* $\boldsymbol{y} \in \mathbb{R}^m$ of f is given by $y_i = f(t_i)$ where $t_1, \ldots, t_m \in \mathcal{D}$. Here, the samples are evenly spaced so that $t_{i+1} - t_i = \Delta t$ is constant. For example, $t_i = a + (i-1)\frac{b-a}{m}$.

Definition 10 (Random Sampling)**.** A (uniformly) *random sampled signal* $\boldsymbol{y} \in \mathbb{R}^m$ on an interval $[a, b]$ is given by $y_i = f(t_i)$ where $t_1, \dots, t_m \sim \mathcal{U}(a, b)$. For convenience, we re-index the drawn t_i so that $t_1 \leq \dots \leq t_m$.

Definition 11 (Sample Rate)**.** The *sample rate* for evenly sampled signals is $s = \frac{1}{\Delta t}$ and can be expressed in the unit of frequency *hertz* Hz when t is given in seconds.

Definition 12 (Nyquist Rate)**.** The *Nyquist rate* is the minimum sample rate needed to accurately reproduce an evenly sampled signal from its discrete Fourier coefficients and is equal to twice the highest frequency in the signal. Conversely, the highest frequency that can be reproduced from an equally sampled signal at a sample rate s is $\omega_{\max} = s/2$ and is called the *Nyquist frequency* [6].

2.4.2 Sinusoids and Intrinsic Mode Functions

Definition 13 (Sinusoids)**.** We often work with general *sinusoidal* functions of the form

$$g(t) = A \sin(\omega t + \varphi)$$

and interchangeably call them *pure tones* or *harmonics.* We call $A \in \mathbb{R}_{>0}$ the amplitude, $\omega \in \mathbb{R}_{>0}$ the (angular) frequency, and $\varphi \in [0, 2\pi)$ the phase or phase-shift.

Note. In musical applications, we use *harmonics* to specifically refer to the sinusoidal functions

$$\{A_n \sin(\omega_n t + \varphi_n)\}_{n=1}^{\infty}$$

where their frequency is some integer multiple (or near-integer multiple in real-world signals) $\omega_n = (n+1)\omega_0,\ n \in \mathbb{Z}_{>0}$ to a *fundamental* pure tone $A_0 \sin(\omega_0 t + \varphi_0)$.

Definition 14 (Units of Frequency)**.** When t is given in the unit seconds, the angular frequency ω is given in the units *radians per second* written $\frac{\text{rad}}{s}$. This can be converted to (ordinary) frequency in Hz by the relationship $\omega_{\text{ordinary}} = \dfrac{\omega_{\text{angular}}}{2\pi}$.

Definition 15 (Intrinsic Mode Function)**.** An *intrinsic mode function* (IMF) [13] is an amplitude-frequency modulated sinusoidal function of the form

$$f(t) = a(t) \sin(\phi(t)),$$

where $a(t), \phi'(t) > 0\ \forall t$.

Assumption 1. Typically, we further restrict $a(t)$ and $\phi'(t)$ to vary slower than $\phi(t)$. More precisely, when a and ϕ have continuous second derivatives, we say $a(t)$ varies ε slower than $\phi(t)$ for some $\varepsilon \in \mathbb{R}_{>0}$ if

$$\left|\frac{a'(t)}{a(t)}\right| < \varepsilon\phi'(t) \quad \& \quad \left|\frac{a''(t)}{a(t)}\right| < \varepsilon\phi'(t)^2,$$

and $\phi'(t)$ varies ε slower than $\phi(t)$ if

$$|\phi''(t)| < \varepsilon\phi'(t)^2.$$

Using Assumption 1, we present Proposition 1 which, to the author's knowledge, is not found in the literature and leads to a well motivated definition for the instantaneous frequency and amplitude of an IMF.

Proposition 1 (IMF's Sinusoidal Approximation). *Let $f(t) = a(t)\sin(\phi(t))$ be an IMF where, a and ϕ have continuous second derivatives, $a(t)$ and $\phi'(t)$ vary $\tilde{\varepsilon}$ slower than $\phi(t)$, and $g(t) = A\sin(\omega t + \varphi)$ be a pure tone with $A = a(\tau)$, $\omega = \phi'(\tau)$, and $\varphi = \phi(\tau) - \tau\phi'(\tau)$ for some $t = \tau$. Then g well approximates f at $t = \tau$. Specifically, $\forall\varepsilon > 0$, $\exists\delta > 0$ such that $|t - \tau| < \delta$ implies:*

$$\begin{aligned} |f(t) - g(t)| &< \varepsilon, \\ |f'(t) - g'(t)| &< \varepsilon + C_1\tilde{\varepsilon}, \\ |f''(t) - g''(t)| &< \varepsilon + C_2\tilde{\varepsilon}, \end{aligned}$$

where $C_1 = a(\tau)\phi'(\tau)$ and $C_2 = 4a(\tau)\phi'(\tau)^2$.

Proof. See Appendix A. □

This lets us appropriately define the instantaneous frequency and amplitude.

Definition 16 (Instantaneous Frequency). The *instantaneous frequency* (IF) of an IMF is given by the rate-of-change of the phase $\omega_{IF}(t) = \phi'(t)$. The instantaneous amplitude of an IMF is simply $a(t)$.

Definition 17 (Intersecting IMFs). Using the idea of IF, we say two IMFs f_1 and f_2 *cross* or *intersect* if there exists some $\tau \in \mathbb{R}$ such that their IFs are the same ${\phi_1}'(\tau) = {\phi_2}'(\tau)$ at $t = \tau$.

2.4.3 Time-Frequency Analysis

In time-frequency analysis, we aim to calculate or visualize the frequencies within a time-series .

Definition 18 (L^p spaces)**.** We define the function space L^p on a set $X \subseteq \mathbb{R}$ as

$$L^p(X) = \{f : X \to \mathbb{C} \ \mid \|f\|_p < \infty\}.$$

The L^p*-norm* of f is defined as

$$\|f\|_p = \left(\int_X |f(x)|^p \mathrm{d}x\right)^{\frac{1}{p}}.$$

Definition 19 (Inner Product)**.** The *inner product* $\langle \cdot, \cdot \rangle : L^2(\mathbb{R}) \times L^2(\mathbb{R}) \to \mathbb{C}$ of two functions $f, g \in L^2(\mathbb{R})$ is

$$\langle f, g \rangle = \int_{-\infty}^{\infty} f(t)\overline{g(t)}\mathrm{d}t.$$

We use $\overline{z} = x - iy$ to denote the complex conjugate of $z = x + iy \in \mathbb{C}$ where $x, y \in \mathbb{R}$.

Definition 20 (Convolution)**.** We also define the continuous *convolution* $* : L^2(\mathbb{R}) \times L^2(\mathbb{R}) \to L^2(\mathbb{R})$ of the functions $f, g \in L^2(\mathbb{R})$ as

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t - \tau) \ \mathrm{d}\tau.$$

Definition 21 (Fourier Transform)**.** The *Fourier transform* of $f \in L^1(\mathbb{R})$ is the function

$$\hat{f}(\omega) = \int_{-\infty}^{\infty} f(t)e^{-2\pi i\omega t}\mathrm{d}t, \tag{2.1}$$

and its inverse, when $\hat{f} \in L^1(\mathbb{R})$, is given by

$$f(t) = \int_{-\infty}^{\infty} \hat{f}(\omega)e^{2\pi i\omega t}\mathrm{d}\omega. \tag{2.2}$$

On its own, $|\hat{f}(\omega)|$ only gives the magnitude of a particular frequency ω in the *entire* signal. When the input signal f has time varying frequencies like an IMF with $\phi''(t) \neq 0$, the Fourier transform only says how much of each frequency appears without localizing the frequency information in time. This is where the STFT becomes helpful in analyzing these signals.

Definition 22 (Short-Time Fourier Transform)**.** The *short-time Fourier transform* (STFT) [29] of a signal $f \in L^1(\mathbb{R}) \cap L^2(\mathbb{R})$ with respect to the *window function* $W \in L^2(\mathbb{R})$ is given by

$$F_W(\omega, \tau) = \int_{-\infty}^{\infty} f(t) W(t-\tau) e^{-2\pi i \omega t} \mathrm{d}t \tag{2.3}$$

and its inverse, when $\hat{f} \in L^1(\mathbb{R})$, is defined as

$$f(t) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F_W(\omega, \tau) e^{2\pi i \omega t} \mathrm{d}\tau \mathrm{d}\omega. \tag{2.4}$$

Here, we assume W has the following properties:

1. $\displaystyle\int_{-\infty}^{\infty} W(t) \mathrm{d}t = 1$
2. $|W(t)| \leq 1$ and equal to 1 at $t = 0$
3. $|W(t)|$ is close to or exactly zero outside some finite interval $T = [-t_0, t_0] \subset \mathbb{R}$.

Note. To define the inverse STFT, we require W to have the above property 1. Properties 2 & 3 are for convenient analysis of f to localize the frequency information at $t = \tau$.

Example. A Gaussian $W_\sigma(t) = \exp(-t^2/2\sigma^2)$ is a commonly used window function.

Definition 23 (Discrete STFT)**.** The *discrete short-time Fourier transform* of a signal $\boldsymbol{y} \in \mathbb{R}^N$ is given by

$$Y_{k,m} = \sum_{n=0}^{N-1} y_n w_{n-m} e^{-2\pi i n k/N}$$

where $\boldsymbol{w} \in \mathbb{R}^N$ is the sampled window function $W(t_n) = w_n$ that is periodic in its index $n \in [N]$. That is, $w_n = w_{n+N}\ \forall n \in \mathbb{Z}$.

The STFT offers similar frequency analysis as the Fourier transform, but with additional time-localized information. Not only does the STFT give information about the magnitude of a frequency ω in the input signal, it also expresses when that frequency occurs τ. This means the STFT of an IMF can reveal its instantaneous frequency and amplitude since a windowed IMF at τ looks like a pure tone $\sin(\omega t + \varphi)$ as described in Proposition 1. Loosely, we would observe $|F_W(\omega, \tau)|$ is large if the time-frequency pair $(\tau, \omega) \in \mathbb{R}^2$ is close to $(\tau, \phi'(\tau))$, and small otherwise.

A drawback to the STFT is that smaller window sizes T, although better for localizing time information, leads to poorer frequency localization. This phenomenon can be summarized by the uncertainty principle given in Theorem (2.3.1) of [29].

Theorem 1 (Uncertainty Principle). *Let $W \in L^2(\mathbb{R})$ be a window function and $\varepsilon_T, \varepsilon_\Omega \in \mathbb{R}_{>0}$ be such that*

$$\int_{\mathbb{R}\setminus T} |W(t)|^2 \mathrm{d}t \leq \varepsilon_T^2 \|W\|_2^2 \qquad \& \qquad \int_{\mathbb{R}\setminus \Omega} |\hat{W}(\omega)|^2 \mathrm{d}\omega \leq \varepsilon_\Omega^2 \|\hat{W}\|_2^2.$$

Then, $|T||\Omega| \geq (1-\varepsilon_T-\varepsilon_\Omega)^2$, where $|T|, |\Omega|$ are the length of the intervals $T, \Omega \subset \mathbb{R}$.

The uncertainty principle states that any window which is well localized in time (small $|T|$ and ε_T), will ensure its frequencies are spread over a large frequency interval (big $|\Omega|$ and/or ε_Ω) and visa versa. To circumvent the principle, wavelets can be used rather than windowed sinusoids. This allows us to define a continuous wavelet transform, where time-frequency pairs with lower frequency are better localized in frequency, and time-frequency pairs with higher frequency are better localized in time.

Definition 24 (Mother Wavelet). A *mother wavelet* is a function $\psi \in L^1(\mathbb{R}) \cap L^2(\mathbb{R})$ that permits a family of scaled and shifted wavelets $\{\psi_{a,b}(t)\}$ given by

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}\|\psi\|_2}\psi\left(\frac{t-b}{a}\right), \quad a \in \mathbb{R}_{>0} \text{ and } b \in \mathbb{R},$$

and satisfies $\int_{-\infty}^{\infty} \psi(t)\mathrm{d}t = 0$ [29]. We call a the scale and b the shift.

Definition 25 (Continuous Wavelet Transform). The *continuous wavelet transform* (CWT) [29] of a function $f \in L^2(\mathbb{R})$ with respect to a mother wavelet ψ is defined as

$$F_\psi(a,b) = \langle f, \psi_{a,b}\rangle = \int_{-\infty}^{\infty} f(t)\overline{\psi_{a,b}(t)}\,\mathrm{d}t, \tag{2.5}$$

and its inverse, if $C_\psi := \int_{-\infty}^{\infty} \frac{|\hat{\psi}(\omega)|^2}{\omega}\mathrm{d}\omega < \infty$, is given by

$$f(t) = \frac{1}{C_\psi}\int_0^\infty \int_{-\infty}^{\infty} \frac{1}{a^2} F_\psi(a,b)\psi_{a,b}(t)\,\mathrm{d}b\,\mathrm{d}a, \tag{2.6}$$

at points where f is continuous at t.

Note. The CWT's scale a is *inversely* related to the STFT's frequency ω.

Various example wavelets such as the Haar and Daubechies wavelets can be found in [17]. The wavelet we use when we refer to *the* CWT is the Morlet wavelet given in the following example. This is most similar to the STFT since it has a Gaussian window and makes for a fair comparison between the STFT and CWT.

Example (Morlet Wavelet). The *Morlet wavelet* [43] with respect to $\beta \in \mathbb{R}_{>0}$ is the function

$$\psi_\beta(t) = \exp(-\beta^2 t^2/2)\left(\cos(\pi t) - \exp(-\pi^2/2\beta^2)\right).$$

2.5 Random Feature Models

The standard random feature model (RFM) architecture consistent of a two-layer fully connected neural network whose single hidden layer is randomized and not trained [5, 59, 60, 46, 50]. That is, we assume a function $f : \mathbb{R}^d \to \mathbb{C}$ can be approximated by a collection of N functions [30] $\phi_j : \mathbb{R}^d \to \mathbb{C}$, $j \in [N]$ where

$$f(\boldsymbol{x}) \approx \sum_{j=1}^{N} c_j \phi_j(\boldsymbol{x}).$$

The only layer that is trained is the output layer, i.e. learn suitable c_j, thus yielding a linear training model. The functions $\phi_j(\boldsymbol{x}) = g(\boldsymbol{x}; \boldsymbol{\theta}_j)$ are parametrized by $\boldsymbol{\theta}_j$ which are drawn randomly from some distribution.

There is a wide range of theoretical results for RFM used for interpolation or regression [61, 64, 42, 18, 30, 49, 2, 67]. In [60] using N random features is shown to yield a uniform error bound of $\mathcal{O}(N^{-\frac{1}{2}} + m^{-\frac{1}{2}})$ for target functions in a certain class related which are embedded within a reproducing kernel Hilbert space when the RFM is trained using Lipschitz loss functions. For the ℓ^2 loss, if the number of features scales like $N \sim \sqrt{m} \log m$ where m is the number of data points, then the test error is bounded by $\mathcal{O}(m^{-\frac{1}{2}})$ [64], see also [42]. This result requires that the target function f is in the associated reproducing kernel Hilbert space and some additional assumptions on the kernel. By analyzing the structure of the RFM with respect to the dimension d and the parameters N and m, results found in [49, 12] showed that regression using RFM often achieve their minimal risk in the over-parameterized region, where the number of random features exceeds the number of data samples.

Chapter 3

The Signal Decomposition Problem

3.1 Overview

Signal decomposition is a challenging problem and used in many fields including seismology [35, 77, 82], digital audio [57, 37, 45], and medical signal processing [11, 69] to name a few. Also called source separation, the problem involves separating an input signal into important sources or modes to analysis the input, or as a preprocessing step before applying further processing that is better suited to individual sources. Signal decomposition makes it easy to extract important features in a signal and can be adapted for other subproblems such as denoising.

The problem can be be applied generally on multi-channel or multi-dimensional signals. Examples of multi-channel separation include [76, 81, 80], and image decomposition is an example of multi-dimensional signal separation [21]. In Section 3.2 and for the remainder of the thesis, we formulate the problem where there is a single channel, 1-dimensional input. This is followed by Section 3.3 which explores existing methods in the literature that tackle this problem.

3.2 Formulating the Problem

Formulating the signal decomposition problem can be as challenging as the act of solving it. For single channel, 1-dimensional signal decomposition, the goal is to recover a set of sources or modes $\{s_k(t)\}_{k=1}^K$ from an input mixture $y : \mathbb{R} \longrightarrow \mathbb{R}$ defined as

$$y(t) = \sum_{k=1}^{K} s_k(t) + \epsilon(t), \tag{3.1}$$

often in the presence of noise $\epsilon(t)$. For numerical computations, this problem is discretized by sampling the input $y(t)$ at $\{t_i\}_{i=1}^m \subset \mathbb{R}$ to obtain $\boldsymbol{y} \in \mathbb{R}^m$ with the goal of finding suitable vectors $\{\boldsymbol{s}_k\}_{k=1}^K \subset \mathbb{R}^m$ where

$$\boldsymbol{y} = \sum_{k=1}^{K} \boldsymbol{s}_k + \boldsymbol{\epsilon}, \tag{3.2}$$

and $\|\boldsymbol{\epsilon}\|_2$ is small. Signal decomposition is an inverse problem in the sense that the "forward" problem of calculating a mixture $\boldsymbol{y}$ from a set of sources $\{\boldsymbol{s}_k\}_{k=1}^K$ is as straightforward as summing the source vectors, but the "inverse" is much more challenging and ill-posed without carefully defining the problem.

Stating this explicitly, a mixture vector $\boldsymbol{y}$ can be broken into the sum of two vectors $\boldsymbol{y} = \boldsymbol{a} + \boldsymbol{b}$ in infinitely many ways. This issue is worsened if the number of sources K is also not known. These issues are addressed in different ways depending on the method used and domain of application. However, the notion of a "source" can change even within a specific domain. In musical decomposition, should a source be given by an section of similar instruments like the strings? Or a particular harmony line the violins play within the full chord? Or a single violin playing this line? Chapter 5 explores the assumptions used in the case of musical signals.

A popular assumption—and the assumption made in this thesis—is to assume each source $s_k(t)$ is an intrinsic mode function (IMF) (see Definition 15). This allows methods to take advantage of different properties these functions have such as having a single well defined instantaneous frequency and is used in [13, 16, 34, 73, 78]. We motivate this assumption by considering real-world signals such as seismic [77] and medical [69] which can be well modelled by IMFs.

Assuming the sources are IMFs may not always lead to a consistent decomposition, though. Consider the example given in Equation 1.3 of [13]:

$$\begin{aligned} y(t) &= 0.25\cos\left((\omega-\gamma)t\right) + 2.5\cos\left(\omega t\right) + 0.25\cos\left((\omega+\gamma)t\right) && (3.3)\\ &= 2\cos\left(\omega t\right) + \cos^2\left(\frac{\gamma}{2}t\right)\cos\left(\omega t\right) && (3.4)\\ &= \left(2+\cos^2\left(\frac{\gamma}{2}t\right)\right)\cos\left(\omega t\right). && (3.5) \end{aligned}$$

If ω is much larger than γ, we may prefer the last line (3.5) and say there is only one mode with instantaneous amplitude $2+\cos^2\left(\frac{\gamma}{2}t\right)$ and frequency ω. When ω and γ are of similar scale, then the first decomposition on line 3.3 may be desired and the three frequencies $\omega-\gamma$, ω, and $\omega+\gamma$ identified. Different methods deal with this ambiguity differently. Methods like EMD follow the aforementioned preference based on the relative scales of ω and γ [13], whereas methods like STFT filtering and the novel algorithm introduced in Chapter 4 prefer three distinct pure tones for large window sizes and a single IMF for small window sizes.

3.3 Previous Methods

We now recall popular methods used for signal decomposition and highlight their strengths and weaknesses.

3.3.1 Fourier Filtering and Masking

Fourier filtering considers the case where there is information known about the frequency ranges of its modes. More generally when masking the STFT of a signal, the modes must occupy distinct regions in time-frequency space. Let $Y(\omega,\tau)$ be the STFT of an input signal $y(t)$. Through visualization, such plotting a spectrogram $|Y(\omega,\tau)|^2$, or other means, the regions $M_1,\ldots,M_K\subset\mathbb{R}^2$ corresponding to each mode are identified. The transform of the sources S_k can be constructed by zeroing the domain not part of M_k

$$S_k(\omega,\tau)=\begin{cases} Y(\omega,\tau), & \text{for } (\omega,\tau)\in M_k\\ 0, & \text{otherwise.}\end{cases}$$

The sources $s_k(t)$ are then recovered by taking the inverse STFT of $S_K(\omega,\tau)$. This method can be effective for interactively identifying the regions of interest and extracting each mode. Automating the STFT masking can be performed if the modes always fall within the same regions. Otherwise, on its own without additional information about the sources, the regions corresponding to each mode must be manually found.

3.3.2 Empirical Mode Decomposition

The Empirical Mode Decomposition (EMD) [34] is an adaptive time-frequency method for analyzing and decomposing signals and has been shown to be applicable in a wide range of signal processing applications. In particular, EMD decomposes a given signal into IMFs by detecting local extrema and estimating upper and lower envelops. This effectively partitions the spectrum into certain frequency bands, which is represented by the learned IMFs.

It is important to note the algorithm uses a slightly less restrictive definition of IMFs than the one given in Definition 15, however all IMFs according to Definition 15 satisfy the following 2 criteria [13]. EMD separates a discretized signal $\boldsymbol{y} \in \mathbb{R}^m$ into modes where

1. There are the same number of extrema as there are zero crossings (or they differ by one).
2. The average of the envelopes defined by the local maximums and minimums is the zero function.

Pseudocode for the EMD method is given in Algorithm 2. In the algorithm, we use the definition of standard deviation between consecutive iterations:

$$\mathrm{SD}_{k,j} = \sum_{i=1}^{m} \frac{|\left(\boldsymbol{h}_{k,(j-1)}\right)_i - \left(\boldsymbol{h}_{k,j}\right)_i|^2}{\left(\boldsymbol{h}_{k,(j-1)}\right)_i^2}. \tag{3.6}$$

Common criteria for the outer loop in the algorithm includes stopping once either $\|\boldsymbol{s}_k\|_2$ or $\|\boldsymbol{r}_k\|_2$ are below a given threshold, or until $\boldsymbol{r}$ becomes monotonic. Additionally, the final residual $\boldsymbol{r}_k$ is often given as the final mode $\boldsymbol{s}_{K+1}$ for completeness.

EMD suffers from some problems including “mode mixing” i.e. the appearance of similar frequency information shared between distinct IMFs. The method is also heavily reliant on constructing the extrema envelopes with cubic splines which leads to a high sensitivity to noise and sampling. Improvements such as Ensemble EMD (EEMD) and Complete EEMD with Adaptive Noise (CEEMDAN) have been developed to help alleviate these issues.

Ensemble EMD

The Ensemble EMD (EEMD) method [78] learns the IMFs using an ensemble of the given signal perturbed by random (Gaussian) noise. This helps to mitigate the mode mixing issue by leverage results on EMD applied to white noise [22]; however, the approximated signals often retain aspects of the noise and the perturbations may lead to a different IMF decomposition.

Algorithm 2 Empirical Mode Decomposition (EMD)

```
Input: $\boldsymbol{y} \in \mathbb{R}^m$, tolerance $\in \mathbb{R}_{>0}$
Method:
    let $\boldsymbol{h}_{1,1} = \boldsymbol{y}$
    let $\boldsymbol{r}_1 = \boldsymbol{y}$
    let $k = 1$
    while not converged:
        let $j = 1$
        while $\mathrm{SD}_{k,j} >$ tolerance :
            find local extrema of $\boldsymbol{h}_{k,j}$
            construct cubic splines of minimums $\boldsymbol{a}_{k,j}$ and maximums $\boldsymbol{b}_{k,j}$
            let $\boldsymbol{h}_{k,j+1} = \boldsymbol{h}_{k,j} - \frac{\boldsymbol{a}_{k,j} + \boldsymbol{b}_{k,j}}{2}$
            update $j \leftarrow j + 1$
        let $\boldsymbol{s}_k = \boldsymbol{h}_{k,j}$
        let $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \boldsymbol{s}_k$
        let $\boldsymbol{h}_{k+1,1} = \boldsymbol{r}_{k+1}$
        update $k \leftarrow k + 1$

    Output: $K$ modes $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_K$
```

Complete EEMD with Adaptive Noise

In [73], the Complete EEMD with Adaptive Noise (CEEMDAN) method added different (synthetic) noise to the stages of the decomposition which lead to more stable results. Both EEMD and CEEMDAN suffer from increased computational costs since several applications of EMD must be performed.

3.3.3 Empirical Wavelet Transform

The Empirical Wavelet Transform (EWT) [25] combines aspects of EMD with the CWT. The main idea behind EWT is to partition the Fourier domain and build empirical wavelet filters from the segmented spectrum. This is done by identifying local maxima of the amplitude in the Fourier domain and partitioning the regions to separate the maxima [44]. The EWT is extended to two dimensions for applications in imaging and can be related to other wavelet-like transforms [24].

We define the sources EWT extracts from the mixture $y(t)$ by

$$s_{k+1}(t) = \int_{-\infty}^{\infty}\int_{\infty}^{\infty} y(\tau')\overline{\psi_k(\tau'-\tau)}\psi_k(\tau-t)\mathrm{d}\tau'\mathrm{d}\tau, \tag{3.7}$$

where the wavelets ψ_k, $k = 0, 1, \ldots, K-1$ are defined by their Fourier transforms

$$\hat{\psi}_0(\omega) = \begin{cases} 1, & \text{if } |\omega| \leq (1-\gamma)\omega_1 \\ \alpha_1(\omega), & \text{if } (1-\gamma)\omega_1 \leq |\omega| \leq (1+\gamma)\omega_1 \\ 0, & \text{otherwise,} \end{cases}$$

and

$$\hat{\psi}_k(\omega) = \begin{cases} 1, & \text{if } (1+\gamma)\omega_k \leq |\omega| \leq (1-\gamma)\omega_{k+1} \\ \alpha_{k+1}(\omega), & \text{if } (1-\gamma)\omega_{k+1} \leq |\omega| \leq (1+\gamma)\omega_{k+1} \\ \alpha'_k(\omega), & \text{if } (1-\gamma)\omega_k \leq |\omega| \leq (1+\gamma)\omega_k \\ 0, & \text{otherwise.} \end{cases}$$

The frequencies ω_k are chosen empirically, and $\gamma < \min_k \left(\frac{\omega_{k+1}-\omega_k}{\omega_{k+1}+\omega_k}\right)$. The functions α_k, α'_k are smooth transitions connecting 0 to 1, and appropriate choices are described in [25]. These wavelets can be thought of as bandpass filters which allow only frequencies between $[\omega_k, \omega_{k+1}]$ through.

To select ω_k, we first calculate the location of the maxima $\omega'_1 < \cdots < \omega'_M \in \mathbb{R}_{>0}$, of the Fourier transform magnitude of the signal $|\hat{y}(\omega)|$. If there are too many detected $(M \geq K)$, only the largest $N-1$ maxima are considered. If there are not enough maxima detected, then only $K = M$ modes are returned. We add the points $\omega'_0 = 0$ and the Nyquist frequency $\omega'_{N+1} = \frac{m}{2}$, where m is the sample rate, to the collection of maxima. Finally, the cutoff frequencies $\{\omega_k\}_{k=1}^{K-1}$ are defined as the midpoints $\omega_k = \frac{\omega'_k - \omega'_{k-1}}{2}$.

Unlike EMD, the number of modes K must be given in advance which may not be known in some applications. The extracted modes are also restricted to disjoint frequency ranges that are constant across the entire signal. This means EWT would have trouble extracting IMFs that share the same instantaneous frequency, even if this occurs at different times in the signal and they don't cross.

3.3.4 Variational Mode Decomposition

The Variational Mode Decomposition (VDM) method [16] decomposes the signal into a sum of IMFs (see Definition 15) by solving the optimization problem

$$\begin{aligned} \min_{\substack{s_1,\ldots,s_K \\ \omega_1,\ldots,\omega_K}} \alpha \sum_{k=1}^{K} \left\| \frac{\partial}{\partial t} \left[\left(\delta(t) + \frac{i}{\pi t} \right) * s_k(t) \right] e^{-i\omega_k t} \right\|_2^2 \\ + \left\| y(t) - \sum_{k=1}^{K} s_k(t) \right\|_2^2 + \left\langle \lambda(t), y(t) - \sum_{k=1}^{K} s_k(t) \right\rangle \end{aligned} \tag{3.8}$$

using the split Bregman or ADMM method [26]. Here, $\omega_k \in \mathbb{R}$ represents the center frequency of $s_k(t)$, $\alpha \in \mathbb{R}_{>0}$ is the hyperparameter responsible for weighting the smoothness of the modes, $\delta(t)$ is the Dirac delta function, and $\lambda(t)$ is the Lagrange multiplier.

To unpack the optimization problem, the idea is to find modes $s_k(t)$ that have their range of frequencies tightly centered around ω_k. To achieve this, the convolution removes the negative frequencies, and the complex exponential centers the frequency range around zero. In order to have small frequency bandwidths, we would find shifted modes that are as smooth as possible. In other words, we wish to minimize the L^2-norm of the gradient. The second and third term encourage the sum of the modes to be close to the input $y(t)$.

Rather than decompose the input iteratively like in EMD, the IMFs are obtained simultaneously within the optimization process and the resulting decompositions are more stable to noise than the standard EMD approaches. Similar to EWT, VMD requires the number of modes to be given in advance which may not always be known.

3.3.5 Synchrosqueezing Transforms

The synchrosqueezing transform (SST) [13] improves the standard CWT by calculating instantaneous frequencies and "squeezing" them through a reassigment algorithm, namely, shift them to the center of the time-frequency region [1]. This leads to sharper time-frequency representations than the STFT and CWT, which are often limited by the finite sampling lengths and can create spectral smearing. In addition, the sharpening essentially prunes the unnecessary wavelet coefficient, thus leading to a sparse representation. Other SST based methods have been proposed using other signal transforms for example the STFT [71], S-transform [35], and the wavelet packet transform [79].

We define the SST [13] based on the CWT by

$$T(\omega, b) = \int_{A(b)} F_\psi(a,b) a^{-\frac{3}{2}} \delta(\mathit{\Omega}(a,b) - \omega)\, \mathrm{d}a, \tag{3.9}$$

where $A(b) = \{a \in \mathbb{R}_{>0} \mid |F_\psi(a,b)| \geq \epsilon\}$, $F_\psi(a,b)$ is the CWT of our input y with respect to the wavelet ψ, $\epsilon \in \mathbb{R}_{>0}$ is some small noise threshold, $\delta(\omega)$ is the Dirac delta function, and the instantaneous frequency $\mathit{\Omega}(a,b)$ is defined by

$$\mathit{\Omega}(a,b) = -\frac{i}{F_\psi(a,b)} \frac{\partial F_\psi(a,b)}{\partial b}. \tag{3.10}$$

A similar transform can be defined by using the STFT rather than CWT [71].

To decompose a signal with syncrosqueezing, bands can extracted around the largest instantaneous frequencies curves $\mathit{\Omega}(a,b)$ calculated in the transform and the masked transform inverted. This is an improvement over the standard Fourier Masking technique since the squeezing ensures most of the important information is centered around the IF curves and making it less sensitive to the exact region selected. There are no guarantees these regions are disjoint, in which case the sum of the extracted modes would not reconstruct the original input signal.

3.3.6 Neural Network Methods

Neural Networks have been used in a wide variety of scientific domains and industrial applications from image classification and differential equations to recommendation systems and generative models. They have shown remarkable success as function approximators and the neural network framework is well suited for the training and learning of these functions.

A common way to create a neural network model involves supervised training and some penalty or loss function to assess the performance of the model. The network's architecture—number of layers, hidden nodes, activation functions, and how they are connected—is often fixed before training and parametrized by a set of learnable weights $\boldsymbol{W}_l$ and biases $\boldsymbol{b}_l$. In supervised training, a data set of known input-output pairs for a particular process is collected. In the case of source separation, this could look like $\left\{(\boldsymbol{y}_n, \{\boldsymbol{s}_{k,n}\}_{k=1}^{K_n})\right\}_{n=1}^{N}$. It is the network's goal to be able to receive an input and reproduce the output. The predicted output is compared to the known ground truth output. If the model performs this task well, the loss function will be small, whereas poor performance will yield a large value.

There are many typical architectures for neural networks. A standard fully connected network takes the form of alternating affine and non-linear activation functions

$$F_\theta(\boldsymbol{y}) = \boldsymbol{W}_L\left(\sigma\left(\cdots\sigma(\boldsymbol{W}_1\boldsymbol{y} + \boldsymbol{b}_1)\cdots\right)\right) + \boldsymbol{b}_L, \tag{3.11}$$

where L is the number of layers of the neural network, σ is the activation function, and $\theta = \{(\boldsymbol{W}_l, \boldsymbol{b}_l)\}_{l=1}^{L}$ are trainable weights and biases. Common activations include the hyperbolic tangent $\tanh(x)$, sigmoid $(1+e^{-x})^{-1}$, and rectified linear unit $\max(0,x)$ functions. In the context of source separation, the output $\boldsymbol{s} = F_\theta(\boldsymbol{y})$ could represent the concatenation of the sources $\boldsymbol{s} = \left[\boldsymbol{s}_1{}^\top \mid \cdots \mid \boldsymbol{s}_K{}^\top\right]^\top$. Convolutional networks include convolution layers which are linear layers where the weight matrix $\boldsymbol{W}$ has a block diagonal structure.

Since neural network methods are trained on a particular data set, they are best suited to the domain on which the specific architecture was designed and trained. To this end, we list the popular architectures for a variety of applications. DeepDenoiser [82] used on seismic data, U-Net for biomedical imaging [63] and vocal separation [36] all use some form of convolutional layers to create masks on the STFT of the input automating the process described in Section 3.3.1. These methods require a fixed sized input, so signals that are too short must be padded or repeated, and signals that are too long must be cropped into sub-signals of equal length. *Long short-term memory* networks have layers which feedback into previous parts of the network and typically require only a small chunk of the input at a time enabling them to be naturally applied to signals of varying length. These have been applied on their own in musical decomposition [68] and in conjunction with non neural network decomposition methods such as CEEMDAN in financial forecasting [10]. There are also models that operate directly on the signal without relying on the STFT such as TasNet for speech separation [45]. Finally, an autoencoder—a network with compressive layers followed by expansive ones—is used for audio source separation in [37] and is adapted for vocal separation in Section 5.3.

Chapter 4

Sparse Random Mode Decomposition

4.1 Overview

We now present the main contribution of the thesis: the Sparse Random Mode Decomposition (SRMD) algorithm for the signal decomposition problem detailed in Section 3.2. This method builds off the previous methods presented in 3.3 while introducing some novel techniques to the decomposition problem.

High level motivation for the method is provided in Section 4.2. We present the SRMD algorithm in Section 4.3 and apply it to a variety of mathematical examples in Section 4.4. Additional extensions and modifications of the method for different settings are highlighted in Section 4.5. We conclude this chapter with the method's limitations and possible refinement to be useful in a wider variety of applications in Section 4.6.

4.2 Motivation

This method considers the case where the composite signal can be well represented by a sum of intrinsic mode functions (IMFs) as given in Definition 15. The instantaneous frequencies (IF) of these modes appear as curves $\omega = \phi(\tau)$ on a STFT spectrogram $|F_W(\tau, \omega)|^2$. Unlike manually filtering and masking (see Section 3.3.1) which requires a lot of knowledge of the signal's modes *a priori*, we wish to automate this extraction for generic IMFs.

Finding structure within data is problem that is often tackled by clustering [65]. The issue with clustering directly on sampled time-frequency pairs (τ_j, ω_j) in a discrete STFT is that the points are dense and evenly spaced. Clustering algorithms would therefore always identify a single cluster in the whole time-frequency plane. Clustering in $\mathbb{R}^3$ using a dataset of triples (τ_j, ω_j, c_j) with $c_j = |F_W(\tau_j, \omega_j)|$ could be performed, but would require finding suitable scaling factors in all three dimensions. We could throw out pairs with a small STFT magnitude so only the large coefficients corresponding to IF curves remain. This would involve choosing an appropriate threshold that is simultaneously above the noise level and below the lowest-amplitude mode. Selecting this threshold may be impossible under high levels of noise, or when the noise level is above the lowest-amplitude mode. We also cannot guarantee the accuracy of the reconstruction if the extracted modes where to be summed up and compared to the input signal.

To rectify these issues, we would like a sparse time-frequency representation that can ensure some level of reconstruction accuracy. We motivate the introduction of generating time-frequency localized features and finding a sparse representation of the input signal in these features. These could be generated on a time-frequency grid, but we found randomly generating the (τ_j, ω_j) pairs allows for sparser representations, and one that is equally fair to constant IFs as well as chirps or other changing IF curves. The effect of equally spaced versus random features is explored in Section 4.5.2.

Armed with an appropriately sparse time-frequency representation, we can use a clustering algorithm to identify these curves. We would like a method that does not rely on knowing the number of clusters, and one that works well on arbitrarily-shaped clusters rather than ones localized around a single point. We found DBSCAN [19] to be sufficient but other density-related clustering methods like OPTICS or HDBSCAN could be used. In fact, since density clustering is used, we can generalize the class of modes SRMD can extract to any modes that are connected in their time-frequency representation and in disjoint regions from each other.

4.3 Method

The proposed method builds from the continuous STFT, that is, we represent a signal $f \in L^1(\mathbb{R}) \cap L^2(\mathbb{R})$ by

$$f(t) = \int_{-\infty}^{\infty} f(t) W(t-\tau) \mathrm{d}\tau = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F_W(\omega, \tau)\, W(t-\tau)\, e^{2\pi i \omega t}\, \mathrm{d}\omega \mathrm{d}\tau, \tag{4.1}$$

where F_W is the transform function and W is a (positive) window function such that $\int_{-\infty}^{\infty} W(t-\tau) d\tau = 1$. The assumptions are that the transform function is band-limited, i.e. $F_W(\omega,\tau) = 0$ for all $|\omega| > B$ and that for a fixed τ, the support of $F_W(\omega,\tau)$ is small. Note that since $f \in L^1(\mathbb{R})$, the transform is bounded, i.e. there exists an $M > 0$ such that $|F_W(\omega,\tau)| \leq M$ for all (ω,τ). Using the RFM, we approximate the integrals using $N = N_1 N_2$ random features:

$$f(t) = \int_{-\infty}^{\infty} \int_{-B}^{B} F_W(\omega,\tau)\, W(t-\tau)\, e^{2\pi i \omega t}\, \mathrm{d}\omega \mathrm{d}\tau \approx \sum_{k=1}^{N_2} \sum_{n=1}^{N_1} c_{n,k}\, W(t-\tau_k)\, e^{2\pi i \omega_n t} \tag{4.2}$$

where $\{\omega_n\}_{n\in[N_1]}$ and $\{\tau_k\}_{k\in[N_2]}$ are independent of each other and are drawn i.i.d. $\omega_n \sim \mathcal{U}[0,B]$ and $\tau_k \sim \mathcal{U}[0,T]$. The goal is to learn a representation of the target signal f using m sampling points $\{t_\ell\}_{\ell\in[m]} \subset [0,T]$. The sampling points can either be equally-spaced in time or can be drawn i.i.d. from a probability measure $\mu(t)$ along the interval $[0,T]$. The given output measurements are $y_\ell = f(t_\ell) + e_\ell$ where the noise or outliers $\{e_\ell\}_{\ell\in[m]}$ are either bounded by constant $E > 0$, i.e. $|e_\ell| \leq E$ for all $\ell \in [m]$, or are random Gaussian. Note that if $e_\ell \sim \mathcal{N}(0,\sigma^2)$ and if $m \geq 2\log(\delta^{-1})$, then the noise terms e_ℓ are bounded by $E = 2\sigma$ for all $\ell \in [m]$ with probability exceeding $1-\delta$.

We reindex (τ_n, ω_k) to (τ_j, ω_j) where $j \in [N_1 N_2] = [N]$ so that the N random feature functions take the form

$$\phi_j(t) := W(t-\tau_j)\, e^{2\pi i \omega_j t}. \tag{4.3}$$

Note this means the same value of τ^* could be paired with different values of ω_j in the set of all $\{(\tau_j,\omega_j)\}_{j\in[N]}$ and vice versa. Thus, the approximation becomes

$$f(t) \approx \sum_{j=1}^{N} c_j\, \phi_j(t), \tag{4.4}$$

where the coefficients c_j have also been reindexed. The training problem becomes learning coefficients c_j so the approximation $\sum_{j=1}^{N} c_j\, \phi_j(t)$ is close to the given data y_ℓ. Let $\boldsymbol{A} \in \mathbb{C}^{m\times N}$ be the random feature matrix whose elements are define as $a_{\ell,j} = \phi_j(t_\ell)$, $\boldsymbol{c} = [c_1,\ldots,c_N]^T$ and $\boldsymbol{y} = [y_1,\ldots,y_m]^T$. We wish to find a sparse time-frequency representation, so we learn $\boldsymbol{c}$ by solving an ℓ^1 regularized least squares problem. Following [30], a sparse random feature model can be trained with the ℓ^1 basis pursuit denoising problem [9, 8, 23]:

$$\boldsymbol{c}^\sharp = \min_{\boldsymbol{c}\in\mathbb{C}^N} \; \|\boldsymbol{c}\|_1 \quad \text{s.t.} \quad \|\boldsymbol{A}\boldsymbol{c} - \boldsymbol{y}\|_2 \leq \eta\sqrt{m}, \tag{4.5}$$

where η is a user-defined parameter that is related to the noise bound E. It can be shown that certain random feature matrices are well-conditioned to sparse regression when trained using model (4.5) [30, 12]. When the input data is contaminated by large noise where the noise level is not known (such as the astronomical data in Section 4.5.1), we solve the LASSO optimization problem [72, 31]:

$$\boldsymbol{c}^\sharp = \min_{\boldsymbol{c}\in\mathbb{C}^N} \|\boldsymbol{A}\boldsymbol{c} - \boldsymbol{y}\|_2^2 \quad \text{s.t.} \quad \|\boldsymbol{c}\|_1 \leq \tau. \tag{4.6}$$

Note that the LASSO optimization problem is equivalent to the basis pursuit denoising problem under a mapping between $\tau > 0$ and $\eta > 0$ (see Section 2.2). The LASSO formulation becomes favourable over BPDN if we do not know a reasonable error bound η. Algorithms such as SPGL1 solve a series of LASSO problems to find the solution to BPDN, so manually tuning τ is faster than η in this case.

4.3.1 Sparse Random Feature Representation Algorithm

In the algorithm, we replace the complex exponential by a sine function with a random phase ψ_j and the window function is defined by the Gaussian with a fixed variance w^2:

$$\phi_j(t) = \exp\left(-\frac{(t-\tau_j)^2}{2w^2}\right)\sin(2\pi\omega_j\, t + \psi_j), \tag{4.7}$$

where $\tau_j \in \mathcal{U}(0,T), \omega_j \in \mathcal{U}(0,\omega_{\max})$, and $\psi_j \in \mathcal{U}(0,2\pi)$.

Given a set of time points $\{t_\ell\}_{\ell\in[m]}$, we define the random short time Fourier feature matrix $\boldsymbol{A} = [a_{\ell,j}] \in \mathbb{R}^{m\times N}$ as follows:

$$a_{\ell,j} = \phi_j(t_\ell) = \exp\left(-\frac{(t_\ell-\tau_j)^2}{2w^2}\right)\sin(2\pi\omega_j t_\ell + \psi_j). \tag{4.8}$$

While the standard approaches assume that data is obtained from an evenly spaced time-series, we do not place any restrictions on the sampling of the time points $\{t_\ell\}_{\ell\in[m]}$ (except that they are distinct). This is an important distinction compared to other signal decomposition approaches. In particular, we optimize the coefficients $\boldsymbol{c}$ using a sparse optimization problem, Equation (4.5), with the random short-time sinusoidal feature matrix, which can be shown to lead to well-conditioned training even when m is small [30, 12]. As an added benefit, the random sampling of time-points reduces the computational and storage cost, which depend on the number of samples and number of features. The reconstruction algorithm is summarized in Algorithm 3.

Algorithm 3 Sparse Random Feature Representation for a Time-Series Data

Input: Samples $\{(t_\ell, y_\ell)\}_{\ell=1}^m$, number of random features N, maximum frequency $\omega_{\max}$, window size w, noise level $r \in [0,1]$. Let $\boldsymbol{y} = [y_1, \dots, y_m]^T$.

Method:

Draw random time shifts, frequencies, and phases:

$$\{\tau_j, \omega_j, \psi_j\}_{j=1}^N \sim \mathcal{U}(0,T) \times \mathcal{U}(0, \omega_{\max}) \times \mathcal{U}(0, 2\pi)$$

Construct the random short time Fourier feature matrix

$$\boldsymbol{A} = [\phi_j(t_\ell)] = \left[\exp\left(-\frac{(t_\ell - \tau_j)^2}{2w^2}\right) \sin(2\pi\omega_j t_\ell + \psi_j)\right] \in \mathbb{R}^{m \times N}.$$

Solve: $\boldsymbol{c}^\sharp = \underset{\boldsymbol{c} \in \mathbb{R}^N}{\arg\min} \|\boldsymbol{c}\|_1 \quad \text{s.t. } \|\boldsymbol{A}\boldsymbol{c} - \boldsymbol{y}\|_2 < \sigma = r\|\boldsymbol{y}\|_2.$

Output: Coefficient vector $\boldsymbol{c}^\sharp$ and the sparse random feature representation for the time-series:

$$f^\sharp(t) = \sum_{j=1}^N c_j^\sharp \phi_j(t).$$

Algorithm 4 Sparse Time-Frequency Decomposition for a Time-Series Data

Input: Time-Frequency pairs $X = \{(\tau_j, \omega_j) \mid j \in [N],\ c_j^\sharp \neq 0\}$, their corresponding coefficients and features $\{(c_j^\sharp, \phi_j(t)) \mid j \in [N],\ c_j^\sharp \neq 0\}$, $\texttt{frq\_scale} \in \mathbb{R}_{>0}$, DBSCAN hyperparameters ε and `min_samples`. Let $\boldsymbol{y} = [y_1, \dots, y_m]^T$.

Method:

Scale input points to obtain $\widetilde{X} = \{(\tau_j, \widetilde{\omega}_j) \mid j \in [N], c_j^\sharp \neq 0, \widetilde{\omega}_j = \omega_j(\texttt{frq\_scale})\}$.

Partion $\widetilde{X}$ into clusters $C_1, \dots, C_K$ using DBSCAN. Let

$$I_k = \{j \in [N] \mid (\tau_j, \widetilde{\omega}_j) \in C_k\}, \quad k = 1, \dots K.$$

Output: K modes

$$s_k(t) := \sum_{j \in I_k} c_j^\sharp \phi_j(t), \quad k = 1, \dots K.$$

4.3.2 Sparse Random Mode Decomposition Algorithm

In this section, we discuss how to utilize the learned coefficients $\boldsymbol{c}^\sharp$ to decompose the signal into meaningful modes. It is based on the observation that the sparse optimization extracts a sparse time-frequency representation, which has the added benefit of forming a simple decomposition due to the sharpening of the spectrogram. Specifically, we first collect all time shift-frequency pairs $\{\tau_j, \omega_j\}$ corresponding to the non-zero learned coefficient $c_j^\sharp$. Denote

$$X := \{(\tau_j, \omega_j) \mid j \in [N], c_j^\sharp \neq 0\}. \tag{4.9}$$

We partition X into clusters $\{C_k\}_{k=1}^K$ using a popular clustering method—namely DBSCAN (see Section 2.3). Lastly, the learned coefficients are grouped based on those clusters and used to define the corresponding mode. The main advantages of DBSCAN are that we do not need to specify the number of clusters, and can cluster arbitrary shaped regions. These properties are useful for the signal decomposition problem since we do not assume we know the number of modes, nor how the IF curves are shaped. In Section 4.4, we discuss how to merge modes together if the number of modes is known. While more sophisticated clustering algorithms can be used, one of the benefits of SRMD is that it produces a sparse spectrogram which can be more easily clustered by any algorithm provided the original signal has a sharp time-frequency separation. This decomposition algorithm is summarized in Algorithm 4.

Finally, we combine the representation and decomposition Algorithms 3 and 4 into the proposed SRMD method summarized in Algorithm 5.

4.3.3 Implementation

For all examples of this method shown later in this chapter and in section 5.2, a Python implementation of SRMD is used. An example of this implementation is shown in appendix B and the full code for these examples are available on GitHub.[1]

The SRMD method relies on the ability to solve the BPDN minimization problem presented in Algorithm 3 and partition data points using DBSCAN as shown in Algorithm 4. The BPDN problem is solved using a Python port [15] of the original SPGL1 method [74, 75]. The DBSCAN algorithm is performed by the scikit-learn's implementation [55] of the original method presented in [19].

[1]`https://github.com/GiangTTran/SparseRandomModeDecomposition`

Algorithm 5 Sparse Random Mode Decomposition (SRMD) for a Time-Series Data

```
Input: Samples ${(t_\ell, y_\ell)}_{\ell=1}^m$, number of random features $N$, maximum frequency $\omega_{\max}$,
    window size $w$, noise level $r \in [0,1]$, frq_scale $\in \mathbb{R}_{>0}$, DBSCAN hyperparameters $\varepsilon$
    and min_samples. Let $\boldsymbol{y} = [y_1, \ldots, y_m]^T$.
Method:
    Draw random time shifts, frequencies, and phases:
        $\{\tau_j, \omega_j, \psi_j\}_{j=1}^N \sim \mathcal{U}(0,T) \times \mathcal{U}(0,\omega_{\max}) \times \mathcal{U}(0,2\pi)$
    Construct the random short time Fourier feature matrix
        $\boldsymbol{A} = [\phi_j(t_\ell)] = \left[\exp\left(-\frac{(t_\ell - \tau_j)^2}{2w^2}\right) \sin(2\pi\omega_j t_\ell + \psi_j)\right] \in \mathbb{R}^{m \times N}.$
    Solve: $\boldsymbol{c}^\sharp = \arg\min_{\boldsymbol{c} \in \mathbb{R}^N} \|\boldsymbol{c}\|_1 \quad \text{s.t. } \|\boldsymbol{A}\boldsymbol{c} - \boldsymbol{y}\|_2 < \sigma = r\|\boldsymbol{y}\|_2.$
    Obtain coefficient vector $\boldsymbol{c}^\sharp$.
    Collect $X = \{(\tau_j, \omega_j) \mid j \in [N], c_j^\sharp \neq 0\}$.
    Scale input points to obtain $\widetilde{X} = \{(\tau_j, \widetilde{\omega}_j) \mid j \in [N], c_j^\sharp \neq 0, \widetilde{\omega}_j = \omega_j(\texttt{frq\_scale})\}$.
    Partion $\widetilde{X}$ into clusters $C_1, \ldots, C_K$ using DBSCAN. Let
        $I_k = \{j \in [N] \mid (\tau_j, \widetilde{\omega}_j) \in C_k\}, \quad k = 1, \ldots K.$
    Output: $K$ modes
        $s_k(t) := \sum_{j \in I_k} c_j^\sharp \phi_j(t), \quad k = 1, \ldots K.$
```

4.3.4 Hyperparameter Tuning

We acknowledge there are many hyperparameters involved in the implementation of SRMD. This section walks through each parameter and explains their recommended values, as well as some justification for their existence.

Parameter	**Recommended Value**	**Typical Range**
N	$10m$	$5m$–$50m$
$\omega_{\max}$	$m/(2T)$	0–$m/(2T)$
r	0.05	0.03–0.3
w	0.1	0.01–2
`threshold`	0.05	0–0.1
`frq_scale`	$T/\omega_{\max}$	$T/\omega_{\max}, 1$
ε	$0.1T$	$0.05T$–$2T$
`min_samples`	4	3, 4, 5

Table 4.1: **Summary of SRMD parameters from Section 4.3.4**. In this table, the number of samples is given by m and the length of the sample (in seconds) is given by T.

As a quick guide, the recommended values for most parameters given in Table 5.1 can be used in many settings. The main parameters that require tuning are the number of features N and the feature neighbourhood radius ε. The number of features can be set to 10 times the number of samples $N = 10m$ as a good starting point, and increased if the representation is poor. The neighbourhood ε can be set to 0.1 and increased if there are too many modes recovered, or decreased if there are too few.

Feature Generation. The largest possible frequency that can be generated $\omega_{\max} \in \mathbb{R}_{>0}$ (`max_frq` in the code) can be typically set to the Nyquist rate $\omega_{\max} = m/(2T)$ where T is time length of the signal. If the maximum frequency in the signal in known, this parameter can lowered to that frequency. The Gaussian window size $w \in \mathbb{R}_{>0}$ for the generated features can be set to 0.1 s as a good starting point, with a typical range being between 0.01–1 s. This can be made larger for better frequency localization, or smaller for better time localization. The number of features generated $N \in \mathbb{Z}^+$ (`N_features` in the code) may need to be tuned. Simple signals may only need $N = 5m$ where as complicated signals may need $N = 50m$. It is best to start small, say $N = 10m$, and increase the amount if the signal is not well represented. By this, we mean the SPGL1 algorithm will fail if too few features are generated since there will be no feasible solution $\boldsymbol{c}$ to the constraint $\|\boldsymbol{A}\boldsymbol{c} - \boldsymbol{y}\|_2 \leq r\|\boldsymbol{y}\|_2$.

Representation. The maximum relative error desired when representing the signal is given by $r \in (0, 1)$. By default, $r = 0.05$ is recommended. This can be brought down to as low as 0.01 in some settings if the signal in known to have very little noise, but may require more features N. If the noise level in the signal is known, r can be set slightly above this level. The parameter `threshold` $\in [0, 1)$ performs an optional fine-pruning step when it is bigger than 0. Features with coefficients in the bottom `threshold`-percentile of nonzero-coefficients are thrown from the representation. Setting this to 0.05 can help remove outliers and clean up the representation. In practice we find only 3–10 features are pruned for small values of `threshold` which leads to a cleaner representation and an easier time decomposing.

Decomposition. The scaling factor `frq_scale` $\in \mathbb{R}_{>0}$ is applied to the frequencies of the features before the clustering algorithm. By default, $T/\omega_{\max}$ should be used. This treats the time and frequency dimensions equally, and makes it easiest to perform the decomposition and visualization of the clusters. Smaller values help isolate modes in the time dimension, and larger values isolate modes in the frequency dimension. The clustering algorithm used, DBSCAN, requires two hyperparameters, $\varepsilon \in \mathbb{R}_{>0}$ and `min_samples` $\in \mathbb{Z}_{>0}$. The first parameter ε is the radius of a feature's neighbourhood in time-frequency space. Appropriate values are highly dependent on the structure of the modes in time-frequency space. Bigger ε means fewer modes are identified. When `frq_scale` is set to the default $T/\omega_{\max}$, $\varepsilon = 0.2T$ is a good starting point. Values on the order 1–5 work better when `frq_scale` is set to 1. The second parameter `min_samples` is the number of features in a neighbourhood required to be considered a core point. 4 is a reasonable default value and works well in most settings. This can be set to 3 if too many outliers are found, or set to 5 or larger if there are too many modes found.

4.4 Examples

In this section, we verify the applicability and consistency of our proposed SRMD method on four decomposition and signal representation examples, including three challenging synthetic time-series from [13, 16] and an overlapping time-series. Additionally, two real time-series are tested in later sections including a gravitational signal (Section 4.5.1) and a musical example (Section 5.2.1). In all experiments, we plot the learned coefficients $c_j^\sharp$ obtained from Algorithm 3 on the time-frequency (τ, ω) space and plot the corresponding modal decomposition from Algorithm 4 (i.e. clusters) using DBSCAN.

For the three synthetic signals, we compare our approach with some of the state-of-the-art mode decomposition methods, including the EMD [34], Ensemble EMD (EEMD)

[78], Complete EEMD with Adaptive Noise (CEEMDAN) [73], Empirical Wavelet Transform (EWT) [25], and Variational Mode Decomposition (VMD) [16]. Additionally, for the examples in Section 4.4.1 and 4.4.2, we also compare our method to the spectrogram produced by the short-time Fourier transform (STFT), continuous wavelet transform (CWT), and their Synchro-Squeezed transforms [71, 13]. We also investigate the robustness of our method's signal representation and its corresponding signal decomposition for time-series with noise. Sections 4.5 and 5.2 further explore the application of this method on real world data. All tests were performed using Python and our code is available on GitHub (see Section 4.3.3). PyEMD [41] is used to test EMD and the related methods, while ewtpy and vmdpy were used to test EWT and VMD, respectively [11]. The hyperparameters used in all methods are choosen to optimize their resulting output and based on the methods' suggestion. More precisely, for EMD, EEMD, and CEEMDAN, the threshold values on standard deviation, on energy ratio, and on scaled variance per IMF check are $\texttt{std\_thr} = 0.1$, $\texttt{energy\_ratio\_thr} = 0.1$, and $\texttt{svar\_thr} = 0.01$, respectively. For EEMD and CEEMDAN, the number of noise perturbed ensemble trials is set to $\texttt{trials} = 100$. For VMD, we set the balancing parameter of the data-fidelity constraint $\texttt{alpha} = 50$, the time-step of the dual ascent $\texttt{tau} = 1$, the convergence tolerance $\texttt{tol} = 10^{-6}$, and all frequencies ω are initialized randomly.

Regarding the hyperparemeters of our proposed SRMD, we need to choose $(\omega_{\max}, w, N)$ to generate the dictionary matrix $\boldsymbol{A}$, the noise-level parameter σ for the SPGL1 algorithm, and $(\texttt{frqscale}, \texttt{min\_samples}, \varepsilon)$ for the DBSCAN. We use the following values, unless stated otherwise. The maximum possible frequency $\omega_{\max}$ is set to be the Nyquist rate, $\omega_{\max} = \dfrac{m}{2T}$, the window size is fixed to be $w = 0.1$, and the number of generated features N can range from $5m$ to $50m$ depends on the complexity of the signal. The parameter of the SPGL1 is set to be $\eta\sqrt{m} = 0.06\|y_{\text{input}}\|_2$, yielding at most 6% reconstruction error. Also, since the time-points and the frequencies are at different scales, we divide all learned frequencies by $\texttt{frqscale}$, which is set by default to be $\texttt{frqscale} = \frac{T}{\omega_{\max}}$ before applying the clustering algorithm on the learned time-frequency pairs. In some experiments, we choose $\texttt{frqscale} = 1$ to obtain better clustering by frequency. The radius of a feature's neighborhood in time-frequency space ε is chosen depend on the structure of the modes in time-frequency space. A bigger ε means fewer modes are identified. When $\texttt{frqscale}$ is set to the default, $\varepsilon = 0.2T$ is a good starting point. Values on the order 1–5 work better when $\texttt{frqscale}$ is set to 1. Finally, the number of features in a neighborhood required to be considered a core point $\texttt{min\_samples} = 4$. If there are too many found modes, we can increase $\texttt{min\_samples} = 5$.

In EWT, VMD, and SRMD, the number of principal modes from the input signal are specified for each experiment. In most cases, SRMD identifies the correct number of modes without this restriction. EWT and VMD require the number of extracted modes to be given as parameters so this is expected. To force SRMD to return the desired number of modes when there are n more learned modes than true modes, the $n+1$ modes with the smallest ℓ^2-norm are merged together. This is appropriate when the number of true modes is known, in the case of our comparison, and it is expected they have similar ℓ^2-norm. If too few modes are extracted, ε is lowered until enough modes are found. To be fair to EMD and related methods (EEMD & CEEMDAN), extra modes are merged in the order they are extracted for these methods only. This avoids fine-tuning their hyper parameters which would otherwise need to be tuned to ensure the correct number of modes are extracted with their stopping criteria. This makes sense for these methods since they typically extract modes from high to low frequency, where additional modes may be large amplitude ultra-low frequency biases. Lastly, we plot the magnitude of the non-zero entries of the learned coefficient vector $c^\sharp$ on the time-frequency space. Note that all non-zero coefficients can be re-assigned to the positive value after shifting the phase of the corresponding basis term. This leads to a sparse spectrographic representation of the signal.

4.4.1 Discontinuous Time-Series

The first signal is a modified example from [13], where the input signal $y(t)$ is a composition of a linear trend $s_1(t)$, a pure harmonic signal $s_2(t)$, and a harmonic signal with a nonlinear instantaneous frequency $s_3(t)$:

$$\begin{aligned} s_1(t) &= \pi t \, 1_{[0,5/4)}(t) \\ s_2(t) &= \cos(40\pi t) \, 1_{[0,5/4)}(t) \\ s_3(t) &= \cos\left(\frac{4}{3}\left((2\pi t - 10)^3 - (2\pi - 10)^3\right) + 20\pi(t-1)\right) 1_{(1,2]}(t), \end{aligned} \tag{4.10}$$

and $y(t) = s_1(t) + s_2(t) + s_3(t)$ where $t \in [0,2]$ and $1_{\mathcal{I}}(\cdot)$ denotes the indicator function over the interval $\mathcal{I}$ where

$$1_{\mathcal{I}}(t) = \begin{cases} 1, & \text{for } t \in \mathcal{I} \\ 0, & \text{otherwise.} \end{cases} \tag{4.11}$$

Notice that the input signal has a sharp transition at $t = \frac{5}{4}$. The number of modes is fixed to be three for this experiment. The dataset contains $m = 320$ equally spaced time points from $[0,2]$ and the total number of random features for our algorithm is $N = 50m = 16000$ (overparameterized regime). When we apply the clustering algorithm DBSCAN in the time-frequency space, we set the minimum number of core points in a cluster to be three, and the maximum distance between any two points in a neighborhood is set to $\varepsilon = 0.1$.

Figure 4.1 shows that SRMD can decompose the discontinuous signal with minimal mode mixing, i.e. a clear separation between the learned modes. Specifically, the three learned modes are closest to the ground truth ones with the errors mainly appeared at the discontinuous time-point $t = \frac{5}{4}$ (see also Figure 4.2). Moreover, it may be possible to achieve even better mode decomposition with a choice of basis better suited to discontinuities (see Section 6.2). It is shown that the randomized Morlet like basis, whose wavelet equivalent is not well suited to discontinuities, is still sufficient for our proposed SRMD to outperform the existing methods. Moreover, the representation and clustering plots of Figure 4.1 show that the sparse spectrogram and modal decomposition are indeed sharp and well-separated. This is a better separation than the various STFT and CWT, and their synchro-squeezed versions shown in Figures 1, 2, and 6 of [13]. Additionally, our proposed SRMD has the advantage of locating the individual regions that define each mode in the time-frequency domain. This could be done with a synchro-squeezed version of STFT or CWT by extracting a thin region around the instantaneous frequency curves; however, the reconstruction accuracy is not guaranteed. Specifically, using that approach, the selected regions may exclude areas of the time-frequency domain that contain important information, or double count intersecting regions like in Equation (4.12).

In Figure 4.2, the input signal and the three true instantaneous frequencies are shown in black, while the extracted modes from our and the other five methods are shown in blue.

For comparison, EMD and CEEMDAN reconstruct the signal with machine precision (the error is on the order of 10^{-16}), VMD also displays a reconstruction error of only 0.4%, while EEMD and EWT have poor signal reconstruction. One possible reason that leads to a poor reconstructed signal for EEMD and EWT is because these methods do not guarantee the sum of reconstructed intrinsic modes equals the original signal. In terms of signal decomposition, the VMD and EWT extract the linear trend well (the second column of the last two rows in Figure 4.2) while their learned second and third modes agree with the corresponding ground truth ones on the time interval $t \in [0, 1]$ but create a false oscillatory patterns on the remaining interval $t \in [1, 2]$. This indicates that the decomposition produced by the EWT and VMD approaches experience a nontrivial amount of mode mixing. On the other hand, the remaining methods used in this comparison are unable to extract any modes that agree with the ground truth ones.

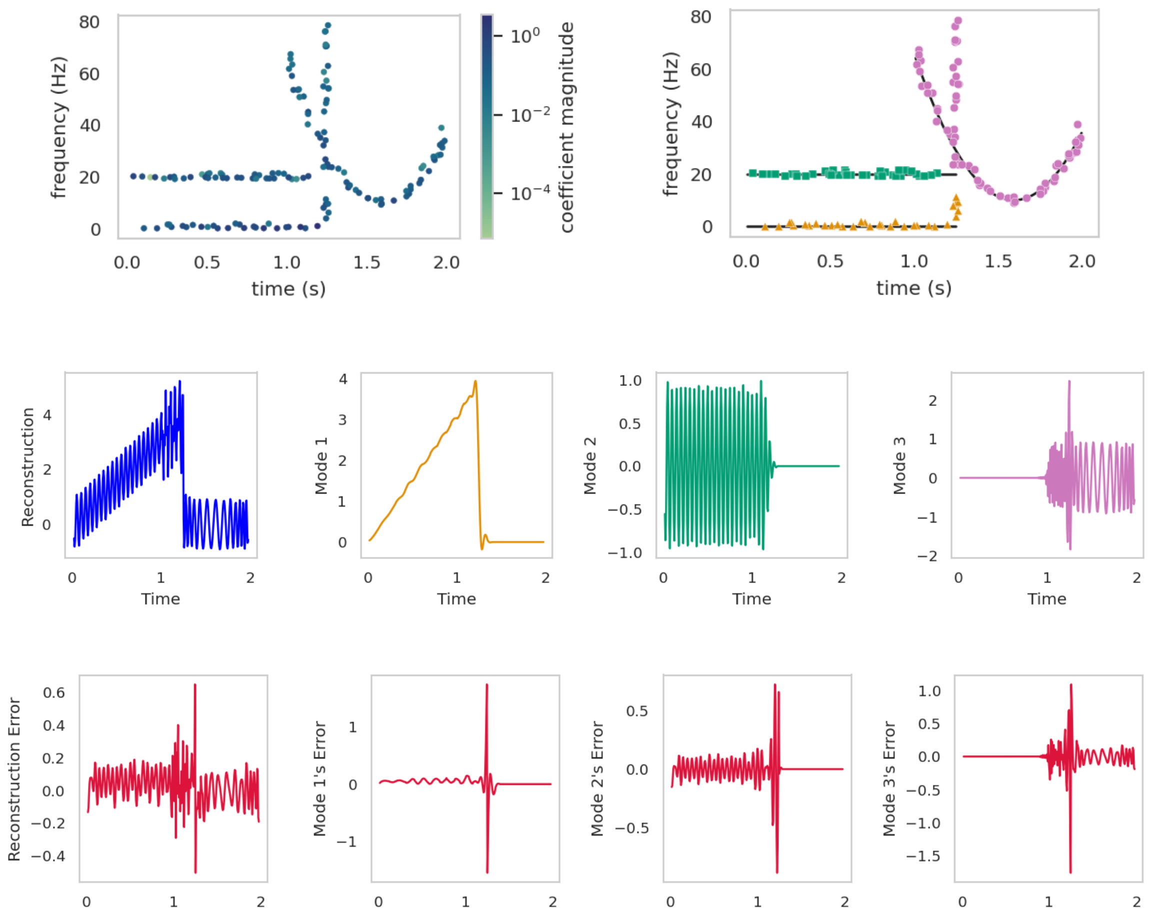


Figure 4.1: **Example from Section 4.4.1:** Results of our method. Top left: Magnitude of non-zero learned coefficients. Top right: Clustering of non-zero coefficients into three modes of different colours. Middle row from left to right: reconstructed signal (in blue) and the three extracted modes matching the colours of the top right clusters. Last Row: Error of the reconstruction and the three modes compared to the ground truth.

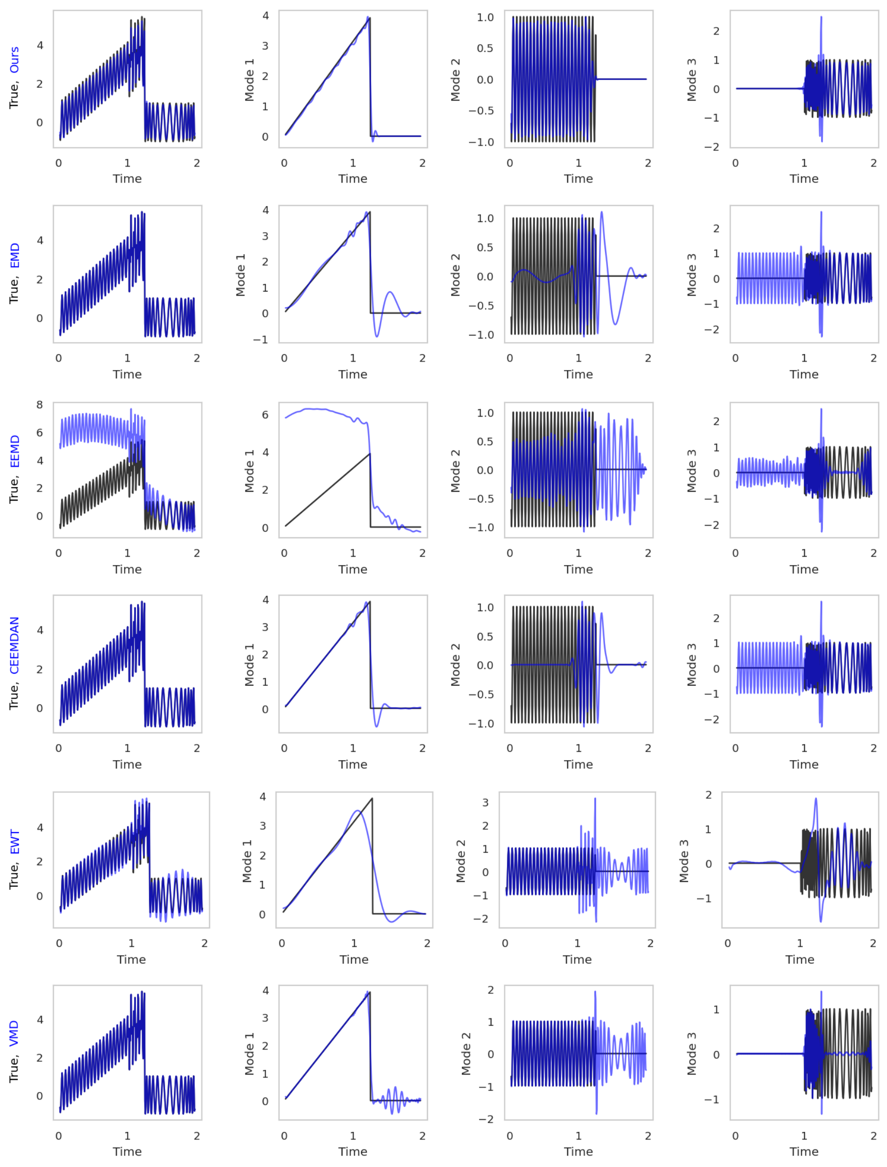


Figure 4.2: **Example from Section 4.4.1:** Comparing different methods on the discontinuous time-series example (Equation (4.10)). Top to bottom rows are our proposed SRMD, EMD, EEMD, CEEMDAN, EWT, and VMD. The first column presents the noiseless ground truth (in black) and the learned signal representation (in blue). The remaining three columns are the first, second, and third modes where the true are plotted in black and the learned ones are in blue.

4.4.2 Instantaneous Frequencies of Intersecting Time-Series

For the second example, we test another challenging signal from [13],

$$\begin{aligned} s_1(t) &= \cos\big(t^2 + t + \cos(t)\big), \\ s_2(t) &= \cos(8t), \end{aligned} \tag{4.12}$$

and the true signal is

$$y(t) = s_1(t) + s_2(t), \quad t \in [0, 10]. \tag{4.13}$$

Notice that the instantaneous frequency trajectories of those two modes $s_1(t)$ and $s_2(t)$ (see Definition 16), $\omega_1(t) = (2t + 1 - \sin t)$ and $\omega_2(t) = 8$, respectively, intersect.

We discuss the ability of revealing the instantaneous frequencies of our method, address the issue of oversegmentation, and compare our results with the other state-of-the-art decomposition methods. Specifically, we expect to see these instantaneous frequency curves in our spectrogram-like plots since our method finds the magnitude A of a windowed pure tone $\sin(\omega t + \varphi)$ at a time-step $t_0 = \tau$. When we solve the over-complete system with BPDN, we expect large magnitudes for the random features to be near the instantaneous frequency curves (and in phase).

In this example, the dataset contains $m = 1600$ equally spaced in time points from $[0, 10]$. We use $N = 10m = 16000$ number of random features for our algorithm. Also, we set $\omega_{\max} = 5$, since all frequencies are less than 5 Hz. Note this scale is the physical frequency in Hz whereas the formula for the modes in this example are given in the angular frequency unit rad/s, thus we set `frq_scale`$= 2\pi$ rather than 1. Lastly, we set DBSCAN's neighbourhood radius to $\varepsilon = 2.0$.

The graph of our learned pairs $\{(\tau_j, \omega_j)\}_j$ (with non zero coefficients $c_j^\sharp$) can also reveal the physical instantaneous frequencies of the full input signal as indicated in the first row of Figure 4.3. Moreover, our method represents the instantaneous frequencies clearer than the STFT, CWT, synchrosqueezed transforms based on wavelets (Figure 8 in [13]), modified STFT [71], and CWT [70] (see Figure 4.3). For the STFT and its synchrosqueezed results (second row), a Gaussian window with standard deviation 0.75 seconds = 60 samples, Fourier transform width of 512 samples, and hop-size of 1 as used. Note this standard deviation is chosen to match the window size used in our method. For the continuous wavelet transform and its synchrosqueezed version (third row), 232 scales were chosen between 3.8 samples (≈ 21 Hz) and 512 samples (≈ 0.16 Hz) with the logarithmic spacing (the default settings of the package `ssqueezepy`, see also [51]), and so the maximum scale matches the Fourier transform width in the STFT used.

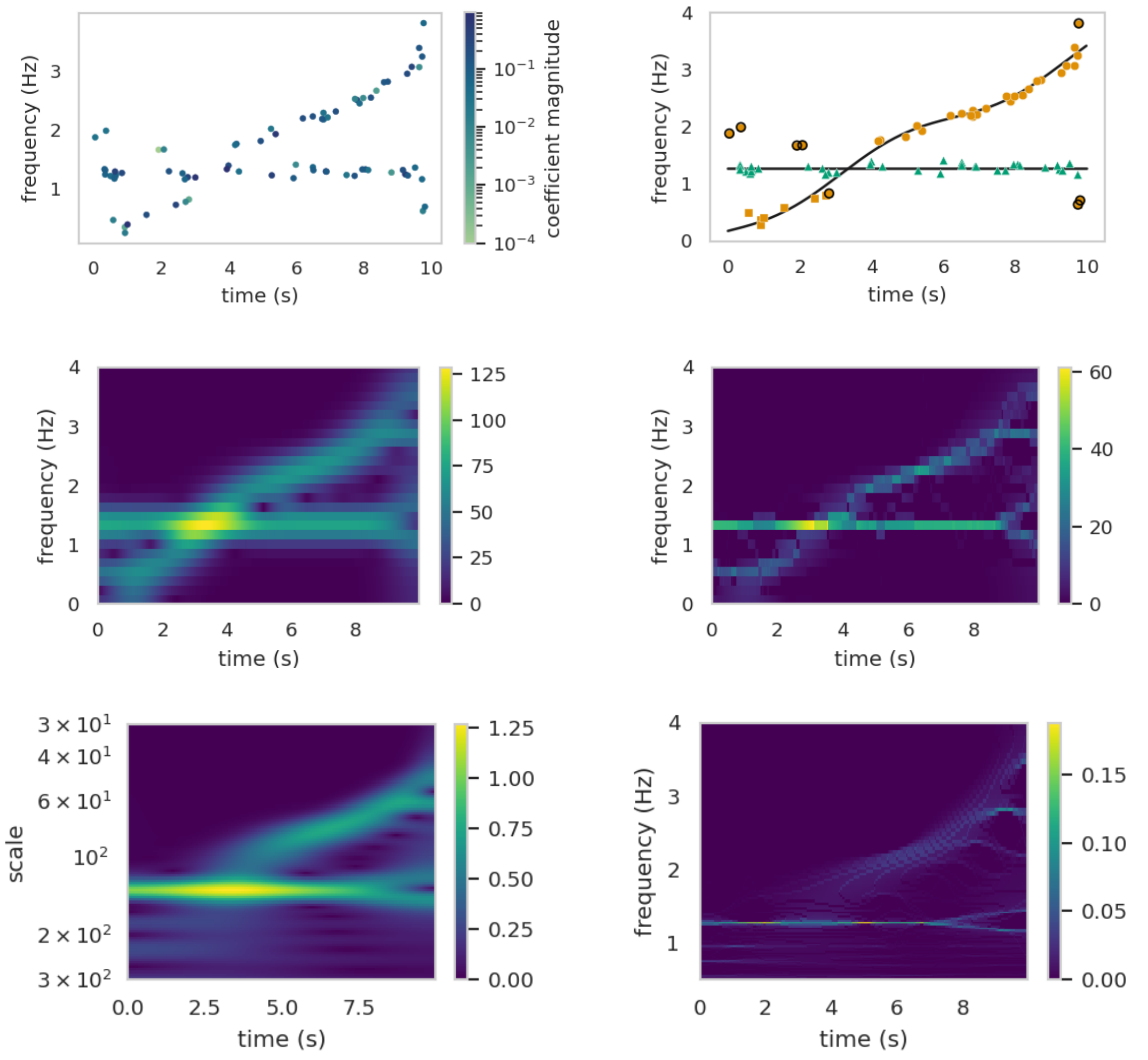


Figure 4.3: **Example from Section 4.4.2:** First row presents the results of our method: plot of the magnitude of learned coefficients (left) and plot of the clustering results of coefficients (right). Dots with a black outline indicate coefficients that DBSCAN labelled as noise and were re-labelled with the nearest cluster (in scaled frequency space). The true instantaneous frequencies of those two modes are solid lines in black. Second row plots the zoomed in to the frequency range $[0, 4\,\text{Hz}]$ of the absolute values of the STFT and its synchrosqueezed version. Third row plots the zoomed-in absolute values of the continuous wavelet transform and its synchrosqueezed version.

Note the formula to convert between scales and frequencies is: frequency $= \frac{\text{samplerate}}{\text{scale}}$. The Morlet wavelet is used since it has a Gaussian window for the most fair comparison.

In addition, applying the DBSCAN on those learned time-frequency pairs $\{(\tau_j, \omega_j)\}_j$ yields three clusters, green, orange circles, and orange squares (see right plot in first row of Figure 4.3). The corresponding learned modes are plotted in Figure 4.4. If we aim to decompose the input signal into two modes, we keep the mode with the largest ℓ^2-norm and combine the two learned modes (among three modes) with smallest ℓ^2-norm to build the second mode. The decomposition result into two modes and the errors with the ground truth modes are shown in Figure 4.5. Our proposed SRMD provides a reasonable extraction of modes where the errors between the extracted modes and the true ones are almost zero everywhere, except at a time-shift region corresponding to the intersection of instantaneous frequencies (around $t \in [3, 5]$).

As seen in Figure 4.6, all examples struggle to separate the two modes. The main reason this example is difficult is that both modes share the same instantaneous frequency at around $t = 3.5$, and the scale of the first mode sweeps a large range of frequencies above and below the stationary frequency in the second mode. Nevertheless, our decomposition results are much better than EMD, its derivatives, EWT, and VMD.

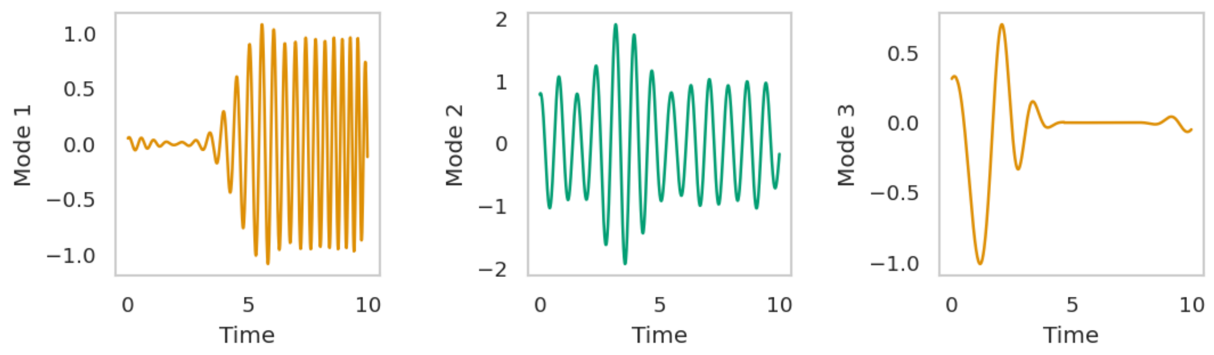


Figure 4.4: **Example from Section 4.4.2:** Decomposition results of our proposed SRMD method into 3 modes (from left to right): orange circles, green triangles, and orange squares.

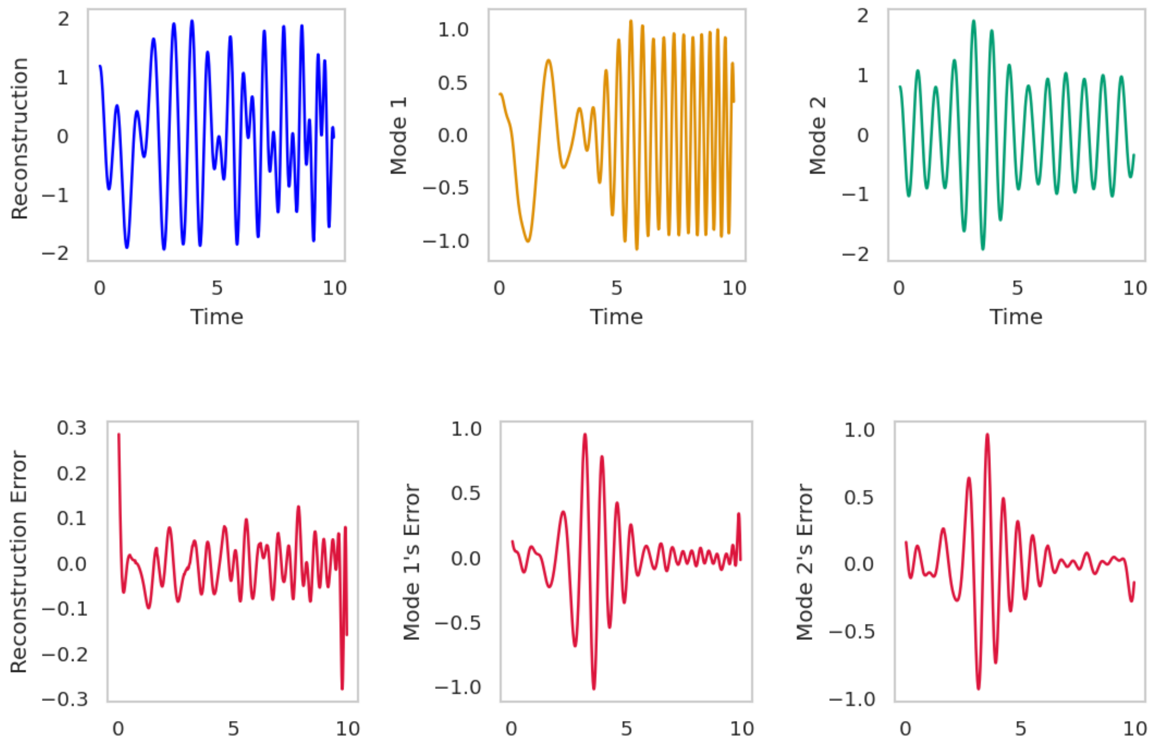


Figure 4.5: **Example from Section 4.4.2:** Decomposition results of our proposed SRMD method into two modes. First row: The learned signal (left) of the full signal and the two learned modes (middle & right). Last row from left to right: Errors between noiseless ground truth and the learned representation, between the true modes and the extracted modes.

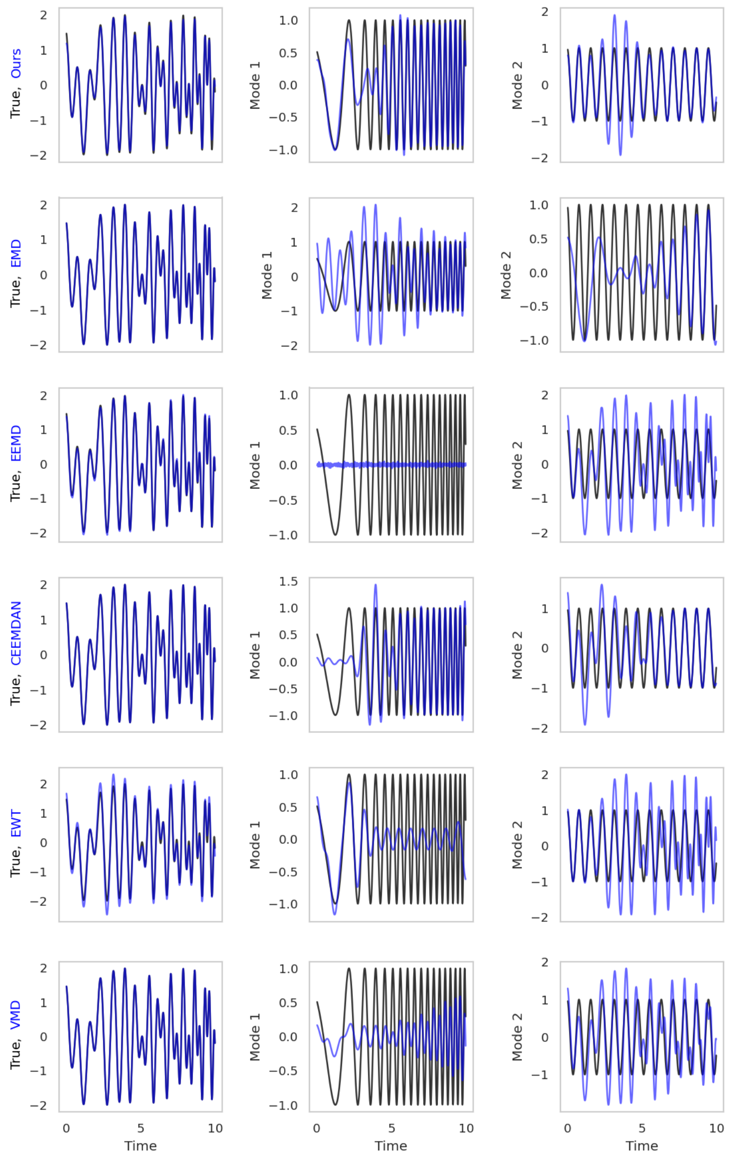


Figure 4.6: **Example from Section 4.4.2:** Comparing different methods on the intersecting time-series example (Equation (4.12)). Top to bottom rows are our proposed SRMD, EMD, EEMD, CEEMDAN, EWT, and VMD. The first column presents the ground truth (in black) and the learned signal representation (in blue). The remaining two columns are the two modes, where the true are plotted in black and the learned ones are in blue.

4.4.3 Overlapping Time-Series with Noise

In this experiment, we investigate another interesting example, where the input signal $y(t)$ is the summation of two modes $s_1(t)$ and $s_2(t)$ with overlapping frequencies and is contaminated by noise:

$$y(t) = y_{true}(t) + \varepsilon = s_1(t) + s_2(t) + \varepsilon, \quad \varepsilon \sim \mathcal{N}\left(0, \frac{r\|y_{true}\|_2}{\sqrt{m}}\right). \tag{4.14}$$

Here the modes s_i are defined by their Fourier transform $\hat{s}_i$ for $i = 1, 2$, where

$$\begin{aligned} \hat{s}_1(k) &= me^{-i\pi k}\left(e^{-\frac{9(k-16)^2}{32}} - e^{-\frac{9(k+16)^2}{32}}\right), \\ \hat{s}_2(k) &= me^{-i\pi k}\left(e^{-\frac{9(k-20)^2}{32}} - e^{-\frac{9(k+20)^2}{32}}\right), \end{aligned} \tag{4.15}$$

for $k \in \mathbb{Z}$ and $t \in [0, 1]$. Note that the modes $s_1(t)$ and $s_2(t)$ produce Gaussians in the Fourier domain centered at $k = 16$ and 20 Hz, respectively. The leading term, $me^{-i\pi k}$, centers the wave packets in time space to $t = 0.5\,\mathrm{s}$ where $m = 160$ is the number of samples. For our proposed SRMD, the hyperparameters to generate the basis are $\omega_{\max} = 40$, the total number of random features is $N = 20m = 3200$, the window size $w = 0.2$. The hyperparameters for the DBSCAN algorithm is $\varepsilon = 1.5$ and we do not need to scale the frequency before clustering.

The noiseless and the noisy time-series with $r = 25\%$ are shown in Figure 4.11. All reconstruction and decomposition results are compared against the true signal (or modes) $y_{true}(t), s_1(t)$, and $s_2(t)$. We compare our results with other methods on noisy signals with different noise ratios $r = 5\%, 15\%$, and 25%. From the results in Figures 4.7, we can see that only VMD and SRMD can successfully decompose the noisy signal. Moreover, when the noise level r is small ($\sim 5\%$), the VMD approach produces comparable results with our method. When r increases, our method is still able to capture the intrinsic modes and denoise the input signal. On the other hand, the VMD is able to identify some parts of those two intrinsic modes but has a hard time in denoising, see Figure 4.8 and Figure 4.9.

The clustering of non-zero coefficients of our methods on noisy signal with the noise level $r = 5\%$, 15%, and 25% are shown in Figure 4.10. Notice that the clusters are surrounded the the two Gaussian peaks (16 Hz and 20 Hz) that define the true input signal in all three cases, which explains the effectiveness of our method on working with these noisy signals. The other functions in the representations offer slight corrects to the overall shape to ensure the reconstruction error is below the specified upper bound. This implies one has flexibility in choosing the width and number of the wavelets to generate since extra wavelets are used to ensure the reconstruction is reasonable.

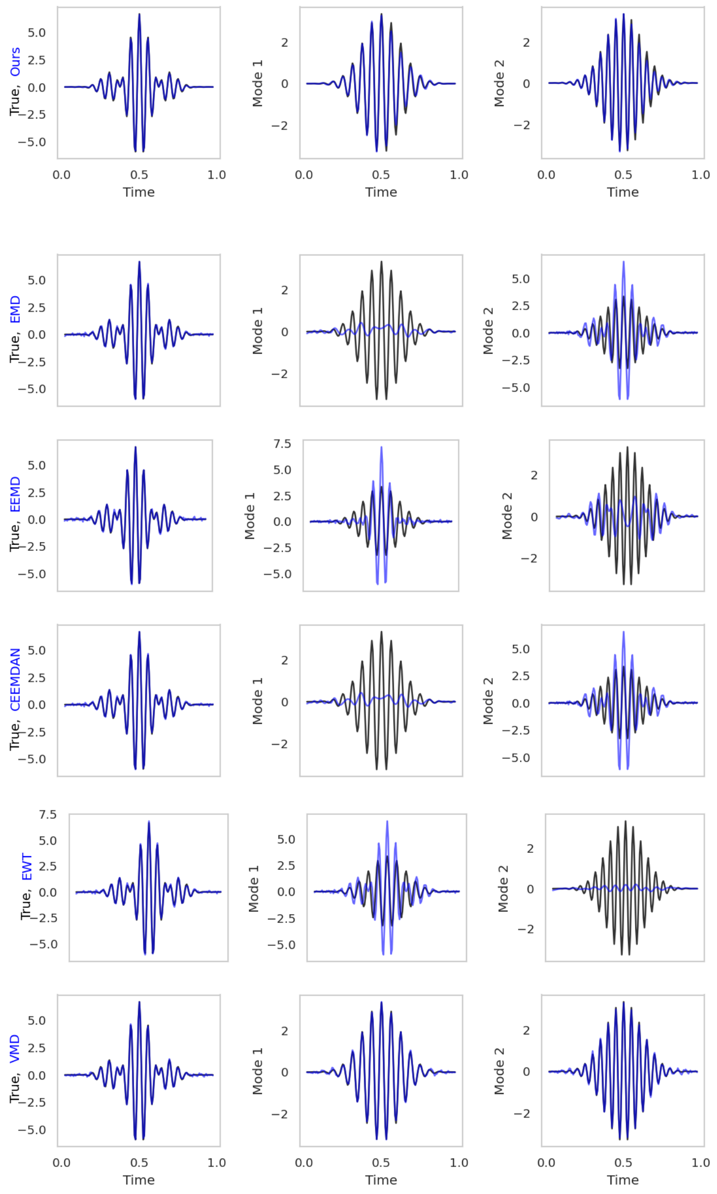


Figure 4.7: **Example from Section 4.4.3:** Decomposition results with $r = 5\%$ noise using six different methods. Top to Bottom Row: Ours, EMD, EEMD, CEEMDAN, EWT, and VMD. First column: the noiseless ground truth (black) and the learned (blue) full signal. Middle and last columns: the first and second ground truth modes (black) with the extracted modes (blue).

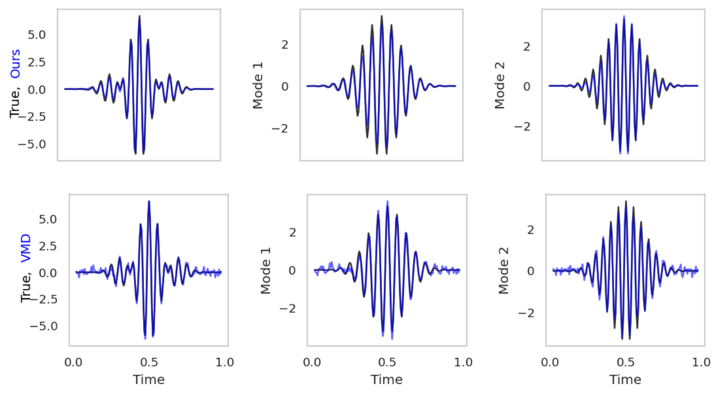


Figure 4.8: **Example from Section 4.4.3:** Decomposition results with $r = 15\%$ noise using Ours (first row) and VMD (second row). First column: the noiseless ground truth (black) and the learned (blue) full signal. Middle and last columns: the first and second ground truth modes (black) with the extracted modes (blue).

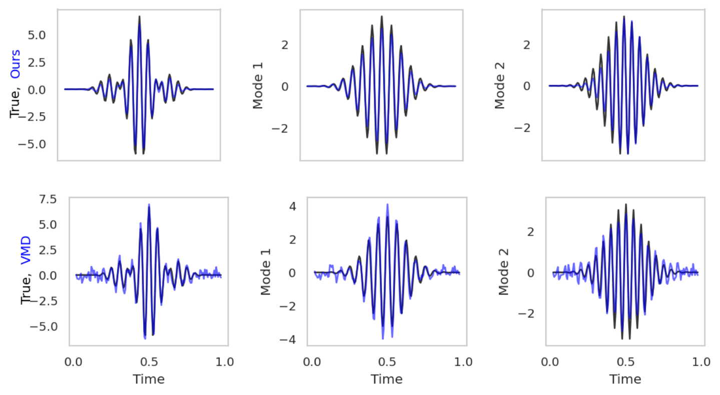


Figure 4.9: **Example from Section 4.4.3:** Decomposition results with $r = 25\%$ noise using Ours (first row) and VMD (second row). First column: the noiseless ground truth (black) and the learned (blue) full signal. Middle and last columns: the first and second ground truth modes (black) with the extracted modes (blue).

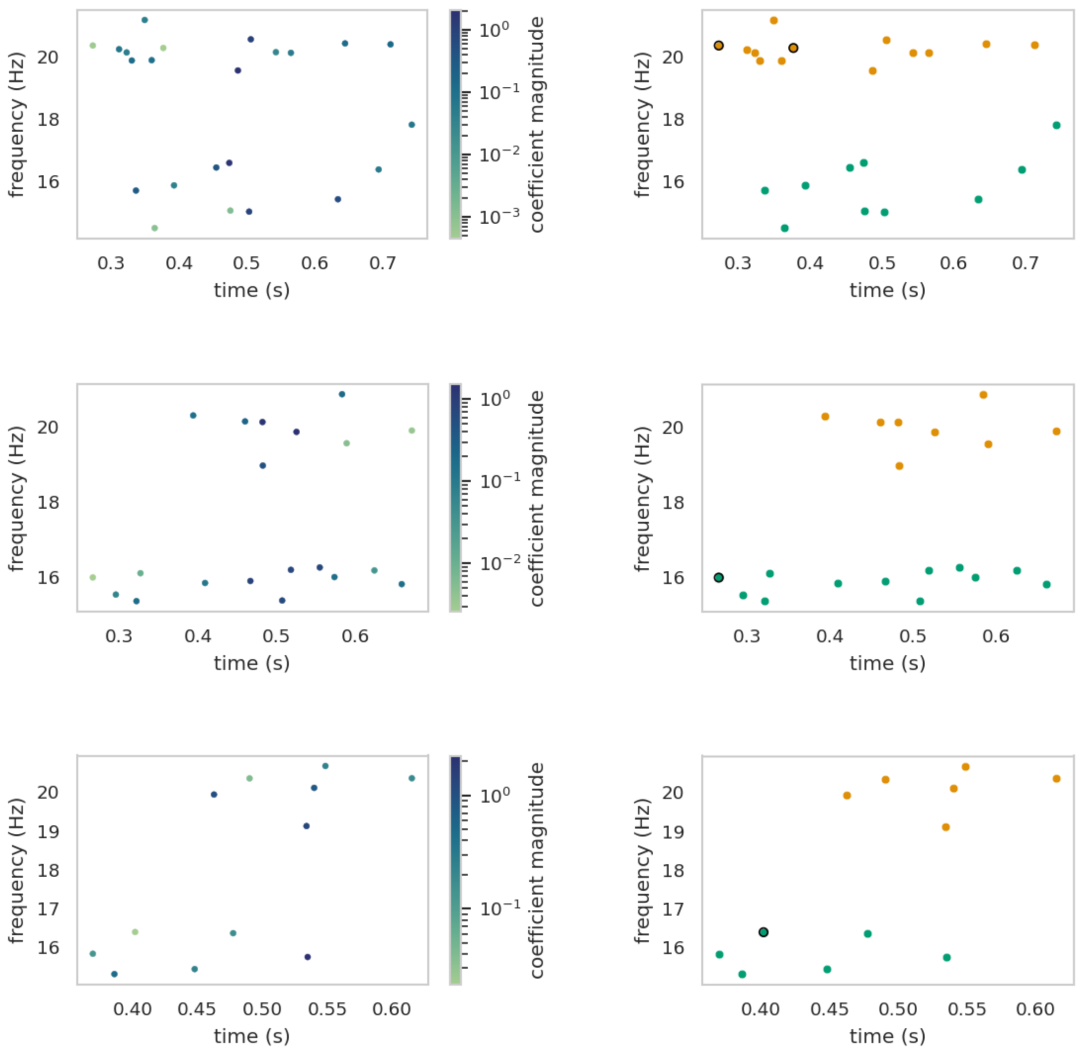


Figure 4.10: **Example from Section 4.4.3:** Results of our method. First column: Magnitude of non-zero learned coefficients for noisy signal with $r = 5\%$, 15% and 25%. Second column: Clustering of corresponding non-zero coefficients into two modes (green and orange).

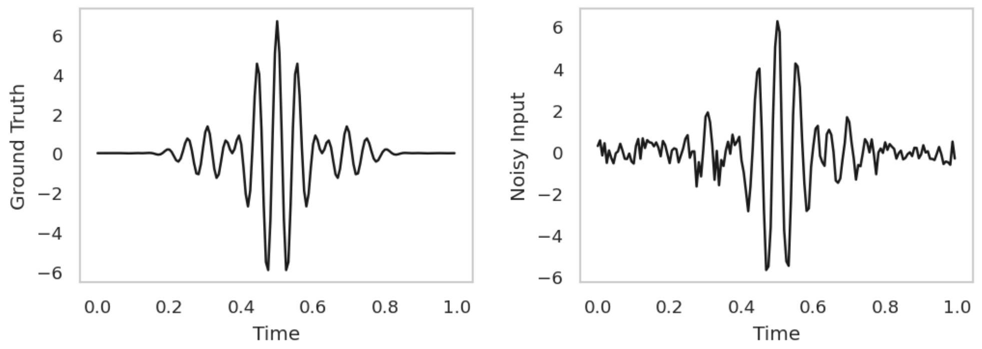


Figure 4.11: **Example from Section 4.4.3:** Left: Ground truth signal. Right: Noisy input with $r = 25\%$.

4.4.4 Pure Sinusoidal Signals with Noise

In this example, we aim to decompose a noisy signal into three modes where the noise level is significant in amplitude with respect to two out of three modes. Consider a noisy signal, which is suggested in [16]:

$$y(t) = s_1(t) + s_2(t) + s_3(t) + \epsilon(t), \quad \epsilon(t_\ell) \sim \mathcal{N}(0, 0.1), \tag{4.16}$$

where

$$s_1(t) = \cos(4\pi t),\ s_2(t) = \frac{1}{4}\cos(48\pi t),\ s_3(t) = \frac{1}{16}\cos(576\pi t).$$

The hyperparameters chosen are $\omega_{max} = 500$, $w = 2s$, $m = 1000$, $N = 50m$, $r = 15\%$, `threshold` $= 0$, $\varepsilon = 1.5$ (for DBSCAN), `min_samples` $= 4$, and `frq_scale` $= 1$. For this example only, the signal is extended from the domain $[0, 1]$ to $[-1, 2]$ by an even periodic extension before training. This is to remove end effects that would otherwise appear in the decomposition given the signal starts and ends far from zero. The number of data points m and features N are thus scaled by a factor of 3 during the learning of the signal. The results of our method are shown in Figure 4.12.

Our method can extract the first two learned modes with high accuracy. Note that both VMD and our method have difficulty in extracting the weak and high-frequency mode $s_3(t) = \frac{1}{16}\cos(576\pi t)$. Nevertheless, our method can identify the frequencies of all three modes. More precisely, the median frequencies of three learned clusters are 1.99, 24.03, and 288.02 Hz, which are very close to the ground truth frequencies 2, 24, and 288 Hz.

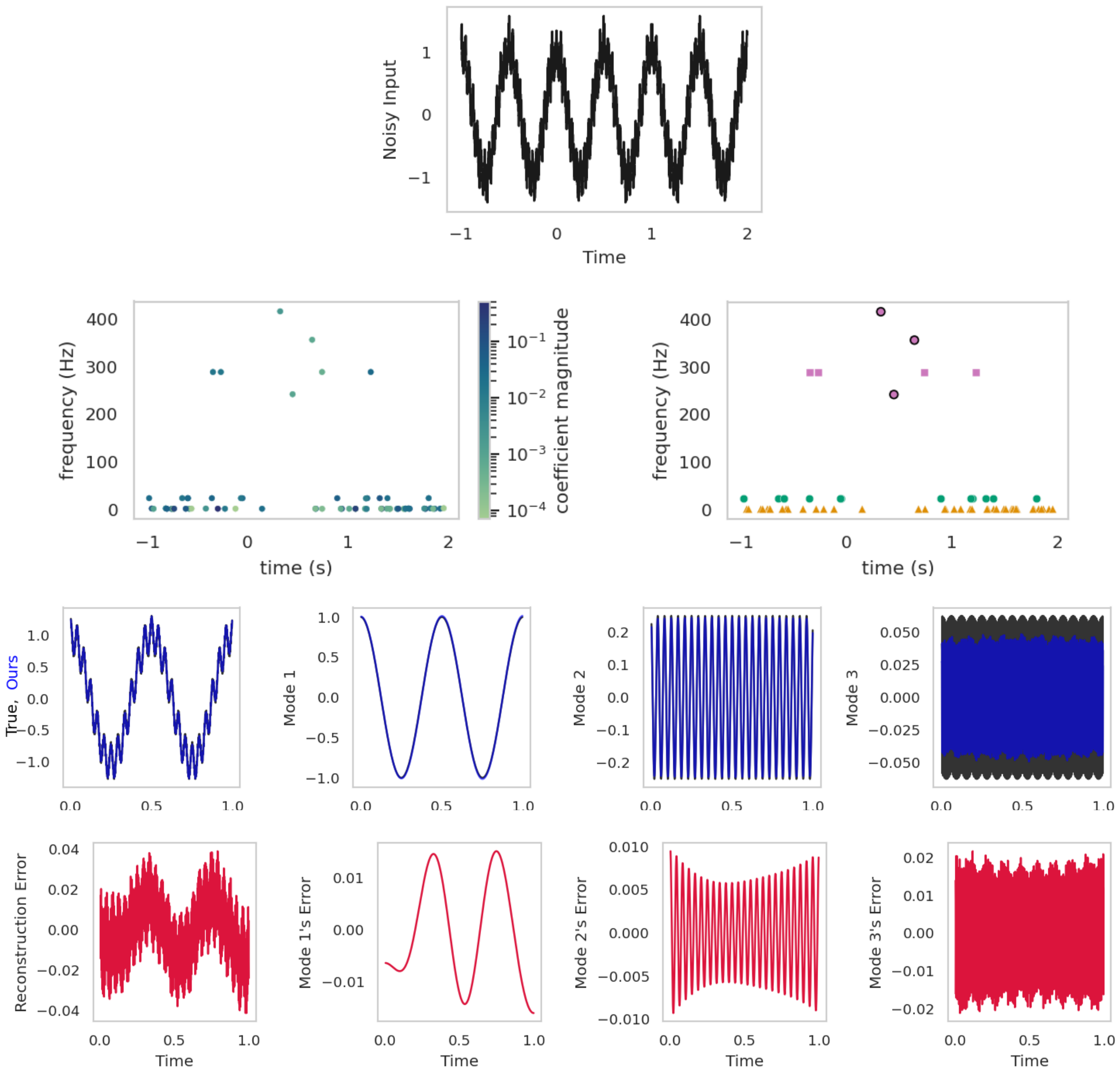


Figure 4.12: **Example from Section 4.4.4:** Results of our method. First row: The noisy input signal. Second row: Magnitude of non-zero learned coefficients (left) and clustering of non-zero coefficients into three modes represented by different colours (right). Outliers identified by DBSCAN are circled in black and re-clustered into the nearest cluster. Third row from left to right: reconstructed signal (in blue) and the three extracted modes (in blue) versus the corresponding noiseless ground true signal and modes (in black). Last row: Error of the reconstruction and the three modes compared to the ground truth.

4.5 Extensions

We now extend SRMD outside its original formulation. Given there are two-stages of the method, representation and decomposition, the output of the representation can be valuable on its own, and modifications can be made in one stage independent of the other. The representation stage of the method is applied to astronomical data in Section 4.5.1, and the case where the feature parameters (τ, ω, ψ) are chosen deterministic is considered 4.5.2. Extensions to music is considered in Chapter 5 and additional extensions that may be worth exploring in future work are given in Section 6.2.

4.5.1 Visualizing Gravitational Data

As an application to real data, in the astronomical time-series example, we show that our (semi-supervised) approach can extract a localized and noise-free representation of the first observed gravitational wave.

In this example, we apply our method to the detected signal of the first observation of gravitational wave, GW150914 [20]. The noisy input data is preprocessed in the same way as in [20], which consists of whitening (ensuring the Fourier transform of the signal is flat), filtering (remove the highest and lowest frequencies outside the calibrated range), and downsampling steps (only keep every 16 samples to reduce memory). The preprocessing steps are necessary in order to reveal the black hole chirp signal hidden under a layer of biases and noise that cannot immediately seen from the data directly outputted from the instruments. In addition, we also normalize the input data by dividing the signal by the maximum value in order to speed up the algorithm and avoid numerical error since the original signal is on the order of 10^{-19}. The goal here is to visualize the black hole merger and denoise the preprocessed signal. Specifically, we would like to automatically detect an accurate time-frequency chirp of the signal in the spectrogram plot.

The hyperparameters for our method are $m = 861$ measurements, $N = 20m$ number of random features, window size $w = 0.01$, maximum possible frequency $\omega_{\max} = 2\,048$ Hz. The parameter for LASSO is $\tau = 12$. Due to the large amount of noise in the data, we keep only the top 3% or 5% largest non-zero coefficients to visualize the black hole merger in the time-frequency plane and construct the learned signals. The results are shown in Figure 4.13.

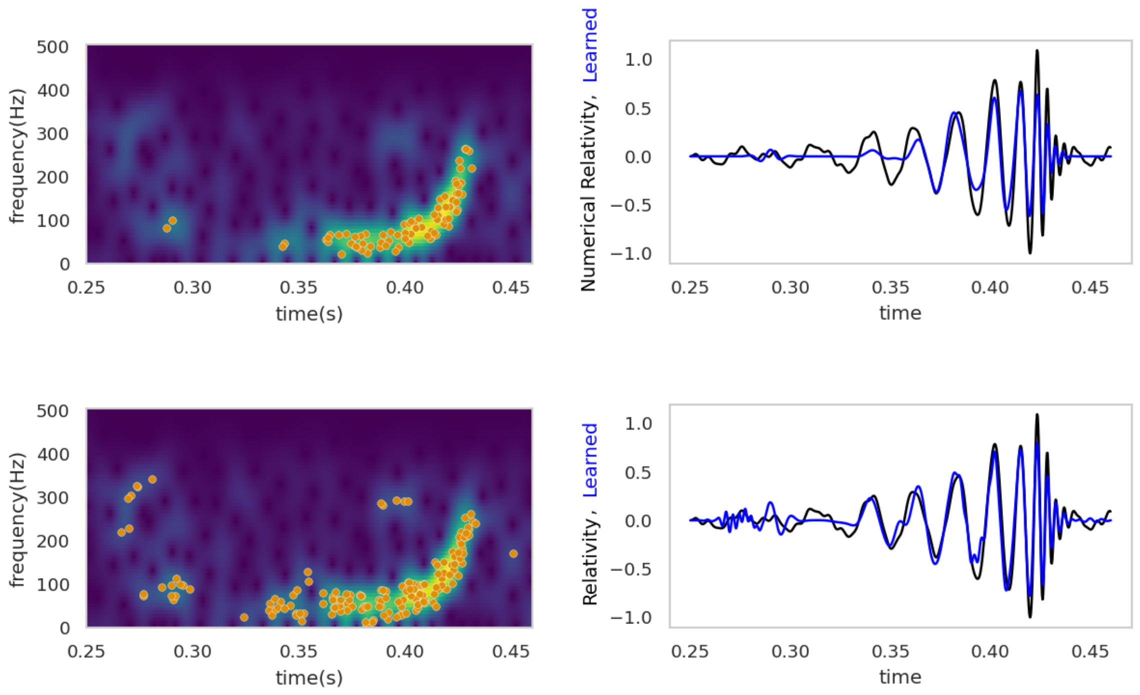


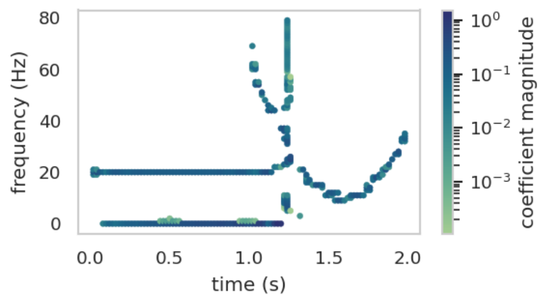
Figure 4.13: **Example from Section 4.5.1:** Left column: The largest 3% (first row) and 5% (2nd row) learned nonzero-coefficients overlay STFT in the time-frequency domain. Right column: Corresponding learned signals (blue) and numerical relativity data (black). Time is given as seconds after September 14, 2015 at 09:50:45 UTC.

4.5.2 Non-random Features

This section explores a modification of the SRMD method where the feature parameters (τ, ω, ψ) are chosen from a fixed grid—rather than from a uniformly random distribution—to highlight the benefits of random features. We reuse the discontinuous example 4.4.1 to showcase the uniform features' behaviour on constant and changing instantaneous frequency modes.

Given the number of desired features $N = 50m = 16\,000$, length of signal $T = 2\,\mathrm{s}$, and maximum frequency $\omega_{\max} = \frac{m}{2T} = 80$ Hz, we generate all triples $(\tau_j, \omega_j, \psi_j)$ such that $\tau_j \in [0, T]$, $\omega_j \in [0, \omega_{\max}]$, and $\psi_j \in \{0, \pi/2\}$, and $\tau_j - \tau_{j+1} = \Delta\tau$ and $\omega_j - \omega_{j+1} = \Delta\omega$ are constant. This leaves one free parameter, the ratio of time-shifts to frequencies so that the product of the number of time-shifts, the number of frequencies, and the two possible phases equals N. The ratio 1.25 is selected to give us 100 time-shifts and 80 frequencies which are the pair of integer factors of $N/2 = 80\,000$ that are as close to each other as possible. This lets us have the most number of unique time-shifts and frequencies. We set $\varepsilon = 0.08$ to ensure three modes were returned, but otherwise all parameters are identical to example 4.4.1. The magnitude of learned coefficients are compared to the original SRMD method in Figure 4.14 and the full results are shown in Figure 4.15.

Figure 4.14 showcases how the equally spaced features lead to a very good representation of the instantaneous frequency curves if they are horizontal or vertical in the time-frequency plane. The IF representation is poorer than the usual random features proposed by SRMD when these curves are not in these directions. This can be seen by the blocky nature of the chirped mode which is caused by the features existing only along this grid.



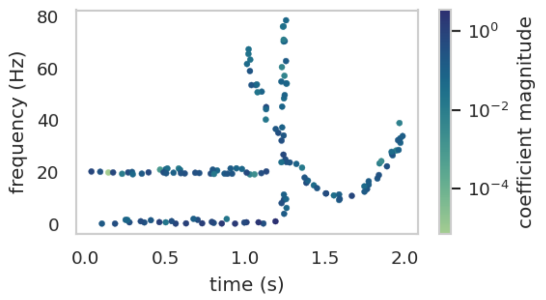


Figure 4.14: **Example from Section 4.5.2:** Left: Magnitude of non-zero learned coefficients of the SRMD method using evenly spaced feature parameters. Right: Magnitude of non-zero learned coefficients of the standard SRMD method.

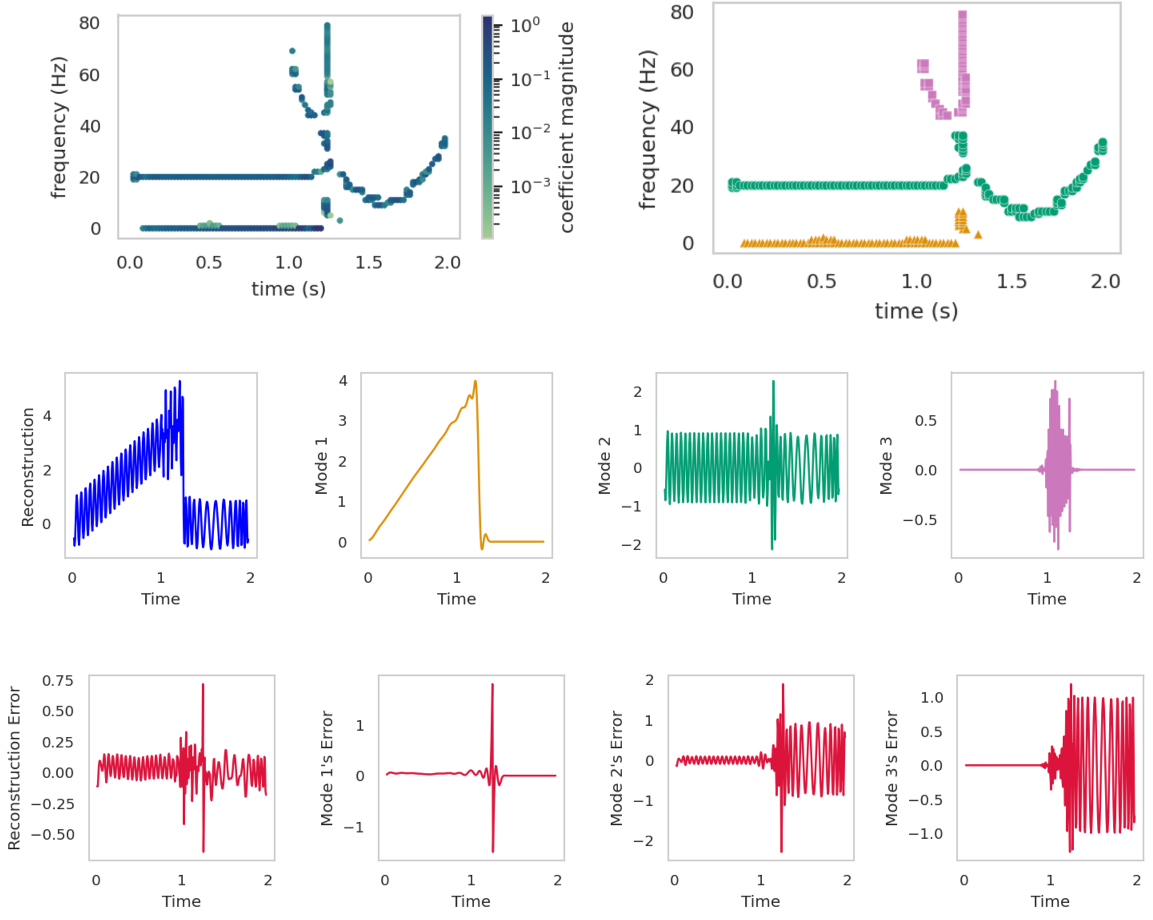


Figure 4.15: **Example from Section 4.5.2:** Results of the SRMD method using evenly spaced feature parameters. Top left: Magnitude of non-zero learned coefficients. Top right: Clustering of non-zero coefficients into three modes of different colours. Middle row from left to right: reconstructed signal (in blue) and the three extracted modes matching the colours of the top right clusters. Last Row: Error of the reconstruction and the three modes compared to the ground truth.

The equally spaced features also lead to a more dense representation, requiring 283 nonzero features to achieve the same reconstruction error as the proposed SRMD which only uses 141—about half as many. This is expected since both phases $0, \pi$ are needed to reproduce the correct phase of the input. This denser representation leads to the horizontal mode at 20 Hz touching the chirp at $t = 1.25\,\mathrm{s}$, which means DBSCAN is unable to correctly separate these clusters. This can be seen in the green circle cluster of the top right plot of Figure 4.15. The denser representation, and poorer visualization in the case of modes that are not horizontal or vertical in the time-frequency plane, justifies the use of random features in the SRMD method.

4.6 Discussion

The SRMD method makes improvements over previous methods given in Section 3.3. Unlike the Fourier filtering and masking methods, SRMD automatically identifies and extracts the constituent modes. The time-frequency visualization is sharper and sparser than STFT, CWT and its Synchrosqueezed version. SRMD more consistently and accurately extracts the modes when compared to EMD and its related methods. This method also does not require the number of modes to be known in advance, which is required by EWT and VMD, and is able to denoise the input more aggressively than VMD in the presence of high levels of noise.

Despite these improvements, limitation to this method should be acknowledged. SRMD performs well if a good sparse representation can found from the generated features. The modes need to be well connected and disjoint from each other in frequency-time space. The number of modes needs to be known in advance if their IFs cross so that the individual clusters can be pieced together as shown in example 4.4.2. When the number of modes is not known, the learned coefficients and decomposition need to be observed to ensure reasonable values for the hyperparameters are chosen, especially the number of features N and DBSCAN's core neighbourhood size ε. This does limit the usefulness of this method as a *blind* decomposition method—one that can be used to automatically decompose a signal without knowing additional information about the modes within the signal. Possible future work which could lead to further improvements is explored in Section 6.2.

Chapter 5

Musical Source Separation

5.1 Overview

Musical source separation is an application of signal decomposition of interest to both industry and academia [53]. In this context, a digitally recorded song is treated as the input and, typically, individual instruments (bass, drums, vocals, etc.) are treated as the constituent sources. Instrument duplicates, for example two singers, are combined and treated as one source. This may be done to remove vocals for a karaoke track, or isolate them for remixes. Other applications when the isolated instrument tracks are not available include historical analysis of old recordings, or remastering of live recordings. Isolating sources also enables more involved signal processing techniques to be used such as re-pitching or re-timing, denoising sources, or transcribing each part to sheet music.

There are domain specific assumptions that distinguish musical source separation from the signal decomposition problem SRMD is originally designed to to tackle. Modern digital music recordings are often distributed in stereo and consist of a left and right channel: one for each ear when wearing headphones, or for two speakers placed some distance away from each other and the listener. Movie and video game soundtracks often contain even more audio channels such as Dolby Digital's 5.1 surround sound system [38]. A source may be more present in one channel than another, and time delays or phase shifts between channels can exist. Single channel separation can be used on multi-channel recordings by converting the recording to a monophonic (*mono*) single channel. This is typically done by averaging the channels together, however, the phases shifts can introduce artifacts in the process. Modern recordings are also sampled at a very high rate, typically 44 100 or 48 000 Hz, to ensure the entire range of human hearing can be captured. For a 4 minute song, the number of samples m required is on the order of 10 million. To follow SRMD's recommendation that the number of features is $N = 10m$, 100 million features would need to be generated leading to a feature matrix $\boldsymbol{A}$ with 10^{15} entries. One way to reduce the scale of the problem is to *downsample* the recording by considering only a fraction of the samples. This removes high frequency information and can cause aliasing artifacts when not carefully treated, but is sometimes necessary to reduce the size of the problem.

It is often the case that a recorded song has some non-linearity where the song is not a simple sum of the sources. For example, if side-chain compression is applied, the amplitude of a background instrument may be reduced every time the kick drum is played. A basic model of this could be written as $y(t) = s_d(t) + (1 - c|s_d(t)|)s_o(t)$ where $s(t)_d$ is the drum source, $s(t)_o$ are the other instruments, and c is the amount of side chain. For simplicity, we often assume the sources are purely summed to produce the mixture since this reasonably approximates many live recordings of western genres like jazz and rock, and older styles of music before these modern mixing techniques were used.

SRMD is also designed with each source being represented by IMFs. This is often *not* the case for musical sources, where a single note contains additional frequencies higher than the fundamental frequency. We call these frequencies *overtones*, and when an overtone is a near-integer multiple of the fundamental, we call them *harmonic*. Different instruments within a recording are also highly correlated in the sense that notes will commonly start and end at the same time, and their harmonics will overlap making it difficult to determine which frequency belongs to which source. Percussion hits such as cymbals contain overtones that are not integer-multiples of the fundamental pitch, and often so many that they would appear as blurred vertical lines on a spectrogram. These complications lead to musical signals being *dense* in the time-frequency domain—contrary to SRMD's assumption that the signal can be sparsely represented by the features it generates.

Despite these challenges, SRMD will be used to represent a simple musical example in Section 5.2. The decomposition, however, is performed by manually identifying the frequencies corresponding to each source. To add value to this exercise, a comparison is made between downsampling the input with equally spaced samples and randomly chosen samples to showcase SRMD's ability to accurately reconstruct signals under a nonuniformly sampled time-series.

When applying the sparse representation to a typical music clip in Section 5.2.2, it is shown that the representation is not sparse and a standard clustering algorithm would not be able to identify the sources contained with in it. Rather than conclude the discussion, a preliminary neural network inspired by [37] and [63] is applied to music source separation in Section 5.3.

5.2 Random Feature Model

5.2.1 Simple Musical Example

We generate a music dataset using sources (flute and guitar) and thus can compare our method to the ground truth modes. The corresponding audio files of the extracted modes can be found on GitHub.[1]

In this example, we use our method to decompose a two-second clip of a guitar and flute playing simultaneously. The representation into sparse random features is performed identically as before, but the clustering is performed by splitting wavelets with a frequency above and below the frequency cutoff at 480 Hz. This cutoff is chosen because of the visual information provided by the plot of nonzero random features' time-shift and frequency. Specifically, the fundamental and first 3 flute harmonics can be most clearly seen at $t = 0.5\,\mathrm{s}$ in the top left plot of Figure 5.3, suggesting a cutoff slightly below the fundamental frequency is most appropriate. This is similar to traditional signal processing techniques that rely on STFT to observe and isolate regions in time-frequency space. Our method has the advantage of finding a sparse representation so individual harmonics can be clearly seen, like a Synchrosqueezing STFT or CWT, but with the additional advantage that the signal is denoised in the process.

We examine the decomposition results of SRMD in two cases when the input signal is equally-spaced downsampled and randomly downsampled. An illustration of the sampled data in both cases are presented in Figure 5.1, where the original full signal is sampled at 44.1 kHz, the equally-spaced downsampling is at 2.8 kHz, and the random sampling has $1/16^{\text{th}}$ as many points as the original signal. This downsampling would be problematic for songs that feature instruments with a lot of high frequency content, including drum and cymbals, where the full range of hearing is important for high quality audio. However, for a clip with just flute and guitar, most of the frequency content is contained below 2000 Hz so this does not drastically effect the sound. The original and extracted audio files for this example can be found in our repository. The hyperparameters of our method are $m = 5\,107$ samples, $N = 10m$ random features, the window size $w = 0.03$, and $r = 10\%$. For the equally spaced downsampling, we must choose $\omega_{\max} = 2.8\,\mathrm{kHz}/2 = 1\,378$ Hz (the Nyquist frequency), where as the random sampling uses twice this with $\omega_{\max} = 2.8\,\mathrm{kHz}$.

The zoomed-in plots of the learned signals and modes from the equally-spaced and random downsampled data are plotted in Figure 5.2. The results indicate that our method works well even when the given data is sampled randomly. Moreover, the random sampling

[1] https://github.com/GiangTTran/SparseRandomModeDecomposition

allows for higher frequencies in the flute to be captured. As demonstrated in [7, 33], frequencies above the typical Nyquist frequency can be recovered when randomly sampling. In this case, the maximum frequency of the features generated are twice as high as the equally sampled, while keeping the same number of generated features. Finally, Figure 5.3 shows the results of the randomly downsampled experiment. We note the reconstruction error plots given in the bottom row show the learned audio sources are close to the ground truth.

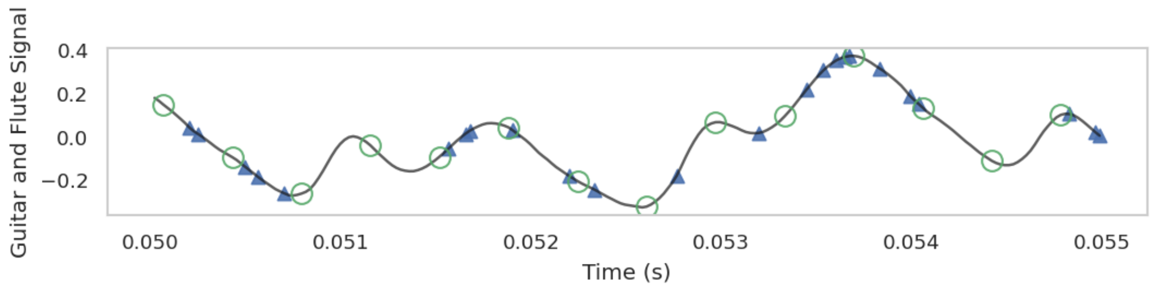


Figure 5.1: **Example from Section 5.2.1:** Zoomed in input. Original signal is sampled at 44.1 kHz, equally-spaced downsampling is at 2.8 kHz (circles), and random downsampling has 1/16th as many points (solid triangles). The random downsampling does not have exactly 2 800 samples per second, but will have that many on average.

5.2.2 Challenges

We now apply the sparse representation method to the first song in the MUSDB18 music data set [58]: "A Classic Education by Night Owl". The song is truncated to the first 2 seconds, and otherwise all hyperparameters were kept identical to the randomly downsampled example in Section 5.2.1.

In Figure 5.4, we can see the non-zero learned coefficients are *not* sparse in the time-frequency plane. A few features such as the onset of a note at 1.5 s with frequencies around 600 and 900 Hz can be seen, but the representation is otherwise an incoherence collection of points. DBSCAN is therefore unable to identify any meaningful clusters within this representation and thus the decomposition cannot be performed. It is possible that decomposition could be performed by modifying SRMD so that a suitable representation can be found. These modifications are explored in Section 6.2, but for now, an alternate approach is needed to decompose these types of signals.

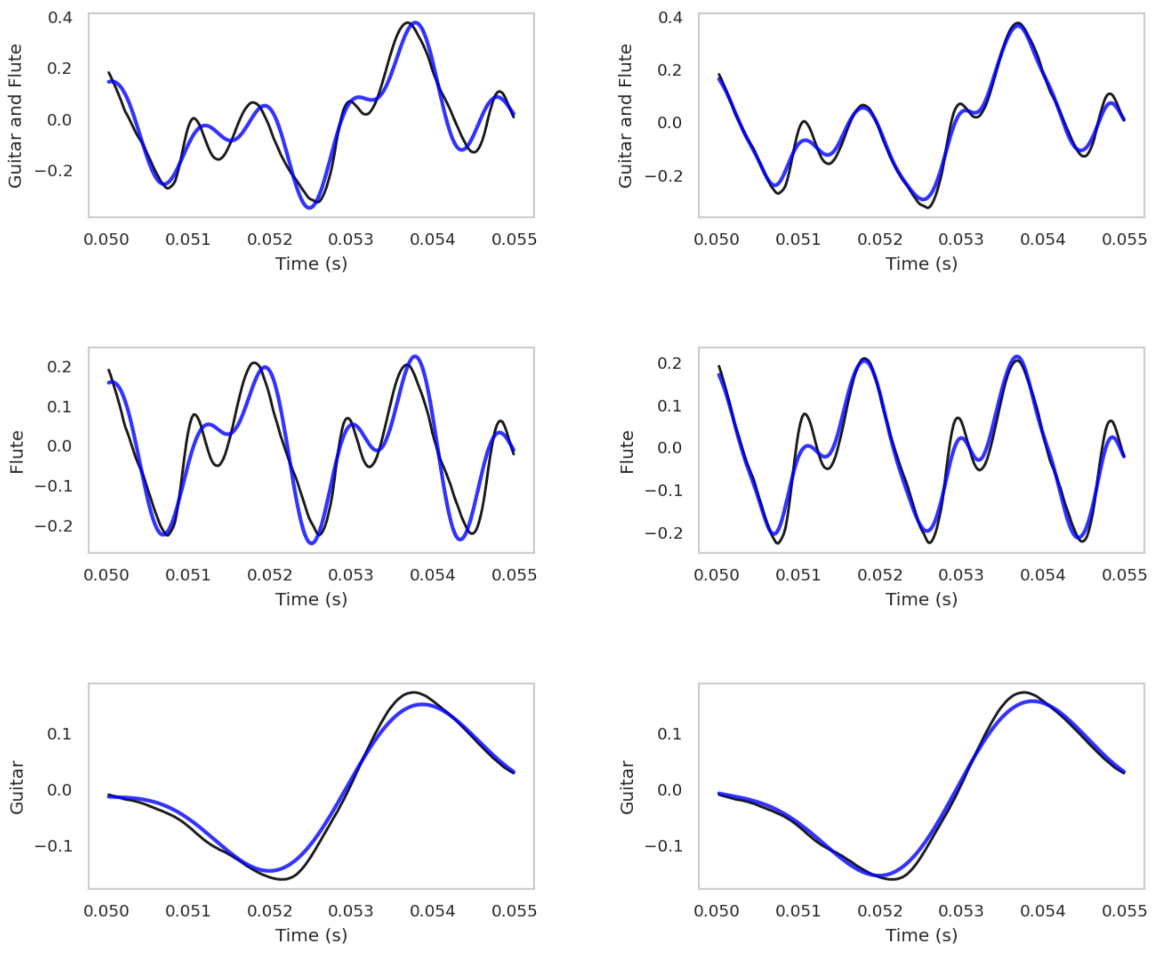


Figure 5.2: **Example from Section 5.2.1:** Zoomed in plots of the learned signals and modes (blue) from our method as compared to the original fully sampled ground truth (black). Left: Results using equally-spaced data. Right: Results using random samples. Input signal is downsampled by a factor of 16.

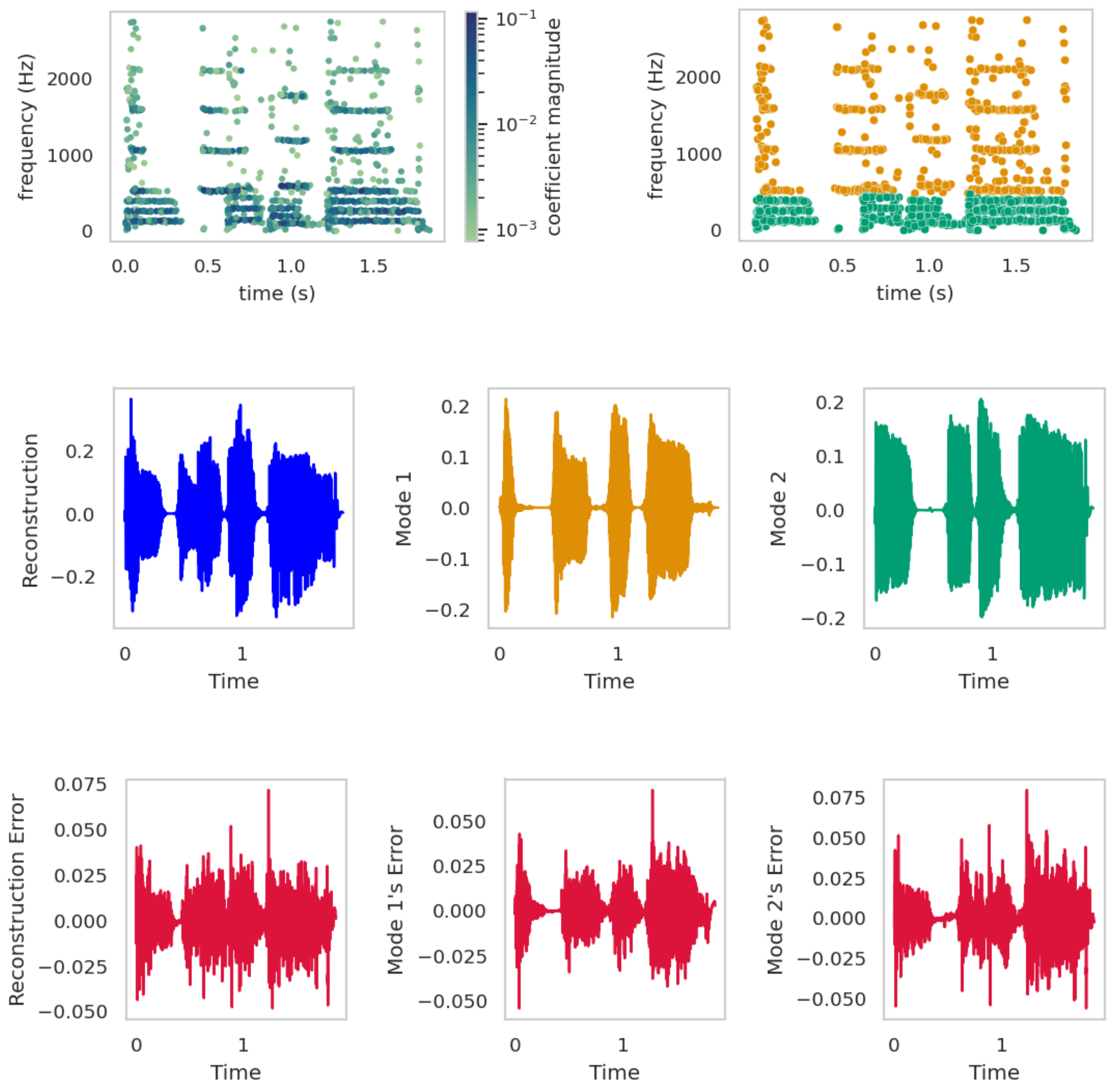


Figure 5.3: **Example from Section 5.2.1:** Results of our method with randomly downsampled input by a factor of 16. First Row: magnitude of non-zero learned coefficients (left), clustering of coefficients into two modes (right). Second row: Learned signal (blue, left) and the learned flute and guitar modes (middle and right). Last Row: error between the learned signals and the downsampled input.

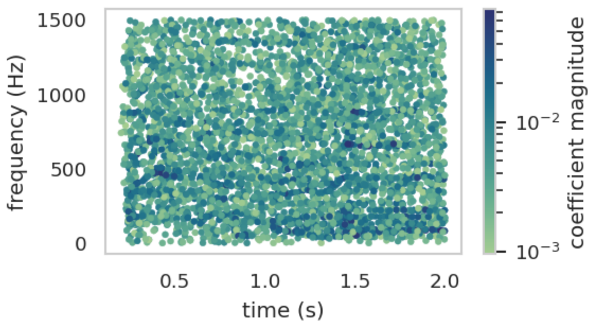


Figure 5.4: **Example from Section 5.2.2:** Magnitude of non-zero learned coefficients of our method with uniformly downsampled input by a factor of 16.

5.3 A Neural Network Attempt

To be able to separate complicated signal such as digital musical recordings, we must incorporate some element of *learning*. Incorporating learning or training helps our model know where to look within the input based off prior knowledge of musical signals. This motivates the use of a neural network which has shown success in learning complicated audio signals [57]. The architecture is inspired by [37, 63] and combines an autoencoder with convolutional layers, with skip connections that bridge the middle, latent space layer.

5.3.1 Method

Data Generation. To create our training set $\mathcal{T}$, we start with the 7 second demo clips from the MUSDB18 database [58] (143 songs with ground truth sources) and re-mix them to create $N = 1000$, 2 second clips by augmenting the data in a variety of ways. To create these "incoherent" clips, we can apply the following transformations to different sources within the database:

- mix and match the vocals and instruments from different songs
- start them at different points within the 7 second clips (time-shift)
- speed them up or slow them down (speed-shift)
- scale all frequencies to change the musical key (pitch-shift).

To create "coherent" clips, the sources must all be from the same song and the operations like time-shift, speed-shift, and pitch-shift have to be the same. Our training set is a combination of coherent and incoherent mixtures to encourage better generalization. We want the network to properly learn each source separately and not, for example, predict the bass part based on the chords in the other instruments. Our testing set contains solely coherent mixtures since we only want to evaluate the performance based on real world-like songs.

The database has the sources split into the 4 sources: vocals, drums, bass, and other, so we add together the drums, bass, and other sources to create our accompaniment source. The database also contains the songs sampled at 44 100 Hz so we evenly downsample the clips by a factor of 4 to reduce the size of the problem.

Input Processing. We take $T = 2$ second single-channel clips $\boldsymbol{y} \in \mathbb{R}^{sT}$ sampled at $s =$ 11 025 Hz. The clip is preprocessed by applying the STFT and obtaining $\boldsymbol{Y} = \mathrm{STFT}(\boldsymbol{y}) \in \mathbb{C}^{\Omega \times H}$ where $\Omega = 257$ is the number of frequency bins and $H = 174$ is the number of hops or time bins. We wish to decompose $\boldsymbol{y}$ as the sum of 2 sources $\boldsymbol{s}_v$, $\boldsymbol{s}_a$ where $\boldsymbol{s}_v$ are the vocals and $\boldsymbol{s}_a$ is the accompaniment. Since the STFT is linear, we also have $\boldsymbol{Y} = \boldsymbol{S}_v + \boldsymbol{S}_a$ where the capitalization indicates the STFT. We then convert the complex STFT representation to a real magnitude spectrogram before normalizing and rescaling the spectrograms to exaggerate detail:

$$\left(\frac{|\boldsymbol{Y}|}{\max_{i,j}(|Y_{ij}|)} \right)^{\frac{1}{2}} \tag{5.1}$$

where the exponent acts element-wise.

Architecture. The architecture combines convolutional layers and fully connected layers with skip connections and an autoencoder framework. A diagram of the architecture is provided in Figure 5.5 and the 6 layers in the network along with the number of channels and convolution filters used are shown in Table 5.1. This architecture has been adapted from [37] and [63]. We justify the non-square kernel shapes by first convolving all frequency bins $\Omega = 257$ together so the model consolidates all frequency information at that time bin, before convolving the near-by time information in the following layer. This ensures the two dimensions are treated differently and incorporates the domain specific assumptions. Specifically, that low frequencies can be correlated to harmonics across the frequency spectrum, and that notes can change quickly in time.

After each layer, batch-normalization and a hyperbolic tangent activation is used, except the final layer which used a sigmoid activation function and no batch-normalization. Additionally, 2 skip connections were added from the output of the first layer (after batch-normalization and activation) to the output of the fifth layer (before batch-normalization and activation), and from the output of the second layer to the output of the fourth layer similarly. The skip connections represent alternate paths the data can flow through and can make the network easier to train [32]. In our case, these connections are mathematically given by the sum of matrices using the standard matrix addition. Outputs between layers were reshaped as necessary. This gives a total of 427 573 learnable parameters.

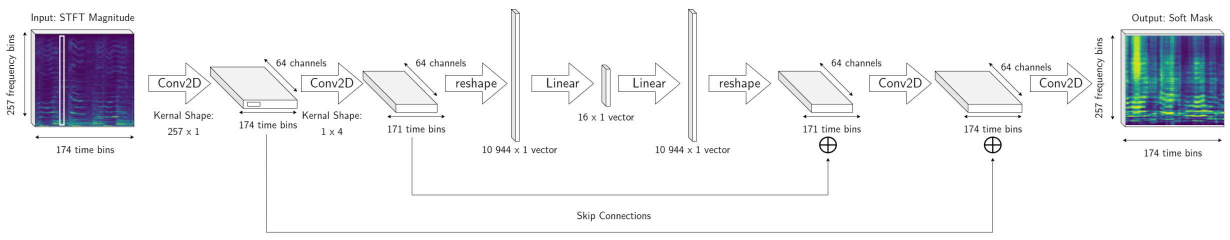


Figure 5.5: **Diagram of Neural Network from Section 5.3:** A sample input STFT magnitude in shown to the left, where a 257×1 kernal is shown in white atop. The skip connections are shown with the symbol $\oplus$ to indicate the two matrices are summed. A sample output soft mask is shown to the right.

Layer	**1**	**2**	**3**	**4**	**5**	**6**
Type	Conv2D	Conv2D	Linear	Linear	ConvT2D	ConvT2D
Input Channels	1	64	10 944	16	64	64
Output Channels	64	64	16	10 944	64	1
Kernel Shape	257×1	1×4	N/A	N/A	1×4	257×1

Table 5.1: **Layers used in the neural network from Section 5.3**. Conv2D stands for 2 dimensional convolutions, and ConvT2D are the transposed convolutions. All convolutional layers use the default stride of 1.

Neural Network Output. The output of the neural network is the matrix $\boldsymbol{M}_v \in \mathbb{C}^{\Omega \times H}$ which has entries between 0 and 1 corresponding to the vocal mask. This is enforced by the output layer's sigmoid activation function which has a range between 0 and 1. The accompaniment mask is the element-wise subtraction $(\boldsymbol{M}_a)_{ij} = 1 - (\boldsymbol{M}_v)_{ij}$. We use these masks to extract the source STFT as

$$(\boldsymbol{S}_k)_{ij} = (\boldsymbol{M}_k)_{ij} (\boldsymbol{Y})_{ij} . \tag{5.2}$$

The source in the time domain can be recovered by applying the inverse STFT.

This STFT masking approach assumes the source's STFT complex phase is identical to the input's STFT. If the sources do not overlap in the time-frequency domain, this masking approach does reproduce the correct phase. When sources overlap, it is possible for their phases to be different from each other, and different from the mixture's STFT. Attempting to reconstruct the correct phase from an STFT is a challenging problem on its own and is necessary for high quality signal recovery [28, 40, 56]. In most cases, this simple approach is sufficient to extract reasonable source's signals in the time domain [48].

Training & Implementation. The loss function used between the predicted outputs $\widetilde{\boldsymbol{S}}_v^{(n)}$ and $\widetilde{\boldsymbol{S}}_a^{(n)}$ and the ground truth sources $\boldsymbol{S}_v^{(n)}$ and $\boldsymbol{S}_a^{(n)}$ is

$$\mathcal{L} = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{2}\sum_{k\in\{v,a\}}\frac{\sum_{ij}\left||(\boldsymbol{S}_k^{(n)})_{ij}|^{\frac{1}{2}} - |(\widetilde{\boldsymbol{S}}_k^{(n)})_{ij}|^{\frac{1}{2}}\right|^2}{\sum_{ij}|(\boldsymbol{S}_k^{(n)})_{ij}|} \tag{5.3}$$

where the superscript (n) denotes the different elements from the training set. The loss function is minimized and network parameters trained using the ADAM Optimizer [39] from PyTorch [54] with a learning rate of $\eta = 5 \times 10^{-4}$ and batch size 20. The Pytorch package is also used to generate the model, and the nussl package is used to process and handle the data [47].

5.3.2 Results

We now detail the results of training. As seen in Figure 5.6, the model is able to learn the data set and the training loss (blue) levels out around 3000 iterations. The validation loss (green) converges to its minimum after only 500. This highlights the difficulty the networks has in generalizing the training set to any song.

Figure 5.8 compares the learned vs the ground truth masks for a sample song clip from the test set. The input spectrogram is shown in Figure 5.7. Results are similar across this set so we select this sample to showcase the general behaviour of the trained model. The full results and code can be explored on GitHub.[2]

The most striking difference between the learn and ideal masks is their contrast between low and high values. This can be interpreted as how *confident* the model is at labelling a particular time-frequency bin as vocals or accompaniment. The ideal masks contain values that mostly near-zero or near-one indicating that particular time-frequency bin belongs to exclusively the vocals or accompaniment. The learned masks are significantly

[2]https://github.com/njericha/masters-thesis

softer and contain few near-zero or near-one regions, and many regions closer to 0.5. The model can confidently identify the lowest frequencies as accompaniment, as well as the first few harmonics belonging to the vocals. It should be expected that the model identifies the lowest frequencies as accompaniment no matter the input song since singers cannot produce frequencies in this range. The impressive part of the learned mask is how well the network is able to identify the harmonics of the voice, especially around the time 1 s.

We now turn our attention to the resulting spectrograms after the masks from Figure 5.8 have been applied to the input spectrogram shown in 5.7. Figure 5.9 compares these resulting spectrograms with the ground truth sources. The bottom left figure highlights how the model correctly reproduces the low frequency information in the accompaniment, and the top right shows the reproduction of the lower few vocal harmonics. Figure 5.9 more clearly shows the *bleeding* of the vocals in the accompaniment track since the highest vocal harmonics can be seen in the bottom left plot. This is challenging for any model to identify for a few reasons. Both the percussion hits and highest vocal harmonics share these frequencies, so it is difficult to identify which time-frequency bin belongs to which source. This is made more difficult by the fact that higher harmonics tend to have a smaller amplitude than the lower frequencies. The top right spectrogram showcases this clearly. This means the loss function does not penalize errors in the highest frequencies as much as lower frequency errors. We could account for this by re-scaling the STFT magnitudes based on their frequency, but this introduces a new hyperparameter which is possibly different for every song and could be challenging on its own to learn.

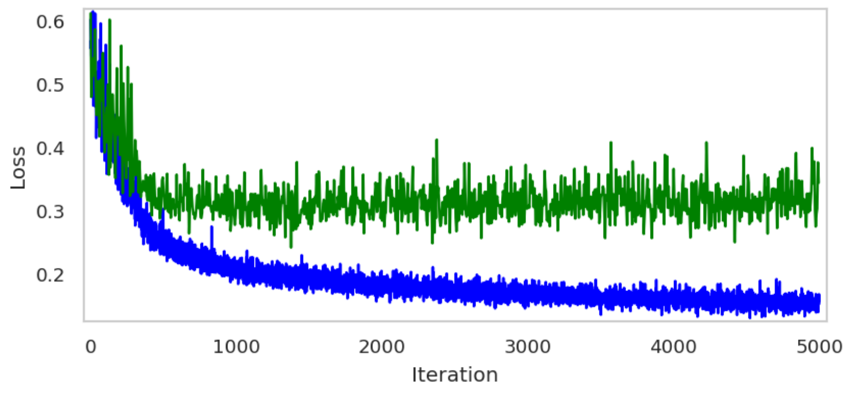


Figure 5.6: **Neural Network Loss from Section 5.3:** Training (blue) and validation (green) loss from the neural network after 5000 iterations.

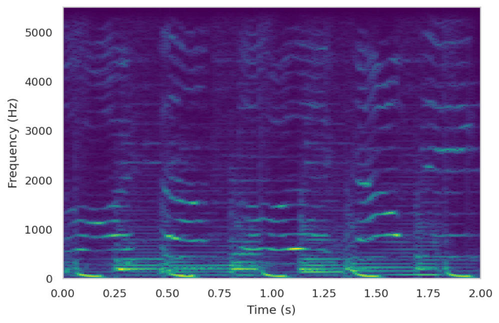


Figure 5.7: **Sample Input from Section 5.3:** The sample input spectrogram from the testing set in Section 5.3. The learned and ideal source masks are shown in Figure 5.8 and the resulting spectrograms are shown in 5.9.

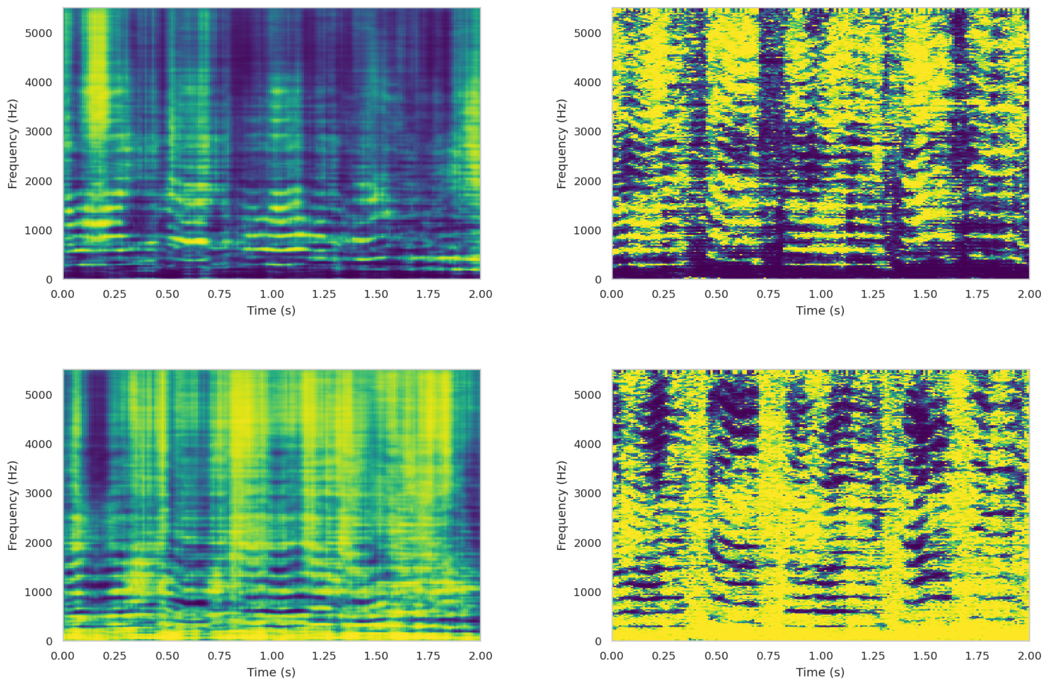


Figure 5.8: **Predicted and Ideal Masks from Section 5.3:** Top Row: Learned vocal mask (left) and ideal vocal mask (right). Bottom Row: Learned accompaniment mask (left), ideal accompaniment mask (right). Brighter pixels indicate values closer to 1 and darker pixels indicate values closer to 0.

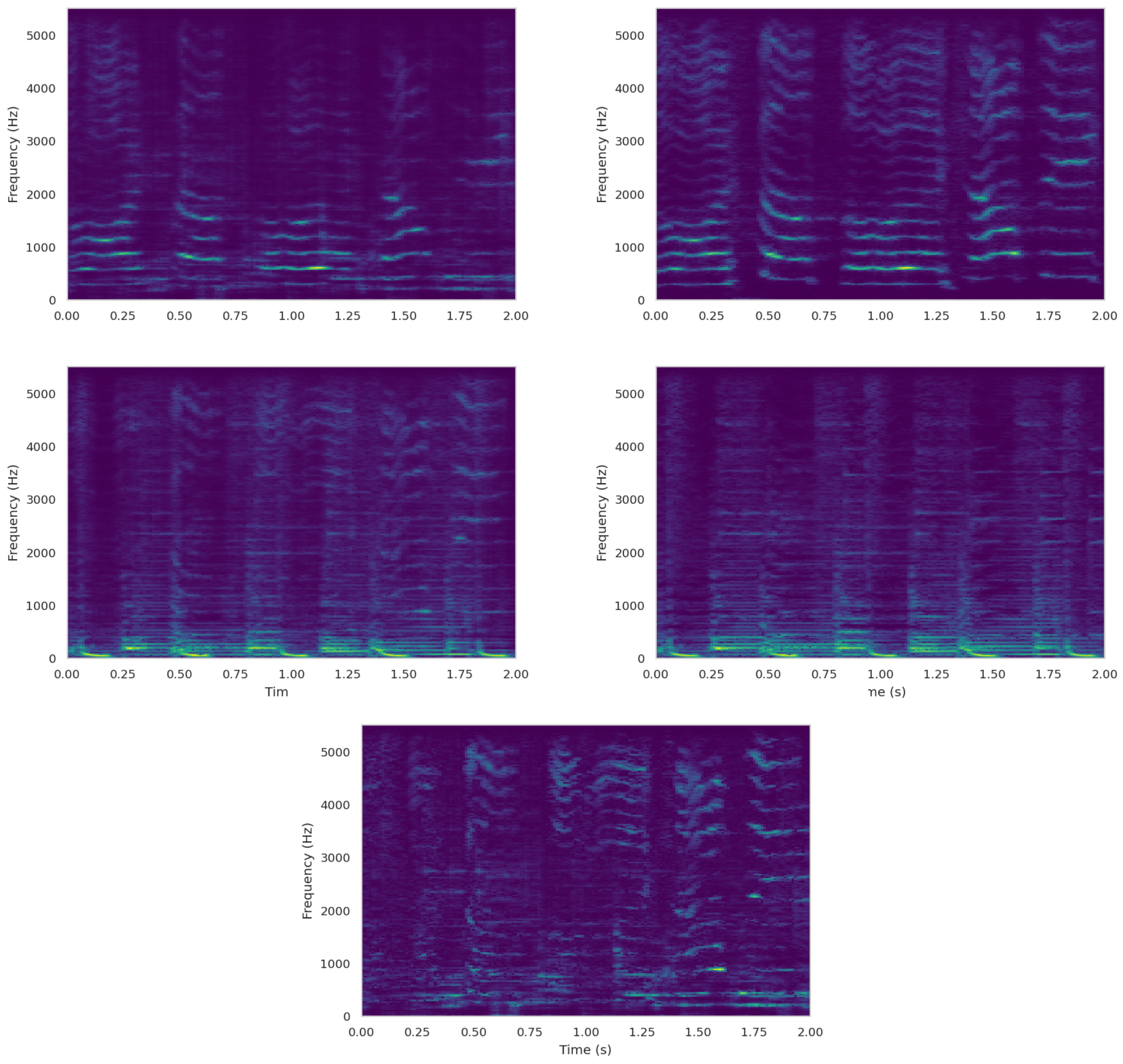


Figure 5.9: **Predicted and True Spectrograms from Section 5.3:** Top Row: Learned spectrogram of the vocals (left) and ground truth vocals spectrogram (right). Middle Row: Learned spectrogram of the accompaniment (left) and ground truth accompaniment spectrogram (right). Bottom: Absolute difference between the learned and true spectrograms. Note this difference is the same for the vocals and accompaniment since the learned sources and true sources both sum to the input spectrogram. The difference plot is re-scaled to highlight detail. Brighter pixels indicate larger values in all plots.

Chapter 6

Final Remarks

6.1 Conclusion

We explore the signal decomposition problem and previous methods used on one-dimensional signals including EMD, EEMD, CEEMDAN, EWT, VMD, SST, and neural networks. The SRMD method is proposed by combining a random feature model with a suitable clustering algorithm. SRMD is shown to outperform popular methods on a variety of mathematical examples from the literature. The method has many favourable attributes from reconstruction error guarantees, efficient use of representation and decomposition algorithms, and flexibility when side information such as maximum frequency, noise level, or number of modes is known. The recommended hyperparameter values are typically sufficient for representing and decomposing signals. We acknowledge the limitations of the model including 1) well-separated and sparse IMF modes are needed for reliable decomposition and 2) fine tuning of the number of features and neighbourhood radius hyperparameters may be required for sensible results. Extensions to real world data, such as astronomical and musical signals, are also considered. The signal decomposition problem in the musical setting is further discussed by examining domain specific assumptions and challenges. Musical decomposition with SRMD is attempted on a simple song, however the method's limitations cause poor performance for typical songs. A neural network model is therefore created to tackle musical decomposition, specifically the sub-problem of vocal-accompaniment separation. The architecture is based on an autoencoder network model with skip-connections, and is able to produce reasonable STFT vocal masks for the training set. The network shows some success in generalizing to song clips it is not trained on but more work is needed to minimizing bleeding between the vocals and accompaniment.

6.2 Future Work

Future work can be done on expanding SRMD to two dimensional signals for use in imaging applications. More attention can be given to the reconstruction stage of the algorithm and used for compression of signals. The reconstruction error could also be improved by adding an additional step after SPGL1 to scale the non-zero coefficients in an effort to remove the downward bias from the BPDN problem. It would also be worth investigating how the SRMD representation can be used to perform style transfer if the learned non-zero features can be mapped to another set of non-zero features.

Given the modular nature of the SRMD framework, many aspects can be modified that may be worth exploring. For example, the features can be generated using a variety of windows other than a simple Gaussian such as the Hamming, Blackman-harries, or Flat Top, generated from a family of wavelets where the random parameters are replaced by the shift and scale. When working with discontinuous signals, functions such as the Haar wavelets (i.e. square function) instead of windowed sinusoidal functions may lead to sharper time-frequency representations with fewer artifacts. In the case of musical decomposition, features that take into consideration the harmonics of common sources may lead to sparser representations. For example, combining pure tones with other sinusoids at integer multiple frequencies. The appropriate linear combination could be learned by observing the natural amplitude decay of these harmonics in the instrument we wish to extract. It is also worth considering alternate clustering methods such as OPTICS or HDBSCAN that may reliably identify the modes of interest on a wider range of hyperparameters. Lastly, the decomposition stage could consider a non-zero features belongs to multiple clusters if the modes intersect, in which case the coefficient for that feature can be split between the modes it belongs to.

Variations with the neural network architecture and training can also be considered. Sparse regularization could help prioritize learning important features in the song. Adding a step to estimate the source's phase—rather than assuming it is identical to the input's STFT—could also produce higher fidelity waveforms. Finally, a hybrid approach may combine the sparse representation of the SRMD method, with the training of a neural network to reliably identify the constituent sources for the signal decomposition problem.

References

[1] François Auger, Patrick Flandrin, Yu-Ting Lin, Stephen McLaughlin, Sylvain Meignen, Thomas Oberlin, and Hau-Tieng Wu. Time-frequency reassignment and synchrosqueezing: An overview. *IEEE Signal Processing Magazine*, 30(6):32–41, 2013.

[2] Francis Bach. On the equivalence between kernel quadrature rules and random feature expansions. *The Journal of Machine Learning Research*, 18(1):714–751, 2017.

[3] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.

[4] Ernesto G. Birgin, José Mario Martínez, and Marcos Raydan. Nonmonotone spectral projected gradient methods on convex sets. *SIAM Journal on Optimization*, 10(4):1196–1211, 2000.

[5] Hans-Dieter Block. The perceptron: A model for brain functioning. i. *Reviews of Modern Physics*, 34(1):123, 1962.

[6] A. Boggess and F.J. Narcowich. *A First Course in Wavelets with Fourier Analysis*. Wiley, 2011.

[7] G Larry Bretthorst. Nonuniform sampling: Bandwidth and aliasing. *AIP conference proceedings*, 567(1):1–28, 2001.

[8] T Tony Cai, Guangwu Xu, and Jun Zhang. On recovery of sparse signals via ℓ^1 minimization. *IEEE Transactions on Information Theory*, 55(7):3388–3397, 2009.

[9] Emmanuel J Candes and Terence Tao. Near-optimal signal recovery from random projections: Universal encoding strategies? *IEEE Transactions on Information Theory*, 52(12):5406–5425, 2006.

[10] Jian Cao, Zhi Li, and Jian Li. Financial time series forecasting model based on CEEMDAN and LSTM. *Physica A: Statistical Mechanics and its Applications*, 519:127–139, 2019.

[11] Vinícius R. Carvalho, Márcio F.D. Moraes, Antônio P. Braga, and Eduardo M.A.M. Mendes. Evaluating five different adaptive decomposition methods for EEG signal seizure detection and classification. *Biomedical Signal Processing and Control*, 62:102073, 2020.

[12] Zhijun Chen and Hayden Schaeffer. Conditioning of random feature matrices: Double descent and generalization error. *arXiv preprint arXiv:2110.11477*, 2021.

[13] Ingrid Daubechies, Jianfeng Lu, and Hau-Tieng Wu. Synchrosqueezed wavelet transforms: An empirical mode decomposition-like tool. *Applied and Computational Harmonic Analysis*, 30(2):243–261, 2011.

[14] Chris Ding, Xiaofeng He, and Horst D. Simon. On the equivalence of nonnegative matrix factorization and spectral clustering. In *Proceedings of the 2005 SIAM International Conference on Data Mining (SDM)*, pages 606–610, 2005.

[15] Andreas Doll, Matteo Ravasi, and David Relyea. SPGL1: A solver for large-scale sparse reconstruction, 2019. https://pypi.org/project/spgl1/.

[16] Konstantin Dragomiretskiy and Dominique Zosso. Variational mode decomposition. *IEEE Transactions on Signal Processing*, 62(3):531–544, 2013.

[17] Igor M Dremin, Oleg V Ivanov, and Vladimir A Nechitailo. Wavelets and their uses. *Physics-Uspekhi*, 44(5):447–478, may 2001.

[18] Weinan E, Chao Ma, Stephan Wojtowytsch, and Lei Wu. Towards a mathematical understanding of neural network-based machine learning: what we know and what we don't. *arXiv preprint arXiv:2009.10713*, 2020.

[19] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, page 226–231. AAAI Press, 1996.

[20] B. P. Abbott et. al. Observation of gravitational waves from a binary black hole merger. *Phys. Rev. Lett.*, 116:061102, 2 2016.

[21] M. Jalal Fadili, Jean-Luc Starck, Jérôme Bobin, and Yassir Moudden. Image decomposition and separation using sparse representations: An overview. *Proceedings of the IEEE*, 98(6):983–994, 2010.

[22] Patrick Flandrin, Gabriel Rilling, and Paulo Goncalves. Empirical mode decomposition as a filter bank. *IEEE Signal Processing Letters*, 11(2):112–114, 2004.

[23] Simon Foucart and Holger Rauhut. *A mathematical introduction to compressive sensing*. Springer New York, 2013.

[24] Jérôme Gilles, Giang Tran, and Stanley Osher. 2D empirical transforms. wavelets, ridgelets, and curvelets revisited. *SIAM Journal on Imaging Sciences*, 7(1):157–186, 2014.

[25] Jérôme Gilles. Empirical wavelet transform. *IEEE Transactions on Signal Processing*, 61(16):3999–4010, 2013.

[26] Tom Goldstein and Stanley Osher. The split Bregman method for L1-regularized problems. *SIAM Journal on Imaging Sciences*, 2(2):323–343, 2009.

[27] Allaire Grégoire and Sidi Mahmoud Kaber. *Numerical Linear Algebra*. Springer, 2008.

[28] Daniel Griffin and Jae Lim. Signal estimation from modified short-time Fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 32(2):236–243, 1984.

[29] Karlheinz Gröchenig. *Foundations of Time-Frequency Analysis*. Birkhäuser Boston, Boston, MA, 2001.

[30] Abolfazl Hashemi, Hayden Schaeffer, Robert Shi, Ufuk Topcu, Giang Tran, and Rachel Ward. Generalization bounds for sparse random feature expansions. *arXiv preprint arXiv:2103.03191*, 2021.

[31] Trevor Hastie, Robert Tibshirani, and Martin Wainwright. *Statistical learning with sparsity: the lasso and generalizations*. Chapman and Hall/CRC, 2019.

[32] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[33] Caiyun Huang. Random sampling and signal reconstruction based on compressed sensing. *Sensors & Transducers*, 170(5):48–53, 2014.

[34] Norden E. Huang, Zheng Shen, Steven R. Long, Manli C. Wu, Hsing H. Shih, Quanan Zheng, Nai-Chyuan Yen, Chi Chao Tung, and Henry H. Liu. The empirical mode decomposition and the Hilbert spectrum for nonlinear and non-stationary time series analysis. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 454(1971):903–995, 1998.

[35] Zhong-lai Huang, Jianzhong Zhang, Tie-hu Zhao, and Yunbao Sun. Synchrosqueezing S-transform and its application in seismic spectral decomposition. *IEEE Transactions on Geoscience and Remote Sensing*, 54(2):817–825, 2015.

[36] A. Jansson, E. Humphrey, N. Montecchio, R. Bittner, A. Kumar, and T. Weyde. Singing voice separation with deep U-Net convolutional networks. October 2017.

[37] Ertug Karamatli, Ali Taylan Cemgil, and Serap Kirbiz. Audio source separation using variational autoencoders and weak class supervision. *IEEE Signal Processing Letters*, 26(9):1349–1353, Sep 2019.

[38] Mark Kerins. *Beyond Dolby (stereo): cinema in the digital sound age*. Indiana University Press, 2010.

[39] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[40] Martin Krawczyk and Timo Gerkmann. STFT phase reconstruction in voiced speech for an improved single-channel speech enhancement. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22(12):1931–1940, 2014.

[41] Dawid Laszuk. Python implementation of empirical mode decomposition algorithm. https://github.com/laszukdawid/PyEMD, 2017.

[42] Zhu Li, Jean-Francois Ton, Dino Oglic, and Dino Sejdinovic. Towards a unified analysis of random Fourier features. In *International Conference on Machine Learning*, pages 3905–3914. PMLR, 2019.

[43] Jing Lin and Liangsheng Qu. Feature extraction based on Morlet wavelet and its application for mechanical fault diagnosis. *Journal of Sound and Vibration*, 234(1):135–148, 2000.

[44] Wei Liu and Wei Chen. Recent advancements in empirical wavelet transform and its applications. *IEEE Access*, 7:103770–103780, 2019.

[45] Yi Luo and Nima Mesgarani. Conv-TasNet: Surpassing ideal time–frequency magnitude masking for speech separation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 27(8):1256–1266, aug 2019.

[46] Wolfgang Maass and Henry Markram. On the computational power of circuits of spiking neurons. *Journal of Computer and System Sciences*, 69(4):593–616, 2004.

[47] Ethan Manilow, Prem Seetharaman, and Bryan Pardo. The northwestern university source separation library. In . Proceedings of the 19th International Society of Music Information Retrieval Conference (ISMIR 2018), Paris, France, September 23-27, 2018.

[48] Ethan Manilow, Prem Seetharman, and Justin Salamon. *Open Source Tools & Data for Music Source Separation*. https://source-separation.github.io/tutorial, Oct 2020.

[49] Song Mei, Theodor Misiakiewicz, and Andrea Montanari. Generalization error of random features and kernel methods: hypercontractivity and kernel matrix concentration. *arXiv preprint arXiv:2101.10588*, 2021.

[50] Frank Moosmann, B Triggs, and Frederic Jurie. Randomized clustering forests for building fast and discriminative visual vocabularies. In *NIPS*. NIPS, 2006.

[51] John Muradeli. ssqueezepy. *GitHub*, 2020. https://github.com/OverLordGoldDragon/ssqueezepy/.

[52] Yurii E Nesterov. A method for solving the convex programming problem with convergence rate O(1/kˆ2). In *Dokl. akad. nauk Sssr*, volume 269, pages 543–547, 1983.

[53] Bryan Pardo, Zafar Rafii, and Zhiyao Duan. *Audio Source Separation in a Musical Context*, pages 285–298. Springer Berlin Heidelberg, Berlin, Heidelberg, 2018.

[54] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS-W*, 2017.

[55] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[56] Zdeněk Průša, Peter Balazs, and Peter Lempel Søndergaard. A noniterative method for reconstruction of phase from STFT magnitude. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 25(5):1154–1164, 2017.

[57] Hendrik Purwins, Bo Li, Tuomas Virtanen, Jan Schlüter, Shuo-Yiin Chang, and Tara N. Sainath. Deep learning for audio signal processing. *CoRR*, abs/1905.00078, 2019.

[58] Zafar Rafii, Antoine Liutkus, Fabian-Robert Stöter, Stylianos Ioannis Mimilakis, and Rachel Bittner. The MUSDB18 corpus for music separation, December 2017.

[59] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2007.

[60] Ali Rahimi and Benjamin Recht. Uniform approximation of functions with random bases. In *2008 46th Annual Allerton Conference on Communication, Control, and Computing*, pages 555–561. IEEE, 2008.

[61] Ali Rahimi and Benjamin Recht. Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. *Advances in Neural Information Processing Systems*, 21:1313–1320, 2008.

[62] Nicholas Richardson, Hayden Schaeffer, and Giang Tran. SRMD: Sparse random mode decomposition. *Communications on Applied Mathematics and Computation*, 2022. (Submitted). https://arxiv.org/abs/2204.06108.

[63] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[64] Alessandro Rudi and Lorenzo Rosasco. Generalization properties of learning with random features. In *NIPS*, pages 3215–3225, 2017.

[65] Joerg Sander. *Density-Based Clustering*, pages 349–353. Springer US, Boston, MA, 2017.

[66] Erich Schubert, Sibylle Hess, and Katharina Morik. The relationship of dbscan to matrix factorization and spectral clustering. In *Lernen, Wissen, Daten, Analyse (LWDA) 2018*, 2018.

[67] Bharath Kumar Sriperumbudur and Zoltan Szabo. Optimal rates for random Fourier features. *Advances in Neural Information Processing Systems*, 2015:1144–1152, 2015.

[68] Fabian-Robert Stöter, Stefan Uhlich, Antoine Liutkus, and Yuki Mitsufuji. Open-unmix-a reference implementation for music source separation. *Journal of Open Source Software*, 4(41):1667, 2019.

[69] Xianlun Tang, Wei Li, Xingchen Li, Weichang Ma, and Xiaoyuan Dang. Motor imagery EEG recognition based on conditional optimization empirical mode decomposition and multi-scale convolutional neural network. *Expert Systems with Applications*, 149:113285, 2020.

[70] Gaurav Thakur, Eugene Brevdo, Neven S Fučkar, and Hau-Tieng Wu. The synchrosqueezing algorithm for time-varying spectral analysis: Robustness properties and new paleoclimate applications. *Signal Processing*, 93(5):1079–1094, 2013.

[71] Gaurav Thakur and Hau-Tieng Wu. Synchrosqueezing-based recovery of instantaneous frequency from nonuniform samples. *SIAM Journal on Mathematical Analysis*, 43(5):2078–2095, Jan 2011.

[72] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[73] María E. Torres, Marcelo A. Colominas, Gastón Schlotthauer, and Patrick Flandrin. A complete ensemble empirical mode decomposition with adaptive noise. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4144–4147. IEEE, 2011.

[74] E. van den Berg and M. P. Friedlander. Probing the pareto frontier for basis pursuit solutions. *SIAM Journal on Scientific Computing*, 31(2):890–912, 2008.

[75] E. van den Berg and M. P. Friedlander. SPGL1: A solver for large-scale sparse reconstruction, December 2019. https://friedlander.io/spgl1.

[76] Ehud Weinstein, Meir Feder, and Alan V Oppenheim. Multi-channel signal separation by decorrelation. *IEEE transactions on Speech and Audio Processing*, 1(4):405–413, 1993.

[77] Hao Wu, Bo Zhang, Tengfei Lin, Fangyu Li, and Naihao Liu. White noise attenuation of seismic trace by integrating variational mode decomposition with convolutional neural network. *GEOPHYSICS*, 84(5):V307–V317, 2019.

[78] Zhaohua Wu and Norden E. Huang. Ensemble empirical mode decomposition: a noise-assisted data analysis method. *Advances in Adaptive Data Analysis*, 1(1):1–41, 2009.

[79] Haizhao Yang. Synchrosqueezed wave packet transforms and diffeomorphism based spectral analysis for 1d general mode decompositions. *Applied and Computational Harmonic Analysis*, 39(1):33–66, 2015.

[80] Ozgur Yilmaz and Scott Rickard. Blind separation of speech mixtures via time-frequency masking. *IEEE Transactions on Signal Processing*, 52(7):1830–1847, 2004.

[81] D. Zennaro, P. Wellig, V.M. Koch, G.S. Moschytz, and T. Laubli. A software package for the decomposition of long-term multichannel EMG signals using wavelet coefficients. *IEEE Transactions on Biomedical Engineering*, 50(1):58–69, 2003.

[82] Weiqiang Zhu, S. Mostafa Mousavi, and Gregory C. Beroza. Seismic signal denoising and decomposition using deep neural networks. *IEEE Transactions on Geoscience and Remote Sensing*, 57(11):9476–9488, 2019.

Appendices

Appendix A

Proof of Proposition 1

Assume $f(t) = a(t)\sin(\phi(t))$ is an IMF where, a and ϕ have continuous second derivatives, $a(t)$ and $\phi'(t)$ vary $\tilde{\varepsilon}$ slower than $\phi(t)$, $\tau \in \mathbb{R}$, and $g(t) = A\sin(\omega t + \varphi)$ is a pure tone with $A = a(\tau)$, $\omega = \phi'(\tau)$, and $\varphi = \phi(\tau) - \tau\phi'(\tau)$.

Let $\varepsilon > 0$. Since a and ϕ have continuous second derivatives, the functions g, g', g'' and f, f', f'' are continuous by composition of continuous functions. Let δ_g be such that $|t - \tau| < \delta_g$ implies $|g(t) - g(\tau)| < \varepsilon_g = \varepsilon/2$. Similarly with $\delta_{g'}, \delta_{g''}, \delta_f, \delta_{f'}$, and $\delta_{f''}$ so that $\varepsilon_{g'}, \varepsilon_{g''}, \varepsilon_f, \varepsilon_{f'}$, and $\varepsilon_{f''} = \varepsilon/2$. Finally, suppose

$$|t - \tau| < \delta = \min\{\delta_g, \delta_{g'}, \delta_{g''}, \delta_f, \delta_{f'}, \delta_{f''}\}.$$

We then have, for f and g,

$$\begin{aligned}
|f(t) - g(t)| &= |f(t) - f(\tau) + f(\tau) - g(t) + g(\tau) - g(\tau)| \\
&\le |f(t) - f(\tau)| + |g(t) - g(\tau)| + |f(\tau) - g(\tau)| \\
&< \varepsilon/2 + \varepsilon/2 + 0 \\
&= \varepsilon,
\end{aligned}$$

for f' and g',

$$\begin{aligned}
|f'(t) - g'(t)| &\le |f'(t) - f'(\tau)| + |g'(t) - g'(\tau)| + |f'(\tau) - g'(\tau)| \\
&< \varepsilon/2 + \varepsilon/2 + |a'(t)\sin(\phi(t))| \\
&= \varepsilon + |a'(t)||\sin(\phi(t))| \\
&< \varepsilon + \tilde{\varepsilon}a(\tau)\phi'(\tau),
\end{aligned}$$

and for f'' and g'',

$$\begin{aligned}|f''(t)-g''(t)| &\leq |f''(t)-f''(\tau)|+|g''(t)-g''(\tau)|+|f''(\tau)-g''(\tau)|\\ &< \varepsilon/2+\varepsilon/2+|2a'(t)\phi'(t)\cos(\phi(t))+a(t)\phi''(t)\cos(\phi(t))+a''(t)\sin(\phi(t))|\\ &\leq \varepsilon+2|a'(t)|\phi'(t)|\cos(\phi(t))|+a(t)|\phi''(t)||\cos(\phi(t))|+|a''(t)||\sin(\phi(t))|\\ &< \varepsilon+2\tilde{\varepsilon}a(\tau)\phi'(\tau)\phi'(\tau)+\tilde{\varepsilon}a(\tau)\phi'(\tau)^2+\tilde{\varepsilon}a(\tau)\phi'(\tau)^2\\ &= \varepsilon+4\tilde{\varepsilon}a(\tau)\phi'(\tau)^2.\end{aligned}$$

Appendix B

Python Implementation of SRMD

```
import numpy as np
from spgl1 import spg_bpdn
from sklearn.cluster import DBSCAN

twopi = 2 * np.pi

def SRMD(f, t, N_features=None, eps=None, *, max_frq=None, w=0.1, r=0.05,
         threshold=None, frq_scale=None, min_samples=4, seed=None, n_modes=None):
    """Implimentation of the Sparse Random Mode Decomposition algorithm

    Parameter details can be found within the thesis or in the source code on
    GitHub. Note this code is compressed from the implementation used by the
    srmdpy package for brevity.

    Inputs
    ------
    f : numpy array
        The input signal.
    t : numpy array
        Time points the signal f was sampled on.

    Returns
    -------
    modes : numpy array, modes.shape == (m, n_modes_recovered)
        List of modes s1, ..., sk recovered from the decomposition algorithm as
        time series.

    Example
    -------
    >>> import numpy as np
    >>> from srmdpy import SRMD
    >>> t = np.linspace(0,1,num=200)
    >>> f = np.cos(2*np.pi*5*t) + np.sin(2*np.pi*20*t)
    >>> kwargs = {"eps":1, "frq_scale":1, "seed":314}
    >>> modes = SRMD(f, t, **kwargs)
    >>> mode_1, mode_2 = modes[:,1], modes[:,2]
    """
```

```
# Define useful constants
m = len(f)       # Number of data points
L = t[-1] - t[0] # Length of signal in time

# Default parameter Handeling
if N_features is None: N_features = 10 * m
if max_frq    is None: max_frq = 0.5 * m / L
if eps        is None: eps = 0.2 * L
if frq_scale  is None: frq_scale = L / max_frq

# Generate random features
features, (tau, frq, phs) = generate_features(N_features, t, w=w, seed=seed,
                                              max_frq=max_frq)

# Represent f sparsely in terms of the features
weights, _, _, _ = spg_bpdn(features, f, sigma=r*np.linalg.norm(f))
weights = weights.squeeze() # convert shape from (N,1) to (N,)

# Optional thresholding step
abs_wghts = np.abs(weights)
if threshold:
    gate = np.percentile(abs_wghts[abs_wghts != 0], threshold)
    keep_index = abs_wghts >= gate
else:
    keep_index = np.not_equal(abs_wghts, 0)

# Extract desired features
tau, frq, phs = tau[keep_index], frq[keep_index], phs[keep_index]
features, weights = features[:,keep_index], weights[keep_index]

# Cluster near-by features in tau-frq space
X = np.column_stack((tau,frq*frq_scale)) # Package into a 2 column matrix
labels = DBSCAN(eps=eps, min_samples=min_samples).fit(X).labels_

# Extract modes by label
n_labels = len(set(labels)) - (1 if -1 in labels else 0)
modes = np.zeros((m, n_labels))
for i in set(labels):
    if i == -1: continue # Skip features thrown out by DBSCAN
    mode_index = np.equal(labels, i)
    modes[:, i] = features[:,mode_index] @ weights[mode_index]

# Sort modes by their norm in decreasing order
norms = np.linalg.norm(modes, axis=0)
sort_order = np.argsort(norms)[::-1]
modes[:,sort_order]

# Relabel features to match new order
re_label = {k:v for v, k in enumerate(sort_order)}
re_label[-1] = -1
labels = np.array([re_label[l] for l in labels])

# Merge (sum) extra modes
if n_modes and n_labels > n_modes:
    modes = np.hstack((modes[:,:n_modes-1],
                 np.sum(modes[:,n_modes-1:],axis=1,keepdims=True)))
    labels = np.array([(l if l < n_modes else n_modes - 1) for l in labels])

return modes
```

```python
def generate_features(N, t, max_frq=None, w=default_w, seed=None):
    """Generates N random features.

    Given the desired number of features N, generates windowed sinusoidal
    features with random time-shifts, frequencies, and phases evaluated at the
    time points t.

    Inputs
    ------
    N : int
        The number of features to generate.
    t : numpy array
        The time points to compute the value of the features

    Outputs
    -------
    features : numpy array, features.shape == (m, N)
        The value of the features at time points t.
    (tau, frq, phs) : tuple of numpy arrays,
        Time-shifts, frequencies, and phases of the features used in the
        representation of the input f. Arrays have shape (N,).
    """
    def _window(t,w):
        """Creates a truncated gaussian window."""
        # Standard deviation s. Half of the window width w
        s = w / 2

        # Zero time values outside three standard deviations
        domain = (np.sign(3*s + t) + np.sign(3*s - t)) / 2

        return np.exp(-0.5 * (t/s)**2) * domain

    # Constants and argument parsing
    L = t[-1] - t[0]

    if max_frq is None:
        m = len(t)
        max_frq = 0.5 * (m+1) / L  # using m+1 to be slightly above Nyquist rate

    # Generate random times, frequencies, and phases
    rng = np.random.default_rng(seed)
    tau = rng.random((1, N)) * L + t[0]
    frq = rng.random((1, N)) * max_frq
    phs = rng.random((1, N)) * twopi

    _t = np.reshape(t, (-1, 1))

    features = _window(_t - tau, w) * np.sin(twopi*frq*_t + phs)

    # Reshape from (1, N) to (N,)
    tau = tau.squeeze()
    frq = frq.squeeze()
    phs = phs.squeeze()

    return  features, (tau, frq, phs)
```