THE UNIVERSITY OF CHICAGO

# MULTIRESOLUTION MATRIX FACTORIZATION

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

BY

NEDELINA TENEVA

CHICAGO, ILLINOIS
AUGUST 2017

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor, Risi Kondor, for his help, guidance, inspiration and kindness on countless occasions during my Ph.D. His ability to approach a problem from the most simple and intuitive direction drawing analogies between various math and computer science areas has had a profound effect on my attitude towards research. I would be forever grateful for the opportunity to see machine learning through this prism. I am also very thankful for Risi's help with the sometimes trivial logistics involved in being a graduate student.

I would like to thank John Lafferty and Lek-Heng Lim for being on my thesis committee and for the valuable comments and suggestions they had throughout the process.

Finally, I am grateful to all my friends who in the past six years were directly or indirectly involved in helping this thesis happen .

# ABSTRACT

In this thesis we introduce a new type of structure in matrices, called multiresolution factorizability, which is an alternative to the ubiquitous low rank assumption in machine learning and numerical linear algebra. We show the connections between classical Fourier, wavelet and multiresolution analysis — three concepts which have shaped most of applied math in the last couple of decades — and (low rank) matrix factorizations, which are often implicitly present in machine learning algorithms as subroutines and which have been one of the main drivers of the scalability of these algorithms in recent years. We propose several different Multiresolution Matrix Factorization (MMF) algorithms, some of which, like Parallel MMF (publicly available in the form of a C++ software library) scale to modern size data, and demonstrate that MMF can be used for compression of the type of large scale matrices and graphs typically arising in machine learning applications.

# INTRODUCTION

As the size of modern datasets and the need for fast manipulation of large scale data in learning problems grow, there has been an increased interest in developing fast methods for low rank matrix approximations, matrix factorizations, matrix compression, matrix completion and randomized algorithms for matrices. Some of the advantages that these methods offer are: efficient ways of storing large matrices, exploiting the matrix sparsity for downstream algorithms, or speeding up certain matrix operations. Typically, these methods leverage the fundamental theorem of linear algebra and assume that the underlying matrix can be well approximated by another low rank matrix. Yet, for matrices arising from real data, such as graphs, it is often the case that the low rank assumption, despite the computational advantages that it offers, does not suffice to explain many complex and frequently observed phenomena. For one, the matrix being factorized might be close to full rank, or it could have a structure that cannot be fully captured by the notion of rank (e.g., multilevel block structure, rather than low rank structure). Therefore, rank alone is not sufficient to capture the rich structural information encoded in matrices, such as distance or kernel matrices.

Similarly, in machine learning applications, such as clustering and community detection, it has become clear that real graphs (and, as a result, matrices derived from them) have nontrivial intrinsic structure — for example, they exhibit strong locality properties (e.g., the degree distribution of those graphs follow the so called power law, described by Barabási and Albert (1999)), and typically have a multiscale structure (i.e., vertices are organized in a hierarchical, "clusters–of–clusters" manner based on an appropriately chosen distance metric). Disentangling this complex structure is vital for: (i) the construction of efficient and scalable methods for graphs/matrices, and (ii) the characterization and modeling of graphs.

Despite the similar multilevel/multiscale/multiresolution structure of graphs and data matrices agreed upon in these different fields, no unified framework exists yet for analyzing, interpreting and modeling this type of structure in graphs/matrices. The algorithmic framework described in this thesis for tackling this type of structure leverages ideas from

the theory of wavelets and multiresolution analysis. Its broader context includes multilevel and multiscale representations and factorizations of matrices and graphs in machine learning, numerical linear algebra and harmonic analysis. In the last few decades multiscale numerical analysis methods, such as multigrid and multipole ideas (Mallat, 1989; Livne and Brandt, 2012; Greengard and Rokhlin, 1987), have been quite impactful in physics, biology and other fields requiring large scale simulations. Applied math concepts such as Fourier and wavelet transforms have transformed signal, image and audio processing not only because they provide strong theoretical guarantees, but also because these transforms can be very efficiently computed in practice. Similar tools for processing less structured data arising in machine learning, such as matrices and graphs, would allow for more efficient computations in many of the above mentioned matrix and graph applications. While the idea of performing multiresolution analysis on a graph is not new (Coifman and Maggioni, 2006; Hammond et al., 2011; Chen and Maggioni, 2011; Allard et al., 2012), Multiresolution Matrix Factorization (Kondor et al., 2014; Teneva et al., 2016; Kondor et al., 2015a,b) is based on the key and novel observation that performing multiresolution analysis on a graph is equivalent to performing a matrix factorization of its graph Laplacian (or an analogous type of matrix encoding of the graph). The subsequent works of Ong and Lustig (2016); Le Magoarou et al. (2017); Malgouyres and Landsberg (2017) are among some of the notable advances in the theory of graph and matrix transforms.

# CHAPTER 1

# PRELIMINARIES

## 1.1 Overview

In this chapter we introduce notation that we will use throughout the thesis. Chapter 2 begins with an overview of concepts from classical harmonic analysis, such as Fourier transforms, and quickly goes on to review the related idea of multiresolution analysis. The goal of this chapter is to introduce key concepts and notation that we will often refer to in the rest of the thesis. As the title of the thesis itself suggests, the work presented in this thesis is rooted in multiresolution analysis and other related fundamental ideas of functional analysis.

Although Multiresolution Analysis, as defined by Mallat (Mallat, 1989), had a huge impact on applied mathematics in the last couple of decades, there have only recently been attempts to generalize this idea to discrete, unstructured cases, where the underlying representation space is a graph or a nonlinear manifold. In Chapter 3 we investigate how classical harmonic analysis ideas can be extended into "signal processing on graphs", a term coined by Shuman et al. (2013a), and survey several multiresolution on graphs designs.

Chapter 4 reviews various types of multiscale/multilevel matrix factorizations. Some of these factorizations are widely used in machine learning, while others, for the most part, have originated from numerical linear algebra. Nevertheless, what connects these two groups of factorizations is that they exploit some type of multiresolution, multiscale or multilevel structure of the matrix at hand. Additionally, depending on the community they originate from, some of these factorizations might not be typically described as matrix factorizations per se, however, here we will certainly approach them from that perspective.

While the focus of Chapter 3 is on multiresolution analysis on graphs and Chapter 4 mainly covers the somewhat disparate subject of factorizing matrices, in Chapter 5 we bring those two ideas together by introducing and defining the concept of Multiresolution Matrix Factorization (MMF). The key insight behind MMF is the observation that multiresolu-

tion analysis on a graph, and more generally any symmetric matrix derived from a graph (assuming the matrix exhibits a certain type of structure), is equivalent to a matrix factorization obeying certain constraints. We present several algorithms for computing the MMF of symmetric matrices and introduce multiresolution factorizability as an alternative to the ubiquitous low rank assumption in machine learning and numerical linear algebra. This chapter is based on the work originally published in (Kondor et al., 2014).

Classically, one of the most popular applications of wavelet and Fourier transforms is for data compression. Analogously, in the matrix factorization setting, representing a given matrix in some compact, compressed form would allow speeding up many downstream applications that the matrix is involved in (for example, matrix–vector product or matrix inversion which are, of course, the key "ingredients" of machine learning algorithm). Naturally, in Chapter 6 we apply MMF for the compression of matrices derived from graphs (e.g., graph Laplacians and kernel matrices). Chapter 6 also describes various existing matrix factorization techniques for compressing data matrices and demonstrates that, if used as a compression tool, MMF beats existing state-of-the-art factorizations by a significant margin.

Chapter 7 presents a fast parallel MMF algorithm for computing the factorization of large scale matrices and describes the rationale behind the resulting pMMF C++ library. We demonstrate that pMMF is not only a viable compression tool in terms of its accuracy, but also in terms of its computational efficiency. This chapter is based on the work published in (Teneva et al., 2016) and (Kondor et al., 2015a), while the library documentation is publicly available in (Kondor et al., 2015b).

## 1.2 Notation

Let $[n] = \{1, 2, \ldots, n\}$. The $n$ dimensional identity matrix is denoted by $I_n$, unless $n$ is obvious from the context and we omit it, in which case we use $I$. The $i$–th row and $j$–th column of a matrix $M$ are denoted respectively $M_{i,:}$ and $M_{:,j}$. $\uplus$ denotes the disjoint union of two sets, so $S_1 \uplus S_2 \uplus \cdots \uplus S_m = S$ is a partition of the set $S$. $O(n)$ denotes the group of

orthogonal matrices of dimension $n$. Strictly speaking, this group consists of matrices that correspond to pure rotations as well as matrices that correspond to rotations combined with refections.

**Submatrices**. Given a matrix $M \in \mathbb{R}^{n\times m}$ and two sequences of indices $I = (i_1, \ldots, i_k)$ and $J = (j_1, \ldots, j_\ell)$, where $I, J$ are subsets of $[n]$, the submatrix $M_{I,J}$ denotes the matrix $M \in \mathbb{R}^{k\times\ell}$ consisting of the rows $(i_1, \ldots, i_k)$ and the columns $(j_1, \ldots, j_\ell)$ of $M$. The entry at position $(a, b)$ in $M_{I,J}$ will be denoted by $[M_{I,J}]_{a,b}$. Similarly, if we let $P = \{i_1, i_2, \ldots, i_k\} \subseteq [n]$ and $T = \{j_1, j_2, \ldots, j_\ell\} \subseteq [m]$ (assuming $i_1 < i_2 < \cdots < i_k$ and $j_1 < j_2 < \cdots < j_\ell$) then $M_{P,T} \in R^{k\times\ell}$ with entries $[M_{P,T}]_{a,b} = M_{i_a,j_b}$

**Block diagonal matrices**. Given $M_1 \in \mathbb{R}^{n_1\times m_1}$ and $M_2 \in \mathbb{R}^{n_2\times m_2}$, $M_1 \oplus M_2$ is the $(n_1 + n_2) \times (m_1 + m_2)$–dimensional matrix with entries

$$[M_1 \oplus M_2]_{i,j} = \begin{cases} [M_1]_{i,j} & \text{if } i \le n_1 \text{ and } j \le m_1, \\ [M_2]_{i-n_1,j-m_1} & \text{if } i > n_1 \text{ and } j > m_1, \\ 0 & \text{otherwise.} \end{cases}$$

A matrix $M$ is said to be **block diagonal** if it is of the form

$$M = M_1 \oplus M_2 \oplus \cdots \oplus M_p \tag{1.1}$$

for some sequence of smaller matrices $M_1, M_2, \ldots, M_p$. In this thesis we will generally mostly deal with block diagonal matrices in which each of the blocks is square.

For the purposes of this exposition the ordering of the rows/columns of a block diagonal matrix will be irrelevant, so we further relax (1.1) by removing the constraint that each block must involve a contiguous set of indices and and introduce the concept of **generalized block diagonal** matrices defined as

$$M = \oplus_{(i_1^1, i_2^1 \ldots, i_{k_1}^1)} M_1 \oplus {}_{(i_1^2, i_2^2 \ldots, i_{k_2}^2)} M_2 \cdots \oplus {}_{(i_1^p, i_2^p, \ldots, i_{k_p}^p)} M_p. \tag{1.2}$$

Each of the entries of a generalized block diagonal matrix $M$ is then given by

$$M_{a,b} = \begin{cases} [M_u]_{q,r} & \text{if } i_q^u = a \text{ and } i_r^u = b \text{ for some } u, q, r, \\ 0 & \text{otherwise.} \end{cases}$$

We may sometimes abbreviate expression (1.2) by dropping the first $\oplus$ operator and its indices. The reason for this is that $(i_1^1, i_2^1 \ldots, i_{k_1}^1)$ is fully determined by the other index tuples, assuming $i_1^1 < i_2^1 < \cdots < i_{k_1}^1$.

**Tensor products**. Given $M_1 \in \mathbb{R}^{n_1 \times m_1}$ and $M_2 \in \mathbb{R}^{n_2 \times m_2}$, the tensor (sometimes called Kronecker) product matrix $M_1 \otimes M_2$ is an $n_1 n_2 \times m_1 m_2$ matrix whose elements are

$$[M_1 \otimes M_2]_{(i_1-1)n_2+i_2,\,(j_1-1)m_2+j_2} = [M_1]_{i_1,j_1} \cdot [M_2]_{i_2,j_2} \,,$$

with the obvious generalization to $k$–fold products $M_1 \otimes M_2 \otimes \ldots \otimes M_p$. We will sometimes use the multi-index notation $M_{I,J}$ to denote the individual entries of such products, specifically,

$$M_{I,J} = [M_1]_{i_1,j_1} \cdot [M_2]_{i_2,j_2} \cdot \ldots \cdot [M_k]_{i_p,j_p} \,, \tag{1.3}$$

where $I = (i_1, \ldots, i_p) \in [n_1] \times [n_2] \times \ldots \times [n_p]$ and $J = (j_1, \ldots, j_p) \in [m_1] \times [m_2] \times \ldots \times [m_p]$. We define $M^{\otimes p}$ as the $p$–fold product $M \otimes M \otimes \ldots \otimes M$.

# CHAPTER 2

# FOURIER AND MULTIRESOLUTION ANALYSIS

This chapter is intended as a preamble and the concepts introduced here are key to understanding the rationale behind the topics of Chapters 3 and 4, as well as the inspiration behind Multiresolution Matrix Factorization. We review key functional analysis concepts, such as the Fourier transform, wavelets and multiresolution analysis, for functions on the real line as well as the intricacies and challenges involved in the translation of these ideas to the discrete setting.

## 2.1 Classical Fourier Transform

The problem which initiated the study of Fourier analysis is whether a function can be represented as a sum of simpler periodic functions. More specifically, Fourier series is motivated by the idea of breaking down complex periodic functions on the unit circle into a sum (or integral) of "waves", which are represented by sine and cosine functions. This idea can be extended relatively easily from periodic functions on the unit circle to periodic functions on the real line. Somewhat more miraculously, the idea of representing a function as a sum of waves is applicable not just to periodic integrable functions on the real line, but also to *nonperiodic* integrable functions on the real line (see (Stein and Shakarchi, 2011) for review).

The classical **Fourier transform** is defined on the space of periodic functions on the real line with period $2\pi$ (which are equivalent to functions on the unit circle) as

$$\hat{f}(k) = \frac{1}{2\pi} \int_0^{2\pi} e^{-ikx} f(x)\, dx, \qquad k \in \mathbb{Z}, \tag{2.1}$$

or nonperiodic functions on the real line as

$$\hat{f}(k) = \int e^{-2\pi ikx} f(x)\, dx, \qquad k \in \mathbb{R}, \tag{2.2}$$

where $i \coloneqq \sqrt{-1}$ is the imaginary unit. For each $k$, $e^{-2\pi ikx}$ is a periodic function with frequency $k$. Note that the Fourier transform of periodic functions is known as Fourier series.

The values $\hat{f}(k)$ in (2.1) and (2.2) are called, respectively, **Fourier series coefficients** and **Fourier coefficients**. The motivation for this naming convention comes from the fact that these values serve as coefficients in a formula which can recover the original function $f$. The reconstruction of $f$ is called the **inverse Fourier transform**. For the two cases above, the corresponding inverse Fourier transforms are respectively

$$\begin{aligned} f(x) &= \sum_{k=-\infty}^{\infty} \hat{f}(k)\, e^{ikx}, \\ f(x) &= \int \hat{f}(k)\, e^{2\pi ikx}\, dk. \end{aligned} \tag{2.3}$$

Intuitively, the $k$ values can be thought of as the "frequency" and $x$ values can be thought of as the "time", hence the representations $\hat{f}$ and $f$ are called the "frequency domain" and the "time domain", respectively. If the function $f$ is smooth, then the Fourier coefficients decay rapidly and so $f$ can be closely approximated using just the first few low frequency coefficients.

For functions on the real line, Euler's identity $e^{ix} = \cos x + i \sin x$ provides a nice connection between these trigonometric functions and the complex numbers: the sums in (2.3) can be interpreted, for a given $k$, as a weighted average of $f$ with the complex exponentials $e^{2\pi ikx}$ as weights. Due to the derivations in (2.3) involving complex exponentials, the Fourier coefficients are complex valued. Note that the the set of functions that can be recovered from their Fourier transform is dense in $L_2(\mathbb{R})$.

When the function $f$ is discrete, or if it is discretized on a finite set of cardinality $n$, the integrals in (2.1) and (2.2) can be expressed as a finite sum, also called the **discrete**

**Fourier transform** (DFT)

$$\hat{f}(k) = \sum_{x=0}^{n-1} e^{-2\pi ikx/n} f(x), \qquad 0 \le k \le n-1. \tag{2.4}$$

with its corresponding inverse Fourier transform given by

$$f(x) = \frac{1}{n} \sum_{k=0}^{n-1} \hat{f}(k)\, e^{2\pi ikx/n}. \tag{2.5}$$

Another way to think of the discrete Fourier transform (2.4) is in terms of its matrix form — it is equivalent to a linear transformation $\mathcal{F} : \mathbb{C}^n \to \mathbb{C}^n$, and so

$$\hat{f} = \mathcal{F} f, \tag{2.6}$$

where both the original function $f$ and its transform $\hat{f}$ are vectors in $\mathbb{C}^n$, and the **DFT matrix** $\mathcal{F} \in \mathbb{C}^{n \times n}$. The rows of $\mathcal{F}$ contain the vectors forming the **Fourier basis** consisting of the set $\{e^{-2\pi ikx/n}\}_{k \in \{0,1,\ldots,n-1\}}$. In other words, the Fourier transform $\hat{f}$ is just $f$ expressed in a different basis. Multiplication of $f$ by $\mathcal{F}$ on the left allows for an easy conversion between the time and the frequency domain. The **inverse DFT** is given by $\mathcal{F}^{-1}$ and it allows for switching back from the frequency domain to the time domain (i.e., the inverse operation of (2.6)) as follows

$$\mathcal{F}^{-1} \hat{f} = \mathcal{F}^{-1}(\mathcal{F} f) = f. \tag{2.7}$$

Note that, since the Fourier transform is unitary (at least with the convention used in (2.4)), $\mathcal{F}^{-1} = \mathcal{F}^{\dagger}$.

## 2.2 From Fourier to Wavelets

Fourier analysis decomposes signals into different frequency components, but it does not provide any information about the time at which these frequencies occurred. In particular, for

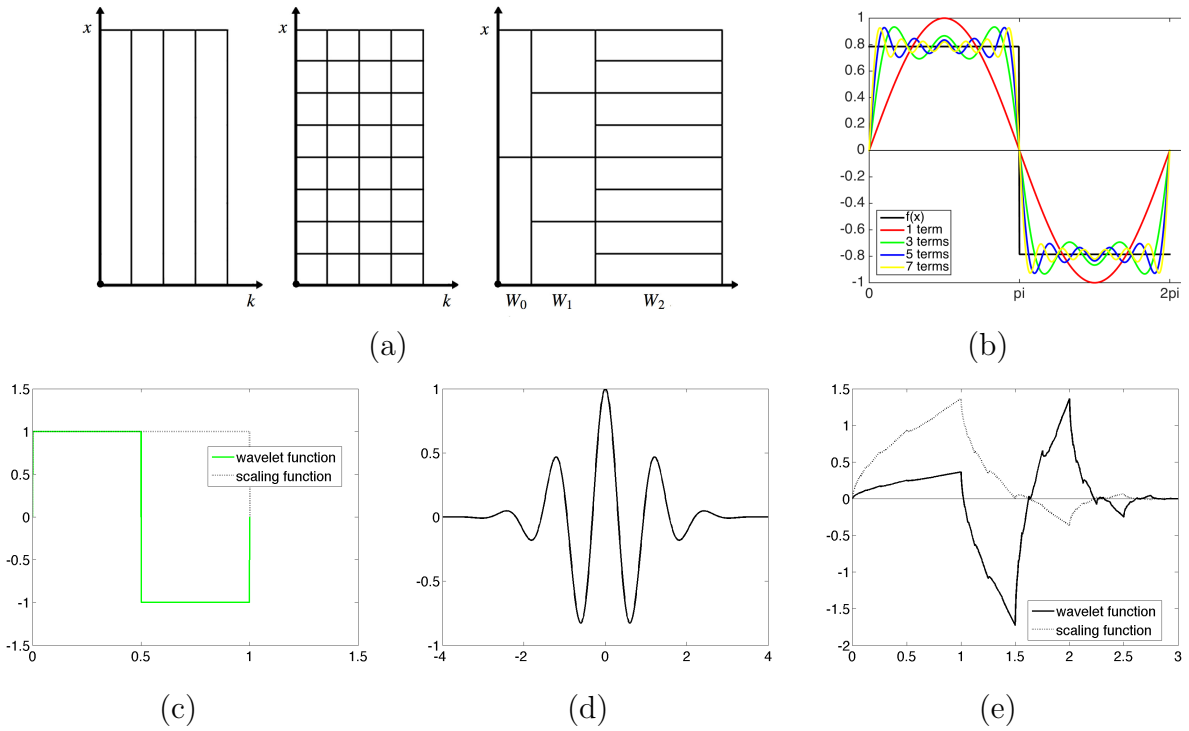



Figure 2.1: **Wavelet concepts and examples**. (a) Fourier transform (right), windowed Fourier transform (middle) and wavelet transform (right) phase planes with frequency $k$ and time $x$ — unlike the Fourier transforms, the wavelet transform has different resolution levels. (b) Discontinuities, such as those in the step function shown in black, induce a characteristic pattern of spurious oscillations and over/underhoots in partial Fourier sums called the Gibbs phenomenon. (c) Haar wavelet and scaling function. (d) Morlet wavelet. (e) Daubechies D2 wavelet and scaling function.

nonperiodic functions, computing the Fourier transform (2.2) would require us to integrate over all times, $x$. The Fourier phase plane shown in the left panel of Figure 2.1(a) illustrates this point — the sinusoidal waves in the Fourier transform are localized in frequency ($k$), but global in time ($x$). As a result, if the frequency content of a signal $f$ varies with time, its Fourier transform does not detect any local time variations and so it cannot be used to analyze nonstationary signals. In order to localize the exponentials in the Fourier transform (2.2) in a certain neighborhood of the time domain, we can use an auxiliary function. Gabor (1946) proposed using a symmetric window function $g(t) = g(-t)$ defined as follows

$$g_{k,u}(x) = e^{ikx} g(x-u), \qquad k, u \in \mathbb{R},$$

where $u$ is a translation parameter and $\|g\|_2 = 1$. One function that can serve as a window is, for example, a Gaussian, in which case the resulting transform is called Gabor transform (Gabor, 1946; Mallat, 2008).

Using the translation of the window $g$ in both time and frequency, we can define the **windowed Fourier transform**, analogously to the Fourier transform (2.2). For a square integrable function on the real line $f \in L_2(\mathbb{R})$ the windowed Fourier transform is

$$Sf(k,u) = \left\langle f, g_{k,u} \right\rangle = \int f(x)\, g(x-u)\, e^{-ikx} dx, \qquad k \in \mathbb{R}, \tag{2.8}$$

where the multiplication by the translated window essentially localizes the Fourier integral in the neighborhood of $u$. The original function can then be recovered by

$$f(x) = \frac{1}{2\pi} \int \int Sf(k,u)\, g(x-u)\, e^{ikx} dk\, du. \tag{2.9}$$

The windowed Fourier transform finds applications in many signal processing tasks, such as music and speech recognition, where frequencies vary with time. Yet, it is not appropriate for signals which have short durations of high frequency events, for example, signals arising in seismology or molecular dynamics. As illustrated in the middle panel in Figure 2.1(a), this is due to the observation that as the frequency parameter $k$ increases, the transform translates in frequency, but always by the same width — in other words, it lacks resolution. A transform which has different levels of resolution, would be better suited to capture the short–lasting high frequency and long–lasting low frequency parts of a signal. This type of transform is called a wavelet transform and its phase plane is shown on the right panel of Figure 2.1(a). Starting at the beginning of the 80's, the wavelet transform and multiresolution ideas, which we discuss in Section 2.4, were formalized by Gabor, Morlet and Grossman, among others — see (Mallat, 2008) for a historical review.

## 2.3 Smoothness

One of the central questions of harmonic analysis is how to filter the space of functions on a space $X$ into a nested sequence of spaces according to their smoothness. Before we explain how these nested spaces can be constructed in the next section, we will first define smoothness and, in particular, what it means depending on the type of the starting space $X$.

If $X$ is a Euclidean space or a differentiable (smooth) manifold, which often is the underlying assumption for a lot of problems arising in machine learning, the smoothness of $f : X \to \mathbb{R}$ may be defined in terms of the Laplace operator or the Laplace–Beltrami operator, respectively.

For functions defined on Euclidean space, for example, when $X = \mathbb{R}^d$, the Laplace operator, or the **Laplacian**, is a second order differential operator

$$\Delta = \frac{\partial^2}{\partial x_1^2} + \frac{\partial^2}{\partial x_2^2} + \cdots + \frac{\partial^2}{\partial x_d^2}\,, \tag{2.10}$$

which can be applied to a function $f : X \to \mathbb{R}$ in $d$–dimensional Euclidean space as follows

$$\Delta f = \frac{\partial^2 f}{\partial x_1^2} + \frac{\partial^2 f}{\partial x_2^2} + \cdots + \frac{\partial^2 f}{\partial x_d^2}\,.$$

As a side note, the Laplacian comes up in many applications in physics in the form of the Laplace equation $\Delta f = 0$.

Going back to the general form of the Laplace operator (2.10), we can quantify the smoothness of $f$ by considering the ratio

$$\eta = \frac{\langle f, \Delta f\rangle}{\langle f, f\rangle}, \tag{2.11}$$

where $\langle \cdot, \cdot \rangle$ is the inner product. If the ratio (2.11) is small, we say that the function $f$ is **smooth**. Intuitively, this is equivalent to saying that, when applied to $f$, the Laplace

operator does not change it too much.

In differential geometry, the counterpart of (2.10) is the **Laplace–Beltrami operator**

$$\Delta = \frac{1}{\sqrt{|\det(g)|}} \sum_{i,j=1}^{d} \frac{\partial}{\partial x_j} \sqrt{|\det(g)|}\, g_{ij} \frac{\partial}{\partial x_i}, \tag{2.12}$$

which is just a generalization of the Laplace operator to functions defined on $d$–differentiable manifolds with a metric tensor $g$ (the metric tensor is a generalization of the Euclidean dot product to manifolds). Smoothness is measured analogously to (2.11), except that the Laplace–Beltrami operator is used instead.

In the discrete setting, when $X$ is the vertex set of a finite simple graph on $n$ vertices, the smoothness of $f$ is defined in terms of the discrete, graph analog of the Laplace operator, called the **graph Laplacian** $\mathcal{L}$, which is a symmetric matrix of a certain form (see Section 2.5 for details). In this case (2.11) simplifies to $\eta = \frac{\langle f, \mathcal{L} f \rangle}{\langle f, f \rangle}$.

Another way to think about function smoothness, without involving differentiation, is based on defining the semigroup $\{A_\alpha\}_{\alpha \in [0,+\infty)}$ of positive semi-definite smoothing operators. A natural choice for $A_\alpha$ is the diffusion operator. When applied to a function $f$ on $\mathbb{R}^d$, the diffusion operator takes the form

$$(A_\alpha f)(y) = (2\pi\alpha)^{d/2} \int f(x)\, e^{-2\pi \|x-y\|^2/\alpha}\, dx. \tag{2.13}$$

$A_\alpha$ has exactly the opposite effect to that of $\Delta$ in the sense that $A_\alpha$ tends to smooth functions while $\Delta$ tends to roughen them. So, in contrast to (2.11), here we expect the ratio $\frac{\langle f, A_\alpha f \rangle}{\langle f, f \rangle}$ to be large. On Euclidean spaces, differentiable manifolds and graphs, the diffusion operator and the Laplacian share the same system of eigenvectors (specifically, in the case of the diffusion operator $A_\alpha = e^{-\alpha\Delta}$) so the Laplacian and diffusion based approaches to harmonic analysis are equivalent.

## 2.4 Classical Multiresolution Analysis

Fourier analysis filters the space of square integrable functions $L_2(X)$ defined on $\mathbb{R}$ according to smoothness by explicitly constructing an orthogonal basis of eigenfunctions of the $\Delta$, $\mathcal{L}$ or $A_\alpha$ operators. In this case, however, the eigenvectors, while perfectly localized in frequency, (i.e., corresponding to a single eigenvalue of $\Delta$ or $A_\alpha$) are not at all localized in the spatial domain due to the fact that the Fourier transform (discrete or continuous) is constructed as superpositions of dilations of the function $g(x) = e^{-ix}$. Wavelets, on the other hand, are designed to capture information both in the frequency and in the time (space) domain and so a function in $L_2(X)$ could be better represented by wavelet series expansion in an orthonormal basis generated by the dilation *and* translation of a wavelet function. Wavelets can not only separate a signal $f$ into components at different levels of resolution, but can also differentially resolve different parts of $f$ at different levels of detail. This is because, in contrast to the global nature of Fourier eigenvectors, wavelets follow a "what is local must stay local" philosophy, in the sense that the high frequency basis functions, meant to capture the local behavior of $f$, have small support, and hence do not interact with parts of $f$ that are far away. The concrete problem that sparked the development of the modern theory of wavelets in the early 1980's was the need to accurately model functions, specifically seismological signals consisting of a combination of long smooth segments and sudden discontinuities (Morlet, 1983). Fourier analysis and band–limited functions are ideally suited for decomposing smooth functions. If the function $f$ being approximated is continuous, then the inverse Fourier transform (2.3) converges uniformly to $f(x)$ as more Fourier series coefficients are added. However, if $f$ is discontinuous, then (2.3) converges pointwise, nonuniformly and as a result, displays a characteristic pattern of over/undershoots, even when a large number of coefficients are used. This pattern is known as the Gibbs phenomenon, shown in Figure 2.1(b).

The theoretical foundation of wavelets is **Multiresolution analysis** (MRA) which, as an alternative to Fourier analysis, decomposes the space $L_2(X)$ into a wide range of scales

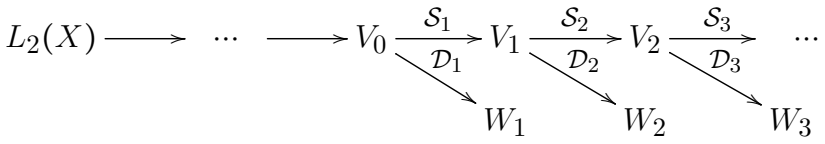



Figure 2.2: **MRA**. Multiresolution analysis repeatedly splits the function spaces $V_0, V_1, \ldots$ into a smoother part $V_{\ell+1}$ and a rougher part $W_{\ell+1}$. $\mathcal{S}_\ell$ and $\mathcal{D}_\ell$ denote the scaling and the detail transform, respectively.

by constructing a nested sequence of subspaces

$$L_2(X) \supset \ldots \supset V_{-1} \supset V_0 \supset V_1 \supset V_2 \supset \ldots \supset \{0\} \tag{2.14}$$

of increasing smoothness. Each $V_\ell$ is split into a smoother (or low frequency) part $V_{\ell+1}$ and rougher (or high frequency) part $W_{\ell+1}$ (Figure 2.2). In the sequence of subspaces, the smaller $\ell$ is, the shorter the length scale over which functions in $V_\ell$ vary. In other words, if $k > \ell$, the space resolution $2^\ell$ of the $\ell$–th subspace is higher than the resolution $2^k$ of the $k$-th subspace. We will refer to each of the subspaces $V_\ell$ as a **level of resolution**.

Mallat (1989) proposed defining MRA on $L_2(\mathbb{R})$ (i.e., $X = \mathbb{R}$) as a sequences of embedded spaces using the following axioms

A1. $\bigcap_\ell V_\ell = \{0\}$, i.e., when the resolution decreases to zero, the approximation signal contains less and less information and converges to zero.

A2. $\bigcup_\ell V_\ell$ is dense in $L_2(\mathbb{R})$, i.e., when the resolution increases to infinity, the approximation signal contains more and more information.

A4. Translation invariance of $V_\ell$: $f \in V_\ell \iff f'(x) = f(x - 2^\ell m) \in V_\ell$, for all $m \in \mathbb{Z}$.

A4. Dilation property: $f \in V_\ell \iff f'(x) = f(2x) \in V_{\ell-1}$.

A5. There exists a function $\phi(x)$ such that $\{\phi(x - m)\}_{m \in \mathbb{Z}}$ is an orthonormal basis (and more generally a Riesz basis) for $V_0$. This property is key for the construction of a basis for each of the $V_\ell$ subspaces.

A direct consequence of the existence of a so-called "father" wavelet $\phi$, as specified by axiom A5, is the observation that $V_\ell$ is spanned by an orthonormal basis of the form

$$\Phi_\ell = \{\phi_m^\ell(x) = 2^{-\ell/2}\phi(2^{-\ell}x - m)_{m\in\mathbb{Z}}\}. \tag{2.15}$$

However, the totality of all these orthonormal bases, consisting of the set

$$\{\phi_m^\ell(x) = 2^{-\ell/2}\,\phi(2^{-\ell}x - m)\}_{m,\ell\in\mathbb{Z}}\,,$$

is not an orthonormal basis for $L_2(X)$ because the **approximation** subspaces $V_\ell$ are not mutually orthogonal. In order to avoid this problem, we define the wavelet **detail** subspace $W_\ell$ to be the orthogonal complement of $V_\ell$ in $V_{\ell-1}$. In other words,

$$V_\ell = W_{\ell+1} \bigoplus V_{\ell+1}, \tag{2.16}$$

where $\oplus$ denotes the direct sum of mutually orthogonal subspaces. For an approximation subspace at a particular level of the hierarchy, w.l.o.g let that subspace be $V_0$, and $\ell'$ levels of resolution, it follows that $V_0 = V_{\ell'} \oplus (\bigoplus_{\ell=0}^{\ell'} W_\ell)$. As $\ell' \to \infty$, it follows from the MRA axioms A1 and A2 above that

$$L_2(X) = \bigoplus_{\ell\in\mathbb{Z}} W_\ell\,,$$

which means that if a function belongs to a set of functions which is dense in $L_2(X)$, it can be expressed as a sum of functions in the orthogonal detail subspaces $W_\ell$.

It can be further shown that the scaling function $\phi$ determines the "mother" wavelet $\psi$ such that $\{\psi(x-m)\}_{m\in\mathbb{Z}}$ is an orthonormal basis of $W_0$. Similarly to the way $\{\phi(x-m)\}_{m\in\mathbb{Z}}$ can be used to construct an orthonormal basis for some $V_\ell$ in (2.15), it can be shown that $W_\ell$ is spanned by an orthonormal basis

$$\Psi_\ell = \{\psi_m^\ell(x) = 2^{-\ell/2}\,\psi(2^{-\ell}x - m)_{m\in\mathbb{Z}}\}. \tag{2.17}$$

Various alternatives have been proposed for $\psi$, including the simplest and earliest wavelet, the Haar wavelet, shown in Figure 2.1(c). One of the disadvantages of the Haar wavelet is that it has discontinuities, which renders it unsuitable for a basis for smooth functions. Instead, many other smoother wavelets are used in practice — classical smooth wavelets include those named after Meyer and Morlet (Figure 2.1(d)), and the famous Daubechies' wavelets (Daubechies, 1988), shown in Figure 2.1 (e), which have the property of being *both* compactly supported and having a fixed number of vanishing moments (see (Mallat, 2008; Daubechies, 1992) for an overview). However, in general, $\psi$ is chosen as a function that is fairly narrow in the frequency domain (therefore, is wave–like) and localized in time (space). The higher $\ell$ is, the more (2.17) dilates the mother wavelet $\psi$, and so $\psi_m^\ell$ becomes smoother. Unlike in the Fourier transform, however, the high frequency (i.e., low $\ell$) wavelets are also very narrow, so they only contribute to describing $f$ in a small neighborhood. In contrast to (2.3), here the scale parameter $\ell$ is discrete. There is an alternative approach to wavelets based on the so-called continuous wavelet transform, where the levels form a continuum. The corresponding theory has deep roots in harmonic analysis, but it computationally less attractive as the wavelet frames that it leads to are highly overcomplete.

The $V_j$ subspaces can be used to approximate general functions by defining projections onto these subspaces. Since the union of all $V_j$ is dense in $L_2(X)$, it is guaranteed that any function in $L_2(X)$ can be approximated arbitrarily close by such a projection. Let $\mathcal{P}_\ell f$ denote the orthogonal projection of $f$ onto $V_\ell$, i.e.,

$$\mathcal{P}_{V_\ell} f = \sum_{m\in\mathbb{Z}} \langle \phi_m^\ell, f\rangle \, \phi_m^\ell. \tag{2.18}$$

Note that $\mathcal{P}_\ell f$ is the best approximation of $f$ in the subspace $V_\ell$. From (2.16) is follows that a function $f \in V_\ell$ can be expressed as the sum $f = \mathcal{P}_{W_{\ell+1}} f + \mathcal{P}_{V_{\ell+1}} f$. In turn, after applying (2.16) to $\mathcal{P}_{V_{\ell+1}} f$, it decomposes into $\mathcal{P}_{V_{\ell+1}} f = \mathcal{P}_{W_{\ell+2}} + \mathcal{P}_{V_{\ell+2}} f$, leading to $f = \mathcal{P}_{W_{\ell+1}} f + \mathcal{P}_{W_{\ell+2}} f + \mathcal{P}_{V_{\ell+2}} f$, and so on until after $t$ levels $f = \mathcal{P}_{W_{\ell+1}} f + \mathcal{P}_{W_{\ell+2}} f + \cdots + \mathcal{P}_{W_{\ell+t}} f + \mathcal{P}_{V_{\ell+t}} f$. If

we let $f : X \to \mathbb{R}$ be a function residing at a particular level of the hierarchy, w.l.o.g. $f \in V_0$, then its wavelet transform is

$$\begin{aligned} f(x) &= (\sum_{\ell=1}^{\infty} \mathcal{P}_{W_\ell} + \mathcal{P}_{V_\infty}) f(x) \\ &= \sum_{\ell=1}^{\infty} \sum_m \langle f, \psi_m^\ell \rangle \, \psi_m^\ell(x) + \sum_m \langle \phi_m^\infty, f \rangle \phi_m^\infty(x), \end{aligned} \tag{2.19}$$

For computational tractability we truncate the sum in (2.19) to the first $L$ terms so $f$ can be reconstructed by

$$f(x) = \sum_{\ell=1}^{L} \sum_m \langle f, \psi_m^\ell \rangle \psi_m^\ell(x) + \sum_m \langle f, \phi_m^L \rangle \, \phi_m^L(x). \tag{2.20}$$

The representation of $f$ in the form (2.20) is called **wavelet transform**. Note that the sum (2.20) consists of $L$ different levels of "detail", expressed using the wavelet bases, and a rough part, expressed using the scaling function. Algorithmically, the coefficients $a_m^\ell = \langle f, \psi_m^\ell \rangle$ and $d_m = \langle \phi_m^L, f \rangle$ can be efficiently calculated using schemes which ensure that the $\mathcal{S}_\ell : V_\ell \to V_{\ell+1}$ scaling transforms and the $\mathcal{D}_\ell : V_\ell \to W_{\ell+1}$ detail transforms are sparse, resulting in increasingly sparse wavelet basis at increasing resolution of the MRA.

Two of the main applications of wavelets in signal processing are signal **compression** and signal **denoising**. Signal compression is effectively an **approximation** of a signal $f$ with as little data as possible. Using a wavelet transform with $L$ levels of multiresolution, compression can be performed by retaining the scaling function part (i.e., zeroing out the first sum in (2.20)) and computing

$$f(x) = \sum_m \langle f, \phi_m^L \rangle \, \phi_m^L(x). \tag{2.21}$$

Note that in the Fourier transform case the function approximation (2.5) is linear (as the basis elements do not depend on the function). Similarly, the wavelet approximation of the form (2.21) is linear, however unlike the Fourier basis, the wavelet basis is adaptive to the

function $f$. Thus, by applying wavelet coefficient thresholding or wavelet shrinkage (see (Mallat, 2008) for details) wavelets can be used for nonlinear function approximation. On the other hand, signal denoising deals with removing noise, which is typically concentrated at the finer scales (at higher wavelet frequencies). The easiest approach is to discard the detail subspaces after a certain level, however, this also removes the features of the signal encoded in those finer scales, so in practice various thresholding techniques are applied instead (see (Mallat, 2008)).

## 2.5 Spectral Graph Theory Concepts

Recall from Section 2.3 that one of the ways to define smoothness of a function is by applying the Laplace operator to it and measuring the ratio (2.11). However, the Laplace operator defined in (2.10) applies only to continuous functions. The discrete Laplace operator is the discrete analog of the continuous Laplace operator, defined for discrete spaces such as graphs. The graph Laplacian, which we mentioned earlier, and the normalized graph Laplacian, are the two most widely used ways to define the discrete Laplace operator.

Let $G(V, E)$ denote an unweighted, undirected graph containing no graph loops or multiple edges, where $V$ and $E$ denote the vertex and the edge set, respectively. The discrete Laplace operator can be defined for graphs in the form of the **graph Laplacian** (also sometimes called combinatorial graph Laplacian), which is $\mathcal{L} \coloneqq D - \mathcal{A}$ for unweighted graphs, and $\mathcal{L} \coloneqq D - W$ for weighted graphs. Here $D$ denotes the degree matrix of $G$, which is zero everywhere except on its main diagonal. For unweighted graphs, $D_{i,i}$ equals the number of edges incident on vertex $i$ (denoted by $\texttt{deg}\ (i)$), while for unweighted graphs $D_{i,i}$ is the sum of the weights of all the edges incident to $i$ for weighted graphs. $\mathcal{A}$ is the adjacency matrix, which is a binary matrix in which $\mathcal{A}_{i,j} = 1$ if vertices $i$ and $j$ are connected by an edge. $W$ is the weight matrix with $W_{i,j}$ representing the weight of edge $i, j$. Thus, for an undirected, unweighted graph, the graph Laplacian has entries

$$\mathcal{L}_{i,j} \coloneqq \begin{cases} -1 & \text{if } i,j \text{ are adjacent} \\ \mathtt{deg}(i) & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases} \tag{2.22}$$

The **normalized graph Laplacian** (Chung, 1997), which is a normalized version of $\mathcal{L}$ (2.22), is given by $\tilde{\mathcal{L}} \coloneqq D^{-1/2}\mathcal{L}D^{-1/2}$. In other words, $\tilde{\mathcal{L}}$ is derived from $\mathcal{L}$ by normalizing each edge by the square root of the degrees of its end vertices. $\tilde{\mathcal{L}}$ is given entry-wise is given by

$$\tilde{\mathcal{L}}_{i,j} \coloneqq \begin{cases} 1 & \text{if } i = j \text{ and } D_{i,i} \neq 0 \\ -1/\sqrt{\mathtt{deg}(i)\ \mathtt{deg}(j)} & \text{if } i \neq j \text{ and } i,j \text{ are adjacent} \\ 0 & \text{otherwise.} \end{cases} \tag{2.23}$$

Note that both the graph Laplacian $\mathcal{L}$ and the normalized graph Laplacian have a full set of orthonormal eigenvectors since they are real symmetric matrices. The eigenvalues of both the graph Laplacian and the normalized graph Laplacian are bounded from below by 0. In the case of the graph Laplacian $\mathcal{L}$ the eigenvector associated with the zero eigenvalue is constant. In addition, the eigenvalues of the normalized Laplacian $\tilde{\mathcal{L}}$ are contained in the interval $[0, 2]$.

When analyzing functions on graphs, which can be represented simply by a vector $f \in \mathbb{R}^{|V|}$, it is important to remember that measuring function smoothness is always with respect to the intrinsic structure of the graph, which is given by $G$. Often the graph Laplacians arise from a discrete sampling of a smooth manifold and so, if the density of the sampling is high enough, under certain assumptions about the sampling probability distribution the discrete Laplacian converges to its continuous counterpart at a certain rate (additional detail can be found in (Belkin and Niyogi, 2008; Hein et al., 2005) and the reference cited therein). The edge derivative of a function $f = \mathbb{R}^n$ at vertex $i$, with respect to some edge $e$, is $\frac{\partial f(i)}{\partial e}$. So the

graph gradient of $f$ at vertex $i$, with incident edges $\mathcal{N}_i = \{e_1, e_2, \ldots, e_p\}$, is just the vector

$$\gamma_i = \left[ \frac{\partial f(i)}{\partial e_1}, \frac{\partial f(i)}{\partial e_2}, \ldots, \frac{\partial f(i)}{\partial e_p} \right], \tag{2.24}$$

and so the local variation of $f$ at vertex $i$ can be measured by the $\ell^2$ norm of (2.24). Accordingly, the global variation of $f$ with respect to all the vertices of $G$ is then given by

$$\frac{1}{2} \sum \| \gamma_i \|_2^2 = \frac{1}{2} \sum_{i \in V} \sum_{j \in \mathcal{N}_i} W_{i,j} (f(i) - f(j))^2. \tag{2.25}$$

Generalization of (2.25) to other $\ell^p$ norms is provided in (Shuman et al., 2013a).

Expressing equation (2.25) in vector form leads to the so-called **graph Laplacian quadratic form** (Spielman, 2009)

$$f^\top \mathcal{L} f = \sum_{(i,j) \in E} W_{i,j} (f(i) - f(j))^2 \tag{2.26}$$

$$= \frac{1}{2} \sum_{i \in V} \sum_{j \in \mathcal{N}_i} W_{i,j} (f(i) - f(j))^2. \tag{2.27}$$

Intuitively, the quadratic form can be used for measuring the graph smoothness of $f$, since it will be small if the function does not "jump" too much at the two endpoints of any edge of $G$. For example, if $f$ is constant on all vertices, the quadratic form will be 0. In general, the quadratic form is small if for any pair of vertices $i$ and $j$ connected by an edge with high weight, the values of $f$ at those vertices have similar weights (i.e., $f_i$ and $f_j$ are both positive or both negative). In fact, when a situation like this arises, we say that the graph signal $f$ is *smooth*. Analogously to the inner product $\langle f, \Delta f \rangle$ in (2.11), the quadratic form (2.26) can be expressed as $f^\top \mathcal{L} f = \langle f, \mathcal{L} f \rangle$ [1].

The graph Laplacian is one of the central subjects of spectral graph theory, which is the study of the principal properties and structure of graphs from the eigenspectrum of matrices

1. In general, the inner product between two vectors $f$ and $g$ is defined as $\langle f, g \rangle = \int f(x)\, g(x)\, dx$, but if $f$ and $g$ are finite vectors, the integral is discretized by their dot product $\langle f, g \rangle = f^\top g$

associated with them (Chung, 1997). Expanding a function defined on a graph using the eigendecomposition of a matrix derived from it is precisely the graph equivalent of expanding a function on the real line using its Fourier transform — in this sense spectral graph theory amounts to performing Fourier analysis on graphs. In the following section we discuss the graph analog of the Fourier transform on the real line. Further details on spectral graph theory can be found in the seminal works by Chung (1997) and Spielman (2009), while Shuman et al. (2013a,b) provide an overview of spectral graph theory as a form of Fourier analysis on graphs.

## 2.6 Spectral Fourier Transform

Since the graph Laplacian $\mathcal{L}$, as defined in (2.22), is a real positive definite symmetric matrix, it has $n$ orthonormal eigenvectors $\{q_\ell\}_{0\leq\ell\leq n-1}$ with their corresponding eigenvalues $\{\lambda_\ell\}_{0\leq\ell\leq n-1}$ being real and nonnegative. We can assume that the eigenvalues are ordered $0 = \lambda_0 < \lambda_1 \leq \cdots \leq \lambda_{n-1} \coloneqq \lambda_{max}$, where $\lambda_0 = 0$ is an eigenvalue with multiplicity equal to the number of connected components in the graph.

Recall from Section 2.1 that on the real line the discrete Fourier transform is

$$\hat{f}(k) \coloneqq \sum_{j=0}^{n-1} e^{-2\pi ijk/n} f(j), \qquad 0 \leq k \leq n-1,$$

where $i \coloneqq \sqrt{-1}$ is the imaginary unit and both $\hat{f}$ and $f$ are represented as vectors in $\mathbb{C}^n$. $\hat{f}$ is the expansion of a function $f$ in terms of the complex exponentials $e^{2\pi ijk}$, which are the eigenfunctions of the one–dimensional Laplace operator $\frac{d^2}{dx^2}$ (i.e., the Laplace operator defined in (2.10) in the specific case when $d = 1$). The spectrum of the Laplace operator consists of all eigenvalues $\lambda$ for which there is a corresponding eigenfunction $q$ such that $-\Delta q = \lambda q$. In this specific case $q = e^{2\pi ijk}$ and so

$$-\frac{\partial^2}{\partial k^2} e^{2\pi ijk} = (2\pi k)^2 e^{2\pi ijk}. \tag{2.28}$$

In analogy to the real line case, the **graph Fourier transform** $\hat{f}$ of a function $f \in \mathbb{R}^n$ defined on the vertices of the graph is an expansion of $f$ in terms of the eigenvectors of $\mathcal{L}$ and is given by

$$\hat{f}(\lambda_\ell) \coloneqq \langle f, q_\ell \rangle = \sum_{i=0}^{n-1} q_\ell^*(i)\, f(i). \tag{2.29}$$

The **inverse graph Fourier transform** is then given by

$$f(i) = \sum_{\ell=0}^{n-1} \hat{f}(\lambda_\ell)\, q_\ell(i). \tag{2.30}$$

A more detailed analysis and illustrations of the graph Fourier transform and its inverse are provided by Shuman et al. (2013a).

In classical Fourier analysis the eigenvalues $\{(2\pi k)^2\}_{k\in\mathbb{R}}$ in (2.28) have a specific meaning of frequency in the sense that for $k$ close to zero (low frequencies) the associated complex exponential eigenfunctions are slowly oscillating (smooth) functions and they carry information about global behavior. On the other hand, when $k$ is far from zero (high frequencies), the associated complex exponential eigenfunctions oscillate much faster and reflect local variations of the function. Analogously, in the graph Fourier transform the graph Laplacian eigenvalues and eigenfunctions provide a similar notion of frequency. We will refer to the space of eigenfunctions of the discrete graph Laplacian $\mathcal{L}$ as the **graph spectral domain**. For connected graphs the Laplacian eigenvector $q_0$, with its corresponding eigenvalue $\lambda_0 = 0$, is constant with entries equal to $1/\sqrt{n}$ (for fully disconnected graphs the eigenvalues are all zero). The graph Laplacian eigenvectors $q_\ell$ associated with the low frequencies $\lambda_\ell$ are smooth in the sense that if two vertices in the graph are connected by an edge with large weight, the entries of the eigenvector corresponding to those vertices are likely to have similar values. The eigenvalues associated with the large eigenvalues, on the other hand, oscillate much faster (high frequencies) and the entries corresponding to vertices in the graph that are connected by an edge with large weight are likely to be dissimilar.

Note that effectively (2.29) and (2.30) can be used to represent a signal $f \in \mathbb{R}^n$ in two

different domains — the vertex domain (i.e., $f$ is defined on the vertices of the graph with graph Laplacian $\mathcal{L}$) or the spectral domain (i.e., $f$ can be constructed in the form (2.30) using the eigenvalues and eigenvectors of $\mathcal{L}$). For a wide range of signals on the real line wavelet transforms are preferable to Fourier transforms since their different levels of resolution localize signals in both time and frequency. Similarly, a graph wavelet transform should be able to localize a signal $f$ defined on its vertices *both* in the vertex and graph spectral domains. In the following chapter we review several examples of wavelet designs for graphs, some of which are defined in the spectral domain, while other are defined in the vertex domain. In the classical wavelets setting there is a tradeoff between time and frequency resolution (generally knowns as the Heisenberg uncertainty principle, originating from physics (Gabor, 1946)). It is still an open question whether such a tradeoff exists in the graph wavelet scenario. For generalization of uncertainty principles to the graph setting see the recent work by Perraudin et al. (2016); Tsitsvero et al. (2016); Pasdeloup et al. (2015).

## 2.7 Multiresolution on Discrete, Unstructured Spaces

Both a signal on a graph with $n$ vertices and a discrete–time signal with $n$ samples can be viewed as vectors in $\mathbb{R}^n$. In other words, in this case $X$ is a space of finite cardinality $n$, and so $V_0 = L_2(X) = \mathbb{R}^n$ (recall the nested sequence of subspaces in (2.14)). While in the classical case of multiresolution on the real line, operations such as translation and dilation are *both* intuitive and mathematically well defined notions, they cannot necessarily be easily applied to irregular domains such as graphs. For example, it is not immediately obvious what "translating" and "dilating" a graph signal means. A naive way to translate a function on the vertices of the graph might be to number the vertices, however this numbering/ordering is arbitrary and it is not shift–invariant. At the same time the overarching idea of a sequence of sparse unitary transforms $U_\ell : V_\ell \to V_{\ell+1} \oplus W_{\ell+1}$ that successively split spaces of function on $X$ into smoother and rougher parts carries through from the real line to the discrete setting. In order to derive the equivalent of Mallat's axioms for the discrete, unstructured

case, we let the symmetric matrix $A \in \mathbb{R}^{n \times n}$ play the role of the smoothing operator discussed in Section 2.3. To construct a multiresolution on $X$, then one needs to construct a nested sequences of subspaces

$$V_L \subset \ldots \subset V_2 \subset V_1 \subset V_0 = L(X) = \mathbb{R}^n, \tag{2.31}$$

with dimensions $1 \leq \delta_L \leq \cdots \leq \delta_1 \leq \delta_0 = n$, where each $V_\ell$ has an orthonormal basis $\Phi_\ell := \{\phi_m^\ell\}_{m \in \mathbb{Z}}$ and each $W_\ell$ has orthonormal basis $\Psi_\ell := \{\psi_m^\ell\}_{m \in \mathbb{Z}}$ with $V_\ell = W_{\ell+1} \bigoplus V_{\ell+1}$ (recall that $\dim(W_\ell) = \delta_\ell - \delta_{\ell+1}$) and $\Phi_0 = \{\phi_m^0 = e_m\}_{m=1}^n$ (i.e., $\Phi_0$ the standard basis of $\mathbb{R}^n$), satisfying three conditions:

**MRA1**: The sequence (2.31) is a filtration of $\mathbb{R}^n$ in terms of smoothness with respect to $A$ in the sense that, if we let

$$\eta_\ell = \sup_{v \in V_\ell} \langle v, Av \rangle / \| v \|^2, \tag{2.32}$$

the sequence $\eta_0 \geq \eta_1 \geq \cdots \geq \eta_L$ decreases sufficiently fast. Note that this is identical to the definition of smoothness in (2.11).

**MRA2**: The wavelets are localized in the sense that if we let

$$\mu_\ell = \max_{m \in \{1,\ldots,\delta_\ell\}} \| \psi_m^\ell \|_0, \tag{2.33}$$

(where $\| \cdot \|$ denotes the vector $\ell^0$ norm, i.e., the number of nonzero elements), the sequence $\mu_0 \geq \mu_1 \geq \cdots \geq \mu_L$ increases no faster than a certain rate.

**MRA3**: Letting $U_\ell$ be the matrix form of the orthogonal transform taking the basis

$\Phi_{\ell-1}$ to the basis $\Phi_\ell \cup \Psi_\ell$, i.e.,

$$\begin{aligned}
\phi_m^\ell &= \sum_{i=1}^{\delta_{\ell-1}} [U_\ell]_{m,i}\, \phi_i^{\ell-1}, \qquad m = \{1, \ldots, \delta_\ell\}, \\
\psi_m^\ell &= \sum_{i=1}^{\delta_{\ell-1}} [U_\ell]_{m+\delta_{\ell-1},i}\, \phi_i^{\ell-1}, \qquad m = \{1, \ldots, \dim(W_\ell)\},
\end{aligned} \tag{2.34}$$

each $U_\ell$ is sparse, guaranteeing the existence of a fast wavelet transform.

To what extent it is possible to simultaneously satisfy these three conditions, of course, depends on the operator $A$. In particular, while MRA3 automatically promotes some degree of sparsity as required by MRA2, the first two conditions are in conflict with each other for Heisenberg uncertainty relationship type reasons (Nahmod, 1994). In this sense, multiresolution analysis is always a balance between opposing forces.

# CHAPTER 3

# MULTIRESOLUTION DESIGNS FOR DISCRETE SPACES

While in the last couple of decades harmonic analysis has been widely used in signal processing and in regression analysis in statistics (Hardle et al., 1998; Vidakovic, 1999), it has only recently been applied to problems pertaining to high–dimensional Euclidean data or non-Euclidean data such as graphs. Just like classical wavelet transforms, which are designed to localize signals in both time (space) and frequency, wavelet transforms on graphs are designed with a similar goal in mind. There are two main types of ways of defining wavelets on graphs (see (Shuman et al., 2013a) and the references cited therein for a detailed review):

- **spectral domain designs** utilize precisely the spectral properties (i.e., information encoded in the eigendecomposition) of matrices defined on graphs (such as the (normalized) graph Laplacian or the adjacency matrix). Generally, graph spectral designs construct bases that are localized in *both* the vertex and the graph spectral domains Two notable example of spectral domain designs are Diffusion Wavelets (Coifman and Maggioni, 2006) (Section 3.1) and Spectral Graph Wavelets (Hammond et al., 2011) (Section 3.2). Diffusion wavelets are based on the compressed representations of the powers of a diffusion operator. The basis functions at each resolution level are obtained through a Gram–Schmidt orthogonalization scheme. Spectral graph wavelets are dilations and translations of a kernel function defined in the spectral domain of the graph Laplacian.

- **vertex domain designs** are based on the spatial features of the graph itself, such as the distance between the graph nodes or the connectivity of the graph. The idea here is to localize the vertex domain transform by filtering some $k$–neighborhood of vertices around a given vertex based on some distance metric *without* relying on the spectral properties of the graph. For example, the "wavelets on trees" design by Gavish

et al. (2010), which is reviewed in Section 3.3, is a notable example of this approach. Wavelets on trees are constructed by first fitting a balanced hierarchical tree (similar to the type of three obtained by agglomerative clustering) on a set of data points (using some pairwise distance metric) and then defining orthonormal bases for the space of functions over the hierarchically nested subtrees of the tree.

In this chapter we closely follow the material presented in (Coifman and Maggioni, 2006; Mahadevan and Maggioni, 2006; Hammond et al., 2011; Gavish et al., 2010; Lee et al., 2008), changing the notation for consistency, if necessary.

## 3.1 Diffusion Wavelets

Recall from our discussion at the end of Section 2.3 that one of the ways of defining smoothness is with respect to the diffusion operator $A = e^{-\alpha\Delta}$. The Diffusion Wavelets (DWs) method (Coifman and Maggioni, 2006, 2004; Maggioni, 2005) uses the diffusion operator $A = e^{-\alpha\mathcal{L}}$ (the Laplace operator $\Delta$ is now replaced by its graph analog, the graph Laplacian $\mathcal{L}$ of a finite weighted graph $X$ on $n$ vertices) to induce a multiresolution analysis. DWs interpret the powers of $A$ as smoothing operators acting on functions and construct a downsampling scheme to efficiently represent the multiscale structure. The goal is to efficiently compute the powers of $A^{2^j}$ (where $0 \leq j \leq J$ for some level/scale $J$), which describe the long term behavior of the diffusion process (or the random walk) on $X$. This yields a construction of scaling functions and wavelets at different levels. Below we follow the notation and intuition described by Coifman and Maggioni (2006).

While classical wavelet analysis constructs multiresolution on $X$ using the span of translations of a scaling function at a certain scale, DWs rely on the use of the span of all eigenfunctions of $A$ at a certain scale. Let $\Lambda_A = \{\lambda_i\}_{0 \leq i < n}$ denote the eigenvalues of $A$, ordered in decreasing order, and $Q_A = \{q_i\}_{0 \leq i < n}$ their corresponding eigenvectors. Recall that for any power $t$, the eigenvalues of $A^t$ are given by $\{\lambda_i^t\}_{0 \leq i < n}$. Using this observation,

we can threshold the spectrum $\Lambda_A$ and the eigenvalues $Q_A$ by defining, respectively,

$$\lambda_{A,j} = \{\lambda_i \in \Lambda_A : |\lambda_i^{2^{j+1}-1}| \geq \epsilon\} \quad \text{and} \quad q_{A,j} = \{q_i \in Q_A : \lambda_i \in \lambda_{A,j}\}$$

for a threshold parameter $\epsilon$. Letting $V_j := q_{A,j}$ and $V_0 = L_2(X)$, the sequence of subspaces $V_0, V_1, \ldots, V_J$ is reminiscent of the way classical MRA (2.14) decomposes $L_2(X)$ — see Section 4.3 in (Coifman and Maggioni, 2006) for details. Note that DWs do not satisfy the MRA axioms described in Section 2.4, in particular, the crucial dilation property A4 is not satisfied. Even though each subspace $V_j$ has an orthonormal basis consisting of eigenfunctions of $A$, these basis functions are generally highly nonlocalized. To get localized bases for each of the $V_j$ subspaces, DWs use an explicitly constructed downsampling scheme, starting with the standard basis for the finest subspace $V_0$ and constructing an orthonormal basis for each subsequent coarser subspace $\{V_j\}_{0<j\leq J}$.

Starting by applying the diffusion operator $A$ once to a space of test functions at the finest scale $j = 0$, the scheme is based on the repeated application of the following procedure — orthonormalize the columns of $A^{2^0}$ to get a new, downsampled basis for its column space (up to some precision), represent $A$ in this compressed basis and then compute the next dyadic power $A^{2^1}$ on this compressed basis; on the next scale $j = 1$ repeat this procedure, and so on for subsequent scales. In general, at scale $j$ we obtain a compressed representation of $A^{2^j}$ acting on a family of scaling functions spanning the column space of $A^{1+2+2^2+\cdots+2^{j-1}}$, for which we have a *compressed* orthonormal basis, and then apply $A^{2^j}$, locally orthonormalize and compress the result to get the next coarser subspace.

In matrix form DWs can be interpreted as a multiscale orthogonalization procedure in which the dyadic powers $A^{2^j}$ are interpreted as smoothing operators acting on functions in $L_2(X)$, while downsampling and translation of the wavelet basis are achieved by rank revealing QR. Starting with the basis $\Phi_0 := I_n$, which is just the identity matrix with the basis vectors $\{\delta_x\}_{0\leq x<n}$ on its columns, DWs construct an orthonormal basis for the column

space of $A^{2^0} = A$ by a Gram–Schmidt orthonormalization scheme. In matrix form this amounts to finding the reduced rank revealing QR factorization of $A$,

$$A \approx Q_0 R_0, \tag{3.1}$$

where $Q \in \mathbb{R}^{n \times p}$ $(p < n)$ has orthogonal columns and $R \in \mathbb{R}^{p \times n}$ is upper triangular. Note that in practice rank revealing QR is used to perform (3.1). Additional details on the various types of QR decompositions can be found in (Trefethen and Bau III, 1997). The columns of $Q_0$ form an orthonormal basis $\Phi_1$ of scaling functions for the column space of $A$, written as a linear combination of the initial basis $\Phi_0$. The basis functions of $A^{2^{j+1}}$ are computed using only the lowest frequency basis functions of $A^{2^j}$ since the high frequency ones are discarded in the rank revealing QR procedure and so (3.1) can be interpreted as downsampling of $V_1$.

The first dyadic power of $A$ can then be expressed on the basis $\Phi_1$ as follows (using the fact that the matrix $A$ is symmetric)

$$A^2 = AA^* \approx Q_0 \underbrace{R_0 R_0^*}_{\approx Q_1 R_1} Q_0^*,$$

where $A^2 \in \mathbb{R}^{p \times p}$ is now a compressed matrix expressed in terms of the column space of the first power of the diffusion operator $A$. By downsampling of $R_0 R_0^*$ by reduced rank revealing QR orthogonalization the second dyadic power of $A$ can be further expressed on the bases $\Phi_1$ and $\Phi_2$ as

$$A^4 = A^2 A^{2^*} \approx Q_0 Q_1 R_1 R_1^* Q_1^* Q_0^*.$$

Recursively this procedure continues up to the highest level $J$, where the $2^J$–th dyadic power of $A$ is given by the factorization

$$A^{2^J} \approx Q_0 Q_1 \dots Q_{J-1} R_{J-1}^* R_{J-1} Q_{J-1}^* \dots Q_1^* Q_0^*.$$

Wavelet bases for the spaces $W_j$ can be built analogously using the rank revealing QR by factorizing $I_{V_j} - Q_{j+1}Q_{j+1}^*$, i.e., the orthogonal projection of the complement of $V_{j+1}$ into $V_j$.

The main assumptions of diffusion wavelets are: (i) the underlying graph $X$ is strongly connected and local, i.e., each vertex is connected to only a small number of vertices; (ii) the operator $A$ is local in the sense that when applied to the Dirac delta function $\delta_x$ (where $x \in X$), the resulting $A\delta_x$ has numerically small support; and (iii) the powers of the diffusion operator $A$ have low numerical rank, i.e., $\text{rank}(A^{2^j}) < \text{rank}(A^{2^{j-1}})$. DWs assume that the powers of $A$ are of *low rank* and as a result they can be efficiently represented on a small set of basis vectors obtained by rank revealing QR at each level. However, for each higher level $j$ the operator $A^{2^j}$ is no longer local in the original space $X$, but in the new, compressed space obtained from performing $QR$ decomposition on $R_{j-1}R_{j-1}^*$. In many practical applications, the main drawback of DWs is that in practice at each level the QR factorization destroys the wavelet sparsity. The high frequency wavelets end up having a very wide support, which is at odds with the very goal of constructing well localized wavelets. In the worst case, when the diffusion operator is not of low rank, the running time of DWs is $O(n^3)$, however, by exploiting the sparsity and low rank of the dyadic powers of the diffusion operator, it can be reduced to $O(n^2 \log^2 n)$.

## 3.2 Spectral Graph Wavelets

While DWs represents an orthogonal wavelet transform, constructed by an explicit orthogonalization procedure, Hammond et al. (2011) propose a continuous graph wavelet transform — recall from Section 2.4 to in classical MRA continuous wavelets transforms have a continuous scale parameter $\ell$. They construct a wavelet transform of a function $f \in \mathbb{R}^n$ on the $n$ vertices of an arbitrary weighted graph $G(V, E)$ by defining scaling functions and wavelets using the graph Fourier transform (2.29). Their Spectral Graph Wavelet Transform (SGWT) consists of one scaling function centered at each vertex and $L$ wavelets centered at

each vertex (each one of them at a different scale).

Using the notation introduced in Section 2.6, we let $\mathcal{L}$ be the graph Laplacian of $G$ whose eigenvalues are $\{\lambda_\ell\}_{0\leq\ell\leq n-1}$ (w.l.o.g $0 = \lambda_0 < \lambda_1 \leq \cdots \leq \lambda_{n-1} := \lambda_{max}$) and their corresponding eigenvectors $\{q_\ell\}_{0\leq\ell\leq n-1}$. Letting $T_g = g(L) : L_2(V) \mapsto L_2(V)$, be the so-called Fourier multiplier, which acts on a given function $f$ by modulating each Fourier mode as

$$\widehat{T_g f}(\ell) = g(\lambda_\ell)\hat{f}(\lambda_\ell),$$

where $\hat{f}(\lambda_\ell)$ is defined in (2.29). The continuous function $g : \mathbb{R}_+ \mapsto \mathbb{R}_+$ (also called kernel as it is defined in the spectral domain) acts as a band–pass filter (i.e., $g(0) = 0$ and $\lim_{x\to\infty} g(x) = 0$) and satisfies certain admissibility conditions which guarantee that it is localized in the spectral domain (see (Hammond et al., 2011) for details). Applying the inverse Fourier transform yields

$$T_g\ f(x) = \sum_{\ell=0}^{n-1} g(\lambda_\ell)\hat{f}(\lambda_\ell)q_\ell(x).$$

The spectral graph wavelet operator at scale $t$ is then defined by $T_g^t = g(t\mathcal{L})$. So a spectral wavelet at scale $t \in \mathbb{R}_+$ which is localized at vertex $v_m \in V$ is given by

$$\psi_{t,m}(x) = T_g^t\ \delta_m(x) = \sum_{\ell=0}^{n-1} g(t\lambda_\ell)\, q_\ell^*(m)\, q_\ell(x), \qquad 0 \leq m \leq n-1, \tag{3.2}$$

where $\delta_m$ is the Dirac delta function located at vertex $v_m$. In other words, for a given scale $t$ there are total of $n$ wavelets, each centered at each of the $n$ graph vertices. Dilation of the wavelets is performed by the use of the scaling parameter $t$, while translation of the wavelets at scale $t$ is then effectively achieved by "localizing" the wavelet operator in the vertex domain of $G$. The number of scales $L$ is a design parameter adapted to the upper bound $\lambda_{max}$ of the eigenspectrum of $\mathcal{L}$ and is thoroughly described in (Hammond et al., 2011). Note that despite the fact that the spatial domain for the graph is discrete, the scale $t$ is actually defined for any positive real number since the kernel $g$ is continuous.

The scaling function is given by

$$\phi_m(x) = \sum_{\ell=0}^{n-1} h(\lambda_\ell)\, q_\ell^*(m)\, q_\ell(x), \tag{3.3}$$

where $h(\lambda) : \mathbb{R}_{\geq 0} \to \mathbb{R}$ is a real valued kernel function which smoothes the low frequencies (i.e., the eigenvalues of $\mathcal{L}$ close to 0). An example of such a kernel function is $h(x) = \gamma \exp(-(x/(0.6\lambda_{\min})^4)$, where $\gamma$ is set such that $h(0)$ has the maximum value of $g$ and $\lambda_{\min} = \lambda_{\max}/K$ for a parameter of choice $K$. Note that the SGWT scaling function $\phi$ (despite its name) differs from the scaling function introduced in Section 2.4, in the sense that $\phi$ is not used to generate the SGWT wavelets themselves in contrast to the "father"–"mother" wavelet relationship in classical MRA. Constructed analogously to, but independently of, the SGWT wavelets (3.2), the goal of the scaling functions (3.3) is simply to smoothly represent the low frequency content on the graph. In a sense, the SGWT wavelets (3.2) represent a band–pass filter (i.e., they focus on frequencies within a certain range), while the scaling functions (3.3) are a low–pass filter (i.e., they focus on frequencies below a certain threshold) geared towards capturing the low frequency content of $f$.

The wavelet coefficients $W_f(t, m)$ can be directly derived by applying the wavelet operator to $f$

$$\begin{aligned} W_f(t,m) = \langle \psi_{t,m}, f \rangle = (T_g^t f)(m) &= \sum_{\ell=0}^{n-1} g(t\lambda_\ell)\, q_\ell^*\, f\, q_\ell(m) \\ &= \sum_{\ell=0}^{n-1} g(t\lambda_\ell)\, \hat{f}(\lambda_\ell)\, q_\ell(m), \end{aligned} \tag{3.4}$$

where $\hat{f}(\lambda_\ell) = \langle f, q_\ell \rangle$ (2.29) is the graph Fourier transform of $f$. Intuitively, the wavelet coefficient $W_f(t, m)$ provides a measure of the degree to which the wavelet $\psi_{t,m}$ is present in the signal $f$. At scale $t$ the wavelet transform of $f$ is given by

$$W_f^t = [W_f(t,0), W_f(t,1), \ldots, W_f(t,n-1)]$$

and so the entire transform $W : \mathbb{R}^n \to \mathbb{R}^{n(L+1)}$ across all $L$ scales $t \in \{t_1, \ldots, t_L\}$ is

$$W = [W_h; W_f^{t_1}; W_f^{t_2}; \ldots, W_f^{t_L}], \tag{3.5}$$

where $W_h = \langle \phi_m, f \rangle$ are the SGWT scaling coefficients computed using (3.3).

The naive way of computing the transform, by directly using equation (3.4), requires explicit computation of the entire eigenvector decomposition of the graph Laplacian, which is unfeasible for large matrices since even general purpose routines such as QR decomposition have computational complexity of $O(n^3)$ in the worst case. In order to circumvent this computational problem, Hammond et al. (2011) approximate the scaled generating kernel $g$ by a low order polynomial and so the wavelet coefficients at each scale can be computed as a polynomial of $\mathcal{L}$ applied to the input signal $f$ through matrix–vector multiplication.

The SGWT is overcomplete in the sense that there are more wavelets than vertices in the graph — specifically, there are $n(L+1)$ coefficients in the entire SGWT transform (3.5). Ideally, at each scale one would want to subsample a collection of vertices which would serve as wavelets centers in (3.2) instead of calculating all $n$ wavelet coefficients at each scale. However, how one can perform this subsampling in a meaningful way remains an open question. On a regular graph the underlying geometric regularities can be used to find appropriate subsampling, however, the more interesting *and* much more difficult question, is how one can perform vertex subsampling on an irregular graph.

## 3.3 Multiscale Wavelets on Trees

Gavish et al. (2010) argue that current techniques in machine learning rarely exploit the rich geometric structure of high dimensional and graph data. Many methods used to process functions defined on undirected graphs are based on the graph Laplacian $\mathcal{L}$ (2.22), which is usually a product of applying a similarity kernel. The global function smoothness w.r.t. the graph is measured by the graph Laplacian quadratic form $f^\top \mathcal{L} f$ (2.26). However, measuring

function smoothness using the quadratic form leads to ill–posed problems for semi-supervised learning with high dimensional data as the number of unlabeled data grows to infinity (see (Nadler et al., 2009) for details). Another problem is that eigenvector–based methods might not be suitable for representing functions on graphs since basis vectors have global support and become increasingly oscillatory, which leads to a restricted number of coefficients that can be reliably calculated. Thus, it makes more sense to construct basis vectors which have local (rather than global) support and to that end Gavish et al. (2010) rely on the underlying geometry of the graph itself.

The proposed "wavelets on trees" framework assumes that the geometry of the input graph or high dimensional data can be captured by one (or many) hierarchical tree(s). The exact method of hierarchical tree construction is not relevant, as long as the input dataset $X$ is equipped with such a tree $\mathcal{T}$. The only requirement is that $\mathcal{T}$ is *balanced*. Let the tree have $L$ levels with $\ell = 0$ denoting the root level and $\ell = L - 1$ denoting the deepest level in the tree, the level at which each datapoint $x \in X$ is a leaf node.

Let $V = \{f \mid f : X \to \mathbb{R}\}$ denote the space of all functions on $X$ and let $X_k^\ell$ denote the $k$–th subtree (equivalently, the set of all leaves of the $k$-th subtree) of a tree rooted at some node which resides on level $\ell$. Since the tree constructed from the data is balanced, the branching factor at each node of $\mathcal{T}$ is bounded between some constants $\underline{b}$ and $\bar{b}$. The tree representation of the data induces a multiresolution analysis with an associated Haar–like wavelet basis. In particular, if we let $V_\ell$ denote the spaces of functions constant on *all subtrees* at a given level $\ell$, the approximation spaces can be defined as

$$V_\ell = \{f \mid f : X \to \mathbb{R},\ f \text{ is constant on all subtrees } X_j^\ell\}.$$

So by construction $V = V_{-L-1} \supset \ldots \supset V_{-2} \supset V_{-1} \supset V_0$ and $V_\ell = W_{\ell+1} \oplus V_{\ell+1}$. This relationship between the subspaces resembles the one we saw in classical MRA in Figure (2.2).

Consider a subtree $X_k^\ell$ at level $\ell$ with *two* subtrees $X_i^{\ell+1}$ and $X_j^{\ell+1}$, then there is a

zero–mean Haar–like function $\psi_1^{\ell,k}$ (see Section 4.4 for description of Haar wavelets) that is supported only on these two subtrees and is piecewise constant on each of them. In general, a subtree $X_k^\ell$ with branching factor $b$ gives rise to $b-1$ Haar–like orthonormal (wavelet) functions $\psi_1^{\ell,k}$, $\psi_2^{\ell,k}, \ldots, \psi_{b-1}^{\ell,k}$. More specifically, the first function $\psi_1^{\ell,k}$ is supported only on the first two subtrees of $X_k^\ell$, the second function $\psi_2^{\ell,k}$ is supported on the first three subtrees of $X_k^\ell$, and so on until the last wavelet $\psi_{b-1}^{\ell,k}$ is supported on all the $b$ subtrees of $X_k^\ell$. For each subtree $X_k^\ell$ of $\mathcal{T}$, there is one scaling function given by $\phi^{\ell,k} = \mathbb{1}_{X_k^\ell}$, i.e., $\phi^{\ell,k}$ is just the constant function supported on the leaves of $X_k^\ell$. So at level $\ell = 0$, the smoothest subspace $V_0$ is spanned by the scaling function $\phi^{0,1}$ which is just the constant function on the dataset $X$ (note that $X$ is equivalent to the tree $X_1^0 = \mathcal{T}$). In other words, $V_0 = \text{span}_{\mathbb{R}}\{\mathbb{1}_X\}$ is the one–dimensional space of constant functions on $X$.

The collection of all of these functions, together with the constant function on $X$, forms an orthonormal basis of $V$. Gavish et al. (2010) show that the wavelet coefficients for this Haar–like orthonormal basis decay fast as long as the function satisfies certain smoothness conditions w.r.t. the tree, which are analogous to Hölder smoothness in the Euclidean setting. While the "wavelets on trees" design exploits the geometric structure in the vertex domain, it suffers from the oversimplifying assumption that one can fit a balanced tree on the dataset in the first place. On one hand, defining a tree on a dataset relies upon finding an appropriate distance metric between the data points, and so, it is not immediately clear which metric would produce a tree which best captures the inherent structure of a given dataset. On the other, while one might be able to come up with a good metric given a dataset, it might not be feasible to expect that the resulting tree would be balanced. Finally, the algorithm has the overhead associated with preprocessing $X$ to construct a clustering tree.

# CHAPTER 4

# MULTILEVEL AND MULTISCALE DESIGNS FOR FACTORIZING MATRICES

Multilevel/multiscale factorizations usually involve "coarsening" of a matrix level by level until a small number of variables/coordinates remain. By coarsening we mean, roughly, clustering the coordinates using a carefully crafted objective function. The goal is to compress a matrix to a more manageable size (often in the form of a product of several smaller, more compact matrices), while preserving the *local* information in the vicinity of each variable/coordinate and keeping the global structure of the matrix *approximately* unchanged. These factorizations generally obey the wavelet principle of "what is local must stay local", which we mentioned in Section 2.4.

Classically, matrix factorizations are in the realm of numerical linear algebra, which deals with reducing matrices to some canonical form such as a diagonal or upper triangular matrix. One canonical matrix factorization is Jacobi's method for eigenvalue decomposition (Jacobi, 1846), which is possibly one of the earliest linear algebra algorithms. It has an interpretation as a multilevel method in the sense that it iteratively applies very local updates (in the form of Givens rotations) to the matrix being diagonalized (Section 4.1). Much more recently, Lee et al. (2008) showed that a modification of Jacobi's algorithm results in a matrix factorization, called the Treelet transform (Section 4.2), which has an interpretation as a MRA (2.14).

Another group of factorizations — such as the Fast Walsh–Hadamard transform and the earlier mentioned Haar transform (4.3 and Sections 4.4) — originate from harmonic analysis and for that reason are not typically presented as matrix factorizations. They are, nevertheless, expressible (and computable) in matrix form, multilevel in nature and closely relate to the central subject of this thesis, i.e., the Multiresolution Matrix Factorization introduced in Chapter 5.

In machine learning, on the other hand, the goal of multilevel/multiscale dictionary learning and sparse coding (Section 4.5) is to decompose a matrix into a dictionary of local, rather than global and dense eigenfunctions.

Finally, in numerical analysis, fast multipole methods are algorithms that can efficiently compute the interactions between a large number of particles by aggregating them at multiple scales (Greengard and Rokhlin, 1987; Beatson and Greengard, 1997). The multipole hierarchy is related to classes of matrices with specific structure such as hierarchical matrices (Section 4.6).

More generally, the idea of "coarsening" a graph or matrix is often applied in practice for solving numerous problems in scientific computing. For example, many large–scale graph problems are solved by multilevel/multiscale/multiresolution algorithms, in which at each level of coarsening one defines a set of "coarse" unknown variables together with the constraints they should satisfy or the objective function they should minimize. Each coarsened unknown is defined in terms of the next, finer–level unknowns, which are in turn defined in the unknowns on even finer scales — this procedure is iterated until the finest scale is reached. Then, after the coarse level problem is approximately solved, the fine level problem is interpolated from that solution (again, this is performed recursively up to the finest scale) (for example, see (Ron et al., 2011) and the references cited therein). A notable example in this category are multigrid methods (Brandt, 1973; Hackbusch, 2003; Livne and Brandt, 2012; Briggs et al., 2000), which solve systems of partial differential equations by applying such a coarsening multiresolution scheme.

## 4.1 Jacobi's Algorithm

Jacobi's eigenvalue algorithm iteratively calculates the eigendecomposition $A = U\Lambda U^{-1}$ of a symmetric matrix (Jacobi, 1846). The algorithm works by repeatedly applying a series of rotations $U_1$, $U_1$, ..., $U_L$ to a symmetric matrix $A \in \mathbb{R}^{n\times n}$ on the left and on the right side

until the product

$$\Lambda = U_L^\top \dots U_2^\top U_1^\top A U_1 U_2 \dots U_L, \tag{4.1}$$

converges to a diagonal matrix, up to numerical precision. The matrix $\Lambda$ is diagonal with its main diagonal containing the eigenvalues of $A$. The eigenvectors of $A$ can be obtained from the columns of the cumulative matrix

$$U = U_1 U_2 \dots U_L. \tag{4.2}$$

Each column $U_{:,i}$ contains an eigenvector corresponding to the eigenvalue $\Lambda_{i,i}$ (the eigenvalues are not in a specific order). The index $1 \le \ell \le L$ denotes each iteration of Jacobi's algorithm. We refer to $\ell$ as the "level" of the factorization.

Writing out the full $L$ level factorization of $A$ in terms of $U_1 U_2 \dots U_L$, we get

$$A = U_1 U_2 \dots U_L \Lambda U_L^\top \dots U_2^\top U_1^\top, \tag{4.3}$$

where the left hand side equals the right hand side up to numerical precision.

In numerical linear algebra the rotation matrices $U_1$, $U_1$, …, $U_L$ are known as **Givens rotations**. A Givens rotation is a plane rotation represented by a matrix of the form

$$U_{i,j,\theta} = \begin{pmatrix} 1 & \vdots & & \vdots & \\ \dots & c & \dots & -s & \dots \\ & \vdots & 1 & \vdots & \\ \dots & s & \dots & c & \dots \\ & \vdots & & \vdots & 1 \end{pmatrix} \qquad \begin{matrix} c = \cos\theta \\ s = \sin\theta, \end{matrix} \tag{4.4}$$

where the dots denote the fact that the matrix is zero everywhere except for two off-diagonal entries and the main diagonal, which contains either 1's or $c = \cos\theta$ for some angle $\theta \in (0, 2\pi)$, at the intersections of the $i$–th and $j$–th row/column.

Applying a Givens rotation at level $\ell = 1$ to both sides of $A$ results in the matrix

$$\mathcal{A}_1 = U_{i,j,\theta} A U_{i,j,\theta}^{\top} = \begin{pmatrix} & & \mathcal{A}_{1,i} & & \mathcal{A}_{1,j} & \\ & & \vdots & & \vdots & \\ \mathcal{A}_{i,1} & \dots & \mathcal{A}_{i,i} & \dots & \mathcal{A}_{i,j} & \dots \mathcal{A}_{i,n} \\ & & \vdots & & \vdots & \\ \mathcal{A}_{j,1} & \dots & \mathcal{A}_{j,i} & \dots & \mathcal{A}_{j,j} & \dots \mathcal{A}_{j,n} \\ & & \vdots & & \vdots & \\ & & \mathcal{A}_{n,i} & & \mathcal{A}_{n,j} & \end{pmatrix}, \tag{4.5}$$

where the dots denote that *only* the $i$–th and $j$–th rows/columns of $A$ are altered by the rotation. In other words, (4.5) is very *local* operation. Explicitly performing the multiplication in equation (4.5), we can compute the entries in the $i$–th and $j$–th rows/columns of $\mathcal{A}_1$ as follows (assuming $\mathcal{A}_0 = A$)

$$\begin{aligned}
[\mathcal{A}_1]_{i,j} &= sc[\mathcal{A}_0]_{i,i} - s^2[\mathcal{A}_0]_{i,j} + c^2[\mathcal{A}_0]_{i,j} - cs[\mathcal{A}_0]_{j,j} \\
[\mathcal{A}_1]_{i,i} &= c^2[\mathcal{A}_0]_{i,i} + s^2[\mathcal{A}_0]_{j,j} - 2sc[\mathcal{A}_0]_{i,j} \\
[\mathcal{A}_1]_{j,j} &= s^2[\mathcal{A}_0]_{i,i} + c^2[\mathcal{A}_0]_{j,j} + 2sc[\mathcal{A}_0]_{i,j} \\
[\mathcal{A}_1]_{k,i} &= [\mathcal{A}_1]_{i,k} = c[\mathcal{A}_0]_{k,i} - s[\mathcal{A}_0]_{k,j} && k \neq i,j,\ 1 \leq k \leq n \\
[\mathcal{A}_1]_{k,j} &= [\mathcal{A}_1]_{j,k} = s[\mathcal{A}_0]_{k,i} + c[\mathcal{A}_0]_{k,j} && k \neq i,j,\ 1 \leq k \leq n \\
[\mathcal{A}_1]_{k,p} &= [\mathcal{A}_0]_{k,p} && k,p \neq i,j,\ 1 \leq k \leq n.
\end{aligned} \tag{4.6}$$

Note that applying the rotation matrices on both sides of $\mathcal{A}_0$ preserves its symmetry so $[\mathcal{A}_1]$ is also symmetric.

The key idea of Jacobi's method is to greedily zero out the off-diagonal elements of $A$ by performing a series of Given rotations. Thus, at the first level the rotation angle $\theta$ of $U_{i,j,\theta}$ needs to be set such that the off-diagonal element $[\mathcal{A}_1]_{i,j}$ (given by the first equation in (4.6)) is zeroed out. Setting

$$[\mathcal{A}_1]_{i,j} = sc[\mathcal{A}_0]_{i,i} - s^2[\mathcal{A}_0]_{i,j} + c^2[\mathcal{A}_0]_{i,j} - cs[\mathcal{A}_0]_{j,j} = 0,$$

and solving for $\theta$ yields the closed form solution

$$\theta = \frac{1}{2}\arctan(2[\mathcal{A}_0]_{i,j}/([\mathcal{A}_0]_{j,j} - [\mathcal{A}_0]_{i,i})). \tag{4.7}$$

Replacing $\mathcal{A}_0$ with $\mathcal{A}_\ell$ and $\mathcal{A}_1$ with $\mathcal{A}_{\ell+1}$ in (4.6) and (4.7), we now have an iterative way of updating the $(i,j)$–th pair of rows/columns of $\mathcal{A}_{\ell+1} = U_{\ell+1}^\top \dots U_2^\top U_1^\top A U_1 U_2 \dots U_{\ell+1}$ in closed form by computing the rotation angle of $U_{\ell+1} = U_{i,j,\theta}$. Thus, at each level, in order to apply a rotation $U_{\ell+1}$ to $\mathcal{A}_\ell = U_\ell^\top \dots U_2^\top U_1^\top A U_1 U_2 \dots U_\ell$, only two rows and columns of $\mathcal{A}_\ell$ need to be updated. Note that $\mathcal{A}_L = \Lambda$, i.e., upon convergence of Jacobi's algorithm the main diagonal of $\mathcal{A}_L$ contains the eigenvalues of $A$.

Analogously to the iterative computation of $\Lambda$ by (4.6), the eigenvectors (4.2) can also be computed iteratively. At iteration $\ell + 1$ the cumulative matrix $\mathcal{U}_{\ell+1} = U_1 U_2 \dots U_{\ell+1}$ can be computed by performing the following closed form updates of $\mathcal{U}_\ell = U_1 U_2 \dots U_\ell$

$$\begin{aligned}
[\mathcal{U}_{\ell+1}]_{k,i} &= c[\mathcal{U}_\ell]_{k,i} - s[\mathcal{U}_{\ell+1}]_{k,j} && k \neq i,j,\ 1 \le k \le n,\\
[\mathcal{U}_{\ell+1}]_{k,j} &= s[\mathcal{U}_\ell]_{k,i} + c[\mathcal{U}_{\ell+1}]_{k,j} && k \neq i,j,\ 1 \le k \le n,\\
[\mathcal{U}_{\ell+1}]_{k,p} &= [\mathcal{U}_\ell]_{k,p} && k,p \neq i,j,
\end{aligned} \tag{4.8}$$

which means that to compute $[\mathcal{U}_{\ell+1}]$ only the $i$–th and $j$–th columns of $[\mathcal{U}_\ell]$ need to be updated. Finally, upon convergence, at level $L$, we have $\mathcal{U}_L = U$, i.e., the eigenvectors of $A$ can be read off from the columns of $\mathcal{U}_L$.

While Jacobi's algorithm is designed to zero out off-diagonal entries of $\mathcal{A}_\ell$ at each iteration, there are various pivoting strategies (which empirically affect the rate of convergence of Jacobi's algorithm) for zeroing out those elements (Hansen, 1963; Giménez et al., 1996). Typically, the maximum off-diagonal entry is zeroed out at each level $\ell$. It is important to note that even though each rotation might increase/decrease the entries in $\mathcal{A}_\ell$ which have been zeroed out by the previous rotations, the total off-diagonal mass decreases at each iteration (i.e., the sums of the squares of the off-diagonal entries of $\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_L$ form a

decreasing sequence).

Finally, the eigenvalues of $A$ are determined by finding the roots of the characteristic polynomial, however explicit algebraic formulas for finding the roots of a polynomial exists only if its degree is $\leq 4$. So,in general, $A$ cannot be diagonalized to infinite precision in a finite number of rotations. In practice, it has been empirically shown that the number of rotations $L$ required for the convergence of the algorithm is on the order of $3n^2$ to $5n^2$ (each of which can be performed in order $n$ operations) (Rutishauser, 1966). Thus, the total running time of Jacobi's method is $O(n^3)$. Details about implementing Jacobi's method can be found in (Press et al., 1996).

## 4.2 Treelets

Lee et al. (2008) propose a multilevel matrix factorization, called Treelets, which is a modification of Jacobi's algorithm. Therefore, below we use the notation we already introduced in the previous section. Similar to above, we will sometimes refer to the $\ell$–th iteration of the Treelet algorithm as the $\ell$–th level.

The Treelets algorithm constructs a multiscale basis of a symmetric matrix $A$ (more specifically, the covariance matrix $A \in \mathbb{R}^{n\times n}$ of a data set $X = \{x_1, \ldots, x_m\}$ of points in $\mathbb{R}^n$) in the form

$$A \approx U_1 U_2 \ldots U_L \Lambda U_L^\top \ldots U_2^\top U_1^\top,$$

where, just like in Jacobi's algorithm in (4.3), the matrices $U_1, \ldots, U_L$ are Givens rotations, but the matrix $\Lambda$ is no longer strictly diagonal (but still symmetric). Hence, the equality sign in (4.3) is now replaced with the $\approx$ sign.

In addition to each of the $U_\ell$ matrices being a Givens rotation, the indices $i$ and $j$ involved in each rotation $U_\ell = U_{i,j,\theta}$, as well as the order in which the rotations are applied, obey additional constrains. As a result of these constraints, the orthogonal basis vectors of $\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_L$ can be interpreted as wavelets supported on different subsets of variables. To

put it differently, the Treelets algorithm is able to find a hierarchical cluster tree of the variables and more importantly, is able to do so without fitting the data to some predetermined structure, such as, for example, the type of hierarchical tree used by Gavish et al. (2010) and discussed in Section 3.3. Instead, the construction of the Treelet transform is data driven in the sense that the basis vectors are derived from the local structure of the data itself. As discussed in Chapter 3, standard wavelets are not well suited for the analysis of unordered data, such as graphs. However, the Treelets transform circumvents this problem by finding a *data adaptive* multiscale representation of the data *regardless* of the ordering of the variables in $A$.

In addition to iteratively updating $\mathcal{A}_\ell$ and $\mathcal{U}_\ell$ (again using the above convention $\mathcal{A}_0 = A$), Treelets also iteratively update a set of indices $S$ — the way $S$ is updated guarantees that the Givens rotations $U_1, \ldots, U_L$ obey the constrains mentioned above. Starting with a covariance matrix $A \in \mathbb{R}^{n \times n}$ and the set $S = [n]$ (each index denotes a row/column of $A$) , the Treelets algorithm consists of repeating the following three steps at each iteration/level $\ell$:

1. The first step is to find the two variables $i, j \in S$ which are most similar to each other according to the similarity matrix $\mathcal{A}_\ell$. So, finding the $(i, j)$ pair corresponds to finding the maximum off–diagonal element of the submatrix $[\mathcal{A}_\ell]_{S,S}$.

   Note that this step is very similar to the way Jacobi's algorithm proceeds, with the added constraint that the maximum off-diagonal entry is found not among all $(i, j)$ pairs of rows/columns of $\mathcal{A}_\ell$, but only among the pairs of rows/columns still in $S$.

2. In this step the variables $i$ and $j$ are replaced by a coarse–grained sum variable and a residual difference variable. This is equivalent to applying a Givens rotation $U_\ell = U_{i,j,\theta}$ (4.4) to $\mathcal{A}_{\ell-1}$ in order to obtain $\mathcal{A}_\ell = U_\ell^\top \ldots U_2^\top U_1^\top A U_1 U_2 \ldots U_\ell$. Just like in Jacobi's algorithm, the $\theta$ angle of the rotation $U_\ell = U_{i,j,\theta}$ is set according to (4.7) and as a result $[\mathcal{A}_\ell]_{i,j}$ is zeroed out.

   Applying $U_\ell$ on both sides of $\mathcal{A}_{\ell-1}$ can be thought of a local PCA operation in the

sense that $U_\ell$ diagonalizes the $2 \times 2$ matrix

$$\begin{pmatrix} [\mathcal{A}_{\ell-1}]_{i,i} & [\mathcal{A}_{\ell-1}]_{i,j} \\ [\mathcal{A}_{\ell-1}]_{j,i} & [\mathcal{A}_{\ell-1}]_{j,j} \end{pmatrix}. \tag{4.9}$$

In order to apply $U_\ell$ on $\mathcal{A}_{\ell-1}$, all that is required is updating $\mathcal{A}_{\ell-1}$ according to the set of closed form equations (4.6). Now, one of the two diagonal entries $[\mathcal{A}_\ell]_{i,i}$ and $[\mathcal{A}_\ell]_{j,j}$ goes up, while the other goes down in value, relative to their respective values $[\mathcal{A}_{\ell-1}]_{i,i}$ and $[\mathcal{A}_{\ell-1}]_{i,i}$ before the rotation.

This step is identical to applying a Givens rotation in Jacobi's algorithm.

3. As soon as $U_\ell$ is applied on both sides of $\mathcal{A}_{\ell-1}$ to obtain $\mathcal{A}_\ell$, the index corresponding to the smaller of the two entries $[\mathcal{A}_\ell]_{i,i}$ and $[\mathcal{A}_\ell]_{j,j}$ is eliminated from $S$. In other words, if $[\mathcal{A}_{\ell-1}]_{i,i} \leq [\mathcal{A}_{\ell-1}]_{j,j}$, the set $S$ is updated to $S = S \backslash i$.

   This "elimination" step is not present in Jacobi's algorithm.

These three steps are repeated level by level for a total of $L = n$ levels, i.e., until the set $S$ becomes empty. By performing, at each level $\ell$, a rotation $U_\ell = U_{i,j,\theta}$, involving an $(i, j)$ pair of coordinates (step 2), followed by the elimination of either $i$ or $j$ from $\mathcal{S}$ (step 3) (which means that subsequent rotations cannot involve the $i$–th or the $i$–th coordinate), the Treelets algorithm constructs a multiscale orthonormal basis supported on a hierarchical *binary* tree. After $L = n$ iterations the columns of $\mathcal{U}_L = U_1 U_2 \dots U_L$ contain all the wavelets.

Treelets is motivated by the need to find higher–order dependences/hierarchies in covariances matrices arising in classification related tasks, such as feature selection. Therefore, in Treelets Jacobi's objective function of greedily minimizing the off-diagonal norm of $\mathcal{A}_\ell$ is replaced with the objective of constructing a basis supported on nested clusters of variables (i.e. rows/columns of the covariance matrix $A$).

## 4.3 The Fast Walsh–Hadamard Transform

The Fourier transform of a function $f$ on the $d$–dimensional unit cube $\{0,1\}^d$ (equivalently, on the group $\mathbb{Z}_2 \times \ldots \times \mathbb{Z}_2$), in some contexts also called the Hadamard or Walsh–Hadamard transform, is

$$\hat{f}(i_1, \ldots, i_d) = \frac{1}{2^{d/2}} \sum_{j_1,\ldots,j_d=0}^{1} (-1)^{i_1 j_1 + i_2 j_2 + \ldots + i_d j_d} f(j_1, \ldots, j_d). \tag{4.10}$$

Vectorizing $f$ and $\widehat{f}$, we can write $\widehat{f} = \mathcal{H} f$ with $\mathcal{H}$ being

$$\mathcal{H} = \underbrace{H \otimes H \otimes \ldots \otimes H}_{d \text{ times}} = H^{\otimes d}, \qquad \text{where } H = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}. \tag{4.11}$$

The Fast Walsh–Hadamard Transform (FWHT) is based on the observation that $\mathcal{H}$ factors as a product $W_d W_{d-1} \ldots W_1$, where

$$W_k = I_{2^{d-k}} \otimes H \otimes I_{2^{k-1}},$$

and, as usual, $I_m$ denotes the $m$ dimensional identity. Notice that each row of $W_k$ contains exactly two non-zero elements. Therefore, multiplying any vector by $W_k$ takes only $2 \cdot 2^d$ operations, and by applying $W_1, W_2, \ldots, W_d$ in series, the transform $f \mapsto \mathcal{H} f$ may be computed in $d 2^{d+1}$ time, in contrast to the $2^{2d}$ complexity of the naive approach.

As one would expect from a Fourier transform, $\mathcal{H}$ diagonalizes the corresponding Laplacian,

$$\Delta = \mathcal{H}^\top D \mathcal{H} = W_1^\top W_2^\top \ldots W_d^\top D W_d \ldots W_2 W_1, \tag{4.12}$$

where $\Delta$ is now the Laplacian of the hypercube, given (in multi-index notation), by

$$\Delta_{I,J} = \begin{cases} -1 & \text{if } I = (i_1, \ldots, i_d) \text{ and } J = (j_1, \ldots, j_d) \text{ differ in exactly one coordinate,} \\ d & \text{if } I = J, \\ 0 & \text{otherwise.} \end{cases}$$

## 4.4 The Fast Haar Wavelet Transform

As mentioned in Section 2.4, one of the first wavelet functions ever proposed is the Haar wavelet (Haar, 1909), which was introduced in the context of decomposing functions on the real line. As Figure 2.1(c) shows, the Haar mother wavelet and scaling function are respectively

$$\psi(x) = \begin{cases} 1 & 0 \le x < 0.5 \\ -1 & 0.5 \le x < 1 \\ 0 & \text{otherwise} \end{cases} \qquad \text{and} \qquad \phi(x) = \begin{cases} 1 & 0 \le x < 1 \\ 0 & \text{otherwise.} \end{cases}$$

Given a function $f : [0,1) \to \mathbb{R}$ discretized at resolution $2^{\ell_0}$ for some $\ell_0 < 0$, i.e., $f \in V_{\ell_0} \cap \mathbb{R}^{[0,1)}$, according to (2.20) the wavelet transform of $f$ is

$$f(x) = \sum_{\ell=\ell_0+1}^{0} \sum_{m=0}^{2^{-\ell}-1} \langle f, \psi_m^\ell \rangle \psi_m^\ell(x) + \langle f, \phi_0^0 \rangle \, \phi_0^0(x),$$

where $\psi_m^\ell = 2^{-\ell/2} \psi(2^{-\ell} x - m)$ and $\phi_m^\ell = 2^{-\ell/2} \phi(2^{-\ell} x - m)$.

The Haar scaling functions $\phi_m^\ell$ and wavelets $\psi_m^\ell$ are averaging and difference operators applied to the interval $[2^\ell m, 2^\ell (m+1))$ and so the wavelet transform can be easily computed. Now we let $f = (\langle f, \phi_0^{\ell_0} \rangle, \ldots, \langle f, \phi_{2^{-\ell_0}-1}^{\ell_0} \rangle)^\top$ be the vector representation of $f$ in $V_{\ell_0}$, and $\hat{f}$ be the vector of wavelet coefficients in the particular order that puts $\hat{f}_1 = \langle f, \phi_0^0 \rangle$ and $\hat{f}_{2^{\ell-\ell_0-1}(1+2m)+1} = \langle f, \psi_m^\ell \rangle$. It is easy to check that the wavelet transform $f \mapsto \hat{f} = \mathcal{W} f$

factorizes in the form

$$\mathcal{W} = U_0 \dots U_{\ell_0+2} U_{\ell_0+1}, \tag{4.13}$$

where $U_\ell$ is defined as

$$U_\ell = I \oplus_{T_\ell} (I_{2^{-\ell}} \otimes \hat{H})$$

with $\hat{H}$ defined in (4.11), a set $T_\ell = (1, k+1, 2k+1, \dots, 2^{-\ell}k+1)$, where $k = 2^{\ell-\ell_0-1}$, and $\otimes$ denoting the tensor product (1.3). The Haar wavelet transform diagonalizes matrices with nested (hierarchical) block-diagonal structure of the form

$$A = \bigoplus_{m=1}^{\ell_0} \gamma_m I_{2^{\ell_0-m}} \otimes \mathbf{1}_{2^m \times 2^m},$$

for any setting of the coefficients $\gamma_1, \dots, \gamma_{\ell_0}$, into

$$A = U_1^\top U_2^\top \dots U_L^\top D U_L \dots U_2 U_1, \tag{4.14}$$

with $D$ being a diagonal matrix.

## 4.5 Multilevel and Multiscale Dictionary Learning

Often the assumption in numerical linear algebra and certain machine learning methods is that a matrix is of *low rank*. If a matrix $A \in \mathbb{R}^{n \times n}$ is of rank $r \ll n$, it can be expressed as a sum of a dictionary of $r$ mutually orthogonal unit vectors $\{q_1, q_2, \dots, q_r\}$ in the form

$$A = \sum_{i=1}^{r} \lambda_i q_i q_i^\top, \tag{4.15}$$

where $q_1, q_2, \dots, q_r$ are the top normalized eigenvectors of $A$ with their corresponding eigenvalues $\lambda_1, \lambda_2, \dots, \lambda_r$. When $A$ is a covariance matrix, the decomposition of the form (4.15) is also known as Principal Component Analysis (PCA) (Jolliffe, 1986; Eckart and Young, 1936). The decomposition (4.15) is one of the ways to approximate (or compress) the original

matrix $A$, since selecting the top eigenvectors guarantees that the error $\|A - A_r\|$, measured in Frobenius, operator or trace norm, is minimized. The computational drawback of using PCA for matrix approximations is that eigenvectors are generally dense, i.e. supported on all $n$ dimension of $A$, while matrices arising in graph learning problems are sparse, since each graph vertex is generally connected to just a few other "nearby" vertices. In cases like this, conceptually it makes sense to decompose $A$ into a dictionary $\mathcal{D}$ of local, rather than global, dense eigenfunctions. Modeling $A$ in terms of a dictionary also avoids the added overhead of finding the first $r$ eigenvectors, which becomes prohibitive in cases where the underlying matrix is very large.

The study of sparse representation of signals (also known as sparse coding) using dictionaries attempts to achieve precisely this task. Traditionally used in image and signal processing, more recently dictionaries have also found applications in compression, regularization in inverse problems and feature extraction (Chen et al., 1998; Mairal et al., 2008; Aharon et al., 2006).

The goal of sparse coding is to model a signal $y \in \mathbb{R}^n$ using a dictionary $D \in \mathbb{R}^{n \times k}$ whose set of columns $\{d_i\}_{i=1}^k$ form a collection of $k$ waveforms (also known as atoms). The signal $x$ can be represented as a linear combination in the form $x = D\alpha$ (i.e., $\| x - D\alpha \|_p \leq \epsilon$ for some precision parameter $\epsilon$ and appropriately chosen $\ell^p$ norm), where $\alpha \in \mathbb{R}^k$ is a vector of representation coefficients. If the dictionary consists of more than $n$ atoms (i.e., $k > n$), it is called overcomplete and in this case, if $D$ is a full rank matrix (which is often the case), the representation problem of finding $\alpha$ has an infinite number of solutions and, therefore, certain constraints need to be imposed. One simple approach is to find an $\alpha$ with the highest sparsity, i.e., largest number of zero coefficients, which is equivalent to minimizing the $\ell_0$ norm of $\ell_0$. Note that the $\ell_0$ norm of a vector (i.e., the number of nonzero entries in it) is not technically a norm. However, this approach is very sensitive to perturbations in the signal. Typically, sparse coding with $\ell_1$ regularization is used, which is equivalent to optimizing the

following cost function

$$f(D) = \min_{\alpha \in \mathbb{R}^k} \| x - D\alpha \|_2^2 + \lambda_1 \| \alpha \|_1,$$

where $\alpha$ is a regularization parameter and the $\ell_1$ norm of an $n$–dimensional vector is given by $\| x \|_1 = \sum_{i=1}^{n} |x_i|$. It is well known that $\ell_1$ regularization results in few nonzero coefficients in $\alpha$ — in other words the solution of the optimization problem above is very sparse.

If, instead of considering an individual signal, we consider a training set $\{x_1, x_2, \ldots, x_m\}$ of $m$ observations/signals in $\mathbb{R}^n$, our goal is to learn a dictionary $D \in \mathbb{R}^{n \times k}$ such that each observation can be well approximated by a linear combination of $k$ columns of $D$. The coefficients determining the linear combination are now contained in a matrix $A \in \mathbb{R}^{k \times m}$ (rather than a $k$–dimensional vector $\alpha$) and so the approximation of $X$ reduces to finding the decomposition $\tilde{X} = DA$ such that $\tilde{X}$ is as close (in some norm) to $X$ as possible. Denoting the matrix of observations by $X \in \mathbb{R}^{n \times m}$ and the matrix of coefficients by $A \in \mathbb{R}^{k \times m}$, the matrix form of the above equation becomes

$$\min_{D,A} \| X - DA \|_{\text{Frob}}^2 + \lambda_1 \sum_{i_1, i_2} |\alpha_{i_1, i_2}|, \tag{4.16}$$

where the $\ell_2$ norm of the columns of $D$ is typically constrained. Note that (4.16) is not jointly convex with respect to both $D$ and $A$, however it is convex with respect to $D$ when $\alpha$ is fixed, and the other way around.

Optimization of the form (4.16) is exactly the approach taken by the LASSO (Tibshirani, 1996) and the "pursuit" (e.g., matching pursuit, basis pursuit among others) family of algorithms (Mallat and Zhang, 1993; Tropp, 2004; Chen et al., 1998). The properties of the dictionary $D$ set the limit on the sparsity level of $\alpha$, which in turn directly affects the convergences properties of the pursuit algorithms. Thus, the way such a dictionary is constructed in the first place is crucial for finding the sparse representations $\alpha$.

Designing dictionaries has developed into a separate line of research. On one hand, wavelet methods offer an easy solution by simply composing $D$ as a union of orthonormal

bases. However, one of the major disadvantages of this approach is that the bases are predefined in the sense that the wavelet transform used is chosen in advance and is not learnt from the data points. On the other hand, many machine learning approaches focus on designing *data adapted* dictionaries — finding an optimal dictionary is treated as a learning problem in which $D$ is iteratively updated based on the training data points.

Sparsity is not the only desirable properly in dictionary learning, another one is structure among the dictionary elements. As pointed out by Jenatton et al. (2010), it is often desirable to encode higher order information about the supports of the dictionary elements that reflects the structure of the data. This becomes particularly clear in image data where features associated to the pixels of an image are naturally organized on a grid. In this case the supports of the dictionary elements explaining the variability of images are naturally expected to be localized or have some regularity with respect to that grid.

Structured sparsity type dictionaries, also knowns as structured sparse PCA, (see (Jenatton et al., 2010) and the references they cite) are based on the so-called structure sparsity inducing norm (Jenatton et al., 2011). This norm induces structured sparsity in the following sense: the solutions to a learning problem regularized by this norm have a sparse support which belongs to a certain set of groups of variables.

A variety of other methods, such as but not limited to K-SVD(Aharon et al., 2006), Bayesian methods(Lewicki and Sejnowski, 2000), and discriminative dictionaries (Mairal et al., 2009), operate at a single scale (Lewicki and Sejnowski, 2000).

Yet another type of dictionaries incorporate a more refined notion of structure, one that is constructed in a multiscale/multilevel manner (Mairal et al., 2008; Thiagarajan et al., 2011; Magoarou and Gribonval, 2014). Another notable example in this category is the multiresolution inspired and data–adaptive dictionary proposed by Allard et al. (2012). Their dictionary consisting of geometric wavelets is based on a multiresolution analysis that adapts to arbitrary nonlinear manifolds modeling the data space.

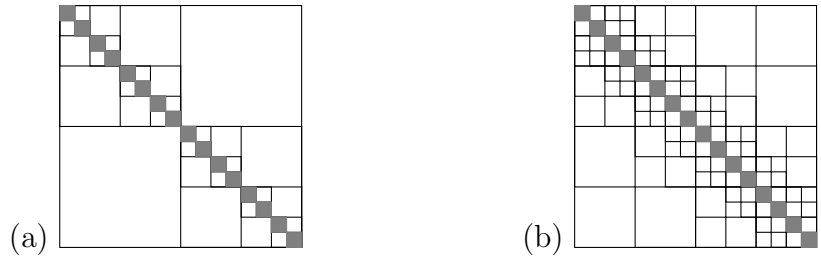



Figure 4.1: **Hierarchical matrix structure**. Depending on the constraints discussed in Section 4.6 different types of hierarchical matrices have quite different tessellations. For visual clarity in these figures we assume an ideal scenario, in which the clusters consist of the same number of coordinates and the cluster trees are binary and balanced, of height 4. (a) In an HODLR matrix the diagonal blocks are dense (shown in gray), while the off-diagonal blocks (shown in white) can be approximated by low rank matrices. (b) $\mathcal{H}^2$ matrices are a refinement of this idea. Blocks which can be approximated by low rank matrices are shown in white.

## 4.6 Hierarchical Matrices

Hierarchical matrices are a class of dense matrices which have a special subblock structure predicated on the assumption that: (a) these dense matrices can be recursively subdivided into subblocks based on a tree structure; and (b) certain subblocks arising at different levels in the tree are well approximated by low rank matrices. To describe the way hierarchical matrices are organized we start with $n$ points $X = \{x_1, x_2, \ldots, x_n\}$. The points are typically low dimensional, i.e., $x_i \in \mathbb{R}^d$ with $d \leq 3$. Each $A_{i,j}$ entry of a hierarchical matrix $A$ is interpreted as an interaction between points $x_i$ and $x_j$. Let the set $C \subset X$ denote a cluster of points which are close to each other in some metric (e.g., in term of their Euclidean distance). If $C_I \subset X$ and $C_J \subset X$ are respectively the $I$-th and $J$-th cluster, then $A_{I,J} \in \mathbb{R}^{|C_I| \times |C_J|}$ is a submatrix of $A$ which contains the interactions between the points in the clusters $C_I$ and $C_J$. We can define a hierarchical division of the points in $X$ in the form of a (typically balanced) tree. Letting each cluster be a node in that tree, the child, parent and sibling relationships between the clusters in the tree is based on the cluster assignments of the points, as follows

(i) cluster $C_I$ is a child of cluster $C_J$ (with $C_J$ being the parent of $C_I$), if $C_I \subset C_J$. (Note that the ancestors of $C_J$ in the cluster tree, not just its child cluster $C_I$, are subsets

of $C_J$);

(ii) two clusters $C_I, C_J$ are siblings if they have the same parent, i.e., they are different subsets of the same set.

If two disjoint clusters $C_I, C_J$ are separated by a distance of at least $m$, the clusters are **well separated**, and otherwise, they are considered **neighbors**. Cluster $C_I$ is said to be in the **interaction list** of cluster $C_J$ (and vice versa) if the matrix $A_{I,J}$ can be approximated by a low rank matrix.

Based on the above dependencies, various hierarchical matrices arise depending on several factors: (i) the tree type (e.g., binary or quad tree); (ii) the presence of interactions between neighboring clusters; (iii) a nested basis structure (i.e., whether or not the basis of a submatrix cut out by a parent cluster can be constructed from the bases of the submatrices cut out by its children clusters); (iv) the type of low rank, e.g., analytic (using SVD, CUR, QR and so on.) or algebraic (using Taylor series, Multipole expansion, interpolation). Ambikasaran (2013) and Börm et al. (2003) describe in much detail the construction of hierarchical matrices, as well as the differences between the various subtypes of hierarchical matrices (some of which are diagramed in Figure 4.1).

**Hierarchically off diagonal low rank (HODLR)** matrices are hierarchical matrices in which the interaction between the points in any cluster $C_I$ and the points in any nonintersecting cluster $C_J$ (i.e., $C_I \cap C_J = \varnothing$) leads to a low rank subblock $A_{I,J}$. More specifically, every off-diagonal block in an HOLDR matrix should have rank at most $r$, for some $r \ll n$. For example, assuming that we can define a binary, balanced tree (of height 2) on clusters of

$X$ (as described above), the corresponding two level HOLDR matrix $A$ would have the form

$$A = \left[\begin{array}{c|c} A_1^1 & U_1^1 A_{1,2}^1 V_2^{1\top} \\ \hline U_2^1 A_{2,1}^1 V_1^{1\top} & A_2^1 \end{array}\right] = \left[\begin{array}{cc|c} A_1^2 & U_1^2 A_{1,2}^2 V_2^{2\top} & U_1^1 A_{1,2}^1 V_2^{1\top} \\ \hline U_2^2 A_{2,1}^2 V_1^{2\top} & A_2^2 & \\ \hline U_2^1 A_{2,1}^1 V_1^{1\top} & & \begin{array}{c|c} A_3^2 & U_3^2 A_{3,4}^2 V_4^{2\top} \\ \hline U_4^1 A_{4,3}^2 V_3^{2\top} & A_4^2 \end{array} \end{array}\right],$$

where the diagonal blocks are denoted $A_i^k \in \mathbb{R}^{n/2^k \times n/2^k}$, and each off-diagonal block $A_{i,j}^k$ is of size $r \times r$ (where $r$ is the rank of the interactions between the sibling clusters of points), while the columns of $U_i^k \in \mathbb{R}^{n/2^k \times r}$ and $V_i^k \in \mathbb{R}^{n/2^k \times r}$ form a basis for, respectively, the column and row space of that block. Note that for a symmetric matrix $U_i^k = V_i^k$. Additionally, by construction it is required that the basis matrices can be expressed hierarchically — for example, the row coordinates that form the matrix $U_1^1 A_{1,2}^1 V_2^{1\top}$ are the union of the row coordinates that form $U_1^2 A_{1,2}^2 V_2^{2\top}$ and $U_2^2 A_{2,1}^2 V_1^{2\top}$. Thus, the basis $U_2^1$ needs to be expressible in terms of $U_1^2 \in \mathbb{R}^{n/4 \times r}$ and $U_2^2 \in \mathbb{R}^{n/4 \times r}$ in the form

$$U_2^1 = \begin{pmatrix} U_1^2 & 0 \\ 0 & U_2^2 \end{pmatrix} U_2',$$

for some small matrix $U_2' \in \mathbb{R}^{2r \times r}$. The matrices $U_i^k$ at higher levels can be computed recursively, but efficiently by storing only the relatively small $U_i'$ matrices. In general, at level $k$ each diagonal block $A_i^k$ (with $1 \le i \le 2^k$, $k \ge 0$) is subdivided into

$$A_i^k = \left[\begin{array}{c|c} A_{2i-1}^{k+1} & U_{2i-1}^{k+1} A_{2i-1}^{k+1} V_{2i}^{k+1\top} \\ \hline U_{2i}^{k+1} A_{2i,2i-1}^{k+1} V_{2i-1}^{k+1\top} & A_{2i}^{k+1} \end{array}\right],$$

where $A_{2i-1}^{k+1}$ and $A_{2i}^{k+1}$ denote the interactions between the sibling cluster, $U_{2i-1}^{k+1}$ and $U_{2i}^{k+1}$ denote the bases for the column space of the respective subblock, while $V_{2i-1}^{k+1}$ and $V_{2i}^{k+1}$ denote the bases for the row space of the respective subblock.

Figure 4.1(a) illustrates the tessellation pattern for an HODLR matrix similar to the one in this example, except the number of levels in the figure is four, rather than two.

**Hierarchically semi-separable (HSS)** matrices (Chandrasekaran et al., 2005) are a subclass of HODLR matrices which have the added constraint that a low rank basis for the interaction of a cluster with its siblings can be constructed from the low rank basis of the interaction of its children. Similarly to the HODLR example above, if we assume a binary, balanced tree (of height 2) on the clusters of $X$, the resulting two level HSS representation is of the form

$$A = \left[\begin{array}{c|c} A_1^1 & U_1^1 A_{1,2}^1 V_2^{1\top} \\ \hline U_2^1 A_{2,1}^1 V_1^{1\top} & A_2^1 \end{array}\right]$$
$$= \left[\begin{array}{cc|cc} A_1^2 & U_1^2 A_{1,2}^2 V_2^{2\top} & \begin{array}{c|c} U_1^2 & S_1^2 \\ \hline U_2^2 & S_2^2 \end{array} A_{1,2}^1 \left[\begin{array}{c|c} V_3^2 & R_3^2 \\ \hline V_4^2 & R_4^2 \end{array}\right]^\top & \\ U_2^2 A_{2,1}^2 V_1^{2\top} & A_2^2 & & \\ \hline \begin{array}{c|c} U_3^2 & S_3^2 \\ \hline U_4^2 & S_4^2 \end{array} A_{2,1}^1 \left[\begin{array}{c|c} V_1^2 & R_1^2 \\ \hline V_2^2 & R_2^2 \end{array}\right]^\top & & A_3^2 & U_3^2 A_{3,4}^2 V_4^{2\top} \\ & & U_4^1 A_{4,3}^2 V_3^{2\top} & A_4^2 \end{array}\right],$$

where $U_i^k \in \mathbb{R}^{n/2^k \times r}$ and $V_i^k \in \mathbb{R}^{n/2^k \times r}$. However, the additional constraint is that each $U_i^k$ can be constructed from the basis of the interactions of its children from level $k+1$. So, in general, at level $k$, $U_i^k$ and $V_i^k$ can be constructed, respectively, in the following block form

$$U_i^k = \left[\begin{array}{c|c} U_{2i-1}^{k+1} & S_{2i-1}^{k+1} \\ \hline U_{2i}^{k+1} & S_{2i}^{k+1} \end{array}\right] \quad \text{and} \quad V_i^k = \left[\begin{array}{c|c} V_{2i-1}^{k+1} & R_{2i-1}^{k+1} \\ \hline V_{2i}^{k+1} & R_{2i}^{k+1} \end{array}\right],$$

where the matrices $S_{2i-1}^{k+1}$ and $S_{2i}^{k+1}$ are called the downward pass operators, while $R_{2i-1}^{k+1}$ and $R_{2i}^{k+1}$ are called the upward pass operators (Ambikasaran, 2013). The downward and upward pass operators can be computed by orthogonalizing $U_i^k$ and $V_i^k$, respectively (for example, by a truncated SVD (Lessel et al., 2016)).

**$\mathcal{H}$ matrices** (Hackbusch, 1999; Hackbusch and Khoromskij, 2000) and **$\mathcal{H}^2$ matrices** (Börm and Garcke, 2007; Börm, 2007) are two classes of hierarchical matrices which share a slightly more restrictive property in comparison to the HODLR and HSS subclasses. Rather than limiting the interactions of nonintersecting clusters to low rank, for $\mathcal{H}$ and $\mathcal{H}^2$ matrices the interactions between neighboring clusters are full rank and the interactions between well separated clusters are low rank. The distinction between $\mathcal{H}^2$ and $\mathcal{H}$ stems from the fact that $\mathcal{H}^2$ matrices are a subclass of $\mathcal{H}$ matrices with the additional constraint that low rank basis for the interaction of a cluster with its siblings can be constructed from the low rank basis of the interaction of its children. The resulting tessellation pattern for $\mathcal{H}^2$ matrices is illustrated in Figure 4.1(b). Finally, **FMM matrices** are $\mathcal{H}^2$ matrices whose matrix–vector product can be computed in time $O(m+n)$ (for a matrix of size $m \times n$) using the Fast Multiple Method (FMM) (Greengard and Rokhlin, 1987; Beatson and Greengard, 1997). FMM matrices have constraints on the type of clustering tree as well as the low rank structure imposed by the interactions between sibling clusters.

Large and dense matrices such as kernel/covariance matrices and operators arising in applications such as integral equations, interpolation and inverse problems can be approximated by hierarchical matrices (Ambikasaran, 2013; Ambikasaran and Darve, 2014). Once a matrix is represented in a hierarchical form, downstream operations on it such as matrix–matrix multiplication, inversion and LU factorization, become significantly faster (Hackbusch, 1999; Börm and Garcke, 2007; Ambikasaran and O'Neil, 2014). For example, inverting an $\mathcal{H}$ matrix yields a good approximate inverse, which, in turn can be used either for preconditioning of large linear system or for constructing fast direct solvers (Chandrasekaran et al., 2005; Ghysels et al., 2016).

Constructing and storing the cluster tree, which defines the hierarchical matrix block structure, is all that is required for storing such an approximation. While computing the cluster tree can be expensive, the computational cost of constructing hierarchical matrices (and therefore its downstream operations) can be reduced for the symmetric case (Ambikasaran

and O'Neil, 2014). Additionally, Martinsson (2011) propose reducing the complexity of constructing hierarchical matrices by randomized low rank matrix approximation algorithms (see Chapter 6 for a review of low rank approximations).

The connections between hierarchical matrices and the other multilevel/multiscale methods discussed in this chapter become more apparent when the constraint that the same rank parameter $r$ is used at every level is dropped and the rank is instead an adaptive parameter (Börm, 2007).

Finally, hierarchical matrices rely on the fact that the underlying data is low dimensional and as the number of dimensions grows, hierarchical matrix construction becomes nontrivial mainly due to the infeasibility of building cluster trees in higher dimensions. Thus, while hierarchical matrices offer tremendous speed up for low dimensional problems arising in physics, their adoption, with a few notable exceptions (Ambikasaran and O'Neil, 2014; Cheng et al., 2014; Koutis et al., 2011), has been fairly limited in machine learning applications.

# CHAPTER 5

# MULTIRESOLUTION MATRIX FACTORIZATION

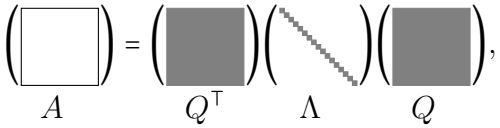
This chapter defines Multiresolution Matrix Factorization (MMF) and introduces several algorithms for computing the MMF of symmetric matrices. This is an expanded version of the material we presented in (Kondor et al., 2014).

## 5.1 Multiresolution Matrix Factorization (MMF)

In Sections 2.1 and 2.6 we saw that Fourier analysis and the eigendocomposition of a symmetric matrix $A \in \mathbb{R}^{n \times n}$ are closely related, since finding the Fourier basis reduces to finding the eigendecomposition

$$A = \sum_{i=1}^{r} \lambda_i \, q_i \, q_i^\top, \tag{5.1}$$

where $\{q_1, q_2, \ldots, q_r\}$ are the normalized eigenvectors of $A$ with their corresponding eigenvalues $\{\lambda_1, \lambda_2, \ldots, \lambda_r\}$ and $r \leq n$ is the rank of $A$. The matrix factorization equivalent of (5.1) is also called Principal Component Analysis (PCA) (Eckart and Young, 1936) and has the form




(5.2)

where $Q$ is an orthogonal matrix whose columns contain the eigenvectors and $\Lambda$ contains the eigenvalues on its main diagonal. The drawback of PCA is that eigenvectors are almost always dense, while matrices occurring in learning problems, especially those related to graphs, often have strong locality properties, whereby they more closely couple certain clusters of nearby coordinates than those farther apart with respect to some underlying topology. In such cases, modeling $A$ in terms of a basis of global eigenfunctions is both computationally wasteful and conceptually absurd: a localized dictionary would be more appropriate. This is part of the reason for the recent interest in sparse PCA (sPCA) algorithms (Jenatton et al., 2010, 2011), in which the vectors $Q_{i,:}$ in (5.2) are constrained to be sparse, while the

orthogonality constraint may be relaxed. However, sPCA is liable to suffer from the opposite problem of capturing structure locally, but failing to recover larger scale patterns in $A$.

In contrast to the "one–shot" diagonalization of $A$ in (5.2), Multiresolution Matrix Factorization (MMF) is a multilevel factorization — it applies not just one but a sequence of sparse orthogonal transforms to $A$, which capture structure at different resolution levels. After the first orthogonal transform $U_1$ is applied, the subset of rows/columns of $U_1 A U_1^\top$ which interact the least with the rest of the matrix capture the finest scale structure in $A$, so the corresponding rows of $U_1$ are designated level one wavelets, and these dimensions are subsequently kept invariant. Then the process is repeated by applying a second orthogonal transform to yield $U_2 U_1 A U_1^\top U_2^\top$ and splitting off another subspace of $\mathbb{R}^n$ spanned by second level wavelets, and so on, until some desired number of dimensions are split off

$$A \mapsto U_1 A U_1^\top \mapsto U_2 U_1 A U_1^\top U_2^\top \mapsto \ldots \cdots \mapsto U_L \ldots U_2 U_1 A U_1^\top U_2^\top \ldots U_L^\top. \tag{5.3}$$

Ultimately, the sequence (5.3) results in an $L$ level factorization of the form

$$A = U_1^\top U_2^\top \ldots U_L^\top H U_L \ldots U_2 U_1, \tag{5.4}$$

where the matrix $H$ is close to diagonal.

The central idea behind MMF is to convert multiresolution analysis into a matrix factorization by focusing on how multiresolution analysis compresses the matrix $A$. Recall from Section 2.7 and Figure 2.2 that multiresolution analysis with respect to a symmetric smoothing matrix $A \in \mathbb{R}^{n \times n}$ consists of finding a sequence of spaces $V_L \subset \cdots \subset V_2 \subset V_1 \subset V_0 = L(X) \cong \mathbb{R}^n$, where $\delta_\ell = \dim(V_\ell)$. Also recall that according to MRA3 (2.34) the scaling transform $S_\ell$, responsible for the $\Phi_{\ell-1} \to \Phi_\ell$ basis change, and the detail transform $D_\ell$, responsible for the $\Phi_{\ell-1} \to \Psi_\ell$ basis change, can be combined into a single orthogonal matrix $U_\ell$ of size $\delta_{\ell-1} \times \delta_{\ell-1}$ . If we instead set

$$U_\ell \leftarrow U_\ell \oplus I_{n-\delta_{\ell-1}}, \tag{5.5}$$

each $U_\ell$ matrix can be extended to size $n \times n$. So $A$ becomes

$$\underbrace{U_1 A U_1^\top}_{\text{denoted } A_1}$$

in the basis $\Phi_1 \cup \Psi_1$, then $A$ becomes

$$\underbrace{U_2 U_1 A U_1^\top U_2^\top}_{\text{denoted } A_2}$$

in the basis $\Phi_2 \cup \Psi_2 \cup \Psi_1$, and so on until after $L$ levels $A$ becomes

$$H \coloneqq \underbrace{U_L \dots U_2 U_1 A U_1^\top U_2^\top \dots U_L^\top}_{\text{denoted } A_L} \tag{5.6}$$

in the basis $\Phi_L \cup \Phi_{L-1} \cup \cdots \cup \Psi_1$. Therefore, similar to the way that Fourier analysis corresponds to eigendompositions, multiresolution analysis essentially factorizes $A$ in the form

$$A = U_1^\top U_2^\top \dots U_L^\top H U_L \dots U_2 U_1, \tag{5.7}$$

subject to the constraints:

(i) Similar to the sparsity requirements for the MRA transforms $\mathcal{S}_\ell$ and $\mathcal{D}_\ell$ in Figure 2.2, in MMF each of the $U_\ell$ orthogonal matrices must also be sufficiently sparse.

(ii) Outside a square block of size $\delta_{\ell-1} \times \delta_{\ell-1}$ each $U_\ell$ is the identity. For simplicity for now we assume that this block is in the upper left corner — Figure 5.1 illustrates the shape of the blocks in each of the $U_1, \dots, U_L$ matrices (after a permutation of their rows/columns by $\Pi$ for the sake of visual clarity).

(iii) By MRA3 (2.34) the first $\delta_L = \dim(V_L)$ rows of $A_L = U_L \dots U_2 U_1$ (5.6) contain the scaling functions $\{\psi_m^L\}_{m \in \mathbb{Z}}$ and the remaining rows contain the $\{\phi_m^L\}, \{\phi_m^{L-1}\}, \dots, \{\phi_m^1\}$

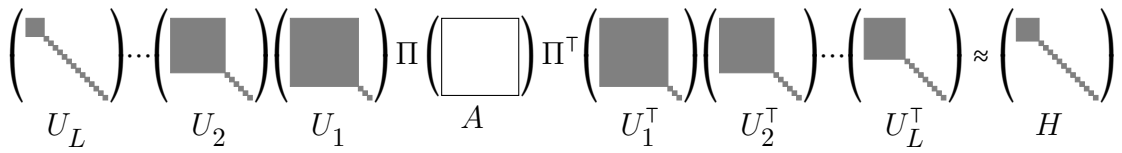



Figure 5.1: **MMF factorization schematic**. As $\ell$ increases, an increasingly large part of the $U_\ell$ matrices, specifically all but a $\delta_{\ell-1} \times \delta_{\ell-1}$ submatrix (shown as a gray square), is just the identity $I_{n-\delta_{\ell-1}}$ (shown as gray "tails" along the diagonal of $U_\ell$). The purpose of the permutation matrix $\Pi$ is to ensure, purely for visualization purposes, that it is always the *last* $n - \delta_{\ell-1}$ coordinates that are fixed by $U_\ell$. MMF algorithms do not impose this permutation constraint and at each level *some* $n - \delta_{\ell-1}$ coordinates can be fixed instead.

wavelets.

While in the Fourier (i.e., eigendecomposition) case $H$ would be simply diagonal, in MMF it has a more complex structure — in particular, it consists of four distinct blocks

$$H = \begin{pmatrix} H_{\Phi,\Phi} & H_{\Phi,\Psi} \\ H_{\Psi,\Phi} & H_{\Psi,\Psi} \end{pmatrix} = \begin{pmatrix} H_{1:\delta_L,1:\delta_L} & H_{1:\delta_L,\delta_L:n} \\ H_{\delta_L+1:n,1:\delta_L} & H_{\delta_L+1:n,\delta_L+1:n} \end{pmatrix}. \tag{5.8}$$

In the above formula $H_{\Phi,\Phi}$ is effectively a compression of $A$ down to size $\delta_L \times \delta_L$ and is therefore dense. The structure of the other three block matrices reflects the extent to which MRA1 is satisfied — in particular, the closer the wavelets are to being eigenfunctions, the better they filter the space by smoothness with respect to $A$, as defined in MRA1 (2.32). Below we define multiresolution factorizable matrices as those for which this condition is perfectly satisfied, i.e., matrices which have an MMF factorization with $H_{\Phi,\Psi} = H_{\Psi,\Phi}^\top = 0$ and $H_{\Phi,\Phi}$ being diagonal.

In order to define multiresolution factorizability, below we relax the form of (5.7) by allowing each $U_\ell$ to "fix" *some* set of $n - \delta_{\ell-1}$ coordinates, not necessarily the last $n - \delta_{\ell-1}$ coordinates as implied by the direct sum in (5.5). In other words, each $U_\ell$ transform must fix an $(n - \delta_{\ell-1})$–dimensional subspace of $\mathbb{R}^n$ spanned by some set of $n - \delta_{\ell-1}$ standard basis vectors. This generalized structure underlying each $U_\ell$ is captured by the following definition.

**Definition 1.** *Given a sequence of dimensions $n = \delta_0 \geq \delta_1 \geq \cdots \geq \delta_L \geq 1$ there is a nested sequence of sets*

$$n = S_0^{act} \supseteq S_1^{act} \supseteq S_2^{act} \supseteq \cdots \supseteq \ldots S_L^{act}, \text{ where } |S_\ell^{act}| = \delta_\ell,$$

*such that, if we let $S_\ell^{inact} = [n] \backslash S_{\ell-1}^{act}$, then $[U_\ell]_{S_\ell^{inact}, S_\ell^{inact}} = I_{n-\delta_{\ell-1}}$. $S_\ell^{act}$ and $S_\ell^{inact}$ are called the **active set** and **inactive set** at level $\ell$, respectively. Using the direct sum notation introduced in Section 1.2, this is equivalent to*

$$U_\ell = \oplus_{S_{\ell-1}^{inact}} I \oplus_{S_\ell^{act}} O \tag{5.9}$$

*for some $\delta_{\ell-1} \times \delta_{\ell-1}$ orthogonal matrix $O$.*

As a side note, since the elements in an (in)active set represent row indices (equivalently, by symmetry of $A$ they are also column indices) and there is a one to one correspondence between the elements of $S_\ell^{\text{act}}$ and $S_\ell^{\text{inact}}$ and the rows/columns of $A$, we will sometimes refer to the rows/columns of $A$ corresponding to the (in)active sets as **(in)active rows/columns**, or more generally as **(in)active coordinates**. The submatrices $[A_\ell]_{S_\ell^{\text{act}}, S_\ell^{\text{act}}}$ and $[A_\ell]_{S_\ell^{\text{inact}}, S_\ell^{\text{inact}}}$ will sometimes be called the **active submatrix/part** and the **inactive submatrix/part** of $A$, respectively.

The MMF wavelet and scaling functions can be obtained directly from the matrices $U_1, U_1U_2, \ldots, U_1U_2 \ldots U_L$. According to MRA3 (2.34), the scaling functions at level one are the rows of $U_1$ indexed by the coordinates in the second level active set $S_2^{\text{act}}$, while the wavelets for level one are the rows $S_1^{\text{act}} \backslash S_2^{\text{act}}$ of $U_1$ (i.e., the coordinates added to the inactive set at level one)

$$\begin{aligned} \phi_m^1 &= [U_1]_{m,:} \quad \text{with } m \in S_2^{\text{act}}, \\ \psi_m^1 &= [U_1]_{m,:} \quad \text{with } m \in S_1^{\text{act}} \backslash S_2^{\text{act}}. \end{aligned}$$

At the second level, the scaling functions $\{\phi_m^2\}$ and the wavelets $\{\psi_m^2\}$ can be read off from the rows of the cumulative matrix $U_2U_1$ and so on. In general, for level $\ell$ the scaling and wavelet functions are, respectively,

$$\phi_m^\ell = [U_\ell \dots U_2U_1]_{m,:} \quad \text{with } m \in S_{\ell+1}^{\text{act}},$$
$$\psi_m^\ell = [U_\ell \dots U_2U_1]_{m,:} \quad \text{with } m \in S_{\ell+1}^{\text{act}} \backslash S_\ell^{\text{act}}.$$

Allowing each $U_\ell$ to fix some $(n - \delta_{\ell-1})$–dimensional set of coordinates also affects the order in which rows are eliminated as wavelets, and the criterion for perfect multiresolution factorizability of $A$ now becomes $H \in \mathcal{H}^n_{S_L^{\text{act}}}$ (with $S_L^{\text{act}}$ being a class of matrices defined below).

**Definition 2.** *Given a set $S \subseteq [n]$, a matrix $H \in \mathbb{R}^{n\times n}$ is $S$–**core diagonal** if $H_i, j = 0$, unless $i, j \in S$ or $i = j$. Equivalently, by the direct sum notation from Section 1.2, if $H$ is $S$–**core diagonal**, it can be written in the form $H = D \oplus_S \bar{H}$ for some $|S| \times |S|$ matrix $\bar{H}$ and $D$ diagonal. The set of all $S$–core diagonal symmetric matrices of size $n \times n$ are denoted by $\mathcal{H}^n_S$.*

Note that we may sometimes refer to $\bar{H} = H_{S_L^{\text{act}}, S_L^{\text{act}}}$ as the **core** of the compression (assuming the MMF has $L$ levels) — implicit in this statement is the assumption that $H$ is $S_L^{\text{act}}$–core diagonal.

**Definition 3.** *Given an appropriate subset $\mathcal{O}$ of the group $O(n)$ of $n$–dimensional rotation matrices, a depth parameter $L \in \mathbb{N}$, and a sequence of integers $n = \delta_0 \geq \delta_1 \geq \delta_2 \geq \cdots \geq \delta_L \geq 1$, a **Multiresolution Matrix Factorization (MMF)** of a symmetric matrix $A \in \mathbb{R}^{n\times n}$ over $\mathcal{O}$ is a factorization of the form*

$$A = U_1^\top U_2^\top \dots U_L^\top H U_L \dots U_2U_1, \tag{5.10}$$

*where each $U_\ell \in \mathcal{O}$ orthogonal transformation matrix satisfies $[U_\ell]_{[n]\backslash S_{\ell-1}^{act}, [n]\backslash S_{\ell-1}^{act}} = I_{n-\delta_{\ell-1}}$*

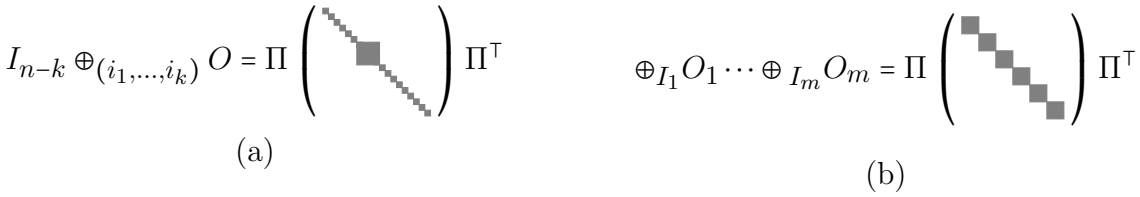



Figure 5.2: **Schematic of rotation matrices**. The two types of sparse rotation matrices that we consider are: (a) a simple rotation of order $k$ (Definition 5), (b) a compound rotation of order $k$ (Definition 6). Similarly to Figure 5.1, the purpose of the $\Pi$ permutation matrices is just to ensure that the blocks of the matrices appear contiguous in the figure.

*for some nested sequence of active sets* $[n] = S_0^{act} \supseteq S_1^{act} \supseteq \cdots \supseteq S_L^{act}$ *with* $|S_\ell^{act}| = \delta_{\ell-1}$ *and* $H \in \mathcal{H}^n_{S_L^{act}}$.

**Definition 4.** *We say that a symmetric matrix* $A \in \mathbb{R}^{n\times n}$ *is* ***fully multiresolution factorizable*** *over* $\mathcal{O} \in O(n)$ *with respect to the sequence* $(\delta_0, \delta_1, \ldots, \delta_L)$ *if it has a decomposition of the form described in Definition 3.*

The sequence $(\delta_0, \delta_1, \ldots, \delta_L)$ may follow some predefined law, such as geometric decay (i.e., $\delta_\ell = \lceil n\eta^\ell \rceil$ for some $\eta \in (0,1)$) or arithmetic decay (i.e., $\delta_\ell = n - \ell m$ for some $m \in \mathbb{N}$). Different types of MMFs arise depending on how the set $\mathcal{O}$ of sparse rotations (from which the $U_\ell$ transformations are chosen) is defined. In addition to the requirement that each $U_\ell$ fixes some $(n-\delta_{\ell-1})$–dimensional set of coordinates, it has to satisfy two additional requirements: (i) wavelets must be localized, as required by MRA2 (see Section 2.7), and (ii) each $U_\ell$ must be sparse in order to get a fast wavelet transform, as required by MRA3 in Section 2.7. We consider two alternatives defined below — elementary and compound rotations of order $k$. Figure 5.2 shows schematics illustrating the block form of these two types of matrices.

**Definition 5.** *We say that* $U \in \mathbb{R}^{n\times n}$ *is an* ***elementary rotation of order k*** *(sometimes also called a* $k$*–****point rotation)*** *if it is an orthogonal matrix of the form*

$$U = I_{n-k} \oplus_{i_1,\ldots,i_k} O \tag{5.11}$$

*for some* $\{i_1, \ldots, i_k\} \subseteq [n]$ *and* $O \in O(k)$*. The set of all such matrices we denote* $O_k(n)$*.*

A $k$–order elementary rotation is very local since it only touches coordinates $\{i_1, \ldots, i_k\}$ and leaves the rest invariant. The simplest case is the second order (i.e., $(i_1, i_2)$) rotation, which is equivalent to a Givens rotation $U = U_{i_1,i_2,\theta}$ where $(i_1, i_2)$ is a pair of rows/columns and $\theta \in [0, 2\pi)$ is the rotation angle between them — recall (4.4). Applying a Givens rotation $U_{i_1,i_2,\theta}$ allows for the diagonalization of the $2\times2$ submatrix $[A_{\ell-1}]_{(i_1,i_2),(i_1,i_2)}$ (in closed form according to the updates (4.6)) in the form

$$\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} [A_{\ell-1}]_{i_1,i_1} & [A_{\ell-1}]_{i_1,i_2} \\ [A_{\ell-1}]_{i_2,i_1} & [A_{\ell-1}]_{i_2,i_2} \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} = \begin{pmatrix} b_1 & 0 \\ 0 & b_2 \end{pmatrix} = \begin{pmatrix} [A_\ell]_{i_1,i_1} & [A_\ell]_{i_1,i_2} \\ [A_\ell]_{i_2,i_1} & [A_\ell]_{i_2,i_2} \end{pmatrix}$$

Indeed, we already saw these updates when reviewing Jacobi's eigenvalue decomposition in Section 4.1. Jacobi's algorithm works precisely by constructing an MMF factorization over Givens rotations with respect to the sequence $n = \delta_0 = \delta_1 = \cdots = \delta_L$ — in other words, Jacobi's algorithm is an MMF which does not fix any coordinates at all at each level $\ell$.

**Definition 6.** *We say that* $U \in \mathbb{R}^{n\times n}$ *is a* ***compound rotation of order k*** *if it is an orthogonal matrix of the form*

$$U = \oplus_{\{i_1^1,\ldots,i_{k_1}^1\}} O_1 \oplus_{\{i_1^2,\ldots,i_{k_2}^2\}} O_2 \cdots \oplus_{\{i_1^m,\ldots,i_{k_m}^m\}} O_m \tag{5.12}$$

*for some partition* $\{k_1^1, \ldots, k_{k_1}^1\} \cup \cdots \cup \{k_1^m, \ldots, k_{k_m}^m\}$ *of* $[n]$ *with* $k_1, \ldots, k_m < k$*, and some sequence of orthogonal matrices* $O_1, \ldots, O_m$ *of the appropriate sizes. The set of all such matrices we denote* $O_k^*(n)$*.*

Intuitively, the compound rotations can be thought of as a set of multiple elementary rotations executed in parallel, which consequently allow for computing MMF factorizations faster and in much more compact form.

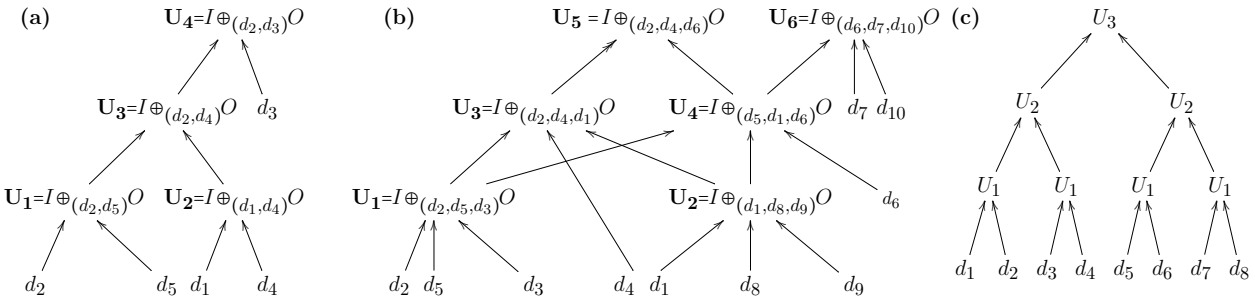



Figure 5.3: **MMF rotation hierarchy**. (a) The rotation tree of a second order Jacobi MMF of a matrix $A \in \mathbb{R}^{5\times5}$. (b) The partial rotation hierarchy of a third ($k = 3$) order Jacobi MMF of $A \in \mathbb{R}^{10\times10}$. Here the MMF only eliminates one dimension after each rotation. (c) A similar tree for a second order greedy parallel MMF of $A \in \mathbb{R}^{8\times8}$. An example of three rotation matrices which are described by this rotation tree is: $U_1 = \oplus_{(d_1,\, d_2)} O \oplus_{(d_3,\, d_4)} O \oplus_{(d_5,\, d_6)} \oplus_{(d_7,\, d_8)} O$, $U_2 = I_6 \oplus_{(d_1,\, d_3)} O \oplus_{(d_5,\, d_7)} O$ and $U_3 = I_8 \oplus_{(d_3,\, d_5)} O$. Unlike the other two figures here each block of the direct sum is represented as a separate leaf. Note that this tree is perfectly balanced.

**Definition 7.** *Multiresolution factorizations with $\mathcal{O} = O_k(n)$ we call **Jacobi MMFs** and multiresolution factorizations with $\mathcal{O} = O_k^*(n)$ we call **parallel MMFs**.*

One way to interpret the sequence (5.3) of elementary (Definition 5) or compound (Definition 6) rotations in MMF is to look at the coordinate ordering induced by these rotations — Figure 5.3 shows several examples. Each leaf node $d_i$ in Figure 5.3 represents a coordinate of $A$, while each internal node represents a rotation. Two internal nodes are connected by an edge if two rotations share a coordinate. For example, in Figure 5.3(b) at level one the orthogonal matrix $U_1 = I \oplus_{(d_2,d_5,d_1)} O$ is applied to both sides of $A$ (accordingly, $d_1$ is added to the inactive list $S_1^{\text{inact}}$), while at the third level the orthogonal matrix $U_3 = I \oplus_{(d_2,d_4,d_3)} O$ is applied to $U_2U_1AU_1^\top U_2^\top$. $U_1$ and $U_3$ share one coordinate, $d_2$, and are therefore connected by an edge. The direction of the edge denotes the level order, i.e., lower level rotations point to higher level ones. We call this type of structure induced by the MMF rotations the **MMF rotation hierarchy**. In some cases — for example, in the case of elementary rotations of order $k = 2$ in Figure 5.3(a) — the MMF rotation hierarchy is a tree, which we call the **MMF rotation tree**. In other cases, such as the third order Jacobi MMF in Figure 5.3(b), the MMF rotation hierarchy has lattice–like structure.

## 5.2 Computing MMFs

Similar to the way PCA expresses matrices in terms of a small dictionary of vectors (5.1), MMF approximates $A$ in the form

$$A^* = \sum_{i,j=1}^{\delta_L} \beta_{i,j} \phi_i^L \phi_j^{L\top} + \sum_{\ell=1}^{L} \sum_{i=1}^{\delta_\ell} \eta_i^\ell \psi_i^\ell \psi_i^{\ell\top}, \tag{5.13}$$

where the $\eta_i^\ell = \langle \psi_i^\ell, A\psi_i^\ell \rangle$ wavelet frequencies are the diagonal elements of the $H_{\Psi,\Psi}$ block of $H$, while the $\beta_{i,j}$ coefficients are the entries of the $H_{\Phi,\Phi}$ block (note the similarity between the compression of a signal $f$ in classical MRA (2.20) and the (5.13) form). Thus, given the appropriate subset of sparse rotations $\mathcal{O}$ and the sequence $(\delta_1, \ldots, \delta_L)$, finding the best MMF factorization of a symmetric matrix $A \in \mathbb{R}^{n\times n}$ requires solving the optimization problem

$$\begin{aligned} &\underset{\substack{[n] \supseteq S_1^{\text{act}} \supseteq \cdots \supseteq S_L^{\text{act}} \\ H \in \mathcal{H}^n_{S_L^{\text{act}}}, U_1, \ldots, U_L \in \mathcal{O}}}{\text{minimize}} \quad \|A - U_1^\top \ldots U_L^\top H U_L \ldots U_1\|. \end{aligned} \tag{5.14}$$

Assuming that the error $\|\cdot\|$ in (5.14) is measured in terms of Frobenius norm, which is rotationally invariant, we can rewrite the equation above as

$$\begin{aligned} &\underset{\substack{[n] \supseteq S_1^{\text{act}} \supseteq \cdots \supseteq S_L^{\text{act}} \\ U_1, \ldots, U_L \in \mathcal{O}}}{\text{minimize}} \|U_L^\top \ldots U_1 A U_1^\top \ldots U_L^\top\|^2_{\text{residual}}, \end{aligned} \tag{5.15}$$

where $\|\cdot\|^2_{\text{residual}}$ is the residual norm

$$\|H\|^2_{\text{residual}} = \sum_{i \neq j \text{ and } (i,j) \neq S_L^{\text{act}} \times S_L^{\text{act}}} |H_{i,j}|^2.$$

Intuitively, the objective of the MMF factorization is to find the series of sparse rotations (discussed at the beginning of Section 5.1),

$$A \equiv A_0 \xrightarrow{U_1} A_1 \xrightarrow{U_2} \cdots \xrightarrow{U_L} A_L \tag{5.16}$$

that bring $A$ to a form as close to diagonal as possible. As soon as we designate a certain set $J_\ell \coloneqq S^{\text{act}}_{\ell-1} \backslash S^{\text{act}}_\ell$ of rows/columns of $A_\ell$ wavelets and the indices contained in $J_\ell$ are removed from the active set $S^{\text{act}}_\ell$, the active part $[A_\ell]_{S^{\text{act}}_\ell, S^{\text{act}}_\ell}$ of the matrix $A_\ell = U_\ell \dots U_2 U_1 A U_1^\top U_2^\top \dots U_\ell^\top = U_\ell A_{\ell-1} U_\ell^\top$ will split into four submatrices:

(i) the new active submatrix $[A_\ell]_{S^{\text{act}}_{\ell+1}, S^{\text{act}}_{\ell+1}}$,

(ii) the inactive submatrix $[A_\ell]_{J_\ell, J_\ell}$,

(iii) the matrices $[A_\ell]_{S^{\text{act}}_{\ell+1}, J_\ell}$ and $[A_\ell]_{J_\ell, S^{\text{act}}_{\ell+1}}$.

Applying subsequent rotations on $[A_\ell]_{S^{\text{act}}_\ell, S^{\text{act}}_\ell}$ (from both left and right), however, must by definition leave the coordinates in $J_\ell$ invariant — in particular, the matrix in (ii) will stay invariant, while each of the two matrices in (iii) can be rotated, only row–wise and only column–wise, respectively. This means that as soon as rotation $U_\ell$ is applied on both sides of $A_{\ell-1}$, the $\ell_2$ norm of the $J_\ell$ rows/columns of $A_\ell$ is already committed to the final error. This leads us to the following proposition about the Frobenius norm approximation error (5.15).

**Proposition 1.** *Given MMF as defined in Definition 3, the objective function (5.15) can be expressed as* $\mathcal{E} = \sum_{\ell=1}^{L} \mathcal{E}_\ell$, *where* $\mathcal{E}_\ell = \|[A_\ell]_{J_\ell, J_\ell}\|^2_{\textit{off-diag}} + 2\,\|[A_\ell]_{J_\ell, S^{act}_\ell}\|^2_{\textit{Frob}}$ *and* $\|M\|^2_{\textit{off-diag}} \coloneqq \sum_{i \neq j} |M_{i,j}|^2$.

Rather than minimizing the error (5.15) globally, we apply a greedy strategy. The algorithms described in the following sections are based on a greedy approach suggested by the proposition above — at each level they find the rotation $U_\ell$ that minimizes the error from

**Algorithm 1** GreedyJacobiMMF(A): computing the Jacobi MMF of $A$ with $\delta_\ell = n - \ell$.

**Input**: $k$, $L$, and a symmetric matrix $A_0 = A \in R^{n\times n}$
**set** $S_0 \leftarrow [n]$ (the active set)
**for**($\ell = 1$ to $L$){
  **foreach** $I = (i_1, \ldots, i_k) \in (S_{\ell-1})^k$ with $i_1 < \ldots < i_k$
    **compute** $\mathcal{E}_I = \min_{O\in O(k)} \mathcal{E}_I^O$, as defined in (5.17)
  **set** $I_\ell \leftarrow \arg\min_I \mathcal{E}_I$
  **set** $O_\ell \leftarrow \arg\min_{O\in O(k)} \mathcal{E}_{I_\ell}^O$
  **set** $U_\ell \leftarrow I_{n-k} \oplus_{I_\ell} O_\ell$
  **set** $S_\ell \leftarrow S_{\ell-1} \backslash \{i_k\}$
  **set** $A_\ell \leftarrow U_\ell A_{\ell-1} U_\ell^\top$
}
**Output**: $U_1, \ldots, U_L$ and $H = A_L \downarrow \mathcal{H}^n_{S_L}$

Proposition 1. In other words, each $U_\ell$ makes $\delta_\ell - \delta_{\ell-1}$ rows/columns of $A$ as close to diagonal as possible and designates them as level $\ell$ wavelets. We introduce two types of MMFs — two *deterministic* algorithms (Sections 5.2.1 and 5.2.2) and a *randomized* algorithm (Section 5.2.3).

### *5.2.1 Jacobi MMFs*

In Jacobi MMFs, where each $U_\ell$ is an elementary rotation of oder $k$, we set $\{\delta_1, \ldots, \delta_L\}$ so as to split off a constant number $m < k$ of wavelets are each level. For simplicity we set $m = 1$ and moreover we make the natural assumption that this wavelet is one of the rows/columns involved in the rotation, i.e., $J_\ell = \{i_k\}$,

**Proposition 2.** *If $U_\ell = I_{n-k} \oplus_I O$ with $I = \{i_1, \ldots, i_k\}$ and $J_\ell = \{i_k\}$, then the contribution of level $\ell$ to the MMF approximation error is*

$$\mathcal{E}_\ell = \mathcal{E}_I^O = 2 \sum_{p=1}^{k-1} [O[A_{\ell-1}]_{I,I} O^\top]_{k,p}^2 + 2[OBO^\top]_{k,k}, \tag{5.17}$$

*where $B = [A_{\ell-1}]_{I,S_\ell^{act}} ([A_{\ell-1}]_{I,S_\ell^{act}})^\top$.*

**Corollary 1.** *In the special case of* $k=2$ *and* $I_\ell = (i,j)$*,*

$$\mathcal{E}_\ell = \mathcal{E}^O_{(i,j)} = 2[O[A_{\ell-1}]_{(i,j),(i,j)}O^\top]^2_{2,1} + 2[OBO^\top]_{k,k}, \tag{5.18}$$

*where* $B = [A_{\ell-1}]_{(i,j),S^{act}_\ell}([A_{\ell-1}]_{(i,j),S^{act}_\ell})^\top$.

According to the greedy strategy, at each level $\ell$, the index tuple $I$ and the rotation $O$ must be chosen so as to minimize (5.17). The resulting algorithm, GreedyJacobiMMF, for the case of $k = 2$ is given in Algorithm 1, where $A_L \downarrow \mathcal{H}^n_{S^{\text{act}}_L}$ stands for zeroing out all the entries in $A_\ell$ except those on the diagonal of $A_\ell$ and and in the $[A_L]_{S^{\text{act}}_L, S^{\text{act}}_L}$ submatrix. The complexity of GreedyJacobiMMF $O(n^3)$. When $k = 2$, the MMF rotation hierarchy is a binary tree in which each $U_\ell$ takes two scaling functions from level $\ell - 1$ and passes on a single linear combination of them to the next level (see Figure 5.2(a)). In general, the more similar two rows $[A_\ell]_{i,:}$ and $[A_\ell]_{j,:}$ to each other, the smaller we can make (5.18) by choosing the appropriate $O$. If $A$ is a kernel or similarity matrix between the vertices of a graphs, for example, the $i$–th row of $A$ measures the similarity of vertex $i$ to all the other vertices in the graph. This means that if Algorithm 1 is run on such a graph kernel/similarity matrix, it will tend to pick pairs of adjacent or nearby vertices and then produce scaling functions that represent linear combinations of those vertices. Thus, second order MMFs effectively perform a hierarchical clustering on the rows/columns of $A$. Uncovering this sort of hierarchical structure is one of the goals of MMF analysis.

The idea of constructing wavelets by forming a tree of Givens rotations also underlies the Treelets algorithm (Lee et al., 2008) discussed in Section 4.2. In fact, for $k = 2$ Jacobi MMF is similar to both the Treelets method and Jacobi's algorithm (hence, the name of this type of MMF) in the sense that the rotation hierarchy is a binary tree (see Figure 5.3(a)). The binary tree is consequence of the fact that, when $k = 2$, the supports of any two MMF wavelets $\psi^\ell_1$ and $\psi^{\ell'}_1$ are either disjoint or one is contained in the other. Despite the identical hierarchy that Jacobi MMF and Treelets recover, the Treelets objective function does not

minimize the approximation error of the matrix factorization, as in (5.15). In particular, instead of minimizing the contribution of each rotation to the matrix approximation error, the Treelets algorithm chooses $I$ and $O$ such that at each level $\ell$ the largest off-diagonal entry of $A_{\ell-1}$ is zeroed out. Thus, the objective is to simply construct a basis supported on nested clusters of coordinates.

Jacobi MMFs with $k \geq 3$ are even more interesting because their rotation hierarchy is lattice–like — as shown in Figures 5.3(a) and (b), each $U_\ell$ in the hierarchy has $k$ children and $k-1$ parents. When $k \geq 3$, a single original coordinate, such as coordinate $d_6$ in Figure 5.3b, can contribute to multiple wavelets (e.g., $\psi_1^3$ and $\psi_1^4$ in the figure) with different weights, determined by all the orthogonal matrices along the corresponding paths in the MMF rotation hierarchy. Thus, higher order MMFs are more subtle than just a single hierarchical clustering: by building a lattice–like hierarchy of subspaces they capture a softer notion of hierarchy and can uncover multiple overlapping hierarchical structures in $A$.

Finally, at the beginning of this section we made the simplifying assumption that if $U_\ell$ is an elementary rotation of oder $k$, we set $\{\delta_0, \ldots, \delta_L\}$ so as to split off exactly one wavelet are each level. However, we could set $m < k$, which corresponds to a more aggressive wavelet elimination strategy and leads to a smaller overall number of levels in the factorization, i.e., $L < n$.

### *5.2.2 Parallel MMFs*

Since MMFs exploit hierarchical cluster–of–clusters type of structure in matrices, towards the bottom of the rotation hierarchy one expects to find rotations that act locally, within small subclusters, and thus do not interact with each other. Parallel MMFs combine many of these independent rotations into a single compound rotation, which yields factorizations that are both more compact and more interpretable in terms of resolving $A$ at a small number of distinct scales. Let's assume that it is the last coordinate in each $(i_1^1, \ldots, i_{k_1}^1), \ldots, (i_1^m, \ldots, i_{k_m}^m)$

**Algorithm 2** GreedyParallelMMF(A): computing the binary ($k = 2$) greedy parallel MMF of $A$ with $\delta_\ell = \lceil n2^{-\ell} \rceil$.

**Input**: $L$ and a symmetric matrix $A = A_0 \in \mathbb{R}^{n \times n}$
**set** $S_0 \leftarrow [n]$ (the active set)
**for** ($\ell = 1$ to $L$){
  **set** $p \leftarrow \lfloor |S_{\ell-1}|/2 \rfloor$
  **compute** $W_{i,j} = W_{j,i}$ as defined in (5.20) $\forall i, j \in S_{\ell-1}$
  **find** the matching $\{(i_1, j_1), \ldots, (i_p, j_p)\}$ minimizing $\sum_{r=1}^{p} W_{i_r, j_r}$
  **for** ($r = 1$ to $p$) **set** $O_r \leftarrow \arg\min_{O \in O(2)} \mathcal{E}^{O}_{(i_r, j_r)}$
  **set** $U_\ell \leftarrow \oplus_{(i_1,j_1)} O_1 \oplus_{(i_2,j_2)} O_2 \oplus \ldots \oplus_{(i_p,j_p)} O_p$
  **set** $S_\ell \leftarrow S_{\ell-1} \backslash \{i_1, \ldots, i_p\}$
  **set** $A_\ell \leftarrow U_\ell A_{\ell-1} U_\ell^\top$
}
**Output**: $U_1, \ldots, U_L$ and $H = A_L \downarrow \mathcal{H}^n_{S_L}$

block that gives rise to a wavelet, therefore $\delta_\ell$ decays by a constant factor of $(k-1)/k$ at each level.

**Proposition 3.** *If $U_\ell$ is a compound rotation of the form $U_\ell = \oplus_{I_1} O_1 \cdots \oplus_{I_m} O_m$ for some partition $I_1 \cup \cdots \cup I_m = [n]$ with $k_1, \ldots, k_m \leq k$, and some sequence of orthogonal matrices $O_1, \ldots, O_m$, then the contribution of level $\ell$ to the MMF error obeys*

$$\mathcal{E}_\ell \leq 2 \sum_{j=1}^{m} \left( \sum_{p=1}^{k_j - 1} [O_j [A_{\ell-1}]_{I_j, I_j} O_j^\top]^2_{k_j, p} + [O_j B_j O_j^\top]_{k_j, k_j} \right), \tag{5.19}$$

*where $B_j = [A_{\ell-1}]_{I_j, S^{act}_{\ell-1} \backslash I_j} ([A_{\ell-1}]_{I_j, S^{act}_{\ell-1} \backslash I_j})^\top$.*

The reason that (5.19), in contrast to (5.17), provides only an upper bound on $\mathcal{E}_\ell$ is that it double counts the contribution of the matrix elements $\{[A_\ell]_{k_j, k_{j'}}\}_{j,j'=1}^{m}$ at the intersection of pairs of wavelet rows/columns. Accounting for these elements explicitly would introduce interactions between the $O_j$ rotations, leading to a difficult optimization problem. Therefore, to find the optimal partition $I_1 \cup \cdots \cup I_m$ and the optimal rotations $O_1, \ldots, O_m$ we use the right hand side of (5.19) as a proxy for $\mathcal{E}_\ell$.

As in the Jacobi MMF, the binary case is the simplest as well. Optimizing $I_1 \cup \cdots \cup I_m$

reduces to finding the minimal cost matching amongst the indices in the active set $S_{\ell-1}^{\text{act}}$ with cost matrix

$$W_{i,j} = 2 \min_{O \in O(2)} \left( [O[A_{\ell-1}]_{(i,j),(i,j)} O^\top]_{2,1}^2 + [OBO^\top]_{k,k} \right), \tag{5.20}$$

where $B = [A_{\ell-1}]_{(i,j),S_{\ell-1}^{\text{act}} \setminus \{i,j\}} ([A_{\ell-1}]_{(i,j),S_{\ell-1}^{\text{act}} \setminus \{i,j\}})^\top$. This optimization problem is equivalent to finding the maximum matching of a fully connected graph whose adjacency matrix is $W$.

Recall that a matching of graph $G(V, E)$ is a subset $M \subset E$ of the edges, such that no two edges in $M$ share a common vertex in $G$. A graph $G$ can have more than one matching. The weight of a matching is the sum of the weights of the edges in $M$, so the maximum matching is the one with the highest weight among all possible matchings of $G$. An exact solution to the maximum matching problem can be found in time $O(|V|^3)$, using a weighted version of the famous Blossom Algorithm by Edmonds (1965). However, it is well known that a simple greedy strategy yields a 2–approximation of the optimal solution in time $O(|V|)$.

Using this greedy strategy, we obtain a 2–approximation of the optimal matching by setting $(i_1, j_1) = \arg\min_{i,j \in S_{\ell-1}^{\text{act}}} W_{i,j}$, then setting $(i_2, j_2) = \arg\min_{i,j \in S_{\ell-1}^{\text{act}} \setminus \{i_1, j_1\}} W_{i,j}$, and so on until the active set becomes empty. In general, the most expensive component of MMF factorizations is forming the $B$ matrices, which naively takes $O(n^k)$ time. However, in practice, techniques like locality sensitive hashing could allow this (as well as the entire algorithm) to run in time close to linear in $n$.

The resulting binary GreedyParallelMMF algorithm is shown in Algorithm 2, while its corresponding rotation hierarchy is shown in Figure 5.3(c).

We remark that the fast Haar transform is nothing but a binary parallel MMF, while the Cooley–Tukey Fourier transform is a degenerate MMF (in the sense that the MMF dimensionality sequence is $\delta_0 = \cdots = \delta_L$) of a complex valued matrix.

### 5.2.3 Randomized MMFs

Both MMF algorithms described in the previous two sections need to solve a global optimization problem at each level $\ell$ — they either find the set $I$ (for elementary rotations, Proposition 2) or the partition $I_1 \cup \cdots \cup I_m = [n]$ (for compound rotations, Proposition 3) over the *entire* active set $S_\ell^{\text{act}}$ in order to construct the rotation matrix $U_\ell$ and eliminate $J_\ell$ coordinates to maximally reduce the MMF residual (5.15). As the size $n$ of the input matrix $A \in \mathbb{R}^{n \times n}$ grows, however, this combinatorial optimization problem quickly becomes forbiddingly expensive. Even in the case of second order elementary rotations in Jacobi MMFs the complexity of level $\ell$ is $O(n^2)$ since at each level the algorithm performs an exhaustive search over all $(i_1, i_2) \in (S_\ell^{\text{act}})^2$ possible pairs to find the rotation $U_\ell = U_{i_1,i_2,\theta}$. In the case of second order compound rotations in parallel MMFs, the complexity is $O(n^3)$, since the algorithm uses the Blossom Algorithm to perform vertex matching over the cost matrix $W$ (5.20). For higher order rotations the complexity of parallel MMFs is even worse. Additionally, the rotation tree induced by Jacobi MMFs is not very well balanced, which naturally ruins the multiresolution character of the MMF factorization.

In order to solve these problems, we introduce a randomized MMF algorithm (Algorithm 3) which, similarly to GreedyJacobiMMF, solves the MMF optimization problem greedily, one rotation at a time, but rather than having to find the best rotation at each level, it finds the best rotation *locally*, considering only a small $k$–neighborhood of coordinates when deciding which tuple of coordinates to rotate at each level.

RandomizedMMF (Algorithm 3) is similar to GreedyJacobiMMF except that, rather than finding the best $k$–tuple $(i_1, \ldots, i_k)$ (in the case of $k = 2$, the best pair) of coordinates in the active set to construct a rotation with, it randomizes the process in the following steps, for each level $\ell$:

(i) pick a coordinate $i$ uniformly at random from the active set $S_\ell^{\text{act}}$.

(ii) find the $k-1$ other rows/columns $i_1, i_2, \ldots, i_{k-1} \in S_\ell^{\text{act}}$ of $A_\ell$ which are the closest to column $[A_\ell]_{:,j}$ in terms of inner product.

The $k$ coordinates $i_1, \ldots, i_{k-1}$ are selected according to some separable objective function $\phi(i_1, \ldots, i_{k-1})$ related to minimizing the contribution to the final error. Specifically, we use

$$\phi(i_1, \ldots, i_{k-1}) = \sum_{r=1}^{k-1} \left\langle [A_{\ell-1}]_{:,i}, [A_{\ell-1}]_{:,i_r} \right\rangle . \tag{5.21}$$

In other words, column $[A_{\ell-1}]_{:,i}$ forms a $k$–tuple with the $k-1$ other columns that it has the highest inner product (in absolute value) with.

(iii) perform the elementary $k$–point rotation $U_\ell = I_{n-k} \oplus_{(i,i_1,\ldots i_{k-1})} O_\ell$.

So, at each level RandomizedMMF has to compute only $k-1$ inner products, which is a significant reduction from the $O(n^2)$ complexity of finding the index $k$–tuple involved in $U_\ell$ in Jacobi MMFs. Of course, RandomizedMMF is no different than GreedyJacobiMMF as it requires computing the Gram matrix $G_\ell = A_{\ell-1}^\top A_{\ell-1}$ at a complexity of $O(n^3)$, but, as before, that needs to be done only at the first level with subsequent Gram matrices computed via the recursion $G_{\ell+1} = U_\ell G_\ell U_\ell^\top$.

Empirically, the advantage of Algorithm 3 becomes apparent when computing the MMF decomposition of medium size matrices with rotation order $k > 2$. As we show later in the experiments in Section 5.6, the approximation quality of Randomized MMFs is comparable to that of Jacobi and parallel MMFs, while allowing the faster factorization of matrices of a few thousand dimensions.

### *5.2.4 Computational Details*

Problems of the form $\min_{O \in O(k)} \|OBO^\top C\|$, called Procrustes problems, generally have easy $O(k^3)$ time closed form solutions. Unfortunately, both (5.17) and (5.19) involve mixed linear/quadratic versions of this problem, which are much more challenging. However, the following result shows that in the $k = 2$ case this may be reduced to solving a simple trigonometric equation.

**Algorithm 3** RANDOMIZEDMMF: computing the randomized MMF of $A$.

**Input**: $L$ and a symmetric matrix $A = A_0 \in \mathbb{R}^{n\times n}$
**set** $S_0 \leftarrow [n]$ (the active set)
**for** ($\ell = 1$ to $L$){
  **select** $i \in S_{\ell-1}$ uniformly at random
  **find** the $k$ neighborhood $(\bar{i}_1, \dots \bar{i}_{k-1})$ of $i$ in $A_{\ell-1}$, as described in (ii) and (5.21) above
  **set** $O \leftarrow \arg\min_{O\in O(k)} \mathcal{E}^O_{(\bar{i}_1,\dots\bar{i}_{k-1})}$
  **set** $U_\ell \leftarrow I_{n-k} \oplus_{(i,\bar{i}_1,\dots\bar{i}_{k-1})} O$
  **set** $A_\ell \leftarrow U_\ell A_{\ell-1} U_\ell^\top$
  **set** $p = \text{argmax}_{j\in(i,\bar{i}_1,\dots\bar{i}_{k-1})} \|A_{:,j}\|_{\text{off-diag}}$
  **set** $S_\ell \leftarrow S_{\ell-1}\backslash\{j\}$
}
**Output**: $U_1, \dots, U_L$ and $H = A_L \downarrow \mathcal{H}^n_{S_L}$

**Proposition 4.** *Let $A \in \mathbb{R}^{2\times 2}$ be diagonal, $B \in \mathbb{R}^{2\times 2}$ symmetric and*

$$O = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix}.$$

*Set*

$$a = (A_{1,1} - A_{2,2})^2/4, \quad b = B_{1,2},$$

$$c = (B_{2,2} - B_{1,1})/2, \quad e = \sqrt{b^2 + c^2},$$

$$\theta = 2\alpha, \quad \omega = \arctan(c/b).$$

*Then if $\alpha$ minimizes $([OAO^\top]_{2,1})^2 + [OBO^\top]_{2,2}$, then $\theta$ satisfies the equation*

$$(a/e)\sin(2\theta) + \sin(\theta + \omega + \pi/2) = 0. \tag{5.22}$$

Putting $A$ and $B$ in diagonal form required by this proposition is easy. While (5.22) is still not an explicit expression for $\alpha$, it is trivial to solve with iterative methods.

Note that Jacobi MMF with $k = 2$ is very similar to the Treelets algorithm, which we discussed in Section 4.2. Thus, it is worth elaborating on the general differences between

MMF algorithms from the previous sections and Treelets.

- In Treelets the choice of $(i, j)$ coordinates to rotate and the rotation angle $\theta$ are chosen based on an analogy with Jacobi's algorithm. In contrast, in MMF they are optimized to reduce the algorithms objective function, which is approximation error.

- Similar to MMF, we can construct a rotation tree for the Treelets algorithm. However, the Treelets rotation tree has a very rigid structure — it is always binary, similar to the GreedyJacobiMMF rotation tree in Figure 5.3(a). Empirically, on real data the Treelets algorithm has a tendency to lead to what we call **cascades**, where a single coordinate is repeatedly rotated against multiple other coordinates at subsequent levels. The resulting rotation tree is not well balanced, but is degenerate in the sense that the longest path through the tree has length close to $n$ nodes. In practice, the rotation tree induced by Jacobi MMFs could also suffer from this cascade problem, which of course can ruin multiresolution. However, parallel and randomized MMFs avoid this problem altogether.

- All the MMFs we have described so far extend to $k \geq 3$, which enables us to find a more subtle lattice–like MMF rotation hierarchy of the coordinates than just a single rotation tree.

## 5.3 Theoretical Analysis

MMFs satisfy properties MRA2 and MRA3 of Section 2.7 by construction. Showing that they also satisfy MRA1 or some analog thereof requires, roughly, to prove that the smoother a function $f: X \to \mathbb{R}$ is, the smaller its high frequency wavelet coefficients are. For this purpose the usual notion of smoothness with respect to a metric $d$ is Hölder continuity, defined

$$\big|f(x) - f(y)\big| \leq c_H d(x, y)^\alpha \qquad \forall x, y \in X,$$

with $c_H$ and $\alpha > 0$ constant. In classical wavelet analysis one proves that the wavelet coefficients of $(c_H, \alpha)$–Hölder functions decay at a certain rate, for example, $\left|\left\langle f, \psi_\ell^m \right\rangle\right| \leq c' \ell^{\alpha+\beta}$ for some $\beta$ and $c'$ (Daubechies, 1992).

As we have seen, MMFs are driven by the similarity between the rows/columns of the matrix $A$. Therefore, relaxing the requirement that $d$ must be a metric, we define $d(i,j)$ in terms of the $\left\langle A_{i,:}, A_{j,:} \right\rangle$ inner products.

$$d(i,j) = \left|\left\langle A_{i,:}, A_{j,:} \right\rangle\right|^{-1}. \tag{5.23}$$

One must also make some assumptions about the structure of the underlying space, classically that $X$ is a so-called space of homogeneous type (Deng and Han, 2009), which means that for some constant $c_{\text{hom}}$,

$$\text{Vol}(B(x,2r)) \leq c_{\text{hom}} \, \text{Vol}(B(x,r)) \quad \forall x \in X, \ \forall r > 0.$$

To capture the analogous structural property for matrices, we introduce a concept of rank–homogeneous matrices, which has connections to the R.I.P condition in compressed sensing (Candès and Tao, 2005).

**Definition 8.** *We say that a symmetric matrix $A \in \mathbb{R}^{n \times n}$ is $\Lambda$–**rank homogeneous** up to order $\bar{K}$, if for any $S \subseteq [n]$ of size at most $\bar{K}$, letting $Q = A_{S,:}A_{:,S}$, setting $D$ to be the diagonal matrix with $D_{i,i} = \|Q_{i,:}\|_1$, and $\tilde{Q} = D^{-1/2}QD^{-1/2}$, the $\lambda_1, \lambda_1, \ldots, \lambda_{|S|}$ eigenvalues of $\tilde{Q}$ satisfy $\Lambda < |\lambda_i| < 1 - \Lambda$, and furthermore $c_T^{-1} \leq D_{i,i} \leq c_T$ for some constant $c_T$.*

Recall that the spectrum of the normalized adjacency matrix of a graph is bounded in $[-1, 1]$ (Chung, 1997). Definition 8 asserts that if we form a graph with vertex set $S$ and edge weights $\left\langle A_{i,:}, A_{j,:} \right\rangle$, its eigenvalues in absolute value are bounded away from both 0 and 1. Definition 8 then roughly corresponds to asserting that $A$ does not have clusters of rows that are either almost identical (parallel) (an incoherence condition) or completely unrelated

(orthogonal). This allows us to now state the matrix analog of the Hölder condition.

**Theorem 1.** *Let $A \in \mathbb{R}^{n\times n}$ be a symmetric matrix that is $\Lambda$–rank homogeneous up to order $\bar{K}$ and has an MMF factorization $A = U_1^\top \dots U_L^\top H U_L \dots U_1$. Assume $\psi_m^\ell$ is a wavelet in this factorization arising from row $i$ of $A_{\ell-1}$ supported on a set $S$ of size $K \le \bar{K}$ and that $\|H_{i,:}\|^2 \le \epsilon$. Then if $f:[n] \to \mathbb{R}$ is $(c_H, 1/2)$–Hölder with respect to 5.23, then its wavelet coefficients obey*

$$|\langle f, \psi_m^\ell \rangle| \le c_T \sqrt{c_H c_\Lambda} \epsilon^{1/2} K \tag{5.24}$$

*with $c_\Lambda = 4/(1-(1-2\Lambda)^2)$.*

Here $\epsilon$ is closely related to the MMF approximation error and is therefore expected to be small. Equation (5.24) then says that, as we expect, if $f$ is smooth, then its "high frequency" local wavelet coefficients (low $K$ and $\ell$) will be small.

## 5.4 Proofs of Propositions and Theorems

Here we provide proofs for the propositions and theorems introduced in previous sections.

**Proof of Proposition 1.** By the nestedness of $S_0^{\text{act}} \supseteq S_1^{\text{act}} \supseteq \dots \supseteq S_L^{\text{act}}$, for some sequence of permutation matrices $\Pi_1, \Pi_2, \dots, \Pi_L$, $H$ decomposes recursively as

$$[H]_{S_\ell^{\text{act}}, S_\ell^{\text{act}}} = \Pi_\ell \begin{pmatrix} [H]_{S_{\ell+1}^{\text{act}}, S_{\ell+1}^{\text{act}}} & [H]_{S_{\ell+1}^{\text{act}}, J_{\ell+1}} \\ [H]_{J_{\ell+1}, S_{\ell+1}^{\text{act}}} & [H]_{J_{\ell+1}, J_{\ell+1}} \end{pmatrix} \Pi_\ell^\top.$$

Unwrapping this recursion tells us that $\|H\|_{\text{residual}}^2$ is equal to

$$\sum_{\ell=1}^{L} \left[ \|H_{J_\ell, S_\ell^{\text{act}}}\|_{\text{Frob}}^2 + \|H_{S_\ell^{\text{act}}, J_\ell}\|_{\text{Frob}}^2 + \|H_{J_\ell, J_\ell}\|_{\text{off-diag}}^2 \right].$$

However, since the rotations $U_{\ell+1}, \dots, U_L$ leave $\text{span}(\{e_i | i \in [n] \backslash S_\ell^{\text{act}}\})$ invariant,

$$\|[A_\ell]_{J_\ell, S_\ell^{\text{act}}}\|_{\text{Frob}}^2 = \|[A_{\ell+1}]_{J_\ell, S_\ell^{\text{act}}}\|_{\text{Frob}}^2 = \dots = \|[A_L]_{J_\ell, S_\ell^{\text{act}}}\|_{\text{Frob}}^2 = \|[H]_{J_\ell, S_\ell^{\text{act}}}\|_{\text{Frob}}^2.$$

By symmetry, $\|[H]_{S_\ell^{\text{act}},J_\ell}\|_{\text{Frob}}^2 = \|[H]_{J_\ell,S_\ell^{\text{act}}}\|_{\text{Frob}}^2$. Similarly, $\|[A_\ell]_{J_\ell,J_\ell}\|_{\text{off-diag}}^2 = \dots$ $= \|[H]_{J_\ell,J_\ell}\|_{\text{off-diag}}^2$.

**Proof of Proposition 2.** Since $J = \{i_k\}$, by Proposition 1

$$\mathcal{E}_\ell = 2\sum_{p=1}^{k-1}[U_\ell A_{\ell-1}U_\ell^\top]_{i_k,i_p}^2 + 2\|[U_\ell A_{\ell-1}U_\ell^\top]_{i_k,S_\ell^{\text{act}}}\|^2.$$

The first term can be written $2\sum_{p=1}^{k-1}[O[A_{\ell-1}]_{I,I}O^\top]_{k,p}^2$, while the second term is

$$\begin{aligned}
&2\|[O[A_{\ell-1}]_{I,S_\ell^{\text{act}}}[U_\ell]_{S_\ell^{\text{act}},S_\ell^{\text{act}}}^\top]_{k,:}\|^2 \\
&= 2\left[O[A_{\ell-1}]_{I,S_\ell^{\text{act}}}[U_\ell]_{S_\ell^{\text{act}},S_\ell^{\text{act}}}^\top[U_\ell]_{S_\ell^{\text{act}},S_\ell^{\text{act}}}[A_{\ell-1}]_{I,S_\ell^{\text{act}}}^\top O^\top\right]_{k,k} \\
&= 2[O[A_{\ell-1}]_{I,S_\ell^{\text{act}}}[A_{\ell-1}]_{I,S_\ell^{\text{act}}}^\top O^\top]_{k,k} = 2[OBO^\top]_{k,k}.
\end{aligned}$$

**Proof of Proposition 3.** Analogous to the proof of Proposition 2, but summed over each $I_1 \times I_1, \dots, I_m \times I_m$ block.

**Proof of Proposition 4.** We want to minimize

$$\phi(\alpha) = \left(\left[O_\alpha\begin{pmatrix} A_1 & 0 \\ 0 & A_2\end{pmatrix}O_\alpha^\top\right]_{2,1}\right)^2 + \left[O_\alpha\begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2}\end{pmatrix}O_\alpha^\top\right]_{2,2}.$$

Expanding, we get

$\phi(\alpha) = ((A_1 - A_2)\sin\alpha\cos\alpha)^2 + B_{1,1}(\sin\alpha)^2 + 2B_{1,2}\sin\alpha\cos\alpha + B_{2,2}(\cos\alpha)^2 =$
$= \left(\frac{A_1 - A_2'}{2}\right)^2(\sin(2\alpha'))^2 + B_{1,2}\sin(2\alpha) + (\sin\alpha')^2B_{1,1} + (\cos\alpha')^2B_{2,2}.$
Rewriting the second two terms as

$$\frac{((\sin\alpha)^2 + (\cos\alpha')^2)(B_{1,1} + B_{2,2})}{2} + \frac{((\sin\alpha)^2 - (\cos\alpha')^2)(B_{1,1} - B_{2,2})}{2}$$

gives

$$\phi(\alpha) = \left(\frac{A_1 - A_2}{2}\right)^2 (\sin(2\alpha))^2 + B_{1,2}\sin(2\alpha) + \frac{B_{1,1} + B_{2,2}}{2} + \frac{B_{2,2} - B_{1,1}}{2}\cos(2\alpha).$$

Introducing $d = (B_{2,2} - B_{1,1})/2$ and the other variables $a, b, c, e$ and $\theta$ gives the new objective function

$$\psi(\theta) = a(\sin\theta)^2 + b\sin\theta + c\cos\theta + d.$$

Setting the derivative with respect to $\theta$ zero,

$$2a\sin\theta\cos\theta + b\cos\theta - c\sin\theta = 0.$$

Again using $\sin(2x) = 2\sin x\cos x$,

$$a\sin(2\theta) + b\cos\theta - c\sin\theta = 0.$$

Now letting $e = \sqrt{b^2 + c^2}$ and $\omega = \arctan(c/b)$

$$a\sin(2\theta) + e(\cos\omega\cos\theta - \sin\omega\sin\theta) = 0.$$

Using $\cos(x + y) = \cos x\cos y - \sin x\sin y$,

$$(a/e)\sin(2\theta) + \cos(\theta + \omega) = 0,$$

which is finally equivalent to (5.22).

**Proof of Theorem** 1. Let $\psi$ be a specific wavelet $\psi_m^\ell$, with support $S = \{s_1, s_2, \ldots, s_K\} = \mathrm{supp}(\psi) \subseteq [n]$, $f_S$ and $\psi_S$ be the restriction of $f$ and $\psi$ to $S$ regarded as a vectors, and $Q, D$

and $\tilde{Q}$ be defined as in Definition 8. The Hölder property then gives

$$f_S^\top \tilde{L} f_S = \sum_{i,j=1}^{K} \tilde{Q}_{i,j}(f(s_i) - f(s_j))^2 \le \sum_{i,j=1}^{K} c_T Q_{i,j}(f(s_i) - f(s_j))^2 \le c_T c_H K^2, \tag{5.25}$$

where $\tilde{L} = I - \tilde{Q}$ is the normalized Laplacian. At the same time, if $\psi_m^\ell$ comes from row/column $i$ of $A_\ell$, then by (5.6), $[A_\ell]_{:,i} = U_\ell \dots U_1 A\psi$, and therefore

$$\psi_S^\top \tilde{Q} \psi_S \le c_T \psi_S^\top Q \psi_S \le c_T \psi_S^\top A_{S,:} A_{:,S} \psi_S = c_T \|A\psi\|^2 = c_T \|[A_\ell]_{:,i}\|^2 = c_T \|H_{:,i}\|^2 \le c_T \epsilon \tag{5.26}$$

Clearly, $\tilde{Q}$ and $\tilde{L}$ share the same normalized eigenbasis $\{v_1, v_2, \dots, v_n\}$. Letting $\{\lambda_1, \lambda_2, \dots, \lambda_n\}$ be the corresponding eigenvalues, $f_i = \langle f_S, v_i \rangle$ and $\psi_i = \langle \psi_S, v_i \rangle$ and taking any $\gamma > 0$

$$\sum_{i=1}^{K} \left( \sqrt{\gamma \lambda_i} \psi_i - \frac{1}{\sqrt{\gamma \lambda_i}} f_i \right)^2 \ge 0, \tag{5.27}$$

which implies

$$\langle f, \psi \rangle = \langle f_S, \psi_S \rangle \le \frac{1}{2} \left[ \gamma \psi_S^\top \tilde{Q} \psi_S + \gamma^{1/2} f_S^\top \tilde{Q}^{-1} f_S \right].$$

The first term on the r.h.s of this inequality is bounded by (5.26), while by (5.25) for any $c_\Lambda \ge 4/(1 - (1 - 2\Lambda)^2)$,

$$f_S^\top \tilde{Q}^{-1} f_S = \sum_{i=1}^{K} \frac{1}{\lambda_i} f_i^2 \le c_\Lambda \sum_{i=1}^{K} (1 - \lambda_i) f_i^2 = \; = c_\Lambda f_S^\top \tilde{L} f_S \le c_T c_H c_\Lambda K^2$$

giving $\langle f, \psi \rangle \le c_T(\gamma \epsilon + \gamma^{-1} c_H c_\Lambda K^2)$. Optimizing this for $\gamma$ yields $\langle f, \psi \rangle \le c_T \sqrt{c_H c_\Lambda} \epsilon^{1/2} K$. By flipping the $-$ sign in (5.27) to $+$, a similar lower bound can be derived for $-\langle f, \psi \rangle$.

## 5.5 Applications of MMF

MMF has a variety of potential applications which we briefly review below. This thesis dives deep into the last application of MMF — as a matrix compression tool — which is the focus

of the following chapter.

- **Sparse approximations**: Exploring sparsity, both theoretically and empirically, is one of the key developments in machine learning in recent years. This trend is triggered largely by the continuously growing volume of real datasets and the need for computational efficiency and scalability of traditional machine learning algorithms. Constructing dictionaries or a hierarchically sparse basis, as we saw in Section 4.5, are crucial for the sparse representation of signals. In this respect MMF can be used to produce a data adapted sparse wavelet basis for sparse approximations on graphs or other datasets.

- **Clustering**: MMF can be used for hierarchical clustering, similarly to the way the Treelets algorithm recovers a binary tree on the coordinates of covariance matrices. Since MMF recovers a more fine tuned hierarchical structure, namely one consisting not of a single hierarchical tree, but a lattice–like hierarchy of multiple interleaving trees. As a result, MMF is more well suited for capturing the higher order dependencies between variables in covariance matrices or vertices in a graph Laplacian, for example.

- **Community structure**: One of the main reasons uncovering community structure in graphs, be it biological, social or physical networks, is quite challenging is the fact that communities are present at different scales and form overlapping hierarchies (see (Fortunato, 2010) for review of community detection algorithms). Thus, in order to determine the best community structure (by some metric) in a computationally efficient way, one needs to find a judicious way of finding these hierarchies. When applied to the (normalized) graph Laplacian, MMF performs very localized (binary or of higher order) rotations on the matrix and finds multiple overlapping hierarchies of the vertices. Hence, it is particularly well suited for the task of finding communities in graphs.

- **Matrix compression**: As shown in Figure 5.1, if $A$ is an operator, MMF provides a computationally efficient way of compressing $A$ to size $\delta_L \times \delta_L$ (plus additional $n - \delta_L$

diagonal elements) and thus, provides a way of efficiently applying the operator $A$ to matrices and vectors in downstream applications. Chapter 6 provides an overview of compression for matrices and large scale data and empirically demonstrates that MMF can be used for compression by comparing MMF to other compression schemes on several datasets.

It is important to note that depending on the application, one of the MMF algorithms introduced above might be more appropriate than the others. Each MMF algorithm is intended to be used for a different regime, determined by the application and the efficiency requirements. For example, while a randomized MMF might be more appropriate for fast matrix compression, a parallel MMF might produce a better sparse wavelet basis. In the following section we show various experiments demonstrating how the various MMF algorithms can be used for the applications listed above.

## 5.6 Experiments

We measure the Frobenius norm error $\mathcal{E}_{\text{Frob}}$ incurred by MMF by summing the $\ell^2$ norm of the rows/columns (except for their diagonal elements) that are designated wavelets at each $\ell$ level of the factorization. According to Proposition 1, this is equivalent to the sum

$$\mathcal{E}_{\text{Frob}} = \mathcal{E} = \sum_{\ell=1}^{L} \mathcal{E}_\ell = \|A - U_1^\top \dots U_L^\top H U_L \dots U_1\|_{\text{Frob}} = \|A - \tilde{A}\|_{\text{Frob}}, \tag{5.28}$$

where we introduce the shorthand $U_1^\top \dots U_L^\top H U_L \dots U_1 = \tilde{A}$. Recall the error $\mathcal{E}_\ell$ incurred at each level is computed differently depending on whether the rotations involved are elementary or compound (see Propositions 2 and 3). In all the experiments in this chapter the error is normalized in the form $\mathcal{E}_{\text{Frob}}/\|A\|_{\text{Frob}}$.

In some of the experiments, when necessary, the MMF wavelets are typically ordered in increasing frequency. Algorithms 1 and 3 provide a natural ordering as in each subsequent level they split off a single wavelet, with each wavelet being increasingly more global. How-

ever, Algorithm 2, by virtue of its parallelism, recovers multiple wavelets at a given level and provides no explicit ordering of the wavelets within a level. So whenever we need to order the wavelets recovered by Algorithm 2, we order them level by level such that the wavelets within each level are ordered in increasing order of the amount of mass they bring to the diagonal of $A$.

### *5.6.1 Comparison to Treelets*

We evaluate the performance of Jacobi MMFs (Algorithm 1) by comparing it with Treelets on two real datasets. Note that in the greedy binary setting, similarly to the Treelets algorithm, MMF removes one dimension at a time, and thus, in both algorithms the off-diagonal part of the rows/columns designated as wavelets contributes to the error $\mathcal{E}_{\text{Frob}}$. The first matrix is based on the well known Zachary's Karate Club (Zachary, 1977) social network consisting of 34 vertices and 78 edges. We set $A$ to be the diffusion kernel $A = e^{-\alpha\mathcal{L}}$ with $\alpha = 0.01$. The second matrix is constructed using simulated data from family pedigrees, as described by Crossett et al. (2013) — $A \in \mathbb{R}^{50\times 50}$ is symmetric matrix in which each $A_{i,j}$ element contains the genetic kinship coefficient between individual $i$ and $j$. Figure 5.4 shows that GreedyJacobiMMF outperforms Treelets for a wide range of compression ratios for both datasets, albeit after a certain level of compression (i.e., when $|S_\ell|$ decreases sufficiently in the karate club case), it is sometimes prone to higher errors in comparison to Treelets. The better approximation quality of MMF, as measured by $\mathcal{E}_{\text{Frob}}$, is not surprising since the Treelets algorithm, unlike MMFs, is not optimized to reduce the approximation error of the decomposition $A \approx U_1^\top U_2^\top \dots U_L^\top H U_L \dots U_2 U_1$.

### *5.6.2 Comparison of MMF Algorithms*

As highlighted in Section 5.2.2, finding the optimal compound rotations in binary parallel MMFs is equivalent to finding the optimal partition $I_1 \uplus I_2 \uplus \cdots \uplus I_m$ of the set $[n]$ by

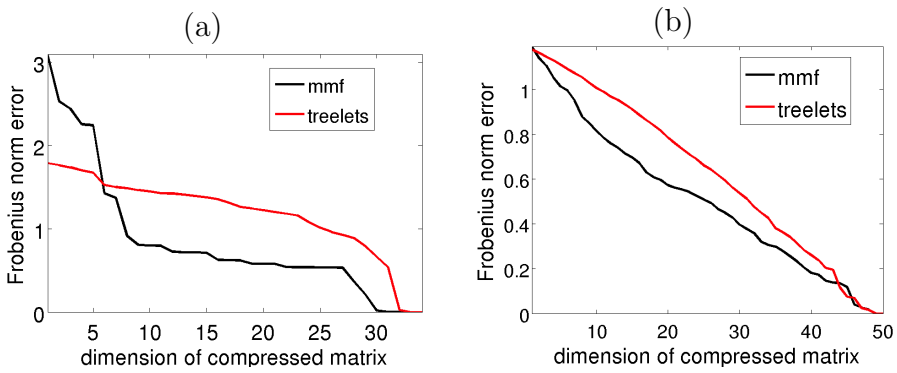



Figure 5.4: **Comparison with Treelets**. Frobenius norm error $\mathcal{E}_{\text{Frob}}$ of compressing matrices with GreedyJacobiMMF (with $k = 2$) vs. the Treelets algorithm as a function of the dimension of the core $H_{S_L,S_L}$ that $A$ is compressed down to. (a) $A$ is constructed from Zachary's Karate Club graph (Zachary, 1977), as described in the text. (b) $A$ is a genetic relationships matrix.

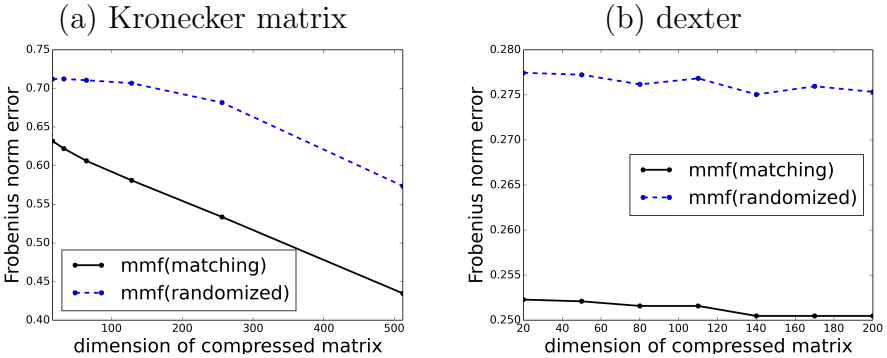



Figure 5.5: **GreeyParallelMMF vs. RandomizedMMF**. Frobenius norm error $\mathcal{E}_{\text{Frob}}$ of compressing matrices with GreedyParallelMMF vs. RandomizedMMF as a function of the dimension of the core $H_{S_L,S_L}$ that $A$ is compressed down to. For both algorithms the rotation order is $k = 2$. (a) $A$ is a $1024 \times 1024$ Kronecker matrix, constructed as described in the text. (b) $A$ is the dexter dataset from Table 6.1.

applying a graph matching algorithm, such as the Blossom Algorithm (Edmonds, 1965). However, since the computational cost of finding the optimal matching is prohibitively high, the randomized version of MMF (Algorithm 3) is often preferred in practice. As Figure 5.5 demonstrates, the approximation error incurred by binary parallel MMF is comparable to the error incurred by the randomized MMF. Hence, the randomized MMF can be used in place of the exact MMFs when the matrix size $n$ becomes too large.

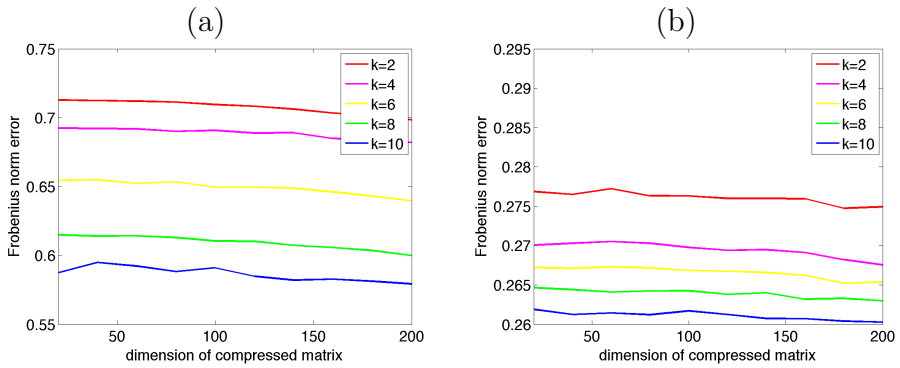



Figure 5.6: **MMF rotation order**. Frobenius norm error $\mathcal{E}_{\text{Frob}}$ of compressing matrices with the RandomizedMMF with different orders $k$, as a function of the dimension of the core $H_{S_L,S_L}$ that $A$ is compressed down to. (a) $A$ is a $1024 \times 1024$ Kronecker matrix, constructed as described in the text. (b) $A$ is the dexter dataset from Table 6.1.

### *5.6.3 Effect of MMF Rotation Order*

The randomized MMFs can be further refined using rotations of order higher than $k = 2$. Figure 5.6 shows a comparison of randomized MMF (Algorithm 3) with $k = 2$ vs. $k \geq 3$. Recall that for $k = 2$, MMF performs Givens rotations, computable in closed form. When $k \geq 3$, on the other hand, MMF involves the diagonalization of the matrix $B$ in Proposition 2, which does not have a closed form solution but can be computed by eigedecompisition instead. Accordingly, the repeated calls to an eigendecomposition routine inside MMF itself lead to slower wall clock time of the randomized MMF. Thus, MMFs with binary rotations are preferable whenever possible.

Regardless of the levels of compression, as the rotation order $k$ increases, the MMF approximation error $\mathcal{E}_{\text{Frob}}$ goes down on both synthetic and real data (Figure 5.6). This is not surprising because as the order of the rotations increases MMF becomes more and more similar a matrix diagonalization procedure. In fact, an MMF with rotations order $k = n$ is equivalent to a performing a complete diagonalization of $A \in \mathbb{R}^{n \times n}$. More importantly, however, the approximation error of the binary randomized MMF is still comparable to rotations of order $k = 10$. This shows that binary MMFs are a viable alternative to higher order MMFs.

### 5.6.4 Recovering Matrix Structure with MMF

As illustrated in Figure 5.3, the MMF rotation hierarchy can take different forms depending on the type of MMF. Figure 5.7 shows that the rotation tree of the GreedyParallelMMF algorithm can indeed recover the hierarchical structure of the underlying matrix. In this experiment $A$ is the graph Laplacian of a Kronecker graph model, which has recursive self–similar/fractal structure.

The Kronecker product graph $K_{1,\kappa}$ on $K_1^\kappa$ nodes is defined as

$$K_{1,\kappa} = \underbrace{K_1 \otimes K_1 \otimes \cdots \otimes K_1}_{\kappa \text{ times}} = K_{\kappa-1} \otimes K_1, \tag{5.29}$$

where $K_1$ is the so-called initiator adjacency matrix of probabilities. The recursive construction of Kronecker product graphs is based on the assumption that communities in the graph grow recursively as well — nodes in a community are recursively expanded into miniature copies of the community itself (Leskovec et al., 2010).

In Figures 5.7(a) and (b) we set $\kappa = 10$ and 4, respectively, while the initiator matrix in both cases is $K_1 = [1, 0.1; 0.1, 1]$. We modify definition (5.29) to $K_{1,\kappa} = K_{\kappa-1} \otimes (K_1 + [0, 0.1; 0.1, 0])$ — note that increasing the off-diagonal entries of the initiator matrix in each recursive step does not change the the overall fractal structure of $K_{1,\kappa}$, but instead makes it even more fine grained.

Figure 5.7 shows that approximating $A = K_{1,10}$ by GreedyParallelMMF not only preserves the structure of $A$ but, more importantly, MMF can recover the structure of the Kronecker product graph itself.

Next, we shuffled $A$ by permuting its columns/rows uniformly at random. The shuffled matrix is denoted by $A_{p,p}$, where $p$ is a random permutation of the set $[n]$. The resulting matrix $A_{p,p}$ is shown in the third column in Figure 5.7. We ran MMF on $A_{p,p}$ and reordered the rows/columns of $A_{p,p}$ according to the MMF rotation hierarchy obtained by running GreedyParallelMMF on $A_{p,p}$. In particular, we order the leaves of the rotation tree by

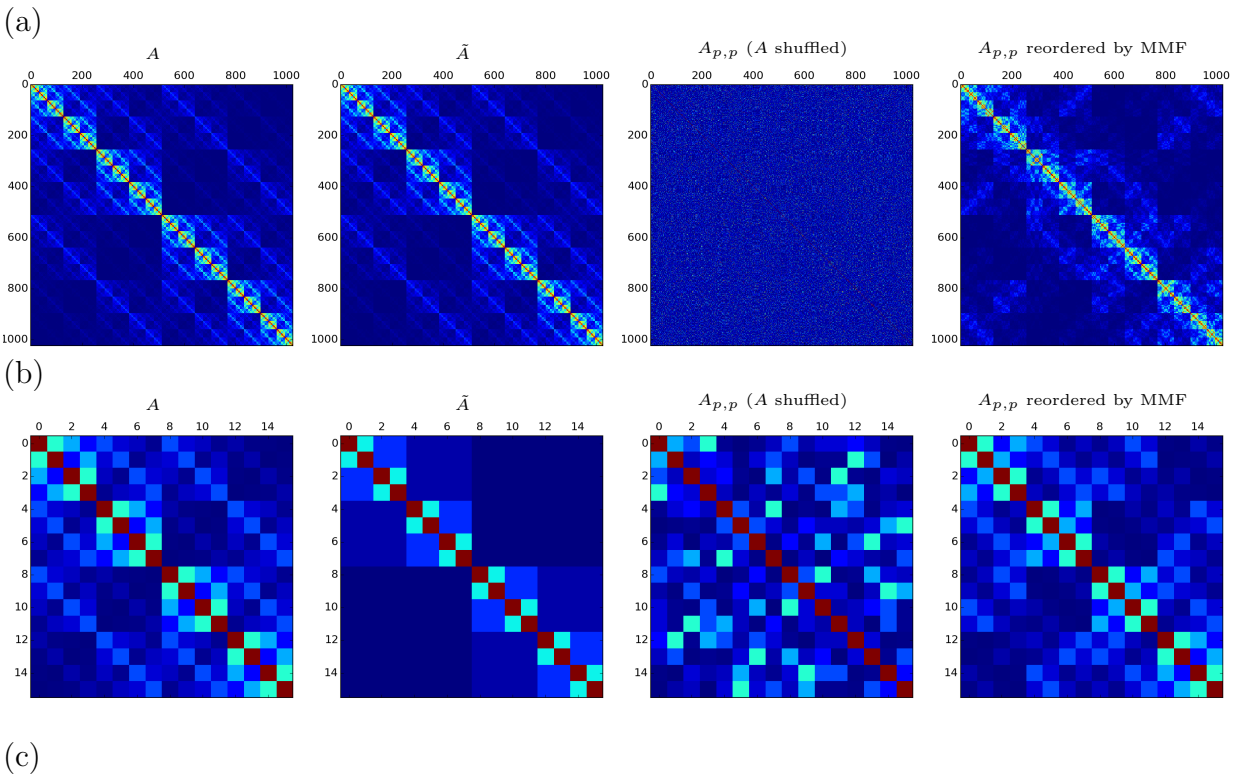
depth first search traversal of the tree. Recall that all of the matrix row/column indices lie on the leaves of the rotation tree and the intermediary nodes denote the $U_\ell$ rotations. Note that the depth first search ordering is not unique due to the so-called "sibling exchange invariance" (each node has two children, but the left/right child order is irrelevant as rotations $\oplus_{(d_i,d_j)}O$ and $\oplus_{(d_j,d_i)}O$ are identical for the purposes of MMF).

Additionally, for the smaller of the two matrices in Figure 5.7(c) we show the rotation tree associated with $\tilde{A}$. Each pair of indices $(i,j)$ in the tree corresponds to $\oplus_{(d_i,d_j)}O$ (recall discussion of MMF rotation hierarchies in Section 5.1). One possible leaf ordering for the tree shown in the figure is $[d_{12}, d_{13}, d_{14}, d_{15}, d_4, d_5, d_6, d_7, d_{10}, d_{11}, d_8, d_9, d_0, d_1, d_2, d_3]$.

When the rows/columns of $A_{p,p}$ are reordered according to the MMF rotation tree, the implicit fractal structure of the shuffled matrix $A_{p,p}$ becomes apparent (see the plots in the fourth column of Figure 5.7). In this sense, MMF "discovers" the underlying fractal structure regardless of the fact that the matrix $A_{p,p}$ on which MMF was performed was shuffled and had no obvious structure.

### *5.6.5 Comparison of MMF and PCA*

Since MMF wavelets can be interpreted as a hierarchical basis, it is natural to compare the MMF basis with the basis recovered by PCA (5.2). We consider a dataset consisting of 200 points in $\mathbb{R}^2$ which is the union of 50 realizations of four Gaussian random variables sampled from normal distribution with means $[1.4, 0.6], [0.6, 1.4], [-2, -2], [-2.8, -2.8]$ and standard deviations all equal to 0.25. The resulting graph, plotted in Figure 5.8(c), has a hierarchical, nested, clusters–of–clusters structure in the sense the each of the two pairs of Gaussian clusters can be interpreted as part of a single Gaussian cluster with larger standard deviation. From this data we construct a pairwise distance matrix $A$, such that for each $(i,j)$ pair of points $A_{i,j} = e^{-\|x_i - x_j\|^2/0.5^2}$. We perform second order parallel MMF (Algorithm 2) on $A$ and plot the resulting top MMF wavelets in Figure 5.8(a). Figure 5.8(b), on the other



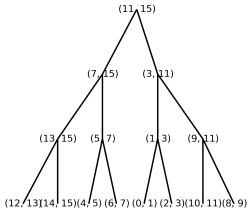
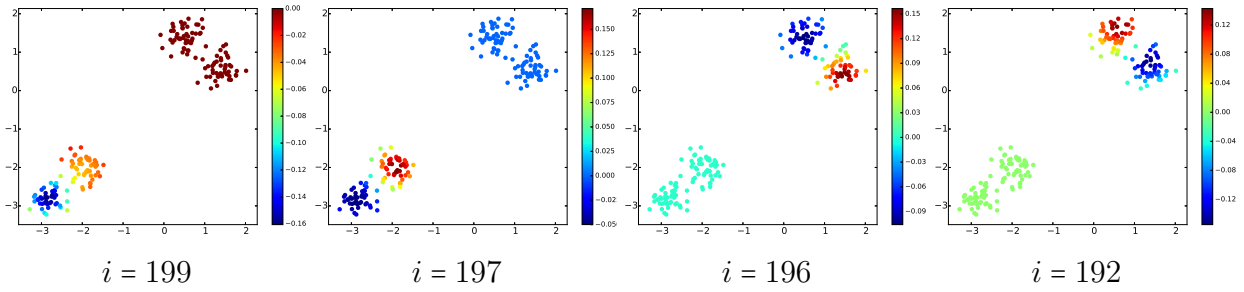



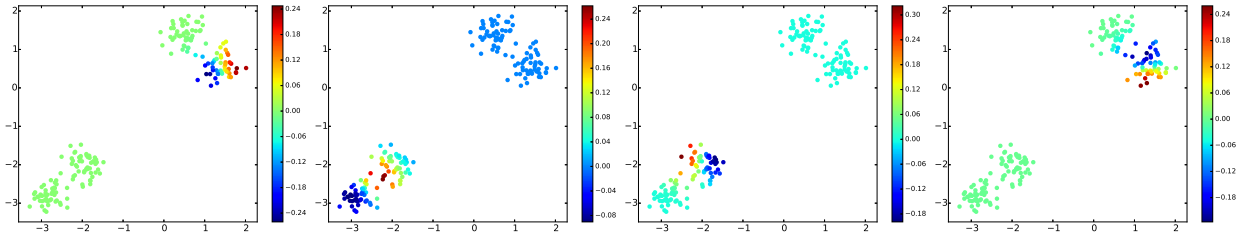
Figure 5.7: **MMF on structured matrices**. Two Kronecker product matrices, denoted by $A$, of different sizes and their approximations, denoted by $\tilde{A}$, by second order GreedyParallelMMF. The third column shows the shuffled matrix $A_{[p,p]}$. The fourth column shows $A_{[p,p]}$ reordered according to the MMF rotation tree. For the smaller matrix, in (c), we also show the MMF rotation tree $\tilde{A}$.

hand, shows the top eigenvectors of $A$. The eigenvectors are ordered in increasing order of their corresponding eigenvalues and so, the eigenvector with the highest eigenvalue is the one corresponding to index $i = 199$. The MMF wavelets, on the other hand, are ordered in increasing level, as described in the very beginning of Section 5.6. Note that MMF is not only able to approximate the top few eigenvectors, but even the more localized, higher frequency MMF wavelets (e.g., wavelets with indices $i = 186, 183$, and even the one with $i = 50$) are able to recover local structure in the graph. On the other hand, the support of

(a) MMF wavelets

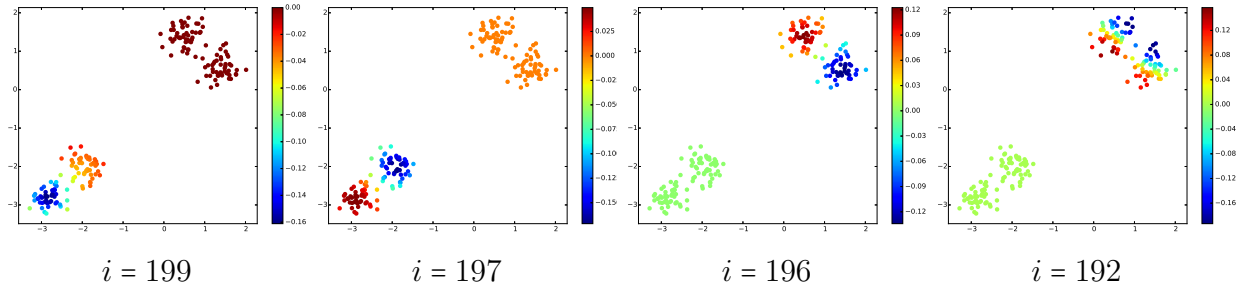


$i = 199$ $i = 197$ $i = 196$ $i = 192$

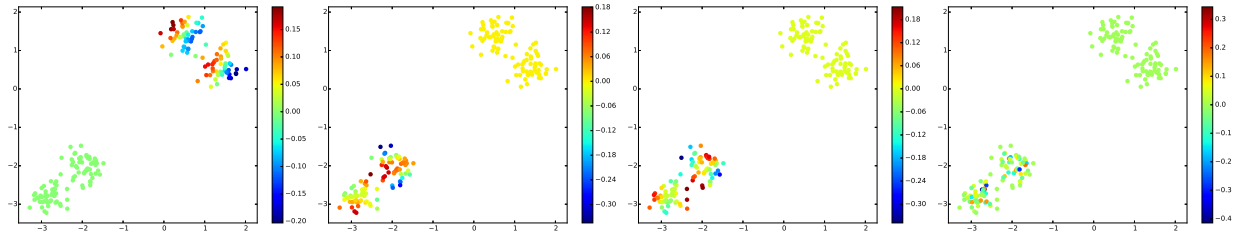


$i = 190$ $i = 186$ $i = 183$ $i = 50$

(b) PCA eigenvectors



$i = 199$ $i = 197$ $i = 196$ $i = 192$



$i = 190$ $i = 186$ $i = 183$ $i = 50$

(c) dataset

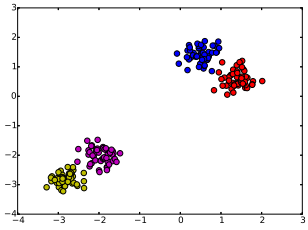


Figure 5.8: **MMF wavelets vs. eigenvectors**. (a) Wavelets recovered by second order parallel MMF on the distance matrix of the dataset of Gaussian random variables, shown in (c) and described in Section 5.6.5. (b) The eigenvectors obtained by PCA on the same matrix. In all plots the graph vertices are colored according to the MMF wavelets or the PCA eigenvectors and the index $i$ corresponds to the $i$–th wavelet/eigenvector.

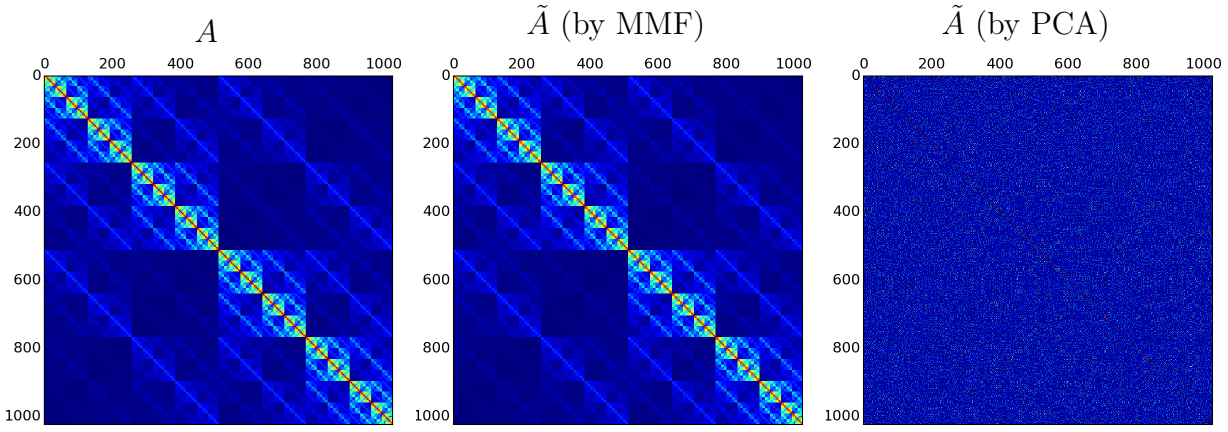



Figure 5.9: **Matrix reconstruction by MMF vs. PCA**. Approximation $\tilde{A}$ of a Kronecker product matrix $A$ by GreedyParallelMMF and PCA.

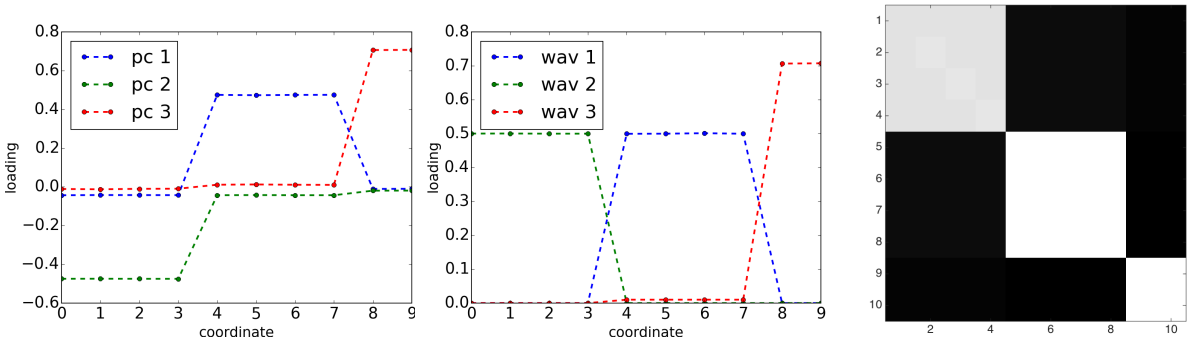
the corresponding eigenvectors is not well localized in the vertex domain of the graph.

We also compare how well MMF can approximate a matrix in comparison to PCA and show the matrix reconstruction results in Figure 5.9. In this experiment use the same modified Kronecker matrix $A = K_{1,\kappa}$ with $\kappa = 10$ that we described in Section 5.6.4. The resulting matrix is of size $1024 \times 1024$. In the MMF case it is compressed down to core size $n/4 \times n/4$. Accordingly, the rank parameter in PCA (5.2) was set to $r = n/4$. The reconstruction of $A$ by MMF is astonishingly good, while the PCA reconstruction has none of the fractal structure present in $A$. Qualitatively, MMF outperforms PCA in terms of the amount of detail it preserves, even when the dimension of core is just one quarter of the dimension $A$.

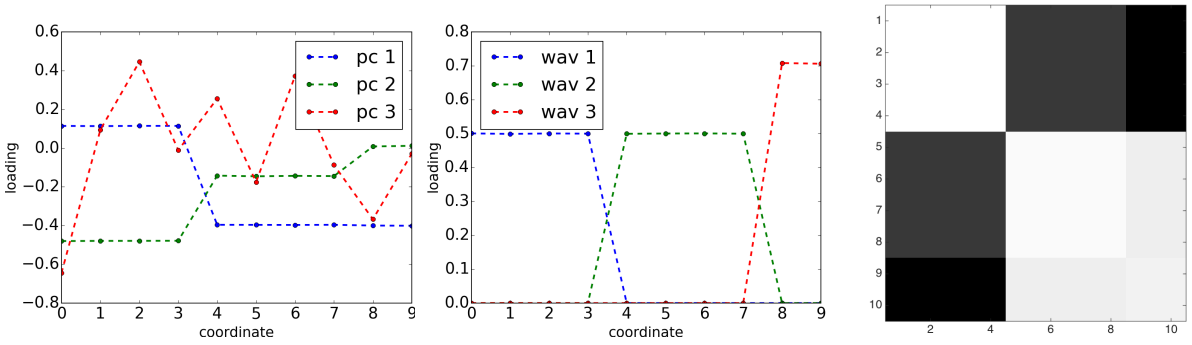
### *5.6.6 MMF on Mixture Models*

Next, we generate several synthetic datasets using linear mixture models, which in certain cases gives rise to covariance matrices with block structure, and investigate the effect the underlying covariance matrix structure has on the MMF wavelets. For this set of experiments we closely follow the examples presented by Lee et al. (2008). Consider a linear mixture model with $K$ components which has additive noise — each multivariate observation $x \in \mathbb{R}^p$

(a) uncorrelated factors, nonoverlapping loading vectors




(b) correlated factors, nonoverlapping loading vectors




(c) uncorrelated factors, overlapping loading vectors

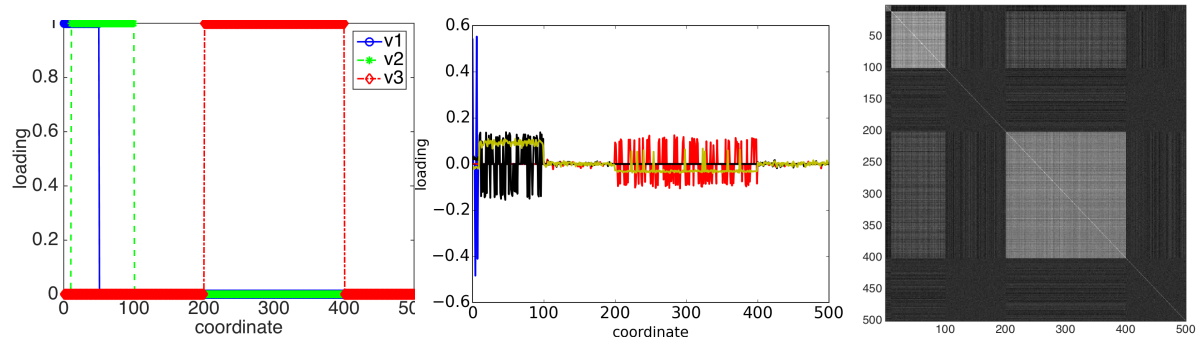



Figure 5.10: **MMF on mixture models**. (a) and (b) Comparison of principle components (left) and MMF wavelets (middle) on two different mixture models with their corresponding covariance matrices (right). (c) Loading vectors for the mixture model (left), the corresponding covariance matrix (left) and some of the recovered MMF wavelets (middle), shown in different colors. This set of experiments was performed using binary GreedyParallelMMF.

has the form

$$x = \sum_{i=1}^{K} u_i v_i + \sigma z, \tag{5.30}$$

where the so-called loading vectors $v_i \in \mathbb{R}^p$ are linearly independent, the components/factors $u_i$ are random (though not necessarily independent) variables with variance $\sigma_i^2$, $\sigma$ is the noise level and $z$ is a $p$–dimensional random vector whose entries are $\mathcal{N}(0,1)$. In the unsupervised setting we are given $n$ observations $x_1, x_2, \ldots, x_n \in \mathbb{R}^p$ sampled according to (5.30) and the goal is to infer the number of components in the mixture ($K$) or the structure of the loading vectors $v_i$, which are unknown. Inferring the structure of $v_i$ can be complicated by the correlations between the components $u_1, \ldots, u_K$, by their variance or by an overlap between the loading vectors $v_1, \ldots, v_K$. Borrowing the biological example by Lee et al. (2008), if $x \in \mathbb{R}^p$ is the measured expression levels of $p$ genes in a sample, $u_i$ can be thought of as the intrinsic activity of pathway $i$ and each binary vector $v_i$ marks the sets of genes belonging to that pathway. Lee et al. (2008) present three mixture models of increasing difficulty.

In each of the three cases we run MMF or PCA on the covariance matrix of the mixture model.

(a) **Uncorrelated factors and nonoverlapping loading vectors**. We sample $n = 1000$ points in $\mathbb{R}^{10}$ from the mixture model (5.30) with nonoverlapping loading vectors and factors, respectively,

$$\begin{aligned} v_1 &= [1,1,1,1,0,0,0,0,0,0]^\top, \\ v_2 &= [0,0,0,0,1,1,1,1,0,0]^\top, \\ v_3 &= [0,0,0,0,0,0,0,0,1,1]^\top, \end{aligned} \tag{5.31}$$

$$u_1 \sim \mathcal{N}(0,\sigma_1^2), \qquad u_2 \sim \mathcal{N}(0,\sigma_2^2), \qquad u_2 \sim \mathcal{N}(0,\sigma_3^2).$$

We set $\sigma_1^2 = 290$, $\sigma_2^2 = 300$, $\sigma_3^2 = 310$. The resulting population covariance matrix is $\Sigma = C + \delta^2 I_p$, where $C$ is a block diagonal matrix. We set $\sigma = 1$, resulting in

$\sigma_1, \sigma_2, \sigma_3 \gg \sigma$. As Figure 5.10(a) shows, both PCA and MMF are able to recover the three hidden loading vectors. This is a trivial example for PCA as the three loading vectors coincide exactly with the three leading principal components.

(b) **Correlated factors and nonoverlapping loading vectors**. We sample $n = 1000$ points in $\mathbb{R}^{10}$ from the mixture model (5.30) with the nonoverlapping loading vectors given in (5.31) and the factors

$$u_1 \sim \mathcal{N}(0, \sigma_1^2), \qquad u_2 \sim \mathcal{N}(0, \sigma_2^2), \qquad u_3 = \alpha_1 u_1 + \alpha_2 u_2.$$

The noise, variance, $\alpha_1$ and $\alpha_2$ parameters are set according to (Lee et al., 2008; Lee, 2006), again resulting in $\sigma_1, \sigma_2, \sigma_3 \gg \sigma$. The covariance matrix $\Sigma \in \mathbb{R}^{10\times 10}$ is no longer block diagonal, as in the model shown in (a), but has a more complex block structure shown in Figure 5.10(b). As a result of the correlation between the factors $u_i$, the loading vectors of the block model do not coincide with the principle eigenvectors of $\Sigma$ and therefore PCA cannot be used to infer the structure of the loading vectors. For example, figuring out the structure of the loading vectors from the top three principal components (Figure 5.10(b), left) is difficult. In particular, the third principal vector is particularly sensitive to the correlations. Even sparse PCA methods (Zou et al., 2006; Jenatton et al., 2010) fail to find the correct structure, unless the number of variables in each block is specified. In contrast, by performing binary parallel MMF (Algorithm 2) on $\Sigma$ we are able to find the underlying structure of the three loading vectors in this unfavorable for (sparse) PCA scenario (Figure 5.10(b), middle). This example shows that, while a global approach, such as PCA, has limitations even when $n > p$, MMF is able to find an adequate basis for this model.

(c) **Uncorrelated factors and overlapping loading vectors**. In the final and most challenging example the loading vectors $v_1$ and $v_2$ overlap, the background noise level is high and $n < p$ (specifically, $n = 100$, $p = 500$). In particular, the loading vectors are

defined as

$$v_1 = I(B_1) + I(B_2), \qquad v_2 = I(B_2) + I(B_3), \qquad v_3 = I(B_4),$$

where $I(B)$ is the indicator function defined on the set $B$ and

$$\begin{aligned} B_1 &= \{0, 1, \ldots, 9\}, \\ B_2 &= \{10, 11, \ldots, 49\}, \\ B_3 &= \{50, 51, \ldots, 99\}, \\ B_4 &= \{200, 201, \ldots, 399\}. \end{aligned}$$

The resulting loading vectors, some of which overlap in the coordinate block $B_2$, are plotted in Figure 5.10(c). The factors are given by $u_1 = \pm 0.5$ with equal probability, $u_2 = I(x < 0.4)$, and $u_3 = I(x < 0.3)$, where $I(.)$ is the indicator function and $x$ is an independent uniform random variables in $[0, 1]$. The block structure of the resulting covariance matrix $\Sigma^{500\times500}$ is shown in Figure 5.10(c). When set as described, the factors $u_1, u_2, u_3$ have variances $0.25, 0.24, 0.21$, respectively. The noise level in (5.30) is set to $\sigma = 0.5$ — note that this means the noise variance, $\sigma^2 = 0.25$, is on the same order as the variance of each of the three factors $u_i$, which makes the inference task particularly hard. Here binary parallel MMF is again able to infer the structure of the loading vectors. We ran Algorithm 2 on $\Sigma$ until it is compressed down to a $2 \times 2$ core and the middle panel of Figure 5.10(c) shows some of the MMF wavelets. The MMF wavelet shown in red is supported on the set $B_4$, which is exactly the support of the loading vector $v_3$. The MMF wavelets shown in black and blue, on the other hand, can distinguish between blocks $B_1, B_2$ and $B_3$, i.e., the support of vectors $v_1$ and $v_2$. The more global wavelet (shown in yellow) is approximately piecewise constant on all four blocks and can distinguish between $v_2$ (the wavelet values are positive on the

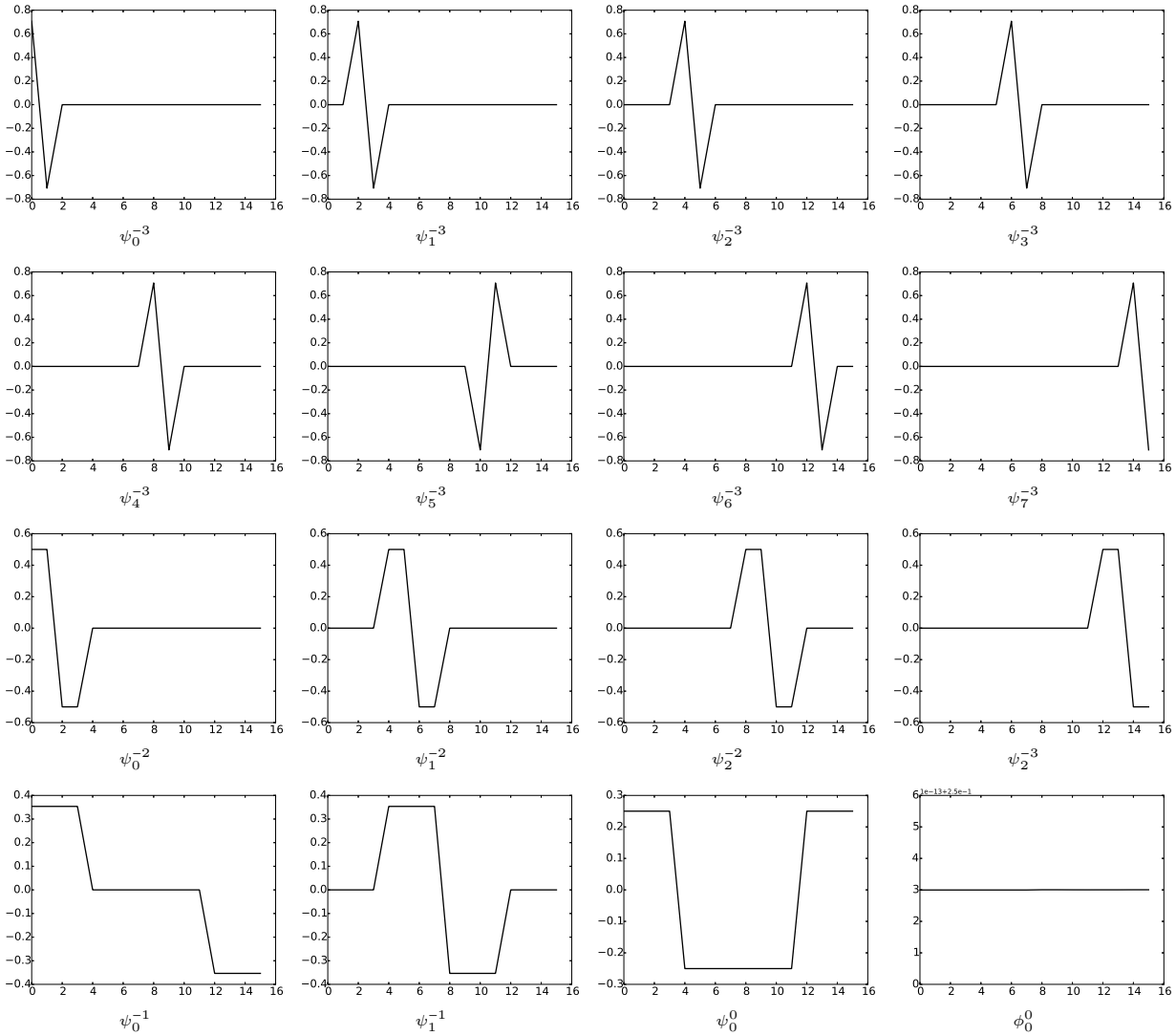



Figure 5.11: **Haar wavelets with MMF**. Reconstruction of the Haar wavelet transform on the diffusion matrix of a cycle graph on $n = 2^4$ vertices by binary parallel MMF.

coordinate blocks $B2$ and $B_3$) and $v_3$ (the wavelet values are mostly negative on the coordinate block $B_4$).

These three mixture models show that PCA is only able to infer the structure of the loading vectors in the first and simplest example, whereas MMF performs well in all three examples, requiring *no prior knowledge* of any structural information about the loading vectors. Therefore, MMFs can be a valuable unsupervised learning tool for discovering structure in matrices with underlying structure, such as covariance matrices.

### *5.6.7 MMF Wavelets*

The extent to which MMF can recover a "good" wavelet basis can have important consequences for downstream applications. In the examples below, we perform MMF on several synthetic graphs. In particular, we are interested in matrices derived from graphs which have some underlying structure.

**Example 1: Haar Wavelets**. Recall from our discussion in Section 4.4 that the Haar wavelet transform diagonalizes matrices with nested (hierarchical) structure of the form (4.13) into (4.14), where the size of the set $T_\ell$ has dimensionality $2^{\ell_0}, 2^{\ell_0-1}, \ldots, 1$. The Haar wavelet transform is equivalent to a second order parallel MMF in which $S_\ell^{\text{act}} = T_\ell$ and the dimensionality $\delta_\ell$ of $S_\ell^{\text{act}}$ is halved at each level. The structure of the matrix $A$ in equation (4.14) suggests that Haar wavelets are particularly well suited to spaces with tree–like metric structure, and therefore it is natural to ask whether MMF can recover the Haar system in this case. To answer that question, we set $A \in \mathbb{R}^{2^m \times 2^m}$ to be the diffusion matrix $A = e^{-\alpha\mathcal{L}}$ of a cycle graph $C_n$ on $n = 2^m$ vertices whose graph Laplacian is $\mathcal{L}$. In particular, we set $m = 4$ and $\alpha = 0.1$. We compute the binary parallel MMF up to depth $L = 5$ (by Algorithm 2), which results in fifteen wavelets and a single scaling function. As Figure 5.11 shows, MMF completely recovers the Haar wavelet transform.

**Example 2: Hypercube Graph**. Recall from Section 4.3 that the Fourier transform of a function on the $d$–dimensional cube is called the Hadamard transform. Similar to the Haar transform, this is another case where MMF recovers a well known algorithm. Equation (4.12) is not quite an MMF, because $H$, and consequently $W_1, \ldots, W_d$, are not rotations (they are orthogonal, but $\det(H) = -1$). However, if we swap the columns of $H$ in the form

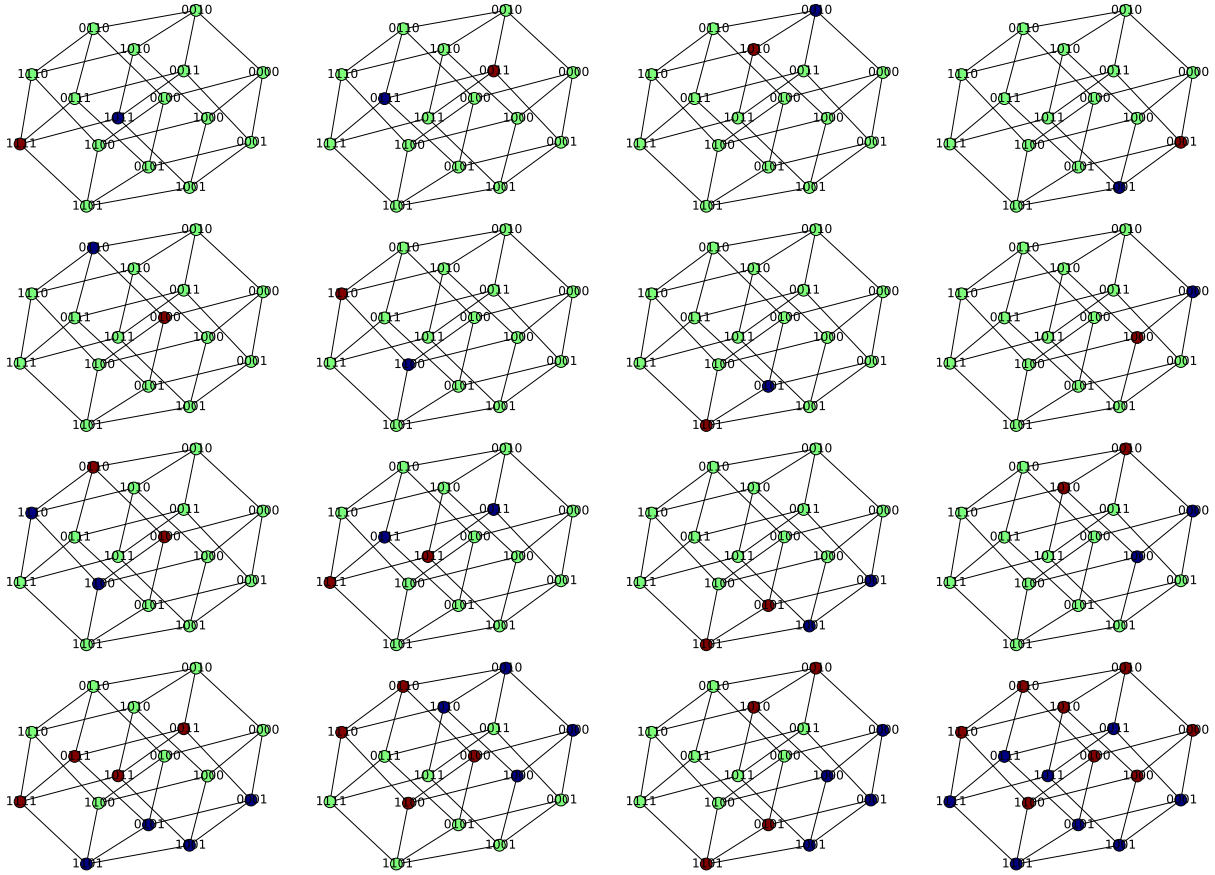


Figure 5.12: Wavelets recovered by binary GreedyParallelMMF on the diffusion matrix of the 4–cube graph. The vertex numbers are shown in binary.

$$\tilde{H} = \Omega H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \qquad \Omega = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \tag{5.32}$$

we can cast the Hadamard transform to a type of MMF. Using (5.32), we get

$$f \mapsto \tilde{\mathcal{H}} f, \qquad \tilde{\mathcal{H}} = \tilde{H}^{\otimes d} = \tilde{W}_d \tilde{W}_{d-1} \dots \tilde{W}_1, \qquad \tilde{W}_k = I_{2^{d-k}} \otimes \tilde{H} \otimes I_{2^{k-1}},$$

which is equivalent to the original (4.10) up to permutation of the entries of the result. Specifically, $\tilde{\mathcal{H}} f = \Omega^{\otimes d} \mathcal{H} f$, and at the same time the Laplacian of the hypercube decomposes into

$$\Delta = \tilde{W}_1^\top \tilde{W}_2^\top \dots \tilde{W}_d^\top \tilde{D} \tilde{W}_d \dots \tilde{W}_2 \tilde{W}_1, \qquad \text{with } \tilde{D} = \Omega^{\otimes d} D \, \Omega^{\otimes d},$$

which is a binary order parallel MMF. In Figure 5.12, we consider the diffusion kernel $A = e^{-\alpha \mathcal{L}}$ of a 4–cube graph with graph Laplacian $\mathcal{L}$ and $\alpha = 0.01$, apply Algorithm 2 to compute the binary parallel MMF of $A$ up to depth $L = 4$ and plot all the wavelets. For ease of notation each of the 16 vertices are labelled in binary.

**Example 3: Cayley Graph**. In the last example we run binary parallel MMF on the graph Laplacian of a Cayley graph of the symmetric group $S_4$ with generators $(1, 2, 3, 4)$ and $(1, 2)$. Figure 5.13 shows the resulting wavelets computed by binary parallel MMF.

**Example 4: Barbell Graph**. For this experiment we use a simple synthetic graph called the barbell graph. A barbell graph $G_{n,k}$ consists of two complete graphs, each one on $n$ vertices, connected by $k$ "bridges". A bridge is an edge connecting a vertex located in one the complete graphs to a vertex located in the other complete graph. We consider the diffusion kernel $A = e^{-\alpha \mathcal{L}}$, where $\mathcal{L}$ is the graph Laplacian of $G_{8,1}$ and $\alpha = 0.01$. We run second order parallel MMF (Algorithm 2) on $A$ and plot all the wavelets in Figure 5.14.

The behavior of the diffusion process on this graph is intuitive — the diffusion progresses more quickly within each of the two complete subgraphs and more slowly along the bridge connecting them. Therefore, we expect to see multiple wavelets supported on each of the two complete subgraphs of the barbell graph and a global wavelet that can distinguish between the components. As Figure 5.14 shows, this is indeed the case — there are various wavelets supported locally on each cluster, while the level $\ell = 0$ wavelet manages to separate the two clusters.

**Example 5: Hierarchical Graphs**. One of the most intuitive ways to conceptualize MMFs is to consider the case when $G$ is a graph with multiscale, hierar-

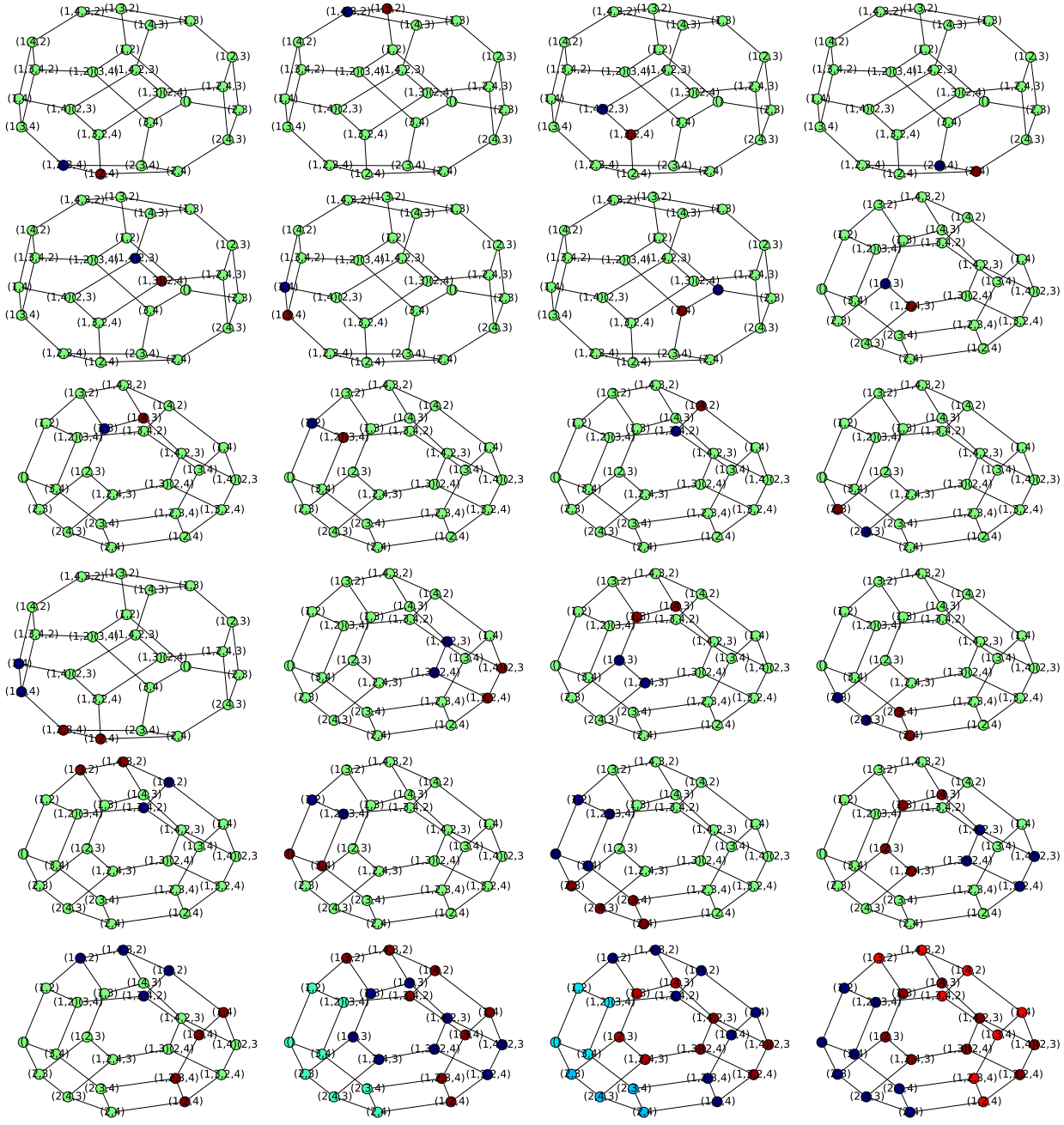


Figure 5.13: **MMF on Cayley graph**. Wavelets recovered by binary GreedyParallelMMF on the graph Laplacian of the Cayley graph of the symmetric group $S_4$.

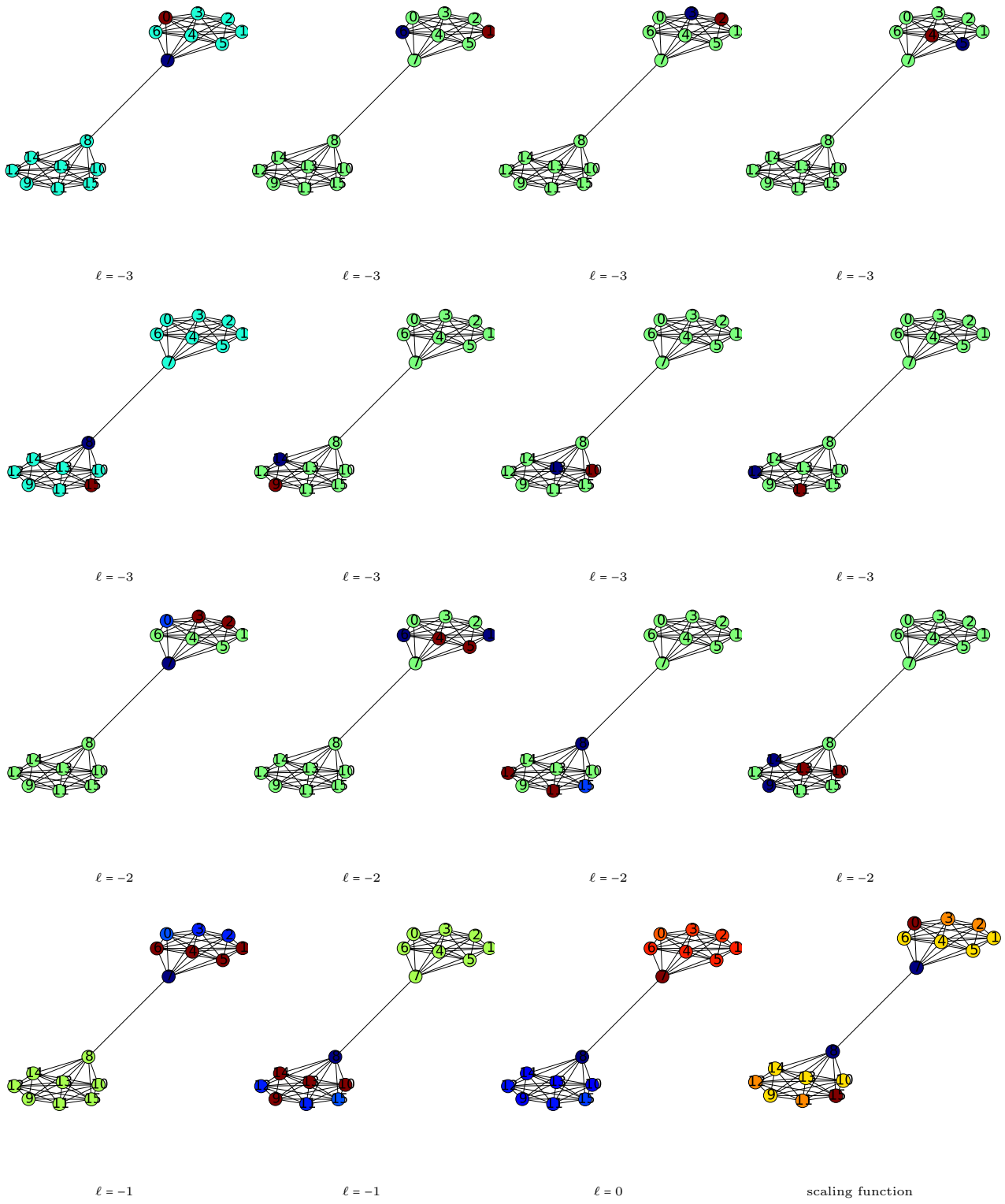



Figure 5.14: **MMF on a barbell graph**. All the wavelets (of different resolution $\ell$) recovered by binary GreedyParallelMMF on the diffusion matrix of a barbell graph $C_{8,1}$.

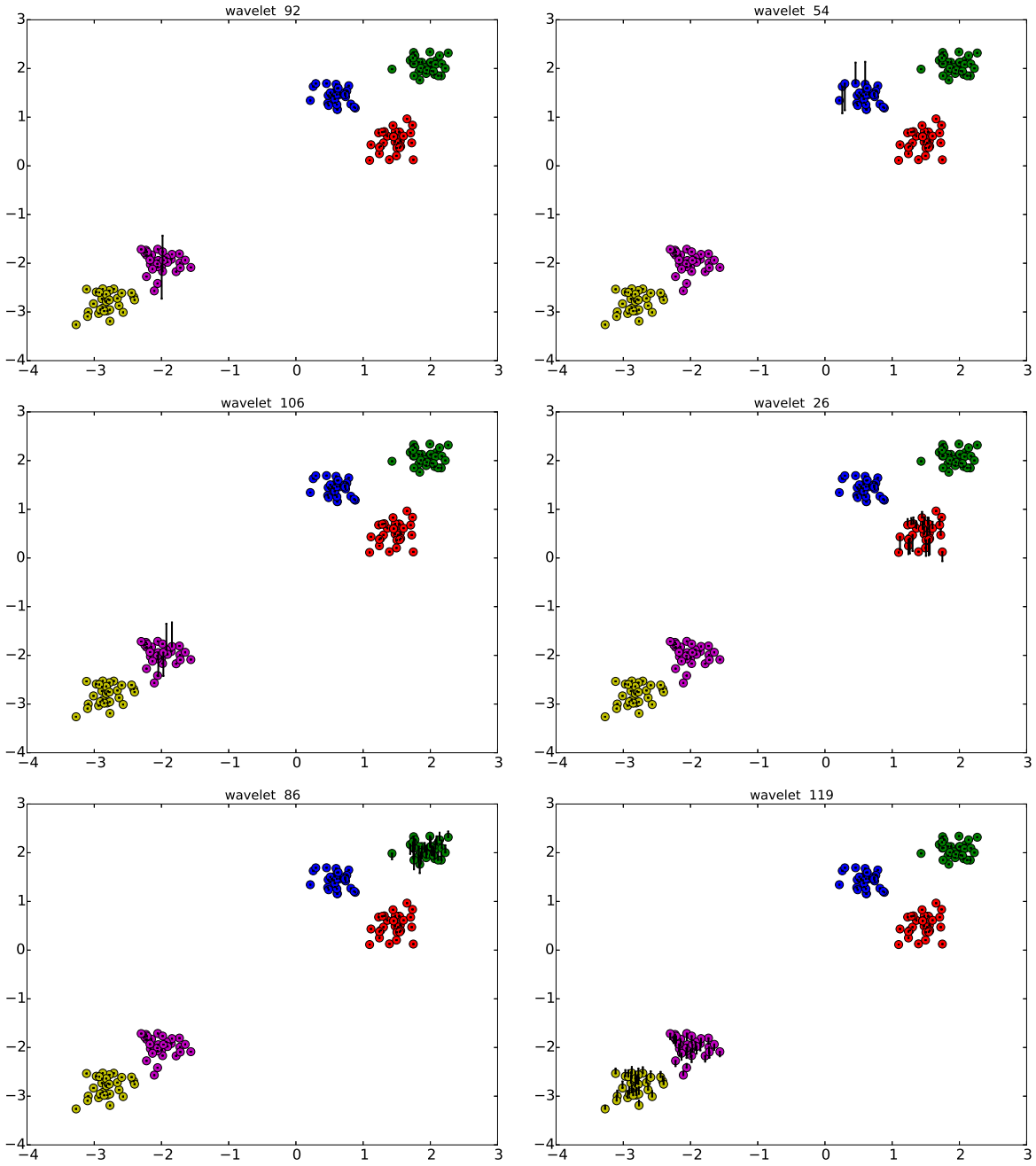



Figure 5.15: **MMF wavelets on a hierarchical graph**. Wavelets of different resolution recovered by binary parallel MMF on the multiscale graph in Example 4 in Section 5.6.7. Points from the blue and the red Gaussians form one meta–cluster, while the other three Gaussians form the other meta–cluster. The bars show the value of specific wavelets at individual data points.

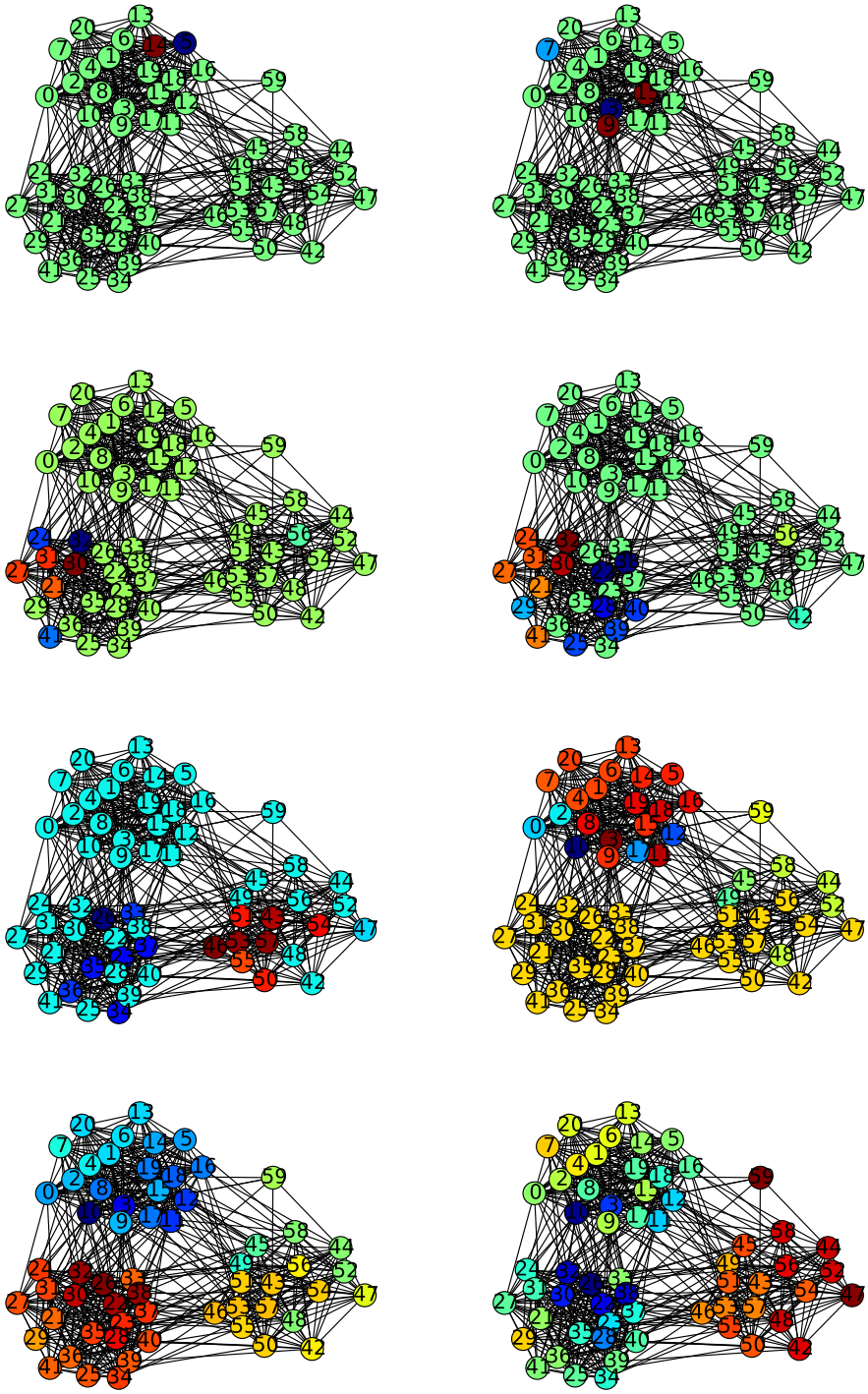


Figure 5.16: **MMF wavelets on a random partition graph**. Wavelets of different resolution recovered by second order parallel MMF on the diffusion kernel of a Gaussian random partition graph described in Example 4 in Section 5.6.7. The plots show the graph connectivity with each node colored according to the wavelet value at that node.

chical structure and $A$ is the diffusion matrix of $G$. In this case, the wavelets returned by MMF provide a multiresolution basis for $G$: the highest frequency wavelets are localized to individual clusters at the bottom of the hierarchy, lower frequency wavelets are localized to clusters at the next level, and so on, until we finally get to the scaling functions, which are fully global. To illustrate this point we construct a multiscale dataset, which is similar to the one we used in Section 5.6.5. We consider a dataset consisting of 250 points in $\mathbb{R}^2$ which is the union of 50 realization of five Gaussian random variables, each on sampled from normal distribution with means $[1.4, 0.6], [0.6, 1.4], [2, 2], [-2, -2], [-2.8, -2.8]$ and standard deviations all equal to 0.25. From this date we construct the pairwise distance matrix $A$ such that for each pair of points $x_i, x_j$, $A_{i,j} = e^{-\|x_i - x_j\|^2/0.5^2}$. We run binary parallel MMF on $A$ and plots some of the resulting wavelets in Figure 5.15 — the bars show the value of specific wavelets at individual data points. MMF captures detail at different scales — for example, wavelets 92, 54 and 106 capture purely local structure; wavelets 26 and 86 separate the red and the green clusters, respectively; wavelet 119 separates the yellow cluster from the purple one.

Another hierarchical graph model we test MMF on is a Gaussian random partition graph (Brandes et al., 2003). Gaussian random partition graphs consist of several partitions whose exact number is determined by several parameters. Each cluster is of size drawn from a normal distribution with mean $s$ and variance $s/v$ for some parameter $v$. The intra–cluster probability of connecting any pair of nodes within the same cluster is $p_{\text{in}}$, while the inter–cluster probability of connecting any pair of nodes belonging to different clusters is $p_{\text{out}}$. The parameters we use to construct the graph in Figure 5.16 are $n = 64$, $s = 20$, $v = 20$, $p_{\text{in}} = 0.9$, $p_{\text{out}} = 0.1$. We run binary parallel MMF on the diffusion kernel of this graph (with $\alpha = 0.1$) and plot some of the recovered wavelets in Figure 5.16. While the first six wavelets in the plot are fairly well localized in different areas in each of the three clusters. In contrast, the last two wavelets are more global

and they can separate the bottom right cluster from the the top on, on one hand, and the bottom left cluster from the remaining two, on the on the other hand.

# CHAPTER 6

# MMF FOR MATRIX COMPRESSION

The massive size of modern datasets often requires matrices arising in learning problems to be reduced in size. Distance matrices or Gram (kernel) matrices, in particular, are often a computational bottleneck, because they are large and dense. If $n$ is the number of data points, the space complexity of these matrices is $n^2$, while the time complexity of the linear algebra operations involved in many learning algorithms, such as eigendecomposition, matrix inversion, or solving least squares problems, usually scales with $n^3$. Without efficient matrix compression methods or, as they are sometimes called, **matrix sketches**, many popular machine learning algorithms are simply inapplicable to today's datasets, where $n$ is often on the order of millions. In numerical analysis, scientific computing and randomized linear algebra matrix sketches are also used for approximations of high order tensors (Drineas and Mahoney, 2007), for preconditioning of linear systems (Woodruff, 2014; Avron et al., 2010), or for the approximation of matrix–matrix and matrix–vector products (Drineas et al., 2006; Martinsson, 2008; Halko et al., 2011).

Classically, a symmetric matrix $A \in \mathbb{R}^{n \times n}$ can be compressed by Principle Component Analysis, however, more recently, as the size of modern datasets continues to grow, several other approximation sketches have been developed. Mostly, they rely on *randomization* and *subsampling* in order to improve on the computational complexity of PCA. Invariably, what all of these approximation algorithms have in common is the underlying assumption that the matrix being compressed has *low rank*, or at least that it can be well modeled by a low rank surrogate. In this chapter we demonstrate that *multiresolution structure* is a viable alternative to the low rank paradigm in machine learning, especially in the context of graph/network data.

We begin with a review the various types of matrix sketches (Sections 6.1, 6.2 and 6.3), discuss how MMF can be used for matrix compression (Section 6.4) and empirically compare the different compression schemes on real data (Section 6.5). In this and the following

chapters we will sometimes use the terms "matrix approximation", "matrix compression" and "matrix sketching" interchangeably.

## 6.1 Principle Component Analysis (PCA)

The most classical way of compressing a symmetric matrix $A \in \mathbb{R}^{n \times n}$ is Principal Component Analysis (PCA) (Eckart and Young, 1936; Jolliffe, 1986)

$$\tilde{A} = Q \Lambda Q^\top, \tag{6.1}$$

where $Q \in \mathbb{R}^{n \times \ell}$ is an orthogonal matrix whose columns contain the $\ell$ leading eigenvectors and $\Lambda \in \mathbb{R}^{\ell \times \ell}$ contains the corresponding eigenvalues on its main diagonal. $\tilde{A}$ is a projection of $A$ onto the subspace spanned by its $\ell$ highest eigenvalue eigenvectors $Q_{:,1}, Q_{:,2}, \ldots, Q_{:,\ell}$. PCA is optimal in the sense that over all the subspaces of size $\ell$ it minimizes the approximation error $\|A - \tilde{A}\|$ as measured in Frobenius, operator or nuclear norm. For that reason approximating $A$ by its top $\ell$ eigenvectors in the form (6.1) is sometimes called the **best rank $\ell$ approximation** of $A$.

As a compression algorithm, PCA reduces the storage requirements of $A$ from $n^2$ to $\ell + n\ell$. Additionally, decomposing matrices by PCA makes it much easier to perform certain linear algebra operations. For example, rather than computing the exact matrix vector product $Av$ in time $O(n^2)$, the approximate matrix vector product $\tilde{A}v$ can be computed in time $O(n\ell)$ (computing $Q^\top v$ takes $n\ell$ operations, rescaling it by the $\mathrm{diag}(\Lambda)$ takes $\ell$ operations, and finally applying $Q$ on its left side takes $n\ell$ operations). Computing the approximate matrix inverse $\tilde{A}^{-1}$ amounts to just taking the reciprocal of the diagonal of $\Lambda$, which can be done in only $\ell$ operations. The main drawback of performing PCA is its computational complexity — unless A has special structure, such as a large degree of sparsity, computing the $\ell$ leading eigenvectors for (6.1) itself costs $O(n^2\ell)$ operations, which for modern size datasets is often prohibitive and usually negates whatever computational savings $\tilde{A}$ might afford afterwards.

## 6.2 Projection Based Methods for Matrix Compression

Projection based methods approximate a symmetric matrix $A \in \mathbb{R}^{n \times n}$ in the form

$$\tilde{A} = CW^{\dagger}C^{\top}, \tag{6.2}$$

with $C = AS$ and $W = S^{\top}AS$, where $S \in \mathbb{R}^{n \times \ell}$ is a random projection matrix with $\ell \ll n$. Note that the projection based sketches described here apply to both symmetric and nonsymmetric matrices, however below we assume that $A$ is symmetric, just like in the rest of this thesis.

Classically, as first introduced by Johnson and Lindenstrauss (1984), $C$ is an orthogonal projection onto a random $\ell$–dimensional space. Their famous Johnson–Lindenstrausss Lemma states that a set of $n$ points in a high dimensional Euclidean space can be embedded via a random projection into an $\ell$ dimensional Euclidean space, where $\ell = O(\log n)$ (independent of the ambient dimension), such that all the pairwise distances between the points are approximately preserved (Johnson and Lindenstrauss, 1984). The Johnson–Lindenstrauss Lemma has since inspired many approximate, randomized linear mappings which are not random projections in the traditional linear algebraic sense, but instead satisfy similar approximate metric preserving properties, hence their name. Halko et al. (2011) and Mahoney (2011) provide detailed reviews of random projection sketches which we summarize below.

For example, $C$ can be constructed by taking a **Gaussian mixtures** of the columns of $A$, i.e., $S$ consists of $\ell$ spherically symmetric random vectors the coordinates of which are i.i.d. $\mathcal{N}(0, 1)$ random variables. These methods use Johnson–Lindenstrauss type arguments to show that the resulting low dimensional random sketch preserves most of the information at least about the part of $A$ spanned by its high eigenvalue eigenvectors (Halko et al., 2011). As a result, they come with strong guarantees, but incur the cost of having to compute $\ell$ dense linear combinations of the columns of $A$. Even if $A$ was sparse, this process destroys the sparsity of the approximation.

**Element-wise random projections** (Achlioptas, 2003) are another possibility — here

the entries of $S$ are random variables sampled independently from $\{-1, 1\}$ with probability 1/6 or 0 with probability 2/3. The advantage of this method is that up to 2/3 of the entries of $S$ can zeroed out out when computing $C$, which makes this algorithm particularly useful for computing random projection sketches at scale.

Another group of random projection based sketches are the so-called **structured random projections**. In their most general form, structured random projections set $S = \sqrt{n/\ell}DTR$, where $D \in \mathbb{R}^{n\times n}$ is a diagonal matrix of Rademacher random variables (i.e., each diagonal entry is drawn independently from $\{-1, 1\}$ with equal probability), $T$ is a unitary matrix, and $R$ restricts the size of $S$ to $n \times \ell$ by some sampling procedure. Depending on the choice of $T$, the approximation guarantees of (6.2) can vary. For example, $T$ could be the normalized Fourier transform, in which case $S$ is the so-called subsampled randomized Fourier transform (SRFT) which has the form

$$S = \sqrt{\frac{n}{\ell}}DFR, \tag{6.3}$$

where $D \in \mathbb{R}^{n\times n}$ is a diagonal matrix of Rademacher random variables, $F \in \mathbb{R}^{n\times n}$ is the discrete Fourier transform applied to the columns of $A$, and $R \in \mathbb{R}^{n\times\ell}$ is simply a selector matrix uniformly sampling $\ell$ of the $n$ columns of the Fourier transform (Halko et al., 2011). However, the SRFT matrix (6.3) is just one possibility — other structured random projections use the discrete cosine transform (Avron et al., 2010) or subsample the Walsh–Hadamard transform matrix (4.11) (in place of the Fourier transform matrix $F$). The latter algorithm is known as the Fast Johnson–Lindenstrauss Transform (FJLT) (Ailon and Chazelle, 2009, 2010; Matoušek, 2008; Tropp, 2011). Additionally, the matrix $R$ does not need to be a selector matrix — each $R_{i,j}$ of its entries can be drawn from an appropriately chosen distribution (Ailon and Chazelle, 2010; Matoušek, 2008).

The running time of random projection based sketches varies widely depending on the type of random projection, the type of the matrix being factorized (dense or sparse) or the

way random projections are implemented. However, overall random projections tend to be either expensive to compute or expensive to apply to a vector/matrix downstream. For example, if $C$ is constructed by taking a Gaussian mixtures of the columns of $A$, computing the matrix–matrix product $C = AS$ requires $O(n^2\ell)$. The $n^2$ factor makes this type of random projection prohibitively expensive for modern datasets (which are typically of size $n \approx 10^5 - 10^6$) and outweighs the computational advantage random projection sketches might bring downstream. In the case of the SRFT, computing the vector–matrix product $v^\top DF$ for some $v \in \mathbb{R}^n$ takes $O(n \log n)$ time, or even just $O(n \log \ell)$ time if only $\ell$ of the coordinates of the product need to be accessed. Thus, implementing the random projection itself (i.e., computing $C$) costs $O(n^2 \log \ell)$. Given that multiplying $\tilde{A}$ of the form (6.2) by a dense vector takes $2n\ell + \ell^2$ operations, the computational overhead of computing random projection sketches in the first place is nontrivial.

## 6.3 Nyström Methods for Matrix Compression

The Nyström method constructs a low rank approximation $\tilde{A}$ of a positive semidefinite matrix $A \in \mathbb{R}^{n \times n}$ as a decomposition of the form

$$\tilde{A} = CW^\dagger C^\top, \tag{6.4}$$

where $C = AP$ and $W = P^\top AP$ with $P \in \mathbb{R}^{n \times \ell}$. The matrix $P$ is just a diagonal binary matrix which acts as a selector for $\ell$ of the columns of $A$ and so, in practice, $C$ is constructed simply by subsampling $\ell$ of the *actual* columns of $A$. The matrix $W$ is formed in the same fashion except that both its columns *and* rows are subsampled identically. A less formal, but perhaps more clear way of describing the way $C$ and $W$ are constructed, is in terms of an $n$–dimensional sampling probability vector $p$. Each element $p_i$ corresponds to the probability of selecting the $i$–th column (and row) to include in $C$ (and $W$, respectively). When the matrix $A$ is a kernel matrix, constructed by applying a pairwise similarity function over a

dataset of $n$ points, the Nyström method has an intuitive explanation — selecting the $\ell$ actual columns/rows of $A$ is equivalent to selecting $\ell$ data points such that $\tilde{A}$ approximates $A$ sufficiently well. Since both $C$ and $W$ consist of actual columns of $A$, the Nyström method offers interpretability — for example, if $A$ is a kernel matrix of pairwise similarities between $n$ genes, the Nyström method simply chooses the most representative genes to approximate $A$. This is in contrast to compression by PCA, in which the principle components cannot be mapped to actual genes. The space requirement of storing the Nyström approximation is $\ell n + \ell^2$. Multiplying $\tilde{A}$ by a dense vector takes $2n\ell + \ell^2$ operations, while inverting $\tilde{A}$ takes $O(\ell^3)$ time.

The key assumption of the Nyström method is that most of the information contained in $A$ can be captured by a small number $\ell \ll n$ of the actual columns of $A$ — this is the so-called **low rank assumption** of matrix approximations. Since $\ell \ll n$, $W$ ends up being a square matrix of dimensions much smaller than $n$. Thus, the overhead of having to store $W$ and having to efficiently compute its inverse, as required in (6.4), is only minimal. The goal then is to construct $C$ by judiciously selecting a small, representative number of the columns/rows of $A$ and then show that $\tilde{A}$ is a sufficiently good approximation of $A$ by measuring the error $\|A - \tilde{A}\|$ in terms of Frobenius, spectral or trace norm. Of course, the extent to which $A$ is approximated by the low rank matrix $\tilde{A}$ depends on the choice of importance sampling distribution $p$ with respect to which the columns/rows of $A$ are sampled. Recent years have resulted in the development of many variations of the Nyström method. **Uniform Nyström** — the earliest Nyström method, yet one of the easiest to compute in practice and surprisingly well working for certain types of graphs — samples $\ell$ of the $n$ columns/rows of $A$ uniformly at random with (Drineas and Mahoney, 2005) or without replacement (Williams and Seeger, 2001).

An alternative technique, introduced by Drineas and Mahoney (2005), uses an importance sampling probability distribution constructed from the $\ell^2$ norm of each column of $A$. In a subsequent series of papers by Mahoney and Drineas (2009); Mahoney (2011); Gittens

and Mahoney (2013) the **importance sampling Nyström** was adapted to use the so-called statistical leverage scores as weights instead. Leverage scores were first introduced in (Mahoney and Drineas, 2009) as a sampling technique for the more general type of low rank approximation, called CUR decomposition, which is the analogue of the Nyström decomposition for nonsymmetric matrices. The leverage scores of a matrix $A \in \mathbb{R}^{n\times n}$ with respect to a rank parameter $r \ll n$ are calculated from the SVD decomposition $A = U\Sigma V^*$ (recall $U = V$ if $A$ is symmetric), where the columns of the unitary matrix $U \in \mathbb{R}^{n\times r}$ contain the top $r$ singular vectors of $A$. The leverage score for the $i$–th row of $A$ is then given by

$$\mu_i = \|U_{i,:}\|_2^2, \qquad 0 \le i \le n-1. \tag{6.5}$$

Since $\sum_i \mu_i = \|U\|_{\text{Frob}}^2 = r$, one can define a probability distribution over the rows of $A$ by setting $p_i = \mu_i/r$. Sampling from this distribution essentially guarantees that the columns/rows of $A$ which are most highly correlated with the span of the top $r$ singular vectors are most likely to be selected for the approximation (6.4). Since leverage scores are defined relative to the best rank $r$ approximation of $A$, the approximation error $\|A-\tilde{A}\|$ of the **leverage scores Nyström** is often evaluated relative to the best rank $r$ approximation of $A$ (6.1). In some cases, for example, when $A$ is a linear kernel or a dense RBF kernel with certain parameters, the leverage scores are fairly uniform, and so sampling using the distribution defined by the leverage scores is not much more informative than uniform sampling, for example (see (Gittens and Mahoney, 2013)). In fact, in these cases the leverage score Nyström has higher approximation error than the Uniform Nyström. In this situation uniform sampling is a more appropriate method of compression since the leverage scores computation requires first obtaining the singular vectors of a potentially dense matrix. Despite the availability of efficient methods (such as the Lanczos iterative eigenvalue algorithm for finding the SVD of a matrix, see (Golub and Van Loan, 2012) for details), in the interest of computational efficiency leverage score also needs to be approximated in the case of large matrices (see

(Drineas et al., 2012; Gittens and Mahoney, 2013) for leverage scores approximation). Additionally, note that $r$ is an additional parameter for this type of Nyström. To obtain certain theoretical guarantees, one might need to set $\ell \geq r$, but in general the choice of $r$ is somewhat arbitrary, while in practice its effect on the uniformity of the distribution $p_i = \mu_i / r$ could be significant.

More recently, Zhang et al. (2008) and Zhang and Kwok (2010) have taken a different approach with their $k$–**means clustering based Nyström** method. It is specifically designed to approximate a kernel matrix $A$, reflecting the pairwise similarity between $n$ data points. Instead of probabilistically selecting columns/rows of the kernel matrix $A$, this Nyström method clusters the data points using $k$–means clustering and then uses only the rows/columns of $A$ corresponding to the cluster centers to reconstruct $A$. This method avoids explicitly constructing $A$, which offers huge computational savings for downstream algorithms, such as kernel methods, where constructing and storing $A$ in the first place could be a bottleneck.

First proposed by Deshpande et al. (2006), and more recently modified by Kumar et al. (2012) and Wang and Zhang (2013), the **adaptive sampling Nyström** method updates the weighted sampling probabilities for each column in a series of rounds instead of using a fixed distribution, with or without replacement. On the first round, a few columns/rows are selected from some initial distribution $p$ (e.g., uniform), the approximation $\tilde{A}$ is performed using those columns/rows and subsequently the probability distribution $p$ is updated based on the approximation error $\|A_{:,i} - \tilde{A}_{:,i}\|$ incurred in each column of $A$. The second round proceeds identically — a few more columns are selected from the updated distribution $p$, the approximation is performed again using the columns that were selected in the first two rounds, the probability distribution $p$ is updated and so on until the desired $\ell$ number of columns is reached.

An alternative approach, called **ensemble Nyström** (Kumar et al., 2009), samples the columns of $A$ not just once, but $t$ times, producing $t$ different approximations $\tilde{A}_1, \tilde{A}_2, \ldots, \tilde{A}_t$

| dataset | n | kernel type |
|---|---|---|
| **HEP**: high energy physics-theory collaboration graph | 9,877 | normalized Laplacian |
| **GR**: general collaboration and quantum cosmology collaboration graph | 5,242 | normalized Laplacian |
| **Gnutella**: peer-to-peer network from August 6, 2002 | 8,717 | normalized Laplacian |
| **Enron**: subgraph of the Enron email graph | 10,000 | normalized Laplacian |
| **Dexter**: a bag of words dataset | 2,000 | linear ($d$=2×10$^4$) |
| **Abalone**: physical measurements of Abalones | 4,177 | RBF ($d$=8) |
| **Wine Quality**: wine physicochemical tests | 4,898 | RBF ($d$=12) |

Table 6.1: **Datasets**. Summary of the datasets used for the matrix compression experiments (Bache and Lichman, 2013; Leskovec and Krevl, 2014; Davis and Hu, 2011). For the normalized Laplacian kernels, the size reflects the number of vertices in the dataset. When a linear or a radial basis function (RBF) kernel is used to construct a symmetric kernel matrix, $n$ is the number of data points and $d$ is the dimensionality of each data point.

by one of the aforementioned techniques, and then averages those $t$ approximations by appropriately chosen weights.

Extensive experiments on numerous real datasets comparing these and other Nyström variations can be found in (Gittens and Mahoney, 2013; Zhang and Kwok, 2010; Wang and Zhang, 2013) and the references they cite. For analysis of the nonsymmetric version of Nyström, called CUR, see the work of Drineas et al. (2008); Mahoney and Drineas (2009); Wang and Zhang (2013).

## 6.4 MMF for Matrix Compression

In contrast to the above matrix sketches, MMF approximates $A$ in the form

$$\tilde{A} = U_1^\top U_2^\top \dots U_L^\top H U_L \dots U_2 U_1, \tag{6.6}$$

which can be intuitively thought of compression down to the core $H_{S_L^{\text{act}},S_L^{\text{act}}}$ (ignoring the remaining $n-|S_L^{\text{act}}|$ values on the diagonal of $H$, as those are trivial to store and process from the perspective of downstream applications). This is similar to (6.4), which can be thought of compression down to a small $\ell \times \ell$ dimensional matrix $W^\dagger$, since the matrix $C$ does not

need to be stored in explicit form as its columns are actual columns of $A$.

However, MMFs operate in a slightly different regime than the low rank sketches described above. Recall from our discussion on MMF and Figure 5.1 that compressing a matrix to size $\delta \times \delta$ means something slightly different for MMF and the other matrix sketches — in addition to the $\delta \times \delta$ core, MMF also preserves a sequence of $n - \delta$ wavelet frequencies. The space requirement for the most compact type of MMF, a $k$–order parallel MMF, therefore consists of $\delta^2 + (n - \delta)$ floats to store $H$ and $k^2 Ln$ floats to store the rotations $U_1, \ldots, U_\ell$ (this is assuming that exactly one row/column is eliminated after each rotation $U_\ell$). Note that the rotations do not need to be stored explicitly in matrix form as they are very sparse — storing a Givens rotation $U_\ell = U_{i,j,\theta}$ requires storing just the two indices $i, j$ and the angle $\theta$. Applying each rotation matrix $U_\ell$ to a vector (i.e., $U_\ell v$) can be computed in $kn$ operations, while computing $Hv$ takes $\delta^2 + (n - \delta)$ operations — thus, computing the approximate matrix–vector product $\tilde{A}v$ requires $2kLn + \delta^2 + (n - \delta)$ operations, and so the total cost is $O(2kLn + n + \delta^2)$.

## 6.5 Experiments

In this chapter we compare how well the different matrix sketches described earlier in the chapter can approximate synthetic and real symmetric matrices. We measure the error $\mathcal{E}_{\text{Frob}} = \|A - \tilde{A}\|_{\text{Frob}}$, as defined earlier in (5.28). This form is exactly the way the other matrix sketches we described in this chapter measure the approximation error, except $\tilde{A}$ is an approximation of the form (6.2) or (6.4).

**Approximation Error on Structured vs. Unstructured Matrices**. To verify that MMF can successfully be used as a compression tool, like the other sketches described above, we measure the approximation error of factoring two types of matrices — random and structured matrices. The structured matrix is a Kronecker product graph $K_{1,\kappa}$ with $K_1 = [1, 0.1; 0.1, 1]$ and $\kappa = 10$ (see (5.29) for definition). Recall that Kronecker product graphs

(a) random matrix (b) Kronecker matrix

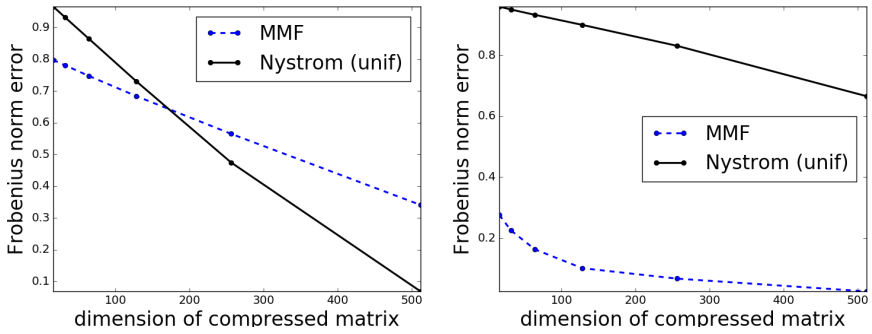



Figure 6.1: **MMF on random vs. structured matrices**. Frobenius norm error $\mathcal{E}_{\text{Frob}}$ of compressing matrices with binary parallel MMF vs. the Uniform Nyström method as a function of the dimension of the core $H_{S_L,S_L}$ that $A$ is compressed down to.

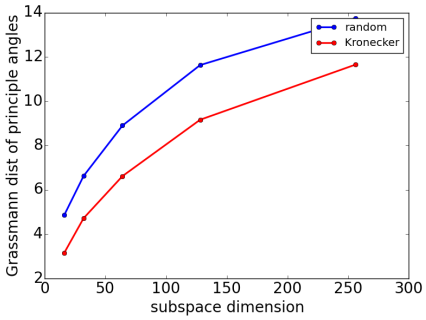



Figure 6.2: **Grassmann distance between subspaces**. Grassmann distance (6.7) between MMF subspaces of different dimensionality. The figure compares structured (Kronecker) and random matrices.

have a fractal, cluster–of–clusters structure. The random matrix is of the same size ($n = 1024$) and consists of i.i.d. normal random variables. The approximation error is measured by (5.28) and normalized in the form $\mathcal{E}_{\text{Frob}}/\|A\|_{\text{Frob}}$. Figure 6.1 shows that binary parallel MMF performs suboptimally when the matrix lacks an underlying multiresolution structure. However, on the well structured Kronecker matrix MMF systematically outperforms other algorithms (in this case the uniform Nyström method).

**Principal Angles between Subspaces.** In addition to measuring the approximation error on structured and unstructured matrices (which is essentially a statement about how well the eigenvalues of $\tilde{A}$ approximates the eigenvalues of $A \in \mathbb{R}^{n\times n}$), we also quantify how

well the resulting subspaces of the MMF basis align with the corresponding best rank $r$ subspaces of $A$ (i.e., the one recovered by PCA). A standard way of measuring the deviation between two equidimensional subspaces is in terms of so-called principal angles. Given two $r$–dimensional subspaces $T$ and $P$ (represented as orthonormal basis matrices of size $n \times r$), there are $r$ principal angles $\theta_1, \theta_2, \ldots, \theta_r$ (where $\theta_i \in [0, 2\pi]$) between $T$ and $P$. Each angle $\theta_i \in \{\theta_1, \theta_2, \ldots, \theta_r\}$ is recursively defined as follows

$$\cos(\theta_i) = \max_{t \in \langle T_{:,1}, \ldots, T_{:,r} \rangle,\, p \in \langle P_{:,1}, \ldots, P_{:,r} \rangle} |t^\top p| = |t_i^\top p_i|,$$

subject to the constraints

$$\|t\| = \|p\| = 1, \quad t^\top t_j = \cdots = t^\top t_j = 0, \quad p^\top p_j = \cdots = p^\top p_j = 0 \quad \text{for } j = 1, \ldots, r-1.$$

The vectors $t_1, \ldots, t_r$ and $p_1, \ldots, p_r$ are called the principal vectors. The principal angles can be computed easily using SVD decomposition (Björck and Golub, 1973) as follows

$$T^\top P = U \Sigma V^\top,$$

where $\Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_r)$ (with $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r$) are the singular values of $T^\top P$. The principal angles between the subspaces $T$ and $P$ are given by $\cos\theta_i = \sigma_i$ $(1 \leq i \leq r)$, while the principal vectors for $T$ and $P$ are respectively the columns of $TU$ and $PV$. Among the various possible distance metrics that can be used to quantify the distance, we use the Grassmann distance

$$d_{\mathrm{Gr}}(T, P) = (\sum_{i=1}^{r} \theta_i^2)^{1/2}. \tag{6.7}$$

Figure 6.2 shows how the Grassmannian distance between the scaling subspace computed by binary parallel MMF (which is equivalent to compressing $A$ down to some core size $r$ and taking the rows of $U_L \ldots U_2 U_1$ indexed by the $r$ elements in the active list $S_L^{\mathrm{act}}$) and the corresponding $r$–dimensional subspace computed by PCA changes as a function of $r$.

The figure compares two types of matrices — a structured Kronecker product matrix and a random matrix as described in the previous experiment in this section. For the Kronecker matrix, the Grassmann distance is lower for all subspace sizes $r$, which means that MMF consistently finds a subspace that aligns with the top $k$ principal components well if the underlying matrix has hierarchical structure.

**Compression of Real Data**. We also evaluate the performance of MMF for matrix compression on several real datasets which are widely used in the matrix sketching literature and are listed in Table 6.1. We compare the randomized MMF (RandomizedMMF with $k$ = 2) with algorithms from each of the two matrix sketching categories described earlier in this chapter — uniform sampling of the columns without replacement, leverage score importance sampling, Sampled Randomized Fourier Transform (SRFT) and Gaussian mixtures of the columns, which are denoted respectively `unif`, `leverage`, `srft` and `gaussian` in Figure 6.3. We set the parameters for these four algorithms and the kernel parameters for the datasets, as described by Gittens and Mahoney (2013). The approximation error is normalized with respect to the best rank $r$ approximation of $A$ (denoted by $A_r$) in order to be able to compare with the leverage score Nyström method, which is parameterized by a rank parameter $r$ required for the computation of the leverage scores. So the error is now normalized in the form $\mathcal{E}_{\text{Frob}}/\|A - A_r\|_{\text{Frob}}$. The approximation error is plotted as a function of the size of the core $H_{S_L,S_L}$ that $A$ is compressed down to (in the case of MMF) or the dimensions of $W^\dagger$ in (6.2) and (6.4) (for the other two types of sketches). The results in Figure 6.3 suggest that, despite similar wall clock times, by leveraging more nuanced structure in matrices arising from data than just low rank, MMF outperforms standard compression techniques by a very large margin.

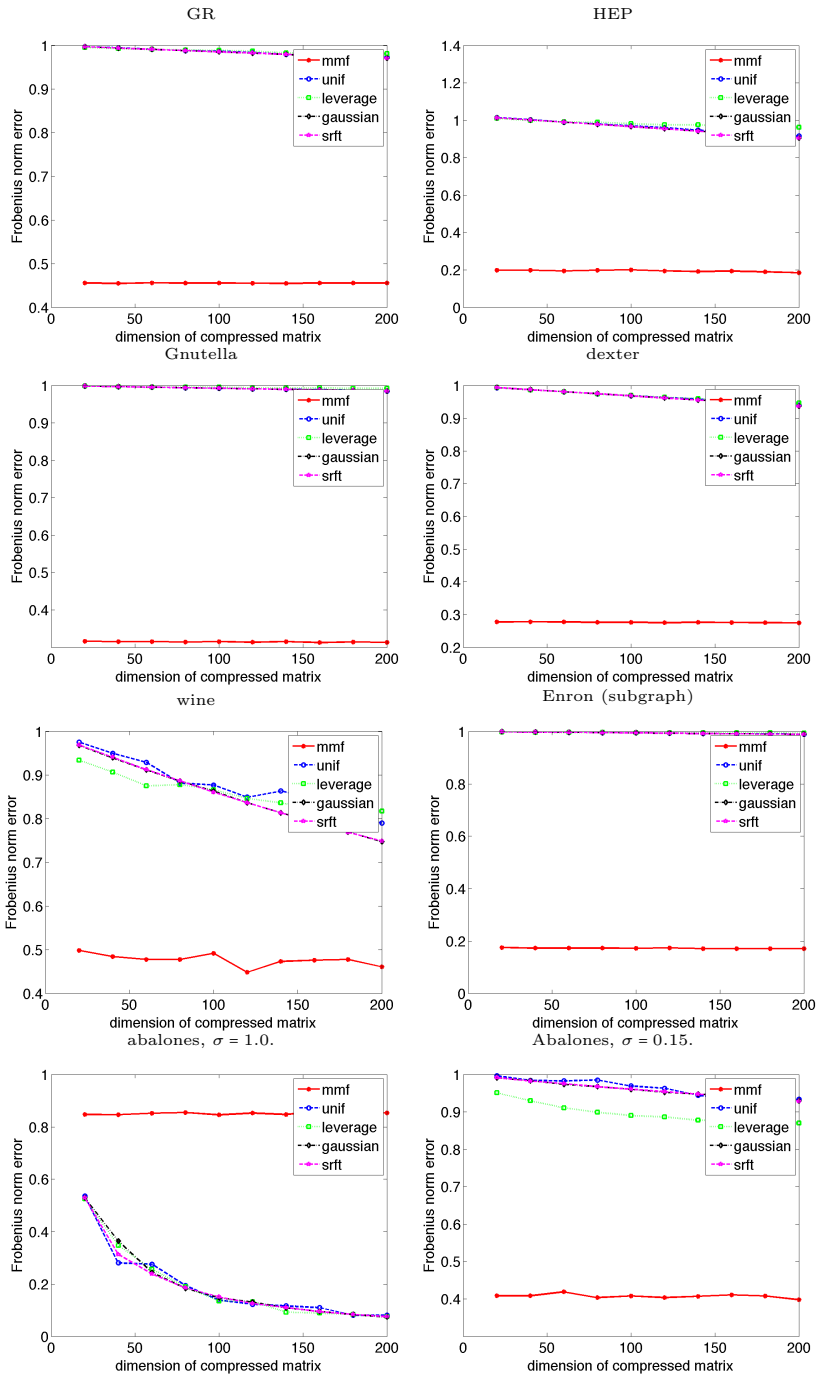



Figure 6.3: **MMF approximation error**. The Frobenius norm error $\mathcal{E}_{\text{Frob}}$ of compressing matrices with binary randomized MMF (Algorithm 3) vs. other sketching methods as a function of the dimension of the core $H_{S_L,S_L}$ that $A$ is compressed down to.

# CHAPTER 7

# PARALLEL MULTIRESOLUTION MATRIX FACTORIZATION

The pMMF algorithm and library we present below are original work first published in (Teneva et al., 2016; Kondor et al., 2015a,b). In this chapter we will use the matrix notation defined in Section 1.2, as well as the MMF notation and definitions we introduced in the rest of Chapter 5.

## 7.1 Limitations of MMF Algorithms

The main obstacle to using MMF for large matrix sketching has been the cost of computing the factorization in the first place. All the MMF algorithms introduced in Section 5.2 construct the sequence of transformations (5.3) in a greedy way, at each level choosing $U_\ell$ so as to allow eliminating a certain number of rows/columns from the active set while incurring the least possible contribution to the final error. Assuming the simplest case of each $U_\ell$ being a Givens rotation $U_{i,j,\theta}$, this involves: (a) finding the pair of indices $(i, j)$ involved in the Givens rotation $U_\ell = U_{i,j,\theta}$ and (b) finding the rotation angle $\theta$. While (b) is generally easy, finding the optimal pair $(i, j)$ in the case of binary rotations (or the optimal set of indices $(i_1, \ldots, i_k)$ in the case of $k$–point rotations) is a combinatorial problem that scales poorly with the matrix dimension $n$. Specifically, the limitations of MMF algorithms introduced in the previous chapter lead to the following computational bottlenecks:

1. The optimization is based on inner products between columns, so it requires computing the Gram matrix

$$G_\ell = A_{\ell-1}^\top A_{\ell-1}, \tag{7.1}$$

   which has complexity of $O(n^3)$. Note that computing the Gram matrix needs to be done only once: once we have $G_1 = A^\top A$, each subsequent Gram matrix can be computed via the recursion $G_{\ell+1} = U_\ell G_\ell U_\ell^\top$.

2. In the simplest MMF algorithm, GreedyJacobiMMF, described in Section 5.2.1, $U_\ell$ is chosen to be a $k$-point rotation which allows removing a single row/column from the active set with the least contribution to the final error. The complexity of finding the indices involved in the Givens rotations is $O(n^k)$. Given that there are $O(n)$ rotations in total, in the $k = 2$ case the cost of finding all of them is $O(n^3)$.

3. The drawback of GreedyJacobiMMF is that sometimes it tends to lead to cascades, as discussed at the end of Section 5.2.4. This motivated an alternative MMF algorithm, GreeyParallelMMF, in which each $U_\ell$ is the direct sum of $|S_\ell|/k$ separate nonoverlapping $k$–point rotations. GreeyParallelMMF avoids cascades, but finding each $U_\ell$ involves solving a nontrivial matching problem between the active rows/columns of $A_{\ell-1}$. In the case of $k = 2$, the matching can be done in $O(n^3)$ time by the so-called Blossom Algorithm (Edmonds, 1965), however for, $k > 2$ it becomes forbiddingly expensive, even for a small matrix size $n$.

## 7.2 Parallel MMF (pMMF)

The Parallel MMF (pMMF) algorithm, described in this chapter, gets around the limitations of previous MMF algorithms by employing a two level factorization strategy. The high level picture is that $A$ is factored in the form

$$A \approx \overline{U}_1^\top \overline{U}_2^\top \dots \overline{U}_P^\top H \overline{U}_P \dots \overline{U}_2 \overline{U}_1, \tag{7.2}$$

where each $\overline{U}_p$, called the $p$–th **stage** of the factorization, is a compound rotation (Definition 6), which is block diagonal in the following generalized sense.

**Definition 9.** *Let $M \in \mathbb{R}^{n\times n}$, and $B_1 \cup B_2 \cup \dots \cup B_m$ be a partition of $[n]$. We define the $(u, v)$ **block** of $M$ (with respect to the partition $(B_1, B_2, \dots B_m)$) as the submatrix*

$$[M]_{u,v} := M_{B_u,B_v}. \tag{7.3}$$

*We say that $M$ is $(B_1, B_2, \ldots B_m)$–**block diagonal** if $[M]_{u,v} = 0$ unless $u = v$.*

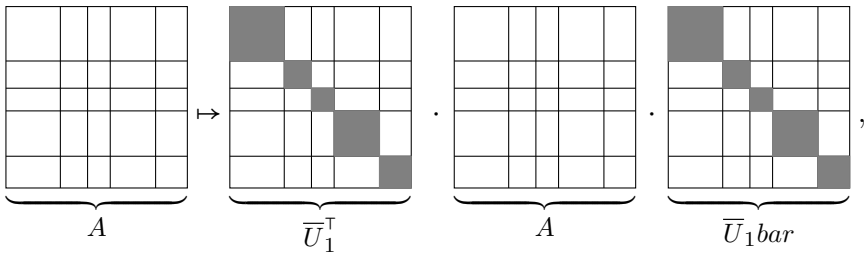
Each stage $\overline{U}_p$ itself, in turn, factors in the form

$$\overline{U}_p = U_{\ell_p} \ldots U_{\ell_{p-1}+2} U_{\ell_{p-1}+1} \tag{7.4}$$

into a product of (typically, many) elementary rotations obeying the constraints described in detail in Section 5.1. Thus, expanding each stage, the overall form of (7.2) becomes

$$A \approx \underbrace{U_1^\top \ldots U_{l_1}^\top}_{\overline{U_1}^\top} \underbrace{U_{l_1+1}^\top \ldots U_{l_2}^\top}_{\overline{U_2}^\top} \cdots\cdots \underbrace{U_{l_P}^\top}_{\overline{U_P}^\top} H \underbrace{U_{l_P} \cdots\cdots}_{\overline{U_P}} \underbrace{U_{l_2} \ldots U_{l_1+1}}_{\overline{U_2}} \underbrace{U_{l_1} \ldots U_1}_{\overline{U_1}}, \tag{7.5}$$

which is identical to the original MMF formulation (5.10), except for the additional constraint that the elementary rotations $U_1, \ldots, U_L$ are forced into contiguous runs $U_{\ell_{p-1}+1}, \ldots, U_{\ell_p}$ conforming to the same block structure. The structure of a single pMMF stage can be graphically represented as




where the block diagonal partitioning (which would sometimes be called **clustering**) that $\bar{U}_1$ conforms to is shown in gray.

Algorithm 4 (PMMF) presents the top level pseudocode for the stage–by–stage pMMF factorization. The FindRotInCluster algorithm called inside Algorithm 4 (line 9) corresponds to finding the elementary rotations within each stage (7.4). Overall, pMMF relies on two main strategies which allow for its scalability to large matrices:

(i) **clustering/blocking**: The blocking of each stage $\bar{U}_p$ allows for parallelization of the factorization,

**Algorithm 4** PMMF(A): top level of the pMMF factorization

U

**Input**: a symmetric matrix $A \in \mathbb{R}^{n \times n}$
**Set** $A_0 \leftarrow A$
**for** $(p = 1$ to $P)$ {
  **cluster** the active columns of $A_{p-1}$ to blocks $(B_1^p, \ldots, B_m^p)$
  **reblock** $A_{p-1}$ according to $(B_1^p, \ldots, B_m^p)$
  **for** $(u = 1$ **to** $m)$ $[\bar{U}_p]_{u,u} \leftarrow$ **FindRotInCluster**$(p, u)$
  **for** $(u = 1$ **to** $m)$ {
    **for** $(v = 1$ **to** $m)$ {
      **set** $[A_p]_{u,v} \leftarrow [\bar{U}_p]_{u,u}[A_{p-1}]_{u,v}[\bar{U}_p]_{v,v}^\top$
}}}
$H \leftarrow$ the core of $A_L$ plus its diagonal
**Output**: $(H, \bar{U}_1, \ldots, \bar{U}_p)$

(ii) **randomization**: FindRotInCluste finds the best rotations within each stage in a randomized fashion. The use of randomization makes it similar to some of the other matrix sketching algorithms (see Chapter 6) in the sense that they also use random sampling to perform certain computations approximately, as a replacement of exact numerical methods.

Below we elaborate on these strategies and describe some other useful properties of pMMF.

### *7.2.1 Clustering*

The key issue in pMMF is determining how to block each pMMF stage so that the block diagonal constraints imposed by (7.2) will have as little impact on the quality of the factorization as possible (measured in terms of the residual (5.28) or some other notion of error). On one hand, it is integral to the idea of multiresolution analysis that at each stage the transform must be *local*. In the case of MMF this manifests in the fact that a given row/column of $A$ will tend to only interact with other rows/columns that have high normalized inner product with it. This suggests blocking $A$ by clustering its rows/columns by normalized

**Algorithm 5** FINDROTINCLUSTER(P,U): subroutine for finding the optimal rotations to perform in each block in a randomized fashion. Here we assume $k = 2$ and a compression ratio $0 < \eta \leq 1$, which is the number of rotations to perform in each cluster/channel in each stage as a fraction of the size of the cluster.

**Input:** The matrix $\mathcal{A} = [A_p]_{:,B_u} \in \mathbb{R}^{n \times c}$ made up of the $c$ columns of $A_{p-1}$ forming cluster $u$ in $A_p$
**compute** the Gram matrix $G = \mathcal{A}^\mathsf{T}\mathcal{A}$
**set** $I = [c]$ (the active set)
**for** ($s = 1$ **to** $\lfloor \eta c \rfloor$){
  **select** $i \in I$ uniformly at random
  **find** $j = \text{argmax}_{I \setminus \{i\}} \left| \langle \mathcal{A}_{:,i}, \mathcal{A}_{:,j} \rangle \right| / \|\mathcal{A}_{:,j}\|$
  **find** the Givens rotation $g_s = U_{i,j,\theta}$ as described in Section 7.2.3
  **set** $\mathcal{A} \leftarrow g_s \mathcal{A} g_s^\mathsf{T}$
  **set** $G \leftarrow g_s G g_s^\mathsf{T}$
  # eliminate coordinate $i$ or $j$ from the active set
  **if** $\left\| \mathcal{A}_{i,:} \right\|_{\text{off-diag}} < \left\| \mathcal{A}_{j,:} \right\|_{\text{off-diag}}$ {
    **set** $I \leftarrow I \setminus \{i\}$}
  **else set** $I \leftarrow I \setminus \{j\}$
}
**Output:** $[\bar{U}_p]_{u,u} = g_{\lfloor \eta c \rfloor} \cdots g_2 g_1$

inner product. For speed, pMMF employs a randomized iterative clustering strategy with additional constraints on the cluster sizes to ensure that the resulting block structure is close to balanced.

On the other hand, enforcing a single block structure (i.e., using the same partitioning, as implied by Definition 9) on all the stages would introduce artificial boundaries between the different parts of $A$ and prevent the algorithm from discovering the true multiresolution structure of the matrix. Therefore, pMMF reclusters the rows/columns of $A_\ell$ before each stage and reblocks the matrix accordingly. The first and crucial step towards pMMF is to cluster the rows/columns of $A$ into $m$ clusters and only consider rotations between $k$–sets of rows/columns that belong to the same cluster. Letting $B_u$ be the indices of the rows/columns belonging to cluster $u$, clustering has three benefits

(i) Instead of having to compute the full Gram matrix $G = A^\mathsf{T}A$, it is sufficient to compute the *local* Gram matrices $\{G^u = [A_p]_{:,B_u}{}^\mathsf{T}[A_p]_{:,B_u}\}_{u=1}^m$ (note that $G^u$ corresponds to

block $u$ of a given stage $p$). Assuming that the clustering is even, i.e., $|B_u| = \Theta(c)$ for some typical cluster size $c$ (and therefore, $m = \Theta(n/c)$), this reduces the overall complexity of computing the Gram matrices from $O(n^3)$ to $O(mc^2n) = O(cn^2)$.

(ii) The complexity of the index search problem in the GreedyJacobiMMF algorithm (the second bottleneck described in Section 7.1) is reduced from $O(n^k)$ to $O(c^k)$. In typical MMFs $\delta_L = O(n)$ and the total number of rotations $L$ scales linearly with $n$. Therefore, the total complexity of searching for rotations in the unclustered case is $O(n^{k+1})$, whereas with clustering it is $O(c^k n)$.

(iii) The computation of the different Gram matrices, as well as the rotations of the different clusters, are completely decoupled and therefore, on a machine with at least $m$ cores, they can be computed in parallel, reducing the computation time of the above to $O(cn^2/m) = O(c^2 n)$ and $O(c^k n/m) = O(c^{k+1})$, respectively.

### *7.2.2 Blocked Matrices*

Moreover, in a given cluster $u$ of a given stage $p$ the local Gram matrix $G^u$ and the rotations can be determined from the columns belonging to just that cluster. However, subsequently, these rotations need to be applied to the entire matrix, from both the right and the left, which cuts across clusters. To be able to perform this part of the algorithm in parallel as well, we partition $A$ not just column-wise, but also row-wise. The resulting data structure is called a symmetrically blocked matrix (c.f., Buluç and Gilbert (2012)).

**Definition 10.** *Given a matrix $A \in \mathbb{R}^{n\times n}$ and a partition $B_1 \cup B_2 \cup \ldots \cup B_m$ of $n$, the $(u, v)$ block of $A$ is the submatrix $A_{u,v} := A_{B_u,B_v}$. The **symmetric blocked matrix** form of $A$ consists of the $m^2$ separate matrices $\{A_{u,v}\}_{u,v=1}^{m}$.*

In pMMF, the matrix $A_\ell$ is always maintained in blocked matrix form, where the block structure is dictated by the clustering of the current stage. pMMF reclusters the

rows/columns of $A_\ell$ before each stage, and reblocks the matrix accordingly. For large matrices, the individual blocks can be stored on separate cores or separate machines and all operations, including computing the Gram matrices, are performed in a block–parallel fashion. As a result of maintaining $A$ in a symmetric blocked blocked matrix form, finding the rotations (line 9 of Algorithm 4) as well as applying the rotations (lines $10-12$ of Algorithm 4) can be naively parallelized. Assuming $m^2$–fold parallelism, the time complexity of the Gram matrix computation is further reduced from $O(cn^2)$ to $O(c^3)$. Similarly, assuming $m^2$–fold parallelism and a total of $\eta c$ rotations in stage $p$, the overall time needed to apply all of these rotations to the entire matrix scales with $O(\eta k c^2)$ (for dense matrices). The corresponding complexities for sparse matrices are shown in Table 7.1.

The blocked matrix data structure is ideally suited to carrying out each $\bar{U}_p$ stage of pMMF on a parallel system because (except for summary statistics) no data needs to be communicated between the different blocks. However, changing the block structure of the matrix from one clustering to another can incur a large communication overhead. It is critical that the reblocking process is done efficiently, maintaining parallelism, without ever having to push the entire matrix through a single processor or a single machine. To retain $m$–fold parallelism, the reblocking is carried out in two phases: first, each column of blocks in the original matrix is reblocked row-wise in parallel, then each row of blocks in the resulting matrix is reblocked column-wise in parallel. Figure 7.1 illustrates the reblocking process – note that the columns belonging to each block are not necessarily contiguous (as the block structure can be easily maintained by keeping an index list), however, the figure assumes, for the sake of visual clarity, that an appropriate permutation is applied to the matrix to reorder it into blocks of contiguous columns. One of the main reasons why we decided to implement dense/sparse blocked matrix classes in the pMMF software library (described in the following section) was that we were not aware of any existing matrix library with this blocking functionality.

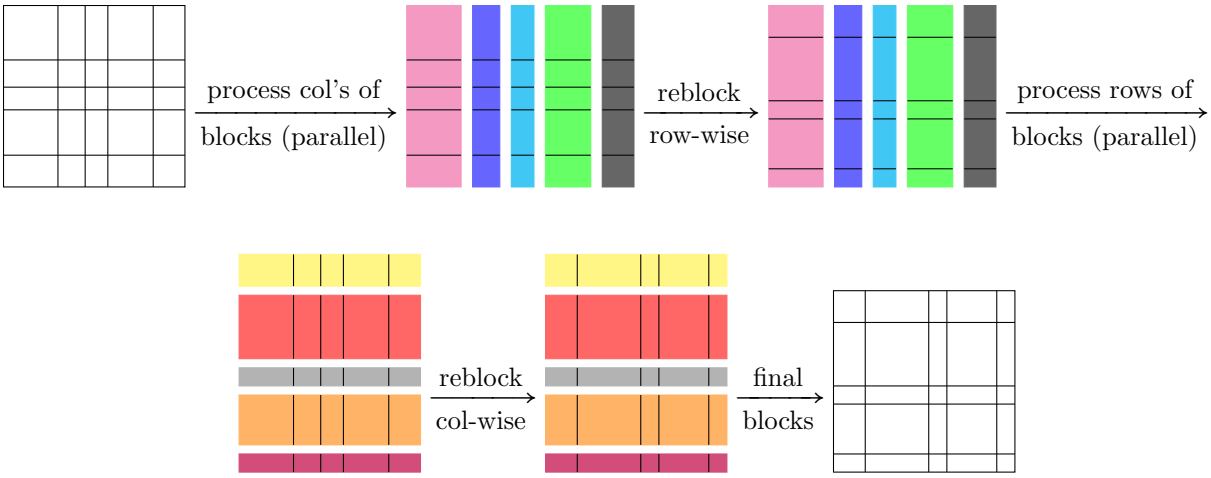



Figure 7.1: **pMMF reblocking schematic**. Illustration of the two stage reblocking strategy used by the pMMF algorithm. The reblocking process starts with a blocked matrix with $5 \times 5$ blocks which are reblocked in another set of $5 \times 5$ blocks. For the sake of visual clarity, here we assume that the blocks are contiguous, but this is generally not the case. The reblocking process involves reorganizing the rows according to the new structure (top panel), then reorganizing the columns of the resulting matrix (bottom panel). To perform this efficiently, the first operation is done in parallel for each column of blocks (shown in different colors) of the original matrix, and the second operation is done in parallel for each row of blocks (shown in different colors) of the resulting matrix.

### *7.2.3 Randomized Greedy Search for Rotations*

Even with blocking, assuming that the characteristic cluster size is $c$, the $O(c^k)$ complexity of finding the indices involved in each rotation by the GreedyJacobiMMF strategy is a bottleneck. To address this problem, pMMF uses randomization. First a single row/column $i_1$ is chosen from the active set (within a given cluster) uniformly at random, and then $k-1$ further rows/columns $i_2, \ldots, i_k$ are selected from the same cluster according to some separable objective function $\phi(i_2, \ldots, i_k)$ related to minimizing the contribution to the final error. For simplicity, in pMMF we use

$$\phi(i_2, \ldots, i_k) = \sum_{r=2}^{k} \frac{\left\langle [A_{\ell-1}]_{:,i_1}, [A_{\ell-1}]_{:,i_r} \right\rangle}{\| [A_{\ell-1}]_{:,i_r} \|},$$

i.e., $[A_{\ell-1}]_{:,i_1}$ is rotated with the $k-1$ other columns that it has the highest normalized inner product (in absolute value) with. Similarly to MMF, the actual rotation angle (or, in the

case of $k$–order rotations, the $k \times k$ nontrivial submatrix of $U_\ell$) is determined by diagonalizing $[G_\ell]_{(i_1 \ldots i_k),(i_1 \ldots i_k)}$ (7.1) at a cost of $O(k^3)$. In the case of $k = 2$, the rotations involved in each pMMF stage are Givens rotations, just like in the MMF algorithms in Section 5.2

This aggressively randomized, greedy strategy reduces the complexity of finding each rotation to $O(c)$, and in our experience does almost as well as exhaustive search of rotations (using matching in Algorithm 2). In fact, even for $k = 2$ the quality of the pMMF approximation is comparable to that of performing MMF with the exhaustive search of rotations. The criterion for elimination is minimal off-diagonal norm, $\|A_{:,i}\|_{\text{off-diag}} = (\|A_{:,i}\|^2 - A_{i,i}^2)^{1/2}$, because $2\|A_{:,i}\|^2_{\text{off-diag}}$ is the contribution of eliminating row/column $i$ to the residual. As a by–product, randomization also eliminates the cascade problem of the GreedyJacobiMMF algorithm, mentioned in Section 7.1. At the same time the randomization does not significantly affect the quality of the matrix approximation and in practice pMMF achieves comparable approximation error as the MMF algorithms from the previous chapter (for empirical evaluation of pMMF see Section 7.4).

### *7.2.4 Sparsity and Matrix Free MMF Arithmetic*

Similarly to the other MMFs, pMMF can be used for approximating (dense or sparse) matrices, albeit significantly more efficiently than other MMF algorithms do. Naturally, the $U_\ell$ elementary rotations are always stored in sparse form, in fact, for maximal efficiency, in the pMMF library they are implemented as separate, specialized objects (rather than explicitly as dense matrices). However, when $n$ exceeds several thousand, even just storing the original matrix in memory becomes impossible unless it is sparse. Therefore, maintaining sparsity during the different subtasks of the pMMF factorization process is critical. As the factorization progresses, due to the rotations, the fill-in (i.e., fraction of nonzero entries) in $A_p$ will increase. Fortunately, at the same time, the active part of $A_\ell$ shrinks, and assuming multiresolution structure, these two factors balance each other out. In practice, we observe

| | serial MMF | | pMMF operations | | pMMF time | | $N_{\text{proc}}$ |
|---|---|---|---|---|---|---|---|
| | dense | sparse | dense | sparse | dense | sparse | |
| Computing Gram matrices | $n^3$ | $\gamma n^3$ | $Pcn^2$ | $\gamma Pcn^2$ | $Pc^3$ | $\gamma Pc^3$ | $m^2$ |
| Finding Rotations | $n^3$ | $n^3$ | $cn$ | $cn$ | $c^2$ | $c$ | $m$ |
| Updating Gram matrices | $n^3$ | $\gamma^2 n^3$ | $c^2 n$ | $\gamma^2 c^2 n$ | $c^3$ | $\gamma^2 c^3$ | $m$ |
| Applying rotations | $kn^2$ | $\gamma kn^2$ | $kn^2$ | $\gamma kn^2$ | $kc^2$ | $\gamma kc^2$ | $m^2$ |
| Clustering | | | $pmn^2$ | $\gamma pmn^2$ | $pcn$ | $\gamma pcn$ | $m^2$ |
| Reblocking | | | $pn^2$ | $\gamma pn^2$ | $pcn$ | $\gamma pcn$ | $m$ |
| Factorization total | $n^3$ | $n^3$ | $Pcn^2$ | $\gamma Pcn^2$ | $Pc^3$ | $\gamma Pc^3$ | $m^2$ |

Table 7.1: **pMMF complexity**. The rough order of complexity of different subtasks in pMMF vs. the serial greedy MMF algorithm (Algorithm 1). Here $n$ is the dimensionality of the original matrix $A$, $k$ is the order of the rotations, and $\gamma$ is the fraction of nonzero entries in $A$, when $A$ is sparse. We neglect that during the course of the computation $\gamma$ tends to increase because concomitantly $A_\ell$ shrinks, and computation time is usually dominated by the first few stages. We also assume that entries of sparse matrices can be accessed in constant time. In pMMF, $P$ is the number of stages, $m$ is the number of clusters in each stage, and $c$ is the typical cluster size (thus, $c = \theta(n/m)$). The "pMMF time" columns give the time complexity of the algorithm assuming an architecture that affords $N_{\text{proc}}$–fold parallelism. It is assumed that $k \leq P \leq c \leq n$, but $n = o(c^2)$. Note that in the simplest case of Givens rotations, $k = 2$.

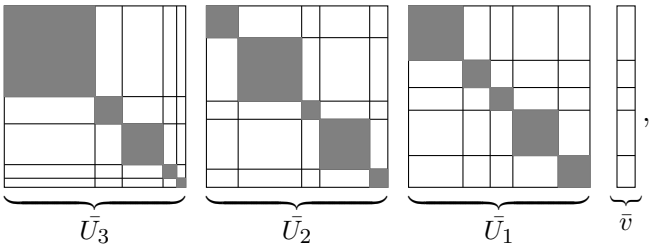
that for most large sparse datasets, given a sufficiently highly parallel system, the overall complexity of pMMF scales close to linearly with the number of nonzeros in $A$, both in space and in time.

Let us denote the complete factorization appearing on the right hand side of (7.5) by $\tilde{A}$. Even if the original matrix $A$ was sparse, in general, $\tilde{A}$ will not be, therefore computing it explicitly is unfeasible. However, many downstream applications, e.g., iterative methods, almost never need $\tilde{A}$ itself, but only need the result of applying $\tilde{A}$ (or e.g., $\tilde{A}^{-1}$) as an operator to vectors.

Therefore, in order to compute $\tilde{A}v$ for a given $v$, we use what in numerical analysis is called the matrix–free approach — we never compute $\tilde{A}$ explicitly, rather, $v$ is stored in the same blocked form as $A$, the rotations are applied individually, and as the different stages are applied to $v$, the vector goes through an analogous reblocking process to that described

for $A$. Graphically, this process can be represented as

$$\tilde{A}v = \bar{U}_1^\top \bar{U}_2^\top \bar{U}_3^\top \dots \bar{U}_P^\top H \bar{U}_P \dots$$




where to multiply $v$ by a given stage $\bar{U}_p$, if $v$ is blocked the same way, $[v]_u$ needs to be multiplied only by $[\bar{U}_p]_{u,u}$. The complexity of matrix–free MMF–vector multiplication is $O(kPn)$. The blocking within each stage makes it possible to distribute the matrix-vector computation over $m$ different processors.

In addition to speeding up matrix–vector products, compressing matrices with pMMF can yield huge savings for certain downstream computations such as matrix inversion. For example, in ridge regression or Gaussian process inference computational efficiency becomes important when computing $K^{-1}v$, where $K$ is typically a large, dense kernel matrix.

The approximate matrix inverse $\tilde{A}^{-1}$ is trivial to compute as follows

$$\tilde{A}^{-1} \approx U_1^\top \dots U_{L-1}^\top U_L^\top H^{-1} U_L U_{L-1} \dots U_1,$$

where inverting the core–diagonal matrix $H$ involves inverting the entries on the diagonal of $H$ and inverting the core of $H$. Since the core of $H$ is only $\delta_L$–dimensional ($\delta_L \ll n$), $H^{-1}$ can be computed in $O(n + \delta_L^3)$ time, which is negligible compared to the $O(n^3)$ cost of inverting $A$ exactly. Similarly, to compute the matrix exponential, pMMF can be used the same way.

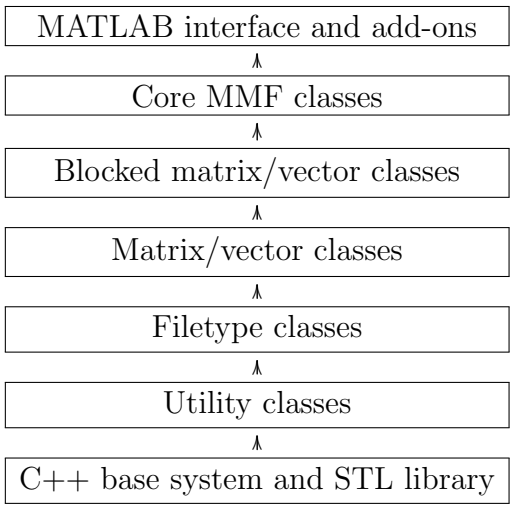
The theoretical complexity of the various pMMF subtasks are summarized in Table 7.1. Of course, requiring $m^2$–fold parallelism (see last column of Table 7.1) as $m \to \infty$ is an abstraction. Note, however, that for sparse matrices, even the total operation count scales with $\gamma n^2$, which is just the number of nonzero elements in $A$.

| | $n$ | $d$ | nnz |
|---|---|---|---|
| Laplacian kernels (Leskovec and Krevl, 2014; Davis and Hu, 2011) | | | |
| **web-NotreDame**: web graph of the University of Notre Dame | $325,729$ | - | $1,497,134$ |
| **soc-Epinions1**: who-trusts-whom network of Epinions.com | $131,828$ | - | $841,372$ |
| **Gnutella31**: peer-to-peer network from August 31, 2002 | $62,586$ | - | $147,892$ |
| **Enron**: Enron email graph | $36,692$ | - | $367,662$ |
| **as-caida**: CAIDA AS Relationships Datasets | $31,379$ | - | $106,762$ |
| **CondMat**: ArXiv condensed matter collaboration graph | $23,133$ | - | $186,936$ |
| **AstroPh**: ArXiv astrophysics collaboration graph | $18,772$ | - | $396,160$ |
| **HEPph**: ArXiv high energy physics collaboration graph | $12,008$ | - | $237,010$ |
| **HEPth**: ArXiv high energy physics theory collaboration graph | $9,877$ | - | $51,971$ |
| **Gnutella06**: peer-to-peer network from August 6, 2002 | $8,717$ | - | $31,525$ |
| **GR**: ArXiv General Relativity collaboration graph | $5,242$ | - | $28,980$ |
| Linear kernels (Bache and Lichman, 2013) | | | |
| **dexter**: bag of words dataset | $2,000$ | 20 | $4\times10^6$ |
| **protein**: feature matrix for *S. cerevisiae* proteins | $6,621$ | 357 | $\approx44\times10^6$ |
| **SNPs**: cancer DNA microarray data | $5,520$ | 43 | $\approx30\times10^6$ |
| **gisette**: handwritten digit images | $6,000$ | $5,000$ | $36\times10^6$ |
| RBF kernels(Bache and Lichman, 2013) | | | |
| **abalone**: physical measurements of abalones ($\sigma=0.15, 1.0$) | $4,177$ | 8 | $\approx17\times10^6$ |
| **wine Quality**: wine physicochemical tests ($\sigma=1.0, 2.1$) | $4,898$ | 12 | $\approx24\times10^6$ |

Table 7.2: **pMMF datasets**. Summary of the datasets used in the pMMF compression experiments (Bache and Lichman, 2013; Leskovec and Krevl, 2014; Davis and Hu, 2011). All datasets are symmetric matrices of size $n \times n$; nnz denotes the number of nonzero entries in each dataset. For the Laplacian kernels, $n$ reflects the number of vertices in the dataset. The linear and RBF kernel matrices are constructed from $n$ data points in $\mathbb{R}^d$. For RBF kernels $\sigma$ is the width of the kernel.

## 7.3 pMMF Implementation: The pMMF Software Library

We implemented pMMF in C++11 with the goal of building a general purpose library that scales to large matrices. The pMMF C++ software library is distributed under the GNU Public License v.3.0 (Kondor et al., 2015b). Critical to the implementation are the data structures used to store blocked vectors and matrices, which must support: (i) sparsity, (ii) block level parallelism, (iii) fast multiplication by Givens rotations and $k$–point rotations from both the left and the right, (iv) fast computation of inner products between columns (and of Gram matrices), (v) fast reblocking, (vi) fast matrix/vector multiplication. Since we could not find any existing matrix library fulfilling all these requirements, we implemented the blocked matrix data structure from scratch directly on top of the `stl::vector`, `stl::list` and `stl::hash_map` containers. The pMMF library is made up of: (a) the pMMF base system, consisting of all the classes involved in the mechanics of computing MMF factorization, and (b) various add–on modules, such as the MATLAB interface. The library's classes form the following hierarchy




The main parameters of pMMF are the order of the rotations $k$, the number of stages $P$, the target number of clusters per stage $m$, and the compression ratio $\eta$, which is the number of rotations to perform in a given cluster, as a fraction of the cluster size. In most of our

experiments, a single row/column is removed from the active set after each rotation. For computational efficiency, we use a rough randomized clustering method, which selects $m$ "anchors columns" uniformly at random from the active set, and clusters the remaining columns to the anchor column with which is has highest normalized inner product. This amounts to greedily assigning the remaining columns in the active set to one of the $m$ clusters until we run out of active columns. The clustering method also has additional parameters intended to ensure that the clustering is approximately even — this is important for the parallelization of the algorithm because, if one of the clusters is significantly larger than the other ones, the operations that block is involved in will result in a bottleneck. If the size of any of the clusters falls below $c_{\min}$, then its columns are redistributed amongst the remaining clusters, whereas if the size of any of the clusters exceeds $c_{\max}$, then its columns are recursively reclustered using the same algorithm. The maximum recursion depth is another parameter $D_{\max}$. There is also a bypass flag, which, when set, signifies that rows/columns which could not be successfully clustered using the above method at a given stage will simply bypass the stage, with no rotations applied to them. Setting these parameters on a given system requires some experience, but overall our results appear stable w.r.t. the parameter values (including $P$, $m$ and $\eta$, most importantly), as long as they are in a reasonable range.

## 7.4 pMMF Experiments

To evaluate the performance of pMMF and contrast it to that of other matrix approximation algorithms, we used a variety of real datasets commonly used in the matrix sketching literature. Table 7.2 provides a brief description and summary statistics of the data used in the experiments in this section. Broadly, each dataset falls into one of three groups:

(i) **undirected graph/network data** (in the form of Laplacian kernels). Most of these datasets are real social networks and as such exhibit a strong hierarchical organization (Leskovec et al., 2009; Fortunato and Barthelemy, 2007). These matrices also have high

degree of sparsity and by definition are not low rank matrices. Some of the smallest datasets in this group were previously used by Gittens and Mahoney (2013) to evaluate the performance of low rank matrix sketches — for those datasets we followed their protocol when setting the parameters of the low rank matrix sketches. In order to evaluate the speed and approximation accuracy of pMMF we also selected other, much larger, datasets from The University of Florida Sparse Matrix Collection by Davis and Hu (2011) and the Stanford Large Network Dataset Collection (Leskovec and Krevl, 2014).

(ii) **linear kernels**. Given a set of points $x_1, x_2, \ldots, x_n \in \mathbb{R}^d$, the $A_{i,j}$ entry of the linear kernel matrix $A$ is given by $A_{i,j} = \langle x_i, x_j \rangle$. Linear kernel matrices are typically very low rank (if $d \ll n$) and so, unsurprisingly, low rank approximation methods are expected to perform well on such datasets.

(iii) **radial basis function (RBF) kernels**. Given a set of points $x_1, x_2, \ldots, x_n \in \mathbb{R}^d$, the $A_{i,j}$ entry of the RBF kernel matrix $A$ is given by $A_{i,j} = e^{-\|x_i - x_j\|_2^2/\sigma^2}$. For the two RBF kernel datasets the width parameter $\sigma$ is set as described by Gittens and Mahoney (2013). Note that depending on the width of the RBF kernel, the resulting matrix can have a vastly different structure. Letting $\sigma \to 0$ makes the spectrum of $A$ decay very slowly and as a result the $A$ is not well approximated by low rank sketches. Conversely, letting $\sigma \to \infty$ makes the spectrum of $A$ decay faster (i.e., $A$ is low rank), which results in $A$ being well approximated by low rank sketches.

Note that both the linear and RBF kernels are dense matrices, with all of their entries being nonzero (the third column of Table 7.2 shows the absolute number of nonzero entries in each dataset). When comparing the approximation performance on these two groups of matrices, we set the parameters of the other matrix sketches as specified by Gittens and Mahoney (2013).

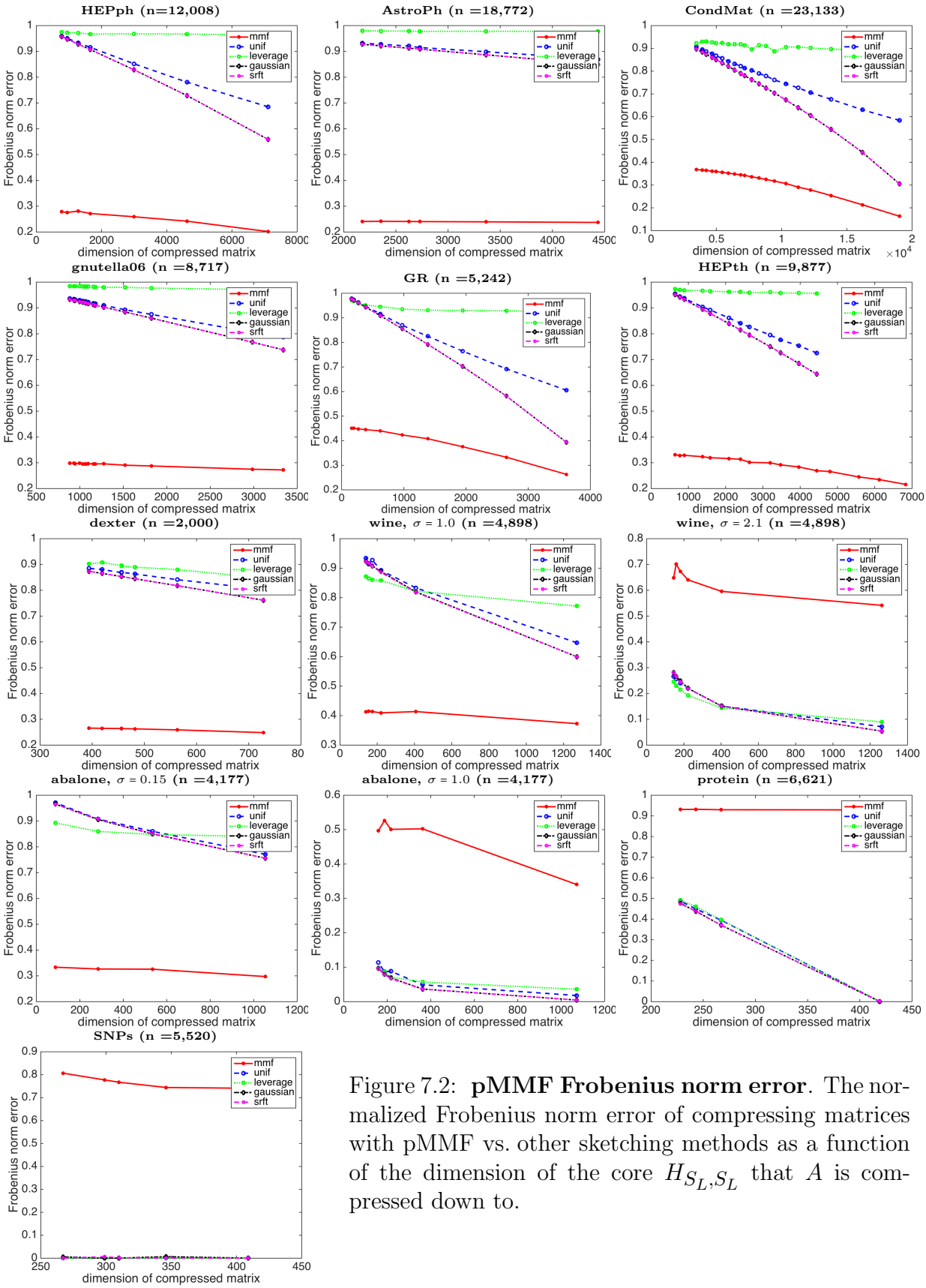



Figure 7.2: **pMMF Frobenius norm error**. The normalized Frobenius norm error of compressing matrices with pMMF vs. other sketching methods as a function of the dimension of the core $H_{S_L,S_L}$ that $A$ is compressed down to.

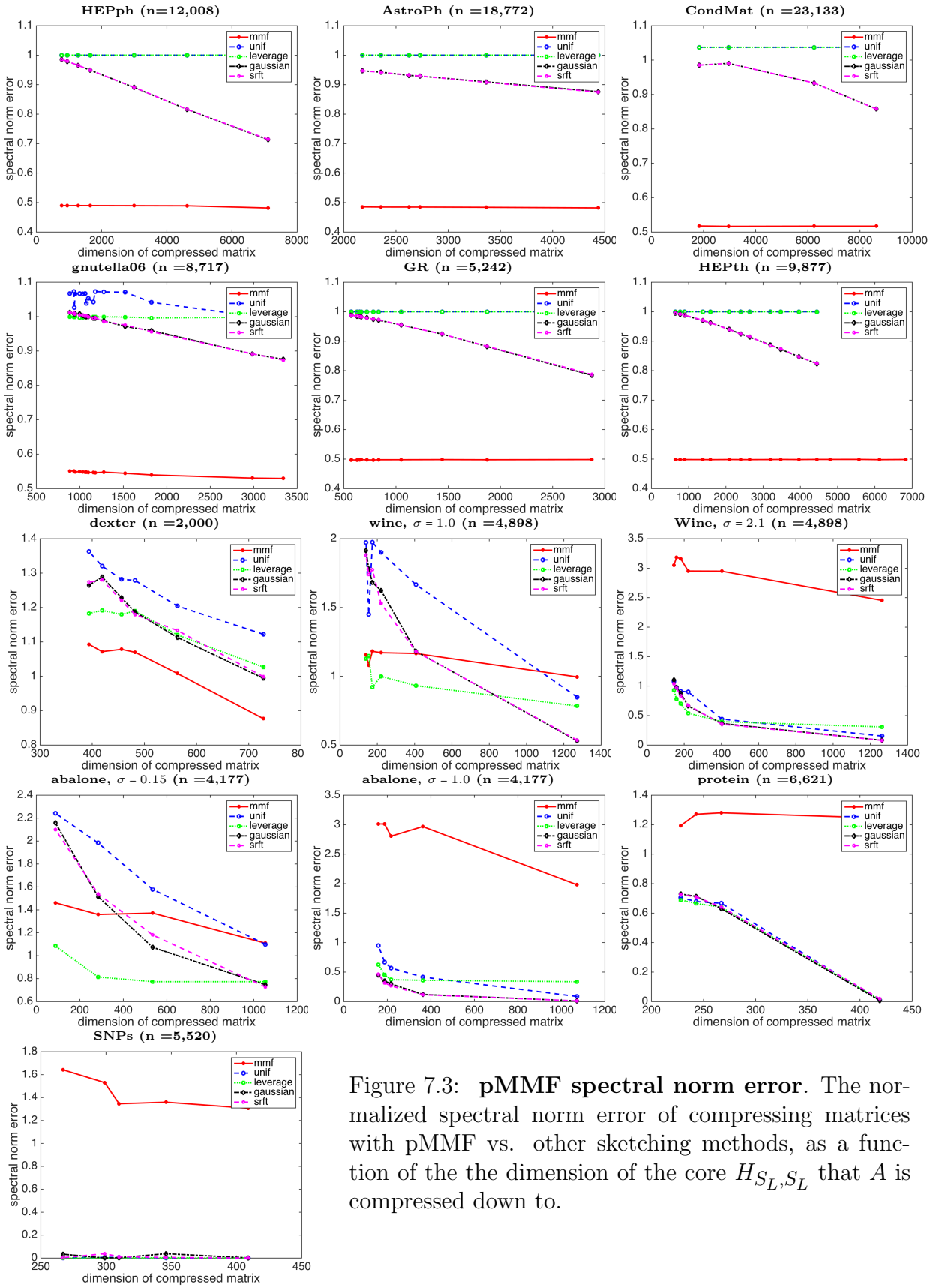



Figure 7.3: **pMMF spectral norm error**. The normalized spectral norm error of compressing matrices with pMMF vs. other sketching methods, as a function of the the dimension of the core $H_{S_L,S_L}$ that $A$ is compressed down to.

### 7.4.1 pMMF Matrix Approximation Quality

To test how well pMMF can approximate a generic data matrix in comparison to other matrix sketching methods, we measure the Frobenius norm error $\mathcal{E}_{\text{Frob}} = \|A - \tilde{A}\|_{\text{Frob}}/\|A - A_r\|_{\text{Frob}}$, as we did earlier in Section 5.6, and the spectral norm error $\mathcal{E}_{\text{Spectral}} = \|A - \tilde{A}\|_2/\|A - A_r\|_2$ on various real data matrices. As before, $A_r$ is the best rank $r$ approximation of $A$ obtained by approximating $A$ with its top $r$ singular vectors/values. Some of these experiments are similar to the ones performed in Section 6.4, except due to the scalability of pMMF are able to test its performance on much larger datasets as well.

Figures 7.2 and 7.3 compare the performance of pMMF to some of the most common matrix sketching algorithms: Nyström with uniform sampling (Williams and Seeger, 2001; Fowlkes et al., 2004), leverage score sampling (Mahoney and Drineas, 2009; Mahoney, 2011; Gittens and Mahoney, 2013), dense Gaussian projections (Halko et al., 2011), and structured randomness with Fourier transforms (Tropp, 2011). In Figures 7.2 and 7.3 these methods are denoted respectively `unif`, `leverage`, `gaussian` and `srft` and a detailed description of these and related methods can be found in Chapter 6. Recall that all of these sketches approximate $A$ in the form $\tilde{A} = CW^{\dagger}C^{\top}$, where $C$ is a judiciously chosen $\mathbb{R}^{n\times\ell}$ matrix with $\ell \ll n$, and $W^{\dagger}$ is the pseudo–inverse of a certain matrix that is of size only $\ell \times \ell$. This approximation is effectively a compression of $A$ down to $\ell$ rows/columns incurring errors $\|A - \tilde{A}\|_{\text{Frob}}$ and $\|A - \tilde{A}\|_2$. For the Nyström compression with leverage score sampling we followed the experiments in Gittens and Mahoney (2013) and used the following target ranks: $r = 5$ for SNPs, $r = 8$ for dexter, $r = 10$ for protein, $r = 20$ for Gnutella06, abalone and wine, $r = 100$ for the rest of the datasets (recall $r$ is the number of top eigenvectors used for computing the leverage scores, see (6.5) in Section 6.3).

On most datasets that we tried, in all of the three groups, pMMF significantly outperforms the other sketching methods in both Frobenius norm error (Figure 7.2) and spectral norm error (Figure 7.3). The advantage of pMMF seems to be particularly great on real graph/network graphs, perhaps not surprisingly, since it has long been conjectured that

networks have multiresolution, clusters–of–clusters structure (Ravasz and Barabási, 2003; Coifman and Maggioni, 2006; Savas et al., 2011). However, we find that pMMF often outperforms other methods on kernel matrices in general. On the other hand, on a small fraction of datasets, typically those which explicitly have low rank or are very close to being low rank, pMMF is outperformed by the other low rank matrix sketching algorithms. For example, in Figure 7.3 and 7.2 pMMF performs subpar in terms of Frobenius and spectral norm error on the Abalone, protein and SNPs datasets. This is likely due to the fact that these datasets are very low dimensional (as Table 7.2 shows, $d$, the number of features in the input space before kernelization, is significantly smaller than $n$ which leads to linear kernel matrices of very low rank). Due to the low rank and the lack of any multiresolution structure, the blocking and randomization subtasks of the pMMF algorithm are not able to capture the low rank structure well enough. The problem is partially alleviated by a combination of a conservative elimination strategy and increasing the ratio of rotations $\eta$ within each cluster, which leads to smaller Frobenius and spectral norm errors for these datasets. However, this essentially amounts to attempting to diagonalize $A$, which is identical to approximating $A$ using its top singular vectors/values. In addition, increasing the ratio of the rotations applied, while restricting elimination, increases in the wall clock time of the pMMF decomposition. Therefore, in cases where the underlying matrix rank is low, a combination of the low rank and multiresolution approaches might be most advantageous.

On RBF kernels pMMF has different performance depending on the width $\sigma$ of the RBF kernel. For example, smaller values of $\sigma$ in the RBF kernel for both the wine and abalone datasets (i.e. $\sigma = 1.0, 0.15$, respectively), lead to better performance of pMMF in comparison to the low rank sketches both in terms of Frobenius and spectral norm error (albeit the latter is somewhat less noticeable). As mentioned at the beginning of Section 7.4, low $\sigma$ results in a slow decay in the spectrum of $A$ and therefore low rank approximations cannot capture the matrix structure well — naturally, this is a favorable scenario for MMF sketching instead.

Additionally, we also tested pMMF approximation on larger datasets for which low rank

| dataset | core size | time (secs) | $\mathcal{E}_{\text{Frob}}$ | $\mathcal{E}_{\text{Spectral}}$ |
|---|---|---|---|---|
| **web-NotreDame** | 1731 | 726.5 | 0.6 | 0.7 |
| **Enron** | 4431 | 530.2 | 0.6 | 0.3 |
| **Gnutella31** | 4207 | 112.2 | 0.8 | 0.6 |
| **as-caida** | 3404 | 91.9 | 0.7 | 0.6 |
| **soc-Epinions1** | 2089 | 1304.4 | 0.5 | 0.5 |

Table 7.3: **pMMF compression on large datasets**. The normalized Frobenius and spectral norm error, time (in seconds), and the dimension of the core $H_{S_L,S_L}$ that $A$ is compressed down to.

sketches are prohibitively expensive to compute (Table 7.3). Note that in Table 7.3, since we do not compare to low rank methods, the errors are normalized in the form $\mathcal{E}_{\text{Frob}} = \|A - \tilde{A}\|_{\text{Frob}}/\|A\|_{\text{Frob}}$ and $\mathcal{E}_{\text{Spectral}} = \|A - \tilde{A}\|_2/\|A\|_2$.

### *7.4.2 pMMF Scalability*

In addition to its approximation accuracy, pMMF has another major advantages — its scalability. pMMF is also dramatically faster than the other methods. This is only partially explained by the fact that several of the competing algorithms were implemented in MATLAB, which limits the size of datasets on which they can be feasibly ran on. For example, Nyström with leverage score sampling (`leverage`) requires estimating the singular vectors of $A$, which, unless $A$ is very low rank, is a computational bottleneck. Several of the Nyström experiments took more than half an hour to run on 8 cores, whereas our custom C++ pMMF implementation compressed the matrix in at most one or two minutes, even the dense datasets which have on the order of $n = 40 \times 10^6$ (i.e. protein, gisette, and SNPs datasets in Figure 7.5). The compression results shown in Table 7.3 could not even be compared to other methods because they could not scale to matrices of this size.

The plots in Figure 7.4 confirm that on many real world datasets, particularly, matrices coming from sparse network graphs, the wall clock time of pMMF scales close to linearly

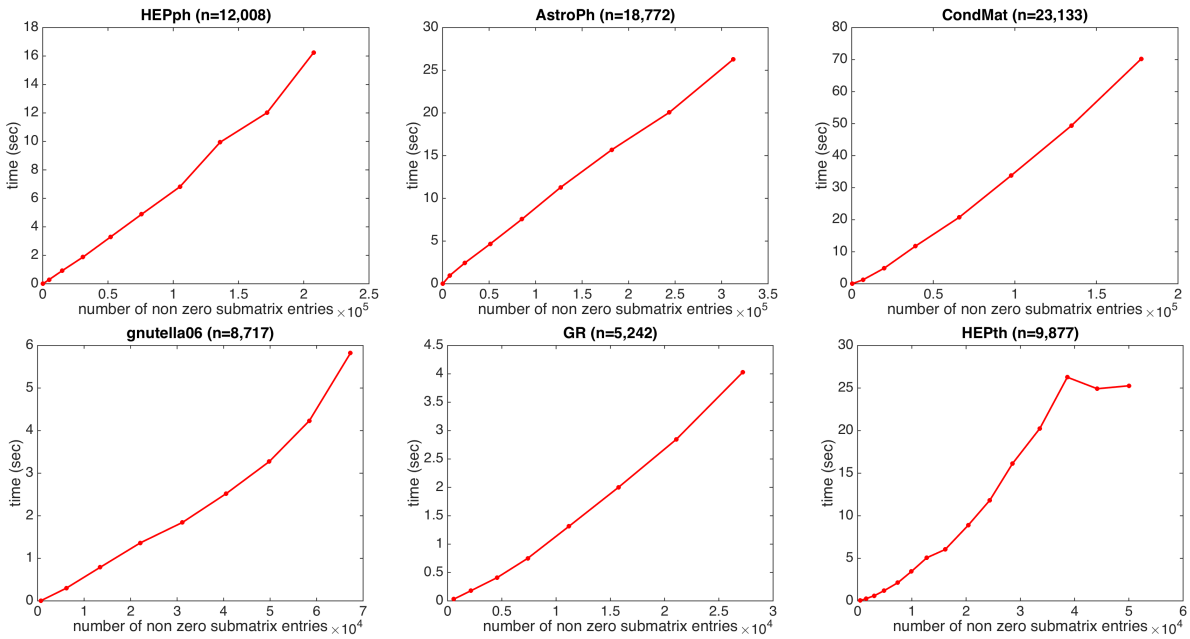



Figure 7.4: **pMMF time vs. sparsity**. Execution time of pMMF as a function of the number of nonzero entries in the input matrix $A$. For each of the graph Laplacians of size $n$, listed in Table 7.2, we take submatrices of varying sizes and compress each of them with pMMF to a $S_L$–core–diagonal matrix with core size of around 100. The $x$ and $y$ axes, respectively, reflect the number of nonzero entries in each of the submatrices and the running time of the MMF compression. Each datapoint is averaged over five runs.

with the dimension $n$. Also note that while in these experiments $n$ is on the order of $10^5$, the factorization time (on a 16–core cluster) is on the order of seconds.

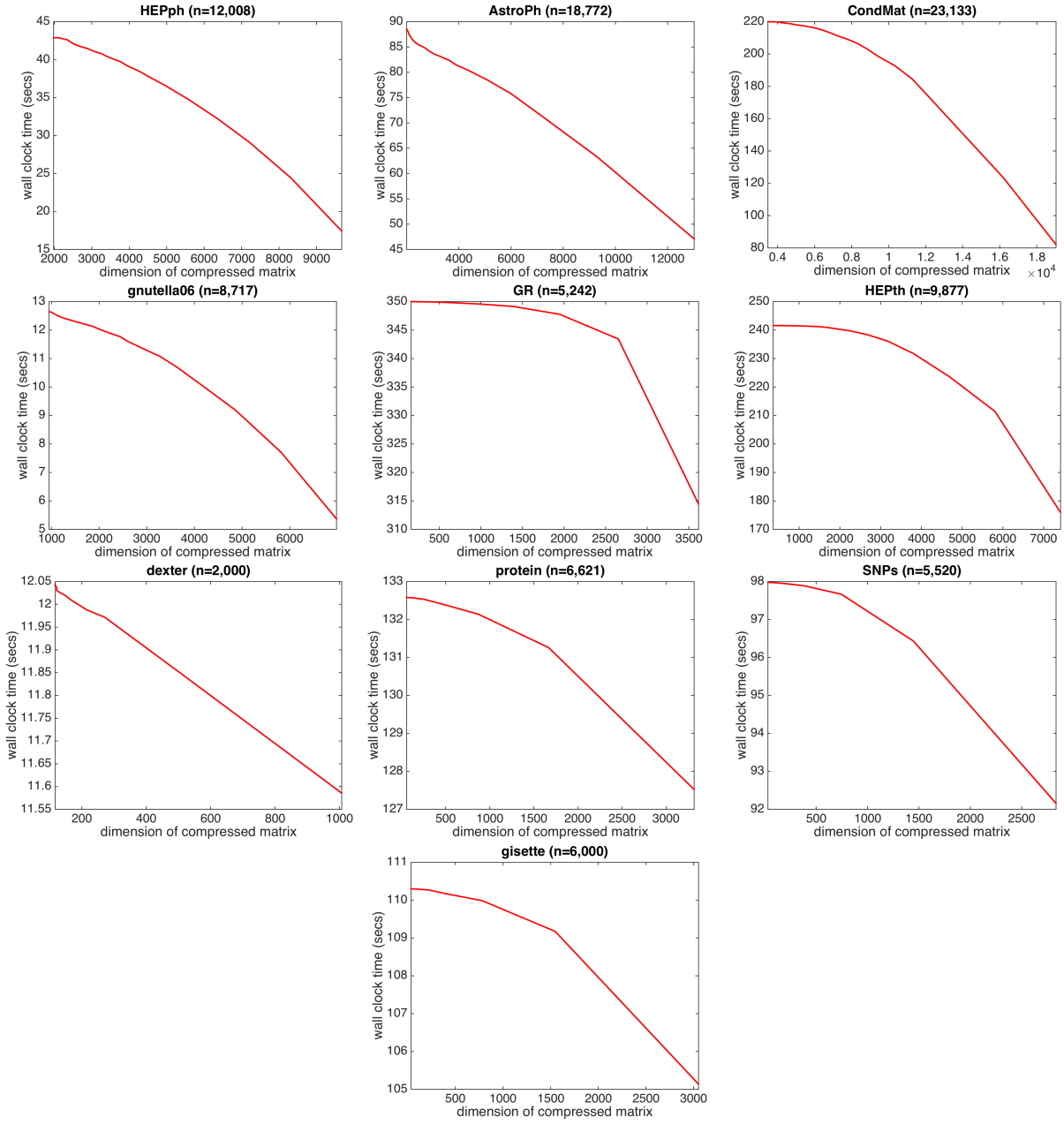



Figure 7.5: **pMMF execution time**. Execution time of pMMF as a function of the size of the compressed submatrix $H_{S_L,S_L}$ on the datasets listed in Table 7.2.

# CHAPTER 8

# CONCLUSION AND CONTRIBUTION SUMMARY

In this thesis we introduced the Multiresolution Matrix Factorization framework and showed how MMFs can be used for compressing matrices. Some of the main results and contributions of this thesis are

- generalizing multiresolution analysis into a matrix factorization problem;
- introducing three algorithms for computing the MMF of symmetric matrices of small to medium size (Chapter 5);
- introducing multiresolution factorizability as an alternative to the low rank assumption in machine learning and numerical analysis (Section 5.3);
- demonstrating that in many cases MMF outperforms current state–of–the–art matrix sketches by a nontrivial margin (Chapter 6);
- introducing a fast, parallel MMF algorithm (pMMF), which scales almost linearly with the number of nonzeros in the matrix being factorized (Chapter 7);
- demonstrating that pMMF is a viable tool for compressing matrices and data at scale.

Naturally, many questions remain unanswered, some of which pertain to the MMF framework itself, while others relate to the challenges associated with generalizing multiresolution analysis to the discrete setting. Possible directions for further research include

- investigating the effect that the underlying graph/matrix structure has on the quality of the MMF approximations and investigating how the clustering technique in pMMF affects the algorithm's performance
- exploring the range of matrix conditions under which MMF is more advantageous than a low rank approximation;

- exploring the range of matrix conditions under which MMFs are more advantageous than low rank approximations;
- further studying the theoretical guarantees of multiresolution factorizability;
- investigating the applicability of MMF for downstream learning algorithms (e.g., can MMF speed up classification without affecting accuracy too much?);
- generalizing uncertainty principles (such as Heisenberg's uncertainty principle, see Section 2.7) to the matrix factorization setting and, more generally, formalizing the meaning of a "good" multiresolution/multiscale basis.

# REFERENCES


Dimitris Achlioptas. Database-friendly random projections: Johnson-Lindenstrauss with binary coins. *Journal of Computer and System Sciences*, 66(4):671–687, June 2003. ISSN 00220000.

Michal Aharon, Michael Elad, and Alfred Bruckstein. K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation. *Signal Processing, IEEE Transactions on*, 54(11):4311–4322, 2006.

Nir Ailon and Bernard Chazelle. The fast Johnson–Lindenstrauss transform and approximate nearest neighbors. *SIAM Journal on Computing*, 39(1):302–322, January 2009. ISSN 0097-5397. doi: 10.1137/060673096.

Nir Ailon and Bernard Chazelle. Faster dimension reduction. *Communications of the ACM*, 53(2):97–104, 2010.

William K Allard, Guangliang Chen, and Mauro Maggioni. Multi-scale geometric methods for data sets II: geometric multi-resolution analysis. *Applied and Computational Harmonic Analysis*, 32(3):435–462, 2012.

Sivaram Ambikasaran. *Fast algorithms for dense numerical linear algebra and applications*. PhD thesis, Stanford University, 2013.

Sivaram Ambikasaran and Eric Darve. The inverse fast multipole method. *arXiv preprint arXiv:1407.1572*, 2014.

Sivaram Ambikasaran and Michael O'Neil. Fast symmetric factorization of hierarchical matrices with applications. *arXiv preprint arXiv:1405.0223*, 2014.

Haim Avron, Petar Maymounkov, and Sivan Toledo. Blendenpik: supercharing LAPACK's least-squares solver. *SIAM Journal on Scientific Computing*, 32, 2010.

K. Bache and M. Lichman. UCI machine learning repository, 2013. URL `http://archive.ics.uci.edu/ml`.

Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

Rick Beatson and Leslie Greengard. A short course on fast multipole methods. *Wavelets, multilevel methods and elliptic PDEs*, 1:1–37, 1997.

Mikhail Belkin and Partha Niyogi. Towards a theoretical foundation for Laplacian-based manifold methods. *Journal of Computer and System Sciences*, 74(8):1289–1308, 2008.

Ake Björck and Gene H Golub. Numerical methods for computing angles between linear subspaces. *Mathematics of computation*, 27(123):579–594, 1973.

Steffen Börm and Jochen Garcke. Approximating Gaussian processes with $H^2$ matrices. In *Machine Learning: ECML 2007*, pages 42–53. Springer, 2007.

Steffen Börm, Lars Grasedyck, and Wolfgang Hackbusch. Hierarchical matrices. *Lecture notes*, 21:2003, 2003.

Wolfgang Hackbusch; Steffen Börm. Data-sparse approximations by adaptive $H^2$ Matrices. *Linear Algebra and its Applications*, 422(2-3):380–403, 2007.

Ulrik Brandes, Marco Gaertler, and Dorothea Wagner. Experiments on graph clustering algorithms. In *European Symposium on Algorithms*, pages 568–579. Springer, 2003.

Achi Brandt. Multi-level adaptive technique (MLAT) for fast numerical solution to boundary value problems. In *Proceedings of the Third International Conference on Numerical Methods in Fluid Mechanics*, volume 18 of *Lecture Notes in Physics*, pages 82–89. Springer Berlin Heidelberg, 1973. ISBN 978-3-540-06170-0. doi: 10.1007/BFb0118663.

William L Briggs, Steve F McCormick, et al. *A multigrid tutorial*. SIAM, 2000.

Aydin Buluç and John R Gilbert. Parallel sparse matrix-matrix multiplication and indexing: implementation and experiments. *SIAM J Sci Comput*, 34(4), 2012.

E. J. Candès and T. Tao. Decoding by linear programming. *IEEE Transactions on Information Theory*, 51(12):4203–4215, Dec 2005.

S. Chandrasekaran, M. Gu, and W. Lyons. A fast adaptive solver for hierarchically semiseparable representations. *Calcolo*, 42(3-4):171–185, 2005.

Guangliang Chen and Mauro Maggioni. Multiscale geometric and spectral analysis of plane arrangements. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 2825–2832. IEEE, 2011.

Scott Shaobing Chen, David L Donoho, and Michael A Saunders. Atomic decomposition by basis pursuit. *SIAM journal on scientific computing*, 20(1):33–61, 1998.

Dehua Cheng, Yu Cheng, Yan Liu, Richard Peng, and Shang-Hua Teng. Scalable parallel factorizations of SDD matrices and efficient sampling for Gaussian graphical models. *arXiv preprint arXiv:1410.5392*, 2014.

F. R. K. Chung. *Spectral Graph Theory*. Number 92 in the Regional Conference Series in Mathematics. AMS, 1997.

Ronald R Coifman and Mauro Maggioni. Multiresolution analysis associated to diffusion semigroups: Construction and fast algorithms. Technical report, Technical report YALE/DCS/TR-1289, Yale University, 2004.

Ronald R. Coifman and Mauro Maggioni. Diffusion wavelets. *Applied and Computational Harmonic Analysis*, 21(1):53–94, July 2006. ISSN 10635203. doi: 10.1016/j.acha.2006.04.004. URL `http://linkinghub.elsevier.com/retrieve/pii/S106352030600056X`.

Andrew Crossett, Ann B. Lee, Lambertus Klei, Bernie Devlin, and Kathryn Roeder. Refining genetically inferred relationships using treelet covariance smoothing. *The Annals of Applied Statistics*, 7(2):669–690, June 2013. ISSN 1932-6157. doi: 10.1214/12-AOAS598. URL `http://projecteuclid.org/euclid.aoas/1372338463`.

Ingrid Daubechies. Orthonormal bases of compactly supported wavelets. *Communications on Pure and Applied Mathematics*, 41(7):909–996, 1988. ISSN 00103640. doi: 10.1002/cpa.3160410705.

Ingrid Daubechies. *Ten lectures on wavelets*. SIAM, 1992.

Timothy A. Davis and Yifan Hu. The University of Florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1):1:1–1:25, December 2011. ISSN 0098-3500. doi: 10.1145/2049662.2049663. URL `http://doi.acm.org/10.1145/2049662.2049663`.

D. Deng and Y. Han. *Harmonic Analysis on Spaces of Homogeneous Type*. Springer, 2009.

Amit Deshpande, Luis Rademacher, Santosh Vempala, and Grant Wang. Matrix approximation and projective clustering via volume sampling. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 1117–1126. ACM, 2006.

P. Drineas and M. W. Mahoney. On the Nyström method for approximating a Gram matrix for improved kernel-based learning. *Journal of Machine Learning Research*, 6:2153–2175, 2005.

Petros Drineas and Michael W Mahoney. A randomized algorithm for a tensor-based generalization of the singular value decomposition. *Linear algebra and its applications*, 420(2):553–571, 2007.

Petros Drineas, Ravi Kannan, and Michael W Mahoney. Fast Monte Carlo algorithms for matrices I-III. *SIAM Journal on computing*, 36(1):158–183, 2006.

Petros Drineas, Michael W. Mahoney, and S Muthukrishnan. Relative-error CUR matrix decompositions. *SIAM Journal on Matrix Analysis and Applicatons*, 30(2):844–881, 2008.

Petros Drineas, Malik Magdon-Ismail, Michael W Mahoney, and David P Woodruff. Fast approximation of matrix coherence and statistical leverage. *The Journal of Machine Learning Research*, 13(1):3475–3506, 2012.

Carl Eckart and Gale Young. The approximation of one matrix by another of low rank. *Psychometrika*, I(3):211–218, 1936.

J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75–174, 2010.

Santo Fortunato and Marc Barthelemy. Resolution limit in community detection. *Proceedings of the National Academy of Sciences*, 104(1):36–41, 2007.

Charless Fowlkes, Serge Belongie, Fan Chung, and Jitendra Malik. Spectral grouping using the Nyström method. *IEEE transactions on pattern analysis and machine intelligence*, 26 (2):214–25, February 2004. ISSN 0162-8828. doi: 10.1109/TPAMI.2004.1262185.

Dennis Gabor. Theory of communication. *Journal of the Institution of Electrical Engineers-Part III: Radio and Communication Engineering*, 93(26):429–441, 1946.

Matan Gavish, Boaz Nadler, and Ronald R Coifman. Multiscale wavelets on trees, graphs and high dimensional data: Theory and applications to semi supervised learning. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 367–374, 2010.

Pieter Ghysels, Xiaoye S Li, François-Henry Rouet, Samuel Williams, and Artem Napov. An efficient multicore implementation of a novel HSS-structured multifrontal solver using randomized sampling. *SIAM Journal on Scientific Computing*, 38(5):S358–S384, 2016.

Domingo Giménez, R Van de Geijn, Vicente Hernández, and Antonio M Vidal. Exploiting the symmetry on the Jacobi method on a mesh of processors. In *Parallel and Distributed Processing, 1996. PDP'96. Proceedings of the Fourth Euromicro Workshop on*, pages 377–384. IEEE, 1996.

Alex Gittens and Michael W Mahoney. Revisiting the Nyström method for improved large-scale machine learning. *J. Mach. Learn. Res*, 28(3):567–575, 2013.

Gene H Golub and Charles F Van Loan. *Matrix computations*, volume 3. JHU Press, 2012.

L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73:325–348, 1987.

Alfred Haar. Zur Theorie der orthogonalen Funktionensysteme. *Mathematische Annalen*, 69 (July):1–37, 1909.

W Hackbusch. A sparse matrix arithmetic based on H-matrices. Part I: Introduction to H-matrices. *Computing*, 62:89–108, 1999.

W. Hackbusch. *Multi-Grid Methods and Applications*. Springer Series in Computational Mathematics. Springer Berlin Heidelberg, 2003. ISBN 9783540127611.

W. Hackbusch and B. N. Khoromskij. A sparse H-matrix arithmetic. Part II: Application to multi-dimensional problems. *Computing*, 64(1):21–47, January 2000. ISSN 0010-485X. URL `http://dl.acm.org/citation.cfm?id=333825.333827`.

Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011.

David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011.

Eldon R Hansen. On cyclic Jacobi methods. *Journal of the Society for Industrial and Applied Mathematics*, 11(2):448–459, 1963.

W. Hardle, G. Kerkyacharian, D. Picard, and A. B. Tsybakov. *Wavelets, Approximation and Statistical Applications*. Lecture notes in statistics; 129. Springer, New York, 1998. ISBN 0387984534.

Matthias Hein, Jean-Yves Audibert, and Ulrike Von Luxburg. From graphs to manifolds–weak and strong pointwise consistency of graph Laplacians. In *Learning theory*, pages 470–485. Springer, 2005.

C. J. G. Jacobi. Über ein leichtes verfahren, die in der theorie der säkularstörungen vorkommenden gleichungen numerisch aufzulösen. *Journal für Reine und Angewandte Mathematik*, 30:51–95, 1846.

R. Jenatton, G. Obozinski, and F. Bach. Structured sparse principal component analysis. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010.

Rodolphe Jenatton, Jean-Yves Audibert, and Francis Bach. Structured variable selection with sparsity-inducing norms. *The Journal of Machine Learning Research*, 12:2777–2824, 2011.

William B Johnson and Joram Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary mathematics*, 26(189-206):1, 1984.

I. T. Jolliffe. *Principal Component Analysis*. Springer-Verlag, New York, 1986.

R. Kondor, N. Teneva, and V. Garg. Multiresolution Matrix Factorization. *International Conference on Machine Learning (ICML)*, 2014.

Risi Kondor, Nedelina Teneva, and Pramod K Mudrakarta. Parallel MMF: a multiresolution approach to matrix computation. *arXiv preprint arXiv:1507.04396*, 2015a.

Risi Kondor, Nedelina Teneva, and Pramod K. Mudrakarta. `pMMF`: a high performance parallel MMF library. Hosted at `https://github.com/risi-kondor/pMMF`, 2015b.

Ioannis Koutis, Gary L. Miller, and David Tolliver. Combinatorial preconditioners and multilevel solvers for problems in computer vision and image processing. *Computer Vision and Image Understanding*, 115:1638–1646, 2011. ISSN 10773142. doi: 10.1016/j.cviu.2011.05.013.

Sanjiv Kumar, Mehryar Mohri, and Ameet Talwalkar. Ensemble Nyström method. In *Advances in Neural Information Processing Systems*, pages 1060–1068, 2009.

Sanjiv Kumar, Mehryar Mohri, and Ameet Talwalkar. Sampling methods for the Nyström method. *Journal of Machine Learning Research*, 13:981–1006, 2012.

Luc Le Magoarou, Rémi Gribonval, and Nicolas Tremblay. Approximate fast graph Fourier transforms via multi-layer sparse approximations. *IEEE Transactions on Signal and Information Processing over Networks*, 2017.

Ann B Lee, Boaz Nadler, and Larry Wasserman. Treelets: an adaptive multi-scale basis for sparse unordered data. *The Annals of Applied Statistics*, pages 435–471, 2008.

Ann B. Lee Lee. Treelet code in MATLAB R2007a. Hosted at `http://www.stat.cmu.edu/~annlee/software.htm`, 2006.

J. Leskovec, K. J. Lang, and M. W. Mahoney. Community structure in large networks: natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1), 2009.

Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. Signed networks in social media. *Proceedings of the 28th international conference on Human factors in computing systems - CHI '10*, page 1361, 2010. doi: 10.1145/1753326.1753532. URL `http://portal.acm.org/citation.cfm?doid=1753326.1753532`.

K Lessel, M Hartman, and Shivkumar Chandrasekaran. A fast memory efficient construction algorithm for hierarchically semi-separable representations. *SIAM Journal on Matrix Analysis and Applications*, 37(1):338–353, 2016.

Michael Lewicki and Terrence Sejnowski. Learning overcomplete representations. *Neural computation*, 12(2):337–365, 2000.

Oren E Livne and Achi Brandt. Lean algebraic multigrid (LAMG): Fast graph Laplacian linear solver. *SIAM Journal on Scientific Computing*, 34(4):B499–B522, 2012.

Mauro Maggioni. Biorthogonal diffusion wavelets for multiscale representations on manifolds and graphs. In *Proceedings of SPIE*, volume 5914, pages 59141M–59141M–13. Spie, 2005. doi: 10.1117/12.616909. URL `http://link.aip.org/link/?PSI/5914/59141M/1&Agg=doi`.

Luc Le Magoarou and Rémi Gribonval. Learning computationally efficient dictionaries and their implementation as fast transforms. *arXiv preprint arXiv:1406.5388*, 2014.

Sridhar Mahadevan and Mauro Maggioni. Value function approximation with diffusion wavelets and Laplacian eigenfunctions. *Advances in neural information processing systems*, 18:843, 2006.

M. W. Mahoney. Randomized algorithms for matrices and data. *Foundations and Trends in Machine Learning*, 3, 2011.

Michael W Mahoney and Petros Drineas. CUR matrix decompositions for improved data analysis. *Proceedings of the National Academy of Sciences*, 106(3):697–702, 2009.

Julien Mairal, Guillermo Sapiro, and Michael Elad. Learning multiscale sparse representations for image and video restoration. *Multiscale Modeling & Simulation*, 7(1):214–241, 2008.

Julien Mairal, Jean Ponce, Guillermo Sapiro, Andrew Zisserman, and Francis R Bach. Supervised dictionary learning. In *Advances in neural information processing systems*, pages 1033–1040, 2009.

Francois Malgouyres and Joseph Landsberg. Stable recovery of the factors from a deep matrix product and application to convolutional network. *arXiv preprint arXiv:1703.08044*, 2017.

Stephane Mallat. *A wavelet tour of signal processing: the sparse way*. Academic press, 2008.

Stephane G Mallat. A theory for multiresolution signal decomposition: the wavelet representation. *IEEE transactions on pattern analysis and machine intelligence*, 11(7):674–693, 1989.

Stéphane G Mallat and Zhifeng Zhang. Matching pursuits with time-frequency dictionaries. *Signal Processing, IEEE Transactions on*, 41(12):3397–3415, 1993.

Per-Gunnar Martinsson. Rapid factorization of structured matrices via randomized sampling. *arXiv preprint arXiv:0806.2339*, 2008.

Per-Gunnar Martinsson. A fast randomized algorithm for computing a hierarchically semiseparable representation of a matrix. *SIAM Journal on Matrix Analysis and Applications*, 32(4):1251–1274, 2011.

Jiří Matoušek. On variants of the Johnson–Lindenstrauss lemma. *Random Structures & Algorithms*, 33(2):142–156, 2008.

Jean Morlet. Sampling theory and wave propagation. In *Issues in Acoustic SignalImage Processing and Recognition*, pages 233–261. Springer, 1983.

Boaz Nadler, Nathan Srebro, and Xueyuan Zhou. Semi-supervised learning with the graph Laplacian: The limit of infinite unlabelled data. In *Proceedings of the 22nd International Conference on Neural Information Processing Systems*, pages 1330–1338. Curran Associates Inc., 2009.

Andrea R Nahmod. Generalized uncertainty principles on spaces of homogeneous type. *Journal of functional analysis*, 119(1):171–209, 1994.

Frank Ong and Michael Lustig. Beyond low rank+ sparse: multiscale low rank matrix decomposition. *IEEE journal of selected topics in signal processing*, 10(4):672–687, 2016.

Bastien Pasdeloup, Réda Alami, Vincent Gripon, and Michael Rabbat. Toward an uncertainty principle for weighted graphs. 2015. URL `http://arxiv.org/abs/1503.03291`.

Nathanael Perraudin, Benjamin Ricaud, David Shuman, and Pierre Vandergheynst. Global and local uncertainty principles for signals on graphs. pages 1–36, 2016. URL `http://arxiv.org/abs/1603.03030`.

William H Press, Saul A Teukolsky, William T Vetterling, and Brian P Flannery. *Numerical recipes in C*, volume 2. Cambridge university press Cambridge, 1996.

Erzsébet Ravasz and Albert-László Barabási. Hierarchical organization in complex networks. *Physical Review E*, 67(2):026112, February 2003. ISSN 1063-651X. doi: 10.1103/PhysRevE.67.026112. URL `http://link.aps.org/doi/10.1103/PhysRevE.67.026112`.

Dorit Ron, Ilya Safro, and Achi Brandt. Relaxation-based coarsening and multiscale graph organization. *Multiscale Modeling & Simulation*, 9(1):407–423, 2011.

Heinz Rutishauser. The Jacobi method for real symmetric matrices. *Numerische Mathematik*, 9(1):1–10, 1966.

Berkant Savas, Inderjit Dhillon, et al. Clustered low rank approximation of graphs in information science applications. In *Proceedings of the SIAM International Conference on Data Mining (SDM)*, 2011.

David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *Signal Processing Magazine, IEEE*, 30(3):83–98, 2013a.

David I Shuman, Benjamin Ricaud, and Pierre Vandergheynst. Vertex-frequency analysis on graphs. *arXiv preprint arXiv:1307.5708*, 2013b.

Daniel Spielman. Spectral graph theory. *Lecture Notes, Yale University*, pages 740–0776, 2009.

Elias M Stein and Rami Shakarchi. *Fourier analysis: an introduction*, volume 1. Princeton University Press, 2011.

Nedelina Teneva, Pramod K Mudrakarta, and Risi Kondor. Multiresolution matrix compression. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, pages 1441–1449, 2016.

Jayaraman J Thiagarajan, Karthikeyan N Ramamurthy, and Andreas Spanias. Multilevel dictionary learning for sparse representation of images. In *Digital Signal Processing Workshop and IEEE Signal Processing Education Workshop (DSP/SPE), 2011 IEEE*, pages 271–276. IEEE, 2011.

R Tibshirani. Regression shrinkage and selection with the Lasso. *Journal of the Royal Statistical Society Series B*, 58(1):267–288, 1996.

Lloyd N Trefethen and David Bau III. *Numerical linear algebra*, volume 50. SIAM, 1997.

Joel A Tropp. Greed is good: Algorithmic results for sparse approximation. *Information Theory, IEEE Transactions on*, 50(10):2231–2242, 2004.

Joel A Tropp. Improved analysis of the subsamples randomized Hadamard transform. *Advances in Adaptive Data Analysis*, 3(1-2):115–126, 2011.

Mikhail Tsitsvero, Sergio Barbarossa, and Paolo Di Lorenzo. Signals on graphs: Uncertainty principle and sampling. *IEEE Transactions on Signal Processing*, 64(18):4845–4860, 2016.

Brani Vidakovic. *Statistical modeling by wavelets*. Wiley series in probability and mathematical statistics. Wiley, New York, 1999. ISBN 978-0-471-29365-1.

Shusen Wang and Zhihua Zhang. Improving CUR matrix decomposition and the Nyström approximation via adaptive sampling. *The Journal of Machine Learning Research*, 14(1): 2729–2769, 2013.

Christopher Williams and Matthias Seeger. Using the Nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems 13*, 2001.

David P Woodruff. Sketching as a tool for numerical linear algebra. *Foundations and Trends® in Theoretical Computer Science*, 10(1–2):1–157, 2014.

W. W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33:452–473, 1977.

Kai Zhang and James T Kwok. Clustered Nyström method for large scale manifold learning and dimension reduction. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 21(10):1576–87, October 2010. ISSN 1941-0093. doi: 10.1109/TNN.2010.2064786.

Kai Zhang, Ivor W Tsang, and James T Kwok. Improved Nyström low-rank approximation and error analysis. In *Proceedings of the 25th international conference on Machine learning*, pages 1232–1239. ACM, 2008.

Hui Zou, Trevor Hastie, and Robert Tibshirani. Sparse principal component analysis. *Journal of computational and graphical statistics*, 15(2):265–286, 2006.