Abstract

A Fast Transform for Spherical Harmonics

Martin James Mohlenkamp

1997

Spherical Harmonics arise on the sphere S^2 in the same way that the (Fourier) exponential functions $\{e^{ik\theta}\}_{k\in\mathbf{Z}}$ arise on the circle. Spherical Harmonic series have many of the same wonderful properties as Fourier series, but have lacked one important thing: a numerically stable fast transform analogous to the Fast Fourier Transform. Without a fast transform, evaluating (or expanding in) Spherical Harmonic series on the computer is slow — for large computations prohibitively slow. This thesis provides a fast transform.

For a grid of N^2 points on the sphere, a direct calculation has computational complexity $\mathcal{O}(N^4)$, but a simple separation of variables and Fast Fourier Transform reduces it to $\mathcal{O}(N^3)$ time. Here we present an algorithm with time $\mathcal{O}(N^2 \log^2 N)$.

The problem quickly reduces to the fast application of matrices of Associated Legendre Functions of certain orders. The essential insight is that although these matrices are dense and oscillatory, locally they can be represented efficiently in trigonometric functions. We construct an adaptive partition of each matrix into rectangles such that each rectangle is sparse in (two dimensional) local trigonometric series. The entire matrix can be represented in $\mathcal{O}(N \log N)$ coefficients in this non–standard form, with constants independent of the order parameter. It can be applied in $\mathcal{O}(N \log N)$ time, but the overhead cost for adaption is $\mathcal{O}(N \log^2 N)$. Adding this over the orders yields the overall time $\mathcal{O}(N^2 \log^2 N)$.

A Fast Transform for Spherical Harmonics

A Dissertation

Presented to the Faculty of the Graduate School

of

Yale University

in Candidacy for the Degree of

Doctor of Philosophy

by

Martin James Mohlenkamp

Dissertation Director: Professor Ronald Raphael Coifman

May 1997

Updated June 10, 1997.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Spherical Harmonics arise on the sphere S^2 in the same way that the (Fourier) exponential functions $\{e^{ik\theta}\}_{k\in\mathbf{Z}}$ arise on the circle. Spherical Harmonic series have many of the same wonderful properties as Fourier series, but have lacked one important thing: a numerically stable fast transform analogous to the Fast Fourier Transform. Without a fast transform, evaluating (or expanding in) Spherical Harmonic series on the computer is slow — for large computations prohibitively slow. This thesis provides a fast transform.

Using spherical coordinates ($\phi \in (0, \pi)$, $\theta \in [0, 2\pi)$), the Spherical Harmonics for S^2 can be represented as $\{P_n^m(\cos\phi)e^{im\theta}\}$ where $n \geq |m|$, m and n are integers, and the P_n^m are Associated Legendre functions. When properly normalized, this forms a basis for the sphere with measure $\sin\phi d\phi d\theta$.

On a $N \times N$ equispaced grid in $\phi \times \theta$ we can resolve $N^2/4$ basis functions. If we expand (or evaluate) directly using these basis functions, it will take $N^4/4$ computations. In the θ variable, our basis is simply Fourier series and we can use the FFT. Applying the FFT for N values of ϕ requires $N \times N \log N$ computations, and leaves us with a set of N problems indexed by m. Each problem is: on an interval of N points, expand (evaluate) a given function using the basis $\{P_n^m(\cos\phi)\}$ with measure $\sin\phi d\phi$. If we do each of these directly, it will take $N \times \mathcal{O}(N^2)$ computations. Our overall cost is then $\mathcal{O}(N^3)$.

Here we present a method for expanding (evaluating) using $\{P_n^m(\cos\phi)\}$ in $\mathcal{O}(N\log^2 N)$ time, with constant independent of m. This method allows us to reduce the overall cost to $\mathcal{O}(N^2\log^2 N)$. We also present an $\mathcal{O}(N^{5/2}\log N)$ algorithm which, although asymptotically slower, performed better in our tests. Both methods use precomputed, compressed representations of the Associated

Legendre function bases.

1.1 Statement of the Problem

1.1.1 The Evaluation Problem

We are given a set of Spherical Harmonics:

$$\left\{\tilde{P}_n^m(\cos\phi)e^{im\theta}/\sqrt{2\pi}\right\}_{(m,n)\in\mathcal{I}_N} \tag{1.1}$$

defined on the sphere ($\phi \in (0,\pi)$, $\theta \in [0,2\pi)$). $\tilde{P}_n^m$ is the Associated Legendre function of degree n and order m, normalized so that our set of functions is orthonormal on the sphere. The indices allowed are restricted to the set

$$\mathcal{I}_N = \{(m,n) \in \mathbf{Z}\times\mathbf{Z} : 0 \le n < N/2, -n \le m \le n\}. \tag{1.2}$$

We are also given a set of coefficients

$$\{\alpha_n^m\}_{(m,n)\in\mathcal{I}_N} \tag{1.3}$$

and we form the sum

$$f(\phi,\theta) = \sum_{\mathcal{I}_N} \alpha_n^m \tilde{P}_n^m(\cos\phi)e^{im\theta}/\sqrt{2\pi}. \tag{1.4}$$

This sum is evaluated on a $N\times N$ equispaced grid in $\phi\times\theta$:

$$\left\{(\phi_j,\theta_k) = \left(\pi\frac{j+1/2}{N}, 2\pi\frac{k}{N}\right) : j,k \in \mathbf{Z}; 0 \le j,k < N\right\}. \tag{1.5}$$

The evaluation problem for Spherical Harmonics is to compute the $\mathcal{O}(N^2)$ output values $f(\phi_j,\theta_k)$ from the $\mathcal{O}(N^2)$ input values α_n^m in a fast, numerically stable way.

1.1.2 The Expansion Problem

The expansion problem for Spherical Harmonics is dual to the evaluation problem. We are given the sampled values $f(\phi_j,\theta_k)$ of a function of the form

$$f(\phi,\theta) = \sum_{\mathcal{I}_N} \alpha_n^m \tilde{P}_n^m(\cos\phi)e^{im\theta}/\sqrt{2\pi}. \tag{1.6}$$

From the $\mathcal{O}(N^2)$ input values $f(\phi_j,\theta_k)$ we wish to compute the $\mathcal{O}(N^2)$ output values α_n^m in a fast, numerically stable way.

1.2 Main Idea

The problem of the Fast Transform quickly reduces to a set of N problems indexed by m ($-N/2 < m < N/2$). (See Section 2.4.) The problems are to find a fast application for the matrices

$$M^m_{n,j} = \left(\sqrt{\sin\phi_j}\,\tilde{P}^m_n(\cos\phi_j)\right)_{n,j}. \tag{1.7}$$

These matrices are dense and oscillatory. (See Figure 2.2.)

To understand these matrices, we model them using quasi–classical (WKB) frequency estimates. (See Section 2.5.) (This model gives the proper intuition, but we use more rigorous techniques in the proof.) The model is

$$M^m_{n,j} \approx \left(\exp\left(\int^{\phi_j} i\sqrt{(n+1/2)^2 - \frac{m^2-1/4}{\sin^2 t}}\,dt\right)\right)_{n,j} = \left(\exp(in\phi_j\Phi^m(n,\phi_j))\right)_{n,j}. \tag{1.8}$$

We now fix m and analyze $\Phi^m(n,\phi)$. (See [BCG93] for a similar model analysis.) If for each n, $\Phi^m(n,\phi)$ is constant as a function of ϕ, this matrix application becomes the evaluation of a Fourier series and so can be done fast using the FFT. $\Phi^m(n,\phi)$ is not constant, but considered on the correct intervals it is close enough for our purposes. For each $\Phi^m(n,\phi)$, we can partition ϕ–space into a small number of intervals, on each of which $\Phi^m(n,\phi)$ is almost constant. Using this partition, we can write $\exp(in\phi\Phi^m(n,\phi))$ as a sum of $\mathcal{O}(\sqrt{N})$ localized exponentials. The partition, however, *depends on* n (and m). Restricting ourselves to dyadic partitions, we can find a suitable partition for each $\Phi^m(n,\phi)$, and yet the total collection of intervals used is manageable. We can now represent the entire matrix (1.8) in terms of $\mathcal{O}(N^{3/2})$ exponentials, apply it in this same $\mathcal{O}(N^{3/2})$ time, and return to normal coordinates with the FFT in $\mathcal{O}(N\log^2 N)$ time.

If for each ϕ, $\Phi^m(n,\phi)$ is constant as a function of n, this matrix application becomes an expansion into a Fourier series, which can also be done fast using the FFT. Thus we can use the same technique in the n coordinate. In fact, we can partition in both coordinates at the same time, which corresponds to breaking the matrix into dyadic rectangles. In ordinary coordinates, each rectangle gives the interaction of some interval in n with some interval in ϕ. We instead represent each rectangle by the interactions of a small number of exponentials in n with a small number of exponentials in ϕ. We can then represent the matrix (1.8) in terms of $\mathcal{O}(N\log N)$ interactions, apply it in $\mathcal{O}(N\log N)$ time, and return to normal coordinates with the FFT in $\mathcal{O}(N\log^2 N)$ time. The partition into rectangles will depend on m, and will not generally be a tensor product partition.

1.3 Summary of Results

Spherical Harmonics arise on the sphere in the same way the (Fourier) exponential functions arise on the circle. Both try to behave as waves. The topology of the circle allows the exponential functions to actually be waves. The topology of the sphere prevents this, however. Choosing a spherical coordinate system, we can embed circles as the lines of latitude, and have the Spherical Harmonics be waves along them. In the other coordinate they are not waves but instead are Associated Legendre functions.

The first result of this thesis is that each Associated Legendre function can be broken into pieces, each of which behaves sufficiently like a wave to compress well in a local trigonometric basis. The number of pieces needed to do this is small enough that we can compress each $\sqrt{\sin\phi}\tilde{P}_n^m(\cos\phi)$ into $\mathcal{O}(\sqrt{N}\log N)$ coefficients, with constants independent of the order (m) and degree (n) parameters. The partition chosen will depend on both m and n, so we adapt using a dyadic local trigonometric basis. The compressed coefficients are precomputed and stored. For each m this compressed form allows us to expand (evaluate) a given function using the basis $\{\tilde{P}_n^m(\cos\phi)\}$ in $\mathcal{O}(N^{3/2}\log N)$ time, and thus do the entire transform in $\mathcal{O}(N^{5/2}\log N)$ time. We call this the One Dimensional Algorithm.

The second result is the observation that, as a function of n, $\{\tilde{P}_n^m(\cos\phi)\}$ also resembles a wave. We can thus regard the matrix in $n \times \phi$ as an image and compress it in both directions. We can then compress the matrix for a fixed m into $\mathcal{O}(N\log N)$ coefficients, with constants independent of the order (m) parameter. Our compression corresponds to a partition of the matrix into dyadic rectangles. The size and shape of the rectangles varies for different portions of the matrix, so we again must adapt, and this adaption will depend on m. The setup and final evaluation steps cost $\mathcal{O}(N\log^2 N)$ time, so they dominate. The entire transform can thus be done in $\mathcal{O}(N^2\log^2 N)$ time. We call this the Two Dimensional Algorithm.

In order to prove our compression estimates, we construct a quantitative type of Quasi–Classical (WKB) theory for solutions to Schrödinger equations in one dimension. Given a space localization, this gives us rigorous bounds on the number of local Fourier coefficients needed for a given precision. This theory also provides an intuition using 'instantaneous frequency' and the ever–elusive justification for this intuition.

We have implemented these algorithms, mainly for diagnostic purposes. Experiments have shown the algorithms to be stable and to work well at high precision. The one dimensional algorithm is outperforming the compression predictions, while the two dimensional is not performing as well. In our implementation the one dimensional algorithm becomes faster than the direct method at $N = 256$. It's true adaptive nature becomes clear by $N = 2048$ when it is approximately 3.3 times faster. Since this implementation is meant for diagnostics, it is not fully optimized. We indicate some areas for improvement.

The techniques developed have general utility. For example, we obtain the Fast Hankel transform as a corollary. The non–standard matrix representation and numerical techniques should allow us to apply other oscillatory matrices quickly and to understand their structure. In particular, integral operators with oscillatory kernels seem a good object for future study.

1.4 Organization of the Thesis

The thesis is organized as follows:

Chapter 2 tells how the Spherical Harmonics arise and why they are interesting. This background gives us our first reductions of the problem, and allows preliminary analysis of the Associated Legendre functions.

Chapter 3 gives the structure of the algorithms used for the Spherical Harmonic transform. It shows they do in fact give the transform but does not discuss the compression rates, which determine the speed.

Chapter 4 gives proofs for the compression achieved in the precomputation stage, and thus shows the algorithms will be fast.

Chapter 5 gathers assorted information and results that, though not necessary for the theoretical result of a fast transform, are essential to its practical application.

Chapter 6 contains the results of numerical testing of these algorithms.

Chapter 7 contains an outline for a different fast transform, theoretically faster, but not fully developed.

1.5 Acknowledgements

I would like to thank:

My parents for being so wonderful. I have been truly blessed.

My advisor, Prof. R.R. Coifman, for all he has taught me. And also for putting up with my stubborn nature, and for bopping me on the head when I had gone astray.

Prof. Vladimir Rokhlin, for his insights into computers.

Prof. Gregory Beylkin, for many useful criticisms of the manuscript.

My cohorts in crime, particularly Jorge Aarao, Nick Bennett, Andreas Coppi, Hany Farag, Frank Geshwind, Roland Guglielmi, Tomasz Hrycak, Nets Katz, Lucas Monzón, Paul Müller, Cristina Pereyra, Christoph Thiele, Ana Vargas, Karl Voss, Lionel Woog, Leslie Ward, and Fred Warner.

Chapter 2

Background

This chapter provides historical notes and the background material, definitions, and basic tools which we need for the problem. None of the facts here are new, although sometimes the perspective is non–standard. We are able to reduce our original problem somewhat, and give a preliminary analysis of the Associated Legendre functions.

2.1 History

This section gives a brief sketch of the history of Spherical Harmonics and the fast transform problem. It is mainly intended to provide references.

Spherical Harmonics arise naturally when one tries to generalize Fourier Series to the next dimension. Because they are so natural, there is a rich literature on their properties. One extensive work is Hobson [Hob31]. It refers to works such as Ferres' *Spherical Harmonics* (1877) and claims the term 'Spherical Harmonic' was introduced by Thomson (Lord Kelvin) sometime before his 1879 work (with Tait) *Natural Philosophy* [Hob31, p119]. The first work on Legendre polynomials was done by Legendre and Laplace in the 1780's [Hob31, p.16]. A good source for related material is Szegö [Sze75]. The Spherical Harmonics can be built from Associated Legendre functions, which are considered 'special functions', and therefore appear in many handbooks of functions (see e.g. [AS64] and [Leb72]). As special functions they can be transformed into other special functions. See the above and also [EMOT54].

In the case of the fast Fourier transfrom, the theoretical result was known since Gauss in 1805 [FPTV92, p.504]. With the advent of computers this became more relevant, and it was reinvented and popularized by Cooley and Tukey [CT65]. This kindled interest in the fast Spherical Harmonic transform, for which there was no theoretical result. Since then there have been many attempts. There has not been a steady progress toward a solution, so the literature is fragmented, with little cross–referencing.

We first note the work of Dilts [Dil85]. This method uses the FFT to reduce the operations count versus simple integration. It does not improve the order of algorithm, however. This method is evaluated in Duvall, Elowitz, and Hill [DEH89]. They find it far too slow and claim the best current (1988) method is due to Brown [Bro85]. He separates variables and does an FFT in the ϕ variable to reduce the operations count to $\mathcal{O}(N^3)$ (See Section 2.4.1). This result is very likely much older, and indeed Brown does not claim originality.

Orzag [Orz86] proved a general result that one can expand or evaluate any set of Sturm–Liouville eigenfunctions (e.g. $\{P_n^m(\cos\phi)\}$ for fixed m) in $\mathcal{O}(N\log^2 N/\log\log N)$ time. His work is based on Quasi–Classical approximations (Section 2.5). He separates each basis function into two regions. On one, the approximation is exact to machine precision and the transform may be done fast using FFT's. On the other we do a direct computation. As N increases the first region dominates. As an asymptotic result this is fine, but it gives no control over the constants involved, or when the asymptotic behavior manifests. In particular, the constant may depend on m. As N increases, the allowed values of m also increase and thus on average we may need a lot more computations. In the present work, especially Section 4.1, we discover that when this approximation fails, the actual behavior is rather benign. By choosing appropriate intervals and compressing numerically, we can capture this behavior.

Alpert and Rokhlin [AR91] have an $\mathcal{O}(N\log N)$ algorithm for Legendre Polynomials (P_n^0). The key to their method is noticing that the matrix consisting of the cosine transforms of P_n^0 is smooth. They can compress this matrix analytically and apply quickly. As m increases, however, the matrix has m ridges, and their technique would break down. A frequency space version of our Section 4.4 would show that in fact one could have uniform bounds in m. We draw much from their ideas and approach.

Driscoll and Healy [DH94] have an $\mathcal{O}(N\log^2 N)$ algorithm for expansions (not evaluations) in $\{P_n^m(\cos\phi)\}$ with constant independent of m. They use the interplay between recurrence relations to transform Chebychev (cosine) coefficients to P_n^m coefficients. Unfortunately, the initial step involves multiplying the given function by $\sin^m\phi$ and then expanding in a cosine series. This operation is poorly conditioned for large m. This instability illustrates a pitfall of many approaches: being 'special functions', the Associated Legendre functions are related to other special functions by myriad formulae. These often suggest fast algorithms, but either slow down or become less accurate as m increases. Philosophically, this is because different special functions really are different: we can get from one to another, but only by passing through a magic (unstable) formula.

The above methods are all for a fixed, equispaced grid. There are results for more general samplings. M. Frumkin [Fru95] has an $\mathcal{O}(N^3 \log^2 N)$ algorithm for expansions on N^2 samples. These methods tend to lack definiteness, which means that issues like numerical stability are not discussed. For a non–equispaced FFT, see Dutt and Rokhlin, [DR93] and [DR95]. Their ideas should extend to Spherical Harmonic transforms.

2.2 What are Spherical Harmonics ... ?

This section gives a development for Fourier series and Spherical Harmonics which emphasizes their parallels. We also determine many properties of Spherical Harmonics which we will need later. Much of this discussion is based on Stein and Weiss [SW71, Chapter IV]. See also [CW68].

2.2.1 $\mathbf{R}^2$

We begin by looking at $\mathbf{R}^2$, $\{(x,y) : x, y \in \mathbf{R}\}$. A very natural operator to consider is the laplacian:

$$\Delta_2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}. \tag{2.1}$$

Considered in polar coordinates, $\{(r,\theta) : r \in \mathbf{R}_+, \theta \in [0, 2\pi)\}$, this becomes:

$$\Delta_2 = \frac{1}{r^2}\frac{\partial^2}{\partial \theta^2} + \frac{\partial^2}{\partial r^2} + \frac{1}{r}\frac{\partial}{\partial r}. \tag{2.2}$$

A given function is called *harmonic* if $\Delta_2 f(r,\theta) = 0$. Suppose we have a harmonic function which we also know is homogeneous of degree n. Then $f(r,\theta) = r^n f(\theta)$ and we have:

$$0 = \Delta_2 f(r,\theta) = n(n-1)r^{n-2} f(\theta) + \frac{1}{r} n r^{n-1} f(\theta) + \frac{1}{r^2} r^n f''(\theta) \tag{2.3}$$

which implies $f''(\theta) = -n^2 f(\theta)$ which in turn implies $f(\theta) = Ae^{in\theta} + Be^{-in\theta}$. We define the *Circular Laplacian* by $\Delta_{2S} = \frac{\partial^2}{\partial \theta^2}$. It is self–adjoint and rotation–invariant, which implies that its eigenspaces Λ_n are orthogonal and rotation–invariant. Λ_n is the space of homogeneous, harmonic polynomials of degree n, restricted to the circle. This space is of dimension two, and we can span it with $\{(x+iy)^n, (x-iy)^n\}$, which, restricted to the circle, is $\{e^{in\theta}, e^{-in\theta}\}$. Thus Fourier series can be extracted in a natural way from the Laplacian.

We recall that they are a (wonderful) basis and that we can represent any function in L^2 by $g(r,\theta) \sim \sum g_n(r)e^{in\theta}$. Recall also that the Fourier transform commutes with rotations: $(z, \zeta \in \mathbf{R}^2)$

$$(h(e^{i\nu}z))\hat{}(\zeta) = \int h(e^{i\nu}z)e^{-i\zeta\cdot z}dz = \int h(t)e^{-i(\xi e^{i\nu})\cdot z}dt = \hat{h}(e^{i\nu}\zeta) \tag{2.4}$$

We also have $(z = re^{i\theta},\ \zeta = se^{i\phi})$

$$e^{in\nu}(g_n(r)e^{in\theta})\hat{}(s,\phi) = (g_n(r)e^{in(\theta+\nu)})\hat{}(s,\phi) = (g_n(r)e^{in\theta})\hat{}(s,\phi+\nu) \tag{2.5}$$

which implies that

$$(g_n(r)e^{in\theta})\hat{}(s,\phi) = \tilde{g}_n(s)e^{in\phi}. \tag{2.6}$$

This means that the spaces $g(r)\Lambda_n$ are also preserved by the Fourier transform. $\tilde{g}_n$ is, up to normalization, the n^{th} order Hankel transform of g_n. Explicitly,

$$\tilde{g}_n(s) = 2\pi(-i)^k \int_0^\infty g_n(r)J_n(2\pi sr)rdr \tag{2.7}$$

with the Bessel functions defined as

$$J_n(x) = \frac{1}{2\pi}\int_0^{2\pi} e^{ix\sin\theta}e^{-in\theta}d\theta. \tag{2.8}$$

See [SW71, p.137].

2.2.2 $\mathbf{R}^3$

In $\mathbf{R}^3$, $\{(x,y,z) : x,y,z \in \mathbf{R}\}$, we have

$$\Delta_3 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}. \tag{2.9}$$

In spherical coordinates $\{(r,\theta,\phi) : r \in \mathbf{R}_+, \theta \in [0,2\pi), \phi \in [0,\pi]\}$ this means

$$\Delta_3 = \csc^2\phi\frac{\partial^2}{\partial\theta^2} + \frac{\partial^2}{\partial\phi^2} + \cot\phi\frac{\partial}{\partial\phi} + 2\rho\frac{\partial}{\partial r} + \rho^2\frac{\partial^2}{\partial r^2}. \tag{2.10}$$

Suppose, as above, we have a harmonic function with $f(r,\theta,\phi) = r^n f(\theta,\phi)$. We then have $0 = \Delta_3 f =$

$$\left[\csc^2\phi\frac{\partial^2}{\partial\theta^2} + \frac{\partial^2}{\partial\phi^2} + \cot\phi\frac{\partial}{\partial\phi}\right] r^n f(\theta,\phi) + 2rnr^{n-1}f(\theta,\phi) + r^2 n(n-1)r^{n-2}f(\theta,\phi). \tag{2.11}$$

Defining the *Spherical Laplacian* by

$$\Delta_{3S} = \csc^2\phi\frac{\partial^2}{\partial\theta^2} + \frac{\partial^2}{\partial\phi^2} + \cot\phi\frac{\partial}{\partial\phi} \tag{2.12}$$

we then have $\Delta_{3S}f(\theta,\phi) = -n(n-1)f(\theta,\phi)$. This f is therefore an eigenfunction of the Spherical Laplacian. Any such eigenfunction is called a *Spherical Harmonic*.

As before, we note that Δ_{3S} is self–adjoint and rotation–invariant, which implies that the eigenspaces Λ_n are preserved by rotations and are orthogonal. As before, $g(r)\Lambda_n$ is also preserved by the Fourier transform. Λ_n consists of the homogeneous harmonic polynomials of degree n restricted to the sphere, and has dimension $2n+1$. On the sphere, the homogeneous harmonic polynomials span the set of all polynomials, which in turn are dense in L^2. Our spherical harmonics therefore span L^2. If we take a basis within each eigenspace then this collection will give a basis for L^2 of the sphere. Each basis element is the restriction of a polynomial and is therefore smooth. Thus Spherical Harmonics arise in $\mathbf{R}^3$ in the same way Fourier series arise in $\mathbf{R}^2$. They are consequently sometimes called 'Fourier series on the Sphere'. (Other less deserving objects sometimes also go by this name in the literature.)

For fixed n, we can organize the Λ_n as

$$\{\tilde{P}_n^m(\cos\phi)e^{im\theta}/\sqrt{2\pi}\}_{-n\leq m\leq n}. \tag{2.13}$$

This organization is not forced, but is natural since we have circles embedded in the sphere as lines of latitude. We would like to find the conditions on $\tilde{P}_n^m$ to make this a set of (smooth) spherical harmonics. We need to have

$$\Delta_{3S}e^{im\theta}\tilde{P}_n^m(\cos\phi) = -n(n-1)e^{im\theta}\tilde{P}_n^m(\cos\phi) \tag{2.14}$$

which means

$$\left[\frac{\partial^2}{\partial\phi^2} + \cot\phi\frac{\partial}{\partial\phi} - \frac{m^2}{\sin^2\phi}\right]\tilde{P}_n^m(\cos\phi) = -n(n-1)\tilde{P}_n^m(\cos\phi). \tag{2.15}$$

This equation, and the condition that $\tilde{P}_n^m(1) \neq \pm\infty$, identifies the $\tilde{P}_n^m$'s up to a constant as the *Associated Legendre Functions (of the first kind) of order m and degree n.* We use the tilde ($\tilde{\cdot}$) to indicate the L^2 normalized version of the classically defined Associated Legendre functions, denoted P_n^m. $\tilde{P}_n^m$ can be constructed explicitly, and we do this in Section 5.1.

2.3 . . . and Why Do We Care?

In this section we consider some of the applications of Spherical Harmonics. We do this mainly by showing some of the applications of Fourier Series and seeing how they extend. These results are well known.

Fourier Series are a basis for the circle. They have the nice property that the eigenspaces Λ_n are rotation–invarient. This property implies that if we know the expansion of a function, we can compute the expansion of any of its rotations by applying a sparse matrix. This matrix is zero except for 2×2 blocks along the diagonal, one block for each n. If we have a band-width limitation $n < N$, this means we can apply this matrix in $2N$ time, instead of $4N^2$ time.

Spherical Harmonics work similarly. The eigenspaces are also rotation–invarient, though now they are not of the same size. The matrix of a rotation in this basis consists of blocks along the diagonal, with a $(2n+1) \times (2n+1)$ block for Λ_n. If we again have a limit $n < N$, we can apply the matrix in $\sum(2n+1)^2 = (4N^3 - N)/3$ time, instead of $[\sum(2n+1)]^2 = N^4$ time.

Next we consider operators which are rotation–invarient. Many operators which correspond to physical processes (gravity, heat, electricity) are coordinate–invarient. In particular, if the process is taking place on a circle or sphere, these operators commute with rotations. On the circle, rotating $e^{in\theta}$ through an angle ν is the same as multiplying by $e^{in\nu}$. For such an operator L, this means

$$(Le^{in\theta})(\xi) = (Le^{in(\theta-\nu)})(\xi + \nu) \tag{2.16}$$

$$= e^{-in\nu}(Le^{in\theta})(\xi + \nu) \tag{2.17}$$

which implies $(Le^{in\theta})(\xi + \nu) = \lambda e^{in(\xi+\nu)}$. Thus the Fourier Series are eigenfunctions for *all* such operators. Since they are also a basis, this means they diagonalize such operators, making them trivial to apply. For the Spherical Harmonics, we have the same result, and the proof is essentially the same.

One particular example of an operator which commutes with rotations is the operator of convolution. The well used property that Fourier series convert convolution to multiplication is equivalent to noting the Fourier Series are eigenfunctions, as we did above. For Spherical Harmonics we have the similar result

$$(f * g)_n^m = f_n^m g_n^0. \tag{2.18}$$

See [DH94, p.210] or [CW68] for its discussion.

2.4 First Reductions of the Problem

We now have enough background information for a major simplification of our original problem as stated in Section 1.1. We also use this opportunity to gather in one place other small simplifications we will use later.

2.4.1 The Reduced Evaluation Problem

The evaluation problem (Section 1.1.1) can be rewritten as

$$f(\phi_j, \theta_k) = \sum_{m=1-N/2}^{N/2-1} \left(\sum_{n=|m|}^{N/2-1} \alpha_n^m \tilde{P}_n^m(\cos\phi_j) \right) e^{im\theta_k}/\sqrt{2\pi}. \tag{2.19}$$

Assuming the values of $\tilde{P}_n^m(\cos\phi_j)$ have been precomputed, this takes $\mathcal{O}(N^4)$ operations to evaluate directly. The outer sum, however, is the evaluation of a Fourier series. For each ϕ_j we can evaluate this in $\mathcal{O}(N \log N)$ time using the Fast Fourier Transform (FFT) to obtain:

$$f(\phi_j, \theta_k) = \left(\sum_{n=|m|}^{N/2-1} \alpha_n^m \tilde{P}_n^m(\cos\phi_j) \right) (\phi_j, \check{\theta}_k). \tag{2.20}$$

Thus, for total computational cost of $\mathcal{O}(N^2 \log N)$, we have reduced the original problem to a set of N problems indexed by m $(-N/2 < m < N/2)$. Each problem is: Given the $\mathcal{O}(N)$ inputs α_n^m, compute the N outputs

$$\sum_{n=|m|}^{N/2-1} \alpha_n^m \tilde{P}_n^m(\cos\phi_j). \tag{2.21}$$

Evaluated directly, each problem takes $\mathcal{O}(N^2)$ time, and so together they take $\mathcal{O}(N^3)$. This dominates the FFT time, so the time for the entire evaluation problem has been reduced to $\mathcal{O}(N^3)$. A reduction of this form appears in Brown [Bro85] but was probably known much earlier.

2.4.2 The Reduced Expansion Problem

For the expansion problem (Section 1.1.2) we have learned first that these Spherical Harmonics are an orthonormal set, and so we should compute α_n^m by

$$\begin{aligned} \alpha_n^m &= \int_0^\pi \int_0^{2\pi} f(\phi, \theta) \tilde{P}_n^m(\cos\phi) \frac{e^{-im\theta}}{\sqrt{2\pi}} \sin\phi \, d\phi \, d\theta && (2.22) \\ &= \int_0^\pi \left(\int_0^{2\pi} f(\phi, \theta) \frac{e^{-im\theta}}{\sqrt{2\pi}} d\theta \right) \tilde{P}_n^m(\cos\phi) \sin\phi \, d\phi. && (2.23) \end{aligned}$$

The inner integral is a Fourier series expansion. With our assumption (1.6) on the true nature of f, the sampling theorem says

$$\int_0^{2\pi} f(\phi, \theta) \frac{e^{-im\theta}}{\sqrt{2\pi}} d\theta = \sum_{k=0}^{N-1} f(\phi, \theta_k) e^{-im\theta_k} \frac{1}{\sqrt{N}}. \tag{2.24}$$

The sum can be computed for each ϕ in $\mathcal{O}(N \log N)$ time with the FFT, to yield $f(\phi, \hat{m})$. We have now reduced the expansion problem to a set of N problem indexed by m, each of which is: Given the N sample values $f(\phi_j, \hat{m})$ compute the $\mathcal{O}(N)$ coefficients α_n^m. The computational cost of this reduction is $\mathcal{O}(N^2 \log N)$ operations.

Our next task is to discretize the remaining integral. There is a system of quadrature weights $w_N(j)$, $0 \leq j < N$ (see Section 5.2) such that

$$\alpha_n^m = \int_0^\pi f(\phi, \hat{m}) \tilde{P}_n^m(\cos\phi) \sin\phi d\phi = \sum_{j=0}^{N-1} f(\phi_j, \hat{m}) \tilde{P}_n^m(\cos\phi_j) \sin\phi_j w_N(j). \tag{2.25}$$

Each reduced problem can be solved by evaluating this sum, which takes $\mathcal{O}(N^2)$ time. We can thus do the entire expansion problem in $\mathcal{O}(N^3)$ time.

2.4.3 The Two Problems Combined

We have reduced both problems to a set of matrix multiplications. For the evaluation problem we apply

$$\left(\tilde{P}_n^m(\cos\phi_j)\right)_{n,j} \alpha_n^m = f(\phi_j, \hat{m}) \tag{2.26}$$

and for the expansion problem

$$\left(\tilde{P}_n^m(\cos\phi_j)\right)_{j,n} (\sin\phi_j w_N(j) f(\phi_j, \hat{m})) = \alpha_n^m. \tag{2.27}$$

To convert from one problem to the other, the matrix is simply transposed. In our algorithm, we find a way to apply these matrices quickly. The method works equally well for the matrix transpose and so the evaluation and expansion problems become one problem. We will usually ignore the expansion problem and only consider the evaluation problem.

We will also find it more convenient to transform our problems slightly. We define

$$\psi_n^m(\phi) = \sqrt{\sin\phi}\, \tilde{P}_n^m(\cos\phi). \tag{2.28}$$

The reduced evaluation and expansion problems become:

$$\left(\psi_n^m(\phi_j)\right)_{n,j} \alpha_n^m = \sqrt{\sin\phi_j}\, f(\phi_j, \hat{m}) \tag{2.29}$$

and

$$\left(\psi_n^m(\phi_j)\right)_{j,n} (\sqrt{\sin\phi_j}\, f(\phi_j, \hat{m}) w_N(j)) = \alpha_n^m. \tag{2.30}$$

We make this transformation because the ψ_n^m satisfy more convenient differential equations, and because the are orthonormal (m fixed) with the uniform measure $d\phi$ instead of the weighted measure $\sin\phi d\phi$.

2.4.4 The Use of Parities

One property of $\tilde{P}_n^m(\cos\phi)$ that is only apparent from its explicit construction in Section 5.1 is that it is either even or odd across $\phi = \pi/2$ as $n-m$ is even or odd. We can solve the reduced problem separately for the even and odd components on $\phi \in (0, \pi/2)$ and then use simple reflections to produce the final solution. Explicitly, for the reduced evaluation problem in matrix form we let

$$\sqrt{\sin\phi_j} f_e(\phi_j, \hat{m}) = \left(\psi_n^m(\phi_j)\right)_{n,j} \alpha_n^m \tag{2.31}$$

for $n-m$ even and $j < N/2$. Similarly for $n-m$ odd

$$\sqrt{\sin\phi_j} f_o(\phi_j, \hat{m}) = \left(\psi_n^m(\phi_j)\right)_{n,j} \alpha_n^m. \tag{2.32}$$

Then we get f itself by

$$f(\phi_j, \hat{m}) = \begin{cases} f_e(\phi_j, \hat{m}) + f_o(\phi_j, \hat{m}) & 0 \leq j < N/2 \\ f_e(\phi_{N-j-1}, \hat{m}) - f_o(\phi_{N-j-1}, \hat{m}) & N/2 \leq j < N \end{cases}. \tag{2.33}$$

We use this explicitly in Section 4.3 when we prove compression results on $(0, \pi/2)$ and in Section 5.4.1 where we use it to justify a specialized search and to deal with edge effects. The use of parities also reduces computation time by a factor of two.

2.5 Quasi–Classical (WKB) Frequency Estimates

There is a method for estimating the solutions to some types of differential equations, in particular Schrödinger equation. See Landau and Lifschitz [LL77, Chapter VII], Bender ond Orzag [BO78], and Slavyanov[Sla96]. We will use this to get a preliminary description of the Associated Legendre functions and to motivate the algorithms in Chapter 3.

Supposing we have a function $\psi(x)$ which satisfies

$$\left[\frac{d^2}{dx^2} - U(x)\right]\psi(x) = -\lambda^2\psi(x), \tag{2.34}$$

this approximates ψ by $\exp(\int^x \pm\sqrt{U(t)-\lambda^2}dt)$. If $U(x)-\lambda^2 > 0$ then the root is real, and we will have exponential growth or decay, depending on the choice of $\pm$. If $U(x)-\lambda^2 < 0$ then the root is imaginary, and we will have oscillations of instantaneous frequency $\sqrt{\lambda^2 - U(x)}$. Points where $U(x) - \lambda^2 = 0$ are called *turning points.*

With the reductions made in Section 2.4.3, we are considering $\{\psi_n^m(\phi) = \sqrt{\sin\phi}\tilde{P}_n^m(\cos\phi)\}$ as a basis with measure $d\phi$. From (2.15) we can deduce the differential equation:

$$\left[\frac{d^2}{d\phi^2} - \frac{m^2 - 1/4}{\sin^2\phi}\right]\psi_n^m(\phi) = -(n+1/2)^2\psi_n^m(\phi) \tag{2.35}$$

which is a nice Schrödinger equation, to which we can apply the Quasi–Classical approximation. This approximation yields 'instantaneous frequency'

$$\nu_n^m(\phi) = \sqrt{(n+1/2)^2 - \frac{m^2 - 1/4}{\sin^2\phi}} \tag{2.36}$$

valid when the argument of the root is positive, and the approximation

$$\psi_n^m(\phi) \approx \exp\left(i\int^{\phi}\nu_n^m(t)dt\right). \tag{2.37}$$

These estimates tell us a bit about the behavior of our basis functions. At the edges of the interval (past the turning point) they decay rapidly and smoothly. From the turning point to the center of the interval, they have instantaneous frequency increasing and concave down. The rate of change of frequency is given by $\nu'(\phi)$. For a plot of an example Associated Legendre function, see Figure 2.1.

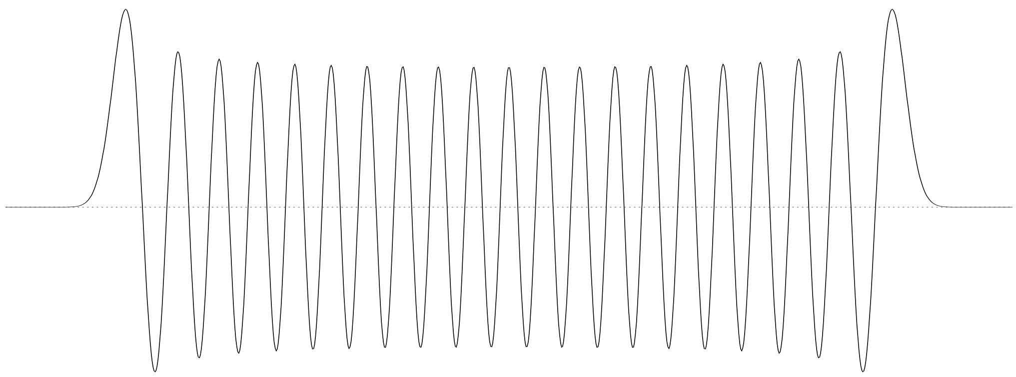

Figure 2.1: $\sqrt{\sin\phi}\tilde{P}_{60}^{20}(\cos\phi)$ on $[0,\pi]$.

Globally this function is complicated, but locally it looks very much like a trigonometric function. This is the motivation behind our one dimesional algorithm (Section 3.2). We partition our function and represent each piece in localized trigonometric functions. These representaions will be very efficient, and convert our matrix to a sparse or 'compressed' form. The partition we choose will depend on m and n. For fixed m, we wish to express the matrix $(\psi_n^m(\phi_j))_{n,j}$ in sparse (compressed) form. Since the partition changes with n, this is not simply a change of basis, but is an adaptive, non–standard representation of the matrix.

The estimate (2.37) can also be used to model the entire matrix $(\psi_n^m(\phi_j))_{n,j}$ (m fixed). See Figure 2.2 for a sample matrix. Note that the matrix is also almost trigonometric as a function of n. With a proper two dimensional partition, we should be able to compress this matrix even further. This is the motivation for the Two Dimensional algorithm in Section 3.3. The partition chosen will depend on m.

The Quasi–Classical method is important in that it gives an understanding of the behavior of the Associated Legendre functions and motivation for our algorithms. It lacks sufficient rigor for use in proofs, however. In Section 4.1 we develop an alternative method which gives rigorous results of the type we need.

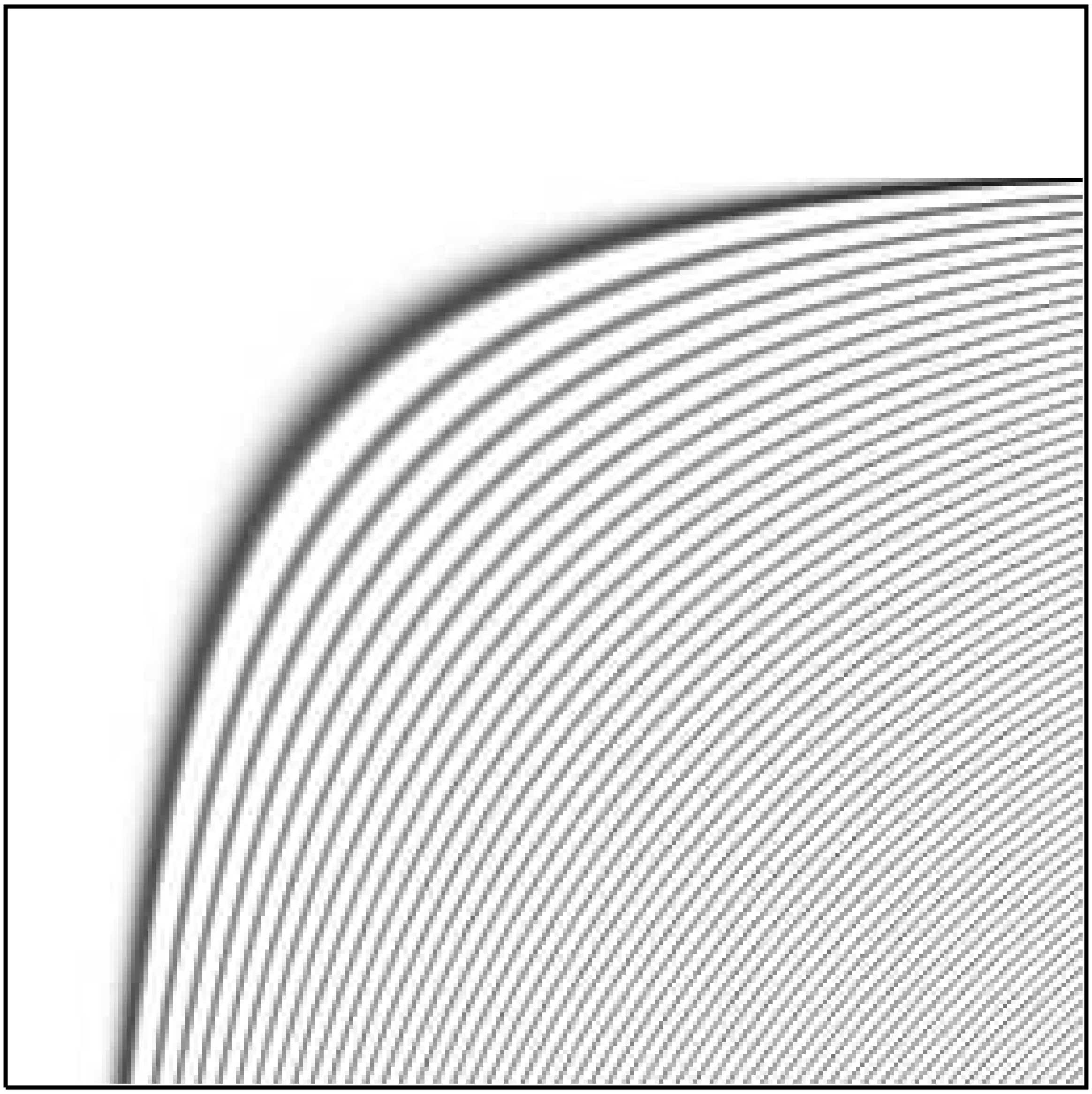

Figure 2.2: The positive part of $\sqrt{\sin\phi}\tilde{P}_n^{40}(\cos\phi)$ for n in $[0, 256]$ and ϕ in $[0, \pi/2]$. The n axis points down, and the ϕ axis to the right.

Chapter 3

The Structure of the Algorithms

This chapter gives the structure of the algorithms used for the Spherical Harmonic transform. The fact that they do in fact give the transform follows from simple facts such as orthogonality and is explained here. The speed of these algorithms depends mostly on the compression which is achieved in the precomputation stage, and this is dealt with in Chapter 4. We deal specifically with the reduced form of the evaluation problem, from Section 2.4.3.

We present two algorithms. The first, the one dimensional algorithm, is asymptotically slower, but has performed better in numerical testing. The core of both algorithms is a non–standard matrix representation and multiplication. By allowing this non–standard representation, we are able to adapt to different behaviors in different portions of the matrix. We note the similarities between our approach and the non–standard matrix multiplication in [BCR91].

3.1 Local Trigonometric Expansions

Our primary tool for compressing the Associated Legendre functions will be local trigonometric expansions. In Section 3.2 we do this in an orthogonal way, using the Local Cosine basis. In Section 3.3 we use a (seemingly) new technique, which is not orthogonal but which allows reconstruction and matrix applications. Both of these techniques can be seen as variations of the standard Local Cosine basis, which we now review. See [CM91] for a proof of its properties and [AWW92] or [Wic94] for an exposition of its uses.

A standard Local Cosine basis is constructed as follows: We begin with a sequence of points on the line (interval, circle) $\cdots a_i < a_{i+1} \cdots$. Let $I_i = [a_i, a_{i+1}]$. We have a set of compatible bells, indexed by their interval, $\{b_i(x)\}$. These bells have the properties that $b_i(x)b_{i-1}(x)$ is even about

a_i, $b_i(x)b_{i'}(x) = 0$ if $i' \neq i \pm 1$, and $\sum_i b_i^2(x) = 1$. We also take them to be real and non–negative. On each interval we have a set of cosines of the proper scaling and shift, denoted

$$\left\{ c_i^p(x) = \sqrt{\frac{2}{a_{i+1} - a_i}} \cos\left(\frac{(p+1/2)\pi(x - a_i)}{a_{i+1} - a_i} \right) \right\}_{p=0}^{\infty}. \tag{3.1}$$

The set $\{b_i(x)c_i^p(x)\}$ forms an orthonormal basis for the line. If $|i - i'| \geq 2$ the two elements are orthogonal because of disjoint support. If $i = i'$, $p \neq p'$ the properties of the bell allow us to reduce this to the orthogonality of cosines of different frequencies i.e.

$$\langle b_i c_i^p, b_i c_i^{p'} \rangle = \int b_i^2 c_i^p c_i^{p'} \, dx = \int_{a_i}^{a_{i+1}} c_i^p c_i^{p'} \, dx = 0. \tag{3.2}$$

If $i' = i + 1$,

$$\begin{aligned} \langle b_i(x)c_i^p(x), b_{i+1}(x)c_{i+1}^{p'}(x) \rangle &= \int [b_i(x)b_{i+1}(x)]c_i^p(x)c_{i+1}^{p'}(x)dx \\ &= \int [\text{even}] \cdot (\text{even}) \cdot (\text{odd}) = \int (\text{odd about } a_{i+1}) = 0. \end{aligned} \tag{3.3}$$

Completeness follows from the completeness of Fourier series.

There is a fast Local Cosine transform based on the FFT, using techniques similar to those in [FPTV92, Chapter 12]. We thus have a way to (quickly) expand a given function on subintervals. If on these subintervals our function is almost trigonometric, we can expect to capture it efficiently, and achieve compression.

3.2 The One Dimensional Algorithm

This section gives the algorithm based on compression in one direction. For motivation, see Section 2.5. The results on the compression itself are in Section 4.3.

In the precomputation stage, we express each ψ_n^m in the Local Cosine basis which requires the least number of coefficients above some given ϵ. We require the intervals to have dyadic size and dyadic position (intervals of length 2^{-j} are $k2^{-j}$ from the edge, $k \in \mathbf{Z}$). There are issues of bell compatibility and of finding the best partition in reasonable time, and these are dealt with in Section 5.4.1. We denote the set of dyadic intervals by $\mathcal{D}$ and the Local Cosine basis element for interval I with frequency p by $b_I(\phi)c_I^p(\phi)$. We can then write

$$\psi_n^m(\phi_j) = \sum_{I \in \mathcal{D}} \sum_{p=0}^{N|I|-1} \lambda_{Ip}^{mn} b_I(\phi_j) c_I^p(\phi_j). \tag{3.4}$$

Each ψ_n^m has λ_{Ip}^{mn}'s only for I's in some particular partition. We approximate the full sum with the sum where we discard all λ_{Ip}^{mn} which are smaller than ϵ.

The partition and coefficient choice can be represented graphically, as in Figure 3.1. We choose some collection of rectangles such that their projections on the horizontal are disjoint, but together cover the whole interval. This collection defines a dyadic partition of the interval. Each subinterval will contain some number of sample points. After applying the Local Cosine transform we have this same number of coefficients, which we store in the rectangle corresponding to that subinterval. Within this graphical representation with N columns and $\log_2 N$ rows, we can represent all dyadic partitions.

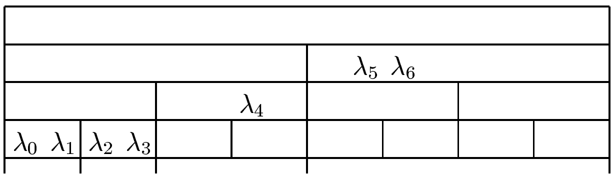

Figure 3.1: A partition and coefficients chosen for one ψ_n^m

For the reduced evaluation problem from Section 2.4.3, m is fixed, we are given $\{\alpha_n^m\}$, and we wish to form the sum

$$\sum_{n=|m|}^{N/2-1} \alpha_n^m \psi_n^m(\phi_j). \tag{3.5}$$

We can reorganize this as

$$\sum_n \alpha_n^m \sum_{I\in\mathcal{D}} \sum_p \lambda_{Ip}^{mn} b_I(\phi_j) c_I^p(\phi_j) = \sum_{I\in\mathcal{D}} b_I(\phi_j) \sum_p c_I^p(\phi_j) \sum_n \alpha_n^m \lambda_{Ip}^{mn}. \tag{3.6}$$

We can sum over n to obtain

$$= \sum_{I\in\mathcal{D}} b_I(\phi_j) \sum_p \gamma_{Ip}^m c_I^p(\phi_j). \tag{3.7}$$

To compute all the γ^m's costs as many computations as we have λ's. (Remember we have discarded $\lambda_{Ip}^{mn} < \epsilon$.) See Figure 3.2 for a schematic of this addition.

To evaluate the sums over I and p at all ϕ_j's, we do Local Cosine transforms. When $|I| = 2^{-q}$ there are 2^q intervals, each with $N2^{-q}$ points, so our cost is $2^q \cdot N2^{-q} \log(N2^{-q}) \leq N \log N$. Since an interval must contain a point, $q \leq \log N$ and so we make at most $N \log^2 N$ computations.

In Section 4.3 we will show that for each ψ_n^m we need at most $\mathcal{O}(\sqrt{m}(\log\epsilon)^{7/2}\log N)$ coefficients. Adding this over n yields a total count of λ's of $\mathcal{O}(\sqrt{m}(\log\epsilon)^{7/2} N \log N)$. The application of the

<table>
<tr><td colspan="8"></td></tr>
<tr><td colspan="4">λ_0'' λ_1'' λ_2'' λ_3''</td><td colspan="4">λ_5 λ_6 $\lambda_4''+\lambda_5'$ λ_6' λ_7 λ_8 λ_5'' λ_6''</td></tr>
<tr><td colspan="2">λ_0' λ_1'</td><td colspan="2">$\lambda_4+\lambda_2'$ λ_3'</td><td colspan="2"></td><td colspan="2"></td></tr>
<tr><td>λ_0 λ_1</td><td>λ_2 λ_3</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>

Figure 3.2: Summing the Local Cosine representations for several ψ_n^m. The α_i's have been suppressed.

compressed matrix (computation of γ) thus dominates the Local Cosine transforms. Doing this evaluation for N values of m gives us a total operations count of $\mathcal{O}(N^{5/2}\log N(\log\epsilon)^{7/2})$.

Remark: Although it is well disguised, this algorithm is really a matrix application. To apply the transpose (which is the inverse) matrix for the expansion problem, we invert all the steps. First expand the input function into all dyadic intervals. Then, for each n, take the inner product of this with the (sparse) representation of ψ_n^m on its given partition. This yields α_n^m.

3.3 The Two Dimensional Algorithm

This section gives the algorithm based on compression in two directions. For motivation, see Section 2.5. The results on the compression itself are in Section 4.4.

We again deal with the reduced evaluation problem (Section 2.4.3) for some fixed m. We consider it in its matrix form

$$(\psi_n^m(\phi_j))_{n,j}\,\alpha_n^m = \sqrt{\sin\phi_j}\,f(\phi_j,\hat{m}). \tag{3.8}$$

We consider the entire matrix at once, as a two dimensional image. Define the matrix $M_{n,j} = \psi_n^m(\phi_j)$ with $\psi_n^m \equiv 0$ for $n < |m|$. Let α be the vector of α_n^m's, and f be the vector of $\sqrt{\sin\phi_j}f(\phi_j)$'s. We can apply M to α directly with computation time equal to the number of nonzero entries of M, which is $N(N-m) = \mathcal{O}(N^2)$.

Suppose we break M into disjoint rectangular submatrices. We can apply M to α by applying each submatrix to some part of α and putting the result in the appropriate place in f. Conceptually, it is best to think of this as applying the transpose of the vector α (α^t) to M from above, and getting out f^t, as in Figure 3.3.

In the end, several rectangles contribute to each f_j, additively. Since the rectangles only interact additively we can treat each as a separate problem, as long as we can (quickly and easily) construct their inputs from α and recombine their outputs to get f. In particular, we can do a distinct change

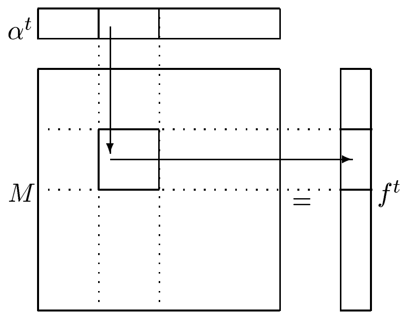

Figure 3.3: Applying a vector to a matrix from above.

of basis *on each rectangle*. As input for these rectangles, we must expand α locally into many different bases. Some rectangles may share input bases. We need to have a small enough set of input bases, each with a fast enough transform, so that preparing all the inputs is fast. We have a similar problem recombining the outputs to get f.

We assure the fast computation of inputs from α by first requiring the rectangles be dyadic (in size and position). This restriction greatly reduces the number of rectangles allowed, and hence the intervals used on α and f, while still allowing fairly general partitions. Second we require that the change of basis used in each direction be a Fourier series, scaled to that side of the rectangle. This restriction greatly reduces the number of change of bases, but is appropriate for our problem. (We will also allow the algorithm to choose not to change bases on some intervals/rectangles. For now this is merely a distraction, which we shall ignore.)

With these restrictions, preparing α and assembling f take at most $N \log^2 N$ computations. There are $\log N$ scales, and each scale takes less than $N \log N$ computations. See Figure 3.4.

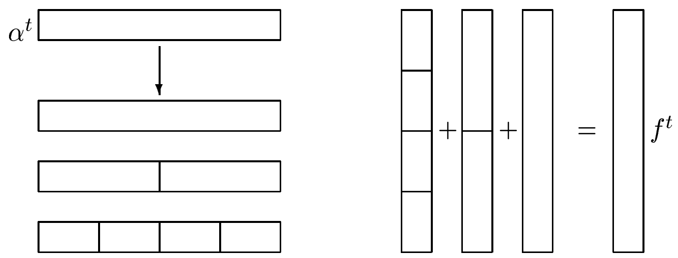

Figure 3.4: Computing inputs and ouputs for a partitioned matrix application.

The objective of this approach is to find a partition so that each rectangle can be represented by a small number of (Fourier) coefficients. The cost to apply this matrix is the total number of these which are greater than ϵ. We would like their total number to be $\mathcal{O}(N\log^2 N)$ or less. Then the overall cost for our evaluation is dominated by the expansions of α and evaluation of f, which takes $\mathcal{O}(N\log^2 N)$ time.

3.3.1 A Smoother Approach: Double Bells

In the above scheme we removed rectangles from M and expanded in Fourier series. Simply removing them creates a discontinuity at the edges, however. This discontinutity makes the Fourier coefficients decay very slowly and so will yield a large number of coefficients larger than ϵ. We would instead like to separate the rectangles smoothly.

We consider first the partitioning in n, which has to do with expansions of α. Fix j and a dyadic interval I. Let α_I be α restricted to I, I^* be the symmetric double of I, and b_I a smooth function supported on I^* with $1/2 \leq b(n) \leq 1$ when $n \in I$. As part of the matrix application, we wish to compute the inner product:

$$\langle M_{n,j}|_I, \alpha\rangle = \langle M_{n,j}, \alpha_I\rangle = \langle M_{n,j} b_I, \alpha_I/b_I\rangle. \tag{3.9}$$

Let $(\hat{\cdot})$ represent a Fourier expansion on I^*. Then

$$\langle M_{n,j} b_I, \alpha_I/b_I\rangle = \langle \widehat{M_{n,j} b_I}, \widehat{\alpha_I/b_I}\rangle. \tag{3.10}$$

Since $M_{n,j}b_I$ is as smooth as M, this eliminates the discontinuity. We will store the coefficients $\widehat{M_{n,j} b_I}$ instead of $\widehat{M_{n,j}|_I}$. These are then applied to the inputs $\widehat{\alpha_I/b_I}$. We refer to this trick as using 'double bells' or 'double sized intervals'. (This method was inspired by Cordoba [Cor77].)

The use of double bells causes a factor of two inefficiency in that there are twice as many coefficients $\widehat{M_{n,j} b_I}$ as entries in $M_{n,j}|_I$. We expect most of these coefficients to be negligible, however.

The evaluation problem for f is similar: Evaluate the series on J^*, then throw away the part not on J. On J, divide by b_J.

Note that with this approach we lose orthogonality. All we really need is the ability to apply the matrix, which we retain. An advantage of this method is that adjacent rectangles do not interact. This removes issues of bell compatibility and greatly simplifies searches (Section 5.5.1). This technique could also be used to simplify the one dimensional search (Section 5.4.1) but is costly numerically (Section 6.1.2).

In Section 4.4 we will show that for each m we need at most $\mathcal{O}(N\log N(\log\epsilon)^7)$ coefficients for M, with constants independent of m. The operations count of $\mathcal{O}(N\log^2 N)$ for the fast Fourier transforms thus dominates. Doing this evaluation for N values of m gives us a total operations count of $\mathcal{O}(N^2 \log^2 N)$.

Chapter 4

Compression Rates

This chapter gives rigorous proofs of bounds on the number of coefficients retained after compressing using the one or two dimensional compression schemes described in Chapter 3. We first construct a general theory of compression for solutions to Schrödinger equations, and then apply this theory to our specific case.

By 'compression' we mean the representation (to some prescribed accuracy) of a given function or matrix in a small number of parameters. We also require that we be able to effectively use these parameters in place of the function. This requirement means that a sparse representation of a function in terms of a basis with fast transform is compression, while the representation of the matrix $(\psi_n^m(\phi_j))_{n,j}$ by the two parameters m and N is not.

4.1 A Quantitative Quasi–Classical Theory

Consider a solution to some Schrödinger equation

$$\psi''(x) = -V(x)\psi(x) \tag{4.1}$$

The Quasi–Classical (WKB, semi–classical) theory (as in Section 2.5) provides the 'approximate' solutions

$$\exp\left(\pm i \int^x \sqrt{V(t)}dt\right) \tag{4.2}$$

and thus the 'instantaneous frequency' $\sqrt{V(x)}$. There is a procedure for constructing more terms to get a 'better' approximation. There is, however, no definite sense in which this approximation is 'better'. There is no norm associated to this problem. The strongest things said are like "Thus we have obtained a quantitative *quasi–classicality condition*: the wavelength of the particle must vary only slightly over distances of the order of itself." [LL77, p.165]. See also [BO78] and [Was65].

Here we use an approach which allows quantitative estimates. We use local Fourier series as our measuring tool and obtain bounds on the main support of this series and the decay away from this main support. In asymptotic cases, these results produce the usual frequency prediction. Their utility is that they allow us to deal concretely with any fixed case and to obtain quantitative bounds.

4.1.1 First Bounds

Theorem 4.1.1 *Let $\psi(x)$ be a solution to $\psi''(x) = -V(x)\psi(x)$ and $b(x)$ a smooth function supported on $[0,1]$. Then:*

$$|\widehat{b\psi}(\xi)| \leq \frac{||(V(x) - V^*)b||_p + ||b''||_p + 2|\xi| \cdot ||b'||_p}{|V^* - \xi^2|} ||\psi||_q \tag{4.3}$$

for any $V^* \in \mathbf{R}$ and $1 = (1/p) + (1/q)$; $1 \leq p, q \leq \infty$.

Proof: Integrating twice by parts, we obtain

$$\widehat{b\psi}(\xi) = \int_0^1 [b''(x)\psi(x) + 2b'(x)\psi'(x) + b(x)\psi''(x)] \frac{e^{-ix\xi}}{(-i\xi)^2} dx. \tag{4.4}$$

Considering the $b'\psi'$ term separately and integrating by parts again, we have:

$$\widehat{b\psi}(\xi) = \frac{1}{-\xi^2} \int_0^1 [-b''(x) - V(x)b(x) + 2i\xi b'(x)] \psi(x) e^{-ix\xi} dx. \tag{4.5}$$

Choosing any V^* we have

$$\widehat{b\psi}(\xi)\left(1 + \frac{V^*}{-\xi^2}\right) = \frac{1}{-\xi^2} \int_0^1 [-b''(x) - (V(x) - V^*)b(x) + 2i\xi b'(x)] \psi(x) e^{-ix\xi} dx \tag{4.6}$$

$$\widehat{b\psi}(\xi) = \frac{1}{V^* - \xi^2} \int_0^1 [-b''(x) - (V(x) - V^*)b(x) + 2i\xi b'(x)] \psi(x) e^{-ix\xi} dx \tag{4.7}$$

and the theorem follows by Hölder's inequality. □

Remarks:

1. Although the result holds for all V^*, we generally obtain the best localization by choosing it to minimize $||(V(x) - V^*)b||_p$.

2. We can reapply this theorem to each term on the right in (4.7), to obtain higher order decay. We will do this in Section 4.3.2 for a specific example.

4.1.2 Measuring Localization

We have 'localized' $\widehat{b\psi}$ near $\xi = \pm\sqrt{V^*}$. The localization is better when we choose $V^* \approx V(x)$. In a rough sense then, we can claim ψ has local frequency $\pm\sqrt{V^*}$ and instantaneous frequency $\pm\sqrt{V(x)}$. To quantify these statements, we would like to know how many frequencies (near $\pm\sqrt{V^*}$) are significant. $\widehat{b\psi}$ cannot be compactly supported, so we must choose some $\epsilon > 0$ and consider $\{\xi : |\widehat{b\psi}(\xi)| > \epsilon\}$ as being significant. We will call this set the ϵ*–support* of $\widehat{b\psi}(\xi)$.

Theorem 4.1.2 *With ψ and b as in Theorem 4.1.1, and with the assumptions $||\psi||_q = 1$ and $0 \le b(x) \le 1$, the length of the overall ϵ–support is bounded by*

$$\frac{12||b'||_p}{\epsilon} + \min\left\{2\sqrt{\frac{6||b''||_p}{\epsilon}}, \frac{6\sqrt{2}||b''||_p}{\epsilon\sqrt{V^*}}\right\} + \\ + \min\left\{2\sqrt{\frac{6||V(x)-V^*||_p}{\epsilon}}, \frac{6\sqrt{2}||V(x)-V^*||_p}{\epsilon\sqrt{V^*}}\right\}. \quad (4.8)$$

Proof: To compute the ϵ–support of our bounds, we compute the $(\epsilon/3)$–support of the three terms in Theorem 4.1.1 and take the union. For the $(V - V^*)b$ term we have

$$\frac{||V(x)-V^*||_p}{|V^*-\xi^2|} \ge \frac{\epsilon}{3} \quad (4.9)$$

when

$$\sqrt{V^* - \frac{3||V(x)-V^*||_p}{\epsilon}} \le |\xi| \le \sqrt{V^* + \frac{3||V(x)-V^*||_p}{\epsilon}} \quad (4.10)$$

which is a set of length

$$2\left(\sqrt{V^* + \frac{3||V(x)-V^*||_p}{\epsilon}} - \sqrt{V^* - \frac{3||V(x)-V^*||_p}{\epsilon}}\right). \quad (4.11)$$

When $V^* \le 3||V(x) - V^*||_p/\epsilon$ we drop the second term. This length is bounded by

$$2\sqrt{\frac{6||V(x)-V^*||_p}{\epsilon}}. \quad (4.12)$$

It is also bounded by

$$6\sqrt{2}\frac{||V(x)-V^*||_p}{\epsilon\sqrt{V^*}}. \quad (4.13)$$

This second bound is only useful when $V^* \ge 3||V(x) - V^*||_p/\epsilon$, so we call it a 'high frequency' bound. Both bounds are deduced from (4.11) using only properties of numbers.

Similarly, for the b'' term, we have a set of length

$$2\left(\sqrt{V^* + \frac{3||b''||_p}{\epsilon}} - \sqrt{V^* - \frac{3||b''||_p}{\epsilon}}\right) \le \min\left\{2\sqrt{\frac{6||b''||_p}{\epsilon}}, \frac{6\sqrt{2}||b''||_p}{\epsilon\sqrt{V^*}}\right\}. \tag{4.14}$$

The b' term behaves differently, however.

$$\frac{2|\xi|||b'||_p}{|V^* - \xi^2|} \ge \frac{\epsilon}{3} \tag{4.15}$$

when

$$\frac{3||b'||_p}{\epsilon}\left(\sqrt{1 + \frac{\epsilon^2 V^*}{9||b'||_p^2}} - 1\right) \le |\xi| \le \frac{3||b'||_p}{\epsilon}\left(\sqrt{1 + \frac{\epsilon^2 V^*}{9||b'||_p^2}} + 1\right) \tag{4.16}$$

which is a set of length

$$\frac{12||b'||_p}{\epsilon}.\square \tag{4.17}$$

Remarks:

1. Since the three regions of large size will overlap, a better bound will often be

$$2\max\left\{\frac{12||b'||_p}{\epsilon}, \min\left\{2\sqrt{\frac{6||b''||_p}{\epsilon}}, \frac{6\sqrt{2}||b''||_p}{\epsilon\sqrt{V^*}}\right\},\right.$$
$$\left.\min\left\{2\sqrt{\frac{6||V(x) - V^*||_p}{\epsilon}}, \frac{6\sqrt{2}||V(x) - V^*||_p}{\epsilon\sqrt{V^*}}\right\}\right\}. \tag{4.18}$$

2. The b' term yielded an ϵ–support of length $12||b'||_p/\epsilon$. We may be able to improve this by applying Theorem 4.1.1 again, but we will always be left with a term which, like this one, is independent of V and V^*. This term gives a minimum on the length of the ϵ–support, and can be viewed as a manifestation of the uncertainty principle.

3. The b'' and $(V - V^*)b$ terms have bounds uniform in V^* (assuming $(V - V^*)$ fixed) and 'high frquency' bounds which go to zero as V^* increases. The second bounds lead to a typical asymptotic statement: Our instantaneous frequency is accurate when the change in frequency is small compared to the frequency, i.e. $|V - V^*| \ll \sqrt{V^*}$. Manipulating, this is equivalent to $|\sqrt{V} - \sqrt{V^*}| \ll \sqrt{V^*}/|\sqrt{V} + \sqrt{V^*}| \approx 1$. Note that this has different homogeneity than the results in [LL77], which require only $|\sqrt{V} - \sqrt{V^*}| \ll V^*$.

4.1.3 Number of Terms

Up to now we have been considering the Fourier transform $\widehat{b\psi}$. Since we are actually on an interval, it makes sense to sample $\widehat{b\psi}$ and consider the Fourier series. Instead of considering the length of the ϵ–support as our measure of localization, we consider the number of Fourier coefficients above ϵ.

Definition 4.1.1 *The* **cost** *of an expansion on an interval of length l is $l \cdot |\epsilon\text{–support}|$.*

(We ignore a factor of 2π.)

Theorem 4.1.3 *With ψ and b as in Theorem 4.1.2, the cost of expansion on an interval of length l is at most*

$$2\max\left\{\frac{12||b'||_p}{\epsilon}, \min\left\{2\sqrt{\frac{6||b''||_p}{\epsilon}}, \frac{6\sqrt{2}||b''||_p}{\epsilon\sqrt{l^2V^*}}\right\},\right.$$
$$\left.\min\left\{2l\sqrt{\frac{6||V(x)-V^*||_p}{\epsilon}}, l\frac{6\sqrt{2}||V(x)-V^*||_p}{\epsilon\sqrt{V^*}}\right\}\right\}. \quad (4.19)$$

To deal with intervals of length l, we must adjust our previous bounds. First we must shrink the bell to fit on this interval. We must consider $b(x/l)$ and thus $||(b(\cdot/l))'|| = ||b'||/l$ and $||(b(\cdot/l))''|| = ||b''||/l^2$. Our new b' cost bound is then

$$\frac{12||b'||_p}{\epsilon l}l = \frac{12||b'||_p}{\epsilon} \quad (4.20)$$

which is unchanged. Thus any expansion on any interval has at least this cost (using this method).

From b'' we have a new cost of

$$\min\left\{2\sqrt{\frac{6||b''||_p}{\epsilon}}, \frac{6\sqrt{2}||b''||_p}{\epsilon\sqrt{l^2V^*}}\right\}. \quad (4.21)$$

The uniform bound is unchanged. In the high frequency bound, instead of V^* we now have l^2V^*, which affects how near we are to the asymptotic case. This scaling is consistent with the interpretation of V^* as the square of a frequency.

From $V - V^*$ we now have

$$\min\left\{2l\sqrt{\frac{6||V(x)-V^*||_p}{\epsilon}}, l\frac{6\sqrt{2}||V(x)-V^*||_p}{\epsilon\sqrt{V^*}}\right\}. \quad (4.22)$$

In this case both bounds have been multiplied by l and so improve linearly if we shrink l. Also note that the norm is taken over a different interval. Thus if we shrink l, for some (different) choice of V^*, we can likely make $||V(x)-V^*||_p$ somewhat smaller. How much smaller we can make it depends on the particular $V(x)$ we are dealing with. We will later use our knowledge of $V(x)$ to choose appropriate l and V^*.

Remarks:

1. In applications we will choose $p = \infty$ and $q = 1$. We assumed earlier that $||\psi||_q = 1$. Actually, ψ will be a basis element, and so have $||\psi||_{L^2[0,1]} = 1$. We can bound $||\psi||_{L^1[x,x+l]} \leq$

$||\psi||_2||1||_{L^2[x,x+l]} = \sqrt{l}$. This bound will reduce our costs by a factor of $\sqrt{l}$. This effect is weak, however, and would dissipate when we attempt higher order estimates in Section 4.3.2. We therefore choose not to utilize it in our main development and to relegate its implications to the remarks.

2. The b'' term is unimportant, since its cost in general can be dominated by the b' term. We will often ignore its presence.

4.2 The Intuition

One goal of the above theory is to give rigorous support, where possible, to our intuitive notions about the phase plane. In this section we develop a simple intuitive model and then justify this with our theory.

We would like to have a simple method for computing the cost of expanding a 'chirp', i.e. a function which has 'instantaneous frequency' some function $\nu(x)$. We graph $\nu(x)$ in the $x \times \xi$ phase plane. On the phase plane a Local Cosine (Fourier) basis element is viewed as a rectangle of area 1 with x–support on its base interval I shifted in ξ by its frequency. Intuitively, the boxes which intersect $\nu(x)$ should correspond to Local Cosine elements which yield significant coefficients. If we set $\Delta = \max_I \nu(x) - \min_I \nu(x)$ we can estimate the number of boxes by $\Delta|I| = \Delta l$. We must always intersect at least 1 box, however, so we need to consider $\max\{\Delta l, 1\}$. See Figure 4.1.

It is informative to see how this model behaves under partitioning. First suppose $\nu(x)$ is a line. If we split I in two we get two (equivalent) problems. On the left interval $\nu(x)$ has range $\Delta/2$ and our cost is $\max\{(\Delta/2)(l/2), 1\}$. The right interval is the same, so the total cost is $2\max\{\Delta l/4, 1\} = \max\{\Delta l/2, 2 \cdot 1\}$. If $\Delta l > 2 \cdot 1$ we have reduced our cost. One can easily solve for the optimal number of subintervals to use. We should use $\sqrt{\Delta l/1}$ intervals, giving total cost $2\sqrt{1\Delta l}$. If $\nu(x)$ is monotonic but not a line, partitioning is even more effective, but should not be into intervals of the same size.

4.2.1 Justification

In Section 4.1.2 we bounded the cost of expansion by

$$\frac{C}{\epsilon}\frac{||V(x) - V^*||}{\sqrt{V^*}}. \tag{4.23}$$

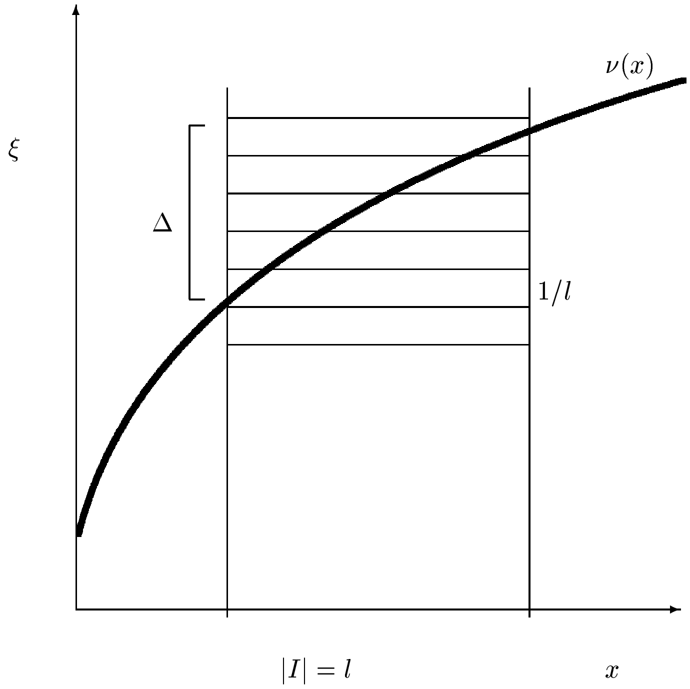

Figure 4.1: Instantaneous frequency cost intuition.

Assuming $V(x) \geq 0$ let $\nu(x) = \sqrt{V(x)}$ and $\sqrt{V^*}$ be the average value of $\sqrt{V(x)}$. Then our cost is

$$\frac{C}{\epsilon}\left|\sqrt{V(x)} - \sqrt{V^*}\right| \frac{\left|\sqrt{V(x)} + \sqrt{V^*}\right|}{\sqrt{V^*}} \leq 2\frac{C}{\epsilon}\left|\sqrt{V(x)} - \sqrt{V^*}\right|. \tag{4.24}$$

In Section 4.1.3 we determined that if the interval is of length l, our cost is multiplied by l, to yield $(2C/\epsilon)l|\sqrt{V(x)} - \sqrt{V^*}|$. Again we let $\Delta = \max_I \nu(x) - \min_I \nu(x)$. The cost to expand on that interval is bounded by $(2C/\epsilon)\max\{l\Delta, B\}$, where B is the minimal bell cost ($||b'||$).

This result is the same as the intuition except that

1. The minimal number of boxes is B instead of 1;

2. We must multiply our estimate by $2C/\epsilon$.

To obtain estimates which depend on ϵ more benignly than $1/\epsilon$, we will apply Theorem 4.1.1 recursively, and so will need derivatives of $V(x)$. This does not lend itself to graphical interpretation, but can be dealt with rigorously, as is done is Section 4.3.2.

Remark: If we use our knowledge that $||\psi||_{L^1[x,x+l]} \leq \sqrt{l}$ we can cut our cost for a linear chirp to $\mathcal{O}(\Delta^{1/4})$ in both the intuitive and rigorous versions. This new cost is misleading as intuiton, however, since the effect dissipates in higher order estimates (Section 4.3.2).

4.3 Associated Legendre Functions

If we look at

$$V(\phi) = (n + 1/2)^2 - \frac{m^2 - 1/4}{\sin^2 \phi} \tag{4.25}$$

on $[0, \pi/2]$, ψ will be an Associated Legendre function of order m, $\sqrt{\sin \phi}\tilde{P}_n^m(\cos \phi)$. We will eventually wish to take many derivatives of $V(x)$ but the $\sin \phi$ will introduce factors of $\cos \phi$ and make the calculations exceedingly messy. Instead we can consider the potential

$$V(x) = (n + 1/2)^2 - \frac{m^2 - 1/4}{x^2} \tag{4.26}$$

which will yield Bessel functions. Qualitatively, these behave the same as the Associated Legendre functions, and so we will have the same overall cost estimate. Making this rigorous would simply introduce some factors of π. See Figure 4.2 for graphs comparing the behavior of an Associated Legendre function and the corresponding Bessel function.

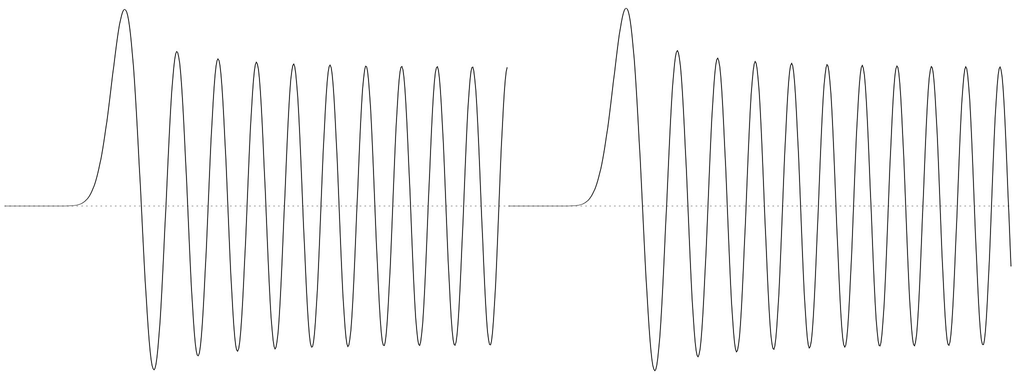

Figure 4.2: $\sqrt{\sin \phi}\tilde{P}_{60}^{20}(\cos \phi)$ and $\sqrt{r}J_{20}(60r)$ on $[0, \pi/2]$

Remark: In the next few sections we will prove compression results for Associated Legendre functions by proving them for Bessel functions. As a consequence, we also obtain a fast version of the Hankel transform for f with compact support. The Hankel tranform is defined as

$$\mathcal{H}_\nu f(y) = \int_0^\infty f(r) J_\nu(ry)\sqrt{ry}\,dr. \tag{4.27}$$

(See [EMOT54, p.3].)

4.3.1 Bessel Functions

We have made a simplification to consider the operator

$$V(x) = (n + 1/2)^2 - \frac{m^2 - 1/4}{x^2} \tag{4.28}$$

on $[0, 1]$. $\psi(x)$ is then the Bessel function of order m, $\sqrt{x}J_m(nx)$. To simplify the following calculations further, we make a further reduction to $V(x) = n^2 - (m/x)^2$. We will not use any algebraic properties of the values of m and n, so this reduction is unimportant. We will need to keep this reduction in mind when applying our results to very small values of m.

The tools we developed above give bounds on the cost (number of coefficients) for a given interval. Our only real freedom is how we partition $[0, 1]$. Our total cost is the sum of the costs from each subinterval. We would like to find the partition which minimizes our total cost. This problem is too hard, however, so we design another scheme, hoping it will be nearly optimal. If not, it still yields upper bounds of the type we need. We will prove the following:

Theorem 4.3.1 *Let ψ be a solution to $\psi'' = (n^2 - (m/x)^2)\psi$ on N points in $[0, 1]$, normalized so $||\psi||_1 \leq 1$. Then there exists a partition of $[0, 1]$ which yields total expansion cost less than*

$$2\sqrt{m}\frac{\sqrt{||b'||}}{\epsilon}\log N. \tag{4.29}$$

The first part of our scheme is a stopping time argument. We fix a cost C to be the maximum allowed cost. Given an interval, if its cost is less than C we keep it; otherwise split it in two and pass them to the next stage. Start with $[0, 1]$ and continue until stopped. First we must argue that we do indeed stop, so that we obtain a finite partition into intervals with individual cost at most C. Second, we must count the number of intervals we kept. Multiplying this by C will give us the total cost.

In deciding whether we stop there are two considerations. First is the bell cost. If we split an interval in two, our non–bell cost (4.22) shrinks by a factor of at least two, but the bell cost (4.20) is unchanged. We will stop if and only if C is greater than the bell cost $||b'||/\epsilon$. The second consideration is that we will be working on a finite number of points N. Each interval we choose must contain a point, so we can have at most $\log_2 N$ levels anyway. (This argument was inspired by [Wol95].) Once the number of points in an interval is less than C, it cannot cost more than C in any case. We will also use this later when we count intervals.

Before trying to count the intervals, we consider the structure of the partition we expect to be chosen. When $x < m/n$, ψ decays rapidly and smoothly and can be expanded into Local Cosine with negligible cost. We thus consider this a 'nothing' interval and only worry about $[m/n, 1]$. For a fixed interval length $l = 2^{-j}$ there will be some set in $[m/n, 1]$ such that the interval $[x, x+l]$ has cost less than C. Using the high frequency bound (4.13), we have cost

$$\leq \frac{2^{-j}||V - V^*||}{\epsilon\sqrt{V^*}}. \tag{4.30}$$

Since V is concave down, we can bound $||V - V^*||$ on $[x, x+2^{-j}]$ by $2^{-j}V'(x)$, choosing $V^* = V(x)$. We then have cost

$$\leq \frac{2^{-2j}m^2}{\epsilon x^2\sqrt{(nx)^2 - m^2}}. \tag{4.31}$$

Note that this cost shrinks as x increases. The 'good' set for size 2^{-j} (G_j) is thus some interval $[x_j, 1]$. The remaining interval $[m/n, x_j]$ is the 'bad' set (B_j). See Figure 4.3.

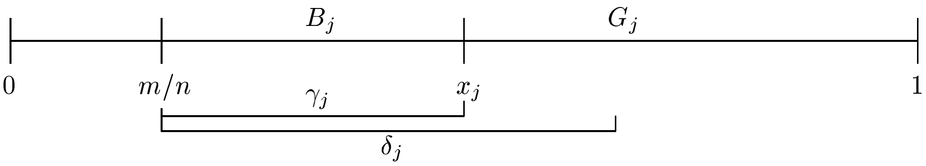

Figure 4.3: Good and bad intervals for size 2^{-j}

It will be most efficient if we break the interval $B_{j-1} \cap G_j = [x_j, x_{j-1}]$ into intervals of size 2^{-j}. It will take $2^j|B_{j-1} \cap G_j|$ such intervals, yielding cost $C2^j|B_{j-1} \cap G_j|$. To estimate $|B_{j-1} \cap G_j|$, we first dominate it by $|B_{j-1}| = x_{j-1} - m/n = \gamma_{j-1}$. Setting our cost bound equal to C and manipulating, we find

$$\frac{2^{-j}m}{\sqrt{C\epsilon}} = x_j((x_j n)^2 - m^2)^{1/4} \tag{4.32}$$

$$= (\gamma_j + m/n)((\gamma_j n + m)^2 - m^2)^{1/4} \tag{4.33}$$

$$= (\gamma_j + m/n)(\gamma_j n)^{1/4}(\gamma_j n + 2m)^{1/4}. \tag{4.34}$$

γ_j is difficult to solve for, but it will be dominated by δ_j, which satisfies

$$\frac{2^{-j}m}{\sqrt{C\epsilon}} = \delta_j^{3/4}(m/n)^{1/4}(\delta_j n)^{1/4}m^{1/4} = \delta_j m^{1/2} \tag{4.35}$$

which means $\delta_j = 2^{-j}\sqrt{m/C\epsilon}$.

We can now bound $[x_j, x_{j-1}]$ by δ_{j-1}. Our cost for this interval using subintervals of size 2^{-j} is thus at most

$$2^{-j+1}\sqrt{\frac{m}{C\epsilon}} \cdot 2^j \cdot C = 2\sqrt{m}\sqrt{\frac{C}{\epsilon}}. \tag{4.36}$$

As noted above, $j \leq \log N$ and thus we can sum over j and bound our overall cost by

$$2\sqrt{m}\sqrt{\frac{C}{\epsilon}}\log N. \tag{4.37}$$

If we let $C \to 0$, this cost goes to 0, which must be nonsense. Of course we cannot let C go to 0, because the bell terms give us a minimal cost per interval $||b'||/\epsilon$. Plugging this in, we obtain total cost

$$2\sqrt{m}\frac{\sqrt{||b'||}}{\epsilon}\log N. \tag{4.38}$$

Remarks:

1. According to these estimates, our best results are obtained by shrinking the intervals until the non–bell cost equals the bell cost. Since the bell cost is independent of the interval size, this indicates that our choice to have all intervals of the same cost should be optimal.

2. If we use our bound $||\psi||_{L^1[x,x+l]} \leq \sqrt{l}$, we obtain cost $2^{-j3/4}\sqrt{m}\sqrt{C/\epsilon}$ for $[x_j, x_{j-1}]$ and total cost $(4/3)\sqrt{m}\sqrt{C/\epsilon}$.

4.3.2 Higher Order Estimates

The above estimates are fine, but depend on ϵ like ϵ^{-1}. Here we apply Theorem 4.1.1 repeatedly to get higher order decay in ξ, and thus more benign dependence on ϵ.

If we apply Theorem 4.1.1 one time, we convert a single term into 3 terms. One of these has multiplicity 2, so we'll consider there to be 4 terms. If we apply the theorem k times, we expect 4^k terms. Along the way we take derivatives of products and powers, which will give us further multiplicities. In going from step k to step $k+1$, we multiply by at most $(2k)(2k+1)$. Our total number of terms, including multiplicities, is thus bounded by $4^k(2k)!$. Within some terms we will also generate a factor of $k!$. We account for this now so that when it appears later, we can ignore it. We therefore proceed as if there were $4^k(2k)!k!$ terms.

Each term has the following form:

$$\frac{|\xi|^a \cdot ||b^{(b)}|| \cdot l^{2k-(a+b)} \cdot ||V - V^*||^c \cdot |V^{(1)}|^{d_1} \cdot |V^{(2)}|^{d_2} \cdots}{|\xi^2 - l^2 V^*|^k} \tag{4.39}$$

with $a+b+2c+\sum d_i(i+2)=2k$, $a \leq k$, and if $a=k$ then $b=k$. We wish to know when this is greater than $\epsilon/4^k(2k)!k!$. We will separate each term into k factors and determine when each is greater than $(\epsilon/4^k(2k)!k!)^{1/k} \approx \epsilon^{1/k}/k^3 = \eta$. (We use Stirling's formula $n! \approx \sqrt{2\pi}n^{n+1/2}e^{-n}$.) The η–support of each factor is two intervals, containing the points $\pm l\sqrt{V^*}$. The union of these intervals is essentially the largest pair of intervals, and will contain the intervals when our term is greater than $\epsilon/4^k(2k)!k!$.

Following Section 4.3.1, we fix a sub–cost C and determine the smallest x which, given an l, yields cost C for this factor. We will have some factors whose cost is independent of l and x (and $V(x)$) and these will act as a minimum on C. When we can affect the cost with l, we will show that l is a linear function of x. It may have x–intercept at 0, in which case its slope is independent of m and n. It could instead have x–intercept at m/n, in which case its slope depends on m like $m^{-1/2}$. As we saw in Section 4.3.1 and using (4.36), a linear dependence between l and x (or $(x-m/n)$) means our total cost will depend on N like $\log N$. The total cost also depends on the slope of the line like its inverse. For us this is either independent of m or depends on m like $\sqrt{m}$. Given this collection of lines, we can choose a line which is under all of them on $[m/n, 1]$, and the inverse of its slope will be $\mathcal{O}(\sqrt{m})$. This line gives us a partition of $[m/n, 1]$ for which all factors obey the sub–cost limit C. A closer examination of its slope will tell us our dependence on C, ϵ, etc.

We organize the various types of factors in tables. Table 4.1 contains the basis factors, the bound used for that factor, the corresponding line, and the total cost if that was the dominant line. The bounds are analogous to those in (4.13), (4.12), and (4.17). The lines are found by either setting the bound equal to C and solving for l, or by using the δ construction as in (4.35). The total cost is found by taking the inverse of the slope and multiplying by C. We will also use the shorthand

$$b_k = \sup_{1\leq j\leq 2k} ||b^{(j)}||^{1/j}. \tag{4.40}$$

Next, in Table 4.2, we list the more complicated, composite terms and show how to factor them. The terms of type (4.47), (4.51), and (4.52) have factorials, but these were accounted for at the beginning and incorporated into η, and so can be ignored.

For large m, the worst total is in terms of type (4.44) and is $\sqrt{mC/\eta}$. The factors of type (4.41) and (4.42) give us bounds independent of l, and thus tell us we should choose $C = b_k/\eta$. Plugging in $C = b_k/\eta$ and $\eta = \epsilon^{1/k}/k^3$ yields $\sqrt{mb_k}k^3\epsilon^{1/k}$. To estimate b_k, we note $b^{(j)}$ should

Factor	Bound used	Line	Total cost	
$\frac{\lvert\xi\rvert b_k}{\lvert\xi^2 - l^2V^*\rvert}$	$\frac{b_k}{\eta}$			(4.41)
$\frac{b_k^2}{\lvert\xi^2 - l^2V^*\rvert}$	$\frac{b_k}{\sqrt{\eta}}$			(4.42)
$\frac{\lvert\xi\rvert lx^{-1}}{\lvert\xi^2 - l^2V^*\rvert}$	$\frac{lx^{-1}}{\eta}$	$l = \eta Cx$	$\frac{1}{\eta}$	(4.43)
$\frac{l^3m^2x^{-3}}{\lvert\xi^2 - l^2V^*\rvert}$	$\frac{l^2m^2}{\eta x^2\sqrt{(nx)^2 - m^2}}$	$l = \sqrt{\frac{C\eta}{m}}\left(x - \frac{m}{n}\right)$	$\sqrt{\frac{mC}{\eta}}$	(4.44)
$\frac{lx^{-1}}{\lvert\xi^2 - l^2V^*\rvert}$	$\sqrt{\frac{lx^{-1}}{\eta}}$	$l = \eta C^2 x$	$\frac{1}{C\eta}$	(4.45)
$\frac{l^2x^{-2}}{\lvert\xi^2 - l^2V^*\rvert}$	$\frac{lx^{-1}}{\sqrt{\eta}}$	$l = \sqrt{\eta}Cx$	$\frac{1}{\sqrt{\eta}}$	(4.46)

Table 4.1: Basic Factors.

have j peaks, about $1/j$ apart. The height of these peaks can be found from the height of $b^{(j-1)}$'s peaks by multiplying by the previous separation $(1/(j-1))$ and an extra factor of two. Inductively this gives $|b^{(j)}| \sim 2^j j!$ and so $b_k \sim 4k$. Note that this is really a lower bound on b_k. Since b is our choice, and can be chosen after ϵ and k, we should be able to get near this estimate. Plugging in this estimate yields cost $\sqrt{m}k^{7/2}\epsilon^{1/k}$. Choosing $k = \log(1/\epsilon)$ gives us a compression cost of $\mathcal{O}(\sqrt{m}(\log\epsilon)^{7/2}\log N)$.

Remarks:

1. If we use our bound $||\psi||_{L^1[x,x+l]} \le \sqrt{l}$ for these higher order estimates, we must factor its effect among k terms. This multiplies our cost for $[x, x+l]$ by $2^{-j/k}$, which allows us to sum over all j. We can thus dispense with our factor of $\log N$, but we gain a factor of $k = \log\epsilon$.

2. The estimate $b_k \sim 4k$ is far from rigorous. In applications, we use bells which are optimized (Section 5.3) using a method that does not look at derivatives of the bell. We have found these bells to be highly efficient.

4.3.3 One Dimensional Compression Conclusion: $\mathcal{O}(N^{5/2}\log N(\log\epsilon)^{7/2})$

We concluded above that we can compress each Associated Legendre function into at most

$$\mathcal{O}(\sqrt{m}(\log\epsilon)^{7/2}\log N) \le \mathcal{O}(\sqrt{N}(\log\epsilon)^{7/2}\log N) \tag{4.53}$$

Term	Factorization	
$\dfrac{\|\xi\|^p \cdot \|\|V^{(p)}\|\| \cdot l^{p+2}}{\|\xi^2 - l^2V^*\|^{p+1}}$	$= \dfrac{m^2l^3x^{-3}}{\|\xi^2 - l^2V^*\|} \dfrac{\|\xi\|}{\|\xi^2 - l^2V^*\|} \left(\dfrac{\|\xi\|lx^{-1}}{\|\xi^2 - l^2V^*\|}\right)^{p-1} (p+2)!$	(4.47)
$\dfrac{\|\|V - V^*\|\|}{\|\xi^2 - l^2V^*\|}$	$\leq \dfrac{m^2l^3x^{-3}}{\|\xi^2 - l^2V^*\|}$	(4.48)
$\dfrac{\|\xi\|^p\|\|b^{(p)}\|\|}{\|\xi^2 - l^2V^*\|^p}$	$\leq \left(\dfrac{\|\xi\|b_k}{\|\xi^2 - l^2V^*\|}\right)^p$	(4.49)
$\dfrac{\|\xi\|^p\|\|b^{(p+q)}\|\|}{\|\xi^2 - l^2V^*\|^{p+q/2}}$	$\leq \left(\dfrac{\|\xi\|b_k}{\|\xi^2 - l^2V^*\|}\right)^p \left(\dfrac{b_k^2}{\|\xi^2 - l^2V^*\|}\right)^{q/2}$	(4.50)
$\dfrac{l^{p+2}\|\|V^{(p)}\|\|}{\|\xi^2 - l^2V^*\|^{1+p/2}}$	$= \dfrac{m^2l^3x^{-3}}{\|\xi^2 - l^2V^*\|} \dfrac{lx^{-1}}{\|\xi^2 - l^2V^*\|} \left(\dfrac{l^2x^{-2}}{\|\xi^2 - l^2V^*\|}\right)^{p/2-1} (p+2)!$	(4.51)
$\dfrac{\|\xi\|^p l^{p+q+2}\|\|V^{(p+q)}\|\|}{\|\xi^2 - l^2V^*\|^{p+1+q/2}}$	$= \dfrac{m^2l^3x^{-3}}{\|\xi^2 - l^2V^*\|} \dfrac{\|\xi\|}{\|\xi^2 - l^2V^*\|} \left(\dfrac{\|\xi\|lx^{-1}}{\|\xi^2 - l^2V^*\|}\right)^{p-1} \left(\dfrac{l^2x^{-2}}{\|\xi^2 - l^2V^*\|}\right)^{q/2} (p+q+2)!$	(4.52)

Table 4.2: Composite Terms.

coefficients. We must add this count over N values of m and N values of n. For the complete Spherical Harmonic transform, this yields a bound of

$$\mathcal{O}(N^{5/2} \log N (\log \epsilon)^{7/2}). \tag{4.54}$$

This cost will dominate the time for the fast Fourier transforms used, and so will be the order of the one dimensional algorithm in Section 3.2.

4.4 Two Dimensional Compression: $\mathcal{O}(N^2 \log N (\log \epsilon)^7)$

The compression scheme described above involves compressing $\sqrt{\sin\phi}\tilde{P}_n^m(\cos\phi)$ as a function of ϕ, and so is one–dimensional. We are actually applying a matrix of $\sqrt{\sin\phi}\tilde{P}_n^m(\cos\phi)$ (m fixed) and so can consider this as a function of both n and ϕ and compress in both directions. In the n direction we don't have a differential equation, but instead a recurrence relation (see Section 5.1). If we model with Bessel functions, however, we are dealing with $\sqrt{nx}J_m(nx)$ for which n and y are interchangeable. Setting $y = n/N$ we consider $\sqrt{yxN}J_m(yxN)$ on $(x, y) \in [0,1] \times [0,1]$ which is fully symmetric. Obtaining rigorous results for the Associated Legendre functions from results for Bessel functions in the n variable seems difficult, but we expect the behavior to be the same. We state the result for Bessel functions:

Theorem 4.4.1 *Let $\psi(x,y) = \sqrt{yxN}J_m(yxN)$ on the unit square, normalized so $||\psi(\cdot,y)||_1 \leq 1$. There exists a partition into squares with total expansion cost bounded by*

$$\mathcal{O}(N\log N(\log\epsilon)^7). \tag{4.55}$$

In Section 4.1 we constructed tools for one dimensional compression and in Section 4.3 we applied them. Here we will try to use these tools and techniques as much as posssible and only highlight the differences.

We are allowed to choose any partition of the unit $(x \times y)$ square into rectangles, but for now we restrict ourselves to squares of sidelength 2^{-j}. We first note that the turning point occurs when $x = m/yN$. Anything above or to the left of this hyperbola is either rapidly decaying or zero, so we don't have to worry about this region. (Recall that since M is a matrix, the x axis points down.)

First we fix a sub–cost C^2 as our goal on the subsquares. Fixing j, we can break the unit square into three regions. The first is a 'good' region for the size 2^{-j}, denoted G_j, where the cost is less than C^2. The second is a 'bad' region (B_j) where the bounds fail, and the third is a 'nothing' region beyond the turning point. We break the unit square into the nothing region and $G_0 \cup_{j=0}^{\infty}(B_j \cap G_{j+1})$. To compute the number of squares needed for a region, we can compute the area of that region and divide it by the area of the squares used. Thus for the region $B_j \cap G_{j+1}$ we need $|B_j \cap G_{j+1}| \cdot 2^{2j+2}$ squares. Multiplying by C^2 will give the number of coefficients. Our problem is now reduced to estimating the area of these regions.

This first step is to assess the cost associated with the square $[x, x+2^{-j}] \times [y, y+2^{-j}]$. This cost will be bounded by

$$\left|\bigcup_{0\leq t\leq 2^{-j}} 2^{-j}\epsilon\text{–sup}\left((b_{[x,x+2^{-j}]}\psi^m_{y+t}(\cdot))\hat{}\right)\right| \cdot \left|\bigcup_{0\leq s\leq 2^{-j}} 2^{-j}\epsilon\text{–sup}\left((b_{[y,y+2^{-j}]}\psi^m_{(\cdot)}(x+s))\hat{}\right)\right| \tag{4.56}$$

If we require this to be less than C^2 it suffices to have each factor less than C. When computing the ϵ–supports, the only dependence on m was through $||V(x) - V^*||$. Recalling now that V also depends on y, we instead consider $\sup|V_y(x) - V^*|$ taken over the square $[x, x+2^{-j}] \times [y, y+2^{-j}]$ with V^* independent of y. The cost computed using this will dominate the first factor in (4.56).

For our particular potential, $V_y(x) = (yN)^2 - m^2/x^2$, we can choose $V^* = (yN)^2 - m^2/x^2$ and achieve the supremum with

$$V_{y+2^{-j}}(x+2^{-j}) - V^* = y^2N^2 + 2y2^{-j}N^2 + l^2N^2 - \frac{m^2}{(x+2^{-j})^2} - yN^2 + \frac{m^2}{x^2} \tag{4.57}$$

$$= m^2\frac{2^{-j}(2x+2^{-j})}{x^2(x+2^{-j})^2} + 2y2^{-j}N^2 + l^2N^2. \tag{4.58}$$

We will consider the costs from these three terms separately. The bad regions for the three terms will not coincide, and the full region B_j will be the union of these. We will show the three bad regions shrink at least at a certain rate, which implies their union also shrinks at this same rate. We then use $|B_j|$ to dominate $|B_j \cap G_{j+1}|$.

The first term gives cost

$$\frac{m^2 2^{-j}(2x+2^{-j})}{x^2(x+2^{-j})^2}\frac{2^{-j}}{\sqrt{(yN)^2 - m^2/x^2\epsilon}}. \tag{4.59}$$

We set this equal to C, define $\gamma = x - m/yN$ and manipulate to find

$$\frac{m^2(2^{-j})^2}{C\epsilon} = \frac{(\gamma + 2^{-j} + m/yN)^2\sqrt{\gamma yN}\sqrt{\gamma yN + 2m}}{2\gamma + 2^{-j} + 2m/yN}. \tag{4.60}$$

Following the argument from Section 4.3.1, we note it is sufficient to use δ such that

$$\frac{m^2(2^{-j})^2}{C\epsilon} = \delta^{1/2}\left(\frac{m}{yN}\right)^{1/2}\frac{m}{N}\sqrt{\delta yN}\sqrt{m} \tag{4.61}$$

which means $\delta = N(2^{-j})^2/C\epsilon$ will suffice. This $\delta = \delta_j$ bounds the thickness of the strip B_j. Since this strip has length at most 1, δ_j also bounds its area. Our cost for the region $B_j \cap G_{j+1}$ is thus less than $\delta(2^{-j})^{-2} = NC/\epsilon$. Since a square must contain a point, $j \le \log N$, and we can add our cost over $\log N$ strips to obtain total cost $(C/\epsilon)N\log N$. See Figure 4.4 for a schematic of the different regions and Figure 2.2 for the original matrix.

The second term gives cost

$$y2^{-j}N^2\frac{2^{-j}}{\sqrt{(yN)^2 - m^2/x^2\epsilon}}. \tag{4.62}$$

We set this equal to C, define $\gamma = x - m/yN$ and manipulate to find

$$\frac{(2^{-j})^2yN^2}{C\epsilon} = \frac{\sqrt{\gamma yN}\sqrt{\gamma yN + m^2}}{(\gamma + m/yN)^2}. \tag{4.63}$$

It will suffice to take δ such that

$$\frac{(2^{-j})^2yN^2}{C\epsilon} = \delta yN \tag{4.64}$$

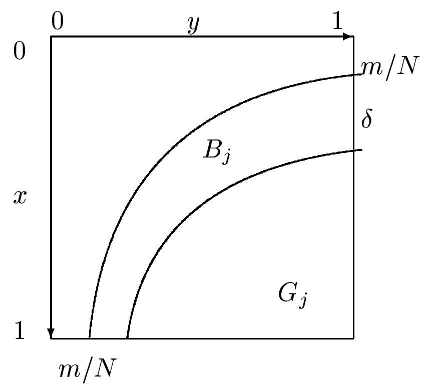

Figure 4.4: The good and bad regions for a certain sized square.

and so $\delta = (2^{-j})^2 N/C\epsilon$ will work. This yields NC/ϵ as the cost for a strip and total cost $(C/\epsilon)N \log N$.

The third term gives cost

$$(2^{-j})^2 N^2 \frac{2^{-j}}{\sqrt{(yN)^2 - m^2/x^2\epsilon}}. \tag{4.65}$$

We set this equal to C, define $\gamma = x - m/yN$ and manipulate to find

$$\frac{(2^{-j})^3 N^2}{C\epsilon} = \frac{\sqrt{\gamma y N}\sqrt{\gamma y N + m^2}}{(\gamma + m/yN)^2}. \tag{4.66}$$

It will suffice to take δ such that

$$\frac{(2^{-j})^3 N^2}{C\epsilon} = \delta y N \tag{4.67}$$

which means $\delta y = (2^{-j})^3 N/C\epsilon$ will work. Anticipating the results from the analysis of the first term in the compression in y, we assume $y \geq \delta + m/x \geq \delta$ (for the same δ) and so we can take $\delta^2 = (2^{-j})^3 N/C\epsilon$ and

$$\delta = \frac{(2^{-j})^2 N}{\sqrt{C\epsilon}} \frac{1}{\sqrt{2^{-j}N}}. \tag{4.68}$$

Noting $2^{-j} \geq 1/N$ and so $1/\sqrt{2^{-j}N} < 1$, we can take $\delta = (2^{-j})^2 N/\sqrt{C\epsilon}$, yielding total cost $(C^{3/2}/\epsilon)N \log N$.

As in the one dimensional case, we can use Theorem 4.1.1 iteratively to obtain higher order decay. Excluding the case (4.44) dealt with above, all the basic factors in Table 4.1 yielded lines of the form $x = lA = 2^{-j}A$ with A independent of m, n, y, etc. These lines give us strip cost of $A/2^{-j}$ and total cost $A\sum_{j=0}^{\log N}(2^{-j})^{-1} = AN$.

We can thus expect ϵ dependence like $((\log \epsilon)^{7/2})^2$. For each m we can compress into

$$\mathcal{O}(N \log N (\log \epsilon)^7) \tag{4.69}$$

coefficients and so have total coefficient count

$$\mathcal{O}(N^2 \log N (\log \epsilon)^7). \tag{4.70}$$

Thus for the two dimensional algorithm in Section 3.3, the FFT time of $\mathcal{O}(N^2 \log^2 N)$ will dominate the time to apply the compressed matrix.

Chapter 5

Computational Details

This chapter is meant to gather information and results that, although not necessary for the theoretical result of a fast transform, are essential to its practical application.

5.1 Generating the $\tilde{P}_n^m$'s

Before we compress a matrix of Associated Legendre functions (or ψ_n^m's) we must be able to construct the matrix. Let $\tilde{P}_k^{(m,m)}$ be the L^2 normalized Jacobi Polynomial. We can construct the L^2 normalized Associated Legendre functions for $m \geq 0$ as:

$$\tilde{P}_n^m(\cos\phi) = (\sin\phi)^m \tilde{P}_{n-m}^{(m,m)}(\cos\phi) \tag{5.1}$$

and for $m < 0$ using $\tilde{P}_n^m(\cos\phi) = \tilde{P}_n^{-m}(\cos\phi)$. Using the recurrence relation and normalization from Szegö, [Sze75, p.68,71] we can construct $\tilde{P}_k^{(m,m)}$ using the recurrence initialized by

$$\tilde{P}_{-1}^{(m,m)}(x) \equiv 0 \tag{5.2}$$

$$\tilde{P}_0^{(m,m)}(x) = \sqrt{\frac{\Gamma(2m+2)}{2}}\frac{1}{2^m\Gamma(m+1)} = \frac{1}{\sqrt{2}}\prod_{j=1}^{m}\sqrt{1+\frac{1}{2j}} \tag{5.3}$$

and with general term

$$\begin{aligned}\tilde{P}_k^{(m,m)}(x) = 2x\tilde{P}_{k-1}^{(m,m)}(x)\left(1+\frac{m-1/2}{k}\right)^{1/2}\left(1-\frac{m-1/2}{k+2m}\right)^{1/2} \\ - \tilde{P}_{k-2}^{(m,m)}(x)\left(1+\frac{4}{2k+2m-3}\right)^{1/2}\left(1-\frac{1}{k}\right)^{1/2}\left(1-\frac{1}{k+2m}\right)^{1/2}\end{aligned} \tag{5.4}$$

For large m this recurrence is poorly conditioned, especially when $m \leq n \leq 2m$. The solution we want corresponds to the larger eigenvalue, however, so the conditioning acts in our favor. The recurrence is also highly prone to underflows. We fix this by using scientific notation,

$\sqrt{\sin\phi}\tilde{P}_n^m(\cos\phi) = A2^B$, $1/2 < A \leq 1$, $B \in \mathbf{Z}$, and storing A and B separately. (We actually pull the exponent B from $|\sqrt{\sin\phi}\tilde{P}_n^m(\cos\phi)| + |\sqrt{\sin\phi}\tilde{P}_{n-1}^m(\cos\phi)|$ in order to avoid problems near zeroes.)

5.2 The Quadrature

When evaluating a series of Associated Legendre functions, we simply evaluate, and there is nothing to worry about. When trying to compute the coefficients of a given function, however, we need to worry whether the discrete integral (sum) is the same as the analytic integral. This brings us to issues of sampling, aliasing, etc. What we would like is a sampling theorem with associated quadrature weights which will make the two integrals equal.

Driscoll and Healy [DH94] produce such a theorem and quadrature weights. They only claim this result for the case $m = 0$ but it actually holds for all m ($|m| < N/2$). We will now go through the proof of this.

They observe that we are essentially dealing with integrals of trigonometric polynomials. The only hindrance to using the usual (Shannon's) sampling theorem is that we are on $[0, \pi]$ instead of $[-\pi, \pi]$ and we have an extra factor of $\sin\phi$ in the measure. Otherwise, we will be integrating the product of two Associated Legendre functions of the same order, and this is always a polynomial in $\cos\phi$. If we extend our functions and measure even across 0, we correct the period and merely double the integral we want. We are now multiplying the functions by the weight $\mathrm{sgn}\phi \sin\phi$.

We have N points in $[0, \pi]$ and we restrict the degree $n < N/2$. Multiplying two such $\tilde{P}_n^m$'s gives us a polynomial in cosine of degree less than $N - 1$, which we will call L_N. Now let Π_N be the (analytic) projection of $\mathrm{sgn}\phi$ onto the frequencies less than N. When we integrate, we would like to obtain the zero frequency coefficient of $L_N \mathrm{sgn}\phi \sin\phi$. We claim, however, that this is identical with the zero frequency coefficient of $L_N \sin\phi \Pi_N$. The higher frequencies in $\mathrm{sgn}\phi$ that we removed, when multiplied by any frequency in $L_N \sin\phi$, would yield a non–zero frequency, which would not contribute to the integral. Now we simply note that $L_N \sin\phi \Pi_N$ is of degree less than $2N$ and we are on $2N$ points. It will alias, but not enough to affect the zero frequency. (The $2N$ frequency, which we don't have, aliases to the zero frequency.) We can thus appeal to the usual sampling theorem for exponentials.

Now we need simply to compute the expansion of $\mathrm{sgn}\phi$ and track down constants. Driscoll and Healy do this [DH94, p.216] and produce a formula which we modify slightly for our context to obtain:

$$w_N(j) = \frac{2\sqrt{2}}{N} \sum_{l=0}^{N/2-1} \frac{1}{2l+1} \sin((2l+1)\phi_j). \tag{5.5}$$

See Section 2.4.2 for its proper use.

Remark: An alternative approach would be to use a Gaussian quadrature. (See e.g. [FPTV92].) For $n, n' < N/2$, $\tilde{P}_n^m(\cos\phi)\tilde{P}_{n'}^m(\cos\phi) = P(\cos\phi)$ is a polynomial in $\cos\phi$ of degree at most $N-2$. We wish to compute the integral

$$\int_0^\pi \tilde{P}_n^m(\cos\phi)\tilde{P}_{n'}^m(\cos\phi)\sin\phi d\phi = \int_{-1}^1 P(x)dx. \tag{5.6}$$

Using Gaussian nodes and weights (in x) we only need $N/2$ points, as opposed to the N points required above. Returning to the sphere, our grid is not equispaced, and so we must redefine ϕ_j. This will distort the computation of Local Cosine coefficients. We also expect aliasing effects.

5.3 Bells

In determining the order of our algorithm, the bell used is unimportant, except perhaps that it have a few bounded derivatives. In determining the constants involved, however, it it crucial. Matviyenko [Mat96] has worked to find the most efficient bell for expanding a single trigonometric function of arbitrary phase and frequency in the Local Cosine basis. Given a target number of parameters B one can solve a linear system to find the bell which puts the most mass in the first B frequencies, thus leaving the rest of the coefficients small. Given ϵ one can find the value of B which gives the bell which yields the fewest cofficients greater than ϵ. The solution depends strongly on ϵ, but can be made very efficient.

He restricts to bells with sum (not squared) of 1, which puts us in the bi–orthogonal case. Defining $w(x) = \sum_i b_i^2(x)$, we note that $b_i(x)/\sqrt{w(x)}$ is an ordinary Local Cosine bell. Thus we have the identity:

$$f(x)\sqrt{w(x)} = \sum_{i,p} \langle f\sqrt{w}, \frac{b_i}{\sqrt{w}} c_i^p \rangle \frac{b_i(x)}{\sqrt{w(x)}} c_i^p(x) \tag{5.7}$$

$$f(x) = \sum_{i,p} \langle f, b_i c_i^p \rangle \frac{b_i(x)}{w(x)} c_i^p(x) \tag{5.8}$$

We thus have another representation for f. We expand using the bells $\{b_i(x)\}$ and reconstruct with the *dual* bells $\{b_i(x)/w(x)\}$. The square sum of the coefficients will yield the L^2 norm not of f, but of $f\sqrt{w}$.

The use of bi–orthogonal bells has condition number the maximum value of $1/\sqrt{w}$. With the restriction that the bells are positive and have sum 1, this can be at most $\sqrt{2}$, and so will never be a problem. We therefore pretend as though we are using orthogonal bells.

5.4 The One Dimensional Search

In Section 4.3 we developed a theory which said we can achieve a certain amount af compression if we can find an optimal partitioning of the interval. It gave some indication what that partition should look like but, because of various unknown constants, should not be used to determine the actual partition.

We would like to find the optimal partition by searching. To make our search space managable, we first restrict all intervals to being dyadic in size and position. This restriction also means that the set of all intervals chosen (for fixed m) is small enough so we can do the local trigonometric expansions (evaluations) in $\mathcal{O}(N \log^2 N)$ time.

We now have a choice on how to proceed. One option is to use double sized bells as developed in Section 3.3.1 and tested in Section 6.1.2. The advantage of this is that we can use a standard search algorithm ([Wic94, p.283]). The disadvantage is a factor of two inefficiency in our compression. The other option is to insist on orthogonality. For this we need all our bells to be compatible, and yet they are supposed to be dilations of the same bell, a seemingly impossible condition. To circumvent this note that the interval size should be increasing (slowly) as we move to the right. With our dyadic restriction this really means non–decreasing. An interval's left neighbor is thus either the same size or one size smaller, so if we can choose from two bells, we can make things compatible.

We also must deal with edge effects, since our basis functions stop abruptly at the edges of the interval. At the left edge, we can localize this effect by using very small intervals. Our functions have a zero of order $m + 1/2$ at this edge anyway, so for any significant m, this may as well be smooth. At the right edge ($\pi/2$) we cannot use small intervals and we don't have high order zeroes. Instead we note that our function is either even or odd across this boundary (as $n - m$ is even or odd). (See Section 2.4.4.) We can choose our local trigonometric basis to agree with this, and then dispense with bells at this edge. When the function is odd we use the usual Local Cosine basis (3.1). When the function is even, we replace the frequency factor $(p + 1/2)$ with p.

Because the search occurs in the precomputation stage, it is allowed to be slow. As a practical matter, however, we would like it to be fast, and so here present a fast algorithm. For our purposes, best is determined by fewest coefficients above ϵ, but the search generalizes to other cost functions.

5.4.1 A Best Non–Decreasing Dyadic Partition Search

For a given function on $[0,1]$, we wish to find the best dyadic Local Cosine basis with the following restrictions:

1. An interval's right neighbor can either be the same size or one size larger.

2. The bells are as large as possible.

We determine 'best' using any additive cost function, such as entropy, $||\cdot||_p^p$, or the number of coefficients above a threshhold.

In the construction of the Local Cosine basis (Section 3.1) we require bells from nonadjacent intervals to have disjoint support. Now we impose the slightly stronger condition that the bell does not extend past the midpoint of the adjacent interval. With this restriction and symmetry properties of the bells, the largest bell possible for an interval will be supported on the double of the interval. We call this a 'full bell'. We may also encounter the situation where the left neighbor of our interval is half its size. Then we adjust the left half of our bell to be compatible with the full bell of the smaller neighbor. We then have an 'asymmetric bell'. Each dyadic interval will have two sets of coefficients, one for each bell. In some cases (smallest interval size), the asymmetric bell does not occur. (See Figure 5.1. Boxes with X's are not used.)

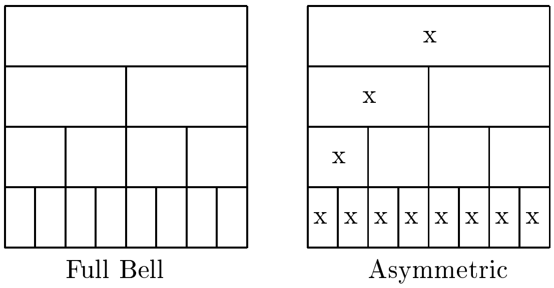

Figure 5.1: The coefficient organization scheme.

A standard best basis search grows a tree with top node corresponding to the largest interval and each node with two children, corresponding to the interval's left and right halves. We then work from the bottom. The bottom nodes are given the costs of the corresponding intervals. We then compare the sum of the costs of the children *nodes* to the cost of the parent *interval*, place the smaller in the parent *node*, and sever the children. Eventually we are left with the minimal cost in the root node. Some bookkeeping will allow us to find the corresponding partition. This method depends heavily on the independence of the various choices made. In our case we have dependency, so this cannot work.

Instead we place the root node to the left of our interval. It has $\log(N)$ children, which correspond to the leftmost dyadic intervals. Now working left to right, we assign each node children corresponding to allowed right neighbors of the current interval. (See Figure 5.2.) Intervals at the right edge have no children. All others have a child which is its right neighbor of the same size. Some may also have a child which is its right neighbor of the next larger size. Many nodes will have two parents. We now have an extremely lopsided directed acyclic graph. The cost of a partition is the cost of a path from the root node to a leaf node (right edge), so we want to find the path of least cost. Most nodes actually have two costs, corresponding to symmetric and asysmmetric bells. Our cost thus depends on which path we take into a node, not just the node itself.

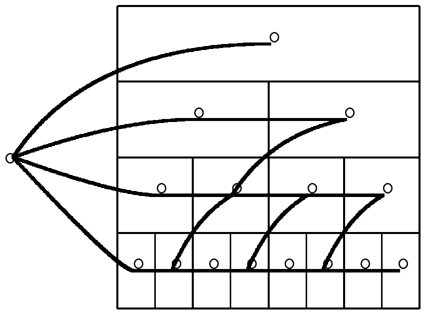

Figure 5.2: The search graph.

The generic decision step is as follows: We assume the current node's children contain the minimal cost of a path from them to the right. We compare the costs for the symmetric child of the same size and the asymmetric child of the next larger size, and *add* the smaller to the symmetric and asymmetric costs of the current node. Sometimes the larger child will not exist.

Applying this generic step in the correct order reduces us to the children of the root node, which we compare by hand. There is a geometric order we can use, which assures children are processed before their parent(s). To each box associate its center (See Figure 5.2). Process the boxes in order of the x coordinate of their center, starting at the right. It costs $2N\log_2^2 N$ to compute the coefficients and $4N$ to run the search.

Remark: This algorithm can be generalized to a search for the best dyadic partition with the restriction (fiat) that, given the partition, the bells are as large as possible. In general we could have about $\log N$ right edges and $\log N$ left edges. Each represents a neighbor of a different size. Each interval corresponds to at most $\log^2 N$ sets of coefficients, for the $\log N$ left edges and $\log N$ right edges. Each branching is by $\log N$ instead of two. The geometric ordering remains unchanged. To compute the coefficients would cost about $N \log N \cdot \log N \cdot \log^2 N = N \log^4 N$. The search would cost about $2N \cdot 2 \log^2 N = 4N \log^2 N$.

5.5 The Two Dimensional Search

For the two dimensional compression scheme in Section 4.4 we also require a search algorithm. The algorithm described there used only dyadic squares, but we may as well allow dyadic rectangles. Again, this search occurs at the precomputation stage and so is allowed to be slow, but we would rather it not be.

For each rectangle, we allow four types of compression: expanding in x and y, expanding in x only, expanding in y only, and not expanding at all. All expansions use the double sized bells and intervals as developed in Section 3.3.1. The double bells eliminate all problems of bell compatibitity. We allow the algorithm to choose not to expand because we get this variety for free, and because it assures we never do more work than the direct matrix application. In the search described below, the cost for a rectangle should be taken as the minimum cost of the four types of compression.

We still must deal with edge effects. As mentioned in Section 5.4, the $\phi = 0$ edge has a high order zero and the $\phi = \pi/2$ edge has parities so they are no problem. At the $n = m$ edge we extend the matrix as 0. At the $n = N/2$ edge we simply continue the matrix for $n > N/2$.

5.5.1 A Best Dyadic Rectangle Search

Nick Bennett [Ben97] has developed an algorithm for determining the best partition into dyadic rectangles. The search using squares only is given in [Wic94, p.299]. See also [TV96] for a similar algorithm. As above, we can determine 'best' using any reasonable cost function, such as entropy, $|| \cdot ||_p^p$, or the number of coefficients above a threshhold. For our application we use the number of coefficients above ϵ.

This result is not yet published, so we sketch is here. The search is based on the following observations: Given a rectangle, we should either keep it, or split it in the center in x or split it

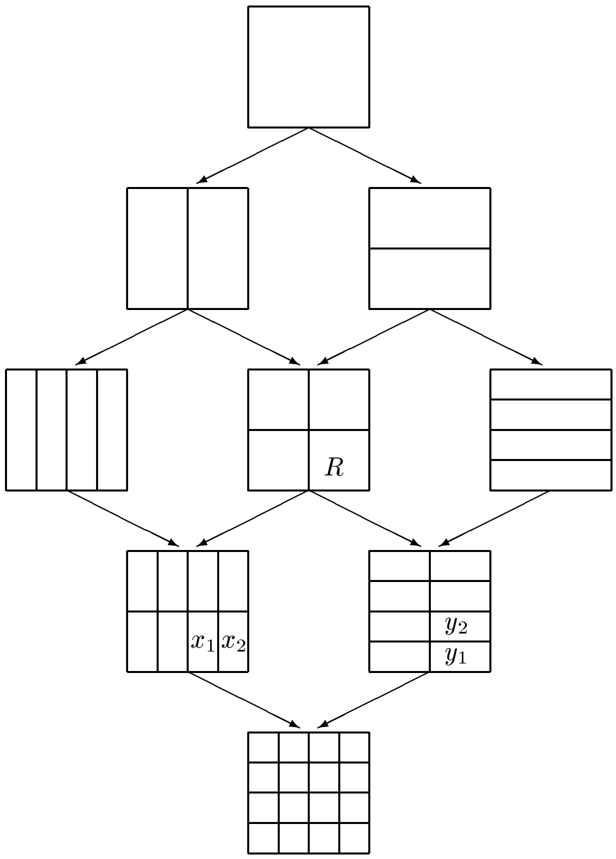

Figure 5.3: The dyadic rectangle search tree.

in the center in y. To know whether to split in (say) x, we need to know the minimal cost for all partitions which include this split in x. Thus we are faced with two problems identical to the original problem; one on the left half of the original rectangle and one on the right half. We compare the cost for keeping the rectangle with that of the two splits and keep the minimum.

If we do this process recursively by following all possible splits down from our rectangle, the number of comparisons grows exponentially. Since the operations of splitting in x and splitting in y commute, however, many of these are repeats and can be eliminated. We instead work our way up from the smallest rectangles to the largest and let the minimal cost percolate up to the top. This approach avoids any repeated comparisons. Each dyadic rectangle has two parents, one in x and one in y and thus contributes to two comparisons. Counting the number of dyadic rectangles thus counts the number of comparisons. There are two comparisons each but each comparison is counted twice. Thus the number of comparisons equals the number of dyadic rectangles, which is less than

$4N^2$.

The search tree is shown in Figure 5.3. The generic decision step is to compare a rectangle to its two sets of children, if they exist. For example, we compare the cost R to $x_1 + x_2$ and $y_1 + y_2$ and take the smallest. We need to be sure that the decisions for the children of the current rectangle have already been made. One way to assure this is to work our way from bottom to top. Another way is to loop through the scales in x from finest to coarsest, and within this loop through the scales in y from finest to coarsest. This approach highlights the separable nature of the search.

Remarks:

1. When N becomes very large we will not be able to hold a matrix of size $N \times N$ in active memory. In this case we cannot do the search without either swapping memory to disc, or recomputing the matrix many times. Both these problems can be avoided if we restrict our search to rectangles of a size which can be held in memory. Since very large rectangles are unlikely to be chosen anyway, this should not affect performance. The search algorithm is the same, but on rectangles which are too large we do not try to compute coefficients, but simply mark the rectangle as an illegal choice. Even for smaller values of N, we use the technique in [Wic94, p.313]: discard coefficients once you have determined their cost, and then recompute those coefficients needed for the chosen partition.

2. Numerical results (Section 6.1.3) show that the use of double size bells hurts us a lot. The search described here is essentially a standard dyadic search 'cross' another standard search. We use the double bells to assure our decisions are independent. What we really want is a non–decreasing dyadic partition search (Section 5.4.1) cross another. The dependencies become too entangled, however, and we have been unable to produce a search which assures bell compatibility. It is possible, however, to take a non–decreasing search and cross it with a standard search with double bells. This would assure that our two dimensional search never loses to the one dimensional. The partition we end up with will have the property: as we move to the right (x increases) the boxes we encounter will have non–decreasing size in x, and non–increasing size in y.

Chapter 6

Numerical Results

This chapter contains the results of numerical experiments on the one and two dimensional algorithms from Chapter 3. Our implementation was for diagnostic purposes and is not meant to represent the optimal performance of these algorithms, particularly when evaluating run times. We seek to answer the following questions:

1. Does the cost of expansion (number of coefficients retained) agree with the prediction in Chapter 4?
   (a) As a function of N?
   (b) As a function of ϵ?
2. When are our algorithms better than the direct method, specifically:
   (a) At what N do they become faster?
   (b) Are there other considerations we need to take into account?
3. Are our algorithms stable, and how large are the errors?

To summarize the results:

1. The one dimensional algorithm performs better than predicted, as a function of both N and ϵ. The two dimensional algorithm performs well in ϵ, but poorly in N.
2. The one dimensional algorithm becomes faster than the direct method at $N = 256$. It's true adaptive nature becomes clear by $N = 2048$ when it is approximately 3.3 times faster. There are several implementation–dependent and purpose–dependent issues which arise when attempting the comparisons.

3. The one dimensional algorithm is stable and has errors about size ϵ.

6.1 Coefficient Counts

This section deals with the number of coefficients retained after trying various compression schemes. This number is called the cost of the expansion in Chapter 4.

6.1.1 One Dimensional

First we consider the coefficents counts obtained from the one dimensional compression scheme from Section 4.3. These coefficients would be used in the one dimensional algorithm in Section 3.2. Our algorithm to obtain the counts is as follows:

1. Fix some ϵ to serve as a cutoff, and the number of points N. ϵ determines which bell to use, as in Section 5.3.

2. Loop through $0 \leq m < N/2$ and $m \leq n < N/2$. For each value:

   (a) Construct $\psi_n^m(\phi) = \sqrt{\sin\phi}\tilde{P}_n^m(\cos\phi)$.

   (b) Do a best non–decreasing dyadic partition search as in Section 5.4.1 in ϕ on $[0, \pi/2]$. This search gives us a cost for each (m, n) which we add to a running total.

If we did no compression, we would be adding over $N/2$ points, $0 \leq m < N/2$, and $m \leq n < N/2$ values of n, for total cost $(N/2)(N/2)(N/2+1)/2 = N^3/16 + N^2/8$. We can divide the number we did keep by this, to obtain the percentage kept. Our prediction in Section 4.3.3 is that the number we keep for each Associated Legendre function should be $\mathcal{O}(\sqrt{N/2})$. To test this, we divide the number kept by $\sqrt{N/2}(N/2)(N/2+1)/2 = \sqrt{N/2}(N^2/8 + N/4)$ to obtain the 'constant' C_N. We want C_N to be independent of N or at worst grow logarithmicly. It is also nice if C_N is small. The results for this method are contained in Table 6.1. In this and the following tables, a * indicates an estimate obtained by sampling the allowed values of m and n.

As desired, C_N is not large. It slowly increases initially, but then starts to decrease. By comparing the C_N obtained at different ϵ in Table 6.1, we can tells how the constant depends on ϵ. In Section 4.3 the dependence was like $(\log\epsilon)^{7/2}$. The numerical results indicate that at worst the dependence is like $\log\epsilon$. Recall that as ϵ changes, so does the bell. It is primarily due to these bells that the constant behaves so well.

ϵ	N	% kept	C_N	ϵ	N	% kept	C_N
10^{-12}	128	66.32	5.31	10^{-6}	128	45.10	3.61
10^{-12}	256	46.52	5.26	10^{-6}	256	33.48	3.79
10^{-12}	512	34.75	5.56	10^{-6}	512	24.34	3.89
10^{-12}	1024	25.87	5.85	10^{-6}	1024	17.07	3.86
10^{-12}	2048	18.44	5.90	10^{-6}	2048	11.56	3.70
10^{-12}	4096	12.72	5.76	10^{-6}	4096	7.70	3.48
10^{-12}	8192	*8.61	*5.51	10^{-6}	8192	*5.16	*3.31
10^{-12}	16384	*5.87	*5.32	10^{-6}	16384	*3.49	*3.16
10^{-12}	32678	*3.97	*5.08	10^{-6}	32678	*2.36	*3.02

Table 6.1: Coefficient counts for compression in ϕ only. * indicates an estimate by sampling.

Qualitative Analysis

Although the overall counts may be the most important, it will improve our understanding if we look at the behavior of the compression algorithm in more detail. This closer look gives us an idea of when the asymptotic behavior will appear. It also may be useful when trying to decide whether similar techniques will work for other problems.

We first look at the counts separately for each order. The scheme is the same but now we plot over m instead of summing over m. We also rescale the plots for different numbers of points so that we can judge relative performance. We plot this in Figure 6.1 for $\epsilon = 10^{-6}$. Note that as $m/(N/2)$ increases, the potential number of coefficients decreases, and so the slanted lines represent fixed percentages.

We can observe several things from this plot. The $m = 0$ case is the worst case, but is shrinking at the fastest rate. This effect is probably due to a derivative discontinuity at $\phi = 0$. $\psi_n^m(\phi)$ has a zero of order $m + 1/2$ at $\phi = 0$. For large m this may as well be smooth, but for small m the Fourier expansion picks this up and the interval which touches this edge tends to have no compression. This interval must then be kept small. As n increases the interval sizes should increase (we approach constant frequency) so we are forced into having $\log N$ intervals, each with a constant (bell) cost.

Once m is a little larger, we do very well, as we observe from the local minimum in the $N = 1024$ plot. As m gets even larger, the behavior of the instantaneous frequency becomes important and we see the predicted behavior. For even larger m, the $(N - m)$ factor dominates and we have decreased costs.

It is also enlightening to look at the compressed matrix for some fixed m and N. We represent this graphically with the following scheme: Each row is a $\psi_n^m(\phi)$ and we compress it with Local

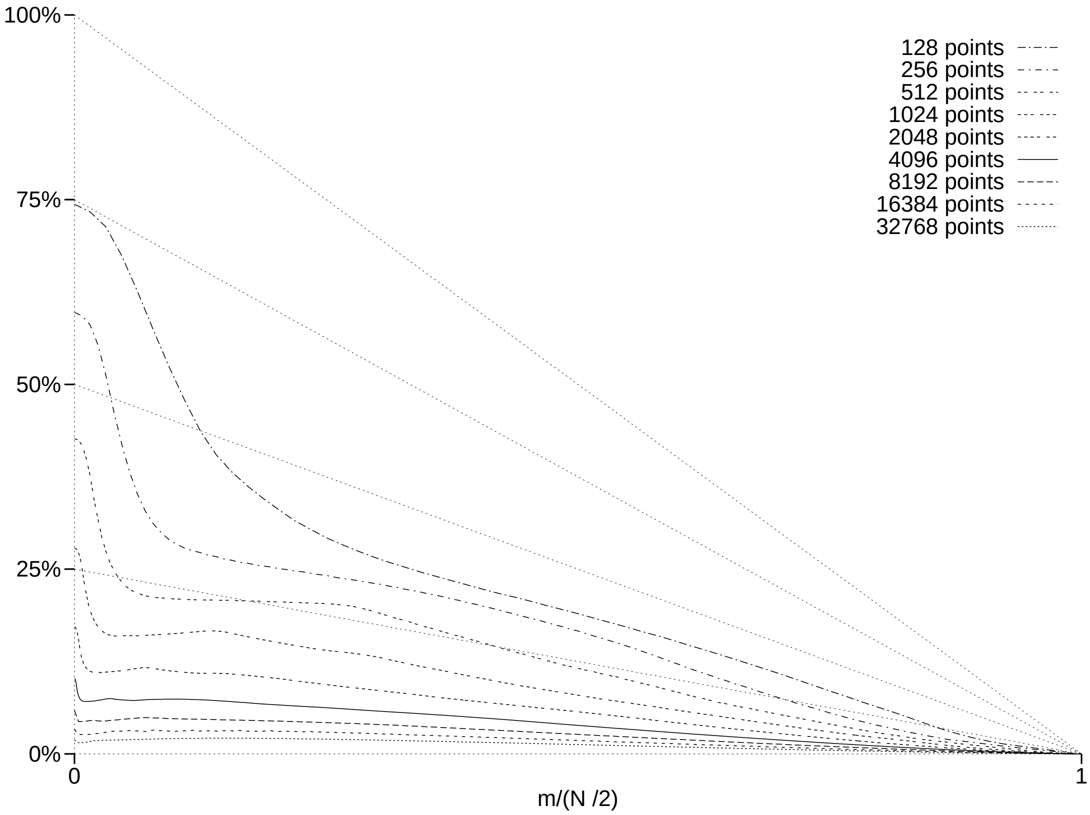

Figure 6.1: One dimensional counts by order at $\epsilon = 10^{-6}$.

Cosine. Since for each subinterval the Local Cosine basis has as many frequencies as we have points, we can store the coefficients obtained in that subinterval. Thus in what was the space (ϕ) direction, we now have both space (since each subinterval stays put) and frequency (within a subinterval) information. (See Figure 3.1.) The partitioning is not explicitly given in our representation, but must be inferred from the ridges. We color pixels to represent the size of the coefficients, with black meaning above 10^{-6} and gray between that and 10^{-12}. Figure 6.2 shows an example of such a representation for small m and Figure 6.3 shows one for large m.

These examples illustrate several common features we encounter. We first discuss Figure 6.2. The very top of the matrix is empty, since we do not allow $n < m$. The next layer (small n) is expanded in a global Fourier series because no matter what their other properties are, they only contain low frequencies. Our main analysis in Chapter 4 was based on the 'high frequency' bound (4.13). We also have the bound (4.12) which is uniform in frequency and will give better results at low frequencies. This second bound seems to be controlling the behavior.

As we move down, n increases and our global representation becomes less efficient. About 1/4 down, it becomes better to split in two. One of these is an efficient 'asymptotic' region, and the other is an inefficient but low frequency region containing the turning point. This behavior then repeats, with the low frequency region becoming less efficient until it is ready to spawn another asymptotic interval. For any fixed m this behavior will eventually dominate and at the highest frequency we will have a logarithmic type partition.

Before we get to this asymptotic regime, several other things may happen. In this example near the bottom there seems to be a debate between using one large and two small intervals versus four small intervals. This debate is probably because we have an inherent bias against switching sizes: our asymmetric bell has a higher cost than the symmetric bell. (See Section 5.4.1.)

Figure 6.3 illustrates some features which appear for larger m. The top of the matrix is again blank. Then we encounter a small region of functions which are both low frequency and localized to the right (near $\pi/2$). As n increases, this expansion based on the low frequency bound (4.12) should become less efficient, causing us to split. Instead we see that the turning point, about which the inefficient interval is based, moves to the left and creates a new inefficient interval. We again see a process by which efficent stripes of coefficients are created. If we continue this picture for even larger n, the turning point would become very close to the left edge, and we would see the behavior of Figure 6.2 repeated in the lower left corner. At the same time, the stripes in the lower right corner would begin to recombine, eventually forming a logarithmic type partition, with the intervals doubling as we move to the right.

We see from these examples that several different phenomena will affect our compression as N increases. For each m, inefficient intervals grow and move as new n are allowed. At the same time, efficient intervals are created, and then combined to form asymptotic intervals. All the while, new m are allowed. Each effect has its own rate, and they are in competition. General statements about asymptotic behavior will not suffice to describe the behavior of compression algorithms like this one; rigorous analysis like that in Section 4.3 is required.

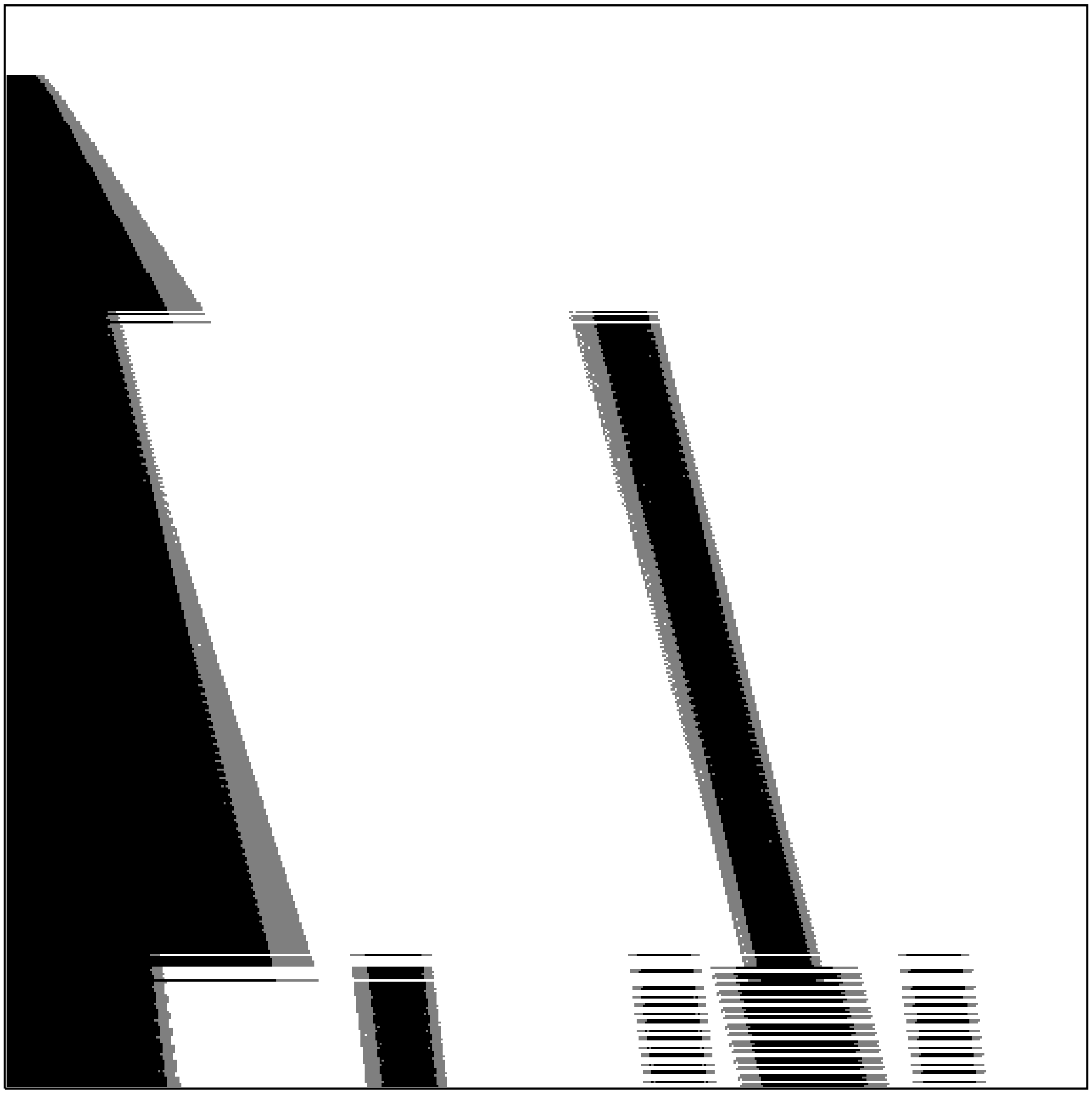

Figure 6.2: One dimensional compressed matrix for $m = 32$ and $N = 1024$.

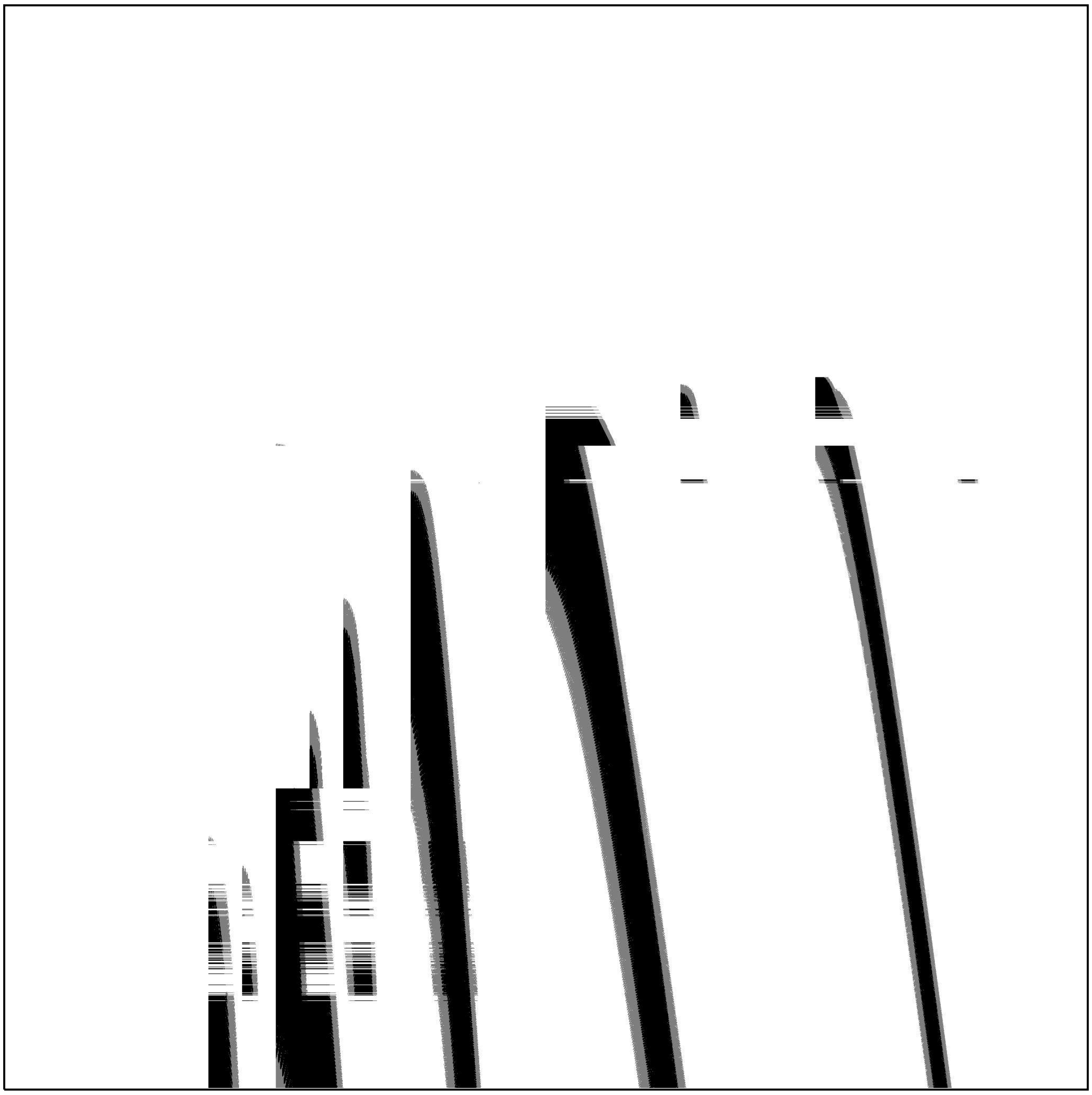

Figure 6.3: One dimensional compressed matrix for $m = 700$ and $N = 4096$.

6.1.2 One Dimensional with Double Bells

Before considering the two dimensional algorithm, we look at a variant of the one dimensional scheme. In order to do the two dimensional search, we do expansions on the double of the box we wanted. This doubling introduces a factor of two inefficiency in the cost of the bell. We would like to know how much this factor matters in the comparison of these two methods. To do this, we run the one dimensional compression scheme using the double sized bells and intervals, as developed in Section 3.3.1. Algorithmically this is simpler than the one dimensional orthogonal approach because we can now use a standard search ([Wic94, p.283]). Results are summarized in Table 6.2.

ϵ	N	% kept	C_N	ϵ	N	% kept	C_N
10^{-12}	128	69.20	5.54	10^{-6}	128	50.26	4.02
10^{-12}	256	52.13	5.90	10^{-6}	256	38.88	4.40
10^{-12}	512	40.38	6.46	10^{-6}	512	31.76	5.08
10^{-12}	1024	32.73	7.41	10^{-6}	1024	27.53	6.23
10^{-12}	2048	28.48	9.11	10^{-6}	2048	20.20	6.46
10^{-12}	4096	*22.36	*10.12	10^{-6}	4096	*13.89	*6.29
10^{-12}	8192	*15.68	*10.03	10^{-6}	8192	*9.42	*6.03
10^{-12}	16384	*10.77	*9.75	10^{-6}	16384	*6.42	*5.81
10^{-12}	32678	*7.42	*9.50	10^{-6}	32678	*4.45	*5.69

Table 6.2: One dimensional coefficient counts with double sized bells. * indicates an estimate by sampling.

At $N = 256$, C_N is only slightly worse than in the orthogonal case. As N increases, however, C_N grows at a faster rate than before. When it approaches twice as large as before, however, its rate slows. For $\epsilon = 10^{-6}$ by $N = 4096$ it has reached almost twice as large, and begins to mimic the previous behavior and decrease. For $\epsilon = 10^{-12}$ this doesn't occur until $N = 8192$.

The conclusions we may draw from this are as follows:

1. The use of double sized intervals will double the number of coefficients we need asymptotically.
2. For small N the performance is better than this. The number of coefficients thus appear to be growing at a faster rate.

Qualitative Analysis

We can also look at the number of coefficients kept for each order. We use the same setup as in Figure 6.1 and plot the results in Figure 6.4 First we observe that the bad behavior for small m is repeated. Second we note that initially as we increase N the percentages for typical m are not

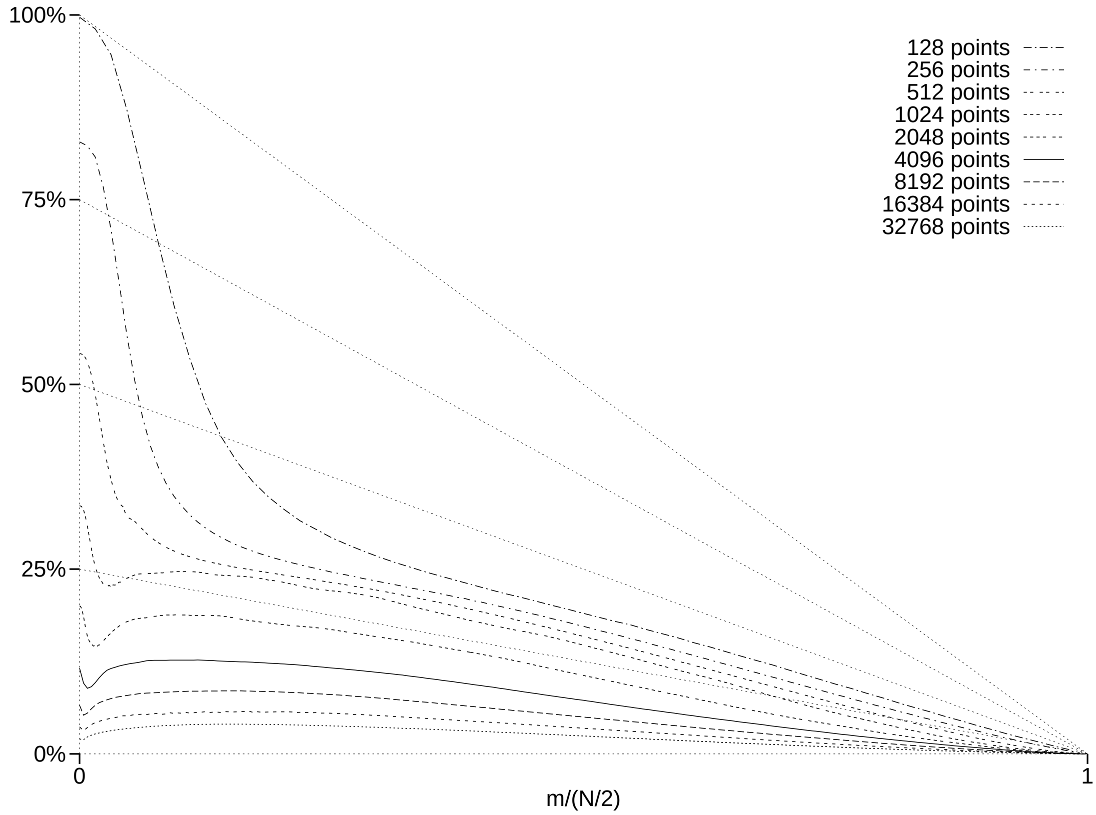

Figure 6.4: One dimensional counts with double bells, by order, with $\epsilon = 10^{-6}$.

decreasing significantly. Once we pass $N = 1024$ the rate improves. This behavior matches the behavior of C_N.

Qualitatively, the picture of our matrices compressed with double bells is the same as with normal bells (Figure 6.2) with the following exceptions:

1. It has a stronger preference for a global expansion. When we expand on the entire interval, we don't use any bells. As n increases, this becomes less efficient and we must split. With the extra cost of double bells, we wait longer to split.

2. The bands of coefficients are twice as thick.

3. There is no bias toward equal sized intervals. In our usual one dimensional compression search, switching to a larger interval size as we move to the right requires an asymmetric bell, with larger cost.

6.1.3 Two Dimensional

Next, in Table 6.3, we consider the two dimensional compression scheme, as in Section 4.4. Our algorithm is now:

1. Fix some ϵ to serve as a cutoff, and the number of points N. ϵ determines which bell to use, as in Section 5.3.

2. Loop through $0 \leq m < N/2$. For each value:

   (a) Construct $\psi_n^m(\phi)$ for all $m \leq n < N/2$.

   (b) Do the two dimensional search described in Section 5.5.1 to our matrix as a function of ϕ and n on $[0, \pi/2] \times [0, N/2]$ as an image, with some portions of this image left blank. For each box chosen, we allow four types of expansions: expanding in ϕ and n as described before, expanding in ϕ only, expanding in n only, and not expanding at all. All expansions use the double sized bells and intervals as developed in Section 3.3.1. This gives us a cost for each m which we add to a running total.

We compute C_N as above, by dividing by $\sqrt{N/2}(N^2/8 + N/4)$. This is *not* the rate predicted for this algorithm, but gives us a good way to compare it with the one dimensional algorithm. For the two dimensional algorithm to work as predicted, we want C_N to decrease at a rate like $1/\sqrt{N}$ or perhaps $\log N/\sqrt{N}$.

ϵ	N	% kept	C_N	ϵ	N	% kept	C_N
10^{-12}	128	68.30	5.46	10^{-6}	128	47.35	3.79
10^{-12}	256	49.92	5.65	10^{-6}	256	36.68	4.15
10^{-12}	512	38.51	6.16	10^{-6}	512	30.48	4.88
10^{-12}	1024	31.41	7.11	10^{-6}	1024	25.00	5.66
10^{-12}	2048	*26.45	*8.47	10^{-6}	2048	*17.12	*5.48
10^{-12}	4096	*19.29	*8.73	10^{-6}	4096	*10.46	*4.74

Table 6.3: Coefficient counts from compressing in ϕ and n. * indicates an estimate by sampling.

The two dimensional algorithm is not performing very well in this range of N. C_N only starts to decrease at $N = 2048$. The results of the one dimensional compression with double bells suggests that this may be an artifact of the double bells, and so may disappear for larger N. We ceased numerical testing at this point because:

1. We reached the largest matrix we could hold in memory. This problem can be eliminated by compressing smaller portions at a time, as mentioned in the first remark in Section 5.5.1. This new approach requires a new implementation, which was underway when we noticed:

2. The search we used is not the best way to do our two dimensional compression. We can eliminate the double bells in one dimension, and assure the two dimensional compression beats the one dimensional. See the second remark in Section 5.5.1.

3. It became clear from timing results that the additional overhead of the two dimensional algorithm would not be worth the additional compression on the numbers of points we were prepared to test.

Qualitative Analysis

We can again look at the counts as a function of m. We should also do a comparison with the one dimension scheme with double bells in Figure 6.4. In Figure 6.5 the lines are from the two dimensional scheme, and we include as a line thickened by diamonds the one dimensional scheme with double bells for $N = 2048$.

We first note from this plot that for large m the two methods coincide. The two dimensional scheme has chosen to only compress in one dimension. For smaller m we do have two dimensional compression and so see improvement. The gain from doing the full two dimensional scheme is small, just starting to become significant at $N = 2048$.

We can look at the compressed matrix for some fixed m and N to further explore this behavior. The setup is similar to that in Figure 6.2, with black meaning above 10^{-6} and gray between that and 10^{-12}. The main difference is that we potentially have twice as many coefficients as function values, so each box is blown up by a factor of two. Thus a box which appears 1/4 full has actually achieved no compression. We have allowed the algorithm to choose not to compress any particular box in any particular direction, so we never do worse than we started. When we don't try to compress in a direction, we don't need this factor of two blow up, so we leave this space blank. This extra space leads to a disjointed appearance of the matrix, with large empty rectangles. We also note that the even and odd parity functions must be done separately, making this matrix rectangular. (See Figure 6.6.)

For this sample matrix, the algorithm has chosen to use a global, ϕ–only expansion on the top

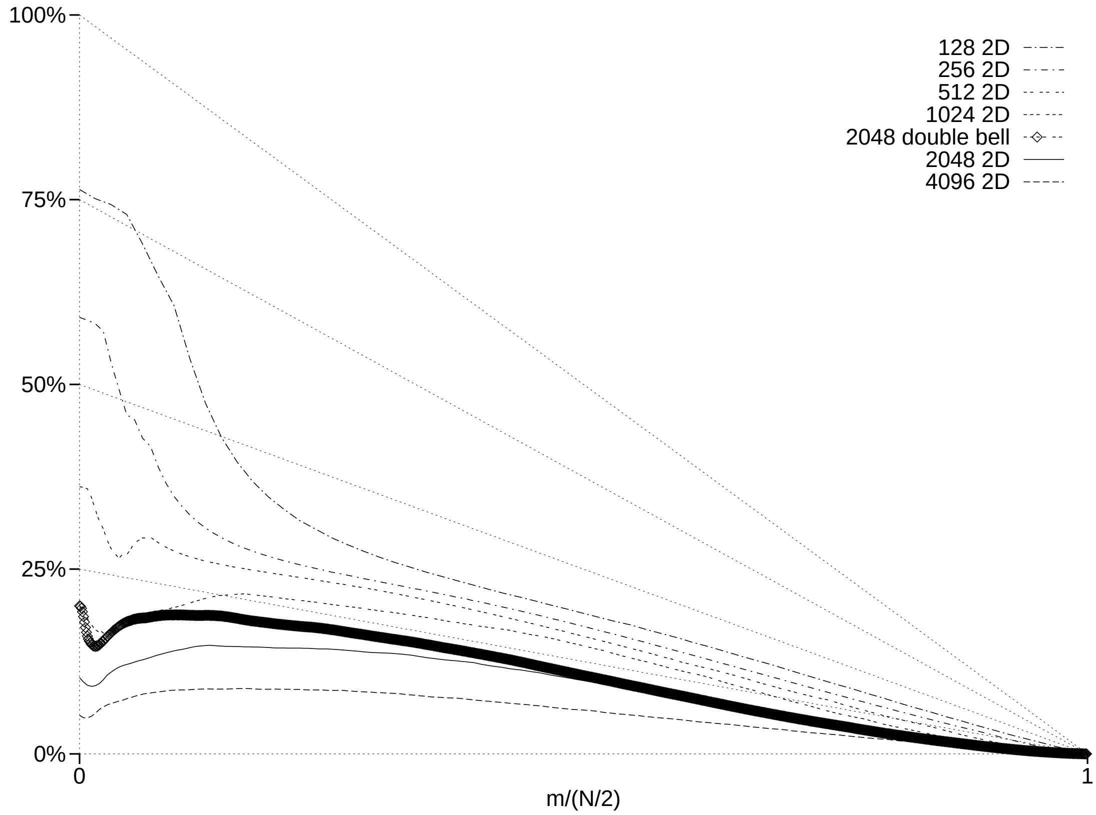

Figure 6.5: Two dimensional counts by order at $\epsilon = 10^{-6}$.

3/4. In the bottom right, It has chosen ϕ–only expansion with partitioning. In the lower left corner it chose a n–only expansion. The only compression improvement we have versus the one dimensional algorithm with double bells is in this region of small ϕ and large n. This analysis gives two explanations for the poor relative performance of the two dimensional algorithm:

1. There is not enough room. Since we do the evens and odds separately, we only have $N/4$ values in $n = y$. With the inefficiencies of the double bell, we can't gain much on so few points.

2. The one dimensional algorithm is performing better than predicted. On regions where the two dimensional method should work for large squares, the function resembles $\cos(xy)$. On this region one dimensional compression should be even better. The one dimensional yields coefficients on a diagonal band, whereas two dimensional must subdivide this to get several clouds. One can also look back at Figures 6.2 and 6.3 and ask where they would benefit from additional compression in the n direction. Figure 6.2 might benefit in the lower left corner, but Figure 6.3 would not seem to benefit at all.

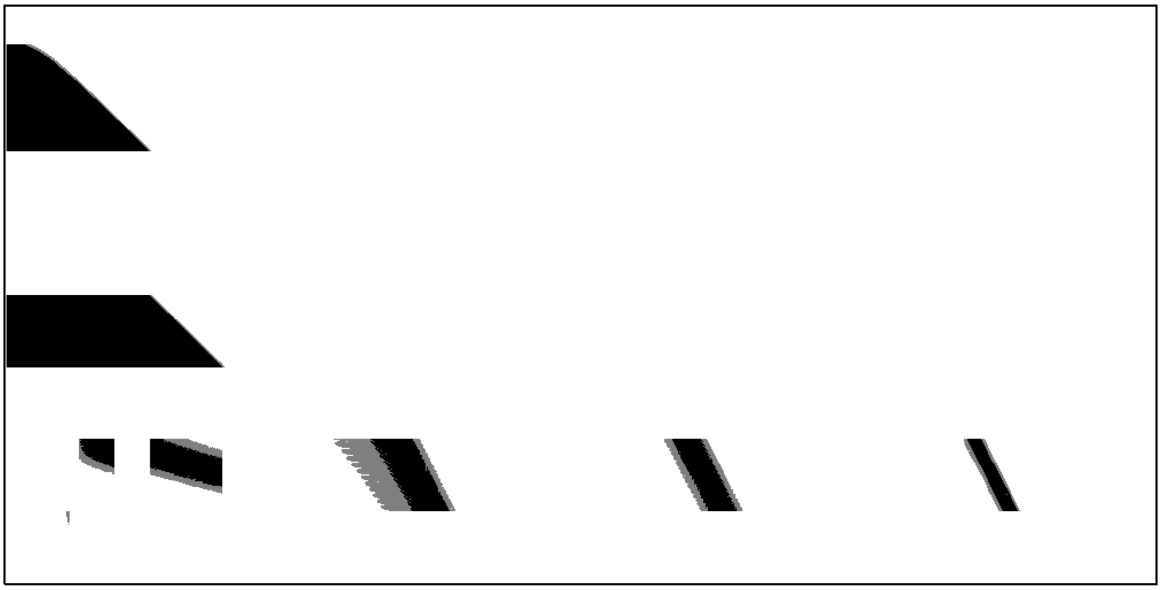

Figure 6.6: Two dimensional compressed matrix for $m = 64$ and $N = 2048$ using only even values of $n - m$.

6.2 Program Timings

The coefficient counts in the previous section are essentially implementation–invarient. When trying to compare the speed of these algorithms to the direct method, we find many things are very much implementation–dependent and purpose–dependent. The purpose of our timing results is not to give estimates of the absolute run times, but rather to give comparisons with the direct method(s), and indicate in which situations our algorithm will be most useful. The system–dependent and purpose–dependent issues which arise are highlighted as •. We emphasize that our implementation was primarily for diagnostics and is not claimed to be optimal.

All programs were implemented in ANSI C and run on a Sun SparcStation 20 SX with a uniprocessor SuperSPARC 32 bit RISC cpu running at 50Mhz. The timing mechanism is based on internal system queries as to how much CPU time has been used. It is neither very accurate nor very precise, but was the best available. The times given are the average over multiple runs in an attempt to give two significant digits. At small N even this should be considered suspect.

6.2.1 Baseline

Before we compare methods, we give some baseline results to indicate the general speed of our machine. In Table 6.4 we give the times needed for the FFT and some derived routines, as follows:

Fast Fourier transform (FFT) contains the time for an FFT on a real function on N points.

Local Cosine Transform (LCT) contains the times for a Local Cosine transform on N points.

LCT with bell includes multiplying by a bell (on $2N$ points) and folding. The values of the bell were precomputed.

All Dyadics has times for a complete dyadic decomposition into Local Cosine, with bells. Expanding on very small intervals is not worthwhile, however, so we only use intervals with at least 16 points. This minimal window size is the same as is used in the compression programs, but is not optimal.

- These results are from a program compiled with `gcc` with `-O` optimization. The same program compiled with `acc -O` was approximately 100 times slower for these FFT's, yet about the same speed for other computations. The FFT is common enough that very good optimizations exist

N	FFT	LCT	LCT with bell	All Dyadics
64	$1.4 \cdot 10^{-4}$	$3.8 \cdot 10^{-4}$	$4.1 \cdot 10^{-4}$	$5.7 \cdot 10^{-4}$
128	$2.9 \cdot 10^{-4}$	$7.5 \cdot 10^{-4}$	$8.2 \cdot 10^{-4}$	$1.9 \cdot 10^{-3}$
256	$5.8 \cdot 10^{-4}$	$1.5 \cdot 10^{-3}$	$1.6 \cdot 10^{-3}$	$5.1 \cdot 10^{-3}$
512	$1.2 \cdot 10^{-3}$	$3.0 \cdot 10^{-3}$	$3.2 \cdot 10^{-3}$	$1.4 \cdot 10^{-2}$
1024	$2.5 \cdot 10^{-3}$	$6.4 \cdot 10^{-3}$	$6.9 \cdot 10^{-3}$	$3.5 \cdot 10^{-2}$
2048	$5.3 \cdot 10^{-3}$	$1.2 \cdot 10^{-2}$	$1.5 \cdot 10^{-2}$	$8.1 \cdot 10^{-2}$
4096	$2.1 \cdot 10^{-2}$	$3.8 \cdot 10^{-2}$	$4.1 \cdot 10^{-2}$	$2.0 \cdot 10^{-1}$
8192	$5.0 \cdot 10^{-2}$	$8.1 \cdot 10^{-2}$	$9.0 \cdot 10^{-2}$	$5.0 \cdot 10^{-1}$
16384	$1.0 \cdot 10^{-1}$	$1.6 \cdot 10^{-1}$	$1.8 \cdot 10^{-1}$	$1.1 \cdot 10^{0}$

Table 6.4: Timing benchmarks.

for it. The availability of such an optimization greatly reduces the cost of FFT's, and can make our algorithm more competitive.

6.2.2 Direct Methods

Here we do timings for the direct methods. 'Methods' is in the plural because it is not clear whether things like constructing matrix values or reading them from memory should be included in the time. We deal with the collection of reduced evaluation problems using parities, as stated in Section 2.4.4. The direct methods and our algorithm have several steps in common, which we do not include in the times. In particular, we don't include:

1. The FFT in θ.
2. Negative values of m. The basis functions are the same as for positive m so this simply doubles all times.
3. Overhead. One time memory setup, initialization of bells, etc.

Direct method times are found in Table 6.5. The various components are:

Construct (C) computes the matrix values.

Write (W) writes the matrix values to disc.

Read (R) reads the matrix values from disc.

Apply (A) applies the matrix.

The prediction is that these times are $\mathcal{O}(N^3)$ and so should increase by a factor of 8 when N is doubled. The behavior we see is consistent with this, although not very precise.

N	R+A	A	C+A	C+W
64	$1.2 \cdot 10^{-2}$	$3.4 \cdot 10^{-3}$	$1.3 \cdot 10^{-1}$	$1.4 \cdot 10^{-1}$
128	$9.5 \cdot 10^{-2}$	$2.7 \cdot 10^{-2}$	$1.0 \cdot 10^{0}$	$1.1 \cdot 10^{0}$
256	$6.9 \cdot 10^{-1}$	$2.1 \cdot 10^{-1}$	$8.2 \cdot 10^{0}$	$8.5 \cdot 10^{0}$
512	$6.6 \cdot 10^{0}$	$1.6 \cdot 10^{0}$	$6.4 \cdot 10^{1}$	$6.5 \cdot 10^{1}$
1024	$5.3 \cdot 10^{1}$	$1.3 \cdot 10^{1}$	$5.3 \cdot 10^{2}$	$5.4 \cdot 10^{2}$
2048	$5.3 \cdot 10^{2}$	$1.4 \cdot 10^{2}$	$4.9 \cdot 10^{3}$	$4.9 \cdot 10^{3}$

Table 6.5: Direct methods times.

- If we only wish to do one Spherical Harmonic transform, our best method is compute and apply (C+A). Writing to disc is not worthwhile, nor is the compression in our algorithms. To use our algorithms one must (pre)compute the compressed version of the Associated Legendre function matrices. For the one dimensional algorithm this takes $\mathcal{O}(N^3 \log^2 N)$ time. and for the two dimensional algorithm $\mathcal{O}(N^3 \log^4 N)$ time.

- If we do many transforms, we would like to do just the apply (A) step. For this we need to hold something of size N^3 in memory, impossible for large N. We could also try loading in the coefficients for one m, doing all the Associated Legendre transforms for that m, and then incrementing m.

6.2.3 The One Dimensional Algorithm

For the values of N we have tested, the one dimensional algorithm is performing best in coefficient counts, so we use it for our performance testing. Again we consider the collection of reduced evaluation problems using parities, as stated in Section 2.4.4. We use $\epsilon = 10^{-6}$ as our cutoff. One dimensional times are found in Table 6.6. The component operations are:

Consruct and compress (CC) computes the matrix values and their compressesed coefficients.

Write compressed (WC) writes the compressed coefficients to disc.

Read compressed (RC) reads the compressed coefficients from disc.

Apply compressed (AC) applies the matrix in compressed form.

FFT does the dyadic Local Cosine transforms to produce the final outputs.

We have noted in several places that each of our intervals must contain at least one data point, thus restricting us to $\log_2 N$ levels. If the number of points in our interval is less than the minimal

N	R+AC+FFT	R+AC	R	CC+W	Minwin
64	$6.0 \cdot 10^{-2}$	$1.4 \cdot 10^{-2}$	$1.1 \cdot 10^{-2}$	$0.55 \cdot 10^{-1}$	8
128	$2.5 \cdot 10^{-1}$	$8.0 \cdot 10^{-2}$	$5.4 \cdot 10^{-2}$	$4.0 \cdot 10^{0}$	16
256	$1.4 \cdot 10^{0}$	$4.7 \cdot 10^{-1}$	$3.0 \cdot 10^{-1}$	$4.3 \cdot 10^{1}$	16
256	$1.1 \cdot 10^{0}$	$4.6 \cdot 10^{-1}$	$2.9 \cdot 10^{-1}$	$3.1 \cdot 10^{1}$	32
256	$7.6 \cdot 10^{-1}$	$4.7 \cdot 10^{-1}$	$3.0 \cdot 10^{-1}$	$2.1 \cdot 10^{1}$	64
256	$6.3 \cdot 10^{-1}$	$4.8 \cdot 10^{-1}$	$2.9 \cdot 10^{-1}$	$16 \cdot 10^{1}$	128
512	$8.0 \cdot 10^{0}$	$2.5 \cdot 10^{0}$	$1.6 \cdot 10^{0}$	$4.5 \cdot 10^{2}$	16
512	$6.5 \cdot 10^{0}$	$2.6 \cdot 10^{0}$	$1.6 \cdot 10^{0}$	$3.6 \cdot 10^{2}$	32
512	$5.2 \cdot 10^{0}$	$2.8 \cdot 10^{0}$	$1.6 \cdot 10^{0}$	$2.6 \cdot 10^{2}$	64
512	$4.5 \cdot 10^{0}$	$2.9 \cdot 10^{0}$	$1.8 \cdot 10^{0}$	$1.9 \cdot 10^{2}$	128
512	$4.4 \cdot 10^{0}$	$3.8 \cdot 10^{0}$	$2.2 \cdot 10^{0}$	$1.3 \cdot 10^{2}$	256
1024	$4.6 \cdot 10^{1}$	$1.8 \cdot 10^{1}$	$1.2 \cdot 10^{1}$	$4.9 \cdot 10^{3}$	16
1024	$4.2 \cdot 10^{1}$	$2.0 \cdot 10^{1}$	$1.3 \cdot 10^{1}$	$4.6 \cdot 10^{3}$	32
1024	$3.7 \cdot 10^{1}$	$2.1 \cdot 10^{1}$	$1.4 \cdot 10^{1}$	$3.4 \cdot 10^{3}$	64
1024	$3.5 \cdot 10^{1}$	$2.5 \cdot 10^{1}$	$1.5 \cdot 10^{1}$	$2.5 \cdot 10^{3}$	128
1024	$3.8 \cdot 10^{1}$	$3.3 \cdot 10^{1}$	$1.8 \cdot 10^{1}$	$1.8 \cdot 10^{3}$	256
1024	$6.1 \cdot 10^{1}$	$5.6 \cdot 10^{1}$	$3.0 \cdot 10^{1}$	$1.3 \cdot 10^{3}$	512
2048	$2.3 \cdot 10^{2}$	$8.2 \cdot 10^{1}$	$4.0 \cdot 10^{1}$	$4.9 \cdot 10^{4}$	16
2048	$2.0 \cdot 10^{2}$	$8.2 \cdot 10^{1}$	$4.5 \cdot 10^{1}$	$4.1 \cdot 10^{4}$	32
2048	$1.8 \cdot 10^{2}$	$8.4 \cdot 10^{1}$	$4.5 \cdot 10^{1}$	$3.4 \cdot 10^{4}$	64
2048	$1.6 \cdot 10^{2}$	$8.9 \cdot 10^{1}$	$4.9 \cdot 10^{1}$	$2.6 \cdot 10^{4}$	128
2048	$2.0 \cdot 10^{2}$	$1.6 \cdot 10^{2}$	$8.3 \cdot 10^{1}$	$2.1 \cdot 10^{4}$	256

Table 6.6: One dimensional times.

bell cost B, then our expansion will be dense. When this is the case, we may as well use the next larger interval. Thus we should really use $\log_2(N/B)$ levels, and so have the smallest interval with B points. We have some flexibility here: Increasing the minimal interval size may make our compression less efficient, but will reduce the number of FFT's we need to perform.

Column 6 (Minwin) of Table 6.6 gives the number of points in the smallest sized interval allowed. At $N = 256$ a good choice allows us to beat the direct method, while a bad choice makes us take over twice as long. For $N = 256$ and $N = 512$ our best choice is to allow only global expansions. We thus win only because of the fact that the FFT of an Associated Legendre matrix will be approximately one fourth full. (see Figure 6.2.) While it is true we have improved run–times, we are not adapting and so not using the force of our theories.

For $N = 1024$ and $N = 2048$ there is an optimal minimal interval size which is not simply the whole interval. Here we have started to adapt, and truly use our algorithm. Note that at $N = 2048$, our algorithm (RC+AC+FFT) takes 160 seconds, as compared to the direct (R+A) time of 530 seconds.

6.2.4 Comparison by Order

The Spherical Harmonic transform is easily parallelized: each m gives a separate problem. There is no particular reason to use the same algorithm for different m. The direct method takes $N(N-m)$ time for the m^{th} Associated Legendre transform, whereas our best takes $CN\log^2 N$ time. If $(N-m) \lesssim C\log^2 N$ then the direct method is better. Asymptotically this is unimportant, but it means our implementation should switch to the direct method for m near N. Similarly, for m near 0, the methods in [AR91] and [DH94] are probably better/faster. We also do not need to use the same settings for parameters like the minimal number of points in an interval. As we saw in Table 6.6, this can make quite a difference. Figure 6.7 illustrates that one should switch methods depending on m. It also shows how the asymptotically unimportant FFT step can be important in practice.

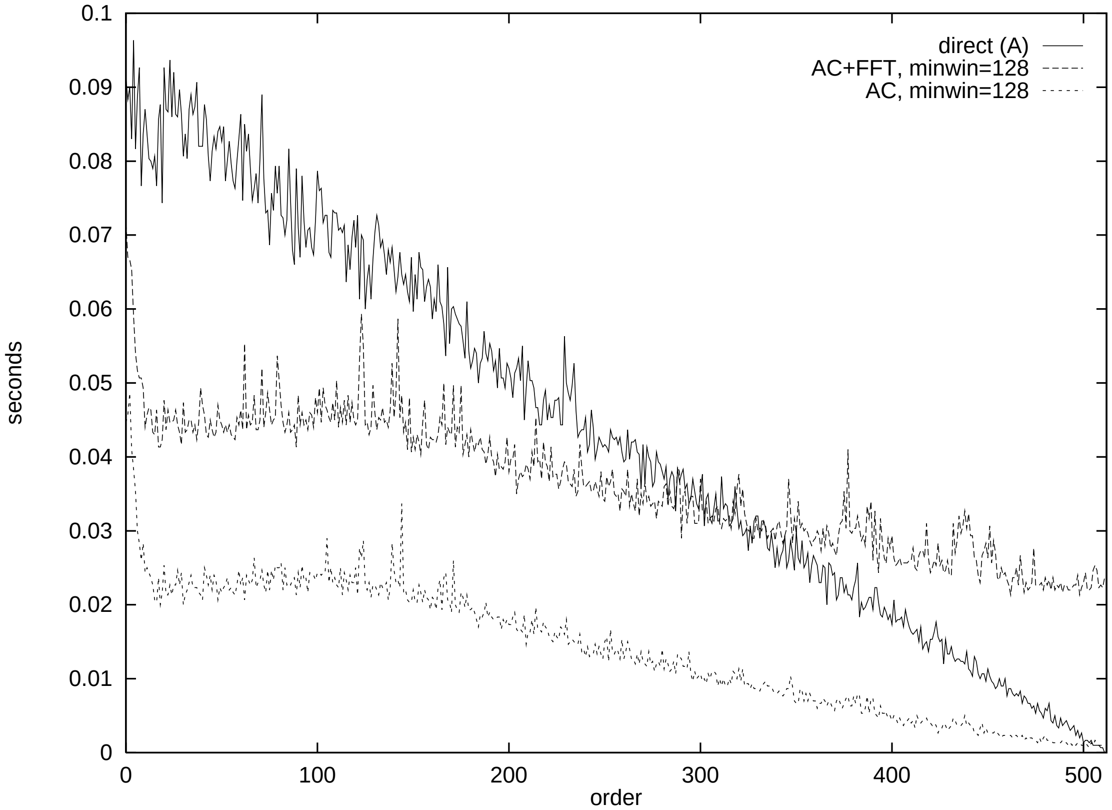

Figure 6.7: Performance dependence on order for $N = 1024$.

6.3 Stability and Errors

Our algorithm consists of a sequence of orthogonal steps, so there is really no room for instability. The only thing which is not orthogonal is the use of the bi–orthogonal optimized bells (Section 5.3), but these have condition number $\sqrt{2}$, and so won't be any problem.

The other issue is error size. In our algorithm we set a cutoff value of ϵ. This should be closely related to the desired accuracy, but perhaps not equal to it. We test our algorithm against the direct method running in double precision (16 digits) for the evaluation problem. Again, common steps such as the FFT in θ should not be included. In fact, there is no reason to do all orders at the same time. It will suffice to do spot checks for some sample orders. We do this in Table 6.7 for $\epsilon = 10^{-6}$. Our procedure for measuring error is to construct a pseudo–random vector, normalize it, compute the two transforms and take the l^2 norm of the difference. We do this for several vectors and take the worst result, to two digits.

	m					
N	0	32	64	128	256	512
64	$3.5 \cdot 10^{-7}$					
128	$4.5 \cdot 10^{-7}$	$6.3 \cdot 10^{-7}$				
256	$6.6 \cdot 10^{-7}$	$1.3 \cdot 10^{-6}$	$1.8 \cdot 10^{-6}$			
512	$1.2 \cdot 10^{-6}$	$1.3 \cdot 10^{-6}$	$1.6 \cdot 10^{-6}$	$1.9 \cdot 10^{-6}$		
1024	$1.7 \cdot 10^{-6}$	$1.5 \cdot 10^{-6}$	$2.6 \cdot 10^{-6}$	$2.1 \cdot 10^{-6}$	$2.5 \cdot 10^{-6}$	
2048	$2.2 \cdot 10^{-6}$	$2.6 \cdot 10^{-6}$	$3.1 \cdot 10^{-6}$	$3.5 \cdot 10^{-6}$	$3.1 \cdot 10^{-6}$	$3.2 \cdot 10^{-6}$

Table 6.7: A sampling of errors for $\epsilon = 10^{-6}$.

As expected, the errors we see are about size ϵ. In Table 6.8 we do the same procedure for $\epsilon = 10^{-12}$. At higher N we lose one additional digit of accuracy. This loss may be due to 'weak' numerical problems such as the error in the FFT on large numbers of points. In any case, this error is not large.

	m			
N	0	32	128	512
128	$1.2 \cdot 10^{-12}$	$1.5 \cdot 10^{-12}$		
512	$1.6 \cdot 10^{-12}$	$2.2 \cdot 10^{-12}$	$2.5 \cdot 10^{-12}$	
2048	$1.7 \cdot 10^{-11}$	$2.8 \cdot 10^{-11}$	$1.1 \cdot 10^{-11}$	$5.7 \cdot 10^{-12}$

Table 6.8: A sampling of errors for $\epsilon = 10^{-12}$.

Chapter 7

An Alternate Algorithm

Here we describe an alternate method for a Fast Spherical Harmonic transform. It has not been implemented, but formally yields an $\mathcal{O}(N^2 \log N)$ algorithm, which is better than our main result of $\mathcal{O}(N^2 \log^2 N)$. It is more esthetically pleasing in that is uses more interesting properties of the Fourier transform. These properties, behavior under rotations and dilations, are more specific to our problem, and thus this method is less likely to have general utility.

The algorithm is based on the following observations:

1. $\sqrt{\sin x} P_n^m(\cos x) \approx \sqrt{x} J_m(nx)$ in some sense, on $[0, \pi/2]$. (See Figure 4.2.) This means the matrix converting an expansion in J_m's to one in P^m's should be almost diagonal.

2. We have a fast transform into $\sqrt{x} J_m(nx)$ (Hankel transform) for compactly supported functions, at least in theory. If we do a two dimensional Euclidean Fourier transform to $g(r, \theta) = \sum g_m(r) e^{im\theta}$, we obtain $\sum \tilde{g}_m(s) e^{imt}$ where $\tilde{g}_m(s)$ is the coefficient of expanding $g_m(r)$ into $\{C_s^m \sqrt{r} J_m(sr)\}$, and s corresponds to n. (See Section 2.2.)

7.1 Details

We now go through the steps in this algorithm. Many of the issues which arise are dealt with along the way. Others are simply highlighted as potential problems. Here we will outline the expansion transform.

Project from the sphere to a disc. We begin with data on the sphere. First project it onto its even and odd parts about the equator, i.e. $f_{\text{odd}}(\theta, \phi) = f(\theta, \phi) - f(\theta, \pi - \phi)$. These will be done separately and we will only discuss the odd case. Project this to a disc using $\phi \to r$.

Multiply by a Local Cosine type bell $b(r)$ which is 0 for $r > 3\pi/4$ and has the usual symmetry about $\pi/2$. We now have smooth data on the disc, and can capture and reproduce the odd part of our data.

Interpolate to the square. Our data is now given on the disc on a polar grid. We interpolate this to a rectangular grid on a slightly larger square. In order to have sufficient precision, it will be necessary for our original data to be given on a much finer grid than we obtain after interpolating. We will thus need to oversample by some constant factor depending on the desired precision ϵ. The interpolation method to be used, and the size of this constant, are unresolved issues.

FFT. Next we do a two dimensional FFT, at cost $\mathcal{O}(N^2 \log N)$. It is essential that our bell is smooth enough and sampling fine enough that we have no aliasing. Our output should have an essentially radial nature, but any aliasing is to the square as if it were a torus. This makes no sense: it is a case of trying to fit square data into a (smaller) round hole.

Interpolate to the disc. Now we re–interpolate to a polar grid on the disc. We have the same issues as above, and again we must end with less samples than we started with.

FFT on circles. Next do an FFT in the angle variable for each radius. This gives us

$$\{\tilde{g}_m(n/N) = \langle g_m(r), C_n^m \sqrt{r} J_m(rn/N) \rangle\}. \tag{7.1}$$

Interpolate to resample in r. We would like to claim that $\{C_n^m \sqrt{r} J_m(rn/N)\}_{n \geq m}$ is a basis on $[0, 3\pi/4]$ so that we have captured our (odd, belled) data. It is not, but would be if we renormalize and replace n by λ_{n-m}^m, the $(n-m+1)^{\text{th}}$ zero of J_m, and then stretch by multiplying by $3\pi/4$. We can correct the normalization with constants A_n^m, but we need to assure A_n^m doesn't grow or decay quickly in m or our problem becomes ill–conditioned. Correcting the dilation factor n amounts to interpolating to choose a different sampling of $\tilde{g}_m$. The resampling depends on m and so must be done after the FFT on circles. As before, our new sampling must be more coarse than the old by some factor depending on ϵ.

Change basis from J_m to P^m. We would now like to convert from J_m to P^m. We are using the J_m's as a basis for $[0, 3\pi/4]$ to represent data which is really odd across $\pi/2$ and has been belled. If we compute the representation of the odd P_n^m's with bell, applying this matrix will give us the spherical harmonic coefficients we were looking for. Here we are using the folding properties built into Local Cosine bells to eliminate the bell's effect.

The matrix we must compute is

$$M_{i,k}^m = \langle b(j/N)\sqrt{\sin(j/N)}\tilde{P}_{m+2i+1}^m(\cos(j/N)), \sqrt{j/N}J_m((j/N)\lambda_k^m 3\pi/4)\rangle \tag{7.2}$$

for $m + 2i + 1 < N/2$, and $m \leq k < 3N/4$. This matrix should be almost diagonal since J_m and P^m are eigenfunctions of

$$\frac{\partial^2}{\partial x^2} - \frac{m^2 - 1/4}{x^2} \tag{7.3}$$

and

$$\frac{\partial^2}{\partial x^2} - \frac{m^2 - 1/4}{\sin^2 x} \tag{7.4}$$

respectively, which are essentially the same. We need to prove this with rigorous theorems like those is Section 4.1. The bell takes care of any edge effects.

One portion of this algorithm which we have verified numerically is the diagonal nature of the P^m to J_m matrix. An example of this matrix is given in Figure 7.1. Compare this to Figure 6.2. Here we plot only the positive part. The striping indicates that there is fine structure in this diagonal band.

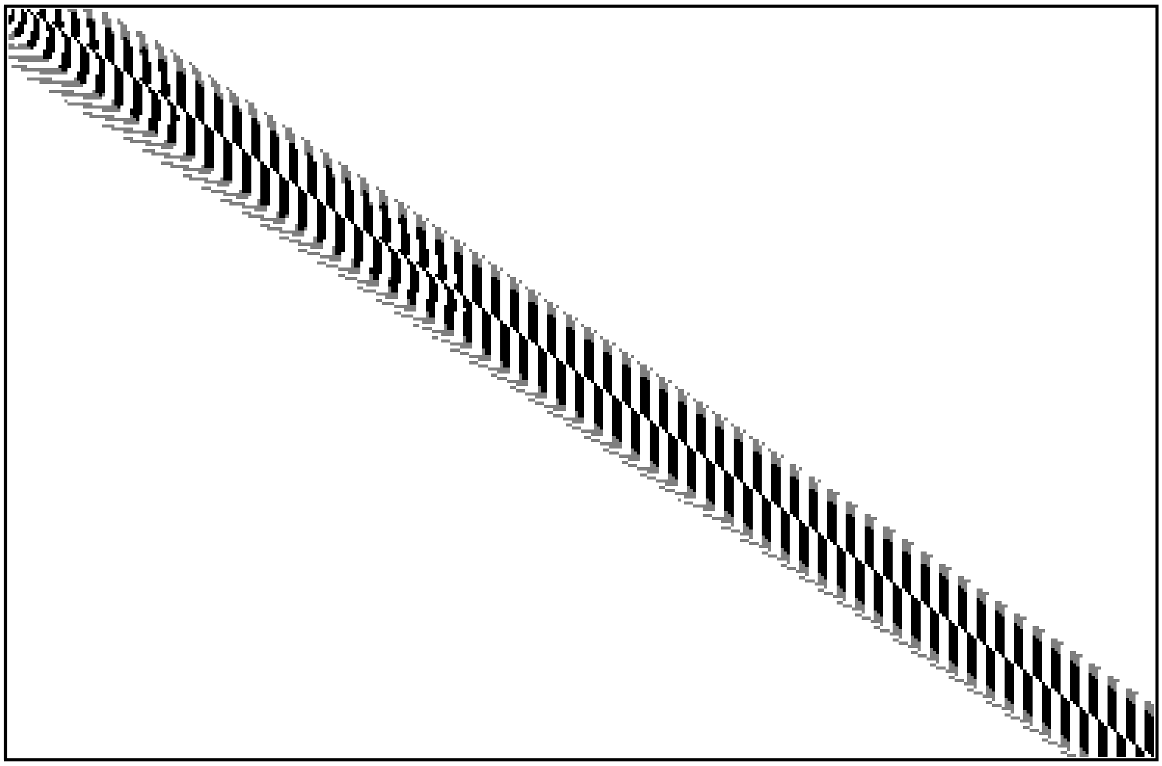

Figure 7.1: The positive part of the P^{32} to J_{32} localized transition matrix with $N = 1024$. Black is $> 10^{-6}$, gray is $> 10^{-12}$.

Bibliography

[AR91] Bradley Alpert and Vladimir Rokhlin. A fast algorithm for the evaluation of Legendre expansions. *SIAM Journal of Scientific and Statistical Computing*, 12(1):158–179, January 1991.

[AS64] Milton Abramowitz and Irene A. Stegun. *Handbook of Mathematical Functions*, volume 55 of *Applied Math Series*. National Bureau of Standards, 1964.

[AWW92] Pascal Auscher, Guido Weiss, and M. Victor Wickerhauser. Local sine and cosine bases of Coifman and Meyer and the construction of smooth wavelets. In C. K. Chui, editor, *Wavelets—A Tutorial in Theory and Applications*, pages 237–356. Academic Press, 1992.

[BCG93] B. Bradie, R. Coifman, and A. Grossmann. Fast numerical computations of oscillatory integrals related to acoustic scattering, i. *Applied and Computational Harmonic Analysis*, 1:94–99, 1993.

[BCR91] G. Beylkin, R. Coifman, and V. Rokhlin. Fast wavelet transforms and numerical algorithms i. *Communications on Pure and Applied Mathematics*, 44:141–183, 1991.

[Ben97] Nicholas N. Bennett. *Signal Analysis of Chirps: Detection, Oscillatory Kernels, and Anisotropic Wavelets.* PhD thesis, Yale University, New Haven CT, May 1997.

[BO78] Carl M. Bender and Steven A. Orszag. *Advanced mathematical methods for scientists and engineers.* International series in pure and applied Mathematics. McGraw–Hill, New York, 1978.

[Bro85] Timothy M. Brown. Solar rotation as a function of depth and latitude. *Nature*, 317(17):591–594, October 1985.

[CM91] Ronald R. Coifman and Yves Meyer. Remarques sur l'analyse de Fourier à fenêtre. *C. R. Académie des Sciences*, 312(1):259–261, 1991.

[Cor77] Antonio Cordoba. The Kakeya maximal function and the spherical summation multiplyers. *American Journal of Mathematics*, 99(1):1–22, 1977.

[CT65] James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computing*, 19:297–301, 1965.

[CW68] Ronald R. Coifman and Guido Weiss. Representations of compact groups and spherical harmonics. *L'Enseignement Mathématique*, 15(2):121–173, 1968.

[CW70] Ronald R. Coifman and Guido Weiss. *Analyse Harmonique Non-Commutative sur Certains Espsaces Homogenes*. Number 242 in Lecture Notes in Mathematics. Springer-Verlag, New York, 1970.

[DEH89] Thomas L. Duvall, Jr., Mark Elowitz, and Frank Hill. A test of a modified algorithm for computing spherical harmonic coefficients using an fft. *Journal of Computational Physics*, 80:506–511, 1989.

[DH94] James R. Driscoll and Dennis M. Healy, Jr. Computing Fourier transforms and convolutions on the 2–sphere. *Advances in Applied Mathematics*, 15:202–250, 1994.

[Dil85] Gary A. Dilts. Computation of spherical harmonic expansion coefficients via fft's. *Journal of Computational Physics*, 57:439–453, 1985.

[DR93] A. Dutt and V. Rokhlin. Fast Fourier trasforms for nonequispaced data. *SIAM Journal of Scientific Computing*, 14(6):1368–1393, November 1993.

[DR95] A. Dutt and V. Rokhlin. Fast Fourier transforms for nonequispaced data, ii. *Applied and Computational Harmonic Analysis*, 2:85–100, 1995.

[EMOT54] Arthur Erdélyi, Wilhelm Magnus, Fritz Oberhettinger, and Francesco G. Tricomi. *Tables of Integral Transforms*, volume 2 of *Bateman Manuscript Project*. McGraw-Hill, New York, 1954.

[FPTV92] B. Flannery, W. Press, S. Teukolsky, and W. Vettering. *Numerical Recipies in C*. Cambridge University Press, Cambridge UK, 2nd edition, 1992.

[Fru95] M. Frumkin. A fast algorithm for expansion over spherical harmonics. *Applicable Algebra in Engineering, Communication and Computing*, 6:333–343, 1995.

[Hob31] E. W. Hobson. *The Theory of Spherical and Ellipsoidal Harmonics.* Chelsea, New York, 1931.

[Leb72] N. N. Lebedev. *Special Functions and Their Applications.* Dover Publications, Inc., New York, 1972. translated by Richard A. Silverman.

[LL77] L. D. Landau and E. M. Lifschitz. *Quantum Mechanics (Non–relativistic Theory).* Pergamon Press, New York, 3rd edition, 1977.

[Mat96] Gregory Matviyenko. Optimized local trigonometric bases. *Applied and Computational Harmonic Analysis*, 3(4):301–323, 1996.

[Orz86] Steven A. Orzag. Fast eigenfunction transforms. *Advances in Mathematics Supplementary Studies*, 10, 1986.

[Sla96] S. Yu. Slavyanov. *Asymptotic Solutions of the One–Dimensional Schrödinger Equation*, volume 151 of *Translations of Mathematical Monographs.* American Mathematical Society, Providence, RI, 1996.

[SW71] Elias Stein and Guido Weiss. *Fourier Analysis on Euclidean Spaces.* Princeton University Press, Princeton NJ, 1971.

[Sze75] Gabor Szegö. *Orthogonal Polynomials.* AMS, Providence, RI, 4th edition, 1975.

[TV96] Christoph M. Thiele and Lars F. Villemos. A fast algorithm for adapted time–frequency tilings. *Applied and Computational Harmonic Analysis*, 3(2):91–99, 1996.

[Was65] Wolfgang Wasow. *Asymptotic Expansions for Ordinary Differential Equations*, volume 14 of *Pure and Applied Mathematics.* Interscience (John Wiley & Sons), New York, 1965.

[Wic94] Mladen Victor Wickerhauser. *Adapted Wavelet Analysis from Theory to Software.* A.K. Peters, Wellesley, MA, 1994.

[Wol95] Thomas Wolff. An improved bound for Kakeya type maximal functions. *Revista Matematica Iberoamericana*, 11(3), 1995.