# FFTs on the Rotation Group

Peter J. Kostelec[*]
and
Daniel N. Rockmore[†]
Department of Mathematics
Dartmouth College
Hanover, NH 03755

December 3, 2003

**Abstract**

Earlier work by Driscoll and Healy [4] has produced an efficient $O(B^2 \log^2 B)$ algorithm for computing the Fourier transform of band-limited functions on the 2-sphere. In this paper, we discuss an implementation of an $O(B^4)$ algorithm for the numerical computation of Fourier transforms of functions defined on the rotation group, $SO(3)$. This compares with the direct $O(B^6)$ approach. The algorithm we implemented is based on the "Separation of Variables" technique, e.g. as presented by Maslen and Rockmore [19]. In conjunction with the techniques developed in [4], the $SO(3)$ algorithm we implemented may be made truly fast, $O(B^3 \log^2 B)$. Basic results will be presented establishing the algorithm's numerical stability, and examples of applications will be presented.

## 1 Introduction

We present and discuss an implementation of an algorithm for the efficient calculation of the discrete Fourier transform of a function defined on the rotation group, $SO(3)$. This software, which we christened "The *SOFT* Package," is freely available on the web [27].

This implementation differs from that as derived by Risbo [24], which uses properties of the Wigner-$d$ function to rewrite the transform over the full group in such a way as to allow the use of three Cooley-Tukey FFTs. However, this introduces overhead which can be avoided, if one is willing to keep the "natural" structure of the algorithm. The algorithm we present preserves this structure, as well as the freedom of applying a Driscoll-Healy based algorithm (e.g. see [12]), which has the potential for an algorithm faster than $O(B^4)$.

This paper is organized as follows. We begin with a brief introduction to $SO(3)$, including some discussion of harmonic analysis. This is then followed by introducing the Sampling Theorem for functions defined on $SO(3)$, and outlining what exactly we mean by the "Separation of Variables" approach. Due to the plethora of conventions that exist, with respect to things such as normalizations, in Section 4 we define explicitly the Wigner-$d$ functions, as implemented in the software. Following this, in Section 5 we present error and timing results of our implementation. This is then followed by presenting an application of Fourier transforms defined on $SO(3)$: correlating two functions defined on $S^2$. This is useful in a variety of applications, including searchable 3-D databases [25], molecular biology [16], and industrial manufacturing [11]. We mention that another application of Fourier transforms functions defined on $SO(3)$ concerns spherical near-field antenna measurements [9]. We conclude with a brief recap and discussion.

## 2 Life in $SO(3)$

The group of proper rotations of $\mathbb{R}^3$ about the origin, denoted $SO(3)$, consists of real 3-by-3 orthogonal matrices of determinant +1. Using the familiar Euler angle decomposition, we may express any such element $g$ of $SO(3)$

[*]Supported by NSF ITR award.

[†]Supported by NSF ITR award and the Santa Fe Institute.

in terms of rotations about the $z$ and $y$ axes. Let

$$u(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad a(\beta) = \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix} \tag{1}$$

where $0 \le \alpha < 2\pi$ and $0 \le \beta \le \pi$. Geometrically speaking, $u(\alpha)$ corresponds to rotation about the $z$-axis, and $a(\beta)$ to rotation about the $y$-axis. Using these two matrices, any rotation $g \in SO(3)$ may be written as:

$$g = u(\alpha)\, a(\beta)\, u(\gamma). \tag{2}$$

Let $L^2(SO(3))$ denote the space of square integrable functions on $SO(3)$. The inner product of two functions $f$ and $h$ on $SO(3)$ is given as

$$< f, h > = \int_0^{2\pi} d\alpha \int_0^{\pi} d\beta \sin\beta \int_0^{2\pi} d\gamma \;\; f(\alpha,\beta,\gamma)\; h^*(\alpha,\beta,\gamma). \tag{3}$$

To each $g \in SO(3)$, we can associate a linear operator $\Lambda(g)$ which acts on a function $f$ in $L^2(S^2)$:

$$\Lambda(g) f(\omega) = f(g^{-1}\omega). \tag{4}$$

This is the left regular representation of $SO(3)$ on $L^2(S^2)$. The invariant subspaces, $V_l$, are indexed by the non-negative integers. The dimension of each is $\dim V_l = 2l+1$. The familiar spherical harmonics span these invariant subspaces. To be precise, for a given $l$, $V_l$ is spanned by the spherical harmonics of degree $l$ and orders $|m| \le l$: $\{Y_l^m(\omega) \mid |m| \le l\}$. We therefore have

$$\Lambda(g) Y_l^m(\omega) = \sum_{|k| \le l} Y_l^k(\omega) D_{km}^l(g) \tag{5}$$

where $D_{km}^l(g)$ is a Wigner-$D$ function. (Brief aside: we will sometimes write the spherical harmonic of degree $l$ and order $m$ as $Y_{lm}$. This should not confuse the reader.) Using the Euler angle decomposition, we can write Wigner-$D$ function as

$$D_{MM'}^J(\alpha,\beta,\gamma) = e^{-iM\alpha}\, d_{MM'}^J(\beta)\, e^{-iM'\gamma}, \qquad J, M, M' \text{ integers} \tag{6}$$

where $d_{MM'}^J(\beta)$ denotes what we will refer to as the **Wigner $d$-function**. The Wigner-$d$ functions are related to the Jacobi polynomials, and satisfy a three-term recurrence relation. Explicit expressions for $d_{MM'}^J(\beta)$ will be given in Section 4.

Applying the operator to a function $f \in L^2(S^2)$ can be thought of as rotating its graph. How do the coefficients of $f$ relate to its rotated cousin? Using (5), we can elegantly obtain the Fourier expansion of the rotated $f$, by means of a matrix multiplication. The matrix in question is a semi-infinite block diagonal matrix, where the $l^{\text{th}}$ block is of size $(2l+1) \times (2l+1)$, and each block is $D^l(g) = \left(D_{mn}^l\right)(g)$. Using all this, the Fourier expansion of the rotated $f$ is the matrix-vector product

$$\left(\begin{array}{c|c|c|c} D^0(g) & & & \\ \hline & D^1(g) & & \\ \hline & & D^2(g) & \\ \hline & & & \ddots \end{array}\right) \cdot \begin{pmatrix} \hat{f}_0^0 \\ \hdashline \hat{f}_1^{-1} \\ \hat{f}_1^0 \\ \hat{f}_1^1 \\ \hdashline \hat{f}_2^{-2} \\ \hat{f}_2^{-1} \\ \hat{f}_2^0 \\ \vdots \end{pmatrix}$$

## 2.1 Harmonic Analysis

By the Peter-Weyl Theorem, the collection of functions $\{D^J_{MM'}(\alpha,\beta,\gamma)\}$ form a complete set of orthogonal functions with respect to integration over $SO(3)$:

$$\int_0^{2\pi} d\alpha \int_0^{\pi} d\beta \sin\beta \int_0^{2\pi} d\gamma \;\; D^{J_2 *}_{M_2 M'_2}(\alpha,\beta,\gamma) D^{J_1}_{M_1 M'_1}(\alpha,\beta,\gamma) = \frac{8\pi^2}{2J_1+1} \delta_{J_1 J_2}\; \delta_{M_1 M_2}\; \delta_{M'_1 M'_2}. \tag{7}$$

Hence, any function $f \in L^2(SO(3))$ may be written as a sum of the Wigner $D$-functions:

$$f(\alpha,\beta,\gamma) = \sum_{J\geq 0} \sum_{M=-J}^{J} \sum_{M'=-J}^{J} \hat{f}^J_{MM'} D^J_{MM'}(\alpha,\beta,\gamma) \tag{8}$$

where

$$\begin{aligned} \hat{f}^J_{MM'} &= \langle f, D^J_{MM'} \rangle \\ &= \frac{2J+1}{8\pi^2} \int_0^{2\pi} d\alpha \int_0^{\pi} d\beta \sin\beta \int_0^{2\pi} d\gamma \;\; f(\alpha,\beta,\gamma) D^{J*}_{MM'}(\alpha,\beta,\gamma). \end{aligned} \tag{9}$$

We will refer to the set of values $\{\hat{f}^J_{MM'}\}$ as the **Fourier coefficients** of $f$, and the map going from $f(\alpha,\beta,\gamma)$ to the set of coefficients $\{\hat{f}^J_{MM'}\}$ as the ***SO*(3) Fourier transform of *f***. As a notational convenience, we will denote the $SO(3)$ Fourier transform of $f$ as $SOFT(f)$.

## 2.2 Convolution on $L^2(SO(3))$

Convolution of two functions $f_1, f_2 \in L^2(SO(3))$ is defined to be

$$f_1 * f_2(h) = \int_{SO(3)} f_1(hg^{-1}) \; f_2(g) \; dg$$

where $h \in SO(3)$. Given the Fourier coefficients in the expansions of $f_1$ and $f_2$:

$$\begin{aligned} f_1(g) &= \sum_{J\geq 0} \sum_{M=-J}^{J} \sum_{M'=-J}^{J} a^J_{MM'} D^J_{MM'}(g) \\ f_2(g) &= \sum_{J\geq 0} \sum_{M=-J}^{J} \sum_{M'=-J}^{J} b^J_{MM'} D^J_{MM'}(g), \end{aligned}$$

we find the Fourier coefficients of their convolution to be

$$c^J_{MM'} = \sum_{k=-J}^{J} a^J_{Mk} b^J_{kM'} \tag{10}$$

Details may be found in Vilenkin [31].

One may obtain the Fourier coefficients of their convolution in a very elegant manner. Note that corresponding to each integer $J$ is an irreducible representation of $SO(3)$. Arrange the coefficients of $f_1$ as a block diagonal matrix, where the $J$-th block, of size $(2J+1)\times(2J+1)$, contains the (suitably ordered) Fourier coefficients at that degree $J$. Now do the same for $f_2$. The Fourier coefficients of the convolution is just the product of the two matrices. So all that is happening is that the $J$-th block in $f_1$'s matrix is multiplied with the $J$-th block in $f_2$'s matrix. Neat.

## 2.3 Sampling Theorem

Much of the discussion which follows is based on [19].

To take the Fourier transform of a function $f$ on $SO(3)$ means to evaluate the inner products (9). In practice, to even attempt to do this on the computer means that we must discretize the integral, i.e. replace it with a finite sum. This compels us to sample $f$ on some finite grid, $X$. Ergo, the sum we need to evaluate is of the general form:

$$\hat{f}^l_{MM'} = \sum_{x \in X} w(x) f(x) D^{l*}_{MM'}(x) \tag{11}$$

where $w(x)$ is some weighting function. Furthermore, to ensure that we may accurately compute the Fourier transform of $f$ this way, certain conditions must be placed on $f$.

**Definition 2.1** (Band-limited functions on $SO(3)$). A continuous function $f$ on $SO(3)$ is **band-limited with band-limit (or bandwidth)** $B$ if $\hat{f}^l_{MM'} = 0$ for all $l \geq B$.

Using the matrix factorization (2), let $X_B$ denote the set of points where we wish to sample:

$$X_B = \{u(\alpha_{j_1}) a(\beta_k) u(\gamma_{j_2}) | 0 \leq j_1, j_2, k < 2B\} \tag{12}$$

where $\alpha_j = \gamma_j = \dfrac{2\pi j}{2B}$ and $\beta_k = \dfrac{\pi(2k+1)}{4B}$. For functions of band-limit $B$, sampled on the grid $X_B$, it is possible to choose a weight function $w(x)$ so that the discrete sum (11) exactly computes their Fourier transform. Let $w_B(k), 0 \leq k < 2B$, form the unique solution to the system of linear equations

$$\sum_{k=0}^{2B-1} w_B(K)\ P_m(\cos \beta_k) = \delta_{0m} \ \text{ for } \ 0 \leq m < B \tag{13}$$

where $P_m$ denotes the Legendre polynomial of degree $m$. Using the solution to (13), we have the following result, the proof of which may be found in [19].

**Theorem 2.1** *Suppose $f \in L^2(SO(3))$ is a band-limited function with band-limit $B$. Then for all $l < B$, we have*

$$f^l_{MM'} = \frac{1}{(2B)^2} \sum_{j_1=0}^{2B-1} \sum_{j_2=0}^{2B-1} \sum_{k=0}^{2B-1} w_B(k) f(\alpha_{j_1}, \beta_k, \gamma_{j_2}) D^{l*}_{MM'}(\alpha_{j_1}, \beta_k, \gamma_{j_2}). \tag{14}$$

*where the sample points are those defined in (12) and weights $w_B(k)$ are the solutions to the linear system (13).*

The method of computing the Fourier coefficients of $f$ by summing (14) will be called the **Discrete $SO(3)$ Fourier Transform** of $f$, $DSOFT(f)$.

It turns out, in fact, that the weights have a closed form expression.

**Lemma 2.2** *[ cf. [4] ] Let $\beta_k = \pi(2k+1)/4B$. The solutions to the linear system of equations (13) have the closed form expression*

$$w_B(k) = \frac{2}{B} \sin\left(\frac{\pi(2j+1)}{4B}\right) \sum_{k=0}^{B-1} \frac{1}{2k+1} \sin\left((2j+1)(2k+1)\frac{\pi}{4B}\right). \tag{15}$$

## 3 Separation of Variables

Details, and a great many references, of the Separation of Variables technique, may be found in [19, 20].

Theorem 2.1 tells us that we can compute exactly the Fourier coefficients of a properly sampled band-limited function $f$ of band-limit $B$. However, if we were to evaluate (14) as it is written, each Fourier coefficient would require $O(B^3)$ operations. There being $O(B^3)$ coefficients $\hat{f}^l_{MM'}$ to compute, the cost of computing the complete Fourier transform of $f$ is a tad on the large side. Is it possible to reorganize the computation and hence reduce the total complexity? Yes, by application of the Separation of Variables technique.

The basic strategy is to make a prudent choice of factorization of the group elements of $SO(3)$ and use of adapted set of representations, or Gel'fand-Tsetlin bases, to re-index the sum (11) as a multiple sum. (Because we are using the Euler angle decomposition, Eqn. (14) is already in the proper form). Next, terms that do not depend on particular indeces are factored through the individual sums, and then finally one evaluates the sums coordinate by coordinate.

Hence, by using the fact that $D^l_{MM'}(\alpha,\beta,\gamma) = e^{-iM\alpha}\, d^l_{MM'}(\beta)\, e^{-iM'\gamma}$, we may rewrite (14) as

$$f^l_{MM'} = \frac{1}{(2B)^2} \sum_{k=0}^{2B-1} w_B(k) d^l_{MM'}(\beta_k) \sum_{j_2=0}^{2B-1} e^{iM'\gamma_{j_2}} \sum_{j_1=0}^{2B-1} e^{iM\alpha_{j_1}} f(\alpha_{j_1},\beta_k,\gamma_{j_2}). \tag{16}$$

This form then easily lends itself to efficiently computing $\hat{f}^l_{MM'}$: first sum over $j_1$, then $j_2$ and conclude with summing over $k$. To be slightly more explicit, one sums the following quantities (*note the indeces they depend on*) for $0 \le j_2, k < 2B$ and $|M|, |M'| \le l < B$:

$$\begin{aligned} S_1(k,M,j_2) &= \frac{1}{(2B)} \sum_{j_1=0}^{2B-1} e^{iM\alpha_{j_1}} f(\alpha_{j_1},\beta_k,\gamma_{j_2}) && (17)\\ S_2(k,M,M') &= \frac{1}{(2B)} \sum_{j_2=0}^{2B-1} e^{iM'\gamma_{j_2}} S_1(k,M,j_2) && (18)\\ \hat{f}^l_{MM'} &= \sum_{k=0}^{2B-1} w_B(k) d^l_{MM'}(\beta_k) S_2(k,M,M'). && (19)\end{aligned}$$

For a given pair $k$, $j_2$, one may evaluate the sum $S_1(k,M,j_2)$ for all $M$ in $O(B\log B)$ operations using a standard, Cooley-Tukey-esque FFT. Therefore, to compute $S_1(k,M,j_2)$ for all $k,M,j_2$ takes $O(B^3\log B)$ operations. For identical reasons, the same big-$O$ is applicable for computing $S_2(k,M,M')$. All that remains is computing that last sum (19). For given $M,M'$, $O(B^2)$ operations are needed to compute $\hat{f}^l_{MM'}$ for all $l$. Therefore, putting all the pieces together yields an $O(B^4)$ algorithm for computing the Fourier transform of $f$, substantially smaller than the original, "naive", $O(B^6)$ computation. Zowie! From now on, we will refer to this method of computing all the Fourier coefficients of $f$ as the ***SO*(3) Fast Fourier transform of *f***, $SOFFT(f)$.

In evaluating that last sum, (19), it is possible to apply Risbo's technique [24], using the identity (from [23])

$$d^l_{MM'}(\beta) = \imath^{M-M'} \sum_{M''=-j}^{j} d^j_{M''M}(\pi/2) e^{-\imath M''\beta} d^j_{M''M'}(\pi/2),$$

and so rewrite the entire computation in terms of three Cooley-Tukey FFTs. However, this algorithm is $O(B^4)$, and its structure precludes the possibility of obtaining an asymptotically faster algorithm. That is not to imply that it is not fast, i.e. in terms of looking at the clock on the wall. It all depends on the application, and what is important to *you*. Using Risbo's technique, or the one presented here, either will get the job done.

Given the well-know numerical reliability of the Cooley-Tukey FFT, there is very little reason to investigate the quality of the sums (17-18). The place where errors would be introduced (and exacerbated) is in the third sum (19). Given its import, we are motivated to make the following definition:

**Definition 3.1** For given integers $(M,M')$, define the **Discrete Wigner Transform (*DWT*) of a data vector s** to be the collection of sums of the form

$$\hat{\mathbf{s}}(l,M,M') = \sum_{k=0}^{2B-1} w_B(k)\; d^l_{M,M'}(\beta_k)[\mathbf{s}]_k \qquad \max(|M|,|M'|) \le l < B \tag{20}$$

where $d^l_{M,M'}$ is a Wigner d-function of degree $l$ and orders $M$, $M'$, and $\beta_k = \dfrac{\pi(2k+1)}{4B}$.

In our case, the data vector $\mathbf{s}$ is simply the original function $f$ sampled at the $2B$-many locations $\beta_k$ defined above. In Section 5 we report the results of our experiments, testing the numerical validity of the DWT for a variety of band-limits, as well as the $SOFFT$ algorithm.

The DWT may be cast in an linear algebraic light. Let $\mathbf{s}$ = data vector, $\hat{\mathbf{s}}$ = coefficient vector, $\mathbf{w}$ = diagonal matrix whose entries are the weights, $\mathbf{d}$ = sampled Wigner-$d$s, $d_{ij} = d^i_{MM'}(\beta_j)$. Then

$$\mathbf{d} * \mathbf{w} * \mathbf{s} = \hat{\mathbf{s}}$$

is the forward transform, and

$$\mathbf{d}^T * \hat{\mathbf{s}} = \mathbf{s}$$

is the inverse transform. Note also that

$$\begin{aligned} \mathbf{d} * \mathbf{w} * \mathbf{s} &= \hat{\mathbf{s}} \\ \mathbf{d} * \mathbf{w} * \left(\mathbf{d}^T * \hat{\mathbf{s}}\right) &= \hat{\mathbf{s}} \\ \left(\mathbf{d} * \mathbf{w} * \mathbf{d}^T\right) * \hat{\mathbf{s}} &= \hat{\mathbf{s}} \end{aligned}$$

Therefore $\mathbf{d} * \mathbf{w} * \mathbf{d}^T = I$, the identity matrix.

**Remark.** To compute the *DWT*, the methods of Driscoll and Healy [4] are certainly applicable here. Instead of taking $O(B^2)$ operations evaluating the sum naively, with the Driscoll-Healy one could calculate the $d$-coefficients in only $O(B \log^2 B)$ operations. Using their approach would then yield an $O(B^3 \log^2 B)$ "really" $SOFFT$ algorithm. However, this was not done in the C code. We will explain our reasoning in Section 5.2.

## 4 Explicit Wigners

Everyone agrees that the Wigner $D$-function may be written as

$$D^J_{MM'}(\alpha, \beta, \gamma) = e^{-iM\alpha}\, d^J_{MM'}(\beta)\, e^{-iM'\gamma},$$

where, we remind the reader, $d^J_{MM'}(\beta)$ denotes the Wigner $d$-function. However, it is highly likely that if one were to meet two mathematicians, Fred and Ethel, on the street and ask each of them to write down an expression for $d^J_{MM'}(\beta)$, one would receive two different (though, of course, equivalent) replies. For example, Fred might refer to [1] and respond with

$$\begin{aligned} d^J_{MM'}(\beta) &= \sqrt{\frac{(J+M')!\,(J-M')!}{(J+M)!\,(J-M)!}} \left(\sin\frac{\beta}{2}\right)^{M'-M} \left(\cos\frac{\beta}{2}\right)^{M+M'} \\ &\quad \times P^{(M'-M, M'+M)}_{J-M'}(\cos\beta) \end{aligned} \tag{21}$$

where $P^{(m,n)}_l(x)$ denotes a Jacobi polynomial. However, Ethel could very easily prefer dealing with derivatives, and cite this particular definition given in [23]:

$$\begin{aligned} d^J_{MM'}(\beta) &= C_{JM'} \sqrt{\frac{(J+M)!}{(J-M)!(2J)!}}\ (1-t)^{-(M-M')/2}(1+t)^{-(M+M')/2} \\ &\quad \times \frac{d^{J-M}}{dt^{J-M}} \left[(1-t)^{J-M'}(1+t)^{J+M'}\right] \end{aligned} \tag{22}$$

where $t = \cos\beta$ and $C_{JM'} = (-1)^{J-M'} 2^{-J} \sqrt{\dfrac{(2J)!}{(J+M')!(J-M')!}}$.

In light of such diversity, we feel it appropriate to provide the reader with the precise definition of $d^J_{MM'}(\beta)$, as well as other relevant functions, upon which our C code for computing the Fourier transform of band-limited functions $f \in L^2(SO(3))$ is based. All these definitions may be found in [30].

Following Fred's lead, we too use Jacobi polynomials to express the Wigner $d$-function, though at first glance, the expressions may not seem equivalent:

$$\begin{aligned} d^J_{MM'}(\beta) &= \zeta_{MM'}\sqrt{\frac{s!(s+\mu+\nu)!}{(s+\mu)!(s+\nu)!}}\left(\sin\frac{\beta}{2}\right)^{\mu}\left(\cos\frac{\beta}{2}\right)^{\nu} \\ &\quad \times P^{(\mu,\nu)}_s(\cos\beta) \end{aligned} \tag{23}$$

where

$$\mu = |M - M'| \quad \nu = |M + M'| \quad s = J - \frac{\mu+\nu}{2}$$

and

$$\zeta_{MM'} = \begin{cases} 1 & \text{if } M' \geq M \\ (-1)^{M'-M} & \text{if } M' < M. \end{cases}$$

Note that unless $J \geq \max(|M|, |M'|)$, we have $d^J_{MM'}(\beta) = 0$. The $d$-functions satisfy the orthogonality condition

$$\int_0^\pi d^J_{MM'}(\beta) d^{J'}_{MM'}(\beta)\ \sin\beta\ d\beta = \frac{2}{2J+1}\delta_{JJ'}, \tag{24}$$

in addition to the following three-term recurrence:

$$\begin{aligned} 0 &= \frac{\sqrt{\left[(J+1)^2 - M^2\right]\left[(J+1)^2 - M'^2\right]}}{(J+1)(2J+1)}\ d^{J+1}_{MM'}(\beta) + \left(\frac{MM'}{J(J+1)} - \cos\beta\right) d^J_{MM'}(\beta) \\ &\quad + \frac{\sqrt{(J^2-M^2)(J^2-M'^2)}}{J(2J+1)}\ d^{J-1}_{MM'}(\beta) \end{aligned} \tag{25}$$

To properly initialize the above recurrence, we have found the following special cases especially useful:

$$\begin{aligned} d^J_{JM}(\beta) &= \sqrt{\frac{(2J)!}{(J+M)!(J-M)!}}\left(\cos\frac{\beta}{2}\right)^{J+M}\left(-\sin\frac{\beta}{2}\right)^{J-M} \\ d^J_{-JM}(\beta) &= \sqrt{\frac{(2J)!}{(J+M)!(J-M)!}}\left(\cos\frac{\beta}{2}\right)^{J-M}\left(\sin\frac{\beta}{2}\right)^{J+M} \\ d^J_{MJ}(\beta) &= \sqrt{\frac{(2J)!}{(J+M)!(J-M)!}}\left(\cos\frac{\beta}{2}\right)^{J+M}\left(\sin\frac{\beta}{2}\right)^{J-M} \\ d^J_{M-J}(\beta) &= \sqrt{\frac{(2J)!}{(J+M)!(J-M)!}}\left(\cos\frac{\beta}{2}\right)^{J-M}\left(-\sin\frac{\beta}{2}\right)^{J+M}. \end{aligned} \tag{26}$$

For us, their utility comes not from any stability-related issues, but rather derives from the fact that when we were prototyping the code in *Mathematica*, we were able to avoid any potential ambiguities, e.g. "Is *Mathematica* placing a '-1' out in front?" Using the above special cases, we were able to confirm correct generation of the $d$-functions by comparison with the tables found in [30].

For computational purposes we deal with the $L^2$-normalized cousins of the Wigner $d$-functions,

$$\tilde{d}^J_{MM'}(\beta) = \sqrt{\frac{2J+1}{2}}\ d^J_{MM'}(\beta). \tag{27}$$

Therefore, for the convenience of the reader, we now provide the three-term recurrence relation the $\tilde{d}^J_{MM'}$ satisfy.

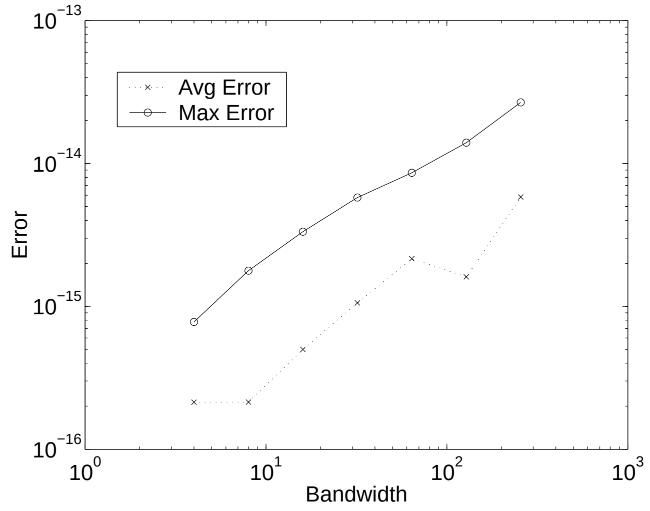


Figure 1: The average and maximum values of Eqns (29) at different bandwidths.

This is the actual relation the C code uses to generate the normalized Wigner $d$-functions:

$$\begin{aligned}\tilde{d}^{J+1}_{MM'}(\beta) \quad &= \sqrt{\frac{2J+3}{2J+1}}\ \frac{(J+1)(2J+1)}{\sqrt{\left[(J+1)^2-M^2\right]\left[(J+1)^2-M'^2\right]}}\left(\cos\beta-\frac{MM'}{J(J+1)}\right)\tilde{d}^{J}_{MM'}(\beta)\\ &\quad -\sqrt{\frac{2J+3}{2J-1}}\ \frac{\sqrt{\left[J^2-M^2\right]\left[J^2-M'^2\right]}}{\sqrt{\left[(J+1)^2-M^2\right]\left[(J+1)^2-M'^2\right]}}\ \frac{J+1}{J}\ \tilde{d}^{J-1}_{MM'}(\beta). \qquad (28)\end{aligned}$$

To test to some extent the stability of the above recurrence, we did the following. At a given bandwidth $B$ and orders $M$, $M'$, we write the DWT as a matrix product: $\mathbf{d}*\mathbf{w}*\mathbf{f}=\hat{\mathbf{f}}$, where $\mathbf{f}$ is the $2B\times1$ vector containing the samples of $f$, $\mathbf{w}$ is the $2B\times 2B$ diagonal matrix containing the weights, and $\mathbf{d}$ is the $(B-\max(|M|,|M'|)\times 2B$ matrix containing the sampled Wigner $d$-functions. The vector $\hat{\mathbf{f}}$ contains the coefficients of $f$. Now, if instead we formed the matrix product

$$\mathbf{d}*\mathbf{w}*\mathbf{d}^T,$$

where $\mathbf{d}^T$ is the transpose of $\mathbf{d}$, we get the identity matrix.

Ok, so how close to the identity matrix do we get in real life? In Matlab, at a given bandwidth $B$, and orders $M,\ M'$, we calculated two quantities:

$$\begin{aligned}1/B^2\sum|\mathbf{d}*\mathbf{w}*\mathbf{d}^T-I|\\ \max|\mathbf{d}*\mathbf{w}*\mathbf{d}^T-I|,\end{aligned} \qquad (29)$$

where $I$ is the identity matrix, and the sum is done over the matrix elements. We calculated the above for orders $M=0,\ldots,B-1$, and $M'=M,\ldots,B-1$. In Figure 1, we plot the maximum values of each quantity obtained at different bandwidths. Agreement with the identity matrix is quite good.

## 5 Numerical Results

In this section, we present numerical evidence which indicates that one may accurately compute the Fourier transform of band-limited functions $f\in L^2(SO(3))$ via the Separation of Variables technique, at least at realistic (to be explained later) problem sizes. We will also present some timing results, and discuss issues of computational efficiency.

All code was written in C, with some of the more basic routines borrowed from *SpharmonicKit* [26]. In certain instances, which will be made clear, we availed ourselves to the Fourier transform routines provided by *FFTW* [8]. Experiments were performed on a variety of mostly GNU/Linux platforms running a variety of processors: an Intel 800 MHz Pentium-III, an Intel 2.4 GHz Xeon, and an AMD 1.66 GHz Athlon MP 2000+. The flavour of GNU/Linux on these machines was either RedHat or Debian. The non-GNU/Linux platform was an HP/Compaq AlphaServer with a 833 MHz Alpha processor running OSF1 V5.1. While some of this hardware was multi-processor, the code was always run on a single processor. No parallelizing was done. All code was compiled with available optimizations. In the case of the HP/Compaq machine, the native compiler, and not gcc, was used.

Figure 2 provides a coarse description of a typical experiment. The process illustrated in the figure would be

$$\begin{tikzcd}
\text{Generate random coefficients} \arrow[d, "\text{Inverse transform}"] \\
\text{Compute sample values} \arrow[d, "\text{Forward transform}"] \\
\text{Compute new coefficients}
\end{tikzcd}$$

Figure 2: Our experiment, philosophically speaking.

iterated some number of times, at the end of which the average absolute and relative errors would be calculated.

## 5.1 "Just the Wigners, ma'am"

Before attempting to compute $SOFFT(f)$, the discrete Fourier transform of $f$ over $SO(3)$, for our first series of experiments we investigated the stability of the discrete Wigner $d$-function transform (DWT), with the sums being evaluated naively. We wanted to ensure that an actual implementation of this critical component of the Fourier transform over the full group was in fact stable for a large range of problem sizes. To flesh out the outline given in Figure 2, a precise description of this experiment now follows.

1. Choose a bandwidth $B$, and orders $M$, $M'$.
2. Generate random Wigner $d$-coefficients $\{\hat{f}^k_{MM'}\}$ where $\max(|M|, |M'|) \leq k \leq B-1$, uniformly distributed between $-1$ and 1.
3. Take the inverse DWT of the above coefficients, resulting in the 2 $B$-many function samples $\{f(\beta_j) \mid 0 \leq j \leq 2\ B-1\}$ where $\beta_j = \dfrac{\pi(2j+1)}{4\ B}$.
4. Take the forward DWT of the above sample values, resulting in a new set of $d$-coefficients, $\hat{g}^k_{MM'}$.
5. Compute the error
$$\max_k \left\| \hat{f}^k_{MM'} - \hat{g}^k_{MM'} \right\|$$
6. Repeat Steps 2 through 5 one hundred thousand times.
7. Compute the average absolute and relative errors over all trials.

The above experiment was run for various bandwidths $B$, and combinations $M$ and $M'$. The results obtained on the 2.4 Xeon are given in Table 1. Runs on other platforms displayed similar behavior.

At the smaller bandwidths, the errors are all acceptably and gratifyingly small. However, a careful examination of Table 1 reveals that as the bandwidth increases, the relative error eventually approaches values that may give one pause. By the time we reach bandwidth $B = 1024$, we are seeing relative errors roughly on the

| Bandwidth B | $M = M' = 0$ | $M = B/2,\ M' = 0$ | $M = M' = B/2$ |
|---|---|---|---|
| 16 | 2.0990e-12<br>9.8256e-11 | 2.1644e-12<br>3.7374e-11 | 1.9806e-12<br>2.6711e-11 |
| 32 | 2.3308e-12<br>3.4070e-10 | 3.3753e-12<br>2.5885e-10 | 2.3639e-12<br>3.0303e-10 |
| 64 | 1.1389e-11<br>3.7872e-09 | 9.1509e-12<br>1.5096e-09 | 1.1076e-11<br>1.6904e-09 |
| 128 | 3.5974e-11<br>3.2129e-08 | 2.8620e-11<br>8.0481e-09 | 3.6464e-11<br>4.5913e-08 |
| 256 | 1.2473e-10<br>2.0330e-07 | 9.2109e-11<br>4.1057e-08 | 1.0939e-10<br>8.3698e-08 |
| 512 | 5.5025e-10<br>1.6429e-06 | 2.9709e-10<br>1.6119e-07 | 4.6540e-10<br>7.4623e-07 |
| 1024 | 2.1756e-08<br>3.1479e-04 | 9.3919e-10<br>5.6260e-07 | 1.5819e-08<br>8.0374e-05 |

Table 1: Discrete Wigner Transform average absolute (first row) and relative (second row) errors, as obtained on the Intel 2.4 GHz Xeon. The errors for the case $B = 1024$ are the average of 1,000, and not 100,000, iterations. As symmetry of the $d$-function would encourage one to predict, errors for the case $M = 0,\ M' = B/2$ were, for all practical purposes, similar to the case $M = B/2,\ M' = 0$.

| Bandwidth | 16 | 32 | 64 | 128 | 256 | 512 | 1024 |
|---|---|---|---|---|---|---|---|
| Forward DWT | 2.3000e-06 | 4.8000e-06 | 2.0900e-05 | 9.6200e-05 | 4.3070e-04 | 1.7334e-03 | 6.7300e-03 |
| Inverse DWT | 1.1000e-06 | 5.2000e-06 | 2.2500e-05 | 9.1200e-05 | 4.2550e-04 | 1.6697e-03 | 6.7200e-03 |

Table 2: Average CPU runtimes (in seconds) of the forward and inverse $M = 0,\ M' = 0$ DWT on the Xeon.

order of $10^{-5}$. However, as will be detailed in Section 5.2, when results of the full transform are presented and discussed, practicalities render any causes of concern at these larger bandwidths unnecessary.

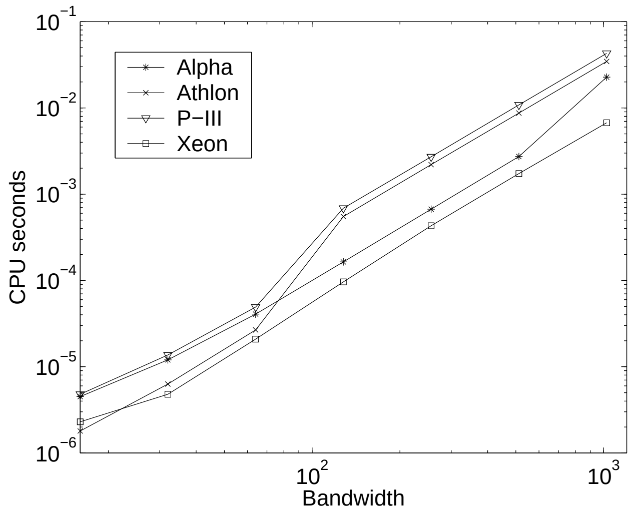
In Figure 3, we show the average CPU runtime of the forward DWT on four different platforms, at a number of different bandwidths $B$. The Wigner functions necessary for the transform were precomputed in advance. Runtimes for the inverse transform were comparable to those of the forward transform. Some numbers indicating this are shown in Table 2.

Precomputing itself does not take long. On the Xeon, for $B = 1024$, it takes roughly 0.04 seconds to compute the Wigners necessary for a forward transform. However, in our particular implementation, precomputing for the inverse transform takes about 4 times as long. The reason? To obtain the sampled Wigners necessary for the inverse transform, we are simply transposing the matrix of sampled Wigners needed for the forward transform. This takes time. We will see later how matrix transposition becomes a real issue in doing a Fourier transform over the full group.

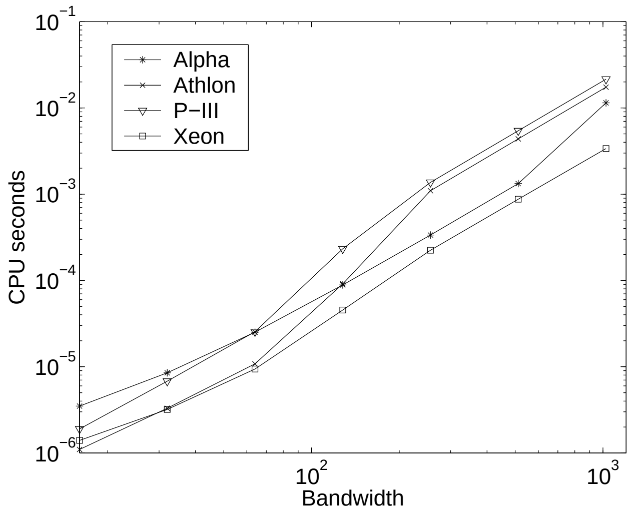
We find it interesting, from a cultural standpoint, that even on the older 'slow' 800 MHz machine, the DWT is still quite fast. E.g. While the 800 MHz is more than twice as slow as the Xeon, at bandwidth = 64, it still takes the 800 Mhz only about $10^{-5}$ seconds to do a DWT. Of course, this is just for a single DWT. Speed becomes a more pertinent issue when considering the full transform.

Now the astute reader may ask why the naive method was used in computing the Wigner-$d$ coefficients. Since the $d$-functions are trigonometric polynomials, certainly methods such as those of Dilts [3] (also called "semi-naive" in [12]) could be used and the discrete sums evaluated in the "discrete cosine transform" domain.

Indeed, as is observed in [19], one could take advantage of the three-term recurrence the Wigner $d$-functions satisfy, and hence employ the techniques of Driscoll and Healy [4], Driscoll, Healy and Rockmore [5], or their more numerically stable variations [12]. The rationale for our "naive" decision will be given in following section.



(a) Order $M = 0$, $M' = 0$ DWT



(b) Order $M = B/2$, $M' = B/2$ DWT

Figure 3: Average CPU runtime for a Wigner Transform on four different platforms: the 833 MHz Alpha, the 1.66 GHz Athlon, the 800 MHz P-III, and the 2.4 GHz Xeon.

## 5.2 Nice 'n' $SOFFT$

The basic structure of the experiment described in Figure 2 was preserved when doing a Fourier transform over the entire group.

1. Choose a bandwidth $B$.
2. Generate random Fourier coefficients, $\{\hat{f}^l_{MM'}\}$ over the appropriate ranges of $l$, $M$ and $M'$. There are $\frac{B}{3}\left(4\,B^2 - 1\right)$ many coefficients. Real and imaginary parts of the coefficients are values uniformly distributed between $-1$ and 1.
3. Compute the inverse $SOFFT$ of the above coefficients, resulting in values of the function $f(\alpha, \beta, \gamma)$ sampled on the equiangular $2B \times 2B \times 2B$ grid.
4. Compute the forward $SOFFT$ of the above sample values, resulting in a new set of Fourier coefficients, $\{\hat{g}^l_{MM'}\}$.
5. Compute the error
$$\max_{l,M,M'} \left\|\hat{f}^l_{MM'} - \hat{g}^l_{MM'}\right\|.$$
6. Repeat Steps 2 through 5 ten times.
7. Compute the average absolute and relative errors, and their standard deviations, over the ten trials.

Note that by going from spectral to spatial to spectral representations, we ensure that the we are truly dealing with band-limited functions $f \in L^2(SO(3))$. Furthermore, the Fourier coefficients in Step 2 are randomly generated without the constraint of ensuring that the resulting sample values should be strictly real, e.g. as is done in [15].

We implemented four fundamentally different versions of the full $SO(3)$ Fourier transform, with a variant or two thrown in. The full transform may be treated as two components: the "regular" FFT portion (eqns. 17-18), and the DWT portion (eqn. 19). With this in mind, here they are:

- **Plain**: The FFT portion is computed via the home-grown FFT available as part of SpharmonicKit. The DWT is evaluated for all legal $M$ and $M'$, computing the Wigner-$d$'s on the fly as needed.
- **Sym**: As for **Plain**, the FFT portion is computed via the home-grown FFT. However, we take advantage of the symmetries the Wigner-$d$ enjoy and get as much mileage out of each set as possible, e.g. the Wigner-$d$ functions needed for orders $M = 4$, $M' = 14$ may also be used for orders $M = 14$, $M' = 4$, and $M = 4$, $M' = -14$, and so on.
- **FFTW**: The DWT is evaluated as in **Sym**, but the FFT portion is computed via the routines in *FFTW*.
- **FFTW-PC**: Just like **FFTW**, except that *all* the Wigner-$d$ functions are precomputed in advance.

These different implementations all exhibit the same stability behavior. Their real differences become apparent when the question of CPU time is addressed.

**Stability** The results of the above experiment, when running **Sym** on the Xeon, are given in Table 3. Results on the other platforms are comparable. Quite clearly, at the bandwidths tested, the implementation is quite stable.

| Band-limit | 8 | 16 | 32 | 64 | 128 |
|---|---|---|---|---|---|
| Average Error | 1.6147e-12 | 5.7296e-12 | 1.5481e-11 | 1.1007e-10 | 7.0047e-09 |
| Std Dev | 1.4289e-13 | 7.4973e-13 | 1.4024e-12 | 1.4696e-10 | 1.8530e-08 |
| Relative Error | 1.4330e-11 | 1.0247e-10 | 8.9718e-10 | 5.3790e-09 | 4.1743e-07 |
| Std Dev | 1.4069e-11 | 4.9962e-11 | 6.6493e-10 | 4.5457e-09 | 8.7350e-07 |

Table 3: Average absolute and relative errors, over ten iterations, of **Sym**, when run on the Xeon.

Now, the reader will note that the full transform is run only through bandwidth $B = 128$. The reason for this is quite simple: we ran out of memory! Or, to be more precise, the computer ran out of memory. For each doubling of the bandwidth, memory requirements increase eight-fold. Suppose the C code uses the data type **double**, which is 8 bytes long. If we make the reasonable assumption that there should be sufficient RAM to contain both the input data, all $(2B)^3$ sample values, and the Fourier coefficients, all $\frac{B}{3}\left(4\,B^2 - 1\right)$ of them, then for problem size $B = 128$, we would need nearly 150 megabytes of RAM. For $B = 256$, requirements would blossom to over 1 gigabyte! And we are not even considering the memory necessary for temporary storage, e.g. matrix transposing. So this is the downside. The upside is that we cannot even an attempt to compute a Fourier transform on the full group at a bandwidth where the numerical stability of the *DWT* we know (by the results of Section 5.1) to be suspect.

**Running Times** The four different versions of the algorithm were run on all four different platforms. At the smaller bandwidths, $B = 8, 16$ and 32, in order to obtain as accurate a timing result as possible, the experiment described at the beginning of this section was slightly modified. First a random set of coefficients was generated. Then the stopwatch would be turned on, and the inverse transform would be performed (on that one set of coefficients) $X$-many times, then the stopwatch would be turned off. This would be repeated for the forward transform, using the random sample points just obtained. For $B = 8$, 16, $X = 1000$, and for $B = 32$, $X = 100$. The remaining bandwidths used $X = 10$.

Furthermore, the different machines had different amounts of RAM. We ran the tests for as large as possible on the particular machine. And it is always possible the amount of RAM affected performance. Still, we believe our results are fairly indicative of what might be expected on other platforms.

Finally, using *FFTW* involves first generating a "plan" before carrying out the actually Fourier transforms. A plan contains information on how to most efficiently perform the Fourier transform on that particular platform. The responsibility of determining how hard *FFTW* should try in finding the optimal transform is up to the user. In our tests, the *FFTW* level we used was `FFTW_MEASURE`. In addition to this, we freely admit the possibility that more efficient plans than our's is possible.

In Figure 4 we plot the average CPU runtime (in seconds) of the forward $SO(3)$ transform. In order to provide a little more quantitative“meat,” in Table 4 we give the average runtimes of the $SO(3)$ transform on the Xeon, both forward and inverse.

| Band-limit | 8 | 16 | 32 | 64 | 128 |
|---|---|---|---|---|---|
| **Plain** | 2.8800e-03 | 3.2120e-02 | 3.2610e-01 | 3.1180e+00 | 3.5070e+01 |
| | 2.9100e-03 | 3.1650e-02 | 2.9960e-01 | 3.1240e+00 | 3.5771e+01 |
| **Sym** | 1.3200e-03 | 1.5390e-02 | 1.6850e-01 | 1.4890e+00 | 1.6631e+01 |
| | 1.5300e-03 | 1.3230e-02 | 1.2510e-01 | 1.1970e+00 | 1.3544e+01 |
| **FFTW** | 5.2000e-04 | 6.7900e-03 | 7.2200e-02 | 7.2900e-01 | 8.2510e+00 |
| | 5.6000e-04 | 7.0700e-03 | 7.6300e-02 | 7.9700e-01 | 8.8970e+00 |
| **FFTW-PC** | 2.9000e-04 | 4.9600e-03 | 5.3900e-02 | 5.2200e-01 | |
| | 3.2000e-04 | 5.4400e-03 | 5.8500e-02 | 5.6000e-01 | |

Table 4: Average CPU runtime (in seconds) for the Forward (first row) and Inverse (second row) $SO(3)$ transform on the Xeon.

As is no surprise, **FFTW-PC** performed fastest on all four platforms, and **Plain** the slowest, although by how much might not be apparent in Figure 4. Therefore, in Figure 5 we plot the ratios of the runtimes for **Sym**, **FFTW**, and **FFTW-PC** versus **Plain**.

Figure 5 appears to confirm the following rules of thumb: taking advantage of the symmetries of the Wigner-$d$s speeds up the runtime by roughly a factor of 2 over **Plain**, and *FFTW* plus symmetries buys a factor of 4 speed-up. However, precomputing the Wigner-$d$s plus *FFTW* plus symmetries yields a noticeable, though not particularly dramatic, gain in performance only on the Xeon. A similar less-than-spectacular gain was observed when we assumed that the function was strictly real-valued. (This assumptions introduces an additional symmetry in the coefficients which can be used for computational advantage.

These results were somewhat disappointing, and they very much relate to the matter raised earlier, concerning why the semi-naive, or Driscoll-Healy based algorithms were not employed in evaluating the DWT. Both of these algorithms depend, in part, on precomputed data, such as the DCT coefficients of the Wigner-$d$ functions. Indeed, a semi-naive version of the DWT was coded up and tested in a full $SO(3)$ transform, but the runtimes were basically no different from those obtained with **FFTW-PC**, which themselves are not significantly different from **FFTW**. This made us skeptical that a Driscoll-Healy based algorithm would offer much more efficiency.

We are not certain why we are failing to see much improvement in the runtimes. However, we do believe we can identify some of the causes. (Which causes are the critical ones, that we do not know for certain.) One is memory, or lack thereof. The problem sizes we are able to run at may be too small to see the gains the asymptotic analyses predict. And this relates to the hidden constant in these analyses - the overhead. An integral part of the algorithm, as far as an actual implementation is concerned, is the matrix transpose. After all, not only are the arrays large, but one has to transpose them. This is a significant portion of the runtime. On the Xeon, more than 40% of the time spent doing a bandwidth $B = 64$ forward $SO(3)$ transform is spent transposing matrices. For $B = 32$, this figure approaches 75% ! This encourages us to believe that $B$ would have to be pretty large before an algorithm like semi-naive would present a big win over, say, **FFTW-PC**. Indeed, it could be argued that precomputing does not yield such significant improvement, either.

We tried many variations of the code, to reduce the number of matrix transpositions, e.g. having a different ordering the data prior to transforming; letting FFTW do the matrix transposing by means of a cleverly chosen plan. For us, the performance gained in doing these things was barely perceptible.

Now, we do not mean to imply that this precludes the possibility of, an optimal semi-naive or Driscoll-Healy like $SO(3)$ transform at bandwidths within our realm. Rather, given how our code is structured, we do not think it is possible. However, the reader should keep in mind that the “slow” algorithms we have tested are, in terms of absolute CPU time, still pretty fast, and hence may be of real use.

## 6 Applications

Given two functions on the sphere, $f$ and $h$, and the knowledge that $h$ is a rotated version of $f$, i.e. $f = \Lambda(g)h$ where $g \in SO(3)$, how does one find that $g$ ? Phrased another way, we have a pattern $f$, and we wish to identify its latitude, longitude, and orientation, on the sphere. This can be accomplished by correlating the two functions:

$$C(g) = \int_{S^2} f(\omega)\ \overline{\Lambda(g)h(\omega)}\ d\omega$$

and finding the $g$ that maximizes the above integral. This has a number of useful applications, in such diverse areas as molecular biology [16], and 3-D shape-matching [14].

Now, instead of undertaking the time-consuming task of evaluating $C(g)$ for all possible rotations, we may efficiently determine the maximum $g$ by means of the FFT on $SO(3)$. We develop this as follows. First of all, since $f$ and $h$ are both functions defined on the sphere, we may take their Fourier expansions:

- $f \in L^2(S^2)$, $f(\omega) = \sum_l \sum_{|m|\leq l} a_{lm} Y_{lm}(\omega)$.
- $h \in L^2(S^2)$, $h(\omega) = \sum_l \sum_{|m|\leq l} b_{lm} Y_{lm}(\omega)$.

We wish to find the $g \in SO(3)$ which maximizes

$$C(g) = \int_{S^2} f(\omega)\ \overline{\Lambda(g)h(\omega)}\ d\omega$$

Let us write things out in gory detail ...

$$\begin{aligned}
C(g) &= \int_{S^2} f(\omega)\ \overline{\Lambda(g)h(\omega)}\ d\omega \\
&= \int_{S^2} \left[\sum_l \sum_{|m|\leq l} a_{lm} Y_{lm}(\omega)\right] \overline{\left[\Lambda(g) \sum_{l'} \sum_{|m'|\leq l'} b_{l'm'} Y_{l'm'}(\omega)\right]}\ d\omega \\
&= \sum_l \sum_{|m|\leq l} \sum_{l'} \sum_{|m'|\leq l'} a_{lm}\ \overline{b_{l'm'}} \int_{S^2} Y_{lm}(\omega)\ \overline{\Lambda(g) Y_{l'm'}(\omega)}\ d\omega \qquad (30)
\end{aligned}$$

At this point, we can achieve some simplification. The integral in (30) equals 0, *unless* $l = l'$. Therefore, we can remove a summation:

$$C(g) = \sum_l \sum_{|m|\leq l} \sum_{|m'|\leq l} a_{lm}\ \overline{b_{lm'}} \int_{S^2} Y_{lm}(\omega)\ \overline{\Lambda(g) Y_{lm'}(\omega)}\ d\omega. \qquad (31)$$

Recall that under rotation, a spherical harmonic of degree $l$ is transformed into a linear combination of spherical harmonics of the same degree. In terms of the Wigner-$D$s, we can express this as:

$$\Lambda(g) Y_{lm}(\omega) = \sum_{|k|\leq l} Y_{lk}(\omega)\ D^{(l)}_{km}(g). \qquad (32)$$

Therefore, we may write (31) as:

$$\begin{aligned}
C(g) &= \sum_l \sum_{|m|\leq l} \sum_{|m'|\leq l} a_{lm}\ \overline{b_{lm'}} \int_{S^2} Y_{lm}(\omega)\ \overline{\sum_{|k|\leq l} Y_{lk}(\omega)\ D^{(l)}_{km'}(g)}\ d\omega \\
&= \sum_l \sum_{|m|\leq l} \sum_{|m'|\leq l} \sum_{|k|\leq l} a_{lm}\ \overline{b_{lm'}}\ \overline{D^{(l)}_{km'}(g)} \int_{S^2} Y_{lm}(\omega)\ \overline{Y_{lk}(\omega)}\ d\omega \qquad (33)
\end{aligned}$$

We are nearly there. Note that the integral in (33) is equal to 0 *unless* $k = m$. This being the case, and using two of the many symmetries the Wigner-$D$ functions satisfy, we can zap one of the summations:

$$\begin{aligned} C(g) &= \sum_{l} \sum_{|m|\leq l} \sum_{|m'|\leq l} a_{lm} \, \overline{b_{lm'}} \, \overline{D^{(l)}_{mm'}(g)} \\ &= \sum_{l} \sum_{|m|\leq l} \sum_{|m'|\leq l} a_{lm} \, \overline{b_{lm'}} \, (-1)^{m'-m} D^{(l)}_{-m-m'}(g) \\ &= \sum_{l} \sum_{|m|\leq l} \sum_{|m'|\leq l} a_{l-m} \, \overline{b_{l-m'}} \, (-1)^{m-m'} D^{(l)}_{mm'}(g) \qquad (34) \end{aligned}$$

That's it. That's the recipe.

So, if $f$ and $h$ are band-limited, combining the Fourier coefficients of $f$ and $h$ thusly and then taking the inverse $SO(3)$-Fourier transform of the result, we can efficiently evaluate the correlation $C(g)$ of $f$ and $h$, at a whole slew of $g$'s. Then we can easily find which rotation $g$ maximizes $C(g)$. In other words, if we knew in advance that $h$ was some rotated copy of $f$, we would know which $g$ satisfies $f = \Lambda(g)h$.

In the next section, we will show some examples of correlating via the technique just described.

## 7 An Example: Correlation

In order to investigate the viability of pattern-matching on the sphere via FFTs on $SO(3)$, we applied the technique described in the previous section on a variety of images. Those results, which are presented in this section, are quite promising.

In Figure 6, we show the signal, pattern, the difference between the signal and aligned pattern. The difference image is shown on the "unrolled" sphere, with the north pole being the top horizontal line, and the south pole the bottom. The bandwidth is $B = 128$. The total CPU running time, from computing the spherical coefficients of both signals through finding the optimal rotation, on the Xeon, was about 10 seconds. We used **FFTW** for the inverse $SO(3)$ Fourier transform, for this and all examples within this section.

The original data was a $256 \times 256$ square image, i.e. sampled in the plane. We simply interpreted the samples as instead living on the equiangular $256 \times 256$ grid on the sphere. Using this grid means we are treating the signal as a bandwidth $B = 128$ signal. We then took the forward and inverse $S^2$ Fourier transform, resulting in samples of a truly bandlimited signal. To rotate, we "massaged" the $S^2$ Fourier coefficients with the Wigner-$D$ functions, as prescribed by Eqn. 5.

Also in Figure 6, we plot the values of the parameters $\alpha$, $\beta$, and $\gamma$ which yield the maximum value of $C(g)$, as a function of the maximum degree of $Y_l^m$s used in the correlation. The plot is only through degree $l = 9$, because by that point, we know the three angles as well as we are going to. Indeed, in this case, as is apparent, correlating using only up to the degree 2 spherical harmonics is sufficient to yield the final answer. Higher degree harmonics do no improve the results. (Later examples will show this is not always the case. Higher degree harmonics are at times necessary.) Qualitatively, we can view this as indicating how small angular features we need to accurately find the pattern in the signal.

Before going further, we feel it appropriate to clarify what we mean by "maximum degree of $Y_l^m$s." A band-limited function $f \in L^2(S^2)$ has the following expansion:

$$f(\omega) = \sum_{l=0}^{B-1} \sum_{|m|\leq l} \hat{f}_l^m \, Y_l^m(\omega).$$

If we say that the maximum degree of $Y_l^m$ used was, say, 5, then we are setting equal to 0 all those coefficients $\hat{f}_l^m$ where $l \geq 6$. In the correlation routine, this zeroing is done *prior* to using the coefficients to construct Eqn. 34, the Fourier expansion of $C(g)$, and hence prior to taking its inverse $SO(3)$ Fourier transform. Note that it is still a bandwidth $B$ inverse transform being performed. Doing this can be considered in one of two ways, as either correlating smoothed versions of the two spherical functions, or taking the inverse $SO(3)$ transform of a smoothed version of $C(g)$, e.g.

$$C(g) = \sum_{l=0}^{5} \sum_{|m|\leq l} \sum_{|m'|\leq l} a_{l-m} \ \overline{b_{l-m'}} \ (-1)^{m-m'} D^{(l)}_{mm'}(g),$$

which amounts to the same thing. It is not the case that one interpretation is superior to the other. They are just two different ways of looking at the same thing. Different strokes for different folks.

Before going on to the next example, there is one more point we wish to make. The average relative error between the signal and aligned pattern is 0.0212. This may seem unacceptably large to some, but bear in mind what we are doing. We wish to find the rotation $g \in SO(3)$ which maximizes $C(g)$. This is done under the assumption that there is, in fact, some $g \in SO(3)$ such that $f(\omega) = \Lambda(g)h(\omega)$. However, the $g$s we are trying are points on a particular $2B \times 2B \times 2B$ grid. The "true" $g$, where equality holds, may not lie on this grid. In this case, the $g$ we find is, in some sense, the "next best $g$." That is the case in this example.

In another experiment we performed (with the same image as in this example), the true angles, showing all the non-zero decimal places, were

$$\begin{aligned} \alpha &= 1.006291 \\ \beta &= 3.000466 \\ \gamma &= 2.012582. \end{aligned}$$

The angles we found, which lie *on the grid*, were

$$\begin{aligned} \alpha &= 1.006291396852981 \\ \beta &= 3.000466421104314 \\ \gamma &= 2.012582793705961. \end{aligned}$$

The average relative error in this case, between the aligned pattern and signal, was $2.5259 \times 10^{-6}$. So the error descreases, the closer the true $g$ lies on the grid.

This also results in the following, but completely reasonable, situation. Correlating a signal with itself (i.e. the pattern equals the signal - **no** rotations are involved), will **not** yield $\alpha = \beta = \gamma = 0$, but rather $\beta = \pi/(4B)$, where $B$ is the bandwidth, and $\alpha$ and $\gamma$ such that $\alpha + \gamma = 2\pi$. At least, this is what our code gives us! The "behavior" of the $\alpha$ and $\gamma$ might seem reasonable, but what about the $\beta$? Recall where we are sampling: $\beta_j = \pi(2j + 1)/(4B)$. And when $j = 0$, we have $\beta = \pi/(4B)$. Bingo.

In our second example, we stereographically projected MR images onto the sphere, placing samples on the appropriate equiangular grid. The interpolation was done using *Mathematica*. The bandwidth was $B = 128$. Results are shown in Figure 7. The reader should compare these results with Figure 6, noting the minimum degree $Y_l^m$ required in order to achieve the optimal correlation. It at least indicates that sometimes one has to go higher than degree 2 spherical harmonics.

Up to this point, the bandwidth of the inverse $SO(3)$ Fourier transform was identical to the bandwidth of the spherical functions we were correlating. However, wanting to do a larger bandwidth $B$ correlation, we quickly run into the problem of RAM (at least for the routines in our test suite). Around 4.5 GBs of ram (ouch) is required for a $B = 256$ correlation versus around 560 MB for for a $B = 128$ correlation.

This motivated us to do the next best thing. While the spherical functions we want to correlate may have bandwidth $B = 256$, there is no reason why the inverse $SO(3)$ Fourier transform needs to be run at that same bandwidth.

Figure 8 shows the results of our efforts. As with the previous experiment, the data was sterographically projected onto the sphere. The original spherical functions were of bandwidth 256, but the correlation function $C(g)$ was treated as having bandwidth 128. Granted, we are effectively correlating smoothed versions of the original functions, but visually, the alignment we achieve is quite satisfactory.

The results of Figure 8 show some minor "juggling," with regards to determining the rotation angles $\alpha$, $\beta$ and $\gamma$, as functions of the maximum degree $Y_l^m$ used. This is at least partially a result of the "true" rotation not being on the grid of rotations we are sampling.

To see what would happen otherwise, we did the following. The original signal, let us call it **S**, was of bandwidth 256. We rotated the signal by an amount that would be sampled *exactly* on $B = 128$ $SO(3)$ grid.

(So if any experiment is doomed to succeed, this is it.) Call this rotated signal **RS**. It is still of bandwidth 256. We then computed the optimal rotation angles $\alpha$, $\beta$ and $\gamma$, treating $C(g)$ as a $B = 128$ function. The angles we are looking for are to rotate **S** to match **RS**. Our results are shown in Figure 9. As can be seen, even though we are correlating with smoothed verions of **S** and **RS**, we recover the angles spot on right, and practically immediately, judging from the maximum degree of $Y_l^m$ required.

We also tried out correlating geophysical data. The data we obtained was from the National Geophysical Data Center [22]. The particular dataset we used was the ETOPO5 5-minute gridded elevation dataset [7]. Our goal was to see how well we could find a particular coastline somewhere in the world.

To place the data on the sphere, we proceeded as follows. We took the original, 5-minute gridded elevation data and, using *Mathematica*, made a function which interpolated those data points, and then evaluated that function on the $B = 128$ bandwidth grid, i.e. the grid the sampling theorem says where one should sample in order to do a Fourier transform of a function $f \in L^2(S^2)$. So we have a grid of size $256 \times 256$. Figure 10 shows the data both on the sphere and "unrolled."

To obtain the coastline we performed, admittedly very crudely, edge-detection on the "sent to the $B = 128$ grid" data, including manually adding and removing the non-zero pixels until we got something we were satisfied with.

In Figure 11, we show our success in detecting the Atlantic coasts of the Americas. In the panel on the left, we show the world and where the coast belongs. The right panel shows the *rotated* world (this is our signal) and where correlation places the coastline. There is excellent agreement.

However, things are not so good when we consider a smaller coastline. Consider Figure 12. 'Oops' says it right! What might be happening? Any number of things, we believe. First of all, there is the fact that we were sloppy in defining the coast, and this would have contributed in the second reason. The coast is not truly bandlimited. (Neither is the original, non-rotated world, for that matter.) Indeed, given that the coast is basically a jagged line, we might say that it is highly non-bandlimited. When taking its spherical Fourier transform, we are projecting the coast onto the space spanned by spherical harmonics of maximum degree $L = 127$. This degree might not be large enough to capture all the subtle features of the coastline. As a result, the correlation misplaces the coast.

The reason why the results were good in Figure 11, we believe, is that there the coast is long enough that the subtleties being missed are not critical. The longer coast has sufficient "mass," if you will, to line up in only one spot, albeit, as seen in Figure 13, we might need the entire bandwidth in order to rotate it to the proper position on the globe.

On the bright side, the results of Figure 12 do establish the following corollary.

**Corollary 7.1** *A bandwidth B greater than 128 is required for there to be at least hope of recovering Pangea. I.e.*

$$\mathrm{Span}\,\{Y_l^m \mid 0 \leq l < 128,\, |m| \leq l\} \neq \textit{Pangea}.$$

## 8 Conclusions

We have developed and discussed an algorithm for the efficient computation of the Fourier transform of functions defined on $SO(3)$. Such efficient algorithms have uses in a wide vaiety of fields, such as searchable 3-D databases [25] and molecular biology [16].

This implementation differs from that of [24], as it allows easy use of efficient implementations of DWT algorithms, including those based on Dilts [3], and the more sophisticated Driscoll-Healy based methods [12], which depend in part on the three-term recurrence the Wigner-$d$ functions satisfy.

## References

[1] L. C. Biedenharn and J. D. Louck, *Angular Momentum in Quantum Mechanics*, Addison Wesley, Reading, 1981.

[2] R. A. Crowther, The Fast Rotation Function, *The Molecular Replacement Method*, M. G. Rossman (ed.), Gordon and Breach (1972), p. 173 - 178.

[3] G. A. Dilts, Computation of spherical harmonic expansion coefficients via FFTs, *Journal of Computational Physics*, **57**(3) (1985), 439-453.

[4] J. R. Driscoll and D. Healy, Computing Fourier transforms and convolutions on the 2-sphere. (extended abstract) in *Proc. 34$^{th}$ IEEE FOCS*, (1989) 344-349; *Adv. in Appl. Math.*, **15** (1994), 202-250.

[5] J. R. Driscoll, D. Healy and D. Rockmore, Fast discrete polynomial transforms with applications to data analysis for distance transitive graphs, *SIAM J. Comput.*, **26**(4) (1997), pp. 1066–1099.

[6] A. R. Edmonds, *Angular Momentum in Quantum Mechanics*, Princeton University Press, Princeton (1957).

[7] Data Announcement 88-MGG-02, "Digital relief of the Surface of the Earth." NOAA, National Geophysical Data Center, Boulder, Colorado, 1988.

[8] *FFTW* is a free collection of fast C routines for computing the Discrete Fourier Transform in one or more dimensions. It includes complex, real, symmetric, and parallel transforms, and can handle arbitrary array sizes efficiently. *FFTW* is available at `www.fftw.org`.

[9] J. E. Hansen (ed), *Spherical Near-field Antenna Measurements*, Peter Peregrinus Ltd., London, 1988.

[10] D. Healy and P. Kim, An empirical Bayes approach to directional data and efficient computation on the sphere, *Ann. Stat.*, **24**(1) (1996), 232-254.

[11] D. Healy, H. Hendriks and P. Kim, Spherical deconvolution with application to geometric quality assurance, *Technical Report, Department of Mathematics and Computer Science, Dartmouth College*, (1993).

[12] D. Healy, D. Rockmore, P. Kostelec and S. Moore, FFTs for the 2-Sphere - Improvements and Variations, *Journal of Fourier Analysis and Applications* **9**(4) (2003), pp. 341–385.

[13] S. Helgason, *Groups and Geometric Analysis*, Academic Press, New York (1984).

[14] M. Kazhdan, T. Funkhouser, and S. Rusinkiewicz, Rotation Invariant Spherical Harmonic Representation of 3D Shape Descriptors, *Eurographics Symposium in Geometry Processing*, L. Kobbelt, P. Schröder, H. Hoppe (eds), (2003).

[15] P. J. Kostelec, D. K. Maslen, D. M. Healy, Jr. and D. N. Rockmore, Computational Harmonic Analysis for Tensor Fields on the Two-sphere, *Journal of Computational Physics*, **162** (2000), 514-535.

[16] J. A. Kovacs and W. Wriggers, Fast rotational matching, *Acta Crystallogr D Biol Crystallogr.*, **58**(8) (2002), 1282-1286.

[17] A. B. Kyatkin and G. S. Chirikjian, Algorithms for Fast Convolutions on Motion Groups, *Applied and Computational Harmonic Analysis*, **9** (2000), 220-241.

[18] D. Maslen, *Fast Transforms and Sampling for Compact Groups*, Ph.D. Thesis, Department of Mathematics, Harvard University, 1993.

[19] D. Maslen and D. Rockmore, Generalized FFTs, in *Proceedings of the DIMACS Workshop on Groups and Computation, June 7-10, 1995*, L. Finkelstein and W. Kantor (eds.) (1997), 183-237.

[20] D. Maslen and D. Rockmore, "Separation of Variables and the Computation of Fourier Transforms on Finite Groups, I", *Journal of the American Math Society*, **10**(1), (1997), 169-214.

[21] A. Messiah, *Quantum Mechanics, volume II*, John Wiley and Sons, New York (1958).

[22] The National Geophysical Data Center (NGDC), located in Boulder, Colorado, is a part of the US Department of Commerce (USDOC), National Oceanic & Atmospheric Administration (NOAA), National Environmental Satellite, Data and Information Service (NESDIS). They are one of three NOAA National Data Centers. The url is `www.ngdc.noaa.gov`.

[23] A. F. Nikiforov, S. K. Suslov, and V. B. Uvarov, *Classical Orthogonal Polynomials of a Discrete Variable*, Springer Series in Computational Physics, Springer-Verlag, Berlin, 1991.

[24] T. Risbo, Fourier transform summation of Legendre series and D-functions, *Journal of Geodesy*, **70** (1996), p. 383 - 396.

[25] The Princeton Shape Retrieval and Analysis Group, `www.cs.princeton.edu/gfx/proj/shape/`, whose goal is to "...investigate issues in shape-based retrieval and analysis of 3D models," has developed a 3D search engine as part of their work. The url is `shape.cs.princeton.edu`.

[26] *SpharmonicKit* is a freely available collection of C programs for doing Legendre and scalar spherical transforms. Developed at Dartmouth College by S. Moore, D. Healy, D. Rockmore and P. Kostelec, it is available at `www.cs.dartmouth.edu/~geelong/sphere/`.

[27] The *SOFT* Package is a freely available collection of C programs for doing Wigner-$d$ transforms, as well as forward and inverse Fourier transforms of functions defined on the Rotation Group, $SO(3)$. The package also includes example routines for correlating functions defined on $S^2$. The package is available at `www.cs.dartmouth.edu/~geelong/soft/`.

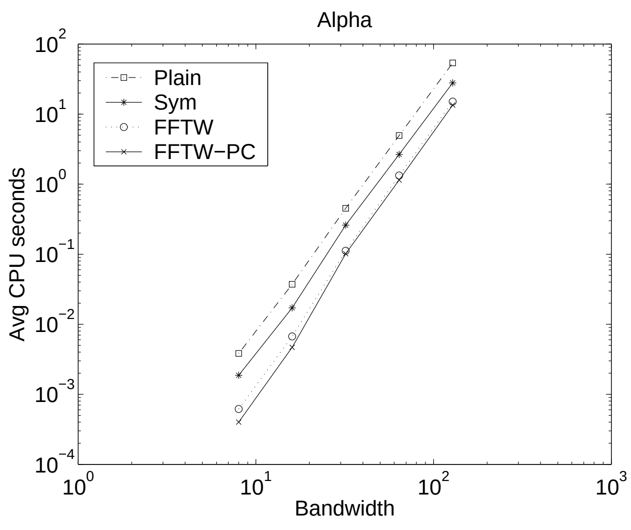
[28] W. F. Spotz and P. N. Swarztrauber, A Performance Comparison of Associated Legendre Projections, *Journal of Computational Physics*, **168** (2001), 339-355.

[29] S. Sternberg, *Group Theory and Physics*, Cambridge University Press, Cambridge, 1994.

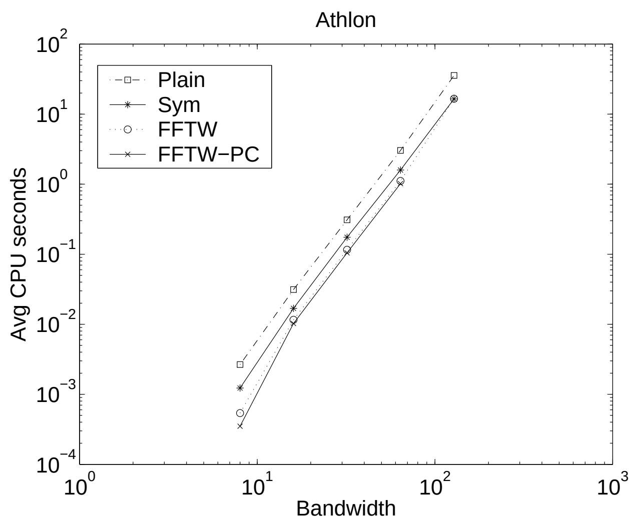
[30] D. A. Varshalovich, A. N. Moskalev and V. K. Khersonskii, *Quantum Theory of Angular Momentum*, World Scientific Publishing, Singapore, 1988.

[31] N. J. Vilenkin, *Special Functions and the Theory of Group Representations*, Translations of Mathematical Monographs, **22**, American Mathematical Society, Providence RI, 1968.

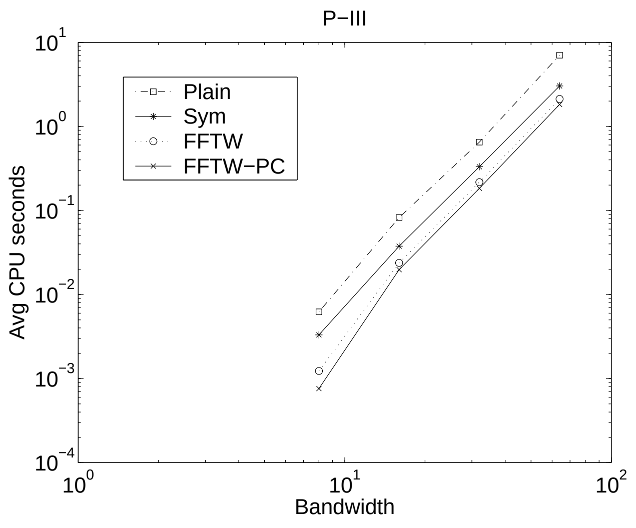
[32] E. P. Wigner, *Group Theory and its Application to the Quantum Mechanics of Atomic Spectra*, Academic Press, New York (1959).



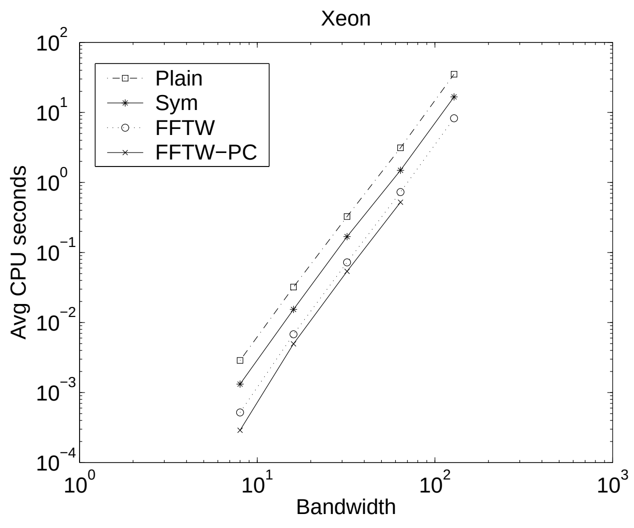
(a) Alpha



(b) Athlon

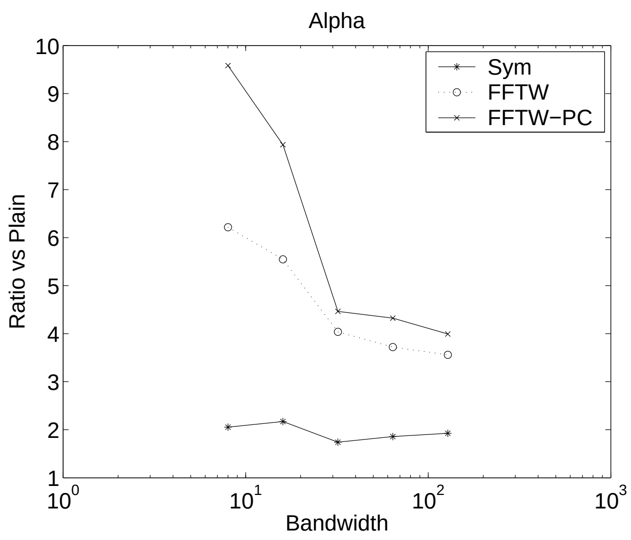


(c) P-III

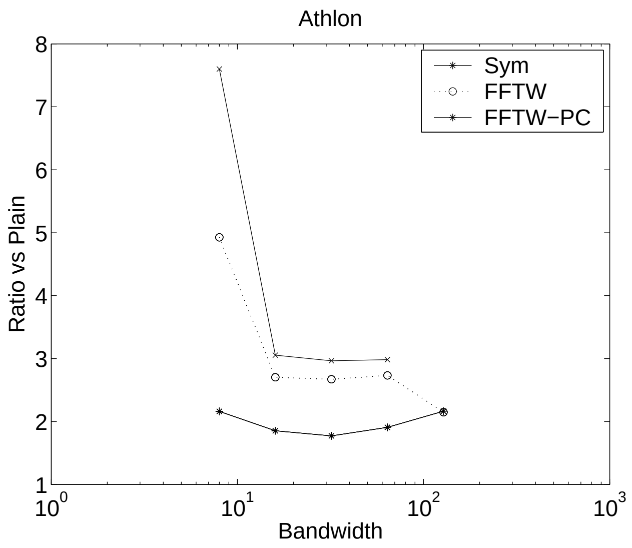


(d) Xeon

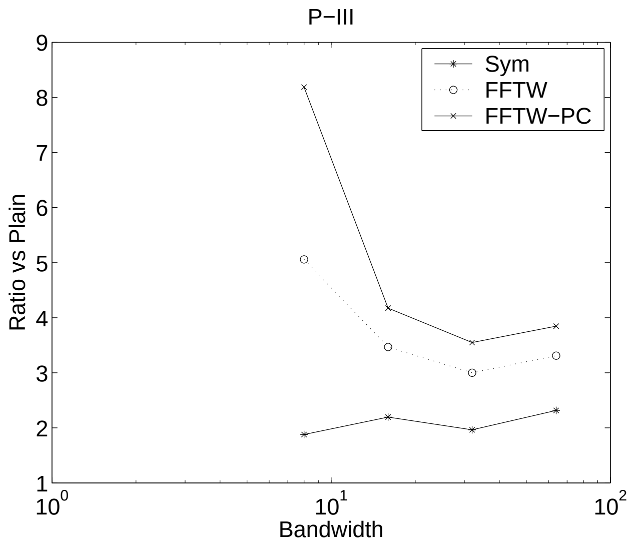
Figure 4: Average CPU runtime for the different Forward $SO(3)$ transforms.



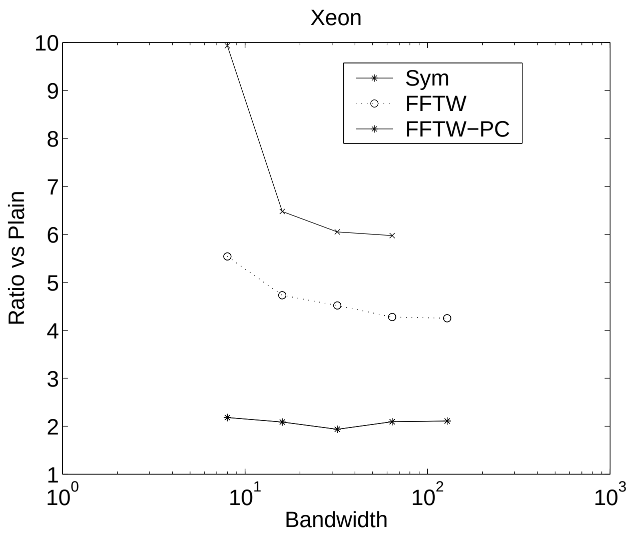
(a) Alpha



(b) Athlon

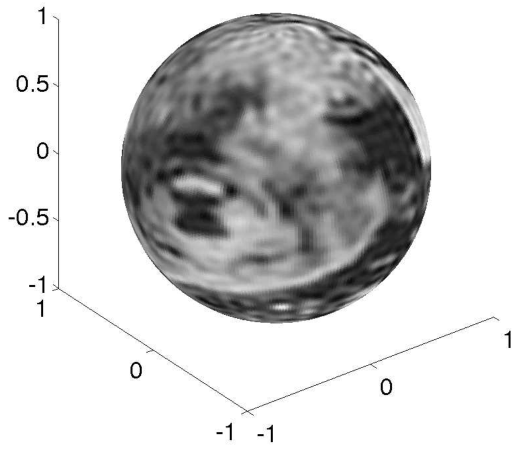


(c) P-III

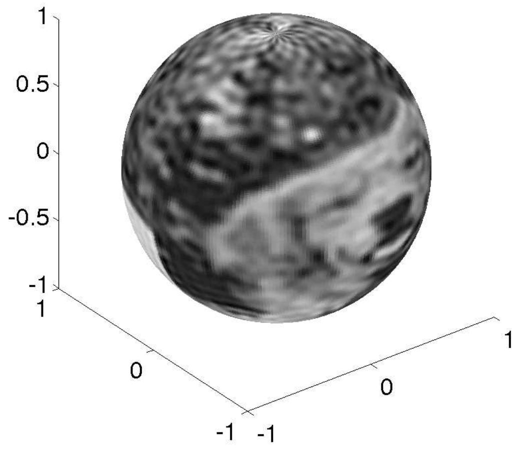


(d) Xeon

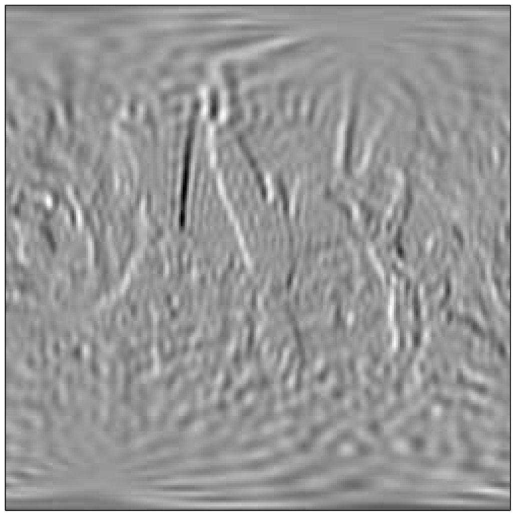
Figure 5: Ratios of average CPU runtime for the different Forward $SO(3)$ transforms.



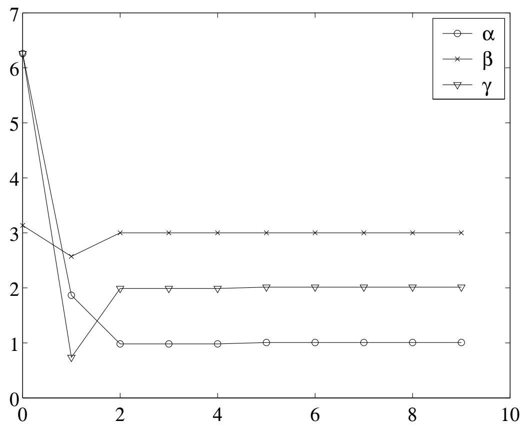
(a) Signal



(b) Pattern

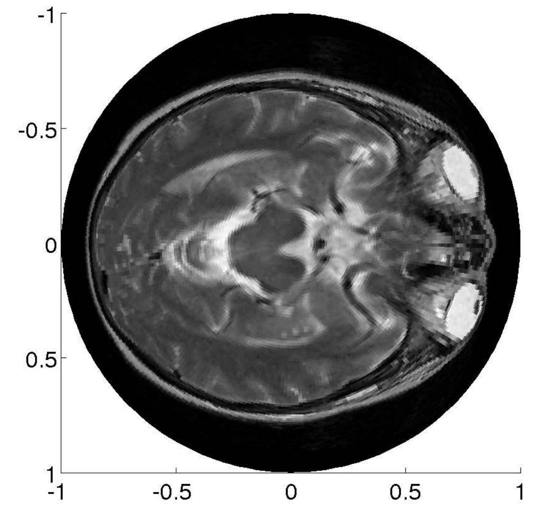


(c) Difference

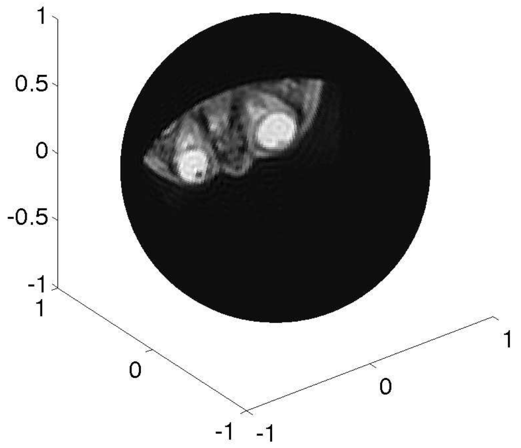


(d) Angles

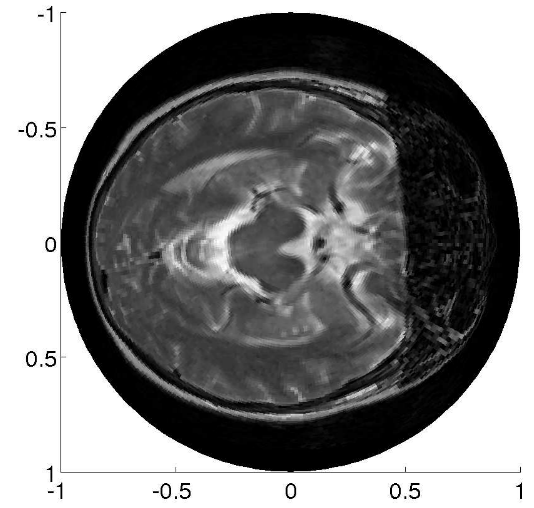
Figure 6: Rotating the pattern to match the signal. The bandwidth is $B = 128$. In the lower left we show the difference between the signal and aligned pattern on the unrolled sphere (the South Pole is at the bottom, the North Pole at the top). The lower right panel shows the rotation parameters $\alpha$, $\beta$, and $\gamma$ which yield the maximum value of $C(g)$, as a function of the maximum degree of $Y_l^m$s used in the correlation.



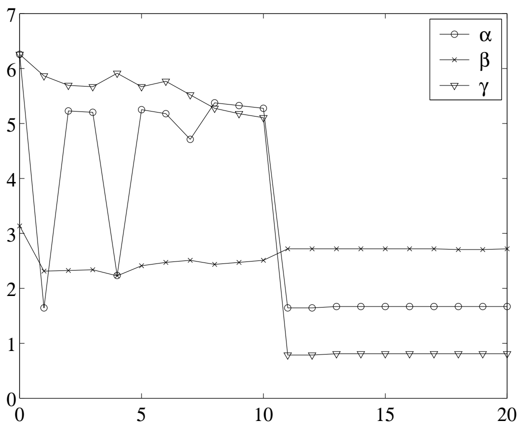
(a) Signal



(b) Pattern

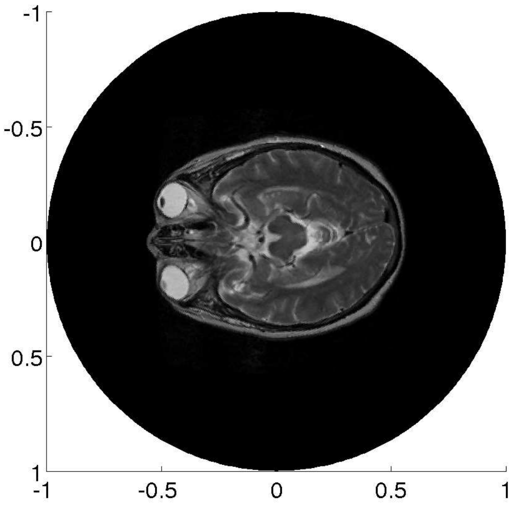


(c) Difference

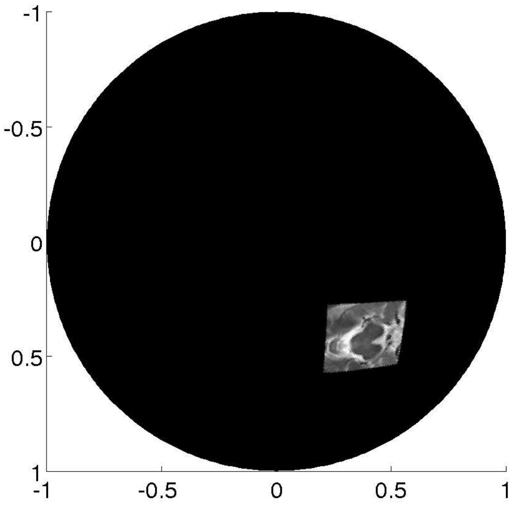


(d) Angles

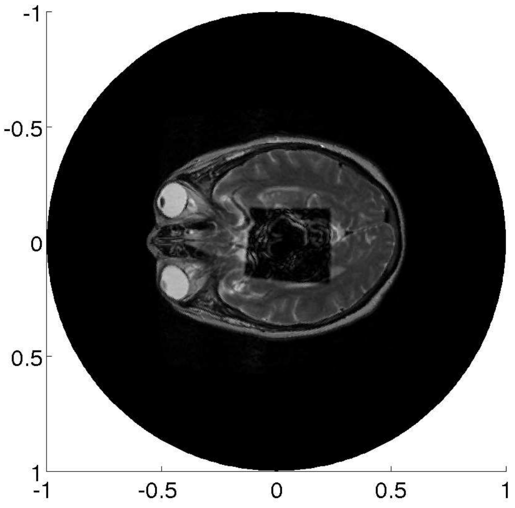
Figure 7: Rotating the pattern to match the signal. The signal is shown from *beneath* the sphere, i.e. from our vantage point, the South Pole is at the center of the image. The bandwidth is $B = 128$. The pattern lives primarily in the Northern hemisphere. The difference image is rendered on the sphere where, as with signal, the South Pole at the center. The lower right panel shows the rotation parameters $\alpha$, $\beta$, and $\gamma$ which yield the maximum value of $C(g)$, as a function of the maximum degree of $Y_l^m$s used in the correlation.



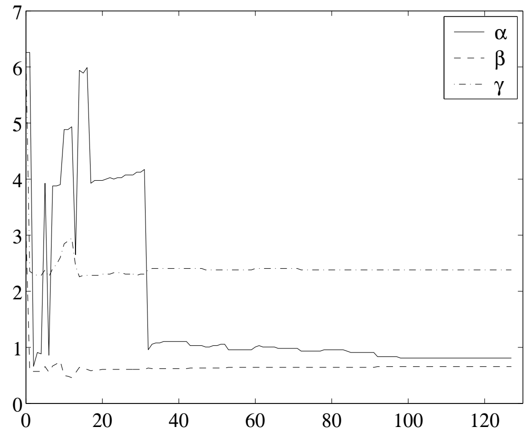
(a) Signal



(b) Pattern

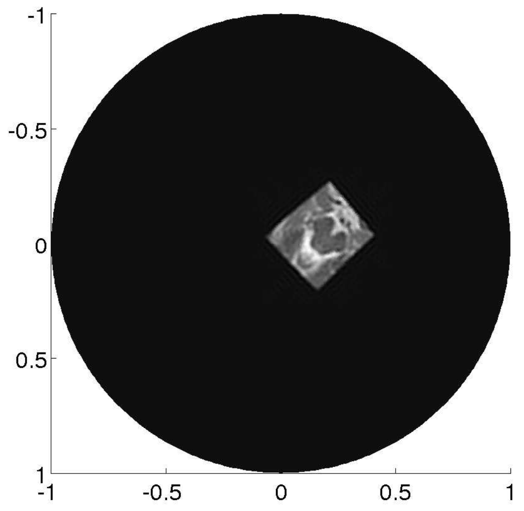


(c) Difference

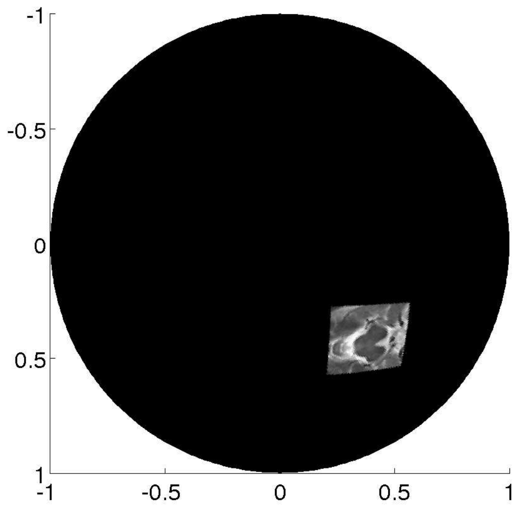


(d) Angles

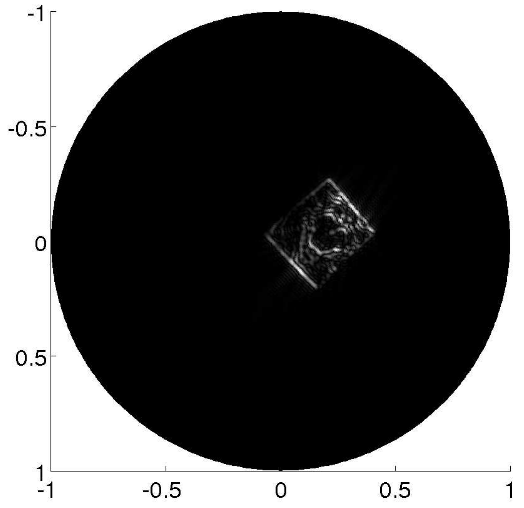
Figure 8: Rotating the pattern to match the signal. The signal and pattern are shown from *beneath* the sphere, i.e. from our vantage point, the South Pole is at the center of the image. The difference image, the absolute value of the signal - the rotated pattern, is shown from the same vantage point. The bandwidth of the signal and pattern are both $B = 256$, but the bandwidth of the inverse $SO(3)$ transform was $B = 128$. The lower right panel shows the rotation parameters $\alpha$, $\beta$, and $\gamma$ which yield the maximum value of $C(g)$, as a function of the maximum degree of $Y_l^m$s used in the correlation.



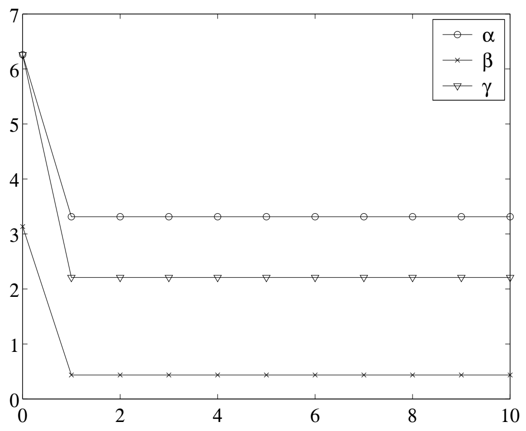
(a) Signal



(b) Pattern



(c) Difference



(d) Angles

Figure 9: Rotating the pattern to match the signal. The signal and pattern are both shown from *beneath* the sphere, i.e. from our vantage point, the South Pole is at the center of the image. The difference image, the absolute value of the signal - the rotated pattern, is shown from the same vantage point, with the South Pole at the center. The bandwidth of the signal and pattern are both $B = 256$, but the bandwidth of the inverse $SO(3)$ transform was $B = 128$. The difference image is somewhat misleading, in that the alignment looks worse than it really is. The average relative error of the difference image is on the order of $10^{-12}$. The lower right panel shows the rotation parameters $\alpha$, $\beta$, and $\gamma$ which yield the maximum value of $C(g)$, as a function of the maximum degree of $Y_l^m$s used in the correlation.

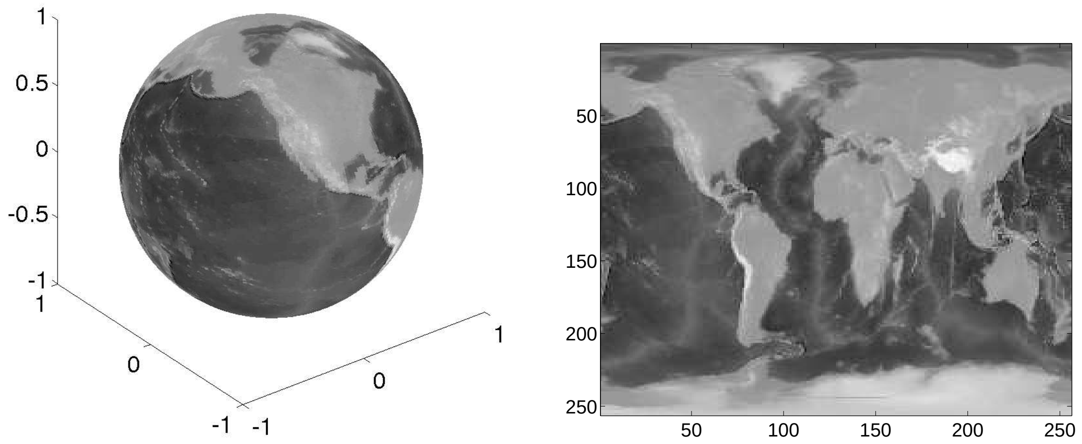


Figure 10: The world: On $S^2$ and unrolled.

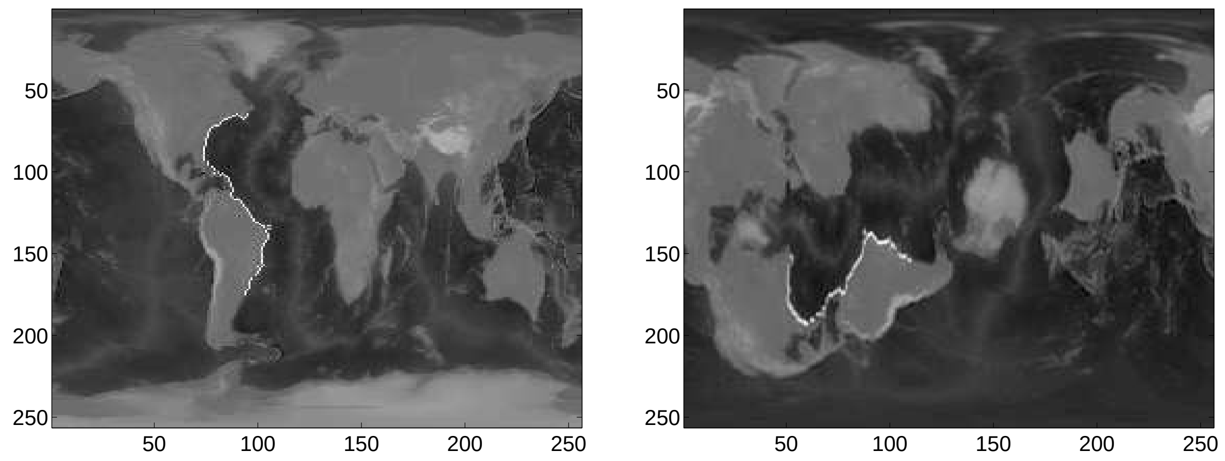


Figure 11: The left panel shows the world and the correct location of the coast. The right panel shows the rotated world (our signal) and where correlation places the coast.

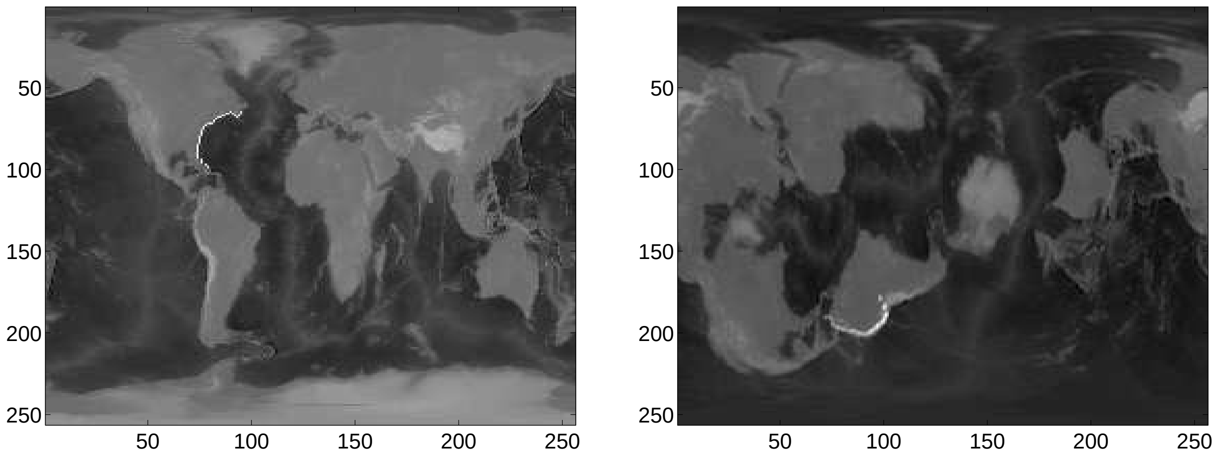


Figure 12: The left panel shows the world and the correct location of the coast. The right panel shows the rotated world (our signal) and where correlation places the coast. Oops!

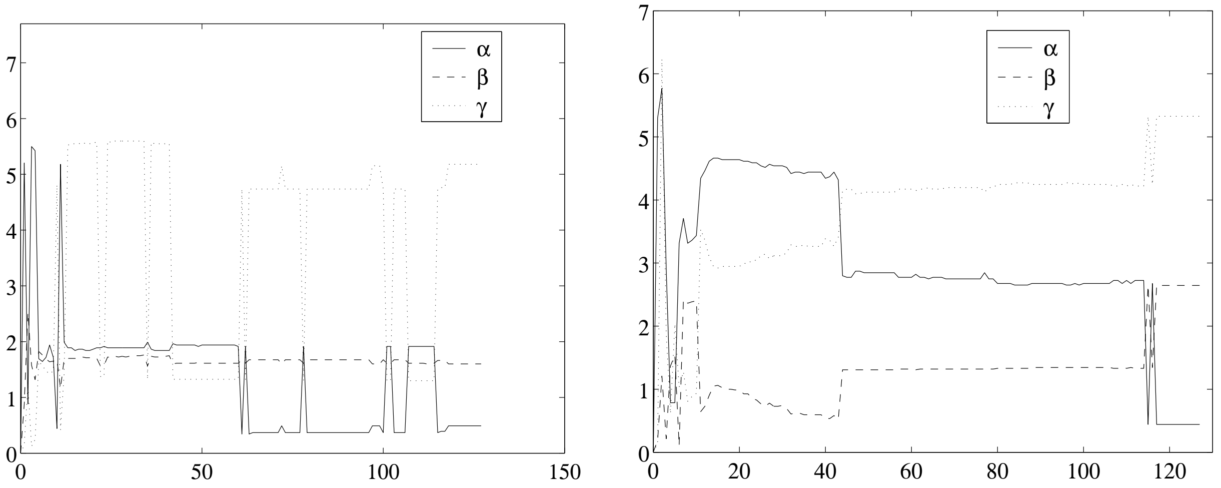


Figure 13: The Euler angles as a function of the maximum degree of $Y_l^m$ used in the correlation. The left panel corresponds to the successful alignment shown in Figure 11, and the right panel to the 'oops' result in Figure 12.