Estimation of Laplace Series Coefficients via 2-D Numerical Quadrature

Evan Voyles

November 25, 2022

1 Abstract

This paper provides an alternative approach to the computation of Laplace Series coefficients C_l^m and S_l^m with algorithmic complexity $O(Nl^2)$ in time and $O(Nl^2)$ in space. We make use of the fact that the spherical harmonics form a complete and orthogonal set to directly compute C_l^m as a 2-dimensional integral. We estimate this integral using numerical quadrature techniques and the resulting model is blazing fast compared to a least squares model running in $O(Nl^4)$. Harnessing the structure of the data sets, we perform many optimization to reduce the amount of data needed to be stored in memory and save time by precomputing values of the associated Legendre polynomials. We explore the scalability of this algorithm when implemented in parallel and we present a high-degree model of the ultra data set.

2 Preliminaries

Any sufficiently regular function defined on a sphere can be written

$$f(\theta,\phi) = \sum_{l=0}^{+\infty}\sum_{m=0}^{l} \bar{P}_l^m(\cos\theta)[C_l^m\cos m\phi + S_l^m \sin m\phi]. \tag{1}$$

Where C_l^m, S_l^m are *spherical harmonic* coefficients (also referred to as Laplace series coefficients) of degree l and order m; $\bar{P}_l^m$ are the fully normalized Associated Legendre Functions of degree l and order m; θ is the polar angle and ϕ is the azimuthal angle, as per ISO convention[1].

The project proposal suggests computing the coefficients C_l^m, S_l^m by setting up an overdetermined linear system of equations and finding the least squares solution. Unfortunately, this problem runs in $O(Nl_{max}^4)$ time and requires about $8N(l_{max}+1)^2$ bytes of memory. That is absurdly prohibitive. Furthermore, there are no *self-evident* parallelization techniques to speed up computation time. In this report we propose an alternative approach to computing the Laplace series coefficients C_l^m and S_l^m that runs in $O(Nl^2)$ time complexity with trivial parallelization models.

2.1 Laplace Series Coefficients

The coefficients C_l^m and S_l^m are defined as the orthogonal projection of $f(\theta,\phi)$ onto the basis functions Y_l^m, the spherical harmonics of degree l and order m. This definition is perfectly analagous to the fourier coefficients for 2π periodic functions. Written in terms of the normalized legendre polynomials, these projections are defined as:

$$C_l^m = \int_0^{2\pi}\int_0^{\pi} f(\theta,\phi)\bar{P}_l^m(\cos\theta)\cos(m\phi)\sin\theta d\theta d\phi$$
$$S_l^m = \int_0^{2\pi}\int_0^{\pi} f(\theta,\phi)\bar{P}_l^m(\cos\theta)\sin(m\phi)\sin\theta d\theta d\phi$$

We propose a solution to directly compute an approximation of the C_l^m and S_l^m using numerical integration.

[1] We have chosen the ISO standard coordinate system to ensure the integrity of future computations. For a discussion on transforming the (ϕ,λ) coordinates provided by NASA, see Appendix A

3 Data Sets

NASA data sets contain the altitude information for a spherical coordinate pair. The spherical coordinates can be represented as the discretization of a 2-d rectangle with width 2π and height π. NASA's data sets contain observations over a discretization of $[-\frac{\pi}{2}, \frac{\pi}{2}] \times [-\pi, \pi]$, represented as a $n_\theta \times n_\phi$ grid with equidistant points.

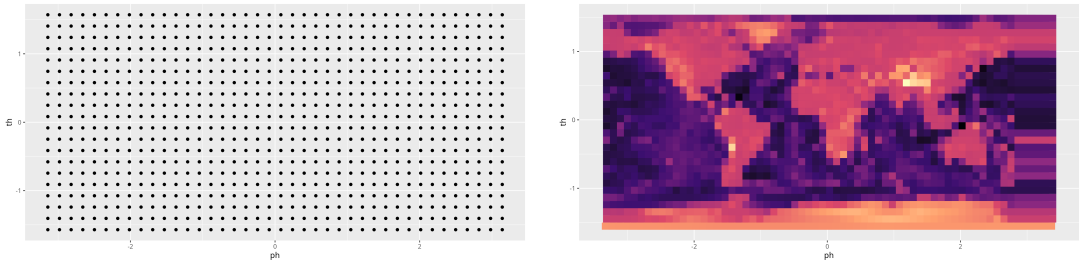

Figure 1: Discretization points (left) of a reduced data set with $n_\theta = 20$, $n_\phi = 40$. On the vertical axis we have $\theta \in [-\pi/2, \pi/2]$, or the latitude; On the horizontal axis we have $\phi \in [-\pi, \pi]$, the longitude.

Due to the linear spacing, we only need to know the number of points n_θ and n_ϕ to recreate the spherical coordinate pairs provided by the data sets, allowing us to decrease the running memory footprint of our program. In order study informative analytics about our model, we first enumerate the characteristics of each data set.

	n_θ	n_ϕ	N
small	180	360	64800
med	540	1080	583200
hi			
ultra			

$$\theta \in [0, \pi], \quad \phi \in [0, \pi 2]$$

We'll use these values later when we study the time it takes to compute our model.

3.1 Visualization

TODO: [ ] add 3-d maps

4 Mathematical Model

We define $\mathcal{M}_L$ a spherical model of degree L to be the set of coefficients

$$\{C_0^0, C_1^0, C_1^1, \ldots, C_L^L\} \cup \{S_0^0, S_1^0, SL^L, \ldots, S_L^L\}$$

that are used in conjonction with Figure (1) to provide an estimate of f, $\hat{f}$, defined as:

$$\hat{f}(\theta, \phi; L) = \sum_{l=0}^{L} \sum_{m=0}^{l} \bar{P}_l^m(\cos\theta)[C_l^m \cos m\phi + S_l^m \sin m\phi]. \tag{2}$$

Notice that the $+\infty$ in the outer summation has been replaced with the degree of the model, L. We will evaluate the efficacy of the model by studying the distance between f and $\hat{f}$ at the observations of our data set. Of notable interest will be the average absolute error (AAE) and the mean squared error (MSE):

$$\text{AAE}(l_{max}) = \frac{1}{N} \sum_{i=1}^{N} |\hat{f}_i - f_i|, \quad MSE(l_{max}) = \frac{1}{N} \sum_{i=1}^{N} (\hat{f}_i - f_i)^2 \tag{3}$$

where $\hat{f}_i$ denotes the estimated altitude evaluated at the ith data point and f_i is the true observed altitude of the ith data point.

4.1 Computation of Coefficients via Numerical Integration

We approximate the Laplace Series coefficients via a 2 dimensional analogue of the rectangle rule.

$$
\begin{aligned}
C_l^m &\approx \sum_{i=1}^{n_\phi}\sum_{j=1}^{n_\theta} f(\theta_j,\phi_i)P_l^m(\cos\theta_j)\cos(m\phi_i)\sin\theta_j\Delta\theta\Delta\phi \\
&\approx \Delta\theta\Delta\phi\sum_{i=1}^{n_\phi}\cos(m\phi_i)\sum_{j=1}^{n_\theta} f(\theta_j,\phi_i)P_l^m(\cos\theta_j)\sin\theta_j \\
S_l^m &\approx \sum_{i=1}^{n_\phi}\sum_{j=1}^{n_\theta} f(\theta_j,\phi_i)P_l^m(\cos\theta_j)\sin(m\phi_i)\sin\theta_j\Delta\theta\Delta\phi \\
&\approx \Delta\theta\Delta\phi\sum_{i=1}^{n_\phi}\sin(m\phi_i)\sum_{j=1}^{n_\theta} f(\theta_j,\phi_i)P_l^m(\cos\theta_j)\sin\theta_j
\end{aligned}
$$

This method computes *individual* Laplace series coefficients in $O(N)$ time! Thus, computing the coefficients for a Discrete Laplace Series (DLS) of order l runs in $O(Nl^2)$. More interestingly, each C_l^m or S_l^m can be computed **independently** of any other coefficient. Whereas solving the linear least squares problem requires computing every C_l^m and C_l^m up to a given L in a single operation, this method allows us the ability to compute any arbitrary C_l^m directly. Why do we care? If I alraedy have the first 500 C_l^m coefficients computed and stored in a text file, I can *directly* compute the next p coefficients in $O(pN)$ time. This method allows us to also compute ranges of C_l^m and S_l^m without messing around with bulky matrices taking up millions of bytes. As a consequence, there are simple yet powerful parallelization schemes that can be easily implemented to speed up computation and increase the performance of our models.

4.2 Comparison with Linear Squares

Let's start by exploring the small data set and benchmarking how long it takes to compute a model of degree l for $l \in \{0, 2, 4, \ldots, 20\}$ via a) an overdetermined system of linear equations b) numerical integration. Our aim in this section is to show that computing the coefficients via numerical integration provides numerous benefits to the overall methodology and allows us to compute higher degree models with lower error more efficiently.

4.2.1 Timing

We experimentally corroborate the theoretical hypothesis that the least squares computation is of time complexity $O(Nl^4)$ for a data set of size N and model degree l.

When we study how long it takes to compute a model of degree 20 (Figure 2) for the small data set we find that the least squares method takes, on average, a staggering 31.678 seconds to run. Coefficients computed by numerical integration, on the other hand, were generated with an average runtime of 0.656s across 5 trials. As we will see in the following analysis, a model of degree 20 is wholly incomplete to accurately represent a data set with 648000 points, and we are not satisfied with the runtime of the least squares method. And that's just for a small l. Running the algorithm in sequential for higher values of l would quickly become intractable due to its steep algorithmic complexity, especially magnified if we wanted to compute models for larger data set with millions of points (The medium data set has half a million) For a model of degree 20, numerical quadrature ran on average 50x faster than its least squares counterpart. As the degree of the model increases, so does the disparity between the two runtimes; at $l = 20$ our algorithm runs 50 times faster but at $l = 100$ our algorithm could be running 1250 times faster.

Clearly the proposed numerical integration model is faster. But how do two models compare when it comes to average squared error?

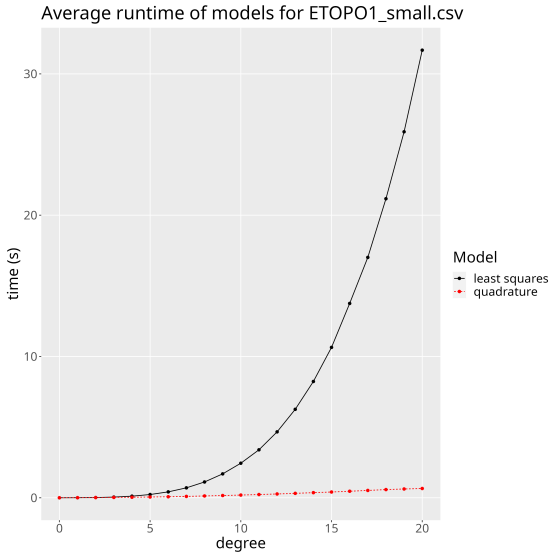

Figure 2: A timing comparison between least squares $\in O(Nl^4)$ and numerical quadrature $\in O(Nl^2)$ to compute the coefficients C_l^m and S_l^m for the small ETOPO1 data set. For each degree of the model Laplace coefficients were computed 5 times. Programs were run in sequential with 1 processor. TODO: Run a linear regression

4.2.2 Model Performance

In the least squares model, coefficients are carefully selected by minimizing the MSE of the prediction, which amounts to solving a linear system in $O(Nl^4)$ time. We are therefore, by construction, finding optimal coefficients that should miniimze the MSE. We compare how the average abs error, the AAE, varies with respect to the degree of the model.

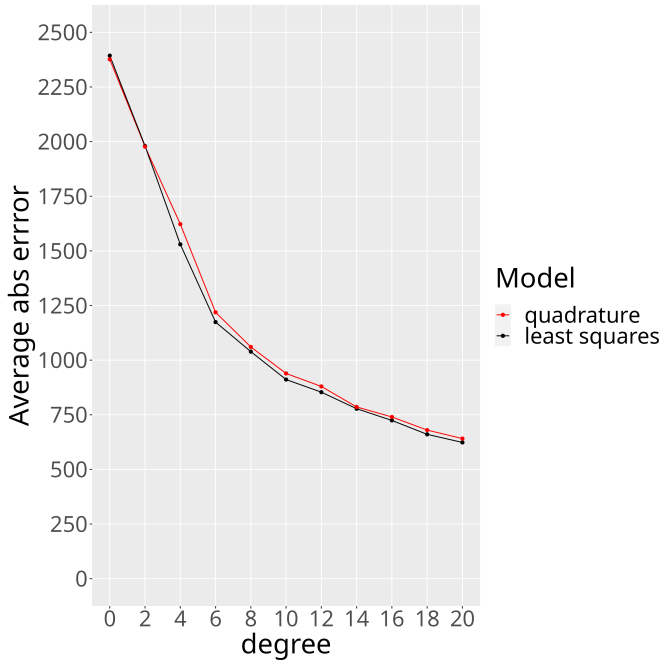

Figure 3: A Comparison between the average abs error between the least squares estimate and numerical quadrature estimate for the small ETOPO1 data set. Coefficients computed via least squares perform slightly better than their quadrature counter parts.

Perhaps surprisingly, we find that there is no significant penalty to using numerical quadrature to estimate a model's coefficients. In fact, computing the coefficients via quadrature allows

us to create higher degree models that, running in the same time as a low-order least squares model, will have far lower average absolute error values and furthermore the distributions of the error will be much tighter, exhibiting lower variance. In fact, as will be shown in the next section, we can run a degree 200 model using numerical integration in around 20 seconds whose average abs error is 148.76, compared to the degree 20 model whos AAE is 641.1.

4.2.3 Higher Degree Models

In this section we explore the distribution of error values $(\hat{f} - f)$ for models up to degree $l = 300$, computed for the small, medium, and high resolution data sets.

We start off by showing the evolution of the model's estimates as the degree l increases

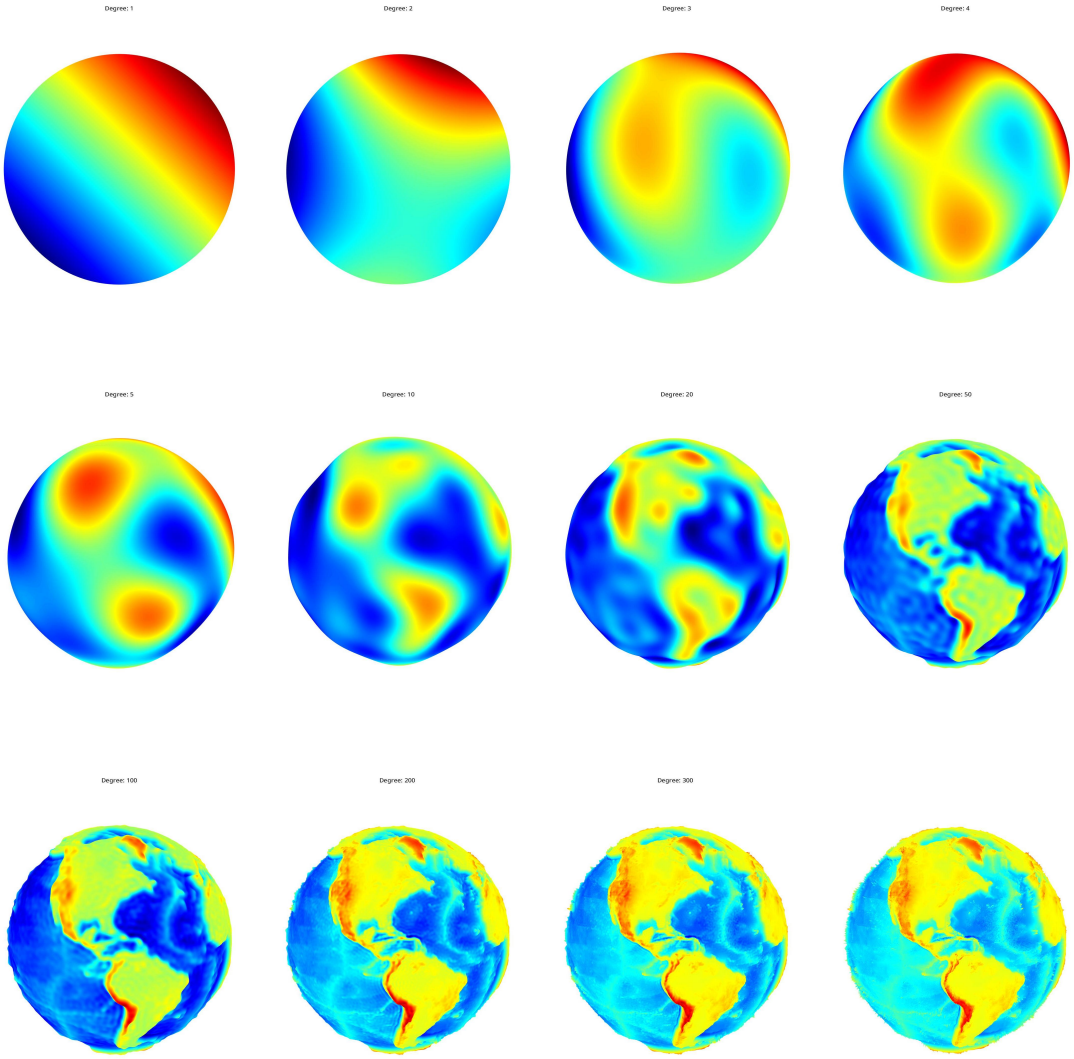

Figure 4: Altitude prediction of a model for varying degrees of a model. In the top row we have models of degree $l \in \{1, 2, 3, 4\}$, middle row $l \in \{5, 10, 20, 50\}$, and finally the bottom row $l \in \{100, 200, 300, +\infty\}$. The final visualization on the bottom right corresponds to the observed values of f.

Evidently, as we increase the degree of the model the difference between the predicted values $\hat{f}_i$ and the observations f_i will tend to shrink in magnitude. Let's take a look at the distributions of $r = \hat{f} - f$ for models of degree up to $l = 300$.

Recall to a simpler time where we computed models of degree $l = 20$. Refer to Figure 3 to verify that the average absolute error sits at around 600 for a model of the small data with

coefficients going up to $l = 20$. Now consider the following distributions of the errors of models built on the small data set with degrees up to 300.

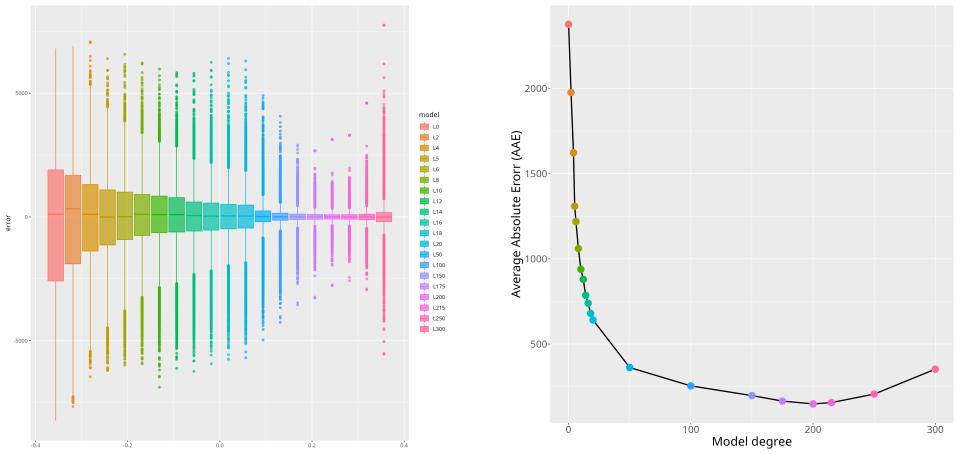

Figure 5: Boxplot of the residuals $r_i = \hat{f}_i - f_i$ (left) for models of degree up to $l = 300$ and the AAE associated (right) built on the small dataset.

We notice from these graphs that there seems to be a certain optimal degree of the model, afterwhich the AAE begins to increase. For the small data set, this optimum sits at just around 200, and we find that the best model for the small data set has an AAE of 148.7594

We conduct an analagous exploration with the medium data set, this time generating models of up to $l = 800$:

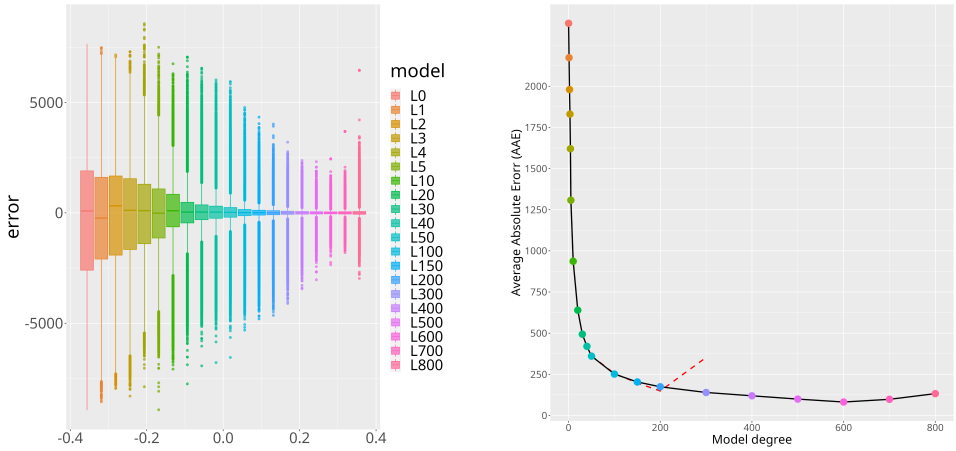

Figure 6: Boxplot of the residuals $r_i = \hat{f}_i - f_i$ (left) for models of degree up to $l = 800$ and the AAE associated (right) built on the medium dataset. The red dashed line shows the AAE(l) of models trained on the small dataset.

Just like with the small dataset, we find that the AAE(l) decreases as l gets bigger, but there is a certain point for the which the model starts to misbehave and the average absolute error, $\frac{1}{N}\sum_{n=1}^{N}|\hat{f}_i - f_i|$, begins to rise. We find that the medium data set has a near optimal model at $l = 600$, with AAE 81.77012. We have nearly halved the AAE by increasing the degree of our model by a factor of 3. Next, we plot the distributions of the residuals for varying l.

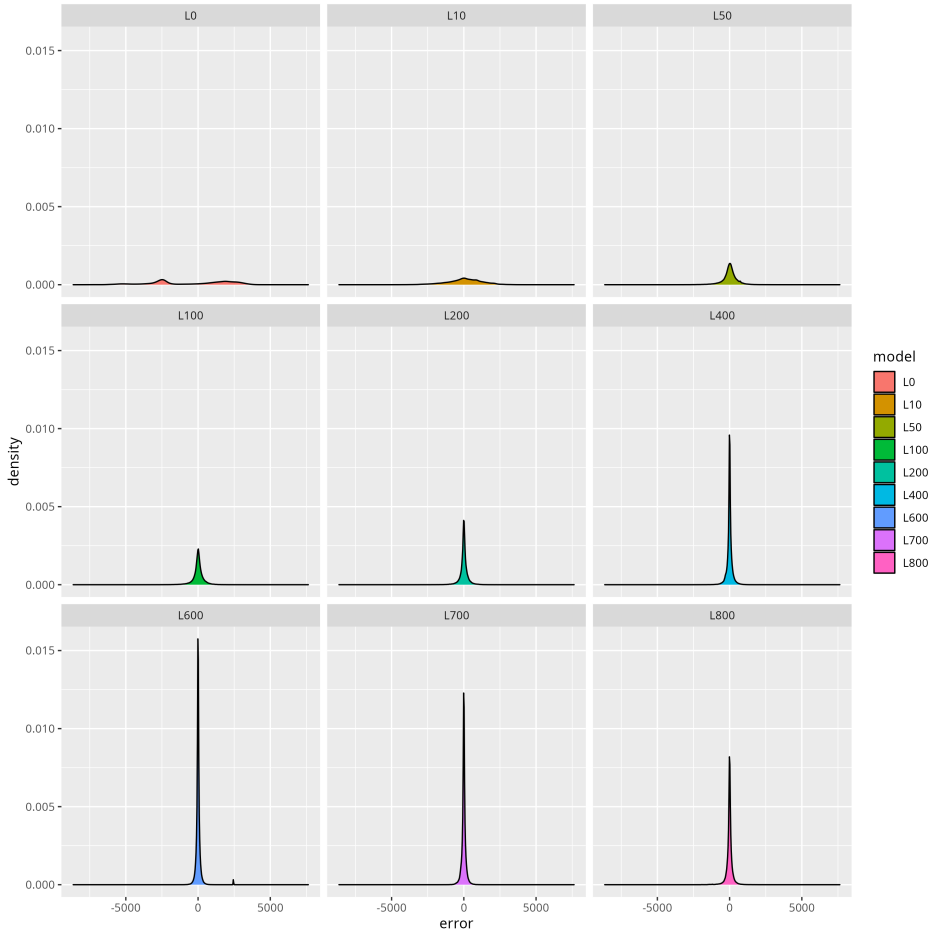

Figure 7: Kernel density estimates for the distribution of r_i. The model with the lowest variance occurs at $l = 600$.

We present the result that the distribution of the residuals approximately follows a cauchy distribution. This observation is important for the analysis of our models because it allows us to conclude that not only do higher degree models have a lower AAE, they also have residuals which are much more tightly packed than their lower model counterpart. For example, for a model of degree 600, we find that not only is the AES(600) = 81.77012, but also that 75% of residuals $|r_i|$ are less than 100. If we compare this with a lower degree model, take $l = 300$ for example, we observe that 83.7% of the residuals for a model of $l = 600$ have magnitude less than AES(300) = 139.704. To put this in perspective, take another look at Figure 4. 83.7% of the residuals are *more accurate* than the *average* prediction for the highest degree model of Figure 4. Conclusion: higher degree models have a lower AAE and a lower variance.

To conclude our analysis of higher degree models, we provide a table with important statistics about the residuals r_i.

5 Parallelization

In this section we present our parallelization algorithms and evaluate their scalability and efficiency.

5.1 Implementation

The meat of our program and most computationally expensive component is the computation of the all of the coefficients up until a certain $l_{\max}$.

5.1.1 OpenMP

The first loop that I tried to parallelize with OpenMP was the internal for loop computing the summation of N observations of $f(\phi_i, \theta_i)$ that needs to be run for each C_l^m, S_l^m pair. This seems completely logical, since the summation is the largest loop and we ideally want to reduce the overhead associated with parallel mechanisms like spawning new threads. However, we quickly observed that models of a sufficiently large degree were actually running slower with this parallel paradigm. We posit that even though we targeted the largest loop, we failed to consider that for every value (l, m) we are incurring overhead by splitting. For a model of $l = 100$ we end up with over 10,000 parallelization entry points.

Our second attempt at parallelization targeted the *outer* loop that iterates all the valid (l, m) pairs, computing C_l^m, S_l^m. This implementation, which only split apart a single for-loop, was immediately successful.

5.1.2 OpenMPI

After wrangling some indexing errors introduced when adapting to OpenMPI's memory model, we managed to succesfully compile a parallel mpi implementation of our algorithm.

5.2 Benchmark

The initial tests of scalability were performed on a desktop with a 12 core cpu. We repeated the computation of models with degre 50, 100, and 200 ten times, then plotted the average runtime of these trials and other pertinent statistics.

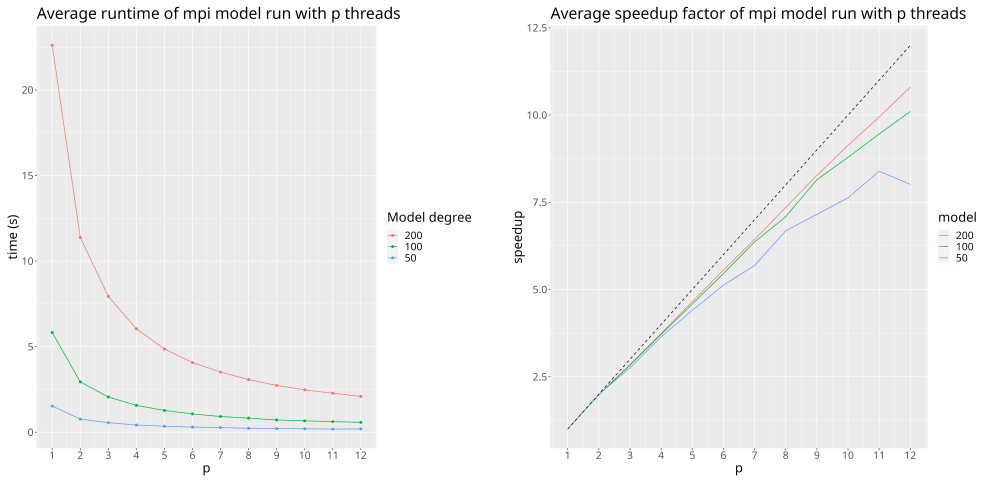

Figure 8: For models degree $l \in [50, 100, 200]$ we computed the average runtime across 10 executions. On the left-hand side we show how the runtime is affected by introducing more threads. On the right-hand side we plot the speedup coefficient, $S(p) = \frac{T_s}{T_p}$, defined as the runtime in sequential divided by the runtime with p mpi processes. The dashed line shows the ideal speedup factor, $S(p) = p$.

For a sufficiently large model (in the case of the small data set, $l = 200$), the speedup factor for mpi programs ran with up to 12 processes approaches ideal values. We cannot overstate the significance of this result. **The speedup factor of computing our models of degree l with p mpi processes approaches ideal levels.**

6 Final phase

In this section we present the implementation of a final optimization which makes use of the fact that we can compute a given (C_l^m, S_l^m) independly from all the other coefficients.

To provide context, we begin by establishing some new notation.

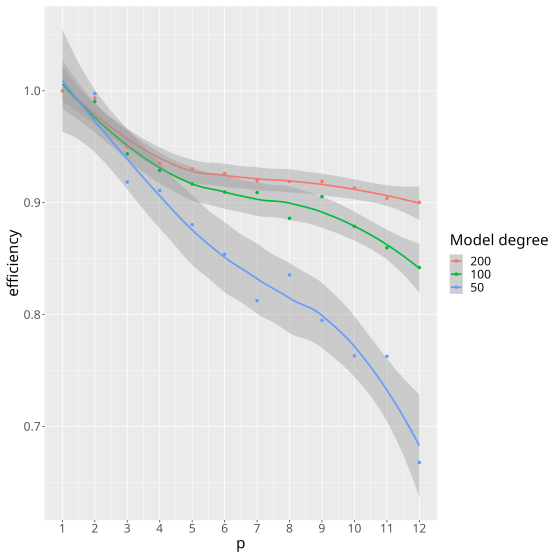

Figure 9: The efficiency, $\frac{S(p)}{p}$, is a test of how much of p threads' theoretical computation power is being harnessed. An ideal system has an efficiency of 1.

6.1 Ranges

Let C_l^* and S_l^* denote the set of Laplace Series coefficients whose degree is equal to l.

$$C_l^* = \{C_l^0, C_l^1, ..., C_l^l\}$$
$$S_l^* = \{S_l^0, S_l^1, ..., S_l^l\}$$

For example, $C_4^* = \{C_4^0, C_4^1, C_4^2, C_4^3, C_4^4\}$. Additionally, Let $C_{[a,b]}$ and $S_{[a,b]}$ denote the union of the coefficients of degree a through b:

$$C_{[a,b]} = \bigcup_{l=a}^{b} C_l^*$$
$$S_{[a,b]} = \bigcup_{l=b}^{b} S_l^*$$

In this section we refer to them as **ranges**. To elucidate this notation, we have

$$\begin{aligned}
S_{[2,3]} &= S_2^* \cup S_3^* \\
&= \{S_2^0, S_2^1, S_2^2\} \cup \{S_3^0, S_3^1, S_3^2, S_3^3\} \\
C_{[0,4]} &= C_0^* \cup C_1^* \cup C_2^* \cup C_3^* \cup C_4^* \\
&= \{C_0^0\} \cup \{C_1^0, C_1^1\} \cup \{C_2^0, C_2^1, C_2^2\} \cup \{C_3^0, C_3^1, C_3^2, C_3^3\} \cup \{C_4^0, C_4^1, C_4^2, C_4^3, C_4^4\} \\
&= \{C_0^0, C_1^0, C_1^1, C_2^0, C_2^1, C_2^2, C_3^0, C_3^1, C_3^2, C_3^3, C_4^0, C_4^1, C_4^2, C_4^3, C_4^4\}
\end{aligned}$$

Any range can easily be partioned into a union of disjoint subranges, a fact that will be exploited to save an enormous amount of computational time when constructing increasingly larger models. For example, $C_{[0,7]}$ can be partitioned into $C_{[0,3]} \cup C_{[4,7]}$. $|C_l^*|$, the cardinality of C_l^*, is equal to $\frac{(l+1)(l+2)}{2}$. We don't immediately see what $|C_{[a,b]}|$ might be equal to but we'd like to find an expression for the cardinality of a general range. We solve for this expression by rewriting $C_{[0,b]}$ as a union of disjoint subranges:

$$C_{[0,b]} = C_{[0,a-1]} \cup C_{[a,b]} \tag{4}$$

$$|C_{[0,b]}| = |C_{[0,a-1]}| + |C_{[a,b]}| \tag{5}$$

$$\frac{(b+1)(b+2)}{2} = \frac{a(a+1)}{2} + |C_{[a,b]}| \tag{6}$$

Solving for $|C_{[a,b]}|$ yields

$$|C_{[a,b]}| = \frac{(b+1)(b+2) - a(a+1)}{2} \tag{7}$$

Thus $\mathcal{M}_{l_{\max}}$, a model of degree $l_{\max}$, can be written as $\mathcal{M}_{l_{\max}} = C_{[0,l_{\max}]} \cup S_{[0,l_{\max}]}$. Assuming that we've already computed the coefficients for a model $\mathcal{M}_l$, all we need to compute a model $\mathcal{M}_{l+n}$ is to compute the new coefficients $C_{[l+1,l+n]}$ and $S_{[l+1,l+n]}$!

Let's explore the consequences of this observation using a concrete example. Suppose we have a precomputed model $\mathcal{M}_{500}$ for the high data set. If we would like to compute a model $\mathcal{M}_{600}$ using traditional means, we would need to compute $\frac{(601)(602)}{2} * 2 = 361802$ new coefficients. If, however, we leverage the fact that each C_l^m and S_l^m can be computed independently, we are only required to compute the coefficients $C_{[501,600]}$ and $S_{[501,600]}$. Computing $|C_{[501,600]}|$ gives us $\frac{(601)(602)-(501)(502)}{2} * 2 = 110300$ new coefficients to compute, which is more than 3 times fewer than for a classic model. What's significant is that once model of a certain degree has been computed, we don't forget the coefficients for subsequent runs. Practically speaking, this optimization paves the way for computation of models with degree in the thousands. A single run with 128 mpi threads to compute $\mathcal{M}_{400}$ clocked in at 180.546s, $\mathcal{M}_{500}$ in 289.548s, and $\mathcal{M}_{600}$ in 442.99s for the `ETOP01_hi.csv` dataset

We have already shown that the coefficients C_l^m and S_l^m computed via quadrature are independent of the $l_{\max}$ of a model. Therefore by storing the coefficients values of computed models in a binary file we accumulate a corpus of models of increasing degree. Storing the previously computed coefficients allows us to spread out the computation of a very large model across separate executions. The original quadrature model (and also the least-squares approach) computes all of the coefficients $(l+1)(l+2)$ in a single program execution. Once the necessary machinery is in place, everytime we run our program we will be inexorably advancing towards a higher degree model, instead of wasting time reevaluating coefficients that may have already been computed hundreds of times throughout testing.

Imagine the difference in wealth accumulated of a fictitious character whose bank account gets reset to zero every single data, compared to someone who only ever makes deposits and that persist across days.

7 Conclusion

Instead of calculating the model with a direct method that runs in $O(Nl_{max}^4)$ time, We propose a hybrid model that first directly computes the coefficients in $O(Nl_{max}^2)$ time then iteratively improves the MSE via stochastic methods running in $O(n_{training} n_{sample} l_{max}^2)$ time.

Results are to follow :)

A Spherical Coordinates Transformation

The data sets `ETOP01_*.csv` provided by NASA represent spherical coordinates in an unconventional format:

$$(\phi, \lambda) \in [-\frac{\pi}{2}, \frac{\pi}{2}] \times [-\pi, \pi]$$

where ϕ refers to the latitude and λ refers to the longitude. In this section we outline the transformation from NASA's coordinate system to the ISO standard representation and prove equivalence between (1) and the Project presentation's $f(\phi, \lambda)$.

To avoid confusion, we denote the spherical coordinate pair in ISO coordinates as $(\theta_{iso}, \phi_{iso})$ and NASA's coordinate scheme as $(\phi_{nasa}, \lambda_{nasa})$. We remark that the polar angle θ_{iso} is simply equal to $\frac{\pi}{2} - \phi_{nasa}$. We corroborate this fact by remarking that a latitude of $\phi_{nasa} = 0$ corresponds

to a polar angle of $\phi_{iso} = \frac{\pi}{2}$. Conveniently, the longitude λ_{nasa} is equivalent to the azimuthal angle ϕ_{iso}.

Proposition 1. Let the spherial coordinate pairs $(\theta_{iso}, \phi_{iso})$, $(\phi_{nasa}, \lambda_{nasa})$ be defined as above.

$$f(\theta_{iso}, \phi_{iso}) \equiv f(\phi_{nasa}, \lambda_{nasa})$$

Proof. $f(\phi_{nasa}, \lambda_{nasa})$ is defined in the Project proposal as

$$f(\phi_{nasa}, \lambda_{nasa}) = \sum_{l=0}^{+\infty} \sum_{m=0}^{l} \bar{P}_l^m(\sin \phi_{nasa})[C_l^m \cos m\lambda_{nasa} + S_l^m \sin m\lambda_{nasa}] \tag{8}$$

$$= \sum_{l=0}^{+\infty} \sum_{m=0}^{l} \bar{P}_l^m(\sin(\pi/2 - \theta_{iso}))[C_l^m \cos m\phi_{iso} + S_l^m \sin m\phi_{iso}] \tag{9}$$

$$= \sum_{l=0}^{+\infty} \sum_{m=0}^{l} \bar{P}_l^m(\cos \theta_{iso}))[C_l^m \cos m\phi_{iso} + S_l^m \sin m\phi_{iso}] \tag{10}$$

$$\equiv f(\theta_{iso}, \phi_{iso}) \tag{11}$$

□

References

[1] V. Rokhlin and M. Tygert, "Fast Algorithms for Spherical Harmonic Expansions," *SIAM Journal on Scientific Computing*, vol. 27, pp. 1903–1928, Jan. 2006. Publisher: Society for Industrial and Applied Mathematics.

[2] R. Blais, "Discrete Spherical Harmonic Transforms: Numerical Preconditioning and Optimization," pp. 638–645, June 2008.

[3] T. Limpanuparb and J. Milthorpe, "Associated Legendre Polynomials and Spherical Harmonics Computation for Chemistry Applications," Oct. 2014. arXiv:1410.1748 [physics].

[4] E. W. Weisstein, "Spherical Harmonic." Publisher: Wolfram Research, Inc.

[5] G. B. Arfken, H. J. Weber, and F. E. Harris, *Mathematical Methods for Physicists: A Comprehensive Guide*. Amsterdam ; Boston: Academic Press Inc, 7e édition ed., Mar. 2012.

[6] P. Dyke, "Fourier Series," in *An Introduction to Laplace Transforms and Fourier Series* (P. Dyke, ed.), Springer Undergraduate Mathematics Series, pp. 83–122, London: Springer, 2014.

[7] E. W. Weisstein, "Laplace Series." Publisher: Wolfram Research, Inc.