Fast Interpolation of Periodic Bandlimited Nonuniform Data

Samuel F. Potter, Nail Gumerov, Ramani Duraiswami

July 7, 2016

Abstract

The nonuniform fast Fourier transform (FFT) is a class of algorithms which brackets the FFT with a pair of interpolating maps which have complexity equal or superior to the FFT (generally $O(n \log n)$, where n is the size of the argument). A set of algorithms based on the fast multipole method (FMM) providing sufficient complexity guarantees exists [1]—each such algorithm relies on a fast algorithm for the Cauchy or cotangent kernel. In this work, a recent method for computing periodic sums is applied in order to improve the speed and accuracy of these kernels, which is then applied to the nonuniform FFT. Error bounds and complexity guarantees are proved, and numerical results are presented.

Contents

1 Introduction

The discrete Fourier transform (DFT) is an orthogonal linear transformation which is a discrete analog of the continuous Fourier transform. Many problems in science and engineering yield to the methods of Fourier analysis, so much so that when the fast Fourier transform (FFT) algorithm of Cooley and Tukey was disclosed in 1965, much of the focus of research in the field of signal processing shifted from research in analog signal processing to digital signal processing, owing to the newfound feasibility of previously speculative numerical methods [7]. Of course, this algorithm has gone on to enable the efficient solution of a myriad of problems since.

A limitation of the real FFT is that its domain consists of real vectors whose components are assumed to correspond to equally spaced samples of some underlying continuous signal; likewise, the range is comprised of coefficients of equally spaced **TODO:** INTEGER? frequency components. When presented with a nonuniformly sampled input signal or when one's goal is to determine the strength of frequency components which are unequally spaced, the usual FFT is of no use without interpolating before or after the transform. A variety of methods have been introduced to formalize an approach to computing such a nonuniform DFT or FFT [1, 6].

In this work, we extend a method for computing the nonuniform FFT which makes use of the fast multipole method (FMM) [1]. In particular, making use of recent work inspired by the kernel independent FMM, we append a step to this method which takes a negligible amount of time but increases the accuracy of the result. Algorithms are derived and their complexity is analyzed, relevant error bounds are established, and numerical results are presented.

2 The Nonuniform FFT

Let $f : \mathbb{R} \to \mathbb{R}$ be a bandlimited function, and let $K \in \mathbb{N}$ be its bandlimit. If, for each k such that $0 \leq k < K$, we let $x_k = 2\pi k/K$ and $f_k = f(x_k)$, then the DFT of $f_0, \ldots, f_{K-1}$ is given by:

$$F_l = \sum_{k=0}^{K-1} f_k e^{-2\pi i k l/K} \tag{1}$$

for l such that $0 \leq l < K$. The f_k's and F_l's can be thought of as ordinates associated with corresponding equispaced ascissae. For arguments off of these regular grids of points (in the time and frequency domains, respectively), the question as to how to evaluate f naturally arises. The general class of solutions provides methods for computing what is referred to as the nonuniform DFT.

The inverse DFT formula leads us to interpret the DFT coefficients $F_0, \ldots, F_{K-1}$ as the coefficients of a trigonometric polynomial:

$$f(x) = \sum_{l=-K/2+1}^{K/2-1} F_l e^{ilx} \tag{2}$$

If, for j such that $0 \leq j < J$, we define $\tilde{x}_j$ such that $0 \leq \tilde{x}_j < 2\pi$, then combining (1) with (2) evaluated at each $\tilde{x}_j$ yields:

$$\tilde{f}_j \triangleq f(\tilde{x}_j) = \frac{1}{K} \sum_{k=0}^{K-1} \left(\sum_{l=-K/2+1}^{K/2-1} e^{il(\tilde{x}_j - x_k)} \right) f_k. \tag{3}$$

It is readily shown that this expression can be manipulated further to yield:

$$\tilde{f}_j = \frac{\sin(K\tilde{x}_j/2)}{K} \sum_{p=-\infty}^{\infty} \sum_{k=0}^{K-1} \frac{(-1)^k f_k}{\tilde{x}_j - x_k - 2\pi p}. \tag{4}$$

A proof of this can be found in the appendix.

Equation (4) allows us to consider the interpolation as a linear operator applied to the values of a fourier series at equispaced points. We can define this operator $\boldsymbol{P} \in \mathbb{R}^{J \times K}$ by:

$$\boldsymbol{P}_{j,k} \triangleq \frac{(-1)^k \sin(K\tilde{x}_j/2)}{K} \sum_{p=-\infty}^{\infty} \frac{1}{\tilde{x}_j - x_k - 2\pi p}. \tag{5}$$

Thus, letting $\boldsymbol{f} \in \mathbb{R}^K$ and $\tilde{\boldsymbol{f}} \in \mathbb{R}^J$ be defined by $\boldsymbol{f}_k = f_k$ and $\tilde{\boldsymbol{f}}_j = \tilde{f}_j$, respectively, we can see that $\tilde{\boldsymbol{f}} = \boldsymbol{P}\boldsymbol{f}$. The expression for $\tilde{f}_j$ provided in (4) can be derived from results in the paper from which this work primarily derives [1]. The inverse interpolation formula is also given in this paper. From this formula, we have that the nominal approximate "inverse" interpolation operator $\boldsymbol{P}^{-1} \in \mathbb{R}^{K \times J}$ is defined by letting:

$$c_j \triangleq \prod_{k=0}^{K-1} \sin\left(\frac{\tilde{x}_j - x_k}{2}\right), \tag{6}$$

$$d_k \triangleq \prod_{\substack{k'=0 \\ k' \neq k}}^{K-1} \frac{1}{\sin\left(\frac{x_k - x_{k'}}{2}\right)}, \tag{7}$$

$$\boldsymbol{P}^{-1}_{k,j} \triangleq c_j \cdot d_k \cdot \sum_{p=-\infty}^{\infty} \frac{1}{\tilde{x}_j - x_k - 2\pi p}. \tag{8}$$

The corresponding formula for recovering f_k from $\tilde{f}_j$ is:

$$x_k = d_k \sum_{p=-\infty}^{\infty} \sum_{j=0}^{J-1} \frac{c_j}{\tilde{x}_j - x_k - 2\pi p}. \tag{9}$$

If we let $\boldsymbol{F}$ denote a linear operator corresponding to the DFT, e.g. an application of some FFT, then there are four operators of interest which together comprise the nonuniform FFT (Figure 1). Of course, further combinations are possible—e.g. $\boldsymbol{P}^{-\top} \circ \boldsymbol{F} \circ \boldsymbol{P}^{-1}$. As mentioned, the goal is to the design algorithms which compute $\boldsymbol{P}$ and its variations with complexity no worse than $\boldsymbol{F}$ and $\boldsymbol{F}^{-1}$. As it happens, each of (5–8) can be computed sufficiently fast. Writing:

$$\log(c_j) = \sum_{j=0}^{K-1} \log \circ \sin\left(\frac{\tilde{x}_j - x_k}{2}\right), \tag{10}$$

and similarly for d_k, allows the FMM for the $\log \circ \sin$ kernel to be applied so that these coefficients may be precomputed, although this is not our focus.

Operator	Description
$\boldsymbol{N} \triangleq \boldsymbol{P} \circ \boldsymbol{F}^{-1}$	Compute the IDFT and interpolate in the time domain.
$\boldsymbol{N}^\top \triangleq \boldsymbol{F}^{-1} \circ \boldsymbol{P}^\top$	Interpolate uniform frequencies from arbitrary ones and compute IDFT.
$\boldsymbol{N}^{-1} \triangleq \boldsymbol{F} \circ \boldsymbol{P}^{-1}$	Approximate uniformly spaced Fourier series values and compute the DFT.
$\boldsymbol{N}^{-\top} \triangleq \boldsymbol{P}^{-\top} \circ \boldsymbol{F}$	Apply the DFT and interpolate in the frequency domain.

Figure 1: the various nonuniform FFT operators.

- **TODO:** use much nicer $X(e^{j\omega})$ notation for frequencies—can overload the notation for both uniform and nonuniform frequencies.

- **TODO:** min/max NUFFT paper has information about how to specify the DFT, "DFT frequency sets" (good terminology), and *possibly* some information about the selection of valid nonuniformly spaced frequencies?
- **TODO:** include figure of interpolating function (this figure is in the poster).
- **TODO:** include note about periodic sinc formula/Dirichlet function? See "Dirichlet kernel" on Wikipedia.
- **TODO:** add note about min-max NUFFT.
- **TODO:** compare error pattern graph with min-max error formulation?

3 The Fast Multipole Method

The acceleration of the interpolation steps in the preceding section relies on a sufficient accurate and optimized fast multipole method (FMM) algorithm. The FMM is an algorithm which computes matrix-vector products of a certain form in sub-quadratic time [3]. Typically, the FMM is developed for a particular radial basis function of a certain dimensionality. In this work, we provide an optimized one-dimensional FMM for the Cauchy kernel:

$$\Phi(y, x) = \frac{1}{y - x}. \tag{11}$$

Implementing an FMM requires the derivation of regular and singular factorizations the kernel function (so-called S-factorizations and R-factorizations), along with derivations of translation operators which reexpand these factorizations: singular-to-singular, singular-to-regular, and regular-to-regular translation operators [4].

The R and S-factorizations of the Cauchy kernel are derived in a straightforward manner from the Taylor expansion of Φ. For the S-factorization, we fix x, y, and x_* such that $|x - x_*| < |y - x_*|$ and define $b_m(x, x_*) = (x - x_*)^m$ and $S_m(y - x_*) = (y - x_*)^{-m-1}$. The point x_* is referred to as the expansion center. Then, the S-factorization of Φ is given by:

$$\Phi(y, x) = \sum_{m=0}^{\infty} b_m(x, x_*) S_m(y - x_*). \tag{12}$$

Very similarly, for the R-factorization, we fix x, y, and x_* such that $|y - x_*| < |x - x_*|$ and define $a_m(x, x_*) = -(x - x_*)^{-m-1}$ and $R_m(y - x_*)^m = (y - x_*)^m$, giving us:

$$\Phi(y, x) = \sum_{m=0}^{\infty} a_m(x, x_*) R_m(y - x_*). \tag{13}$$

These expansions of Φ are used in the implementation of our FMM, as well as in our discussion of error bounds.

The FMM involves evaluating a function which is comprised of a sum of weighted kernels at a set of target points. In doing so, the computational domain is decomposed using a spatial data structure (e.g. a binary tree or octree). The algorithm first expands each weighted kernel using an S-factorization, then applies the translation operators according to the spatial decomposition (Figure 2). We denote the singular-to-singular translation matrix by $\mathbf{S}|\mathbf{S}$. Represented as an infinite matrix, its entries are given by:

$$(\mathbf{S}|\mathbf{S})_{n,m} \triangleq \frac{(-1)^{n-m} n! \delta^{n-m}}{(n-m)! m!}, \tag{14}$$

where δ is the translation vector between the two expansion centers. That is, if the resultant S-factorization is to be expanded about x'_*, then $\delta = x_* - x'_*$. The corresponding S-reexpansion is:

$$S_m(y - x_*) = \sum_{n=0}^{\infty} (\mathbf{S}|\mathbf{S})_{n,m} \, S_n(y - x'_*). \tag{15}$$

Such a reexpansion requires that $|\delta| < |y - x'_*|$. More details can be found in the appendix, but the expressions for the singular-to-regular translation operator $\mathbf{S}|\mathbf{R}$ and the regular-to-regular translation operator $\mathbf{R}|\mathbf{R}$ are given by:

$$(\mathbf{S}|\mathbf{R})_{n,m} \triangleq \frac{(-1)^n (m+n)!}{m!n!\delta^{m+n+1}}, \tag{16}$$

and:

$$(\mathbf{R}|\mathbf{R})_{n,m} = \begin{cases} \frac{m!\delta^{m-n}}{(m-n)!n!} & \text{if } n \leq m, \\ 0 & \text{otherwise,} \end{cases} \tag{17}$$

respectively.

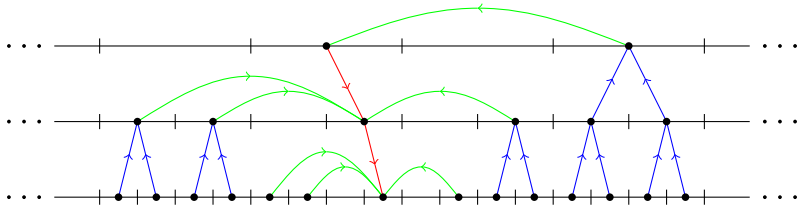

Figure 2: an illustration of the translation phases of the one-dimensional FMM.

- **TODO:** truncations and error.
- **TODO:** label translation operators in FMM figure, label direct sum bins vs far sum bins.
- **TODO:** better colors for FMM figure.
- **TODO:** make FMM figure breath a little better.

4 Periodic Summation

The FMM sees frequent application in dynamics problems such as the n-body problem. For problems with periodic boundary conditions, the FMM on its own is not sufficient. Others algorithms are used instead—for example, Ewald summation, accelerated using the FFT [2]. Recent work inspired by kernel-independent FMM algorithms has resulted in a method for computing periodic sums which is ideally suited for use with the FMM [5]. Our method makes use of a simplified and accelerated version of this approach. We briefly derive the necessary results as they pertain to our problem.

The periodic summation method uses an FMM and least squares fitting to evaluate a sum which is a sum of weighted kernels which periodically repeat in some infinite computational domain. In our case, we are interested in computing the part of (4) which matches this description—namely, we define:

$$\phi(y) \triangleq \sum_{p=-\infty}^{\infty} \sum_{k=0}^{K-1} \frac{(-1)^k f_k}{y - x_k - 2\pi p}, \tag{18}$$

where we have ingored the constant factor. Next, in order to apply the method, we select some region of interest, a larger neighborhood which contains it, and decompose ϕ into a term due to the neighborhood and one due to its complement. Our region of interest is the interval $[0, 2\pi)$. We fix $n \in \mathbb{N}$ and consider it to be an integer "neighborhood radius" and define:

$$\mathcal{P} \triangleq \{-n, -n+1, \ldots, n\}. \tag{19}$$

Then, letting:

$$\phi_{\text{near}}(y) \triangleq \sum_{p \in \mathcal{P}} \sum_{k=0}^{K-1} \frac{(-1)^k f_k}{y - x_k - 2\pi p}, \tag{20}$$

$$\phi_{\text{far}}(y) \triangleq \sum_{p \notin \mathcal{P}} \sum_{k=0}^{K-1} \frac{(-1)^k f_k}{y - x_k - 2\pi p}, \tag{21}$$

we can decompose ϕ as:

$$\phi(y) = \phi_{\text{near}}(y) + \phi_{\text{far}}(y). \tag{22}$$

With $x_* = \pi$, we have that when y satisfies $0 \leq y < 2\pi$, the R-factorization:

$$\frac{1}{y - x_k - 2\pi p} = \sum_{m=0}^{\infty} a_m(x_k + 2\pi p, \pi) R_m(y - \pi) \tag{23}$$

is valid. Making use of (23), we can define, for each $m = 0, 1, \ldots$:

$$c_m \triangleq \sum_{p \notin \mathcal{P}} \sum_{k=0}^{K-1} (-1)^k f_k a_m(x_k + 2\pi p, \pi), \tag{24}$$

which allows us to define:

$$\phi_{\text{far}}(y) = \sum_{m=0}^{\infty} c_m R_m(y - \pi). \tag{25}$$

Reexpressing ϕ_{far} in terms of the R_m basis functions of the R-factorization is the crucial step that enables the method.

The periodic summation method involves using the FMM to compute the neighborhood term ϕ_{near} and then using the FMM and least squares collocation to compute ϕ_{far}. In particular, if we choose y such that $-2\pi n \leq y < 0$ or $2\pi \leq y < 2\pi(n+1)$, then there exists some p such that $\tilde{y} = y - 2\pi p$ satisfies $0 \leq \tilde{y} < 2\pi$. Since $\phi(y) = \phi(\tilde{y})$, we have:

$$\phi_{\text{near}}(y) - \phi_{\text{near}}(\tilde{y}) = \phi_{\text{far}}(\tilde{y}) - \phi_{\text{far}}(y) = \sum_{m=0}^{\infty} c_m (R_m(y - \pi) - R_m(\tilde{y} - \pi)). \tag{26}$$

Fixing $L \in \mathbb{N}$, we let $y_0, \ldots, y_{L-1}$ and $\tilde{y}_0, \ldots, \tilde{y}_{L-1}$ satisfy the same properties as y and $\tilde{y}$. Defining the matrices $\boldsymbol{\phi} \in \mathbb{R}^L, \boldsymbol{c} \in \mathbb{R}^\infty$, and $\boldsymbol{R} \in \mathbb{R}^{L \times \infty}$ by:

$$\boldsymbol{\phi}_l = \phi_{\text{near}}(y_l) - \phi_{\text{near}}(\tilde{y}_l), \tag{27}$$

$$\boldsymbol{c}_m = c_m, \tag{28}$$

$$\boldsymbol{R}_{l,m} = R_m(y_l - \pi) - R_m(\tilde{y}_l - \pi), \tag{29}$$

the coefficients c_m can be recovered by solving the corresponding least squares problems, e.g. by computing each c_m from $\boldsymbol{c} \approx \boldsymbol{R}^\dagger \boldsymbol{\phi}$.

There are two practical considerations in all of this. First, we only compute and apply a finite portion of $\boldsymbol{R}$, although the full matrix will be used in our error analysis. Second, since $\boldsymbol{R}_{l,0} = 0$ for each $l \in \mathbb{N}$, this column can be ignored. As a result, the question as to what to do about c_0 arises—this is addressed in the next section. The resulting submatrix can be dealt with by making use of a convenient matrix decomposition, also addressed in a later section.

- **TODO:** add necessary conditions—or break them out into appendix?

5 Computation of c_0

The first column of the fitting matrix $\boldsymbol{R}$ is zero, which prevents the preceding least squares fitting procedure from recovering c_0. For some problems, this does not present a problem, but for this one it does, and a method of computing c_0 is necessary. As it happens, approximating c_0 from:

$$c_0 = \sum_{p \notin \mathcal{P}} \sum_{k=0}^{K-1} \frac{(-1)^k f_k}{x_k + 2\pi(p-1)} \tag{30}$$

is unrealistic due to slow convergence—the series is similar to the alternating harmonic series, for instance. Other methods have been suggested [5].

Empirically, the coefficients c_m are small for $m > 0$. Then, for some small $\varepsilon > 0$, we should have:

$$f_k \approx \tfrac{1}{K} \sin(K \cdot (x_k + \varepsilon)/2)(c_0 + \phi_{\text{near}}(x_k + \varepsilon)). \tag{31}$$

This suggests that we can approximate c_0 by solving:

$$c_0 \approx \min_c \sum_{k=0}^{K-1} \left(f_k - \tfrac{1}{K} \sin(K \cdot (x_k + \varepsilon)/2)(c_0 + \phi_{\text{near}}(x_k + \varepsilon))\right)^2. \tag{32}$$

If we let $s_k = \frac{1}{K} \sin(K \cdot (x_k + \varepsilon)/2)$ and $\phi_k = \phi_{\text{near}}(x_k + \varepsilon)$, then the analytic solution to (32) is:

$$c_0 \approx \frac{\sum_{k=0}^{K-1} s_k(f_k - s_k\phi_k)}{\sum_{k=0}^{K-1} s_k^2}. \tag{33}$$

Despite this method's heuristic nature, it was found to compute accurate approximations to c_0 in essentially constant time as compared to the direct method mentioned earlier. However, it should be noted that choosing ε to be too small can be problematic.

- **TODO:** add figure showing divergence.
- **TODO:** try using the coefficients c_m for $m > 0$ to recover even more accurately... If this isn't possible, make sure to indicate why (compare `cest_new` with `cest` in `sanitycheck.py`).

6 Fitting Matrix

The least squares fitting matrix $\boldsymbol{R}$ from (29) is given by:

$$\boldsymbol{R} = \begin{bmatrix} R_1(y_0 - \pi) - R_1(\tilde{y}_0 - \pi) & \cdots & R_P(y_0 - \pi) - R_P(\tilde{y}_0 - \pi) \\ \vdots & \ddots & \vdots \\ R_1(y_{L-1} - \pi) - R_1(\tilde{y}_{L-1} - \pi) & \cdots & R_P(y_{L-1} - \pi - R_P(\tilde{y}_{L-1} - \pi) \end{bmatrix} \tag{34}$$

If we define $z_l = y_l - \pi$ and $\tilde{z}_l = \tilde{y}_l - \pi = y_l - 2\pi p$, we have that $\boldsymbol{R}$ can also be written:

$$\boldsymbol{R}_{l,m} = z_l^m - (z_l - 2\pi p)^m. \tag{35}$$

This requires us to choose the same p for each $\tilde{y}_l$. In a later section, we will see that this does not present a problem.

Some algebra yields that this matrix can be decomposed as the product of a Vandermonde matrix in $z_0, \ldots, z_{L-1}$ and an upper triangular matrix. In particular, letting:

$$\boldsymbol{V} \triangleq \begin{bmatrix} 1 & z_0 & z_0^2 & \cdots & z_0^{P-1} \\ 1 & z_1 & z_1^2 & \cdots & z_1^{P-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & z_{L-1} & z_{L-1}^2 & \cdots & z_{L-1}^{P-1} \end{bmatrix} \tag{36}$$

denote the Vandermonde matrix, and:

$$\boldsymbol{W}_{i,j} \triangleq \begin{cases} (-2\pi p)^{j-i+1} \binom{j}{j-i+1}, & \text{if } i \leq j, \\ 0, & \text{if } i > j, \end{cases} \tag{37}$$

where $\boldsymbol{W} \in \mathbb{R}^{P \times P}$, we have that $\boldsymbol{R} = \boldsymbol{V}\boldsymbol{W}$. The appendix contains a proof of this as well as some of $\boldsymbol{W}$ explicitly tabulated.

The properties of the Vandermonde matrix $\boldsymbol{V}$ are well understood. One property, the fact that $\boldsymbol{V}$ is—generally speaking—extremely poorly conditioned, will be considered in the next section. In this section, we make use of a well-known factorization of $\boldsymbol{V}^{-1}$ which will aid us in developing a fast and easily implemented algorithm for computing the fitting coefficients [8]. In particular, we have that there exists a lower triangular matrix $\boldsymbol{L}^{-1}$ and an upper triangular matrix $\boldsymbol{U}^{-1}$ such that $\boldsymbol{V}^{-1} = \boldsymbol{U}^{-1}\boldsymbol{L}^{-1}$ and such that the lower triangular factor is given by:

$$\boldsymbol{L}^{-1}_{i,j} = \begin{cases} 1, & \text{if } i = 1 = j, \\ 0, & \text{if } i < j, \\ \prod_{k=1, k\neq j}^{i} \frac{1}{z_j - z_k}, & \text{otherwise}, \end{cases} \tag{38}$$

and the upper triangular factor by:

$$\boldsymbol{U}^{-1}_{i,j} = \begin{cases} 1, & i = j, \\ 0, & j = 1, i > 1, \\ \boldsymbol{U}^{-1}_{i-1,j-1} - \boldsymbol{U}^{-1}_{i,j-1} z_{j-1}, & \text{otherwise}, \end{cases} \tag{39}$$

where $\boldsymbol{U}^{-1}_{0,j} = 0$. Our algorithm for computing the coefficients c_m, then, is based on writing $\boldsymbol{R}\boldsymbol{c} = \boldsymbol{\phi}$ as $\boldsymbol{W}\boldsymbol{c} = \boldsymbol{U}^{-1}\boldsymbol{L}^{-1}\boldsymbol{\phi}$. The right-hand side can be computed efficiently due to the simple and regular form of the matrices which allows $\boldsymbol{c}$ to be computed using an efficient back-substitution.

7 Checkpoint Distribution

The distribution of the checkpoints $\tilde{y}_0, \ldots, \tilde{y}_{L-1}$ can have a significant effect on the accuracy of the periodic summation method. The Cauchy kernel was found to be very poorly conditioned depending on the checkpoint distribution. The decomposition $\boldsymbol{R} = \boldsymbol{V}\boldsymbol{W}$ reveals the reason—one of its factors is a Vandermonde matrix, which is known to be very poorly conditioned, in general. The determinant of $\boldsymbol{V}$ is given by:

$$\det(\boldsymbol{V}) = \prod_{0 \leq l < l' < L} (z_l - z_{l'}). \tag{40}$$

A necessary condition for the algorithm to work, then, is for the checkpoints to be unique—if any are equal, $\boldsymbol{V}$ is singular. However, beyond this, if the checkpoints are spread too far apart, the determinant quickly becomes unbounded, resulting in a poorly conditioned fitting matrix. As our matrix decomposition requires us to fix p for each $\tilde{y}_l$, we simply let $p = \pm 1$.

- **TODO:** add section that gets into the condition number of the fitting matrix, and talks about how to choose checkpoints based off of this.
- **TODO:** add figure of wacky interpolated polynomials that are all screwed up.
- **TODO:** compute actual determinant and do a little smarter analysis—determinant of $\boldsymbol{W}$ is just $P!$.
- **TODO:** it's probably possible to bound the determinant (and also κ) by using the above two bits. Don't forget about inverse vs. normal matrix...
- **TODO:** does replacing y_l with z_l improve the condition number?
- **TODO:** This might be a little wrong... Determinants don't really seem to be related to the condition number. Look into some references for this.
- **TODO:** Add plot of $\log_{10}(\kappa(\boldsymbol{R}))$ (approximation of number of digits lost).

8 Main Algorithm

Our main algorithms apply $\boldsymbol{P}$ and $\boldsymbol{P}^{-1}$ by evaluating (4) using the periodic summation method as applied to (18). Evaluating $\boldsymbol{P}^{\top}$ and $\boldsymbol{P}^{-\top}$ can be done by interchanging arguments.

Algorithm 1. Compute $\phi(\tilde{x}_j)$ for each j.

1. Evaluate $\phi_{\text{near}}(\tilde{x}_j)$ for each j using the FMM.
2. Compute $\boldsymbol{\phi}$ using the FMM and the checkpoint sets $\{y_l\}$ and $\{\tilde{y}_l\}$.
3. Form $\boldsymbol{U}^{-1}\boldsymbol{L}^{-1}\boldsymbol{\phi}$.
4. Solve $\boldsymbol{W}\boldsymbol{c} = \boldsymbol{U}^{-1}\boldsymbol{L}^{-1}\boldsymbol{\phi}$ for $\boldsymbol{c}$ using back substitution.
5. Compute c_0 **TODO:** *by some method.*
6. Evaluate $\phi_{\text{far}}(\tilde{x}_j)$ for each j using $\boldsymbol{c}$ and (25).
7. Compute $\phi(\tilde{x}_j) = \phi_{\text{near}}(\tilde{x}_j) + \phi_{\text{far}}(\tilde{x}_j)$ for each j.

Algorithm 2. Compute $\tilde{f}_j$ for each j (apply $\boldsymbol{P}$ to $\boldsymbol{f}$).

1. Evaluate $\phi(\tilde{x}_j)$ for each j using Algorithm 1.
2. Compute each $\tilde{f}_j = \sin(K\tilde{x}_j/2)\phi(\tilde{x}_j)/2$.

Algorithm 3. Compute f_k for each k (apply $\boldsymbol{P}^{-1}$ to $\tilde{\boldsymbol{f}}$).

1. Precompute c_j for each j using the FMM.
2. Precompute d_k for each k using the FMM.
3. Evaluate (9) for each k using the precomputed coefficients and Algorithm 1.

- **TODO:** include the main algorithm.
- **TODO:** include Vandermonde step and c_0 step.

9 Error Analysis

- **TODO:** do error analysis of R and S factorizations, and $\mathbf{S}|\mathbf{S}$, $\mathbf{S}|\mathbf{R}$, and $\mathbf{R}|\mathbf{R}$ translation operators.
- **TODO:** do error analysis of least squares fitting...?

10 Complexity

- **TODO:** add complexity analysis from writeup.

11 Numerical Results

- **TODO:** include figure of approximated polynomial, possibly with "off" polynomials that are computed from checkpoints with higher condition number.
- **TODO:** compare manually computing c_0 with our method of estimating c_0.
  - **TODO:** include the plot which shows how our method for estimating c_0 diverges as $\delta \downarrow 0$.
  - **TODO:** include a plot which accurately shows how our estimated c_0 is quite accurate as compared to manually computing c_0 terms. Have the plot include several different choices of δ (i.e. $\delta = 10^{-1}, 10^{-2}, 10^{-3}, \ldots$), use $L = 10^1, 10^2, 10^3, 10^4, \ldots$—i.e. logarithmic on both scales.
- **TODO:** compare our Vandermonde inverse approach with using something like the SVD.

12 Future Work

- **TODO:** talk about Constant-Q transform.
- **TODO:** talk about speeding up Cauchy kernel and how it will also be useful for speeding up FMM rotation operators.
- **TODO:** Chebyshev interpolation/Vandermonde matrix stuff...?
- **TODO:** complex checkpoints and figure.
- **TODO:** extend to multiple dimensions...

A Appendix

A bit of the matrix $\boldsymbol{U}$ is depicted as follows:

$$\boldsymbol{W} = -2\pi p \begin{bmatrix} 1 & -2p\pi & 4p^2\pi^2 & -8p^3\pi^3 & 16p^4\pi^4 & -32p^5\pi^5 & \cdots \\ & 2 & -6p\pi & 16p^2\pi^2 & -40p^3\pi^3 & 96p^4\pi^4 & \cdots \\ & & 3 & -12p\pi & 40p^2\pi^2 & -120p^3\pi^3 & \cdots \\ & & & 4 & -20p\pi & 80p^2\pi^2 & \cdots \\ & & & & 5 & -30p\pi & \cdots \\ & & & & & 6 & \cdots \\ & & & & & & \ddots \end{bmatrix} \tag{41}$$

- **TODO:** prove (**??**).
- **TODO:** prove Vandermonde-upper triangular decomposition of $\boldsymbol{R}$.

B References

[1] A. Dutt and V. Rokhlin. Fast fourier transforms for nonequispaced data, II. *Applied and Computational Harmonic Analysis*, pages 85–100, 1995.

[2] P. P. Ewald. Die Berechnung optischer und elektrostatischer Gitterpotentiale. *Ann. Phys.*, 369(3):253–287, 1921.

[3] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *J. Comput. Phys.*, 73, December 1987.

[4] N. A. Gumerov and R. Duraiswami. *Fast Multipole Methods for the Helmholtz Equation in Three Dimensions*. Elsevier, 2004.

[5] N. A. Gumerov and R. Duraiswami. A method to compute periodic sums, 2014.

[6] J. Keiner, S. Kunis, and D. Potts. NFFT 3.0 tutorial. Technical report, TU Chemnitz, 2006.

[7] A. V. Oppenheim and R. W. Schafer. *Digital Signal Processing*. Prentice Hall, 1st edition, 1975.

[8] R. L. Turner. Inverse of the Vandermonde matrix with applications. Technical report, NASA, 1966.