2017

Nonequispaced Fast Fourier Transform

David Hughey
University of South Carolina

Nonequispaced Fast Fourier Transform

by

David Hughey

Bachelor of Science
University of South Carolina 2015

Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Science in

Mathematics

College of Arts and Sciences

University of South Carolina

2017

Accepted by:

Pencho Petrushev, Director of Thesis

Peter Binev, Reader

Cheryl L. Addy, Vice Provost and Dean of the Graduate School

Dedication

I would like to dedicate my thesis to my parents who have supported and helped me out throughout my studies.

Acknowledgments

I would like to thank everyone that has given me the opportunities that lead to this thesis: Pencho Petrushev for advising me throughout my undergraduate and graduate time, Scott Johnson for helping me learn programming, Jeff Francis for getting me interested in signal processing, and all the others who have taught me a long the way.

Abstract

Two algorithms for fast and accurate evaluation of high degree trigonometric polynomials at many scattered points are presented. Both methods rely on highly localized kernels and the Fast Fourier Transform. The first algorithm uses the function values at uniformly distributed grid points and kernels that reproduce trigonometric polynomials, while the second method uses kernels that approximate well the function on the frequency side. Both algorithm are termed Nonequispaced Fast Fourier Transform. The first algorithm is coded in MATLAB and shown to approximate well the function to be evaluated.

Table of Contents

List of Figures

Chapter 1

Introduction and Background

1.1 Introduction

The Fast Fourier Transform (FFT) is one of the most important and frequently used algorithms in Signal Processing. This algorithm allows to compute rapidly the values $f[j]$, $j = 0, 1, \ldots, N-1$, of a discrete signal given its Fourier coefficients $\hat{f}[j]$, $j = 0, 1, \ldots, N-1$. That is, to compute

$$f[j] = \sum_{\nu=0}^{N-1} \hat{f}[\nu] e^{j\nu \frac{2\pi i}{N}}, \quad j = 0, 1, \ldots, N-1. \tag{1.1}$$

The inversion of (**??**) has a similar form and is also amenable to the FFT. The number of operations required by the FFT is $O(N \log N)$. For details on FFT, see §**??** below.

In this thesis the focus is on the Nonequispaced Fast Fourier Transform (NFFT). The **problem** is: Given the Fourier coefficients $\{\hat{f}_k\}_{k=-N}^{N}$ of a trigonometric polynomial, f, compute its values

$$f(x) = \sum_{k=-N}^{N} \hat{f}_k e^{-2\pi i k x} \tag{1.2}$$

at many scattered points $\{x_j\}_{j=1}^{J}$. Observe that the direct computation of $f(x_j)$, $j = 1, 2, \ldots, J$, requires $O(NJ)$ operations. The problem is to develop algorithms for stable and accurate solution of this problem that use $O(N \log N + J)$ operations. This problem can be viewed as a generalization of FFT.

We consider two algorithms for solution of this problem.

Algorithm 1 employs highly localized kernels of the form

$$K_N(x) = \sum_{j=-(1+\tau)N}^{(1+\tau)N} \varphi\Big(\frac{j}{N}\Big)\, e^{2\pi i j x},$$

where the cutoff function φ is smooth, $\varphi = 1$ on $[-1, 1]$, and supp $\varphi \subset [-1-\tau, 1+\tau]$, for some $\tau > 0$. Clearly, the above kernel K_N reproduces the set T_N all trigonometric polynomials, i.e. $K_N * f = f$ for $f \in T_N$. Let $\mathcal{X}$ be a set of M (sufficiently large) equally spaced grid points on $\mathbb{T} = \mathbb{R}/2\pi\mathbb{N}$. Then the approximating operator takes the form

$$f(x) \approx \sum_{\xi \in \mathcal{X}: \rho(\xi, x) \leq \delta} M^{-1} K_N(x - \xi) f(\xi),$$

where $\delta > 0$ is a small parameter and $\rho(x, y)$ is the distance on $\mathbb{T}$. Here the superb localization of the kernel $K_N(x)$ plays a critical role. In developing Algorithm 1 we adapt some ideas from [**?**] and [**?**].

Algorithm 2 also uses approximation from shifts of a very well localized kernel, but it approximates well f on the frequency side. Our presentation of Algorithm 2 is based on [**?**], [**?**], and [**?**].

Both algorithms heavily depend on the application of FFT.

This thesis is organized as follows. Subsection **??** contains a description of FFT and highly localized trigonometric kernels which are main ingredients for our algorithms. In Chapter 2 we develop Algorithm 1 for fast and accurate evaluation of trigonometric polynomials. Chapter 3 contains a detailed description of Algorithm 2. In Appendix A we place a description of our MATLAB code that realizes Algorithm 1.

1.2 Background

In this subsection we collect some basic well known facts that will be needed in developing algorithms in Chapters 2 and 3.

Fourier Analysis Definitions

Here we establish some definitions from Fourier Analysis.

Definition 1.1. *The Fourier Transform of a function $f \in L^1(\mathbb{R})$ is defined by*

$$\mathcal{F}(f)(\xi) = \hat{f}(\xi) := \int_{\mathbb{R}} f(x) e^{-2\pi i \xi x} dx.$$

Definition 1.2. *The Inverse Fourier Transform of a function $g \in L^1(\mathbb{R})$ is defined by*

$$\mathcal{F}^{-1}(g)(x) = \check{g}(x) := \int_{-\infty}^{\infty} g(\xi) e^{2\pi i \xi x} d\xi.$$

Definition 1.3. *Convolution is defined for 1-periodic functions as follows,*

$$f * g(x) = \int_0^1 f(y) g(x-y)\ dy = \int_\alpha^{\alpha+1} f(y) g(x-y)\ dy$$

for any $\alpha \in \mathbb{R}$.

Definition 1.4. *The Schwarz Space, $\mathcal{S}(\mathbb{R})$ is defined to be*

$$\mathcal{S}(\mathbb{R}) = \{f \in C^\infty(\mathbb{R}) : \|F\|_{\alpha,\beta} < \infty \ \forall\ \alpha, \beta \in \mathbb{Z}_+\}$$

where

$$\|F\|_{\alpha,\beta} = \sup_{x \in \mathbb{R}} |x^\alpha\ \frac{d^\beta f}{dx^\beta}(x)|.$$

The Fourier inversion theorem asserts that if $f \in \mathcal{S}(\mathbb{R})$, then $\hat{f} \in \mathcal{S}(\mathbb{R})$ and

$$f(x) = \mathcal{F}^{-1}(\hat{f})(x). \tag{1.3}$$

Observe that the assumption $f \in \mathcal{S}(\mathbb{R})$ above can be considerably weakened. For instance, (**??**) holds if $|f(x)| \leq c(1+|x|)^{-1-\varepsilon}$ and $|\hat{f}(\xi)| \leq c(1+|\xi|)^{-1-\varepsilon}$ for some $\varepsilon > 0$.

Definition 1.5. *The Discrete Fourier Transform is given by*

$$\hat{f}[j] := \frac{1}{N} \sum_{\nu=0}^{N-1} f[\nu] e^{-j\nu \frac{2\pi i}{N}} \tag{1.4}$$

It is easy to see that the inversion of the Discrete Fourier Transform takes the form

$$f[j] = \sum_{\nu=0}^{N-1} \hat{f}[\nu] e^{j\nu \frac{2\pi i}{N}}. \tag{1.5}$$

This can be seen in the description of the FFT below.

We will also need the **Poisson Summation Formula:** If as above, f is measurable, $|f(x)| \leq c(1+|x|)^{-1-\varepsilon}$, and $|\hat{f}(\xi)| \leq c(1+|\xi|)^{-1-\varepsilon}$ for some $\varepsilon > 0$, then

$$\sum_{n\in\mathbb{Z}} f(n) = \sum_{n\in\mathbb{Z}} \hat{f}(n). \tag{1.6}$$

We refer the reader to [**?**] and [**?**] as general references for Fourier Analysis.

Fast Fourier Transform

The Discrete Fourier Transform (DFT) is commonly used in Computational mathematics and Engineering, and in particular, Signal Processing. Consider the inverse Discrete Fourier Transform

$$f[j] = \sum_{\nu=0}^{N-1} \hat{f}[\nu] e^{j\nu \frac{2\pi i}{N}}, \quad j = 0, 1, \ldots, N-1. \tag{1.7}$$

Here $f[j]$, $j = 0, 1, \ldots, N-1$, are the values of a discrete signal f and $\hat{f}[j]$, $j = 0, 1, \ldots, N-1$, are its Fourier coefficients. The problem is for given Fourier coefficients $\{\hat{f}[j]\}$ to compute $\{f[j]\}$ or vice versa given $\{f[j]\}$ compute $\{\hat{f}[j]\}$.

Clearly, direct computation using (**??**) requires $\mathcal{O}(N^2)$ operations. This becomes quite expensive when repeated many times.

The Fast Fourier Transform is an algorithm that computes $\{f[j]\}_{j=0}^{N}$ for given $\{\hat{f}[j]\}_{j=0}^{N}$ using only $\mathcal{O}(N \log N)$ operations. This method was discovered by Gauss in 1805 and then rediscovered thanks to Cooley and Tukey (1965). The Fast Fourier Transform is one of the most influential algorithms in history. In this section, we describe the Fast Fourier Transform for discrete signals of $N = 2^k, k \in \mathbb{N}$, points. This algorithm plays a critical role in both methods examined by this thesis.

We next describe FFT and prove the following

Theorem 1.6. *The Fast Fourier Transform applied to a discrete function of length* $N = 2^k, k \in \mathbb{N}$*, uses* $\mathcal{O}(N \log N)$ *operations.*

To begin, we introduce the compact notation $w := e^{\frac{2\pi i}{N}}$ in (**??**), leading to

$$f[j] = \sum_{\nu=0}^{N-1} \hat{f}[\nu] e^{j\nu \frac{2\pi i}{N}} = \sum_{\nu=0}^{N-1} \hat{f}[\nu] w^{j\nu} \quad j = 0, 1, \ldots, N-1. \tag{1.8}$$

We now define the vectors $e_k := (1, w^k, \ldots, w^{k(N-1)})^T$ for $k = 0, \ldots, N-1$. Then (**??**) can be rewritten as

$$\begin{bmatrix} f[0] \\ f[1] \\ f[2] \\ \vdots \\ f[N-1] \end{bmatrix} = \hat{f}[0] \begin{bmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} + \hat{f}[1] \begin{bmatrix} 1 \\ w \\ w^2 \\ \vdots \\ w^{N-1} \end{bmatrix} + \cdots + \hat{f}[N-1] \begin{bmatrix} 1 \\ w^{N-1} \\ w^{2(N-1)} \\ \vdots \\ w^{(N-1)(N-1)} \end{bmatrix}$$
$$= \hat{f}[0]e_0 + \hat{f}[1]e_1 + \cdots + \hat{f}[N-1]e_{N-1}.$$

Note that, if $j \neq k$, then

$$\langle e_j, e_k \rangle = \sum_{\nu=0}^{N-1} w^{j\nu} w^{-k\nu} = \sum_{\nu=0}^{N-1} w^{(j-k)\nu} = \frac{1 - w^{(j-k)N}}{1 - w^{j-k}} = 0, \quad \text{since} \quad w^N = 1,$$

and

$$\langle e_j, e_j \rangle = \sum_{\nu=0}^{N-1} 1 = N.$$

Therefore, the vectors $e_0, e_1, \ldots, e_{N-1}$ are orthogonal. We now take inner product with E_j on both sides of (**??**) and use the above identities to obrain

$$\langle f, e_j \rangle = \hat{f}[j] \langle e_j, e_j \rangle = N \hat{f}[j].$$

This implies

$$\hat{f}[j] = \frac{1}{N} \langle f, e_j \rangle = \frac{1}{N} \sum_{\nu=0}^{N-1} f[\nu] w^{-j\nu} = \frac{1}{N} \sum_{\nu=0}^{N-1} f[\nu] e^{-2\pi i \frac{j\nu}{N}},$$

which confirms (**??**).

We next describe the Fast Fourier Transform. Assume $N = 2^k$ for some $k \in \mathbb{N}$. Then, from (**??**)

$$\begin{aligned} N\hat{f}[n] &= \sum_{j=0}^{N/2-1} f[2j]w^{-2nj} + \sum_{j=0}^{N/2-1} f[2j+1]w^{-n(2j+1)} \\ &= \sum_{j=0}^{N/2-1} f[2j]w^{-2nj} + w^{-n} \sum_{j=0}^{N/2-1} f[2j+1]w^{-2nj} \end{aligned}$$

for $n = 0, 1, \ldots, \frac{N}{2}$, and

$$\begin{aligned} N\hat{f}[N/2+n] &= \sum_{j=0}^{N/2-1} f[2j]w^{-2(\frac{N}{2}+n)j} + \sum_{j=0}^{N/2-1} f[2j+1]w^{-(\frac{N}{2}+n)(2j+1)} \\ &= \sum_{j=0}^{N/2-1} f[2j]w^{-2nj} - w^{-n} \sum_{j=0}^{N/2-1} f[2j+1]w^{-2nj} \end{aligned}$$

for $n = 0, 1, \ldots, \frac{N}{2}$, using that

$$w^{\frac{N}{2}} = e^{\frac{-2\pi i}{N}\frac{N}{2}} = e^{-\pi i} = -1.$$

Thus we need only compute the odds and evens a total of N multiplications,

$$\sum_{j=0}^{N/2-1} f[2j]w^{-2nj} \quad \text{and} \quad \sum_{j=0}^{N/2-1} f[2j+1]w^{-2nj} \tag{1.9}$$

to find the discrete Fourier coefficients. We now note that these are of the same form as the Discrete Fourier Transform. Thus we break each of these two sums into their odds and evens, that is,

$$\sum_{j=0}^{N/2-1} f[2j]w^{-2nj} = \sum_{j=0}^{N/4-1} f[4j]w^{-4nj} + w^{-n} \sum_{j=0}^{N/4-1} f[4j+2]w^{4nj}$$

and

$$\sum_{j=0}^{N/2-1} f[2j+1]w^{-2nj} = \sum_{j=0}^{N/4-1} f[4j+1]w^{-4nj} + w^{-n} \sum_{j=0}^{N/4-1} f[4j+3]w^{4nj}$$

and so forth. We repeat this k times since $N = 2^k$.

Note that if $k = 1$ the discrete Fourier Transform (**??**) takes the form

$$\hat{f}[0] = f[0] + f[1], \quad \hat{f}[1] = f[0] - f[1]. \tag{1.10}$$

The final step in the FFT algorithm is to reverse the above process.

Proof of Theorem **??**. We will only count the number of multiplications required by FFT. Denote by $\mathcal{P}(N)$ the minimum number of multiplications needed to compute the discrete Fourier coefficients $\{\hat{f}[j]\}_{j=0}^{N-1}$. We claim that

$$\mathcal{P}(2^k) \leq 2^k(k-1), \quad \forall k \geq 1. \tag{1.11}$$

This estimate follows from the following lemma.

Lemma 1.7. *For any* $k \geq 1$

$$\mathcal{P}(2^k) \leq 2\mathcal{P}(2^{k-1}) + 2^k. \tag{1.12}$$

Proof. By the description of FFT above it is clear that the computation of the discrete Fourier coefficients $\{\hat{f}[j]\}_{j=0}^{N-1}$ reduces to the computation of the sums in (**??**), which requires $2\mathcal{P}(2^{k-1})$ multiplications. Also, one needs 2^k additional multiplications for total of $2\mathcal{P}(2^{k-1}) + 2^k$ multiplications. □

To prove (**??**) we proceed inductively. From (**??**) we have $\mathcal{P}(1) = 0$, i.e. (**??**) holds for $k = 1$. Assume that (**??**) holds for some $k \geq 1$. Then by (**??**) it follows that $\mathcal{P}(2^{k+1}) \leq 2\mathcal{P}(2^k) + 2^{k+1} \leq 2 \cdot 2^k(k-1) + 2^{k+1} = 2^{k+1}k$, which implies (**??**). □

Highly Localized Trigonometric Kernels

In this section we develop the concept of highly localized trigonometric kernels. These kinds of kernels will be our main vehicle in developing our Algorithm 1 in Chapter 2.

Let $\phi \in C^\infty(\mathbb{R})$ with supp $\phi \subset [-a, a]$ for some $a > 0$. Now define the kernel

$$K_N(x) := \sum_{n=-\infty}^{\infty} \phi\Big(\frac{n}{N}\Big) e^{2\pi i n x}. \tag{1.13}$$

Clearly, $K_N(x)$ is a trigonometric polynomial with deg $K_N(x) \leq \lfloor aN \rfloor$.

Theorem 1.8 (Localization of the Kernels)**.** *Let* $K_N(x)$ *be the kernel from* (**??**). *Then for any* $\sigma > 0$ *there exists a constant* $c_{\sigma,a} > 0$ *such that*

$$|K_N(x)| \leq \frac{c_{\sigma,a} N}{\Big(1 + N|x|\Big)^\sigma}, \quad \forall\ x \in [-1/2, 1/2]. \tag{1.14}$$

Proof. With ϕ from the definition of K_N in (**??**) define, $\hat{f}(\xi) := \phi\big(\frac{\xi}{N}\big)e^{2\pi i\xi x}$, which is the Fourier transform of some function f. Then by the Fourier inversion formula (twice),

$$\begin{aligned} f(y) &= \int_{\mathbb{R}} \phi\Big(\frac{\xi}{N}\Big)e^{2\pi i\xi x}e^{2\pi i\xi y}\ d\xi = \int_{\mathbb{R}} \phi\Big(\frac{\xi}{N}\Big)e^{2\pi i\xi(x+y)}\ d\xi \\ &= N\int_{\mathbb{R}} \phi\Big(\frac{\xi}{N}\Big)e^{2\pi i\frac{\xi}{N}(x+y)N}\ d\frac{\xi}{N} = N\check{\phi}(N(x+y)). \end{aligned}$$

Here $\check{\phi}$ stands for the inverse Fourier transform of Φ. Now, applying the Poisson Summation Formula we obtain

$$\sum_{n\in\mathbb{Z}} f(n) = \sum_{n\in\mathbb{Z}} \hat{f}(n) = \sum_{n\in\mathbb{Z}} \phi\Big(\frac{n}{N}\Big)e^{2\pi inx} = K_N(x).$$

Further, by the previous work,

$$\sum_{n\in\mathbb{Z}} f(n) = N\sum_{n\in\mathbb{Z}} \check{\phi}(N(x+n)).$$

Therefore,

$$K_N(x) = N\sum_{n\in\mathbb{Z}} \check{\phi}(N(x+n)). \tag{1.15}$$

As $\phi \in C^\infty(\mathbb{R})$ with compact support, then $\check{\phi}(x) = \int_{\mathbb{R}} \phi(\xi)e^{2\pi i\xi x}d\xi$ is in the Schwartz Class $\mathcal{S}(\mathbb{R})$. Hence, for any $\sigma > 0$ there exists a constant $c_{\sigma,a} > 0$ such that

$$|\check{\phi}(x)| \le \frac{c_{\sigma,a}}{(1+|x|)^\sigma}, \quad x \in \mathbb{R}.$$

Using this and (**??**),

$$\begin{aligned} |K_N(x)| &\le N\sum_{n\in\mathbb{Z}} \Big|\check{\phi}(N(x+n))\Big| \le N\sum_{n\in\mathbb{Z}} \frac{c_{\sigma,a}}{(1+|N(x+n)|)^\sigma} \\ &= \frac{c_{\sigma,a}N}{(1+N|x|)^\sigma} + N\sum_{n\in\mathbb{Z}\setminus\{0\}} \frac{c_{\sigma,a}}{(1+|N(x+n)|)^\sigma}. \end{aligned}$$

We now estimate $\sum_{n\in\mathbb{Z}\setminus\{0\}} \frac{c_{\sigma,a}}{(1+|N(x+n)|)^\sigma}$. Note that $|N(x+n)| \ge N(|n|-\frac{1}{2})$ if $|x| \le \frac{1}{2}$. Hence,

$$\begin{aligned} \sum_{n\in\mathbb{Z}\setminus\{0\}} \frac{c_{\sigma,a}}{(1+|N(x+n)|)^\sigma} &\le \sum_{n\in\mathbb{Z}\setminus\{0\}} \frac{c_{\sigma,a}}{\big(N(|n|-\frac{1}{2})\big)^\sigma} \\ = \frac{2c_{\sigma,a}}{N^\sigma}\sum_{n=1}^{\infty} \frac{1}{(n-\frac{1}{2})^\sigma} &\le \frac{2c_{\sigma,a}}{N^\sigma}\Big(\frac{1}{(\frac{1}{2})^\sigma} + \int_1^\infty \frac{1}{(x-\frac{1}{2})^\sigma}\ dx\Big). \end{aligned}$$

Assuming $\sigma > 1$ we get

$$\sum_{n\in\mathbb{Z}\setminus\{0\}} \frac{c_{\sigma,a}}{(1+|N(x+n)|)^\sigma} \le \frac{2c_{\sigma,a}}{N^\sigma}\Big(2^\sigma + \frac{1}{(\frac{1}{2})^{\sigma-1}(\sigma-1)}\Big) \le \frac{c'_{\sigma,a}}{N^\sigma} \le \frac{\tilde{c}_{\sigma,a}}{(1+N|x|)^\sigma}.$$

Putting the above together we obtain

$$\begin{aligned} |K_N(x)| &\le \frac{c_{\sigma,a}N}{(1+N|x|)^\sigma} + N\sum_{n\in\mathbb{Z}\setminus\{0\}} \frac{c_{\sigma,a}}{(1+|N(x+n)|)^\sigma} \\ &\le \frac{c_{\sigma,a}N}{(1+N|x|)^\sigma} + \frac{\tilde{c}_{\sigma,a}N}{(1+N|x|)^\sigma} \le \frac{c'_{\sigma,a}N}{(1+N|x|)^\sigma}. \end{aligned}$$

which confirms (**??**). □

Chapter 2

The Algorithm for Evaluation of Band Limited Functions

We begin with a top down description of the algorithm. After the mathematical idea has been seated, we will work our way from bottom up to show how the algorithm can be implemented.

2.1 The Idea of the Algorithm

The Function to be Approximated

To start things off, we let $f \in \mathbb{T}_N(\mathbb{R})$. This means that f is a trigonometric polynomial of degree N. More specifically,

$$f(x) = \sum_{n=-N}^{N} \hat{f}\Big(\frac{n}{N}\Big)\ e^{2\pi i n x}.$$

This is the function we are interested in approximating at a large number of scattered points.

Reproducing Kernel

To achieve this, we begin by describing a kernel $K_N(x)$, and defining $M := \lfloor (1+\tau)N \rfloor$,

$$K_N(x) := \sum_{j=-\lfloor (1+\tau)N \rfloor}^{\lfloor (1+\tau)N \rfloor} \varphi\Big(\frac{j}{N}\Big)\ e^{2\pi i j x} := \sum_{j=-M}^{M} \varphi\Big(\frac{j}{N}\Big)\ e^{2\pi i j x}.$$

In otherwords, $K_N(x)$ is a trignomentric polynomial of degree M and has a Fourier Transform of $\varphi(\xi)$. We now make the requirement that $\varphi(\xi) = 1,\ \forall\ \ \xi \in [-1, 1]$.

With this requirement in place, we examine the convolution,

$$\widehat{K_N * f}(\frac{j}{N}) = \hat{K_N}(\frac{j}{N}) \cdot \hat{f}(\frac{j}{N}) = \begin{cases} \hat{f}(\frac{j}{N}) & if \ -N \leq j \leq N \\ 0 & if \ |j| > N. \end{cases}$$

This in turn implies that convolution by K_N preserves polynomials of degree N due to the requirement $\varphi(\xi) = 1 \ \forall \ \xi \in [-1, 1]$; convolution in time domain is equivalent to multiplication in frequency domain. Thus we arrive at this statement,

$$K_N * f \ (x) = \int_{-\frac{1}{2}}^{\frac{1}{2}} K_N(x-y) \ f(y) \ dy = \int_{-\frac{1}{2}}^{\frac{1}{2}} K_N(y) \ f(x-y) \ dy = f(x).$$

Exact Quadrature Formula for Trigonometric Polynomials

Theorem 2.1. *Let $f \in \mathbb{T}_{L-1}$. Then, the quadrature formula*

$$\int_0^1 f(x) \ dx = \frac{1}{L} \sum_{j=0}^{L-1} f(\frac{j}{L})$$

is exact.

Proof. We begin with definition of the Fourier Transform for band-limited one-periodic functions,

$$\hat{f}(\xi) := \int_0^1 f(x) \ e^{-2\pi i \xi x} \ dx.$$

Thus

$$\hat{f}(0) := \int_0^1 f(x) \ dx.$$

We now examine,

$$\frac{1}{L} \sum_{j=0}^{L-1} f(\frac{j}{L}) = \frac{1}{L} \sum_{j=0}^{L-1} \sum_{n=0}^{L-1} \hat{f}(\frac{n}{L}) e^{\frac{2\pi i j n}{L}} = \frac{1}{L} \sum_{n=0}^{L-1} \hat{f}(\frac{n}{L}) \sum_{j=0}^{L-1} e^{\frac{2\pi i j n}{L}}.$$

Looking at the sum,

$$\sum_{j=0}^{L-1} e^{\frac{2\pi i j n}{L}} = \frac{1 - e^{2\pi i n}}{1 - e^{\frac{2\pi i n}{L}}} = \begin{cases} 0 & if \ n = 1, 2, \ldots, L-1 \\ L & if \ n = 0. \end{cases}$$

Thus only the $n = 0$ term is non-trivial so

$$\frac{1}{L}\sum_{j=0}^{L-1} f(\frac{j}{L}) = \hat{f}(0) = \int_0^1 f(x)\ dx.$$

□

Convolution as a Quadrature Formula

We now examine $K_N(x-y)\ f(y)$. This new function is a trigonometric polynomial of degree $M+N$ with respect to y. Thus, the following quadrature formula holds with no error:

$$\begin{aligned} K_N * f\ (x) &= \int_{\frac{-1}{2}}^{\frac{1}{2}} K_N(x-y)\ f(y)\ dy \\ &= \frac{1}{M+N}\sum_{j=0}^{M+N-1} K_N\Big(x - \frac{j}{M+N}\Big)\ f\Big(\frac{j}{M+N}\Big) \\ &= f(x) \end{aligned}$$

.

This means that if we are interested in calculating f at any x we need only $M+N$ samples of f and a suitable K_N that is easy to calculate at any value. Infact, if we impose the localization principle from the introduction, we may cut off terms where $\Big|K_N\Big(x - \frac{j}{M+N}\Big)\Big| < \epsilon$ where ϵ is chosen to meet our error requirement. This brings us into defining K_N explicitly.

Localization of the Kernel

Earlier we defined $K_N(x)$ as,

$$K_N(x) := \sum_{j=-M}^{M} \varphi\Big(\frac{j}{N}\Big)\ e^{2\pi i j x}.$$

Using this, we see that we need only define $\varphi(\xi)$ to define $K_N(x)$. Earlier φ was already restricted to be 1, $\forall\ \ \xi \in [-1, 1]$. So the only thing left is to define $\varphi(\xi)$ on

$[-1-\tau,-1]$ and $[1,1+\tau]$. If we let φ be symmetric about 0 then we see that

$$K_N(x) := \sum_{j=-M}^{M} \varphi\Big(\frac{j}{N}\Big)\ e^{2\pi ijx} = 1 + 2\sum_{j=1}^{M} \varphi\Big(\frac{j}{N}\Big)\cos 2\pi jx.$$

This provides us with a means to directly compute $K_N(x)$ from the sampled points of $\varphi(\xi)$ in half as many computations.

We then use the localization principle from the introduction to make $K_N(x)$ well localized. To this end φ must be contained in $C^\infty(\mathbb{R})$ with supp $\varphi \subset [-1-\tau, 1+\tau]$. Thus we need to smoothly go from 1 to 0 on $[1,1+\tau]$ and $[-1-\tau,-1]$. Since φ is symmetric, we only examine the part of φ on the interval $[1,1+\tau]$. We call this restricted function $h(\xi)$. To create $h(\xi)$, a smooth transition from 1 to 0, we begin by integrating bump functions.

Integrating Bump Functions for Smooth Transitions

We arrive at $h(\xi)$ by integrating smooth bump functions, $g(y)$ with supp $g \subset [0,\tau]$ and then flipping them. Specifically,

$$h^*(\xi) = \frac{1}{\int_0^\tau g(y)\ dy} \int_0^\xi g(y)\ dy$$

and

$$h(\xi) = h^*(\tau-\xi) = \frac{1}{\int_0^\tau g(y)\ dy} \int_0^{\tau-\xi} g(y)\ dy.$$

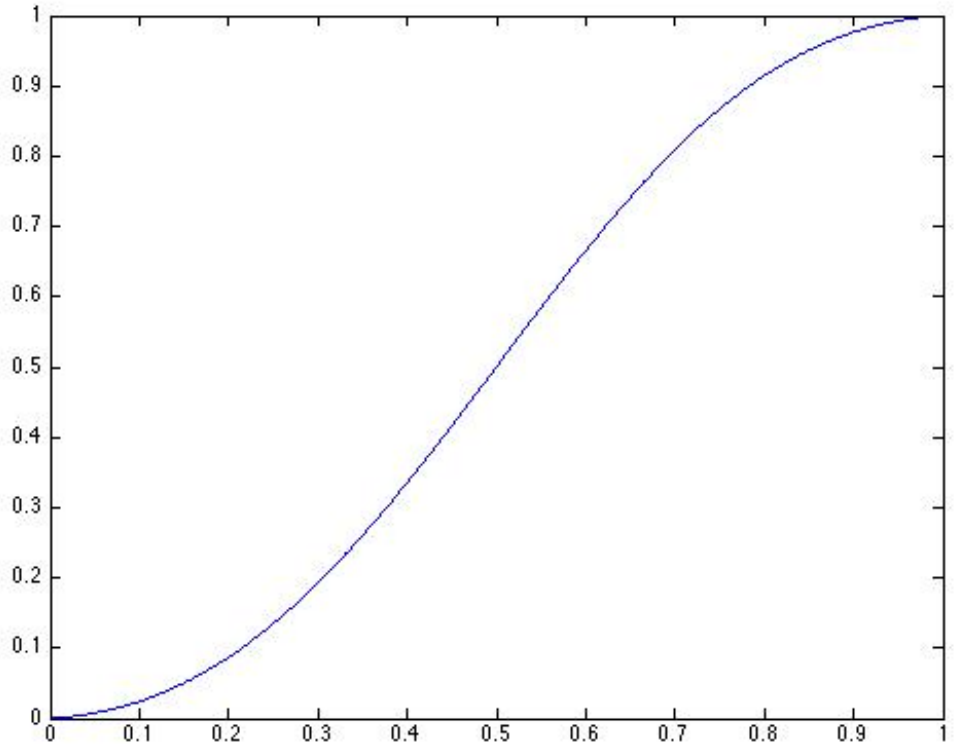

Figure 2.1 A Smooth Transition from 0 to 1 with $\tau = 1$

This creates our smooth transition from 1 to 0. We then attach this on both sides of our block of values 1 on $[-1, 1]$ to extend the support to $[-1-\tau, 1+\tau]$ and arrive at our $\varphi(\xi)$ that satisfies our algorithm all the way up to approximating f at any x.

Finally we arrive at

$$\varphi(\xi) := \begin{cases} h^*(\xi + 1 + \tau) & x \in [-1-\tau, -1) \\ 1 & x \in [-1, 1] \\ h(\xi - 1) & x \in (1, 1+\tau] \\ 0 & \text{otherwise.} \end{cases}$$

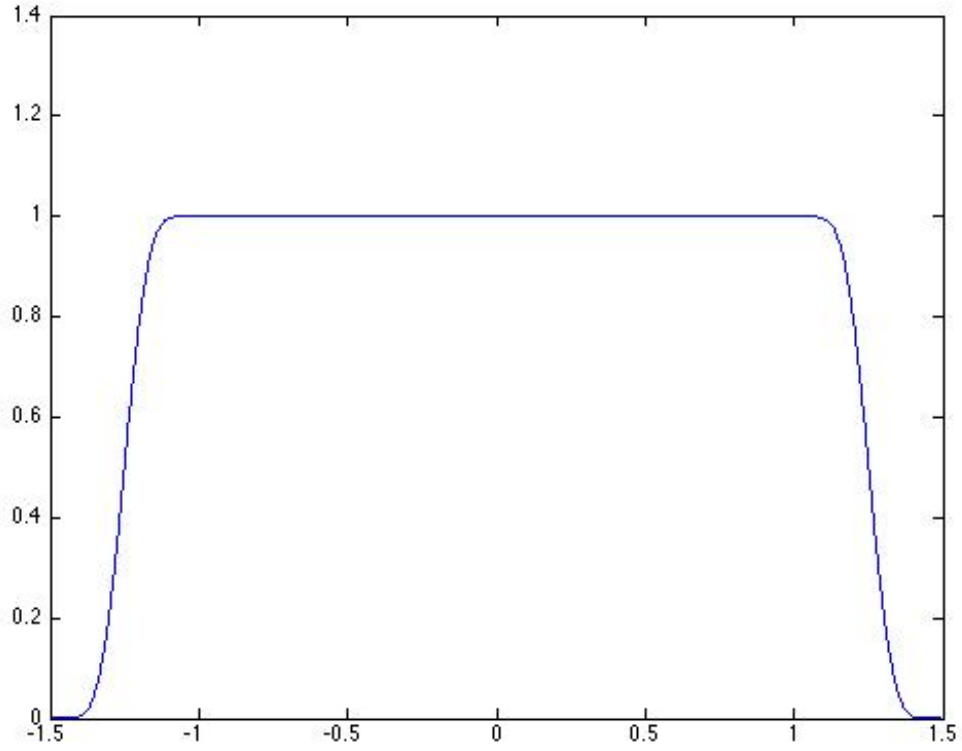

Figure 2.2 Example of φ with $\tau = \frac{1}{2}$

Thus we have φ defined and therefore,

$$K_N = 1 + 2\sum_{j=1}^{M} \varphi\Big(\frac{j}{N}\Big) \cos 2\pi jx.$$

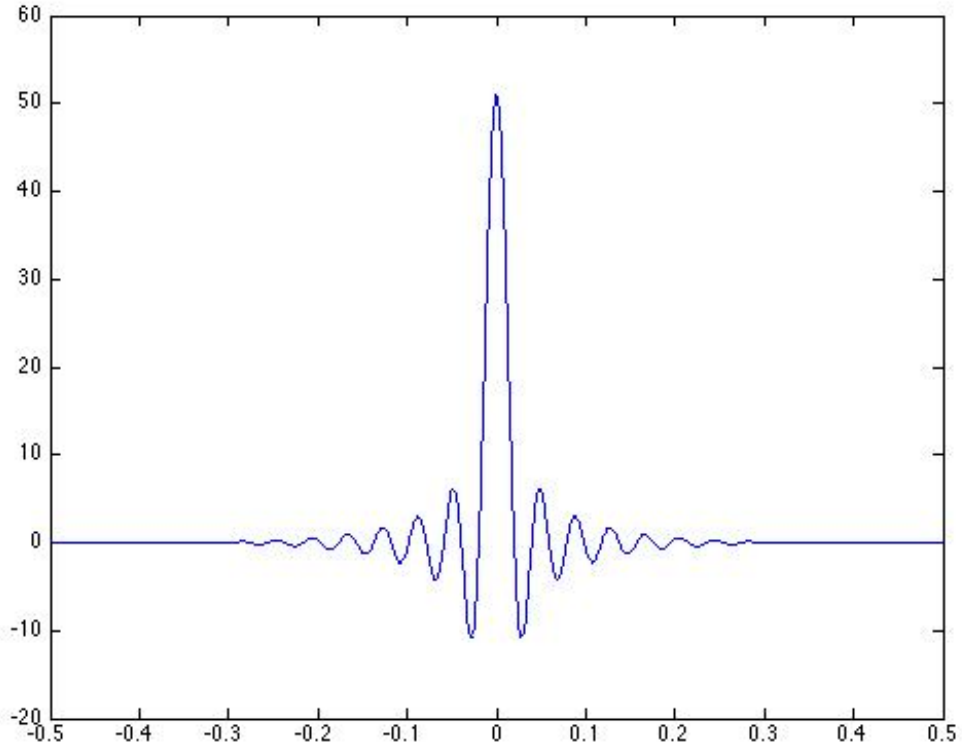

Figure 2.3 Example of K_{30} with $\tau = \frac{1}{2}$

We now work our way back up in the next section, implementation.

2.2 Implementation

Example Bump Functions

We begin our implementation of the algorithm by a few choices of $g(y)$, the bump functions to be integrated to obtain $h(\xi)$:

$$g_1(y) = (y(\tau - y))^b$$
$$g_2(y) = \left(\sin \frac{y\pi}{\tau}\right)^b$$
$$g_3(y) = e^{\frac{-1}{y(\tau-y)}}$$
$$g_4(y) = e^{b\sqrt{y(\tau-y)}}$$

for some b's.

Example Smooth Transitions

We then calculate the respective h^*'s and h's:

$$h_1^*(\xi) = \frac{1}{\int_0^\tau (y(\tau-y))^b \, dy} \int_0^\xi (y(\tau-y))^b \, dy$$
$$h_2^*(\xi) = \frac{1}{\int_0^\tau (\sin \frac{y\pi}{\tau})^b \, dy} \int_0^\xi (\sin \frac{y\pi}{\tau})^b \, dy$$
$$h_3^*(\xi) = \frac{1}{\int_0^\tau e^{\frac{-1}{y(\tau-y)}} \, dy} \int_0^\xi e^{\frac{-1}{y(\tau-y)}} \, dy$$
$$h_4^*(\xi) = \frac{1}{\int_0^\tau e^{b\sqrt{y(\tau-y)}} \, dy} \int_0^\xi e^{b\sqrt{y(\tau-y)}} \, dy$$

and

$$h_1(\xi) = \frac{1}{\int_0^\tau (y(\tau-y))^b \, dy} \int_0^{\tau-\xi} (y(\tau-y))^b \, dy$$
$$h_2(\xi) = \frac{1}{\int_0^\tau (\sin \frac{y\pi}{\tau})^b \, dy} \int_0^{\tau-\xi} (\sin \frac{y\pi}{\tau})^b \, dy$$
$$h_3(\xi) = \frac{1}{\int_0^\tau e^{\frac{-1}{y(\tau-y)}} \, dy} \int_0^{\tau-\xi} e^{\frac{-1}{y(\tau-y)}} \, dy$$
$$h_4(\xi) = \frac{1}{\int_0^\tau e^{b\sqrt{y(\tau-y)}} \, dy} \int_0^{\tau-\xi} e^{b\sqrt{y(\tau-y)}} \, dy.$$

Note that $h_4(\xi)$ does not necessarily end at 0. This means it is not an ideal transition. However, it still serves as a valid function since we may assume the next sample to take value 0, and any continuous function with compact support can be approximated by elements from the Schwarz Class.

Trapezoidal Quadrature Formula

We begin by letting $\tilde{M} \in \mathbb{N}$ be large. Then for $t_k := \frac{k}{\tilde{M}}\tau$, we approximate by,

$$\int_0^{t_k} h(x)\, dx \approx \frac{\tau}{\tilde{M}}\Big(\sum_{j=0}^{k} h(t_j) - \frac{1}{2}h(0) - \frac{1}{2}h(t_k)\Big).$$

Approximating the Smooth Transition

The first step in is evaluating the smooth transitions. To this end, we either calculate the integral directly (in the case of $h_1(\xi)$) or use a quadrature formula. For instance, the trapezoidal with a high number of sample points, $\tilde{M}$, from above.

From here we calculate $h(\frac{j}{N})$ for $j = 0, 1, 2, \ldots \lfloor \tau N \rfloor$; these are equispaced samples. We attach these onto an array of size N where the values are all 1. This makes an array of samples of the positive side of φ. Since φ is symmetric this is all we need.

With φ in hand, we now compute the sampled points of $K_N(x)$. To do this we either tack on an array of 0's to our φ array and compute the inverse fourier transform (preferably using the inverse fast fourier transform) or we compute the points directly using the formula:

$$K_N(x) = 1 + 2\sum_{j=1}^{M} \varphi\Big(\frac{j}{N}\Big) \cos 2\pi j x.$$

We now have a large number of sampled points of $K_N(x)$ to work with, let this number be W many points. $K_N(x)$ should be well localized thanks to our selection of φ. Thus we may cut off the tails of $K_N(x)$. I.e., we restrict the support of $K_N(x)$ to $[-\delta, \delta]$ when $\Big|K_N(x)\Big| < \epsilon \;\forall\; x \in [-\frac{1}{2}, -\delta) \cup (\delta, \frac{1}{2}]$ for δ dependent on ϵ and ϵ is chosen to meet our error requirement.

Approximation of the Kernel Using Lagrange Interpolation

From here we approximate $K_N(x)$ at any x using Lagrange Interpolation on the remaining points. Let $\Big(K_N\Big(\frac{n}{W}\Big)\Big)_{n=-a}^{a}$ be the sampled points of $K_N(x)$ after trun-

cation. Then our approximating function of $K_N(x)$ using Lagrange Interpolation is given by

$$\widetilde{K}_N(x) := \sum_{j=-a}^{a} K_N\Big(\frac{j}{W}\Big) \prod_{\substack{-a\le n\le a \\ n\neq j}} \frac{x-\frac{n}{W}}{\frac{j}{W}-\frac{n}{W}}.$$

Finalizing the Kernel

Now that we have an approximate K_N at any x, we may apply our original quadrature formula for approximation of f at any x using only the sampled points of f. Let $x_j = \frac{j}{M+N}$, then:

$$f(x) \approx \frac{1}{M+N} \sum_{j=0}^{M+N-1} \widetilde{K}_N(x-x_j)\ f(x_j).$$

And since we have supp $K_N(x) \subset [-\delta, \delta]$, we further restrict this sum:

$$f(x) \approx \frac{1}{M+N} \sum_{\substack{j=0 \\ |x-x_j|\le\delta}}^{M+N-1} \widetilde{K}_N(x-x_j)\ f(x_j).$$

Chapter 3

Nonequispaced Fast Fourier Transform: Algorithm 2

In the chapter, we examine another algorithm for fast and accurate evaluation of high degree trigonometric polynomials at many scattered points. This algorithm was proposed by A. Dutt and V. Rokhlin in [**?**] and further developed by other authors, see e.g. [**?**]. Our goal is to compare the performance of the algorithm of Dutt and Rokhlin with the algorithm we developed in Chapter 2. For convenience we shall use the notation from [**?**].

Similarly as before we are interested in the following

Problem. Given the Fourier coefficients $\{\hat{f}_k\}_{k=-\frac{N}{2}}^{\frac{N}{2}-1}$, N even, compute the values of the trigonometric polynomial

$$f(x) = \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} \hat{f}_k e^{-2\pi i k x} \tag{3.1}$$

at many scattered points $\{x_j\}_{j=1}^{M}$.

3.1 The idea of the algorithm

Start by defining $I_N := \{k : -\frac{N}{2} \le k < \frac{N}{2}\}$. Then $f(x) = \sum_{k\in I_N} \hat{f}_k e^{-2\pi i k x}$. The idea is to approximate $f(x)$ using a linear combination of the shifts of very well localized 1-periodic function $\tilde{\varphi}$, that is,

$$f(x) \approx s_1(x) := \sum_{j\in I_n} g_j \tilde{\varphi}\Big(x - \frac{j}{n}\Big), \tag{3.2}$$

where $n = \sigma N$ with $\sigma > 1$.

An important component of this method is to define the function $\tilde{\varphi}$ by one-periodization of a very well localized smooth window function φ, i.e.

$$\tilde{\varphi}(x) := \sum_{r\in\mathbb{Z}} \varphi(x+r) = \sum_{r\in\mathbb{Z}} c_k(\tilde{\varphi}) e^{-2\pi i k x},$$

serving a similar role as φ in Chapter 2 and having a similar localization. We will assume that φ is in the Schwartz class $\mathcal{S}(\mathbb{R})$, i.e. $\varphi \in C^\infty(\mathbb{R})$ and for any $k, \ell \geq 1$ there exists a constant $c(k,\ell) > 0$ such that $|\varphi^{(k)}(x)| \leq c(k,\ell)(1+|x|)^{-\ell}$ for $x \in \mathbb{R}$. Above $c_k(\tilde{\varphi})$ is the kth Fourier coefficients of the 1-periodic function $\tilde{\varphi}$. Thus

$$c_k(\tilde{\varphi}) = \int_{\mathbb{T}} \tilde{\varphi}(x) e^{2\pi i k x} dx = \int_0^1 \Big(\sum_{r\in\mathbb{Z}} \varphi(x+r) \Big) e^{2\pi i k x} dx.$$

We now interchange summation and integration and apply a change of variables $u = x + r$ after multiplying by $e^{2\pi i k r} = 1$ to obtain

$$c_k(\tilde{\varphi}) = \sum_{r\in\mathbb{Z}} \int_0^1 \varphi(x+r) e^{2\pi i k (x+r)} dx = \sum_{r\in\mathbb{Z}} \int_r^{r+1} \varphi(u) e^{2\pi i k u} du = \int_{\mathbb{R}} \varphi(u) e^{2\pi i k u} du = \hat{\varphi}(k).$$

Therefore,

$$\begin{aligned} s_1(x) &= \sum_{j\in I_n} g_j \tilde{\varphi}\Big(x - \frac{j}{n}\Big) = \sum_{j\in I_n} g_j \sum_{k\in\mathbb{Z}} c_k(\tilde{\varphi}) e^{-2\pi i k (x - \frac{j}{n})} \\ &= \sum_{k\in\mathbb{Z}} \Big(\sum_{j\in I_n} g_j e^{2\pi i \frac{k}{n} j} \Big) c_k(\tilde{\varphi}) e^{-2\pi i k x}. \end{aligned}$$

We define

$$\hat{g}_k := \sum_{j\in I_n} g_j e^{2\pi i \frac{k}{n} j}, \tag{3.3}$$

and note that $\{\hat{g}_k\}$ is n-periodic. We split the above sum into two

$$s_1(x) = \sum_{k\in\mathbb{Z}} \hat{g}_k c_k(\tilde{\varphi}) e^{-2\pi i k x} = \sum_{k\in I_n} \hat{g}_k c_k(\tilde{\varphi}) e^{-2\pi i k x} + \sum_{k\in\mathbb{Z}\setminus I_n} \hat{g}_k c_k(\tilde{\varphi}) e^{-2\pi i k x},$$

where similarly as above $I_n := \{k : -\frac{n}{2} \leq k < \frac{n}{2}\}$. Using the n-periodicity of $\{\hat{g}_k\}$ we obtain

$$\begin{aligned} \sum_{k\in\mathbb{Z}\setminus I_n} \hat{g}_k c_k(\tilde{\varphi}) e^{-2\pi i k x} &= \sum_{r\in\mathbb{Z}\setminus\{0\}} \sum_{k\in I_n} \hat{g}_{nr+k} c_{k+nr}(\tilde{\varphi}) e^{-2\pi i (k+nr) x} \\ &= \sum_{r\in\mathbb{Z}\setminus\{0\}} \sum_{k\in I_n} \hat{g}_k c_{k+nr}(\tilde{\varphi}) e^{-2\pi i (k+nr) x}. \end{aligned}$$

Hence

$$s_1(x) = \sum_{k\in\mathbb{Z}} \hat{g}_k c_k(\tilde{\varphi}) e^{-2\pi ikx} = \sum_{k\in I_n} \hat{g}_k c_k(\tilde{\varphi}) e^{-2\pi ikx} + \sum_{r\in\mathbb{Z}\setminus\{0\}} \sum_{k\in I_n} \hat{g}_k c_{k+nr}(\tilde{\varphi}) e^{-2\pi i(k+nr)x}.$$

As $\varphi \in \mathcal{S}(\mathbb{R})$ the coefficients $c_k(\tilde{\varphi}) = \hat{\varphi}(k)$ are small for $|k| \geq n$ and hence

$$\Big| \sum_{r\in\mathbb{Z}\setminus\{0\}} \sum_{k\in I_n} \hat{g}_k c_{k+nr}(\tilde{\varphi}) e^{-2\pi i(k+nr)x} \Big| \leq \varepsilon$$

for some "small" constant $\varepsilon > 0$. Thus

$$s_1(x) \approx \sum_{k\in I_n} \hat{g}_k c_k(\tilde{\varphi}) e^{-2\pi ikx}.$$

We also assume that $c_k(\tilde{\varphi}) \neq 0$ for $k \in I_N$. Now, comparing the above with (**??**) suggests to set $\hat{f}_k = \hat{g}_k c_k(\tilde{\varphi})$. More precisely, we set

$$\hat{g}_k := \begin{cases} \frac{\hat{f}_k}{c_k(\tilde{\varphi})} & \text{for } k \in I_N, \\ 0 & \text{for } k \in I_n \setminus I_N. \end{cases}$$

Then the values $\{g_j\}$ can be obtained from (**??**) by

$$g_j = \frac{1}{n} \sum_{k\in I_N} \hat{g}_k e^{-2\pi i \frac{k}{n} j}$$

applying FFT of size n, see §**??**.

The next step is truncate the "long" sum in (**??**). The good localization of $\varphi(x)$ motivates its approximation by

$$\psi(x) = \varphi(x) \mathbb{1}_{[-\frac{m}{n}, \frac{m}{n}]}(x),$$

where $\mathbb{1}_{[\frac{-m}{n}, \frac{m}{n}]}(x)$ is the characteristic function of $[-\frac{m}{n}, \frac{m}{n}]$, and $\frac{m}{n} \ll \frac{1}{2}$. The 1-periodic version $\tilde{\psi}$ of ψ is defined by

$$\tilde{\psi}(x) = \sum_{r\in\mathbb{Z}} \psi(x + r)$$

and we observe that

$$\tilde{\psi}(x) = \psi(x) = \varphi(x) \mathbb{1}_{[-\frac{m}{n}, \frac{m}{n}]}(x) \quad \text{for} \quad |x| \leq 1/2.$$

Thus we arrive at the following approximation

$$f(x) \approx s_1(x) \approx s(x) = \sum_{j \in I_{n,m}(x)} g_j \tilde{\psi}\Big(x - \frac{j}{n}\Big) = \sum_{j \in I_{n,m}(x)} g_j \varphi\Big(x - \frac{j}{n}\Big), \tag{3.4}$$

where

$$I_{n,m}(x) := \Big\{k \in I_n : \Big|x - \frac{k}{n}\Big| \le \frac{m}{n}\Big\} = \{k \in I_n : nx - m \le k \le nx + m\}.$$

One uses the "short" sum on the right in (**??**) to approximate $f(x)$ at arbitrary scattered points $\{x_j\}_{j=1}^m$.

3.2 Description of the algorithm

Here we describe the algorithm for evaluation of $f(x)$ from (**??**) and arbitrary scattered points $\{x_j\}_{j=1}^M \subset \mathbb{R}$.

Input: $N, M \in \mathbb{N}$, $\{\hat{f}_k\}_{k \in I_N} \subset \mathbb{C}$, $\{x_j\}_{j=1}^M \subset \mathbb{R}$.

One first precomputes the coefficients $c_k(\tilde{\varphi}) = \hat{\varphi}(k)$ for $k \in I_N$ and then proceeds as follows:

(1) For $k \in I_N$ compute

$$\hat{g}_k = \frac{\hat{f}_k}{c_k(\tilde{\varphi})} = \frac{\hat{f}_k}{\hat{\varphi}(k)}.$$

(2) For $j \in I_n$ compute by FFT

$$g_j = \frac{1}{n} \sum_{k \in I_N} \hat{g}_k e^{-2\pi i \frac{k}{n} j}.$$

(3) For $j = 1, 2, \ldots, M$ compute

$$f_j = \sum_{j \in I_{n,m}(x_j)} g_j \varphi\Big(x_j - \frac{j}{n}\Big).$$

Output: Approximate values $\{f_j\}_{j=1}^M$.

Remark. The above algorithm requires the development of an algorithm and code for fast and accurate evaluation of $\varphi(x)$ at an arbitrary point $x \in \Big[-\frac{m}{n}, \frac{m}{n}\Big]$ and $\hat{\varphi}(k)$ for $k \in I_N$.

3.3 Window functions and error bound

Several window functions with good localization in space and frequency domain have been proposed:

(a) The Gaussian

$$\varphi(x) = (\pi b)^{-1/2} e^{-\frac{(nx)^2}{b}} \qquad \Big(b := \frac{2\sigma m}{\pi(2\sigma - 1)}\Big),$$
$$\hat{\varphi}(k) = \frac{1}{n} e^{-b\frac{\pi k}{n}}.$$

(b) B-splines

$$\varphi(x) = M_{2m}(nx),$$
$$\hat{\varphi}(k) = \frac{1}{n}\mathrm{sinc}^{2m}(\pi k/n),$$

where M_{2m} is the centered cardinal B-spline, i.e. $M_{2m}(x) := \mathbb{1}_{[-1,1]} * \mathbb{1}_{[-1,1]} * \cdots * \mathbb{1}_{[-1,1]}$ with the convolution applied $2m$ times and $\mathrm{sinc}\ \xi = \frac{\sin \xi}{\xi}$.

(c) Sinc function

$$\varphi(x) = \frac{N(2\sigma - 1)}{2m} \mathrm{sinc}^{2m}\Big(\frac{\pi n x (2\sigma - 1)}{2m}\Big),$$
$$\hat{\varphi}(k) = M_{2m}\Big(\frac{2mk}{(2\sigma - 1)N}\Big).$$

(d) Kaiser-Bessel functions

$$\varphi(x) = \frac{1}{\pi} \begin{cases} \frac{\sinh\left(b\sqrt{m^2 - n^2x^2}\right)}{\sqrt{m^2 - n^2x^2}} & \text{for } |x| \leq \frac{m}{n}, \quad (b := \pi(2 - 1/\sigma)), \\ \frac{\sin\left(b\sqrt{m^2 - n^2x^2}\right)}{\sqrt{m^2 - n^2x^2}} & \text{for } |x| > \frac{m}{n}, \end{cases}$$

$$\hat{\varphi}(k) = \frac{1}{n} \begin{cases} I_0\Big(m\sqrt{b^2 - (2\pi k/n)^2}\Big) & \text{for } -n\Big(1 - \frac{1}{2\sigma}\Big) \leq k \leq n\Big(1 - \frac{1}{2\sigma}\Big), \\ 0 & \text{otherwise}, \end{cases}$$

where I_0 is the modified zero-order Bessel function.

For more details, see [**?**].

The following error estimate has been proved in [**?**] in the case (a) of the Gaussian:

$$|f(x_j) - f_j| \leq 4e^{-\pi m(1-1/(2\sigma-1))} \sum_{k \in I_N} |\hat{f}(k)|.$$

Similar error estimates are established in cases (b) - (d), see [**?**]. The above estimate shows the Nonequispaced Fast Fourier Transform presented above is a viable algorithm for evaluation of high degree trigonometric polynomials.

3.4 Complexity

As was shown in §**??** the Fast Fourier Transform requires $O(n \log n)$ operations. On the other hand, the parameter m is a "small" constant. Therefore, the complexity of the above described algorithm is $O(N \log N + M)$.

It should be pointed out that the direct computation of the values $f(x_j)$, $j = 1, 2, \ldots, M$, of the trigonometric polynomial from (**??**) requires $O(MN)$ operations, which is considerably larger than $O(N \log N + M)$ for large N and M.

Bibliography

[1] J. Berrut, L. Trefethen, Barycentric Lagrange Interpolation, SIAM Review, 46 (2004), 500-517.

[2] A. Dutt, V. Rokhlin, Fast Fourier transforms for nonequispaced data, SIAM J. Sci. Stat. Comput. 14 (1993), 1368–1393.

[3] B. Elbel, G. Steidl, Fast Fourier transform for nonequispaced data, In C. K. Chui and L. L. Schumaker, editors, Approximation Theory IX, Nashville, 1998, Vanderbilt University Press.

[4] L. Grafakos, *Classical Fourier Analysis*, Springer, New York, 2008.

[5] K. Ivanov, P. Petrushev, Fast memory efficient evaluation of spherical polynomials at scattered points, Adv. Comput. Math. 41 (2015), no. 1, 191–230.

[6] K. Ivanov, P. Petrushev, Highly effective stable evaluation of bandlimited functions on the sphere, Numer. Algorithms, 71 (2016), no. 3, 585–611.

[7] J. Keiner, S. Kunis, D. Potts, Using NFFT 3–a software library for various nonequispaced fast Fourier transforms, ACM Trans. Math. Software 36 (2009), no. 4, 19:1–19:30.

[8] E. Stein, G. Weiss, *Fourier analysis on Euclidean spaces*, Princeton University Press, Princeton, New Jessey, 1971.

Appendix A

Code

Here we include some MATLAB code for the method discussed in Chapter 2. Using this code to test the four Kernels with a exponent (where applicable) of 7, a τ of 1, oversampling 30 times, a cut-off of 10^{-9} (decimation of the Kernel), and with f of degree 50: the Square Root Exponential (h_4) had an average error of 5.3×10^{-13}, the Gaussian (h_3) had an average error of 9.0×10^{-11}, the Power Sine (h_2) had an average error of 1.3×10^{-10}, and the Polynomial (h_1) had an average error of 1.7×10^{-10}. These values were found after computing 1000 random points per randomly generated f, 200 times.

A.1 Algorithm 1

```
function [ out ] = Algorithm1( in, hatf, tau, epsilon,
    oversamp, p, b )
%APPROX1 Realization of Algorithm 1
%   in       : points of approximation
%   hatf     : fourier coefficients of the approximated
    function
%   tau      : choice of tau (length of extension in fourier
    domain)
%   epsilon  : cutoff amplitude for the kernel
%   oversamp : degree of oversampling (pick >1)
```

```
%   p         : choice of phi (1 SqrtExpo, 2 Expo, 3 Sine, 4
    Poly)
%   b         : for phi where there is a selectable b

MN = floor((2+tau)*size(hatf,2));
nodes = 0:1/MN:(MN-1)/MN;

%These are the sampled points of f
%Can use FFT hereor already have the samples precomputed/
    sampled
f = DirectEval(nodes,hatf);

switch p
    case 2
        phi = ExpoFourierKernel((1+tau)*size(hatf,2),tau);
    case 3
        phi = SineFourierKernel((1+tau)*size(hatf,2),b,tau);
    case 4
        phi = PolyFourierKernel((1+tau)*size(hatf,2),b,tau);
    otherwise
        phi = SqrtExpoFourierKernel((1+tau)*size(hatf,2),b,
            tau);
end

%setting up for the oversampled kernel
snodes = 0:1/(oversamp*MN):(oversamp*MN - 1)/(oversamp*MN);
K = snodes*0;
```

```
%calculation of the oversampled kernel
for j = 1:oversamp*MN,

    temp = 0;

    for k = 1:size(phi,2),

        temp = temp + phi(k)*cos(2*pi*k*snodes(j));

    end

    K(j) = 1 + 2*temp;

end

%Decimation of K when strictly under epsilon (from the tails)
for i = 1:floor(size(K,2)/2),
    if abs(K(floor(size(K,2)/2)-i+1)) > epsilon ...
        || abs(K(floor(size(K,2)/2)+i-1)) > epsilon,
        break;
    end
    K(floor(size(K,2)/2)-i+1) = 0;
    K(floor(size(K,2)/2)+i-1) = 0;
end

%calculation of outputs
```

```
out = in*0;

for i = 1:size(out,2),

    tempK = nodes*0;

    for j = 1:MN,

        tempK(j) = Lagrange(nodes(j)-in(i),K,4);

    end

    for j = 1:MN,

        out(i) = out(i) + tempK(j)*f(j);

    end

end

out = out/MN;

end
```