LBNL-41851
Preprint

ERNEST ORLANDO LAWRENCE
BERKELEY NATIONAL LABORATORY

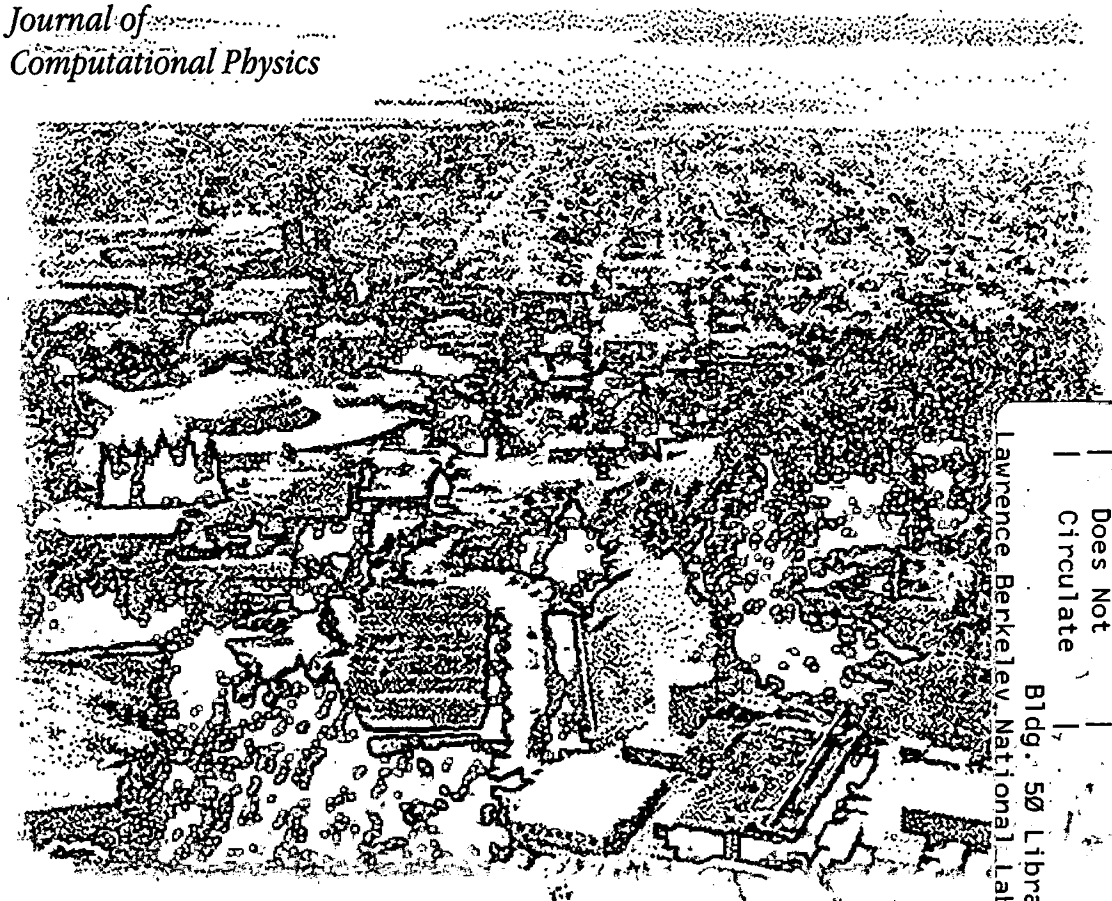
A Framework for Fast Evaluation of Fourier Series

Ping Tak Peter Tang

Computing Sciences Directorate
National Energy Research
Scientific Computing

May 1998

To be submitted to
Journal of
Computational Physics



DISCLAIMER

This document was prepared as an account of work sponsored by the United States Government. While this document is believed to contain correct information, neither the United States Government nor any agency thereof, nor the Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or the Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or the Regents of the University of California.

A Framework for Fast Evaluation of Fourier Series

Ping Tak Peter Tang *
National Energy Research Scientific Computing
Computing Science Directorate
Lawrence Berkeley National Laboratory
Berkeley, CA 94720
ptptang@nersc.gov

June 5, 1998

Abstract

This paper is about rapid evaluation of sums of the form $\sum_\ell \alpha_\ell e^{-\iota 2\pi \ell u}$ at arbitrary u. This include Fourier series as a special case, for example, of the form $\sum_{\ell,m,n} \alpha_{\ell,m,n} e^{-\iota 2\pi(\ell u_1+mu_2+nu_3)}$. This paper presents a framework with detailed analysis from which a variety of rapid evaluation methods can be obtained that offer tradeoffs between accuracy, speed, and memory requirements. Such options will be valuable not only to different applications where accuracy requirements may vary; but also to one single applications where accuracy requirement may vary in the application's different parts or stages. Numerical experiments are performed to illustrate the framework's key ideas and capabilities.

Keywords: Fourier series, fast interpolation, convolution.

1 Introduction

The main problem this paper addresses is the rapid evaluation of the sum in the form

$$F(\mathbf{u}) = \sum_{\boldsymbol{\ell}\in\text{Index}} \alpha_{\boldsymbol{\ell}} e^{-\iota 2\pi \boldsymbol{\ell}\cdot\mathbf{u}}$$

for an arbitrary number of points $\mathbf{u} \in \mathcal{R}^k$. Here, Index is a finite subset of $\mathcal{N}^k$, $\mathcal{N} = \{0, \pm1, \pm2, \ldots\}$; $\alpha_{\boldsymbol{\ell}} \in \mathcal{C}$ are given constants; and $\boldsymbol{\ell}\cdot\mathbf{u}$ is the usual inner product. Clearly, this include as a special case Fourier series of, for example, the form

$$F(u_1, u_2, u_3) = \sum_{-N\leq\ell,j,k\leq N} \alpha_{\ell,j,k} e^{-\iota 2\pi(\ell u_1+ju_2+ku_3)}.$$

Since Fourier series is crucial in many scientific computing, rapid evaluation methods will be highly useful. Although the FFT algorithm is often used for similar problems, it

[*] This work was supported by the the Director, Office of Energy Research, Office of Computational and Technology Research, Division of Mathematical, Information, and Computational Sciences of the U.S. Department of Energy under contract number DE-AC03-76SF00098.

is not capable of delivering results at arbitrary u. In applications such as quantum Monte Carlo simulations where wave functions are represented as products of finite Fourier series in three space, large number of evaluations are needed at arbitrary locations.

One way of tackling the main problem is through a system of recurrence which can be solved, for example, by special linear system [10]. The cost is however $O(N^2M)$ for evaluating a 1-D N-term Fourier series at M points.

Fast algorithms now exist that evaluate a 1-D N-term Fourier series at M places at a cost of $O(N\log N+M)$. The order $N\log N$ work is an initialization cost that has to be paid once and for all. After that, each Fourier series evaluation costs $O(1)$, that is, a constant independent of N. More precisely, Fast Multipole Method (FMM) based methods such as the one based on a series expansion of the function $f(z) = 1/(z - z_j)$ [1, 2] or that based on Chebyshev polynomial expansion of the function $f(x) = 1/x$ [6, 7] solve the problem at hand (in the one dimensional case) at a cost of the form

$$O\left((N\log N + M)\cdot \log^2(1/\epsilon)\right)$$

where ϵ is the desired accuracy.

Another method [5] based on approximation of function of the form $e^{\iota cx}$, c, x real, by functions of the form $\sum \rho_k e^{\iota kx}$ can solve the problem in

$$O\left(N\log N + M\log(\frac{1}{\epsilon})\right).$$

It is reported in [5] that this method is superior to [6] for the problem of Fourier series evaluation.

In this paper, we propose a framework that can produce a large number of interpolation schemes, each of which can evaluate a N-term Fourier series at M different points at a cost of

$$O\left(N\log N + M\log(\frac{1}{\epsilon})\right).$$

The framework is based on convolution and the interplay between continuous and discrete Fourier transforms. Though it looks quite different from the methods discussed previously, it truns out that the method in [5] can be considered a special case of this framework. Our framework offers a large choice of methods with different characteristics such as accuracy capabilities and memory requirements. It admits a natural error analysis that also offer additional insight on the method in [5].

The rest of the paper is organized as follows. Section 2 introduces the framework and its analysis in the one dimensional case. Section 3 discusses in more detail an especially useful case of the framework. Section 4 discusses the extension of the framework to higher dimensions. Section 5 compares our framework with the method in [5]. Section 6 presents some numerical and timing results. Finally, Section 7 gives some concluding remarks and possible future work.

2 Framework

A finite Fourier series

$$F(\mathbf{u}) = \sum_{\boldsymbol{\ell} \in \text{Index}} A_{\boldsymbol{\ell}} e^{\iota 2\pi \boldsymbol{\ell} \cdot \mathbf{u}} \qquad |\text{Index}| < \infty$$

can be thought of as the inverse Fourier transform of a sequence of delta functions placed on a regular grid. Precisely,

$$F(\mathbf{u}) = \int \left(\sum_{\boldsymbol{\ell} \in \text{Index}} A_{\boldsymbol{\ell}} \delta(\mathbf{x} - \boldsymbol{\ell}) \right) e^{\iota 2\pi \mathbf{u} \cdot \mathbf{x}} d\mathbf{x}.$$

Our framework tries to recognize Fourier and inverse Fourier transform of such delta functions as infinite sums that converges rapidly. We first define the basic components for this framework. We then show how we represent (or approximate) the sequence of delta functions as an infinite, but periodic, sequence of delta functions modulated by a window function. The Fourier and inverse Fourier transforms of the latter can then be recognized as convolutions, which give the infinite sum representations. By chosing suitable modulating window functions, the infinite sums can have reasonably fast convergence.

For the sake of simplicity, we focus on one-dimensional forward Fourier transform, that is, sums of the form

$$F(u) = \sum_{\ell \in \text{Index}} A_\ell e^{-\iota 2\pi \ell u} \qquad |\text{Index}| < \infty.$$

Generalization to higher dimension and to inverse Fourier transforms are straightforward and will be discussed towards the end of the paper.

2.1 Basic Ingredients

There are three basic ingredients of the framework: finite impulse train (fit), periodic impulse train (pit), and window function (win).

A finite impulse train is a finite sequence of delta functions placed on a regular grid. Formally:

Definition 1 *A finite impulse train, fit, is a generalized function of the form*

$$f(x) = \sum_{\ell=0}^{N-1} \alpha_\ell \delta(x - (x_0 + \ell \Delta X))$$

where $x \in \mathcal{R}$ is the independent variable, and $\alpha_\ell \in C$, $x_0, \Delta X \in \mathcal{R}$ are given constants.

A periodic impulse train is an infinite sequence of delta functions placed on a regular grid and whose magnitudes form a periodic sequence. We further require that the grid contains the origin so that Fourier transform of periodic impulse trains are periodic impulse trains. Formally,

Definition 2 *A periodic impulse train, pit, is a generalized function of the form*

$$g(x) = \sum_{\ell=-\infty}^{\infty} \alpha_\ell \delta(x - \ell \Delta X)$$

where there is an integer N such that

$$\alpha_\ell = \alpha_{\ell+N} \quad \textit{for all } \ell.$$

Finally, we denote any nonnegative real-valued function, win(x), a window function as long as its Fourier transform

$$\int_{-\infty}^{\infty} \text{win}(x) e^{-\iota 2\pi x u} dx$$

exists in the classical sense.

The main subject of this paper is to use these ingredients to devise fast algorithms to evaluate the Fourier transform of a fit at arbitrary location, that is, calculate

$$F(u) = \int \left[\sum_{\ell=0}^{N-1} \alpha_\ell \delta(x - (x_0 + \ell \Delta X)) \right] e^{-\iota 2\pi x u} dx$$

at arbitrary u. This clearly includes the case of evaluating a finite Fourier series.

2.2 Discrete and Continuous Fourier Transform

Many scientific computation that requires Fourier transform are performed by the FFT algorithm which computes the discrete Fourier transform. We review several relationships between continuous and discrete Fourier transform that are crucial to us.

Let

$$f(x) = \sum_{\ell=0}^{N-1} \alpha_\ell \delta(x - \ell/N),$$

be a finite impulse train, fit, and

$$g(x) = \sum_{\ell=-\infty}^{\infty} \alpha_\ell \delta(x - \ell/N), \quad \alpha_{\ell+N} = \alpha_\ell$$

be a periodic impulse train with f as one period. Finally, let the sequence

$$\{A_0, A_1, \ldots, A_{N-1}\} = \text{DFT}\{\alpha_0, \alpha_1, \ldots, \alpha_{N-1}\}$$

be the result of applying the discrete Fourier transform to the sequence $\{\alpha_0, \alpha_1, \ldots, \alpha_{N-1}\}$, that is,

$$A_k = \sum_{\ell=0}^{N-1} \alpha_\ell e^{-\iota 2\pi \ell k/N}, \qquad k = 0, 1, \ldots, N-1.$$

Then we have the following well-known relationships.

Relationship 1:

$$F(u) = \int_{-\infty}^{\infty} f(x)e^{-\iota 2\pi xu}dx = \mathcal{F}(f)(u)$$

is a periodic function of period N, $F(u) = F(u+N)$ and that

$$F(m) = A_m \qquad \text{for } m = 0, 1, \ldots, N-1.$$

In otherwords, DFT$\{\alpha_\ell\}$ gives the exact value of the continuous Fourier transform of f at the N locations $u = 0, 1, \ldots, N-1$.

Relationship 2:

$$G(u) = \int_{-\infty}^{\infty} g(x)e^{-\iota 2\pi xu}dx = \mathcal{F}(g)(u)$$

is a periodic impulse train

$$G(u) = \sum_{-\infty}^{\infty} A_m \delta(u-m), \qquad A_m = A_{m+N}.$$

Thus DFT$\{\alpha_\ell\}$ can be viewed as obtaining 1 period of g's continuous Fourier transform [3].

2.3 Sinc Interpolation Reviewed

Consider a fit of length N on the grid $0, 1/N, \ldots, N-1/N$. Since DFT$\{\alpha_0, \alpha_1, \ldots, \alpha_{N-1}\}$ gives $\{A_0, A_1, \ldots, A_{N-1}\}$ where

$$\mathcal{F}(f)(kN+m) = A_m, \qquad m = 0, 1, \ldots, N-1; \quad k = 0, \pm 1, \pm 2, \ldots,$$

it is natural to use interpolation based on $\{A_0, A_1, \ldots, A_{N-1}\}$ to obtain $\mathcal{F}(f)(u)$ at arbitrary u. One such kind of interpolation is well-known and we can derive this scheme using our basic ingredients.

Consider the "square window" function

$$\phi_0(x) = \begin{cases} 1, & \text{if } |x| < 1/2; \\ 1/2, & \text{if } |x| = 1/2; \\ 0, & \text{otherwise.} \end{cases}$$

It can be easily show that [3]

$$\mathcal{F}(\phi_0)(u) = \text{sinc}(u) \left(= \frac{\sin(\pi u)}{\pi u} \right).$$

Consider also the pit obtained by replicating f:

$$g(x) = \sum_{\ell=-\infty}^{\infty} \alpha_\ell \delta(x - \ell/N), \qquad \alpha_{\ell+N} = \alpha_\ell.$$

Hence

$$\begin{aligned} f(x) &= g(x)\cdot\phi_0(x-\tau), \quad \tau = (\frac{1}{2}-\frac{1}{N}) \\ &= g(x)\cdot \mathrm{win}(x) \end{aligned}$$

Consequently, the Fourier transform of f is given by the convolution of the Fourier transform of g and win:

$$\begin{aligned} F(u) &= G(u) * \mathrm{WIN}(u) \\ &= \left(\sum_{m=-\infty}^{\infty} A_m\delta(u-m)\right) * \left(e^{-\iota 2\pi\tau u}\mathrm{sinc}(u)\right), \end{aligned}$$

where $A_{m+N} = A_m$, and thus are known. Working out the convolution gives

$$F(u) = \sum_{m=-\infty}^{\infty} A_m e^{-\iota 2\pi\tau(u-m)}\mathrm{sinc}(u-m).$$

In particular, for any fixed u_0, let m_0 be an integer closest to u_0, that is $u_0 = m_0 + \xi_0$, $|\xi_0| \leq 1/2$.

$$\begin{aligned} F(u_0) &= \sum_{k=-\infty}^{\infty} A_{m_0+k}e^{-\iota 2\pi(\xi_0-k)}\mathrm{sinc}(\xi_0-k), \\ &\approx e^{-\iota 2\pi\xi_0}\sum_{k=-K}^{K} A_{m_0+k}e^{\iota 2\pi k}\mathrm{sinc}(\xi_0-k). \end{aligned}$$

The last formula is an interpolation method using "sinc" weighted average of F values close to u_0. Convergence to $F(u_0)$ is ensured if limit is taken with respect to K, that is, symmetrically ([9],Appendix I). Figure 1 illustrates this derivation.

To get an idea of the effectiveness of this scheme, we perform the following experiment. We generated 100 random values for $\{\alpha_\ell\}_0^{99}$ and 1000 random values

$$\{u_j\}_1^{1000}, \qquad u_j \in [0,5].$$

We define a fit,

$$f(x) = \sum_{\ell=0}^{99}\alpha_\ell\delta(x-\ell/100),$$

thus

$$F(u) = \mathcal{F}(f)(u) = \sum_{\ell=0}^{99}\alpha_\ell e^{-\iota 2\pi\ell u/100}.$$

For each u_j, we computed $F(u_j)$ exactly (up to rounding error due to finite precision arithmetic) by the formula above and also by the sinc interpolation scheme

$$F_{\mathrm{app}}(u_j) = \sum_{m=-K}^{K} A_{m_0+m}\mathrm{sinc}((u_j-m_0)-m)$$

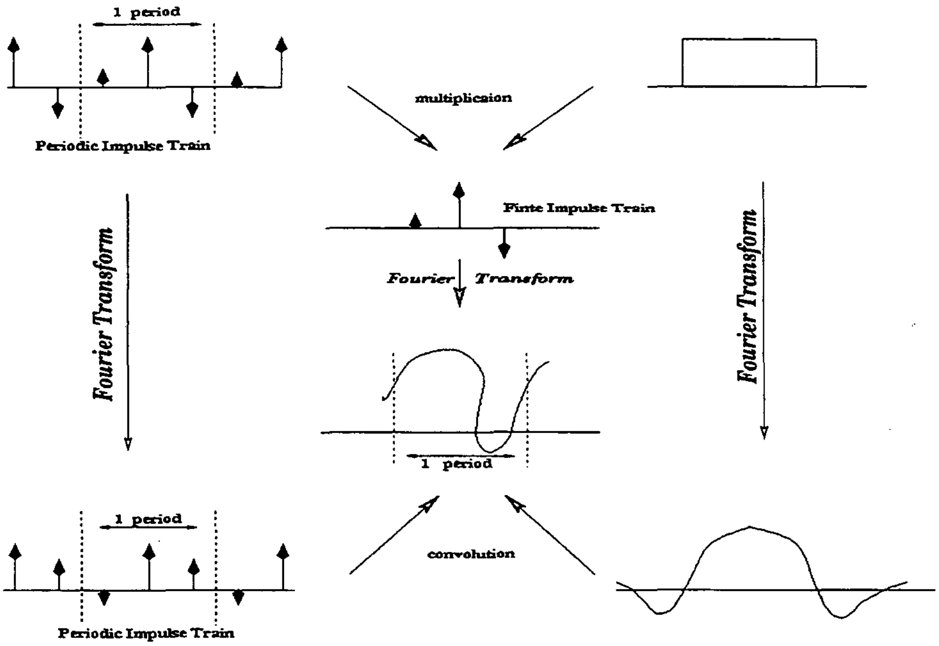


Figure 1: Derivation of Sinc Interpolation

Interpolation Error $||F_{\mathrm{app}} - F||/||F||$
Sinc Interpolation Scheme

K=4	K=8	K=12	K=16	K=20
$1.14e-01$	$4.97e-02$	$4.05e-02$	$3.95e-02$	$3.75e-02$

Table 1: Illustration of Ineffective Sinc Interpolation

where m_0 is an integer closest to u_j. For a fixed value of K, we calculated the error $||F_{\text{app}} - F||/||F||$ where F_{app} and F are vectors of length 1000. Table 1 tabulates the result.

Note that increasing K only increases the accuracy of the scheme slowly. The accuracy is clearly inadequate for many scientific applications at a reasonable number of terms K.

2.4 A Framework for Faster Interpolation

Two observations can be made from the previous section. First, the previous interpolation formula is obtained from the representation

$$\text{fit} = \text{pit} \cdot \text{win}.$$

From this representation, we have

$$\begin{aligned} \mathcal{F}(\text{fit}) &= \mathcal{F}(\text{pit}) * \mathcal{F}(\text{win}), \\ &= \text{PIT} * \text{WIN}, \\ &= \text{infinite sum}, \\ &\approx \text{interpolation formula}. \end{aligned}$$

Second, the obvious cause for the lack of accuracy in the previous interpolation scheme is the slow decaying nature of the sinc function. Since the window function in the time domain, win, was discontinuous, its Fourier transform, WIN, must decay slowly ([8]).

Combining the two observations, we see immediately that it is desirable to have a representation

$$\text{fit} = \text{pit} \cdot \text{win}$$

when win is smooth so that WIN decays rapidly. Clearly, whatever win is used, we must define pit so that the equality is preserved and that WIN $= \mathcal{F}(\text{win})$ is known explicitly so that the interpolation formula is in closed form.

Going further, it is in principle unnecessary to have strict equality between fit and pit · win as long as the difference between the two is tolerably small. Suppose a particular pit · win introduces other values on top of the fit:

$$\text{pit} \cdot \text{win} = \text{fit} + \text{ghost},$$

then

$$\begin{aligned} \mathcal{F}(\text{fit}) &= \text{PIT} * \text{WIN} - \mathcal{F}(\text{ghost}), \\ &= \text{infinite sum} - \mathcal{F}(\text{ghost}), \\ &\approx \text{interpolation formula} - \mathcal{F}(\text{ghost}), \\ &\approx \text{interpolation formula}. \end{aligned}$$

In general, a ghost error must be present when the window function, win, does not have finite support.

In summary, we would be approximating by truncation — interpolation formula instead of infinite sum, the Fourier transform of a slightly wrong function — fit + ghost instead of fit. With this framework, we can propose at least three different classes of interpolation methods.

1. Cosine window:

$$\begin{aligned} \text{win} &= \cos^d(\pi x)\cdot\phi_0(x), \quad d=0,1,2,\ldots; \\ \text{ghost} &= 0; \\ \text{WIN} &= \left(\tfrac{1}{2}(\delta(x+1/2)+\delta(x-1/2))\right)^{*d} * \operatorname{sinc}(x). \end{aligned}$$

2. Spline window:

$$\begin{aligned} \text{win} &= (\phi_0(x))^{*d} \quad d=1,2,\ldots; \\ \text{ghost} &= 0; \\ \text{WIN} &= \operatorname{sinc}^d(x). \end{aligned}$$

3. Gaussian window:

$$\begin{aligned} \text{win} &= \sqrt{\tfrac{\lambda}{\pi}}e^{-\lambda x^2} \\ \text{ghost} &= \text{nonzero} \\ \text{WIN} &= e^{-\pi^2x^2/\lambda}. \end{aligned}$$

Obviously the window function $\phi_0(x)$ is a special case for both the cosine and the spline window.

2.5 Numerical Considerations

A simple application of the cosine window to

$$f(x) = \sum_{\ell=0}^{N-1} \alpha_\ell \delta(x-\ell/N)$$

would be as follows. Define

$$\text{win}(x) = \cos^d(\pi(x-\tau))\phi_0(x-\tau) \qquad \text{where } \tau = \tfrac{1}{2}-\tfrac{1}{N}.$$

Note that $\text{win}(x) > 0$ on $(-\frac{1}{2N}, 1-\frac{1}{2N})$. Define

$$\begin{aligned} g_0(x) &= f(x)/\text{win}(x) \quad \text{for } x \in (-\tfrac{1}{2N}, 1-\tfrac{1}{2N}), \\ &= \sum_{\ell=0}^{N-1} \beta_\ell \delta(x-\ell/N), \end{aligned}$$

and consider a pit, g, by replication of $g_0(x)$:

$$\begin{aligned} g(x) &= \sum_{\ell=-\infty}^{\infty} g_0(x+\ell), \\ &= \sum_{\ell=-\infty}^{\infty} \beta_\ell \delta(x-\ell/N), \quad \beta_{N+\ell} = \beta_\ell. \end{aligned}$$

Interpolation Error $||F_{\text{app}} - F||/||F||$
Traditional Cosine Window

K	$w(x) = \cos^d(\pi x), d = 0, 1, \ldots, 5$					
	$d = 0$	$d = 1$	$d = 2$	$d = 3$	$d = 4$	$d = 5$
2	$1.88e-01$	$1.42e-01$	$5.76e-01$	$1.33e+01$	$3.73e+02$	$3.48e+04$
4	$1.14e-01$	$5.49e-02$	$1.81e-01$	$1.80e+00$	$1.19e+01$	$3.71e+02$
6	$7.03e-02$	$3.61e-02$	$8.95e-02$	$5.15e-01$	$2.75e+00$	$4.64e+01$
8	$4.97e-02$	$2.79e-02$	$5.29e-02$	$1.96e-01$	$1.07e+00$	$1.03e+01$
10	$4.39e-02$	$2.14e-02$	$3.54e-02$	$8.54e-02$	$5.20e-01$	$3.03e+00$
12	$4.05e-02$	$1.59e-02$	$2.59e-02$	$3.91e-02$	$2.84e-01$	$1.03e+00$
14	$4.00e-02$	$1.10e-02$	$1.99e-02$	$1.73e-02$	$1.67e-01$	$3.76e-01$
16	$3.95e-02$	$7.05e-03$	$1.56e-02$	$6.58e-03$	$1.03e-01$	$1.35e-01$
18	$3.93e-02$	$4.40e-03$	$1.23e-02$	$1.66e-03$	$6.64e-02$	$4.15e-02$
20	$3.75e-02$	$4.02e-03$	$9.51e-03$	$2.50e-03$	$4.35e-02$	$7.08e-03$

Table 2: Illustration of Cosine Window and Numerical Problem

Finally, compute $\{B_0, B_1, \ldots, B_{N-1}\} = \text{DFT}\{\beta_0, \beta_1, \ldots, \beta_{N-1}\}$ (via FFT). And we have

$$\begin{aligned} F(u) &= \int f(x) e^{-\iota 2\pi x u} dx, \\ &= \mathcal{F}\left(g(x) \cdot \text{win}(x)\right), \\ &= \sum_{m=-\infty}^{\infty} B_m \text{WIN}(u-m), \end{aligned}$$

where

$$\text{WIN}(u) = e^{-\iota 2\pi\tau u}\left\{\left(\frac{1}{2}\delta(u+\frac{1}{2}) + \frac{1}{2}\delta(u-\frac{1}{2})\right)^{*d} * \text{sinc}(u)\right\}.$$

For $d = 2$, for example, we have

$$\text{WIN}(u) = e^{-\iota 2\pi\tau u}\frac{\sin \pi u}{\pi}\frac{1}{2u(1-u^2)}.$$

Let us repeat the experiment in the previous section, using exactly the same set of randomly generated data. Since $\text{WIN}(u)$ decays faster than $\text{sinc}(u)$ we would expect the numbers of terms needed to achieve a comparable accuracy be fewer as d increases. We tabulate the accuracy achieved for various K and degree d in Table 2. Note that the column for $d = 0$ corresponds to the straightforward sinc interpolation.

The results seem to be inconsistent with the expectation. One observes that for a fixed K, accuracy actually decreases with increasing d. In the rest of Section 2, we discuss the stability and accuracy issues related to the proposed framework. The result is a correct application of the framework that yields fast and accurate interpolation schemes.

2.6 Stability Analysis

An obvious culprit for the bad results obtained in Table 2 is the division:

$$g_0(x) = f(x)/\text{win}(x) \quad \text{for } x \in (-\tfrac{1}{2N}, 1 - \tfrac{1}{2N}),$$

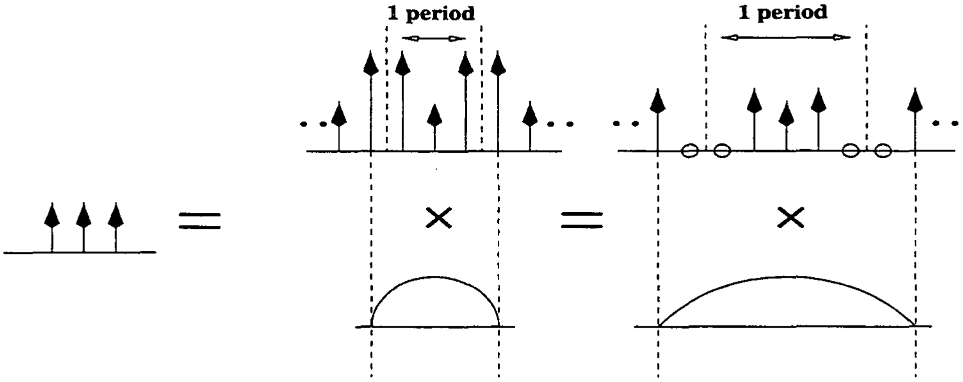


Figure 2: Curbbing Magnification by Zero Padding

$$= \sum_{\ell=0}^{N-1} \beta_\ell \delta(x - \ell/N).$$

Clearly, since win(x) is smooth and vanishes outside $(-\frac{1}{2N}, 1 - \frac{1}{2N})$, it must be small near 0 and $(N-1)/N$ — the end points of f. Consequently, the norm of $g_0(x)$ is much bigger than that of f. This means that

$$\|\{B_0, B_1, \ldots, B_{N-1}\}\|_2 = \|\{\beta_0, \beta_1, \ldots, \beta_{N-1}\}\|_2 \gg \|\{\alpha_0, \alpha_1, \ldots, \alpha_{N-1}\}\|_2.$$

Since $F(x)$ is subsequently evaluated by an interpolation formula using $\{B_0, B_1, \ldots, B_{N-1}\}$, large cancellation has to occur. In fact, it is easy to see that

$$\max_{\{\alpha_0,\alpha_1,\ldots,\alpha_{N-1}\}} \frac{\|\{\beta_0, \beta_1, \ldots, \beta_{N-1}\}\|_2}{\|\{\alpha_0, \alpha_1, \ldots, \alpha_{N-1}\}\|_2} \to \infty \qquad \text{as } N \to \infty.$$

Once the problem is identified, a solution is straightforward. Define the fundamental period g_0 of g so that its nonzero are kept away from the extreme ends of the window function win. This means that we have to lengthen the length of f from N to $N+P$. Figure 2.6 illustrates the idea.

Algebraically, we define

$$f_{\text{pad}}(x) = \sum_{\ell=0}^{N+P-1} \alpha'_\ell \delta(x - \ell/(N+P))$$

such that $\alpha'_\ell = 0$ for $\ell = N, N+1, \ldots, N+P-1$ and

$$f(x) = f_{\text{pad}}\left(\frac{N}{N+P}x\right).$$

Note that because $\delta(\sigma x) = \frac{1}{|\sigma|}\delta(x)$,

$$\alpha'_\ell = \frac{N}{N+P}\alpha_\ell \qquad \text{for } \ell = 0, 1, \ldots, N-1.$$

Now,

$$\mathcal{F}(f)(u) = \frac{N+P}{N}\mathcal{F}(f_{\text{pad}})\left(\frac{N+P}{N}u\right)$$

and thus it suffices to consider calculating $\mathcal{F}(f_{\text{pad}})(u)$ at arbitrary u. We can apply the framework by positioning and scaling the window function to have a center and support that correspond to f_{pad}:

$$\text{win}_{\text{scale}} = \text{win}\left(\frac{x-\tau}{\sigma}\right)$$

where

$$\tau = \frac{N-1}{2(N+P)}, \quad \text{and} \quad \sigma = \frac{N+2P}{N+P},$$

and win is simply the cosine window defined previously. We define

$$\begin{aligned} g_0(x) &= f_{\text{pad}}(x)/\text{win}_{\text{scale}}(x) \\ &= \sum_{\ell=0}^{N+P-1} \beta'_\ell \delta(x - \ell/(N+P)), \end{aligned}$$

and

$$\begin{aligned} g(x) &= \sum_{\ell=-\infty}^{\infty} g_0(x+\ell), \\ &= \sum_{\ell=-\infty}^{\infty} \beta'_\ell \delta(x - \ell/(N+P)). \end{aligned}$$

The magnification

$$\frac{\|\{\beta'_0, \beta'_1, \ldots, \beta'_{N-1}\}\|_2}{\|\{\alpha'_0, \alpha'_1, \ldots, \alpha'_{N-1}\}\|_2} \leq \text{win}_{\text{scale}}^{-1}\left(\frac{N-1}{2(N+P)}\right)$$

is controlled. Finally

$$\mathcal{F}(f_{\text{pad}})(u) = \sum_{m=-\infty}^{\infty} B'_m \text{WIN}(u-m)$$

as before.

The addition cost of padding P zeros involve (1) extra storage requirement of $N+P$ instead of N elements for $\{B_0, B_1, \ldots, B_{N+P-1}\}$ and (2) the cost in computing it, that is, $(N+P)\log(N+P)$ as opposed to $N \log N$, assuming the FFT algorithm is used.

We repeat the same experiment with the cosine window, always padding an amount of zeros so that the worst case magnification is bounded by 10:

$$\max_\alpha \frac{\|\alpha\|}{\|\beta\|} \leq 10.$$

Table 3 summarizes the result in terms of accuracy achieved as well as the amount of zeropadding $(N+P)/N$ needed for various degree d.

Although we use the cosine window as an example, the discussion is clearly applicable to all other window functions that fit the framework. Magnification factors can be calculated rather easily and an amount of zero padding (as a function of the original length) can be determined beforehand.

Interpolation Error $||F_{\text{app}} - F||/||F||$
Cosine Window — Stable Version $\kappa = 10$

	$w(x) = \cos^d(\pi x), d = 0, 1, \ldots, 5$					
K	$d = 0$	$d = 1$	$d = 2$	$d = 3$	$d = 4$	$d = 5$
2	$1.88e-01$	$4.34e-02$	$1.90e-02$	$1.57e-02$	$2.48e-02$	$6.14e-02$
4	$1.14e-01$	$2.99e-02$	$5.44e-03$	$1.66e-03$	$6.87e-04$	$2.63e-04$
6	$7.03e-02$	$1.35e-02$	$1.35e-03$	$3.00e-04$	$8.87e-05$	$2.56e-05$
8	$4.97e-02$	$6.17e-03$	$7.25e-04$	$1.05e-04$	$2.09e-05$	$5.31e-06$
10	$4.39e-02$	$5.39e-03$	$3.68e-04$	$5.72e-05$	$7.47e-06$	$1.76e-06$
12	$4.05e-02$	$4.55e-03$	$2.15e-04$	$2.50e-05$	$3.69e-06$	$6.42e-07$
14	$4.00e-02$	$3.02e-03$	$1.51e-04$	$1.18e-05$	$1.93e-06$	$2.55e-07$
16	$3.95e-02$	$1.89e-03$	$1.05e-04$	$8.93e-06$	$9.99e-07$	$1.24e-07$
18	$3.93e-02$	$1.84e-03$	$7.55e-05$	$5.39e-06$	$5.35e-07$	$6.19e-08$
20	$3.75e-02$	$1.96e-03$	$5.66e-05$	$3.31e-06$	$2.92e-07$	$3.47e-08$
(N+P)/N	1.0	1.1	1.2	1.3	1.5	1.5

Table 3: Illustration of Numerically Stable Cosine Window

2.7 Error Analysis

Recall that our framework is

$$\text{fit} = \text{pit} \cdot \text{win} - \text{ghost},$$

or

$$f_{\text{fit}}(x) = g_{\text{pit}}(x) \cdot \text{win}(x) - \text{ghost}(x).$$

The interpolation formula is an approximation of

$$\begin{aligned} \mathcal{F}(g_{\text{pit}}(x) \cdot \text{win}(x)) &= \text{PIT} * \text{WIN}, \\ &\approx \sum_{m=m_0-K}^{m_0+K} B_m \text{WIN}(u-m) + \text{truncation error}, \\ &\approx \sum_{m=-K}^{K} B_{m_0+m} \text{WIN}(u-(m_0+m)) + \text{truncation error}. \end{aligned}$$

In other words, we compute with some truncation error the Fourier transform of a possibly slightly different function, $f + \text{ghost}$, thus further incurring a backward error.

2.7.1 Truncation Error

The truncation error can be estimated rather easily if we have an asymptotic rate of decay for WIN:

$$\text{WIN}(u) = O(r(u)) \qquad \text{as } |u| \to \infty.$$

$$\left| \sum_{|m|>K} B_{m_0+m} \text{WIN}(u-(m_0+m)) \right| \leq M ||B||_\infty \left| \sum_{|m|>K} r(m) \right|$$

$$\leq M\left(\frac{\|B\|_\infty}{\|B\|_2}\right)(\kappa\|A\|_2)\left|\sum_{|m|>K} r(m)\right|,$$

where κ is the magnification

$$\kappa = \frac{\|B\|_2}{\|A\|_2}.$$

The factor $\|B\|_\infty/\|B\|_2$ is hard to avoid when using an interpolation formula obtained from truncation of an infinite seris. In the extreme case, there can be one coefficient B_i that contains most of the weight of one period $\{B_0, B_1, \ldots, B_{N+P-1}\}$.

2.7.2 Backward Error

Backward error occurs whenever the window function win does not have compact support in the domain space of f. The only window we propose that is of this type is the Gaussian window:

$$\text{win}(x) = \sqrt{\frac{\lambda}{\pi}}e^{-\lambda x^2}.$$

Since win does not have a finite support contained in an interval, we cannot achieve the representation

$$f_{\text{fit}} = g_{\text{pit}} \cdot \text{win}$$

as the periodic impulse train g_{pit} has infinite support. Hence we must have

$$g_{\text{pit}} \cdot \text{win} = f_{\text{fit}} + \text{ghost},$$

where

$$\text{ghost} = -g_{\text{pit}} \cdot \text{win}\Big|_{\Omega},$$

with

$$\Omega = \{x \mid g_{\text{pit}} \cdot \text{win}(x) \neq 0 \quad \text{and} \quad x \notin \text{domain of } f.\}$$

This is illustrated by Figure 3.

The norm $\|\text{ghost}\|$ can be estimated beforehand in terms of f, win, and amount of zero padding. (See the next section for more details.) We first present in Table 4 the result on the same set of data.

For a fixed ghost error level, accuracy improves as K increases until the limit of ghost error is reached, thus giving the "triangular" nature of the table. Compared with Table 3, for example, a Gaussian window with $K = 6$ yields an accuracy comparable to that of a degree 4 cosine window using $K = 10$. It is quite clear that for high accuracy, Gaussian window is an excellent choice.

3 Framework with Gaussian Window

Because of the smoothness of the Gaussian function, it should be an excellent choice if high accuracy is desired. We discuss in this section the choice of λ (with respect to zero padding

$g(x) \cdot \mathrm{win}_\lambda(x) \;=$

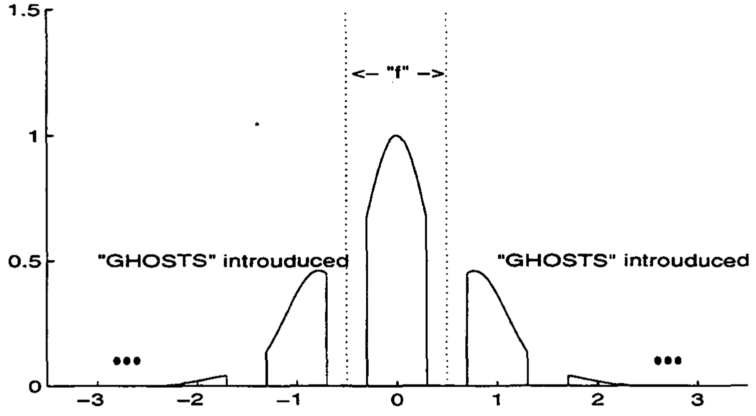


Figure 3: Exaggerated Ghost Error as an Illustration

Interpolation Error $\|F_{\mathrm{app}} - F\|/\|F\|$
Gaussian Window — Stable Version $\kappa = 10$

	$\epsilon_{\mathrm{ghost}} = 10^{-j}$, $j = 4, 6, \ldots, 12$				
K	$j = 4$	$j = 6$	$j = 8$	$j = 10$	$j = 12$
3	$3.83e-03$	$1.38e-02$	$2.59e-02$	$3.86e-02$	$5.2991e-02$
4	$1.21e-04$	$7.82e-04$	$2.49e-03$	$5.57e-03$	$1.1460e-02$
5	$2.45e-05$	$2.79e-05$	$1.45e-04$	$4.62e-04$	$1.0903e-03$
6	$2.52e-05$	$3.93e-07$	$4.44e-06$	$2.72e-05$	$9.3324e-05$
7		$2.52e-07$	$9.86e-08$	$8.60e-07$	$4.8672e-06$
8		$2.51e-07$	$2.45e-09$	$2.18e-08$	$1.6581e-07$
9			$1.96e-09$	$2.78e-10$	$3.7745e-09$
10			$1.95e-09$	$1.89e-11$	$6.1975e-11$
11				$1.81e-11$	$7.0032e-13$
12				$1.81e-11$	$1.5873e-13$
13					$1.6004e-13$
14					$1.6005e-13$

Table 4: Illustration of Numerically Stable Gaussian Window

needed and the ghost error) and the actual computation of the interpolation formula. In this section, we will use the notation

$$\mathrm{win}_\lambda(x) = \sqrt{\frac{\lambda}{\pi}} e^{-\lambda x^2}; \qquad \mathrm{WIN}_\lambda(u) = e^{-\pi^2 u^2/\lambda}, \quad \lambda > 0.$$

3.1 Determination of λ

Since interpolation given by truncation of the infinite sum is of the form

$$F(u) = \sum_{k=-\infty}^{\infty} A_k \mathrm{WIN}_\lambda(u-k),$$

where $\mathrm{WIN}_\lambda(u) = e^{-\pi^2 u^2/\lambda}$, we would like to have $\lambda > 0$ to be as small as possible so that the decay of $\mathrm{WIN}_\lambda(u)$ is rapid. Picking a small λ, however, would render the decay of win_λ (in the physical/time domain) too slow so that the ghost introduced would be large, which in turn limits the ultimate accuracy of how well the infinite sum approximate the Fourier series in question. This dilemma is a manifestation of the uncertainty principle. Finally, there is a third consideration, namely, magnification and the zero padding which curbs it. We now discuss these relationships.

Up to a scaling, we can assume without loss of generality that the data of f_{fit} lives on $[-\gamma, \gamma]$, $\gamma < 1/2$; and that

$$f_{\mathrm{fit}}(x) = g_{\mathrm{pit}}(x) \cdot \mathrm{win}_\lambda(x)$$

where $g_{\mathrm{pit}}(x)$ has period 1, that is, one period lives on $[-1/2, 1/2]$ and

$$f_{\mathrm{fit}}(x) = g_{\mathrm{pit}}(x) \cdot \mathrm{win}_\lambda(x), \qquad \text{for } x \in [-1/2, 1/2],$$

indicating that $g_{\mathrm{pit}} \equiv 0$ on $[-1/2, -\gamma)$ and $(\gamma, 1/2]$, and the amount of zero padding is $\frac{1-2\gamma}{\gamma} N$. The magnification for this amount of zero padding is

$$\kappa = \frac{\mathrm{win}_\lambda(0)}{\mathrm{win}_\lambda(\gamma)} = e^{\lambda \gamma^2}.$$

Thus, if $\kappa_{\mathrm{desired}}$ denotes a desired bound on magnification, we have the first constraint.

Constraint 1:

$$e^{\lambda \gamma^2} \leq \kappa_{\mathrm{desired}}.$$

In this setting,

$$f_{\mathrm{fit}} = g_{\mathrm{pit}}(x) \cdot \mathrm{win}_\lambda(x) - \mathrm{ghost}(x),$$

where, for $|x| \leq 1/2$ and $k = \pm 1, \pm 2, \ldots$,

$$\begin{aligned} \mathrm{ghost}(k+x) &= g_{\mathrm{pit}}(k+x) \cdot \mathrm{win}_\lambda(k+x) - f_{\mathrm{fit}}(k+x), \\ &= \frac{f_{\mathrm{fit}}(x)}{\mathrm{win}_\lambda(x)} \mathrm{win}_\lambda(k+x). \end{aligned}$$

Because of the exponential decay, the ghost function is dominated by

$$\frac{f_{\text{fit}}(x)}{\text{win}_\lambda(x)}\text{win}_\lambda(1+x) \qquad |x| \le 1/2.$$

The highest value is bounded by

$$\frac{\text{win}_\lambda(1-\gamma)}{\text{win}_\lambda(-\gamma)}\|f_{\text{fit}}\| = \frac{\text{win}_\lambda(1-\gamma)}{\text{win}_\lambda(\gamma)}\|f_{\text{fit}}\| = e^{-\lambda(1-\gamma)^2}e^{\lambda\gamma^2}\|f_{\text{fit}}\|.$$

Thus, if we let $\epsilon_{\text{ultimate}}$ denote the smallest error ever achievable in the presence of this ghost error function, we have the next constraint.

Constraint 2:

$$\epsilon_{\text{ultimate}} \ge e^{-\lambda(1-\gamma)^2}e^{\lambda\gamma^2}.$$

Finally, there is an extra constraint on $\epsilon_{\text{ultimate}}$. Clearly, the ultimate accuracy cannot exceed than the underlying arithmetic accuracy of the computer. Moreover, since a magnification of 10^k means that in general there can be k digits of cancellation in the summation process, we arrive at the third constraint.

Constraint 3:

$$\kappa_{\text{desired}}\epsilon_{\text{machine}} \le \epsilon_{\text{ultimate}}.$$

In short, actual uses of a Gaussian window require a choice of λ as well as γ, κ_{desired}, and $\epsilon_{\text{ultimate}}$ that are consistent. We find the following two procedures workable in practice. vspace0.1in **Procedure 1:** When the choice of γ is flexible.

1. Start from a parameter $\epsilon_{\text{ultimate}} > \epsilon_{\text{machine}}$ and decide on a moderate κ_{desired} satisfying

   $$\kappa_{\text{desired}} \le \epsilon_{\text{ultimate}}/\epsilon_{\text{machine}}.$$

   For example,

   $$\kappa_{\text{desired}} = \min(10^3, \epsilon_{\text{ultimate}}/\epsilon_{\text{machine}}).$$

2. Pick γ that satisfies Constraints 1 and 2:

   $$e^{\lambda\gamma^2} \le \kappa_{\text{desired}} \Rightarrow \lambda \le \log\kappa_{\text{desired}}/\gamma^2$$

   and

   $$\epsilon_{\text{ultimate}} \ge e^{-\lambda(1-\gamma)^2}e^{\lambda\gamma^2} \Rightarrow \lambda \ge |\log(\epsilon_{\text{ultimate}})|/(1-2\gamma).$$

   It is easy to see that a unique $\lambda_0 \in (0,1/2)$ exists such that

   $$\frac{\log\kappa_{\text{desired}}}{\gamma_0^2} = \frac{|\log\epsilon_{\text{ultimate}}|}{1-2\gamma_0}.$$

Moreover, for all $0 < \gamma \leq \gamma_0$,

$$\frac{\log \kappa_{\text{desired}}}{\gamma^2} \geq \frac{|\log \epsilon_{\text{ultimate}}|}{1-2\gamma}.$$

Note that the zero padding amount, expressed as a factor of the original data length, is $(1/2\gamma) - 1$. In general, increasing zero padding increases convergence rate, albeit at the expense of increase storage and a slightly higher initialization rate. Hence we can pick γ to be, for example,

$$\gamma = \min(1/(2 \times 1.1), \gamma_0)$$

to ensure a zero padding of at least 10%.

3. Finally, pick λ to be the smallest possible choice:

$$\lambda = |\log \epsilon_{\text{ultimate}}|/(1-2\gamma).$$

Procedure 2: When γ is prescribed.

This is the case when an amount of zero padding is prescribed, usually due to memory constraints.

1. Pick $\epsilon_{\text{ultimate}}$ to suit the application and define κ_0 by

$$\frac{\log \kappa_0}{\gamma^2} = \frac{|\log \epsilon_{\text{ultimate}}|}{1-2\gamma}.$$

If $\kappa_0 \leq \epsilon_{\text{ultimate}}/\epsilon_{\text{machine}}$, then we can pick any κ_{desired} in the range $(\kappa_0, \epsilon_{\text{ultimate}}/\epsilon_{\text{machine}})$. From κ_{desired}, we can define λ by

$$\lambda = \frac{\log \kappa_{\text{desired}}}{\gamma^2},$$

and this ends the procedure.

2. Otherwise, if $\kappa_0 > \epsilon_{\text{ultimate}}/\epsilon_{\text{machine}}$, the original choice of $\epsilon_{\text{ultimate}}$ is unachievable for the prescribed γ. We must redefine a less stringent $\epsilon_{\text{ultimate}}$ by

$$\frac{|\log \epsilon_{\text{ultimate}}|}{1-2\gamma} = \frac{\log \epsilon_{\text{ultimate}} - \log \epsilon_{\text{machine}}}{\gamma^2}$$

From $\epsilon_{\text{ultimate}}$, define κ_{desired} and λ by

$$\begin{aligned} \kappa_{\text{desired}} &= \epsilon_{\text{ultimate}}/\epsilon_{\text{machine}}, \\ \lambda &= \log \kappa_{\text{desired}}/\gamma^2. \end{aligned}$$

3.2 Computing the interpolation

The form of interpolation is

$$\begin{aligned} F_{\text{app}}(u) &= \sum_{k=m_0-K}^{m_0+K} A_k \text{WIN}_\lambda(u-k), \\ &= \sum_{k=-K}^{K} A_{m_0+k} \text{WIN}_\lambda(u-(m_0+k)), \\ &= \sum_{k=-K}^{K} A_{m_0+k} \text{WIN}_\lambda(\xi-k), \end{aligned}$$

where $u = k_0 + \xi$, $|\xi| \leq 1/2$. Thus,

$$\begin{aligned} F_{\text{app}}(u) &= \sum_{k=-K}^{-1} A_{m_0+k} \text{WIN}_\lambda(\xi-k) + A_{m_0} \text{WIN}_\lambda(\xi) + \sum_{k=1}^{K} A_{m_0+k} \text{WIN}_\lambda(\xi-k), \\ &= \sum_{k=1}^{K} A_{m_0-k} \text{WIN}_\lambda(\xi+k) + A_{m_0} \text{WIN}_\lambda(\xi) + \sum_{k=1}^{K} A_{m_0+k} \text{WIN}_\lambda(\xi-k), \\ &= \sum_{k=1}^{K} A_{m_0-k} e^{-\mu\xi^2} \left(e^{-2\mu\xi}\right)^k e^{-\mu k^2} + A_{m_0} e^{-\mu\xi^2} + \sum_{k=1}^{K} A_{m_0+k} e^{-\mu\xi^2} \left(e^{2\mu\xi}\right)^k e^{-\mu k^2}, \end{aligned}$$

where $\mu = \pi^2/\lambda$ is independent of ξ. Once λ is determined, $e^{-\mu k^2}$ can be computed once for $k = 1, 2, \ldots, K$, independent of u (and ξ). Then, F_{app} is in the form

$$F_{\text{app}}(u) = e^{-\mu\xi^2} \left(A_{m_0} + \sum_{k=1}^{K} B_k x^k + \sum_{k=1}^{k} C_k y^k \right),$$

where $x = e^{-2\mu\xi}$ and $y = 1/x$. Hence $F_{\text{app}}(u)$ can be evaluated by 2 exponential evaluations and the cost of two simple length K polynomial evaluation by, say, Horner's recurrence.

4 Higher Dimension and Inverse Transforms

The previous framework is easily generalizable to higher dimension. Typically, we will use separable window functions, say,

$$\text{win}(x, y, z) = \text{win}_1(x)\text{win}_2(y)\text{win}_3(y).$$

Then, the framework becomes

$$\begin{aligned} f_{\text{fit}}(x, y, z) &= g_{\text{pit}}(x, y, z) \cdot \text{win}(x, y, z) - \text{ghost}(x, y, z); \\ \mathcal{F}(f_{\text{fit}})(u, v, w) &= \mathcal{F}(g_{\text{pit}} \cdot \text{win}) - \mathcal{F}(\text{ghost}), \\ &\approx G_{\text{pit}}(u, v, w) * WIN(u, v, w), \\ &\approx G_{\text{pit}}(u, v, w) * (\text{WIN}_1(u)\text{WIN}_2(v)\text{WIN}_3(w)), \\ &\approx \text{interpolation formula.} \end{aligned}$$

Hence,

$$F(u,v,w) \approx \sum_{\|(\ell,j,k)-(\ell_0,j_0,k_0)\| \le K} A_{\ell,j,k}\mathrm{WIN}(u-\ell)\mathrm{WIN}(v-j)\mathrm{WIN}(w-k).$$

Applying the framework to series of the form

$$F(\mathbf{u}) = \sum_{\boldsymbol{\ell}\in\mathrm{Index}} A_{\boldsymbol{\ell}} e^{\iota 2\pi \boldsymbol{\ell}\cdot\mathbf{u}} \qquad |\mathrm{Index}| < \infty$$

is straightforward. One possibility is to note that $F(\mathbf{u}) = \bar{H}(\mathbf{u})$ where

$$H(\mathbf{u}) = \sum_{\boldsymbol{\ell}\in\mathrm{Index}} \bar{A}_{\boldsymbol{\ell}} e^{-\iota 2\pi \boldsymbol{\ell}\cdot\mathbf{x}} \qquad |\mathrm{Index}| < \infty.$$

Another possibility is to use the framework for inverse Fourier transform

$$F(\mathbf{u}) = \int \left(\sum_{\boldsymbol{\ell}\in\mathrm{Index}} A_{\boldsymbol{\ell}}\delta(\mathbf{x}-\boldsymbol{\ell}) \right) e^{\iota 2\pi \mathbf{u}\cdot\mathbf{x}} d\mathbf{x},$$

and

$$f_{\mathrm{fit}} = g_{\mathrm{pit}} \cdot \mathrm{win} - \mathrm{ghost}$$

implies

$$\mathcal{F}^{-1}(f_{\mathrm{fit}}) = \mathcal{F}^{-1}(g_{\mathrm{pit}}) * \mathcal{F}^{-1}(\mathrm{win}) - \mathcal{F}^{-1}(\mathrm{ghost}),$$

and $\mathcal{F}^{-1}(g_{\mathrm{pit}}) * \mathcal{F}^{-1}(\mathrm{win})$ immediately yields an interpolation formula through truncation.

5 Comparison with Dutt/Rokhlin

As alluded to in the introduction, the method by Dutt and Rokhlin [5] can be derived from our framework as a specific instance. In this section, we work out just enough details to illustrate this point.

The key lemma in [5] from which the main theorem and all algorithms are derived is the approximation of the form

$$\left| e^{\iota cx} - e^{bx^2} \sum_{\text{integer } k \text{ near } c} \rho_k e^{\iota kx} \right| < \mathrm{bound}(b,m),$$

where $m \ge 2$ is an integer, c is any real number, x can be anywhere in $[-\pi/m, \pi/m]$, and

$$\rho_k = \frac{1}{2\sqrt{\pi b}} e^{-(c-k)^2/4b}.$$

We show that this is exactly a case of computing the 1-term Fourier series $e^{\iota cx}$ (for a fixed value x) at arbitrary c using our framework with a gaussian window. The scalings employed in [5] are more naturally linked to the following definition of Fourier transforms:

$$\begin{array}{lllll} \text{Forward:} & G(w) & = \mathcal{F}(g)(w) & = & \int g(u)e^{-\iota uw}du, \\ \text{Inverse:} & g(u) & = \mathcal{F}^{-1}(G)(u) & = & \frac{1}{2\pi}\int G(w)e^{\iota wu}dw. \end{array}$$

For any $x \in [-\pi/m, \pi/m]$, consider the finite impulse train

$$F(w) = 2\pi\delta(w - x).$$

Thus $\mathcal{F}^{-1}(F)(u) = e^{\iota ux}$. Next, consider the window function

$$\text{WIN}_b(w) = e^{-bw^2}.$$

and the periodic impulse train defined by repricating

$$\frac{1}{\text{WIN}_b(x)}F(w)$$

to be 2π periodic, thus

$$G(w) = \left[\frac{1}{\text{WIN}_b(x)}F(w)\right] * \left[\sum_{k=-\infty}^{\infty} \delta(w - 2k\pi)\right].$$

We clearly have the approximation

$$F(w) \approx G(w) \cdot \text{WIN}_b(w).$$

Hence

$$e^{\iota ux} \approx \mathcal{F}^{-1}(G) * \mathcal{F}^{-1}(\text{WIN}_b).$$

Now,

$$\begin{aligned} \mathcal{F}^{-1}(G)(u) &= \mathcal{F}^{-1}\left(\frac{1}{\text{WIN}_b(x)}F(w)\right) \cdot \mathcal{F}^{-1}\left(\sum_{k=-\infty}^{\infty} \delta(w - 2k\pi)\right), \\ &= \left(\frac{1}{\text{WIN}_b(x)}e^{\iota ux}\right) \cdot \left(\sum_{k=-\infty}^{\infty} \delta(u - k)\right), \\ &= \frac{1}{\text{WIN}_b(x)} \sum_{k=-\infty}^{\infty} e^{\iota kx}\delta(u - k), \end{aligned}$$

and

$$\mathcal{F}^{-1}(\text{WIN}_b)(u) = \frac{1}{2\sqrt{\pi b}}e^{-u^2/4b}.$$

Hence, we have the approximation

$$\begin{aligned} F(u) &\approx \left[\frac{1}{\text{WIN}_b(x)} \sum_{k=-\infty}^{\infty} e^{\iota kx}\delta(u - k)\right] * \frac{1}{2\sqrt{\pi b}}e^{-u^2/4b}, \\ &\approx e^{bx^2} \sum_{k=-\infty}^{\infty} \frac{1}{2\sqrt{\pi b}}e^{-(u-k)^2/4b} \cdot e^{\iota kx}, \\ &\approx e^{bx^2} \sum_{\text{integer } k \text{ near } u} \rho_k e^{\iota kx}, \quad \text{where } \rho_k = \tfrac{1}{2\sqrt{\pi b}}e^{-(u-k)^2/4b}. \end{aligned}$$

The correspondence is clear. The Gaussian functions [5] that show up as e^{bx^2} and ρ_k are the window function in the time and frequency domains. Hence we have a framework to generate approximations of the form

$$e^{\iota cx} \approx \frac{1}{\mathrm{WIN}_b(x)} \sum \rho_k e^{\iota kx}$$

where $\rho_k = \mathcal{F}^{-1}(\mathrm{WIN})(c-k)$. The parameter m in [5] corresponds to zeropadding since the finite impulse train lives on $[-\pi/m, \pi/m]$ and the periodic impulse train has period 2π. In principle, m needs not be an integer; although the description in [5] seems to require that, it can be easily seen that the algorithms in [5] can work as long as mN is an integer.

We like to emphasize the important relationships of the numerical stability, zeropadding, and ultimate accuracy. The work in [5] suggested a particular choice for b and zeropadding ($m = 2$ corresponds to zeropadding to double the data length) and the number of terms used. The magnification effect (for example in Step 1 of Algorithm 2) however, is not discussed.

We have been focusing on the problem of evaluating Fourier series. Nevertheless, our framework is applicable to all the problems that [5] addresses in exactly the same way as shown in [5].

6 Numerical and Timing Experiments

This section illustrates the correctness and complexity of the algorithm by a number of examples based on randomly generated Fourier series (that is the coefficients) evaluated at randomly generated points in the Fourier series domain. All results here are based on the Gaussian window as we consider this tool to be a most powerful one for general purposes where an accuracy comparable to the machine precision is sought. All evaluation are compared with direct evaluation and their difference in numerical values are around 10^{-13}. We therefore only tabulate the timing results below. All experiments are carried out on a single node (DEC Alpha EV5) of the Cray T3E–900 platform of the National Energy Research Scientific Computing Center.

Figure 4 shows that time required to evaluating a 1-D length N Fourier series at M points. The value of N varies from 2^6 to 2^12 and the value of M varies from 1000 to 5000. The constant spacing that confirms the $O(M)$ part of the complexity. The growth of time with N also follows a typical trend for growth of time required for FFT, which is usually not a simple $N\log(N)$. The reason is that as soon as the length of the data no longer fit in cache, memory access time becomes dominant, breaking the $N\log(N)$ trend.

To get an idea of how well this method works, we compare the time required for a direct evaluation and express the comparison by the base-2 logarithm of the ratio $T_{\text{direct}}/T_{\text{fast}}$. Figure 5 shows the result based on the same set of random test problems used in Figure 4. Observe that for each N, the ratio of the times tends to a constant as M grows. This confirms our understanding that the dominant cost in both methods are $O(M)$ as M becomes large. The almost equal spacing in the $\log_2$ scale is a reflection that the cost of direct evaluation is $O(NM)$. Moreover, if that cost is roughly $O(K_1NM)$, the the graph suggests that the cost of our fast method is $O(N\log N) + O(16K_1M)$.

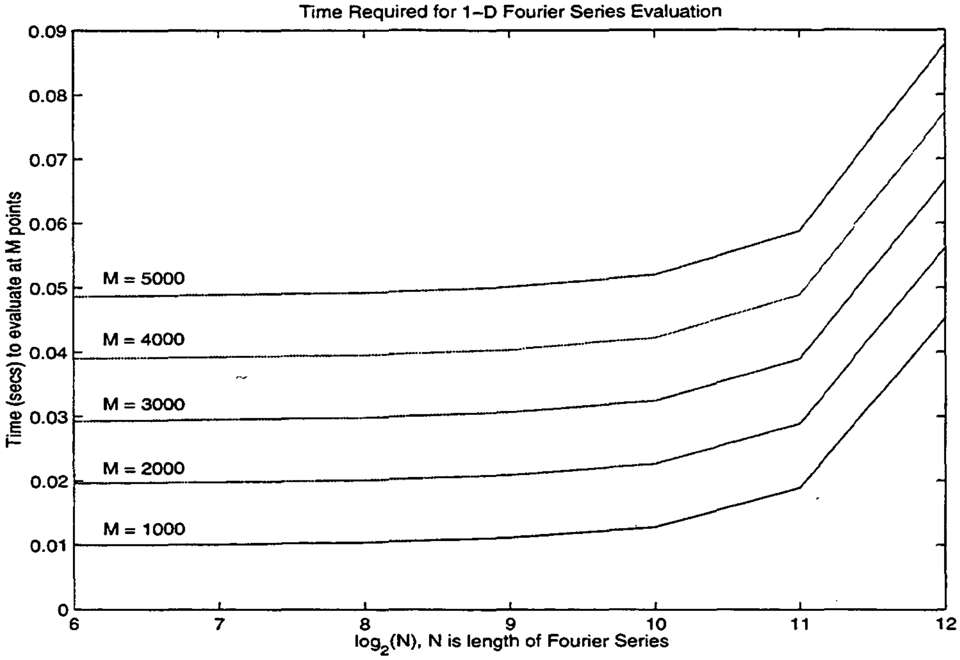


Figure 4: Complexity of Algorithm

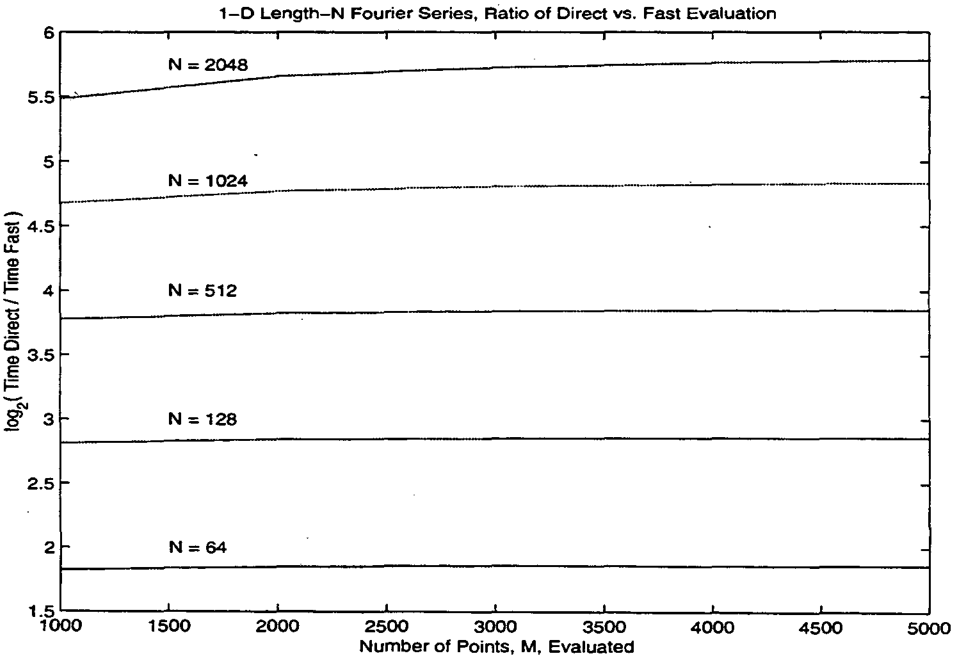


Figure 5: Comparison with Direct Evaluation

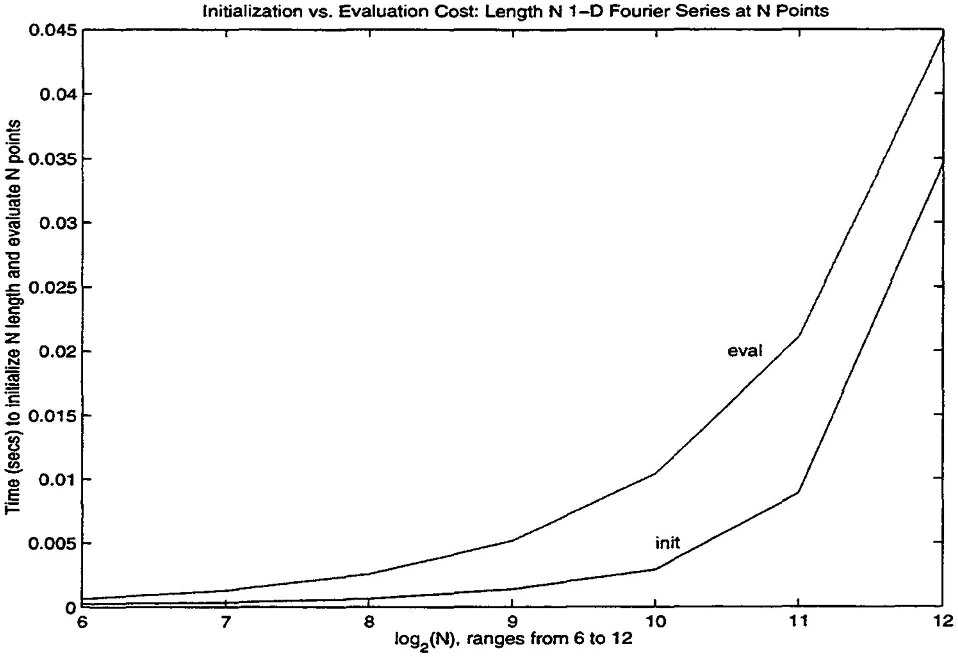


Figure 6: Initialization Cost vs. Evaluation Cost

Next, we would like to get an idea of the cost of irregularity. We compare the initialization time, which is an FFT of roughly double the length N, to the time for the evaluation at N random points in the domain. Figure 6 shows that this ratio is moderate and suggests that it does not grow with N.

Finally, we present a 2-D example. Here a randomly generated N-by-N Fourier series is evaluated at M randomly generated points (x_j, y_j), $j = 1, 2, \ldots, M$. Note that these points are totally irregular, as opposed to points of the form (x_i, y_j); $i = 1, 2, \ldots, M_x$, and $j = 1, 2, \ldots, N_y$. Figure 7 is in the same spirit as Figure 4 and Figure 8 is in the same spirit as Figure 5. Note in particular that Figure 8 does show the trend of the cost of direct evaluation to be $O(N^2)$ more expensive than our fast algorithm. For larger N, however, M probably had to be increased to be comparable to N^2 for that trend to be obvious.

7 Conclusion

We have presented a framework which leads to a number of fast methods for evaluation of Fourier series at arbitrary places. These methods offer tradeoffs between accuracy, speed, and memory requirements and are thus useful in accommodating different kinds of applications.

We see that there are at least two routes of future work that are worth pursuing. First, this algorithm is shown to be a generalization of the method proposed in [5]. Because

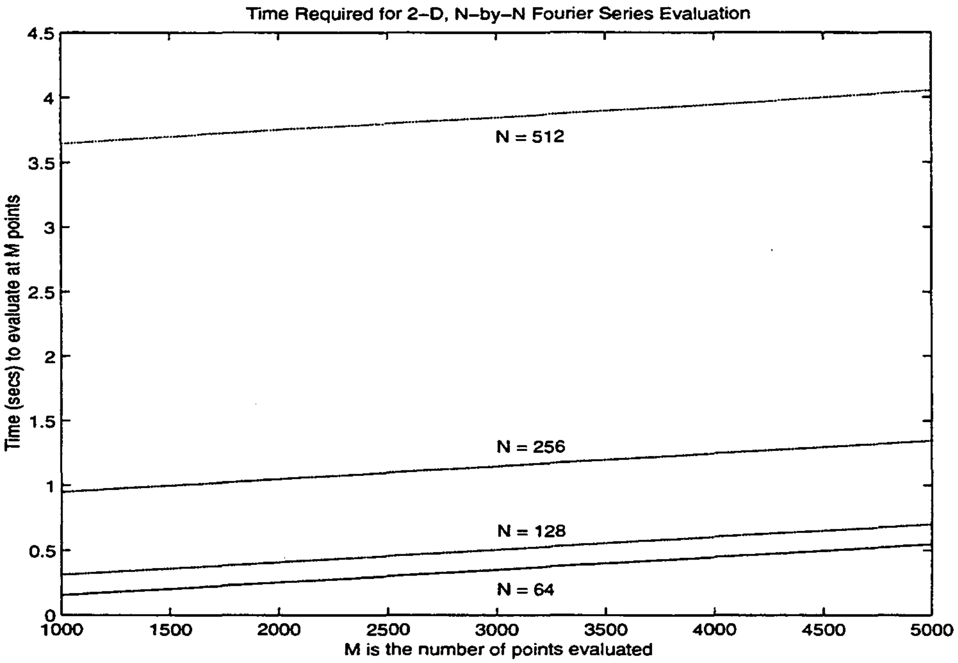


Figure 7: Complexity of Algorithm

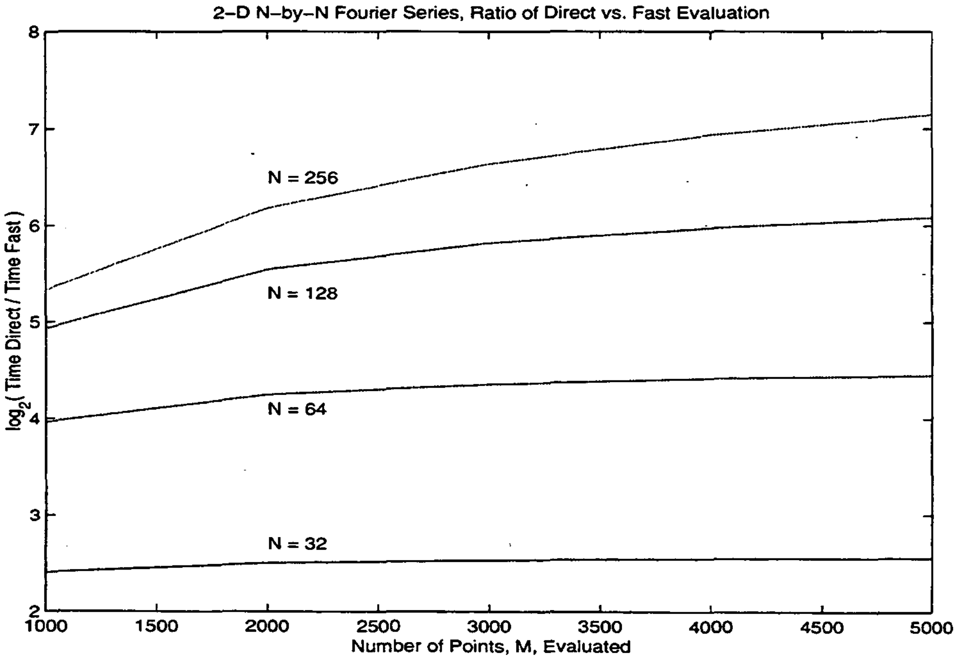


Figure 8: Comparison with Direct Evaluation

of the framework nature of our method, a software package that allows the user to chose from a number of windowing functions based on the need for accuracy and the constraints on memory usage would be a good tool. Moreover, because of the availability of different window functions, some with lower accuracy but faster computation, the inverse problems considered in [5] and shown to be better solved by multipole methods may get the benefit of preconditioners produced by lower accuracy windows.

Second, it is well-known that performing Discrete Fourier Transform for a general length N is still somewhat unsettled. In particular, when N is a large prime value, there are really only two practical choices for a $O(N \log N)$ algorithm. The Rader factorization and the chirp-z transform. The former seems to be difficult to implement for general N [4]. The latter, while general, requires the cost of three FFT's of length at least double the original length [11]. Consequently, the method here provides a viable alternative. Given a length N, we can represent N as a sum of smaller arbitrary lengths, each of which is "friendly" to traditional FFT algorithms. Thus $N = \sum_{j=1}^{k} N_j$. We then use our method to compute the each of the k Fourier series at the N points corresponding to the original Discrete Fourier Transform. The total cost is $O(kN) + \sum_{j=1}^{k} N_j \log(N_j)$. Clearly, we can make this comparable to $N \log(N)$, by for example, picking N_1 to be the largest power of 2 that is smaller than N; N_2 to be the largest power of 2 smaller than $N - N_1$, etc. The key to success lies in how efficiently we can implement the interpolation scheme.

References

[1] John P. Boyd, A Fast Algorithm for Chebyshev, Fourier, and Sinc Interpolation onto an Irregular Grid, *Journal of Computational Physics*, v103, pp. 243–257, 1992.

[2] John P. Boyd, Multipole Expansions and Pseudospectral Cardinal Functions: A New Generalization of the Fast Fourier Transform, *Journal of Computational Physics*, v103, pp. 184–186, 1992.

[3] E. Oran Brigham, *The Fast Fourier Transform and Its Applications*, Prentice Hall, New Jersey, 1988.

[4] C. Sidney Burrus and Ivan W. Selesnick, Automatic Generation of Prime Length FFT Programs, *IEEE Transactions on Signal Processing*, v 44, no.1, January 1996, pp. 14–24.

[5] A. Dutt and V. Rokhlin, Fast Fourier Transforms for Nonequispaced Data, *SIAM Journal on Scientific Computing*, v 14, no.6, November 1993, pp. 1368–1393.

[6] A. Dutt and V. Rokhlin, Fast Fourier Transforms for Nonequispaced Data, II, *Applied and Computational Harmonic Analysis*, v 2, no.1, January 1995, pp. 85–100.

[7] A. Dutt and M. Gu and V. Rokhlin, Fast Algorithms for Polynomial Interpolation, Integration, and Differentiation, *SIAM Journal on Numerical Analysis*, v33, no.5, October 1996, pp. 1689–1711.

[8] Itzhak Katznelson, *An Introduction to Harmonic Analysis*, 2nd edition, Dover, New York, 1976.

[9] Athanasios Papoulis, *The Fourier Integral and its Applications*, 2nd edition, McGraw-Hill, New York, 1984.

[10] P. Stpiczynski and M. Paprzycki, Parallel Algorithms for Finding Trigonometric Sums, *Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing*, SIAM, Philadelphia, pp. 291–292, 1995.

[11] Charles Van Loan, *Computational Frameworks for the Fast Fourier Transform*, SIAM, Philadelphia, 1992.

Ernest Orlando Lawrence Berkeley National Laboratory
One Cyclotron Road | Berkeley, California 94720

Prepared for the U.S. Department of Energy under Contract No. DE-AC03-76SF00098