Journal of Scientific Computing manuscript No.
(will be inserted by the editor)

The Partial Fast Fourier Transform

John C. Bowman · Zayd Ghoggali

Submitted to Journal of Scientific Computing: April 10, 2017

Abstract An efficient algorithm for computing the one-dimensional partial fast Fourier transform $f_j = \sum_{k=0}^{c(j)} e^{2\pi ijk/N} F_k$ is presented. Naive computation of the partial fast Fourier transform requires $\mathcal{O}(N^2)$ arithmetic operations for input data of length N. Unlike the standard fast Fourier transform, the partial fast Fourier transform imposes on the frequency variable k a cutoff function $c(j)$ that depends on the space variable j; this prevents one from directly applying standard FFT algorithms. It is shown that the space–frequency domain can be partitioned into rectangular and trapezoidal subdomains over which efficient algorithms can be developed. As in the previous work of Ying and Fomel (2009), the contribution from rectangular regions can be reduced to a series of fractional-phase Fourier transforms over squares, each of which can be reduced to a convolution. In this work, we demonstrate that the partial Fourier transform over trapezoidal domains can also be reduced to a convolution. Since the computational complexity of a dealiased convolution of N inputs is $\mathcal{O}(N \log N)$, a fast algorithm for the partial Fourier transform is achieved, with a lower overall coefficient than obtained by Ying and Fomel.

Keywords partial Fourier transform · fast Fourier transform · FFT · discrete Fourier transform · fractional-phase Fourier transform · convolution · implicit dealiasing

Mathematics Subject Classification (2000) 65T50 · 86A15

This work was supported by the Natural Sciences and Engineering Research Council of Canada and is based on a report submitted to the University of Alberta in partial fulfillment of the requirements for the degree of Master of Science.

John C. Bowman
Department of Mathematical and Statistical Sciences, University of Alberta, Edmonton, Alberta T6G 2G1, Canada. E-mail: bowman@ualberta.ca

Zayd Ghoggali
Department of Mathematical and Statistical Sciences, University of Alberta, Edmonton, Alberta T6G 2G1, Canada. E-mail: zayd_gho@yahoo.fr

1 Historical context

The discrete Fourier transform (DFT) is one of the most important tools used to perform spectral analysis in many real-world applications. For example, it is used to obtain the spectrum (frequency distribution) of a signal and to accelerate the computation of discrete convolutions [17]. The DFT is the discrete counterpart of the continuous Fourier transform (CFT). In practice, the values of a given signal are obtained either via a digital computer or measured at discrete time intervals; the signal values are thus only known on a discrete grid [1].

The DFT is the natural tool for computing the Fourier transform of discrete data. Even in the case of continuous input signals that are explicitly defined by analytic expressions, the DFT often provides a more convenient and practical approach than the CFT, especially when it comes to getting a global picture of the frequency spectrum of the input signal: it is often very hard to compute the CFT analytically due to the difficulty of expressing the required integrals in closed form [1]. The use of the DFT has expanded quickly over the past few decades. It has found many diverse applications such as digital filtering, digital image processing, digital communications, seismic processing, medical electronics, and wildlife acoustic detection. The motivation behind developing fast DFT and convolution algorithms stems from the huge computational cost of a direct implementation: a DFT or a convolution of length N has a computational complexity proportional to N^2, which quickly becomes excessive for large data sets [21].

Fortunately, the computational cost of a DFT can be dramatically reduced by using a fast Fourier transform (FFT) algorithm, an important example of which was introduced in 1965 by J. W. Cooley and J. W. Tukey [8]. They showed that a DFT of highly composite length N can be computed using $\mathcal{O}(N \log N)$ operations for $N \gg 1$. Since discrete convolutions can be computed via DFTs, the FFT algorithm can be used as well to reduce the number of arithmetic operations required to compute a discrete convolution from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$ [21]. The Cooley–Tukey algorithm was a turning point in the field of digital signal processing and had a revolutionary effect on numerical analysis [17]. For example, directly computing a DFT of length 1024 requires about 10^6 operations. However, only 10^4 operations are needed when an FFT algorithm is used, an improvement by a factor of a 100. This factor $(N/\log_2 N)$ becomes larger as the length of the input data N increases.

Special cases of the FFT were known long before 1965 [13]; in fact Gauss used an algorithm quite similar to the FFT to compute the coefficients of a finite Fourier series [16]. This calculation appeared in his papers dated 1805 and was published after his death in his collected works [12]. More recently, many new FFT algorithms beyond the Cooley–Tukey algorithm have been proposed to generalize and/or further decrease the computational complexity of the DFT. In particular, Bluestein in 1968 developed an FFT algorithm [4] that computes the discrete Fourier transform (DFT) for arbitrary lengths with multiplicative complexity of order $\mathcal{O}(N \log N)$. The Bluestein algorithm (also known as the chirp z-transform algorithm) uses an algebraic identity to express the DFT as a linear convolution of two sequences; the convolution is then computed efficiently using an FFT, after taking care to remove unwanted aliases (cyclic harmonics). Unlike some other FFT algorithms, Bluestein's FFT algorithm is not restricted to power-of-two lengths and can be used to compute more general transforms [24].

The present work is heavily based on Bluestein's algebraic identity. Let us begin by defining the DFT. It is convenient to introduce the Nth primitive root of unity, $\zeta_N \doteq \exp(2\pi i/N)$, where $\doteq$ is used to emphasize a definition. The N-point unnormalized discrete Fourier transform DFT of a complex input vector $\{F_k : k = 0, \ldots, N-1\}$ can be expressed as a weighted sum of powers of ζ_N:

$$f_j \doteq \sum_{k=0}^{N-1} \zeta_N^{jk} F_k, \qquad j = 0, \ldots, N-1. \tag{1.1}$$

The FFT exploits the properties that $\zeta_N^r = \zeta_{N/r}$ and $\zeta_N^N = 1$ for any divisor r of N.

The *partial Fourier transform* discussed in this work restricts the summation over k in Eq. (1.1) with a spatially dependent cutoff $c(j)$:

$$f_j \doteq \sum_{k=0}^{c(j)} \zeta_N^{jk} F_k, \qquad j = 0, \ldots, N-1.$$

The important special case where $c(j)$ is a constant $K < N$ is equivalent to a *pruned Fourier transform* [19], where all but K of the inputs are known to be zero. The corresponding inverse transform is called a *sparse Fourier transform* [27], where all but K of the outputs are known to be zero. For $K \ll N$, practical algorithms with complexity $\mathcal{O}(K \log N)$ for the sparse Fourier transform have been developed [15]. Optimized $\mathcal{O}(N \log N)$ algorithms for the case $K = N/2$ and $K = 2N/3$ have also been developed for dealiasing linear convolutions [5, 25].

In contrast to these previous studies, we consider the one-dimensional partial Fourier transform with a general cutoff $c(j)$ on the frequency variable k, as illustrated in Figures 2–4. Such transforms arise in specialized applications in geophysics [28] and inertial-range turbulence theory. The dependence of k on j through the cutoff $c(j)$ prevents one from using standard FFT algorithms. For some choices of $c(j)$, the direct computation of the partial Fourier transform can require up to $\mathcal{O}(N^2)$ computations for input data of size N. This can be expensive, especially when the length of the input data is large. We motivate the interest in efficiently computing the partial Fourier transform with an application in Section 2 and review related literature in Section 3. In Section 4, we describe the fractional-phase Fourier transform, which we use in Section 5 to develop an efficient algorithm that reduces the cost of computing the partial Fourier transform from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log^2 N)$, with an overall coefficient significantly smaller than that found by Ying and Fomel [28]. We describe an efficient numerical implementation of our algorithms in Section 6 and summarize the contributions of this work in Section 7.

2 Motivating application

Partial Fourier transforms arise naturally in certain physical applications. One challenging application that motivates the development of fast algorithms for partial Fourier transforms is the computation of integrals subject to certain constraints. Such integrals appear in reflection seismology when extrapolating wavefields using seismic migration methods like the phase-shift-plus-interpolation (PSPI)

method. This method was introduced in 1984 by Gazdag and Sguazzero as a generalization of the phase-shift method to allow lateral velocity variations [3]. The PSPI method is based on propagating the wavefield at various depths using a suitable set of reference velocities to obtain the multi-reference wavefields in the frequency–wavenumber domain [3,26]. The wavefields obtained by these constant-velocity continuations are then transformed back into the space domain. Based on the relationship between the medium velocity and the reference velocities, the final result is obtained by interpolating the reference wavefields in the frequency–space domain [3,26].

The two space-domain wavefield extrapolated with wave propagation velocity $v(x)$ at depth z using the PSPI algorithm is

$$\Psi_{\mathrm{PSPI}}(x,z,\omega) = \frac{1}{2\pi}\int_{\mathbb{R}} \hat{\psi}(k_x,0,\omega)\,\alpha(k_x,x,\omega)e^{ik_x x}dk_x,$$

where x is the horizontal spatial position, z is the depth, ω is the frequency, and k_x is the horizontal wave number [9]. Here $\hat{\psi}(k_x,0,\omega)$ refers to the wavenumber domain wavefield at $z=0$, which is given by the spatial Fourier transform of the space-domain wavefield $\Psi(x,0,\omega)$ at the surface $z=0$:

$$\hat{\psi}(k_x,0,\omega) = \int_{\mathbb{R}} \Psi(x,0,\omega)e^{-ik_x x}\,dx.$$

The phase shift operator $\alpha(k_x,x,\omega)$ is given by

$$\alpha(k_x,x,\omega) = \begin{cases} e^{izk_z} & \text{if } |k_x| \le \frac{\omega}{v(x)}, \\ e^{-|zk_z|} & \text{if } |k_x| > \frac{\omega}{v(x)}, \end{cases}$$

where $k_z = \frac{\omega}{v(x)}\sqrt{1-\left(\frac{v(x)k_x}{\omega}\right)^2}$ is the vertical wave number. One typically assumes that the wave propagation velocity $v(x)$ depends only on the horizontal coordinate x and not on the depth z.

In seismic imaging, one is often interested in the propagating modes that arise from the constraint $|k_x| \le \omega/v(x)$. This leads to the restricted integral

$$\Psi_{\mathrm{PSPI}}(x,z,\omega) = \frac{1}{2\pi}\int_{|k_x|\le\frac{\omega}{v(x)}} e^{iz\frac{\omega}{v(x)}\sqrt{1-\left(\frac{v(x)k_x}{\omega}\right)^2}}e^{ixk_x}\hat{\psi}(k_x,0,\omega)\,dk_x.$$

For computational efficiency, the factor $e^{iz\frac{\omega}{v(x)}\sqrt{1-\left(\frac{v(x)k_x}{\omega}\right)^2}}$ can be approximated by the first few terms of a sum of product of two sequences of functions $\{f_m\}$ and $\{h_m\}$:

$$\sum_m f_m(k_x)h_m(x),$$

provided that z is sufficiently small [28]. Taking this approximation into consideration, one can write:

$$\Psi_{\mathrm{PSPI}}(x,z,\omega) \approx \sum_m h_m(x)\int_{|k_x|\le\frac{\omega}{v(x)}} f_m(k_x)\hat{\psi}(k_x,0,\omega)e^{ixk_x}\,dk_x.$$

The task therefore reduces to computing the integral

$$\int_{|k_x|\le \frac{\omega}{v(x)}} f_m(k_x)\hat{\psi}(k_x,0,\omega)e^{ixk_x}dk_x,$$

which, after discretization of the domain, can be written as a discrete partial Fourier transform.

3 Related work

A closely related work was published by Ying and Fomel [28]. They proposed fast algorithms to compute the partial Fourier transform in one and two dimensions. The goal of the current work is to improve on their results by using a different decomposition of the space–frequency domain. Ying and Fomel computed the one-dimensional partial Fourier transform

$$f_j \doteq \sum_{|k|\le c(j)} \zeta_N^{jk} F_k, \qquad j = 0,\dots,N-1$$

by constructing a recursive multiscale decomposition of the summation domain $D = \{(j,k) : |k| \le c(j),\ 0 \le j \le N-1,\ -N/2 \le k \le N/2-1\}$ into a set of squares with dyadic sizes, where the union of all these squares is exactly equal to D. The computation associated with each square inside the partitioned domain D is accelerated using the fractional-phase Fourier transform. The resulting numerical algorithm scales as $\mathcal{O}(N\log^2 N)$ for input data of size N.

To compute the two-dimensional partial Fourier transform

$$f_{\boldsymbol{j}} \doteq \sum_{|\boldsymbol{k}|\le c(\boldsymbol{j})} \zeta_N^{\boldsymbol{j}\cdot\boldsymbol{k}} F_{\boldsymbol{k}},$$

where $\boldsymbol{j} = (j_1, j_2)$ and $\boldsymbol{k} = (k_1, k_2)$, Ying and Fomel decomposed the summation domain $D = \{(j_1, j_2, r) : 0 \le j_1, j_2 \le N-1,\ 0 \le r \le c(j_1, j_2)\}$ into a set of cubes with dyadic sizes having a union equal to D. The projection of each cube onto the r axis is a dyadic interval I. The computation associated with each dyadic interval I involves interactions between points from the circular bands $\{(k_1,k_2) \in \mathbb{Z}\times\mathbb{Z} : -N/2 \le k_1, k_2 \le N/2-1\}$ and $\{(j_1,j_2) \in \mathbb{Z}\times\mathbb{Z} : 0 \le j_1, j_2 \le N-1\}$. This interaction is a special case of the sparse Fourier transform [27], where both the frequency data and the spatial data are sparse. The butterfly algorithm suggested in Refs. [20,22,23] is used to accelerate the computation of this sparse Fourier transform. The resulting algorithm is almost linear for an input of size $N\times N$ and its overall complexity is $\mathcal{O}(N^2\log^2 N)$ [28].

A second related work by Hairer, Lubich, and Schlichte was dedicated to the computation of the fast temporal convolution [14]. For a given sequence $\{F_k\}_{k=0}^{N-1}$, the discrete temporal convolution is given by

$$f_j = \sum_{k=0}^{j} F_k g_{j-k}, \qquad j = 0,\dots,N-1,$$

where $\{g_j\}_{j=0}^{N-1}$ are the convolution weights, which can be constructed from the Laplace transform of the kernel function for the continuous convolution. Hairer,

Lubich, and Schlichte suggested an algorithm that first partitions the triangular summation domain $D = \{(j,k) : 0 \le k < j \le N-1\}$ recursively into a set of squares [14,28]. The union of these squares equals the entire domain D. The FFT is used to accelerate the computation of the convolution in each square within this partition. The overall cost of this one-dimensional algorithm is $\mathcal{O}(N \log^2 N)$ arithmetic operations for input data of size N. An issue arises when handling non-reflecting boundary conditions, where all quadrature weights and past boundary values need to be kept in active memory during the entire computation process. This prevents the algorithm from being extended to solve three-dimensional problems over long periods of time [14,18].

4 The fractional-phase Fourier transform

The definition of the DFT can be generalized to the so-called discrete fractional-phase Fourier transform (FPFFT), where the fractional root $\zeta^\alpha \doteq \zeta_{1/a} = e^{2\pi i \alpha}$ is used instead of the primitive root ζ_N. The FPFFT and its fast algorithm are useful in many applications, such as high-resolution trigonometric interpolation and computing DFTs of prime lengths or sparse sequences [2]. For an input vector $\{F_k : k = 0, \ldots, N-1\}$ of length N, the discrete fractional-phase Fourier transform is defined as

$$f_j \doteq \sum_{k=0}^{N-1} \zeta^{\alpha j k} F_k, \qquad j = 0, \ldots, N-1, \tag{4.2}$$

where $\zeta \doteq \zeta_1$. Here α is not restricted to a rational number; it can be an arbitrary real number. Note that by setting $\alpha = 1/N$, the FPFFT reduces to the ordinary DFT.

Direct computation of the N values in the above definition requires $8N^2$ arithmetic operations, provided the roots of unity have been precomputed. Fortunately, Bailey and Swarztrauber derived an efficient algorithm for computing the FPFFT using only $20N \log_2 N$ floating point operations [2]. The idea behind their algorithm was originally introduced by Bluestein [4], who expressed the chirp z-transform as an FFT-based convolution. Bailey and Swarztrauber's algorithm can be derived by rewriting the power jk of the root of unity in Eq. (4.2) using the algebraic identity $jk = \frac{1}{2}[j^2 + k^2 - (j-k)^2]$.

The discrete fractional-phase Fourier transform can then be rewritten as

$$f_j = \sum_{k=0}^{N-1} \zeta^{\frac{\alpha}{2}[j^2+k^2-(j-k)^2]} F_k = \zeta^{\alpha j^2/2} \sum_{k=0}^{N-1} \zeta^{\alpha k^2/2} F_k \zeta^{-\alpha(j-k)^2/2}. \tag{4.3}$$

On defining $g_k = \zeta^{\alpha k^2/2}$ and $h_k = g_k F_k$, Eq. (4.3) can then be expressed in terms of a discrete convolution:

$$f_j = g_j \sum_{k=0}^{N-1} h_k \overline{g}_{j-k}, \tag{4.4}$$

where the bar denotes complex conjugation.

The convolution in Eq. (4.4) can be computed using discrete FFTs and the convolution theorem. However, the convolution theorem can be used only for circular convolutions, where $g_{j-k} = g_{j-k+N}$. This condition fails to hold in this case, due to the symmetry $g_{j-k} = g_{k-j}$ when $j - k < 0$.

As described in Ref. [2], it is possible to convert the symmetric convolution in Eq. (4.4) into a circular convolution using zero padding. However, a better alternative, known as implicit dealiasing [5,25], unrolls and truncates the outer iteration of the fast Fourier transform to avoid the need for explicit zero padding. To compute the first N outputs $\{f_j : 0 \le j \le N-1\}$ via implicit dealiasing, one precomputes the inverse FFTs

$$\{G_j\}_{j=0}^{N-1} = \mathtt{fft}^{-1}(\{\zeta^{-\alpha k^2/2} + \zeta^{-\alpha(k-N)^2/2}\}_{k=0}^{N-1})$$

and

$$\{V_j\}_{j=0}^{N-1} = \mathtt{fft}^{-1}(\{\zeta_{2N}^{k}[\zeta^{-\alpha k^2/2} - \zeta^{-\alpha(k-N)^2/2}]\}_{k=0}^{N-1}).$$

Then f_j can be calculated as the product of g_j and the jth output of Function conv applied to the sequence $\{h_k\}_{k=0}^{N-1}$. In the listed pseudocode, an asterisk ($*$) denotes an element-by-element (vector) multiply.

Input: vector f
Output: vector f
for $k = 0$ **to** $N - 1$ **do**
  u[k] $\leftarrow \zeta_{2N}^{k}$f[$k$]
end
f $\leftarrow$ fft(fft^{-1}(f) $*$ G)
u $\leftarrow$ fft(fft^{-1}(u) $*$ V)
for $k = 0$ **to** $N - 1$ **do**
  f[k] $\leftarrow$ f[k] $+ \zeta_{2N}^{-k}$u[k]
end
return f/(2N)

Function `conv(f)` computes the in-place implicitly dealiased symmetric convolution in Eq. (4.4) of the complex vector f using a temporary vector u of length N and precomputed inverse transforms G and V.

5 Evaluating partial FFTs via convolutions

The main idea behind our approach is to decompose the space–frequency domain $D = \{(j,k) : \ 0 \le j \le N-1, 0 \le k \le \min\{c(j), N-1\}\}$ into a mesh consisting of rectangles and trapezoids and implement algorithms for each of these special cases. The rectangles will be subdivided into squares over which a fractional-phase Fourier transform can be performed. In contrast, Ying and Fomel [28] decomposed the space–frequency domain D only into squares. Our algorithm can thus be considered as a higher-order version of their subdivision scheme since it uses a piecewise linear rather than a piecewise constant approximation to the constraint function.

Once the mesh is constructed, we compute the partial Fourier transform within each trapezoid as a convolution, using the identity $nk = \frac{1}{2}[n^2 + k^2 - (n-k)^2]$ originally introduced by Bluestein [4] for computing the full Fourier transform and later exploited by Bailey and Swarztrauber [2] for computing fractional-phase Fourier transforms. We consider the case where the cutoff function is an affine function with rational coefficients $c(j) = (pj+s)/q$, where p, q, and s are integers, illustrating the technique first in the special case where $p = q = 1$ and $s = 0$.

5.1 Partial fractional-phase FFT: special case $c(j) = j$

It is convenient to define $\zeta^\alpha \doteq \zeta_{1/a} = e^{2\pi i\alpha}$ so that the unnormalized discrete partial Fourier transform of a complex vector $\{F_k : k = 0, \ldots, N-1\}$ can be written as

$$f_j \doteq \sum_{k=0}^{j} \zeta^{\alpha jk} F_k, \qquad j = 0, \ldots, N-1.$$

The identity $jk = \frac{1}{2}\left[j^2 + k^2 - (j-k)^2\right]$ allows us to express the discrete partial Fourier transform as

$$f_j = \sum_{k=0}^{j} \zeta^{\frac{\alpha}{2}\left[j^2+k^2-(j-k)^2\right]} F_k = \zeta^{\alpha j^2/2} \sum_{k=0}^{j} \zeta^{\alpha k^2/2} F_k \zeta^{-\alpha(j-k)^2/2}.$$

On defining $g_j = \zeta^{\alpha j^2/2} = \zeta_2^{\alpha j^2}$ and $h_k = g_k F_k$, we see that f_j can be expressed as the product of g_j and a convolution of the sequences $\{h_k\}_{k=0}^{N-1}$ and $\{g_k\}_{k=0}^{N-1}$:

$$f_j = g_j \sum_{k=0}^{j} h_k \overline{g}_{j-k}. \tag{5.5}$$

By prepending N zero elements to the sequences $\{g_k\}$ and $\{h_k\}$, indexed as $k = -N,\ -N+1,\ \ldots,\ -1$, the convolution $\sum_{k=0}^{j} h_k \overline{g}_{j-k}$ can be efficiently computed using a cyclic discrete Fourier transform of length $2N$, on noting that $\bar{g}_{j-k} = 0$ when $k > j$:

$$\begin{aligned}
\sum_{k=0}^{j} h_k \overline{g}_{j-k} = \sum_{k=-N}^{N-1} h_k \overline{g}_{j-k} &= \frac{1}{(2N)^2} \sum_{k=-N}^{N-1} \sum_{\ell=0}^{2N-1} \zeta_{2N}^{-k\ell} H_\ell \sum_{m=0}^{2N-1} \zeta_{2N}^{-(j-k)m} G_m \\
&= \frac{1}{(2N)^2} \sum_{\ell=0}^{2N-1} \sum_{m=0}^{2N-1} \zeta_{2N}^{-jm} H_\ell G_m \sum_{k=-N}^{N-1} \zeta_{2N}^{k(m-\ell)} \\
&= \frac{1}{(2N)^2} \sum_{\ell=0}^{2N-1} \sum_{m=0}^{2N-1} \zeta_{2N}^{-jm} H_\ell G_m 2N\delta_{\ell m} \\
&= \frac{1}{2N} \sum_{\ell=0}^{2N-1} \zeta_{2N}^{-j\ell} H_\ell G_\ell,
\end{aligned} \tag{5.6}$$

where $H_\ell = \sum_{k=-N}^{N-1} \zeta_{2N}^{\ell k} h_k$ and $G_m = \sum_{k=-N}^{N-1} \zeta_{2N}^{mk} \overline{g}_k$ are the discrete Fourier transforms of the zero-padded sequences $\{h_k\}$ and $\{\overline{g}_k\}$ of length $2N$, respectively.

The zero padding of the sequences $\{h_k\}$ and $\{\overline{g}_k\}$ removes unwanted aliases (harmonics) that otherwise enter when computing a linear convolution in terms of cyclic discrete Fourier transforms. However, we again use implicit dealiasing [5, 25] since it is more efficient than explicit zero padding. If we precompute

$$\{G_j\}_{j=0}^{N-1} = \texttt{fft}^{-1}(\{\zeta^{-\alpha k^2/2}\}_{k=0}^{N-1})$$

and

$$\{V_j\}_{j=0}^{N-1} = \texttt{fft}^{-1}(\{\zeta_{2N}^{k}\zeta^{-\alpha k^2/2}\}_{k=0}^{N-1}),$$

the value of f_j is efficiently determined as the product of g_j and the jth output of Function conv applied to the input data $\{h_k\}_{k=0}^{N-1}$. The overall computational cost of computing Eq. (5.5) is $\mathcal{O}(N \log N)$.

5.2 Partial fractional-phase FFT: trapezoidal constraint $c(j) = (pj + s)/q$

We now illustrate how the previous technique can be generalized to handle the trapezoidal constraint $c(j) = (pj + s)/q$, where p, q, and s are integers, with $p \neq 0$ and $q > 0$. In practical applications we will be interested in the case where s is a multiple of q, in which case, without loss of generality we may take p and q to be coprime. The unnormalized partial Fourier transform appears as

$$f_j \doteq \sum_{k=0}^{\lfloor (pj+s)/q \rfloor} \zeta^{\alpha jk} F_k, \qquad j = 0, \ldots, M-1,$$

with $\lfloor (p(M-1)+s)/q \rfloor = N-1$.

For each $j \in \{0, \ldots, M-1\}$, we decompose $pj + s = qn + r$ such that $r \in \{0, \ldots, q-1\}$ and $n \in \{0, \ldots, N-1\}$. We may then express

$$\begin{aligned} f_j &= \sum_{k=0}^{n} \zeta_p^{\alpha(qn+r-s)k} F_k \\ &= \sum_{k=0}^{n} \zeta_{2p}^{\alpha q\left[n^2+k^2-(n-k)^2\right]} \zeta_p^{\alpha(r-s)k} F_k \\ &= \zeta_{2p}^{\alpha q n^2} \sum_{k=0}^{n} \zeta_{2p}^{-\alpha q(n-k)^2} \zeta_{2p}^{\alpha q k^2} \zeta_p^{\alpha(r-s)k} F_k. \end{aligned} \tag{5.7}$$

On setting $g_k = \zeta_{2p}^{\alpha q k^2}$ and $h_k = g_k \zeta_p^{\alpha(r-s)k} F_k$, the result can again be written as a convolution of two sequences $\{h_k\}_{k=0}^{N-1}$ and $\{g_k\}_{k=0}^{N-1}$:

$$f_j = g_n \sum_{k=0}^{n} h_k \overline{g}_{n-k}, \qquad j = 0, \ldots, M-1. \tag{5.8}$$

As long as one of the coprime integers p and q is equal to one, the complexity of this algorithm is readily seen to be $\mathcal{O}(N \log N)$. The case where the slope $p/q = 0$ reduces to the rectangular case, which we describe next.

5.2.1 Fractional-phase FFT: $M \times N$ rectangle

If $N > M$ we pad the N inputs with zeros to obtain an $M \times pM$ rectangle, where $p = \lceil N/M \rceil$. The fractional-phase FFT over this extended domain can then be computed as described in Section 5.2.2. Otherwise, if $N \leq M$, we define $q = \lceil M/N \rceil$ and compute the fractional-phase FFT over a $qN \times N$ rectangle, as described in Section 5.2.3, and then discard all but the first M values of j.

5.2.2 Fractional-phase FFT: $M \times pM$ rectangle for $p \in \mathbb{N}$

In this case, we wish to compute

$$f_j = \sum_{k=0}^{pM-1} \zeta^{\alpha jk} F_k, \qquad j = 0, \ldots, M-1.$$

On decomposing $k = mp + r$, we find

$$f_j = \sum_{r=0}^{p-1} \zeta^{\alpha jr} \sum_{m=0}^{M-1} \zeta^{\alpha pjm} F_{mp+r}, \qquad j = 0, \ldots, M-1.$$

5.2.3 Fractional-phase FFT: $qN \times N$ rectangle for $q \in \mathbb{N}$

To compute

$$f_j = \sum_{k=0}^{N-1} \zeta^{\alpha jk} F_k, \qquad j = 0, \ldots, qN-1,$$

we decompose $j = mq + r$ and obtain

$$f_{mq+r} = \sum_{k=0}^{N-1} \zeta^{\alpha qmk} \zeta^{\alpha rk} F_k, \qquad m = 0, \ldots, N-1, \quad r = 0, \ldots, q-1.$$

To complete our implementation of a subdivision procedure, we need to shift the partial FFT over a rectangular or trapezoidal cell with up to N inputs and M outputs, as described next.

5.2.4 Shifted $M \times N$ partial FFT

For any constraint $c(j)$, the partial FFT

$$f_j = \sum_{k=k_0}^{c(j)} \zeta^{\alpha jk} F_k, \qquad j = j_0, \ldots, j_0 + M - 1$$

can be computed by changing to the variables $k' = k - k_0$ and $j' = j - j_0$, so that

$$f_{j_0+j'} = \zeta^{\alpha(j_0+j')k_0} \sum_{k'=0}^{c(j_0+j')-k_0} \zeta^{\alpha j'k'} \zeta^{\alpha j_0 k'} F_{k'+k_0}, \qquad j' = 0, \ldots, M-1.$$

6 Recursive subdivision scheme

We now show how the previous algorithms can be combined to compute the partial FFT recursively for a general constraint function $c(j)$. For simplicity, we assume in this discussion that the length N of the input data is a power of two. The domain is recursively partitioned into two equal halves in both the frequency and space directions until the resulting subdomain either (i) lies entirely below the constraint; (ii) lies entirely above the constraint; or (iii) contains only a piecewise linear portion of the constraint. In the latter case, the trapezoidal algorithm is used to calculate the partial FFT. Pseudocode for the resulting FFT algorithm is given in Algorithm 1.

Algorithm 1 `partition` subdivides $\{(j,k) : x_0 \le j \le x_1,\ y_0 \le k \le y_1\}$.

```
Input: int x0,x1,y0,y1, vectors {f_a}_{a=0}^{N-1}
empty ← true; straight ← true
for j = x0 to x1 do
    if y0 ≤ c[j] then
        below ← true
    if c[j] ≤ y1 then
        above ← true
    if y0 ≤ c[j] and c[j] < y1 then
        empty ←false; break
if empty then
    if below and above then
        empty ←false; straight ←false
else
    left ← first j ∈ [x0, x1) such that c(j) ∈ [y0, y1)
    right ← last j ∈ [x0, x1) such that c(j) ∈ [y0, y1)
    p/q ← round((c[x1 − 1] − c[x0])/(x1 − 1 − x0))
    for j = left to right do
        r ← c[left] ∗ q + (j − left) ∗ p − c[j] ∗ q
        if r < 0 or r ≥ q then
            straight ← false; break
if straight or empty or x0 ≥ x1 − 1 or y0 ≥ y1 − 1 then
    if x0 < x1 and y0 < y1 then
        if empty then
            if y − 1 ≤ c[x0] then
                Rectangle(x0, x1, y0, y1)
            else
                if x0 < left and y1 ≤ c[x0] then
                    Rectangle(x0, left, y0, y1)
                Trapezoid(left,right +1,y0,min(max(c[left], c[right]) + 1, y1), p, q)
                if right + 1 < x1 and y1 − 1 ≤ c[right + 1] then
                    Rectangle(right + 1, x1, y0, y1)
    return
if y1 − y0 ≤ x1 − x0 then
    partition(x0, (x0+x1)/2, y0, y1, {f_a}_{a=0}^{N-1})
    partition((x0+x1)/2, x1, y0, y1, {f_a}_{a=0}^{N-1})
else
    partition(x0, x1, y0, (y0+y1)/2, {f_a}_{a=0}^{N-1})
    partition(x0, x1, (y0+y1)/2, y1, {f_a}_{a=0}^{N-1})
```

6.1 Trigonometric lookup tables

To compute the partial fast Fourier transform $f_j = \sum_{k=0}^{c(j)} \zeta_N^{jk} F_k$, we need to precompute the factors $\{\zeta^{\alpha k} : k = 0, \dots B-1\}$, where $\alpha = A/B$ for coprime integers A and B. When sufficient low-latency memory is available, these trigonometric factors can be stored in a lookup table. However, for problems so large that these tables would not fit in the memory cache, it is advantageous to compute them "on the fly." While direct computation of $(\cos(2\pi\alpha k), \sin(2\pi\alpha k))$ is normally prohibitively expensive, there are two efficient alternatives: recursion and factorization. Although computation of the trigonometric factors ζ_B^{Ak} by straightforward recursion is numerically unstable, the stable trigonometric recursion described by Buneman [7,5] in terms of two small precomputed tables, each of size $\log_2 B$, could be used to compute the required roots of unity. However, on modern hardware it is more efficient to calculate ζ_B^{Ak} with a single complex multiply, using two short precomputed tables $H_a = \zeta_B^{as}$ and $L_b = \zeta_B^{b}$, where $Ak = as + b$ with $s = \lfloor\sqrt{B}\rfloor$, $a = 0, 1, \dots, \lceil B/s\rceil - 1$, and $b = 0, 1, \dots, s-1$. [5]. These one-dimensional tables require only $\mathcal{O}(\sqrt{B})$ complex words of storage. In our implementation, we adaptively choose between trigonometric lookup and factorization, depending on the problem size.

Since the fractional-phase FFT for N data points is computed as a discrete convolution, its computational cost is $\mathcal{O}(N \log N)$. The same applies to the trapezoidal partial FFT discussed in Section 5.2. Following [28], one can then compute an upper bound to the complexity for a general cutoff function $c(j)$. The contribution to the overall complexity from each rectangular or trapezoidal cell of size s that fully satisfies the constraint $k \le c(j)$ is $\mathcal{O}(s \log s)$. The number of cells of area $\mathcal{O}(s^2)$ that intersect the discrete curve $k = c(j)$ of length N can be approximated by $\mathcal{O}(Ns/s^2)$, each of which costs $\mathcal{O}(s \log s)$ operations. The complexity for cells of size s along a band of width s centered on the constraint is therefore $\mathcal{O}(N \log s)$. On summing this cost over all possible cell sizes $s = 2^p$, we obtain an upper bound to the complexity,

$$\mathcal{O}\left(N \sum_{p=1}^{\log_2 N} \log 2^p\right) = \mathcal{O}\left(N \sum_{p=1}^{\log_2 N} p\right) = \mathcal{O}\left(N \frac{\log_2 N(\log_2 N + 1)}{2}\right) = \mathcal{O}(N \log^2 N).$$

While the actual computational cost is a weighted sum of terms $\mathcal{O}(N \log^2 N)$ and $\mathcal{O}(N \log N)$, the dominant scaling for large N is thus seen to be $\mathcal{O}(N \log^2 N)$. However, in the special case of an affine constraint function $c(j)$ such that either the slope or its reciprocal is an integer, we have seen that the partial Fourier transform can be reduced without recursion to a convolution of two sequences of length N, yielding a slightly reduced complexity of $\mathcal{O}(N \log N)$.

6.2 Numerical Results

In this section, we demonstrate our hybrid algorithm for different constraint functions $c(j)$ and input data lengths N. The numerical computations were performed on a 64-bit 3.4GHz Intel i7-2600K processor with an 8MB cache. Like the `FFTW-3.3.4` and `FFTW++-2.03` libraries [6,10,11], our algorithms were vectorized

N	T_r	T_h	C_r	C_h	T_d/T_h	T_r/T_h	T_h/T_f
1024	0.00099	0.00070	2572	860	3.79	1.43	142
2048	0.00216	0.00152	5154	1700	6.94	1.42	127
4096	0.00456	0.00335	10168	3437	12.7	1.36	121
8192	0.00964	0.00724	20520	7132	23.7	1.33	120
16384	0.0208	0.0158	40934	13970	45.2	1.32	107
32768	0.0445	0.0345	81744	27688	89.4	1.29	104
65536	0.0965	0.0758	163694	55338	166	1.27	103
131072	0.213	0.172	327734	110755	298	1.24	108
262144	0.487	0.396	655300	222055	820	1.23	108
524288	1.08	0.891	1310532	441677	2440	1.22	83
1048576	2.32	1.98	2622408	884846	4950	1.17	80

Table 1 Computation statistics of the recursive one-dimensional partial Fourier transform with $c(j) = (N-1)\sin(\pi j/(N-1))$ for $N = 1024$ to $N = 1048576$. Here T_r and C_r are the computation time and cell count for rectangular subdivision, T_h and C_h are the computation time and cell count for hybrid rectangular/trapezoidal subdivision, T_d is the computation time for direct summation, and T_f is the computation time for an unconstrained FFT of length N.

with specialized single-instruction multiple-data (SIMD) code. They were compiled with the optimizations
`-fopenmp -fomit-frame-pointer -fstrict-aliasing -ffast-math -msse2 -mfpmath=sse -march=native`.

For the constraint function $c(j) = (N-1)\sin(\pi j/(N-1))$, with N ranging from 1024 to 1048576, we compare in Table 1 a decomposition (using rectangles only) equivalent to that used by Ying and Fomel [28] and our hybrid decomposition (using both rectangles and trapezoids) by tabulating the computation statistics and number of cells required for each subdivision. We use the following notation:
$c(j)$ is the cutoff function of the partial Fourier transform;
N is the length of the input data;
T_r is the computation time for rectangular subdivision;
C_r is the cell count for rectangular subdivision;
T_h is the computation time for hybrid subdivision;
C_h is the cell count for hybrid subdivision;
T_d is the computation time for direct evaluation of the partial Fourier transform;
T_f is the computation time for the unconstrained FFT of length N.

As explained in the previous section, the computational cost of our hybrid subdivision algorithm is $\mathcal{O}(N \log^2 N)$, far less than the $\mathcal{O}(N^2)$ direct evaluation cost. Therefore, as seen in Table 1, one should expect an (almost) linear growth in the ratio T_d/T_h with respect to the input data length N. The computation time T_h required by hybrid subdivision is 15–30% less than the time T_r required for rectangular subdivision. Furthermore, hybrid subdivision generates far fewer (roughly 34%) cells than produced by rectangular subdivision despite the restriction of the constraint slope p/q to integral or reciprocal integral values. These computation times are illustrated graphically in Figure 1 in comparison to the cost of an unconstrained FFT. For large N, we observe that the computational cost of a partial FFT scales as $\mathcal{O}(N \log^2 N)$, in agreement with the scaling derived in the previous section.

The geometrical differences between rectangular and hybrid subdivision are illustrated in Figure 2 for $c(j) = j$ with $N = 1024$ and in Figures 3 and 4 for $c(j) = (N-1)\sin(\pi j/(N-1))$ with $N = 128$ and $N = 1024$, respectively.

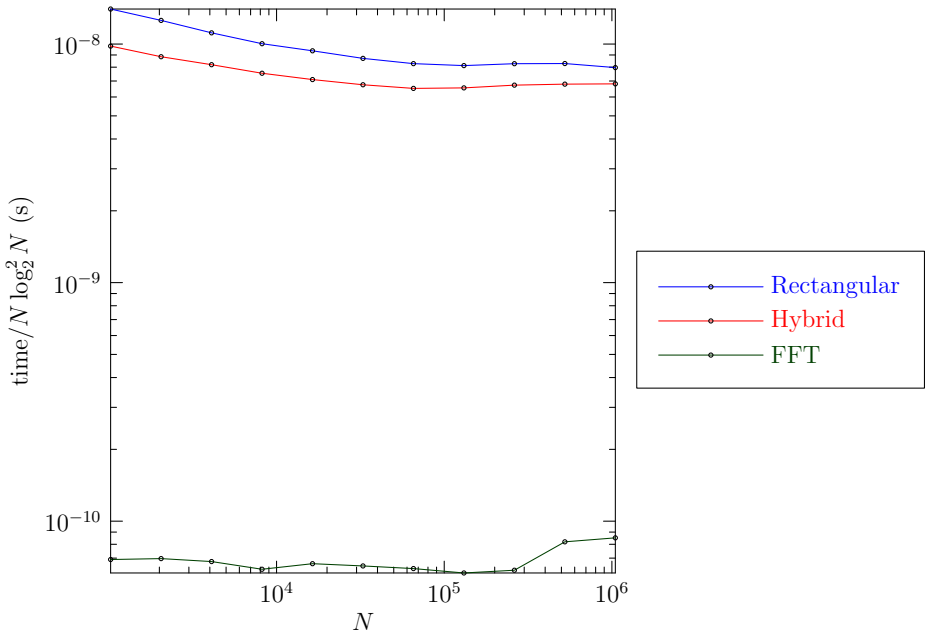
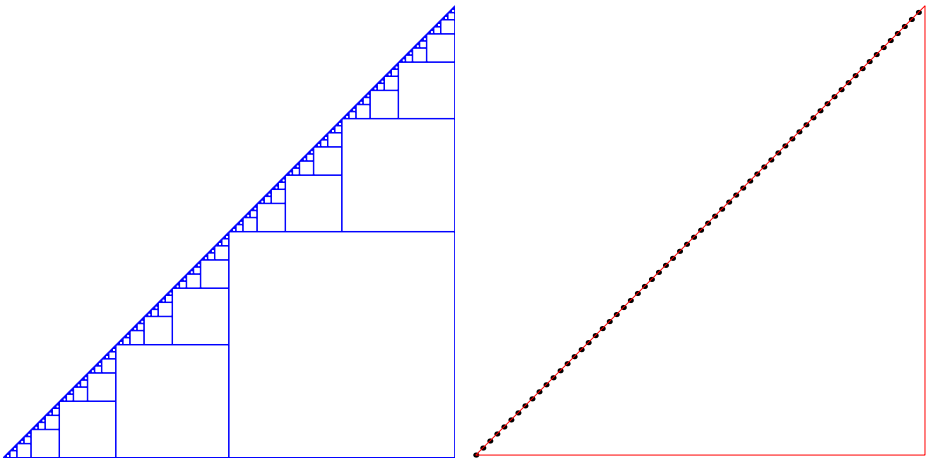
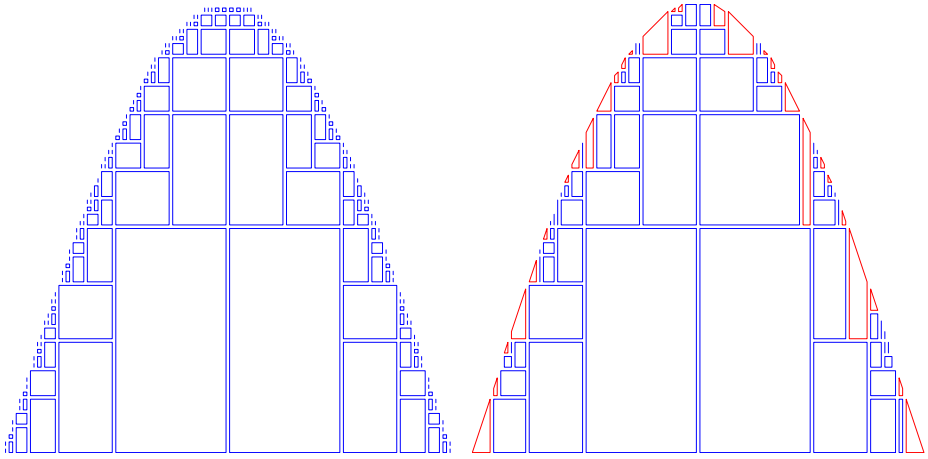


Figure 1 Computation times vs. vector length N for rectangular and hybrid subdivision in comparison with the cost of an unconstrained FFT.



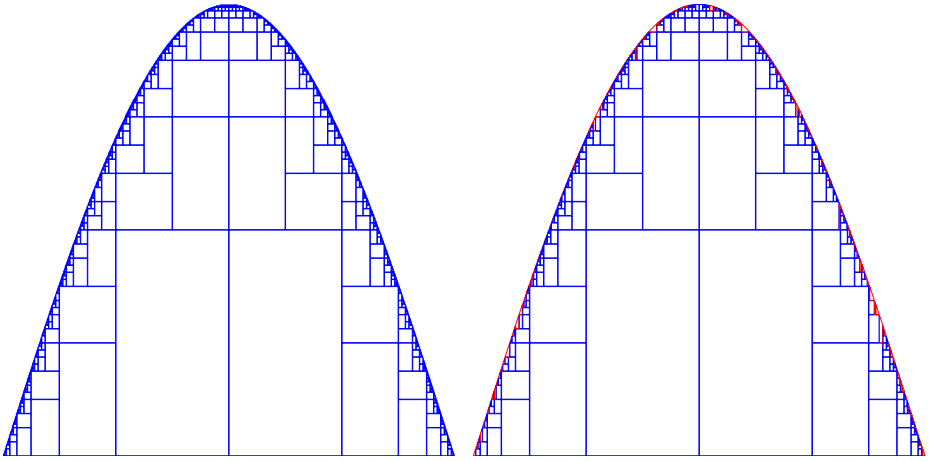
(a) (b)

Figure 2 (a) Subdivision of the summation domain for the triangular constraint $c(j) = j$ with $N = 1024$ using (a) a rectangular decomposition, with 1535 cells, and (b) a triangular decomposition, with only 1 cell. The black dots indicate $\{c(16j) : j = 0 \ldots 63\}$.



(a) (b)

Figure 3 Subdivision of the summation domain for the sinusoidal constraint $c(j) = (N-1)\sin(\pi j/(N-1))$ with $N = 128$ using (a) a rectangular decomposition, with 306 cells, and (b) a hybrid decomposition, with only 103 cells.



(a) (b)

Figure 4 Subdivision of the summation domain for the sinusoidal constraint $c(j) = (N-1)\sin(\pi j/(N-1))$ with $N = 1024$ using (a) a rectangular decomposition, with 2572 cells, and (b) a hybrid decomposition, with only 983 cells.

7 Conclusion

In this work, an efficient algorithm was developed to compute the partial Fourier transform in one dimension. The key idea is based on recursively subdividing the appropriate space–frequency domain $\{(j,k) : \ 0 \le j \le N-1, \quad 0 \le k \le \min(c(j), N-1)\}$ into smaller rectangles and trapezoids, and then applying fast algorithms on each of these resulting subdomains. The contribution to the partial

Fourier transform from a rectangular region that fully satisfies the constraint can be computed by subdividing the rectangle into squares, appropriately padded or truncated, over which the fractional-phase Fourier transform can then be computed. The contribution from trapezoidal regions can be reduced, with the help of the Bluestein identity, to an implicitly dealiased FFT-based convolution.

On comparing our hybrid subdivision algorithm with Ying and Fomel's algorithm [28] for different values of the input data length, we see that the hybrid algorithm is 15–30% faster and generates far fewer cells.

A future generalization of this work would be the extension of the subdivision recursion to handle input data lengths other than the (frequently encountered) case of powers of two. Generalizing this work to compute the one-dimensional partial Fourier transform where the frequency index k is subject to both lower and upper constraints requires a few minor changes to the current algorithm to handle the contributions from inverted trapezoids. Also, one should also consider the use of quasi-uniform grids in the frequency and space domains. In working with quasi-uniform grids, the major issue is that the fractional-phase Fourier transform cannot be applied directly because the points inside the grid are no longer uniformly distributed [28]. Instead, a butterfly algorithm like the one used in Ref. [27] is needed to do the computation inside each resulting rectangle of the partition. In a future paper we will demonstrate how nested implementations of this algorithm can be used to calculate partial Fourier transforms in two and three dimensions. In view of the closing remark of Ying and Fomel [28], another area of future research is to consider better approximations to the exponential phase $iz\frac{\omega}{v(x)}\sqrt{1-\left(\frac{v(x)k_x}{\omega}\right)^2}$ that arises in the constrained integral discussed in Section 2.

Acknowledgements The authors would like to thank Professor Brendan Pass for his comments on an earlier version of this work.

References

1. Amidror, I.: Mastering the discrete Fourier transform in one, two or several dimensions: pitfalls and artifacts, vol. 43. Springer Science & Business Media (2013)
2. Bailey, D.H., Swarztrauber, P.N.: The fractional Fourier transform and applications. SIAM review **33**(3), 389–404 (1991)
3. Biondi, B.: 3D seismic imaging. No. 14 in Investigations in Geophysics Series. Society of Exploration Geophysicists Tulsa (2006)
4. Bluestein, L.I.: A linear filtering approach to the computation of discrete Fourier transform. IEEE Trans. Audio and Electroacoustics **18**(4), 451–455 (1970)
5. Bowman, J.C., Roberts, M.: Efficient dealiased convolutions without padding. SIAM J. Sci. Comput. **33**(1), 386–406 (2011)
6. Bowman, J.C., Roberts, M.: `FFTW++:` A fast Fourier transform C^{++} header class for the `FFTW3` library. `http://fftwpp.sourceforge.net` (May 6, 2010)
7. Buneman, O.: Stable on-line creation of sines or cosines of successive angles. Proceedings of the IEEE **75**(10), 1434–1435 (1987)
8. Cooley, J.W., Tukey, J.W.: An algorithm for the machine calculation of complex Fourier series. Mathematics of Computation **19**(90), 297–301 (1965)
9. Ferguson, R.J., Margrave, G.F.: Prestack depth migration by symmetric nonstationary phase shift. Geophysics **67**(2), 594–603 (2002)
10. Frigo, M., Johnson, S.G.: `FFTW`. `http://www.fftw.org`
11. Frigo, M., Johnson, S.G.: The design and implementation of `FFTW3`. Proceedings of the IEEE **93**(2), 216–231 (2005)

12. Gauss, C.F.: Nachlass: Theoria interpolationis methodo nova tractata. In: Carl Friedrich Gauss Werke, vol. 3, pp. 265–327. Königliche Gesellschaft der Wissenschaften, Göttingen (1866)
13. Goldstine, H.H.: A History of Numerical Analysis from the 16th through the 19th Century, vol. 2. Springer Science & Business Media (2012)
14. Hairer, E., Lubich, C., Schlichte, M.: Fast numerical solution of nonlinear volterra convolution equations. SIAM journal on scientific and statistical computing **6**(3), 532–541 (1985)
15. Hassanieh, H., Indyk, P., Katabi, D., Price, E.: Simple and practical algorithm for sparse Fourier transform. In: Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1183–1194. SIAM (2012)
16. Heideman, M.T., Johnson, D.H., Burrus, C.S.: Gauss and the history of the fast Fourier transform. ASSP Magazine, IEEE **1**(4), 14–21 (1984)
17. Heideman, M.T., Johnson, D.H., Burrus, C.S.: Gauss and the history of the fast Fourier transform. Archive for history of exact sciences **34**(3), 265–277 (1985)
18. Lubich, C., Schädle, A.: Fast convolution for nonreflecting boundary conditions. SIAM Journal on Scientific Computing **24**(1), 161–182 (2002)
19. Markel, J.: FFT pruning. IEEE Transactions on Audio and Electroacoustics **19**(4), 305–311 (1971)
20. Michielssen, E., Boag, A.: A multilevel matrix decomposition algorithm for analyzing scattering from large structures. Antennas and Propagation, IEEE Transactions on **44**(8), 1086–1093 (1996)
21. Nussbaumer, H.J.: Fast Fourier transform and convolution algorithms, vol. 2. Springer Science & Business Media (2012)
22. O'Neil, M., Rokhlin, V.: A new class of analysis-based fast transforms. Tech. rep., DTIC Document (2007)
23. O'Neil, M., Woolfe, F., Rokhlin, V.: An algorithm for the rapid evaluation of special function transforms. Applied and Computational Harmonic Analysis **28**(2), 203–226 (2010)
24. Rabiner, L.R., Schafer, R.W., Rader, C.M.: The chirp z-transform algorithm. Audio and Electroacoustics, IEEE Transactions on **17**(2), 86–92 (1969)
25. Roberts, M., Bowman, J.C.: Multithreaded implicitly dealiased convolutions. J. Comput. Phys. (2017). In press, https://doi.org/10.1016/j.jcp.2017.11.026
26. Schuster, G.T.: Seismic interferometry, vol. 1. Cambridge University Press Cambridge (2009)
27. Ying, L.: Sparse Fourier transform via butterfly algorithm. SIAM Journal on Scientific Computing **31**(3), 1678–1694 (2009)
28. Ying, L., Fomel, S.: Fast computation of partial Fourier transforms. Multiscale Modeling and Simulation **8**(1), 110–124 (2009)