SIAM J. Sci. Comput.
Vol. 39, No. 6, pp. A3089–A3106

A FAST ALGORITHM FOR THE CONVOLUTION OF FUNCTIONS WITH COMPACT SUPPORT USING FOURIER EXTENSIONS[*]

KUAN XU[†], ANTHONY P. AUSTIN[‡], AND KE WEI[§]

Abstract. We present a new algorithm for computing the convolution of two compactly supported functions. The algorithm approximates the functions to be convolved using Fourier extensions and then uses the fast Fourier transform to efficiently compute Fourier extension approximations to the pieces of the result. The complexity of the algorithm is $\mathcal{O}\big(N(\log N)^2\big)$, where N is the number of degrees of freedom used in each of the Fourier extensions.

Key words. convolution, fast Fourier transform, Fourier extension, Fredholm convolution integral, Toeplitz matrix, Volterra convolution integral

AMS subject classifications. 44A35, 65R10, 65T40

DOI. 10.1137/17M1114764

1. Introduction.

Let $f : [a,b] \to \mathbb{C}$ and $g : [c,d] \to \mathbb{C}$ be two compactly supported functions on the real line. We consider the problem of approximating the convolution $h = f * g$, defined via the integral

$$h(x) = \int_{-\infty}^{\infty} f(t)g(x-t)\,dt = \int_{\max(a,x-d)}^{\min(b,x-c)} f(t)g(x-t)\,dt, \quad x \in [a+c, b+d]. \tag{1.1}$$

Since convolution is a commutative operation, we lose no generality in assuming that g has the larger domain: $d - c \geq b - a$. In this case, the limits of integration in (1.1) can be visualized for each value of x by the diagram in Figure 1. Splitting h into the three pieces suggested by the diagram, we have

$$h(x) = \begin{cases} h^L(x) = \displaystyle\int_a^{x-c} f(t)g(x-t)dt, & x \in [a+c, b+c], \quad (1.2a) \\ h^M(x) = \displaystyle\int_a^{b} f(t)g(x-t)dt, & x \in [b+c, a+d], \quad (1.2b) \\ h^R(x) = \displaystyle\int_{x-d}^{b} f(t)g(x-t)dt, & x \in [a+d, b+d]. \quad (1.2c) \end{cases}$$

[*]Submitted to the journal's Methods and Algorithms for Scientific Computing section February 3, 2017; accepted for publication (in revised form) September 7, 2017; published electronically December 21, 2017.

http://www.siam.org/journals/sisc/39-6/M111476.html

Funding: The first author's work was supported by the Royal Society under Research grant RG160236. The second author's work was supported by the U.S. Department of Energy, Office of Science, under contract number DE-AC02-06CH11357. The third author's work was supported by the National Science Foundation under grant DTRA-DMS 1322393. The submitted manuscript was created by UChicago Argonne, LLC, Operator of Argonne National Laboratory ("Argonne"). Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract DE-AC02-06CH11357. The U.S. Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

[†]School of Mathematics, Statistics, and Actuarial Science, University of Kent, Canterbury, CT2 7NF, UK (k.xu@kent.ac.uk).

[‡]Mathematics and Computer Science Division, Argonne National Laboratory, Lemont, IL 60439 (austina@anl.gov).

[§]Department of Mathematics, University of California, Davis, Davis, CA 95616 (kewei@math.ucdavis.edu).

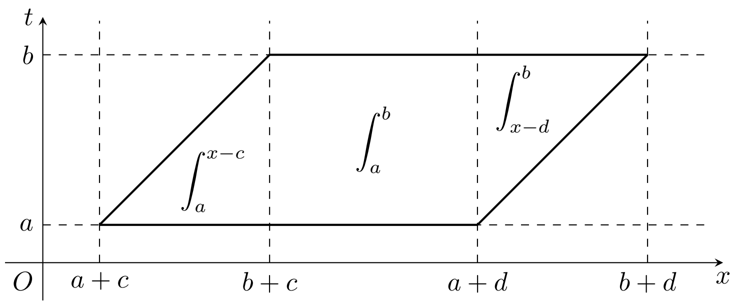

FIG. 1. *Schematic illustrating how the limits of integration in* (1.1) *vary with* x*, assuming that* $d-c \geq b-a$*. Given a value of* x *on the horizontal axis, the limits range over the values of* t *indicated by the corresponding "slice" of the parallelogram.*

We refer to h^L, h^M, and h^R as the left, middle, and right pieces of the convolution, respectively. The integrals for the left and right pieces are known as *Volterra* convolution integrals because each has a limit that varies with x, while the integral for the middle piece, which has constant limits, is a *Fredholm* convolution integral [6, 9, 10].

In [14], Hale and Townsend propose an algorithm for approximating h that first approximates f and g by polynomials represented in the Legendre basis and then convolves the approximations using a version of the convolution theorem for Legendre polynomials. The result is a piecewise polynomial representation that can be evaluated at any x in the domain of h to yield an approximation to $h(x)$. If the polynomials used to approximate f and g have degree at most M, their algorithm produces an approximation to h in $\mathcal{O}(M^2)$ operations.

In this article, we present a new algorithm for approximating h using Fourier extensions instead of polynomials. The algorithm first constructs Fourier extension approximations to f and g from samples at equispaced points and then uses these to compute Fourier extension approximations to h^L, h^M, and h^R. The advantage to using Fourier representations is that the latter computations can be carried out in just $\mathcal{O}(N \log N)$ operations using the fast Fourier transform (FFT), where N is the number of degrees of freedom in each of the Fourier extensions for f and g. Since the extensions can be computed in just $\mathcal{O}(N(\log N)^2)$ operations by using recently developed techniques [17], the overall complexity of our algorithm is $\mathcal{O}(N(\log N)^2)$.

Because Fourier extensions and polynomials approximate functions at different rates, the values of M and N defined in the preceding paragraphs needed to attain the same level of accuracy in the corresponding approximations to f and g may differ. It is therefore not immediately clear from the preceding description how our method compares with the one from [14]. We explore these issues in some of our numerical experiments below.

Note that while our algorithm uses Fourier representations, it differs substantially from the naive method of computing h by approximating (suitably zero-extended versions of) f and g using trigonometric polynomials over a common interval and convolving the approximations via the FFT. The discontinuities introduced by the nonperiodicity of f and g makes the approximations used in this approach inefficient and susceptible to artifacts such as the Gibbs phenomenon. One can ameliorate these problems by smoothing out the discontinuities; however, this comes at the expense of accuracy in the computation of h. Fourier extensions are more complicated theoretically and algorithmically, but they offer a way around these deficiencies.

Our discussion is organized as follows. In section 2, we briefly review some concepts related to Fourier extensions. In section 3, we present the details of our algorithm. In section 4, we test our algorithm on a variety of numerical examples, and in section 5, we conclude the paper with some additional remarks about future work.

2. Fourier extensions.

Fourier extensions are a technique for approximating functions on compact subintervals of the real line by using Fourier series, even if they are not periodic. Briefly, the idea is to smoothly extend the function to a periodic function on a larger domain and then approximate the extension using Fourier series. In practice, we do not compute such an extension explicitly but, instead, directly approximate the original function using the Fourier basis from the larger domain.

In symbols, suppose that F is a complex-valued function on $[-1,1]$. For fixed $T > 1$, we seek an approximation to F of the form

$$F(x) \approx G_n(x), \qquad G_n(x) = \sum_{k=-n}^{n} c_k e^{ik\pi x/T}, \qquad x \in [-1,1]. \tag{2.1}$$

The function G_n is a trigonometric polynomial of degree n that is periodic on the interval $[-T,T]$. The value T is known as the *extension parameter* [3] (or the *continuation length* [11]) and is commonly taken to be 2, although our algorithm will involve Fourier extension approximations of lengths other than this.[1] Note that the approximation is not expected to be good for $x \notin [-1,1]$, and in practice it rarely is.[2]

The approximation in (2.1) is usually made with respect to a least-squares criterion. One can choose G_n to minimize $\|F - G_n\|_{L^2([-1,1])}$, where $\|\cdot\|_{L^2([-1,1])}$ is the L^2 norm on $[-1,1]$; this is called the *continuous Fourier extension* problem. Alternatively, one can solve a *discrete Fourier extension* problem[3] obtained by selecting $M = 2m+1 > N = 2n+1$ points $x_{-m}, \ldots, x_m \in [-1,1]$ and demanding that G_n minimize $\sum_{j=-m}^{m} |F(x_j) - G_n(x_j)|^2$. The number of points M is typically chosen as some fixed multiple of N, the number of degrees of freedom available in the extension. That is, we have $M = \gamma N$ for some $\gamma \geq 1$, known as the *oversampling factor*. For a detailed explanation of why oversampling is important, we refer the reader to [7, section 4.2]. Empirical studies find that taking $\gamma = 2$ yields adequate results in practice [3, 5, 11]. The most commonly used points are the equispaced points $x_j = j/m$, since these enable one to employ numerical methods based on the FFT. Since the FFT is integral to the success of our algorithm, we work exclusively with discrete Fourier extensions based on equispaced points in this article.

It turns out that both the continuous and discrete extension problems are poorly conditioned in that the coefficients c_k in (2.1) are highly sensitive to perturbations in F (measured in the appropriate 2-norm). Nevertheless, when solving these problems numerically, while the computed c_k may differ greatly from those of the exact solution, the function G_n that they determine can still be an excellent approximation to F provided that n is large enough that F can be resolved. These phenomena are studied carefully in [2, 3, 15], to which we refer the reader for further details. We will observe

[1]The widely adopted rule of thumb that $T = 2$ is a good choice for general use originates with [11], and the utility of this choice has been further confirmed by more recent studies [3, 5].

[2]Indeed, F may not even be defined for $x \notin [-1,1]$. Moreover, G_n typically does not converge to anything outside $[-1,1]$ as n increases [15].

[3]The discrete problem is much more frequently discussed in the literature. Some references that treat both the continuous and the discrete problems include [1, 3, 15].

how these effects impact the stability of our proposed algorithm later in our numerical experiments.

Several methods are available for solving the discrete Fourier extension problem in equispaced points. The one we employ is the recently developed algorithm due to Matthysen and Huybrechs [17], which exploits the rapid decay of the singular values of the matrix associated with the least-squares problem using randomized techniques to compute a solution in just $\mathcal{O}\big(N(\log N)^2\big)$ operations.[4,5] At the time of this writing, an implementation of their algorithm in the Julia language is publicly available on GitHub.[6] We have used this implementation to carry out all of the computations that we present here.

For further general information about Fourier extensions, we direct the reader to [3, 7, 11, 15, 17] and the references therein.

3. Fast convolution algorithm. The idea behind our proposed algorithm is simple. To approximate a piece (left, middle, or right) of $h = f * g$, we first approximate f and g using Fourier extensions. Then, we convolve these approximations to obtain a Fourier extension approximation to the result. Since the approximations are expressed in the Fourier basis, their convolutions can be computed efficiently by using the FFT. If we use extensions with N degrees of freedom to approximate f and g, then the first phase of the algorithm can be performed in $\mathcal{O}\big(N(\log N)^2\big)$ operations by using the techniques mentioned at the end of the preceding section. The second phase takes $\mathcal{O}(N \log N)$ operations, so the overall complexity is $\mathcal{O}\big(N(\log N)^2\big)$.

We now present the details of our algorithm in full.

3.1. Changing variables. Although Fourier extensions can be computed for functions defined on arbitrary intervals, it is convenient to describe our algorithm in terms of functions defined on basic intervals centered at the origin. To this end, we define the following mapped versions of f and g:

$$\begin{aligned} \widetilde{f}(y) &= f\left(\frac{b-a}{2}y + \frac{b+a}{2}\right), \qquad && y \in [-1,1], \\ \widetilde{g}(y) &= g\left(\frac{b-a}{2}y + \frac{d+c}{2}\right), \qquad && y \in [-\rho,\rho], \end{aligned} \tag{3.1}$$

where $\rho = (d-c)/(b-a) \geq 1$. (Recall our assumption that $d - c \geq b - a$.)

One can easily relate the convolution of f and g to that of $\widetilde{f}$ and $\widetilde{g}$. Define

$$\left.\begin{aligned} \widetilde{h}^L(y) &= \int_{-1}^{y} \widetilde{f}(t)\widetilde{g}(y-\rho-t)\,dt, \\ \widetilde{h}^R(y) &= \int_{y}^{1} \widetilde{f}(t)\widetilde{g}(y+\rho-t)\,dt, \end{aligned}\right. \qquad y \in [-1,1]. \tag{3.2}$$

Then, $\widetilde{h}^L$ and $\widetilde{h}^R$ are the left and right pieces of $\widetilde{f} * \widetilde{g}$, respectively, translated so that

[4]An earlier algorithm of a similar flavor due to Lyon [16] attains the same complexity; however, it can be used to compute extensions only with $T = 2$.

[5]The implied constant in the big-$\mathcal{O}$ symbol will depend on T and γ. We will not address this issue here and will stick to the standard choices $T = 2$ and $\gamma = 2$ except where deviation is required; see section 3.2. Further information and experiments may be found in [17].

[6]https://github.com/daanhb/FrameFun.jl

their domains are both $[-1,1]$, and

$$\begin{aligned} h^L(x) &= \frac{b-a}{2}\widetilde{h}^L\left(2\frac{x-c-b}{b-a}+1\right), \qquad x \in [a+c, b+c], \\ h^R(x) &= \frac{b-a}{2}\widetilde{h}^R\left(2\frac{x-d-a}{b-a}-1\right), \qquad x \in [a+d, b+d], \end{aligned} \tag{3.3}$$

as can be seen via straightforward changes of the integration variables. For the middle piece, we set

$$\widetilde{h}^M(y) = \int_{-1}^{1} \widetilde{f}(t)\widetilde{g}(y-t)\,dt, \qquad y \in [-(\rho-1), \rho-1], \tag{3.4}$$

and we have

$$h^M(x) = \frac{b-a}{2}\widetilde{h}^M\left(\frac{2x-(a+b+c+d)}{b-a}\right), \qquad x \in [b+c, a+d]. \tag{3.5}$$

Note that the change-of-variable maps used to define $\widetilde{f}$ and $\widetilde{g}$ in (3.1) have identical derivatives. This feature is important because it ensures that the independent variables of $\widetilde{f}$ and $\widetilde{g}$ are scaled relative to one another in the same way as those of the original functions f and g. Without this, the mismatch in scaling would mean that no simple relationship akin to (3.3) and (3.5) would exist between $f * g$ and $\widetilde{f} * \widetilde{g}$.

3.2. Approximation using Fourier extensions. The first step in our algorithm is to approximate $\widetilde{f}$ and $\widetilde{g}$ on $[-1,1]$ and $[-\rho,\rho]$, respectively, using Fourier extensions. We must select a parameter T to use for the extensions; as mentioned in section 2, $T = 2$ is the standard choice. An extension for $\widetilde{f}$ with parameter T will use the Fourier basis on $[-T,T]$, while one for $\widetilde{g}$ will use the Fourier basis on $[-\rho T, \rho T]$.

For reasons that will become clear, however, we need the period on which the extension for $\widetilde{g}$ is based to be an integer multiple of that of the extension for $\widetilde{f}$. Therefore, instead of approximating $\widetilde{g}$ with an extension on $[-\rho T, \rho T]$, we use an extension on $[-\kappa T, \kappa T]$, where κ is the smallest integer such that $\kappa \geq \rho$. This choice ensures that the extension parameter for the approximation to $\widetilde{g}$ is as close to T as possible while still allowing the period of the extension for $\widetilde{g}$ to satisfy the required condition.

Thus, if we use Fourier extensions of lengths $M = 2m+1$ and $N = 2n+1$ to approximate $\widetilde{f}$ and $\widetilde{g}$, respectively, our approximations take the form

$$\widetilde{f}(y) \approx \sum_{j=-m}^{m} a_j e^{ij\pi y/T}, \qquad \widetilde{g}(y) \approx \sum_{k=-n}^{n} b_k e^{ik\pi y/(\kappa T)}. \tag{3.6}$$

3.3. Computing the left and right pieces. We now describe how to compute $\widetilde{h}^L$ and $\widetilde{h}^R$, from which h^L and h^R can be recovered using (3.3). We will focus on the computation of $\widetilde{h}^L$; the procedure for computing $\widetilde{h}^R$ is similar.[7] Substituting the approximations (3.6) into (3.2) and using the fact that

$$\int_{-1}^{y} e^{i(\kappa j-k)\pi t/(\kappa T)}\,dt = \begin{cases} \frac{\kappa T}{i\pi}\left(\frac{e^{i\pi(\kappa j-k)y/(\kappa T)}}{\kappa j-k} + \frac{e^{i\pi(k-\kappa j)/(\kappa T)}}{k-\kappa j}\right), & k \neq \kappa j, \\ y+1, & k = \kappa j, \end{cases}$$

[7]In more detail, $\widetilde{h}^R(y)$, $y \in [-1,1]$, is just the evaluation at $-y$ of the left piece of $\widetilde{f}^* * \widetilde{g}^*$, where $\widetilde{f}^*(y) = \widetilde{f}(-y)$ and $\widetilde{g}^*(y) = \widetilde{g}(-y)$.

we obtain

$$\widetilde{h}^L(y) \approx \sum_{j=-m}^{m} a_j \sum_{k=-n}^{n} b_k e^{ik\pi(y-\rho)/(\kappa T)} \int_{-1}^{y} e^{i(\kappa j - k)\pi t/(\kappa T)}\, dt$$
$$(3.7) \qquad = \sum_{j=-m}^{m} \widetilde{a}_j e^{ij\pi y/T} + \sum_{k=-n}^{n} \widetilde{b}_k e^{ik\pi y/(\kappa T)} + (y+1) \sum_{\substack{j=-m \\ -n \le \kappa j \le n}}^{m} c_j e^{ij\pi y/(\kappa T)},$$

where

$$(3.8) \qquad \widetilde{a}_j = \frac{\kappa T}{i\pi} a_j p_j, \qquad p_j = \sum_{\substack{k=-n \\ k \ne \kappa j}}^{n} b_k e^{-ik\pi\rho/(\kappa T)} \frac{1}{\kappa j - k}, \qquad -m \le j \le m,$$

and

$$(3.9) \quad \widetilde{b}_k = \frac{\kappa T}{i\pi} b_k q_k e^{ik\pi(1-\rho)/(\kappa T)}, \qquad q_k = \sum_{\substack{j=-m \\ \kappa j \ne k}}^{m} a_j e^{-ij\pi/T} \frac{1}{k - \kappa j}, \qquad -n \le k \le n,$$

and

$$(3.10) \qquad c_j = a_j b_{\kappa j} e^{-ij\pi\rho/T}, \qquad -m \le j \le m, -n \le \kappa j \le n.$$

The key observation to make is that the sums defining p_j and q_k in (3.8) and (3.9) can be interpreted as matrix-vector products in which the matrices involved have a Toeplitz structure. Let $b = (b_{-n}e^{-i(-n)\pi\rho/(\kappa T)}, \ldots, b_n e^{-in\pi\rho/(\kappa T)})$, a (column) vector of length N, and let P be the $(2\kappa m + 1) \times N$ matrix whose (ν, k) entry is

$$P_{\nu,k} = \begin{cases} \frac{1}{\nu - k}, & \nu \ne k, \\ 0, & \nu = k, \end{cases}$$

for $-\kappa m \le \nu \le \kappa m$ and $-n \le k \le n$. Let $\hat{p} = Pb$. If $\hat{p}_\nu$ denotes the νth component of $\hat{p}$, $-\kappa m \le \nu \le \kappa m$, then from (3.8) we have $p_j = \hat{p}_{\kappa j}$. Thus, we can compute the p_j by computing $\hat{p}$ and keeping every κth component. Similarly, if $\hat{a}$ is the vector of length $2\kappa m + 1$ defined by

$$\hat{a}_\nu = \begin{cases} a_{\nu/\kappa} e^{-i(\nu/\kappa)\pi/T}, & \nu \equiv 0 \bmod \kappa, \\ 0, & \text{otherwise}, \end{cases}$$

for $-\kappa m \le \nu \le \kappa m$ and $q = (q_{-n}, \ldots q_n)$, then by (3.9) we have $q = Q\hat{a}$, where Q is the $N \times (2\kappa m + 1)$ matrix whose (k, ν) entry is

$$Q_{k,\nu} = \begin{cases} \frac{1}{k - \nu}, & k \ne \nu, \\ 0, & k = \nu, \end{cases}$$

for $-n \le k \le n$ and $-\kappa m \le \nu \le \kappa m$.

Since $P_{\nu,k}$ and $Q_{k,\nu}$ depend only on the differences $\nu - k$ and $k - \nu$, respectively, and not on ν and k individually, P and Q are Toeplitz. Therefore, the multiplications needed to compute $\hat{p}$ and q (and, hence, *all* the p_j and q_k) can be carried out in $\mathcal{O}\big((2\kappa m + N)\log(2\kappa m + N)\big)$ operations by the standard technique of embedding them in circulant matrices and using the FFT [21, section 4.8.4]. Since these multiplications constitute the dominant cost in computing the coefficients $\widetilde{a}_j$, $\widetilde{b}_k$, and c_j, the approximation (3.7) to $\widetilde{h}^L$ can be formed in $\mathcal{O}\big((2\kappa m+N)\log(2\kappa m+N)\big)$ operations.

Noting that the first sum may be rewritten to use complex exponentials of period κT by inserting factors of κ into the numerator and denominator of the exponent, we see that the expansion (3.7) is nearly a Fourier extension approximation to $\widetilde{h}^L$ with extension parameter κT. The only problem is the appearance of the factor of $y+1$ multiplying the third sum. We can eliminate this factor by approximating it on $[-1,1]$ with a Fourier extension

$$y+1 \approx \sum_{r=-d}^{d} \gamma_r e^{ir\pi y/T}, \tag{3.11}$$

where the degree d is chosen large enough to attain a desired level of accuracy. Rewriting this sum using complex exponentials of period κT, we have

$$y+1 \approx \sum_{r=-\kappa d}^{\kappa d} \widetilde{\gamma}_r e^{ir\pi y/(\kappa T)},$$

where $\widetilde{\gamma}_r = \gamma_{r/\kappa}$ for $r \equiv 0 \bmod \kappa$ and $\widetilde{\gamma}_r = 0$ otherwise. Upon multiplying this series with the third sum in (3.7), we obtain an approximation

$$(y+1)\sum_{\substack{j=-m\\ -n\le\kappa j\le n}}^{m} c_j e^{ij\pi y/(\kappa T)} \approx \sum_{\ell=-d'}^{d'} \widetilde{c}_\ell e^{i\ell\pi y/(\kappa T)}, \tag{3.12}$$

where $d' = \kappa d + \min(m, \lceil n/\kappa\rceil)$ and $\lceil x\rceil$ denotes the least integer greater than or equal to x. The coefficients $\widetilde{c}_\ell$ are the finite Cauchy product (discrete convolution) of $\{c_j\}_{j=-m}^{m}$ and $\{\widetilde{\gamma}_r\}_{r=-\kappa d}^{\kappa d}$ and can be computed efficiently in $\mathcal{O}\big(d'\log(d')\big)$ operations using the standard FFT-based algorithm for multiplying trigonometric polynomials.

For the usual choice of $T=2$, we find experimentally that an extension with $d=24$ (49 basis functions) is sufficient to approximate $y+1$ to machine precision uniformly on $[-1,1]$ in double-precision arithmetic. Note that this approximation can be computed once, offline, and then reused as many times as necessary.

3.4. Computing the middle piece. The middle piece is handled similarly to the left and right pieces. In fact, it is slightly simpler because the limits of integration in (3.4) are constant, whereas those in (3.2) vary with y. Inserting the approximations (3.6) into (3.4) and noting that

$$\int_{-1}^{1} e^{i(\kappa j-k)\pi t/(\kappa T)}\,dt = \begin{cases} \frac{2\kappa T}{\pi}\frac{\sin\left(\frac{\pi}{\kappa T}(\kappa j-k)\right)}{\kappa j-k}, & k\ne\kappa j,\\ 2, & k=\kappa j,\end{cases}$$

we find

$$\begin{aligned}\widetilde{h}^M(y) &\approx \sum_{j=-m}^{m} a_j \sum_{k=-n}^{n} b_k e^{ik\pi y/(\kappa T)} \int_{-1}^{1} e^{i(\kappa j-k)\pi t/(\kappa T)}\,dt\\ &= \sum_{k=-n}^{n} \widetilde{b}_k e^{ik\pi y/(\kappa T)} + \sum_{\substack{j=-m\\ -n\le\kappa j\le n}}^{m} \widetilde{c}_j e^{i\kappa j\pi y/(\kappa T)},\end{aligned} \tag{3.13}$$

where

$$\widetilde{b}_k = \frac{2\kappa T}{\pi} b_k r_k, \qquad r_k = \sum_{\substack{j=-m \\ \kappa j \neq k}}^{m} a_j \frac{\sin\left(\frac{\pi}{\kappa T}(\kappa j - k)\right)}{\kappa j - k}, \qquad -n \le k \le n, \tag{3.14}$$

and

$$\widetilde{c}_j = 2 a_j b_{\kappa j}, \qquad -m \le j \le m, -n \le \kappa j \le n. \tag{3.15}$$

The expansion (3.13) is a Fourier extension approximation to $\widetilde{h}^M$ with period κT. The dominant cost in computing the coefficients $\widetilde{b}_k$ and $\widetilde{c}_j$ in this expansion comes from computing the values r_k defined in (3.14); however, as was the case for p_j and q_k in (3.8) and (3.9), we can accelerate the computation of the r_k by taking advantage of the structure of the sum. Letting $r = (r_{-n}, \ldots, r_n)$, a vector of length $N = 2n+1$, we have $r = R\hat{a}$, where $\hat{a} = (\hat{a}_{-\kappa m}, \ldots, \hat{a}_{\kappa m})$ is the vector of length $2\kappa m + 1$ defined by

$$\hat{a}_\nu = \begin{cases} a_{\nu/\kappa}, & \nu \equiv 0 \bmod \kappa, \\ 0, & \text{otherwise}, \end{cases}$$

for $-\kappa m \le \nu \le \kappa m$, and R is the $N \times (2\kappa m + 1)$ Toeplitz matrix with entries

$$R_{k,\nu} = \begin{cases} \frac{\sin\left(\frac{\pi}{\kappa T}(\nu - k)\right)}{\nu - k}, & k \neq \nu, \\ 0, & k, = \nu, \end{cases}$$

for $-n \le k \le n$ and $-\kappa m \le \nu \le \kappa m$. As before, this matrix can be applied to $\hat{a}$ in $\mathcal{O}\big((2\kappa m + N)\log(2\kappa m + N)\big)$ operations by using the FFT.

3.5. Summary of the main algorithm. We summarize our algorithm for computing a Fourier extension approximation to (1.1) as follows:

1. Choose an extension parameter T on which to base the approximations; taking $T = 2$ is standard and should suffice in general.
2. Compute Fourier extension approximations to $\widetilde{f}$ and $\widetilde{g}$ as prescribed by (3.6) using one of the recently developed fast algorithms mentioned in section 2.
3. Compute Fourier extension approximations to $\widetilde{h}^L$, $\widetilde{h}^M$, and $\widetilde{h}^R$ by appropriately convolving the Fourier extensions computed in Step 2. This can be done efficiently by using the FFT to compute the coefficients of the approximations as governed by (3.7) through (3.12) for the left piece (and similarly for the right piece) and (3.13) through (3.15) for the middle piece.
4. Undo the changes of variables according to (3.3) and (3.5) to obtain Fourier extension approximations to h^L, h^M, and h^R.

3.6. An alternative approach to the left and right pieces. In some applications, only the left and/or right piece of the convolution integral is of interest. When this is the case, one may be able to save some operations by approximating only the portion of g that enters into the convolution instead of all of g. We now describe how to do this. As in section 3.3, we focus on the computation of the left piece; the right piece can be handled analogously.

Let

$$\widetilde{g}^L(y) = g\left(\frac{b-a}{2}y + \frac{b-a}{2} + c\right), \qquad y \in [-1, 1].$$

The function $\widetilde{g}^L$ maps the portion of g defined on $[c, c + (b - a)]$ to $[-1, 1]$; this is precisely the portion of g that is involved in the left piece of $f * g$. One can easily see that

$$\widetilde{h}^L(y) = \int_{-1}^{y} \widetilde{f}(t)\widetilde{g}^L(y - 1 - t)\,dt, \qquad y \in [-1, 1], \tag{3.16}$$

where $\widetilde{h}^L$ is as defined in (3.2). To approximate $\widetilde{h}^L$ using (3.16), we approximate $\widetilde{f}$ and $\widetilde{g}^L$ using Fourier extensions:

$$\widetilde{f}(y) \approx \sum_{j=-m}^{m} a_j e^{ij\pi y/T}, \qquad \widetilde{g}^L(y) \approx \sum_{k=-n}^{n} b_k e^{ik\pi y/T}. \tag{3.17}$$

The difference between (3.6) and (3.17) is that since the domains of $\widetilde{f}$ and $\widetilde{g}^L$ are the same, they can be approximated by extensions with the same parameter using complex exponentials with the same period as if κ were equal to 1. Doing this will, in general, reduce the length of the FFTs required to compute the coefficients of the Fourier extension approximation to the convolution. Moreover, since $\widetilde{g}^L$ represents only a part of g, one probably will be able to get away with using a shorter series (smaller value of n) to obtain an approximation of a given accuracy than one would need in order to approximate all of g via $\widetilde{g}$.

From here, we proceed exactly as in section 3.3, replacing $\widetilde{g}$ with $\widetilde{g}^L$ and taking $\kappa = 1$ in all formulas. For completeness, we give the simplified formulas in full here. By inserting (3.17) into (3.16), we obtain

$$\begin{aligned}\widetilde{h}^L(y) &\approx \sum_{j=-m}^{m} a_j \sum_{k=-n}^{n} b_k e^{ik\pi(y-1)/T} \int_{-1}^{y} e^{i\pi(j-k)t/T}\,dt \\ &= \sum_{j=-m}^{m} \widetilde{a}_j e^{ij\pi y/T} + \sum_{k=-n}^{n} \widetilde{b}_k e^{ik\pi y/T} + (y+1) \sum_{\ell=-\min(m,n)}^{\min(m,n)} c_\ell e^{i\ell\pi y/T}, \end{aligned} \tag{3.18}$$

where

$$\widetilde{a}_j = \frac{T}{i\pi} a_j p_j e^{-ij\pi/T}, \qquad p_j = \sum_{\substack{k=-n \\ k\neq j}}^{n} b_k \frac{e^{i\pi(j-k)/T}}{j-k}, \qquad -m \le j \le m, \tag{3.19}$$

and

$$\widetilde{b}_k = \frac{T}{i\pi} b_k q_k e^{-ik\pi/T}, \qquad q_k = \sum_{\substack{j=-m \\ j\neq k}}^{m} a_j \frac{e^{i\pi(k-j)/T}}{k-j}, \qquad -n \le k \le n, \tag{3.20}$$

and

$$c_\ell = a_\ell b_\ell e^{-i\ell\pi/T}, \qquad -\min(m,n) \le \ell \le \min(m,n). \tag{3.21}$$

From (3.19), we have $p = Pb$, where $p = (p_{-m}, \ldots, p_m)$ and $b = (b_{-n}, \ldots, b_n)$ are vectors of length $M = 2m + 1$ and $N = 2n + 1$, respectively, and P is the $M \times N$ Toeplitz matrix whose (j, k) entry is

$$P_{j,k} = \begin{cases} \frac{e^{i\pi(j-k)/T}}{j-k}, & j \neq k, \\ 0, & j = k, \end{cases}$$

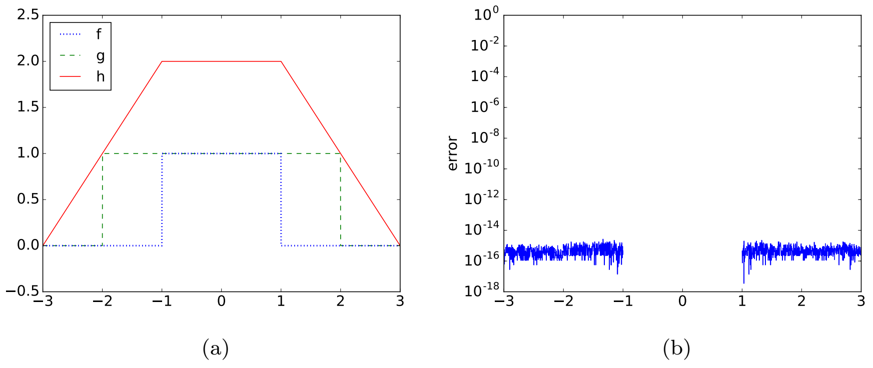

FIG. 2. *Convolution of two boxes.* (a) *The functions f and g being convolved and the approximation to their convolution h.* (b) *The error in the approximation. Because of the simplicity of this example, the error comes almost entirely from the precomputed approximation to $y+1$ in* (3.12). *The error curve for the middle piece is missing because it has been computed exactly: constant functions are part of the extended Fourier basis.*

for $-m \leq j \leq m$ and $-n \leq k \leq n$. Similarly, from (3.20), we have $q = Qa$, where $q = (q_{-n}, \ldots, q_n)$, $a = (a_{-m}, \ldots, a_m)$, and Q is the $N \times M$ Toeplitz matrix defined by

$$Q_{k,j} = \begin{cases} \frac{e^{i\pi(k-j)/T}}{k-j}, & j \neq k, \\ 0, & j = k, \end{cases}$$

for $-m \leq j \leq m$ and $-n \leq k \leq n$.

This takes care of the first and second sums in (3.18). The multiplication of the third sum by $y+1$ can be handled in exactly the same manner as described in section 3.3.

4. Numerical examples.

We now examine the characteristics of the proposed algorithm and evaluate its performance on a few numerical examples. The Fourier extensions were computed using the publicly available implementation of the Matthysen–Huybrechs algorithm mentioned in section 2. In all examples, we use an extension parameter of $T = 2$ and an oversampling factor of $\gamma = 2$ unless otherwise specified.

4.1. Convolution of two boxes. As an easy first test, we set f and g to be the characteristic functions of the intervals $[-1, 1]$ and $[-2, 2]$, respectively. Figure 2(a) depicts f, g, and the computed approximation to their (full) convolution. Figure 2(b) displays the pointwise absolute error in the computation.

Since f and g can each be represented exactly by using just a single (zero-frequency) Fourier coefficient ($N = 1$), this example is perhaps the simplest possible problem to which our algorithm may be applied. The error in the result is also due almost entirely to the error in the precomputed approximation to $y+1$ in (3.12) needed for the left and right pieces. The approximation to the middle piece h^M produced by our algorithm has only $N = 1$ Fourier coefficient. The approximations to h^L and h^R have $N = 97$ coefficients each.[8]

[8]For this problem, $m = n = 0$ and $\kappa = 2$. Taking $d = 24$ in (3.11) as described at the end of section 3.3, we have $d' = 48$ in (3.12), so the right-hand side of (3.12) has $2d' + 1 = 97$ coefficients in total.

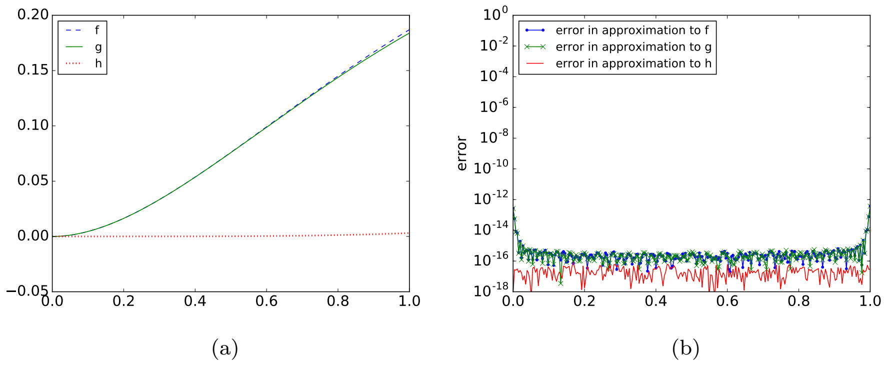

FIG. 3. *A convolution related to renewal theory.* (a) *The functions f and g being convolved and the approximation to the left piece of their convolution.* (b) *Absolute error in the approximations to f, g, and the left piece of their convolution.*

4.2. A convolution from renewal theory. For our next example, we consider the Volterra convolution integral equation

$$f(x) = g(x) + \int_0^x f(t)g(x-t)\,dt, \qquad x \geq 0, \tag{4.1}$$

with

$$g(x) = \frac{1}{2}x^2 e^{-x}.$$

This equation arises in renewal theory [10, Example 1.4.3], [13], and one can show using Laplace transforms that the solution is

$$f(x) = \frac{1}{3} - \frac{1}{3}\left(\cos\frac{\sqrt{3}}{2}x + \sqrt{3}\sin\frac{\sqrt{3}}{2}x\right)e^{-3x/2}.$$

We verify this solution by using our algorithm to approximate the integral that appears in (4.1) and comparing the result with $f - g$.

We approximate f and g (which are similar but not identical) on $[0,1]$ using Fourier extensions with $N = 71$ degrees of freedom. Since only the left piece of $f * g$ is desired, we use the simplified version of our algorithm given in section 3.6 to perform the computation. Figure 3(a) depicts f, g, and the computed approximation to the left piece h^L of their convolution, which is also defined on $[0,1]$. The approximation to h^L uses $N = 119$ Fourier coefficients. The pointwise absolute error in the approximations to f, g, and h^L is displayed in Figure 3(b).

The functions f and g are not complicated and do not require many coefficients to represent accurately, making this still a relatively easy example. Nevertheless, the computation illustrates an interesting fact about approximating convolutions using Fourier extensions. We observe from Figure 3(b) that while the error in the Fourier extension approximations to f and g is as high as 10^{-13}, the error in the approximation to their convolution is uniform at a level just under 10^{-16}. That is, the accuracy in the approximation to h^L (measured in the uniform norm) is roughly three orders of magnitude greater than the accuracy in the approximations used to compute it. This phenomenon can be partially explained by taking into account the fact that h^L is roughly 10 times smaller than f and g (i.e., we should consider measuring the error in a relative rather than an absolute sense), but this reasoning is not enough to account for everything. The other key observation is that the regions of comparatively large error in the approximations to f and g are localized to the endpoints, so they do not contribute much to the convolution integral.

One can make this reasoning more precise as follows. Suppose that we have approximated f and g by $\widetilde{f} = f + \delta_f$ and $\widetilde{g} = g + \delta_g$, respectively, where δ_f and δ_g are functions that satisfy $\|\delta_f\|_\infty \le \varepsilon$ and $\|\delta_g\|_\infty \le \varepsilon$ for some $\varepsilon > 0$. Here, $\|\cdot\|_\infty$ denotes the uniform norm. The error in the approximation to $f * g$ is

$$(\widetilde{f} * \widetilde{g})(x) - (f * g)(x) = (\delta_f * g)(x) + (f * \delta_g)(x) + (\delta_f * \delta_g)(x).$$

Noting that the third term on the right-hand side is $\mathcal{O}(\varepsilon^2)$ as $\varepsilon \to 0$, upon taking absolute values and applying Hölder's inequality to the integrals that define the remaining two terms, we obtain

$$\|\widetilde{f} * \widetilde{g} - f * g\|_\infty \le \|g\|_\infty \|\delta_f\|_1 + \|f\|_\infty \|\delta_g\|_1 + \mathcal{O}(\varepsilon^2), \tag{4.2}$$

where $\|\cdot\|_1$ denotes the L^1-norm over the appropriate domain.

From (4.2), we see that the uniform-norm error in the convolution can be bounded by an expression in which the significant terms depend on $\|\delta_f\|_1$ and $\|\delta_g\|_1$ but not $\|\delta_f\|_\infty$ and $\|\delta_g\|_\infty$. That is, the *integrals* of the errors in the approximations to f and g determine the error in the approximation to the result. Hence, these errors must be large over a significant portion of the domain before their impacts become important.

In Figure 3(b), the areas of large error in f and g near the endpoints together occupy well less than 1/10 of the total interval, and they turn out to be relatively insignificant. This is useful to know, since error profiles similar to those in Figure 3(b) are not uncommon in practice.

The error analysis just performed is a bit cavalier in that it ignores the contribution to the error from the approximation (3.11) to $y + 1$. For the middle piece, the $y + 1$ factor does not enter into the picture at all, and no changes are needed. For the left piece, suppose that we approximate $y + 1$ by $y + 1 + \delta_{y+1}(y)$. Then, we don't compute $\widetilde{f} * \widetilde{g}$ exactly but instead compute an approximation $\widetilde{h} \approx \widetilde{f} * \widetilde{g}$, and according to (3.7) and (3.10), the pointwise absolute error in this approximation will be

$$\left|\widetilde{h}(y) - (\widetilde{f} * \widetilde{g})(y)\right| \le \left|\delta_{y+1}(y) \sum_{\substack{j=-m \\ -n \le \kappa j \le n}}^{m} c_j e^{ij\pi y/(\kappa T)}\right| \le |\delta_{y+1}(y)| \sum_{\substack{j=-m \\ -n \le \kappa j \le n}}^{m} |a_j b_{\kappa j}| \tag{4.3}$$

for all y in the domain of the left piece. The right piece may be analyzed similarly. We conclude that the error in the approximation to $y + 1$ does not make a significant contribution to the overall result, provided that the coefficients of the Fourier extension approximations to f and g are not too large.

4.3. Convolution of oscillatory functions. Next, we apply our algorithm to approximate the (full) convolution of the oscillatory functions $f(x) = \sin(100x) + x/50$ and $g(x) = \cos(200x)^2$, both defined on $[-1, 1]$. The exact answer can be found by using a symbolic computing system or a table to evaluate the necessary integrals. We approximate f and g using Fourier extensions with N degrees of freedom for several values of N ranging from 5 to 605 and measure both the time taken to compute the convolution (averaged over 3 trials) and the maximum absolute error in the result on a grid of 8192 equispaced points in $[-2, 2]$.

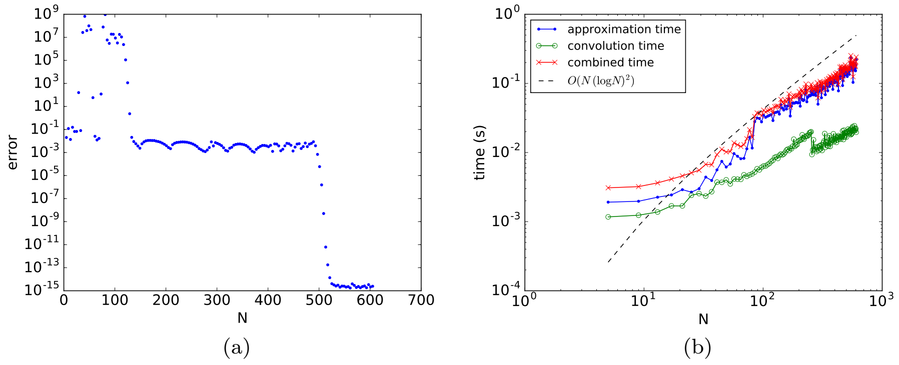

FIG. 4. *Results for the oscillatory convolution of section* 4.3. (a) *Maximum error in the convolution on a grid of* 8192 *equispaced points in* $[-2,2]$ *versus the number of degrees of freedom* N *in the Fourier extensions.* (b) *Time required to compute the convolution versus* N*. The blue and green lines show the time for approximating* f *and* g *and for the subsequent computations required to convolve the approximations, respectively. The red line shows the total time taken (the sum of the two).*

The results are displayed in Figure 4. Figure 4(a) shows that the approximation to the convolution has converged to the exact solution to nearly machine precision on $[-2,2]$ once N exceeds approximately 525. Figure 4(b) shows that the amount of time taken does indeed appear to grow at a rate bounded by $O\big(N(\log N)^2\big)$ as predicted. Moreover, the bulk of the time is clearly spent building the Fourier extension approximations to f and g. This is expected because the singular value decomposition computed by the Matthysen–Huybrechs algorithm to solve the extension problems is considerably more expensive than the handful of FFTs needed to convolve the extensions, even taking the randomized nature of the algorithm into account.

Figure 4(a) displays a trend that initially appears alarming. For several values of N, the error in the approximation is exceedingly high—on the order of 10^9. The behavior disappears around $N = 125$, at which point the error settles down to a nominal level of 10^{-3} before beginning its eventual decay to machine precision around $N = 500$. Looking more closely, we find that the instability arises as a result of the coefficients of the Fourier extension approximations to f and g being excessively large, with some as high as 10^{11} for the values of N for which the instability is most severe.

This behavior is actually a well-understood property of Fourier extensions; for extensive studies with quantitative results, we direct the reader to [2, 3]. The key point is that it does not persist indefinitely as N increases; it emerges only in a regime in which the function being approximated has not been adequately resolved.[9] Thus, while this effect means that our algorithm exhibits instability for some values of N, the instability is harmless, since those values of N all correspond to extensions with insufficiently many degrees of freedom to approximate the function accurately, meaning that a stably computed approximation would be of little utility anyway.

4.4. Comparison with methods based on polynomials. For our final experiment, we compare our algorithm to the one of Hale and Townsend [14], which approximates f and g by polynomials instead of Fourier extensions. Recall from the introduction that this method scales as $\mathcal{O}(M^2)$, where M is the degree of the polynomials employed. At first glance, it would appear that the method of this article,

[9]See [2], especially Theorems 5.3–5.4, and [3], especially section 5.3.

which scales as $\mathcal{O}\big(N(\log N)^2\big)$ (where, as usual, N denotes the number of degrees of freedom in each of the Fourier extensions used), should be superior.

The problem with this reasoning is that it is not clear how M relates to N, so the complexities cannot be directly compared. Fourier extensions and polynomials approximate functions at different rates. If the value of N needed for Fourier extension approximations to f and g to attain a given accuracy is much greater than the value of M needed for polynomial approximations to attain the same accuracy (e.g., if $N \propto M^2$), then the polynomial-based method may win out in the end.

Attempting to provide a definitive answer to the question of how M and N relate in general is beyond the scope of this article. Nevertheless, we can show numerically that there are situations in which our method may be competitive. For $x \in [-1, 1]$, let $f(x) = 1 + 1/(1 + \omega x^2)$, a Runge-like function, where $\omega \in \mathbb{R}$ is a given parameter. We consider the convolution $h = f * f$. As was the case with the previous example, the exact answer to this problem may be found by using a symbolic computing system or consulting a table of integrals.

For a range of values of ω, we determine M such that f is approximated uniformly on $[-1, 1]$ to within an accuracy of approximately 10^{-10} by its degree-M polynomial interpolant in a grid of (second-kind) Chebyshev points. We convolve this interpolant with itself using the algorithm from [14] and measure the time taken. We do not include the time taken to build the interpolant. We compare this with the time taken to approximate and convolve f with itself using our algorithm with a Fourier extension of length N that resolves f to approximately the same level of accuracy.[10]

As ω increases, the spike in f near the origin becomes increasingly narrow, and M and N will need to increase accordingly; thus, this experiment should provide insight into how the different methods scale as the problem complexity increases. All our computations involving polynomials were carried out in MATLAB R2016b using the Chebfun software package [12], which contains an implementation of the algorithm from [14]. Accuracy of both the Fourier extension and polynomial approximations was measured by computing the maximum absolute error on a fine grid in $[-1, 1]$.

The results are displayed in Figure 5. Figures 5(a) and 5(b) show how the quantities M and N described in the previous paragraph vary with ω. We observe from Figure 5(b) that $N \approx 1.4M$ for this problem, which suggests that as ω increases, the algorithm based on Fourier extensions should eventually exhibit superior performance. This is borne out by the plots of Figure 5(c). Because the algorithms have been implemented in different languages using different software packages, making a fair direct comparison between the times is difficult. Nevertheless, the figure clearly shows that the Fourier extension method becomes faster once ω is large enough. Figure 5(d) shows that all computations evaluate the convolutions to comparable levels of accuracy.

Actually, the performance of the Fourier extension method for this particular problem is even better than Figure 5(c) might suggest. The reason is that our implementation of the algorithm does not take advantage of the fact that f is being convolved with itself instead of with some other arbitrary function g. We therefore actually construct the Fourier extension approximation to f *twice* over the course of the computation. Since constructing the Fourier extension is the most expensive part of the algorithm (see Figure 4(b)), taking advantage of this would lead to significant

[10]While M can be determined using features of Chebfun, at the time of this writing, construction of an efficient general procedure for determining N is a matter of ongoing study. We determined N using a brute-force search. The reported times do not include the time taken to compute M and N.

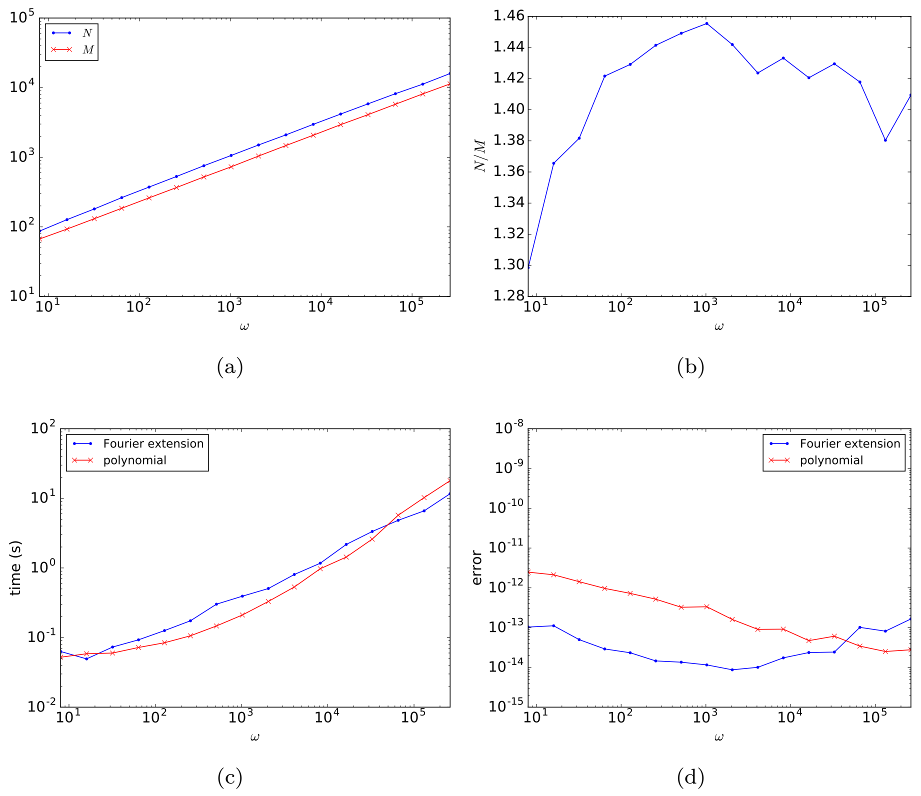

FIG. 5. *Results for the experiment of section* 4.4 *involving the convolution of* $f(x) = 1 + 1/(1 + \omega x^2)$ *with itself.* (a) *Number of degrees of freedom* N *in the Fourier extension and the degree* M *of the polynomial used to approximate* f *to the desired level of accuracy for a given value of* ω. (b) *Ratio of* N *to* M. (c) *Times required to compute the convolution using both methods.* (d) *Approximate uniform-norm error in the convolution computed using both methods.*

savings. The polynomial-based method can benefit similarly; however, we already exclude the times required to build the polynomial interpolants from our measurements because they are negligible, so they do not appear in Figure 5(c) as is.

We close this section by noting a crucial difference between our method and the polynomial-based method to which we have compared: the latter bases its approximations on samples of f at Chebyshev points, whereas the former uses equispaced points. Chebyshev points cluster quadratically near the endpoints of the approximation interval as the number of points increases, while in the center of the interval, they are much less dense. As a result, the resolution power of Chebyshev interpolants is greater near the endpoints than in the center.[11]

This suggests that the polynomial-based convolution method may perform better when the operands exhibit greater variation near the extremes of the interval than in the middle. Figure 6 shows the same four plots as Figure 5 but for the function $f(x) = 1 + 1/\bigl(1 + \omega(x+1)^2\bigr)$. This is the same Runge-like function previously considered, but it has been translated so that the "spike" occurs at $x = -1$ instead of $x = 0$. Comparing Figures 5(a) and 6(a), we see that for our earlier example, both N

[11]Statements like this can be made quantitatively precise using notions from potential theory; see [20, chapter 12] for an introduction.

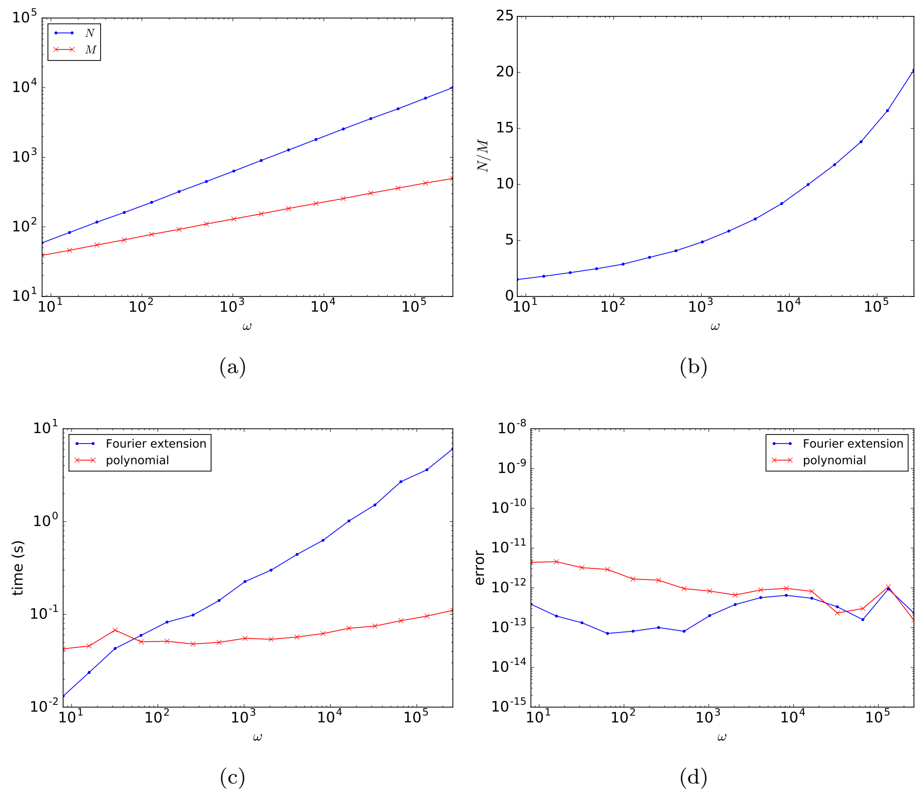

FIG. 6. *Same as Figure 5 but for the function $f(x) = 1 + 1/\big(1 + \omega(x+1)^2\big)$ instead. Most of the variation in this function occurs near the interval endpoint at $x = -1$, where the sampling points for the polynomial-based method (Chebyshev points) are densest. As a result, the polynomial-based method exhibits superior performance in this case.*

and M grow with ω at a rate of $\mathcal{O}(\omega^{1/2})$, while for this example, the rate of growth of M has been cut to $\mathcal{O}(\omega^{1/4})$. This substantial reduction in the number of degrees of freedom needed to approximate f by polynomials relative to those needed for Fourier extensions gives the polynomial-based convolution method a decisive advantage in this case, as indicated by the times in Figure 6(c).

This example does not negate the utility of our method, as not all functions of interest have their variation so heavily concentrated near the endpoints of the interval. Moreover, it is not uncommon for an application to demand samples be taken on an equispaced grid instead of a Chebyshev grid. If this is the case, polynomial interpolants cannot be used because of the Runge phenomenon [20, chapter 13].

One still can use polynomials by performing a discrete least-squares approximation (as is done for Fourier extensions) instead of interpolation; however, this comes with some disadvantages. In order to ensure stable convergence, the number of samples taken must be proportional to the square of the degree of the polynomial used [4, 8, 19]. This severely limits the size of the degree that can be used in practice.[12]

[12]Assuming the constant of proportionality is 1, even storing the (dense) $10^6 \times 10^3$ matrix for the least-squares problem with a degree-1000 polynomial requires 8 GB of memory in double precision. In contrast, a Fourier extension approximation with twice as many degrees of freedom created using the standard oversampling factor $\gamma = 2$ requires only 64 MB.

At the same time, the convergence rate degrades; if the scheme is stable, it can converge only at a rate that is subgeometric in the degree, even when the function being approximated is entire [18]. These effects severely blunt the general advantage polynomials enjoy over Fourier extensions with respect to rates of approximation, meaning that our method will be even more competitive in this context.

5. Conclusion. We have presented a new numerical method for convolving two functions with compact support. The method approximates the functions to be convolved using Fourier extensions and then convolves the extensions efficiently with the aid of the FFT to produce a Fourier extension approximation to the convolution. When the extensions are computed using one of the recently developed fast algorithms, the method takes $\mathcal{O}\big(N(\log N)^2\big)$ operations, where N is the number of degrees of freedom in each of the Fourier extensions.

While we have worked exclusively with Fourier extensions in this article, we note that the techniques described in section 3 can be used with any scheme that produces sum-of-exponentials approximations of the form (3.6). We plan to investigate the utility of our approach for computing convolutions in higher dimensions as well as the possibility of using discretizations based on our method for solving convolution integral equations. Other avenues for future work include the development of a more detailed error analysis for our algorithm by using results from literature on Fourier extensions [2, 3] and studying the behavior of our algorithm in the presence of noise.

Acknowledgments. We thank Daan Huybrechs for his patient and detailed answers to our many questions about Fourier extensions and for providing us assistance with using the FrameFun.jl software package to perform our numerical experiments. We also thank Alex Townsend for his comments on a draft of our manuscript that led us to make several enhancements. Finally, we thank the anonymous referees, whose careful reading and feedback helped us to significantly improve our work.

REFERENCES

[1] B. Adcock and D. Huybrechs, *On the resolution power of Fourier extensions for oscillatory functions*, J. Comput. Appl. Math., 260 (2014), pp. 312–336.

[2] B. Adcock and D. Huybrechs, *Frames and Numerical Approximation*, preprint https://arxiv.org/abs/1612.04464v2 [math.NA], 2017.

[3] B. Adcock, D. Huybrechs, and J. Martín-Vaquero, *On the numerical stability of Fourier extensions*, Found. Comput. Math., 14 (2014), pp. 635–687.

[4] B. Adcock, R. Platte, and A. Shadrin, *Optimal Sampling Rates for Approximating Analytic Functions from Pointwise Samples*, preprint https://arxiv.org/abs/1610.04769v2 [math.NA], 2017.

[5] B. Adcock and J. Ruan, *Parameter selection and numerical approximation properties of Fourier extensions from fixed data*, J. Comput. Phys., 273 (2014), pp. 453–471.

[6] K. Atkinson, *The Numerical Solution of Integral Equations of the Second Kind*, Cambridge University Press, Cambridge, 1997.

[7] J. P. Boyd, *A comparison of numerical algorithms for Fourier extension of the first, second, and third kinds*, J. Comput. Phys., 178 (2002), pp. 118–160.

[8] J. P. Boyd and F. Xu, *Divergence (Runge Phenomenon) for least-squares polynomial approximation on an equispaced grid and Mock-Chebyshev subset interpolation*, Appl. Math. Comput., 210 (2009), pp. 158–168.

[9] H. Brunner, *Collocation Methods for Volterra Integral and Related Functional Differential Equations*, Cambridge University Press, Cambridge, 2004.

[10] H. Brunner and P. J. van der Houwen, *The Numerical Solution of Volterra Equations*, Elsevier, Amsterdam, 1986.

[11] O. P. Bruno, Y. Han, and M. M. Pohlman, *Accurate, high-order representation of complex three-dimensional surfaces via Fourier continuation analysis*, J. Comput. Phys., 227 (2007), pp. 1094–1125.

[12] T. A. Driscoll, N. Hale, and L. N. Trefethen, eds., *Chebfun Guide*, Pafnuty Publications, Oxford, 2014.

[13] W. Feller, *On the integral equation of renewal theory*, Ann. Math. Statist., 12 (1941), pp. 243–267.

[14] N. Hale and A. Townsend, *An algorithm for the convolution of Legendre series*, SIAM J. Sci. Comput., 36 (2014), pp. A1207–A1220.

[15] D. Huybrechs, *On the Fourier extension of nonperiodic functions*, SIAM J. Numer. Anal., 47 (2010), pp. 4326–4355.

[16] M. Lyon, *A fast algorithm for Fourier continuation*, SIAM J. Sci. Comput., 33 (2011), pp. 3241–3260.

[17] R. Matthysen and D. Huybrechs, *Fast algorithms for the computation of Fourier extensions of arbitrary length*, SIAM J. Sci. Comput., 38 (2016), pp. A899–A922.

[18] R. B. Platte, L. N. Trefethen, and A. B. J. Kuijlaars, *Impossibility of fast stable approximation of analytic functions from equispaced samples*, SIAM Rev., 53 (2011), pp. 308–318.

[19] E. A. Rakhmanov, *Bounds for polynomials with a unit discrete norm*, Ann. of Math. (2), 165 (2007), pp. 55–88.

[20] L. N. Trefethen, *Approximation Theory and Approximation Practice*, SIAM, Philadelphia, 2013.

[21] C. Van Loan, *Computational Frameworks for the Fast Fourier Transform*, Front. Appl. Math. 10, SIAM, Philadelphia, 1992.