SIAM J. SCI. COMPUT.
Vol. 33, No. 5, pp. 2519–2535

A SINC FUNCTION ANALOGUE OF CHEBFUN*

MARK RICHARDSON† AND LLOYD N. TREFETHEN†

Abstract. Chebfun is an established software system for computing with functions of a real variable, but its capabilities for handling functions with singularities are limited. Here an analogous system is described based on sinc function expansions instead of Chebyshev series. This experiment sheds light on the strengths and weaknesses of sinc function techniques. It also serves as a review of some of the main features of sinc methods, including construction, evaluation, zerofinding, optimization, integration, and differentiation.

Key words. Chebfun, sinc function, barycentric interpolation, spectral method

AMS subject classifications. 41-04, 65D05, 65T40

DOI. 10.1137/110825947

1. Introduction.

Chebfun is a widely used MATLAB tool for calculating with functions of a real variable. In Chebfun, every function is represented by a piecewise polynomial, with each piece consisting of a polynomial interpolant through appropriately scaled Chebyshev points. The speed and accuracy for many problems are remarkable, even when the polynomial degrees are in the thousands [22].

Polynomials, however, are not efficient at representing functions with singularities. Point discontinuities (e.g., $\mathrm{sign}(x)$) are no problem for Chebfun, which introduces breakpoints quickly and accurately to represent them. More genuine singularities are a challenge, however $(x^{1/2}, x\log x, \Gamma(x), \ldots)$. Chebfun addresses this challenge in three ways [23, Chap. 9]. One is the explicit incorporation of singularities of the x^α type, where α can be any real number, either user-specified or automatically determined. If $0 < \alpha < 1$, then Chebfun attempts to approximate the function $f(x) = g(x)x^{1-\alpha}$, for which it is hoped that an efficient representation can be obtained and then related back to g in the obvious way. Though building such an approximation is often possible, further computations can be awkward; for example, even the seemingly innocuous task of adding a constant to g is difficult. Another method is the use of variable transformations, particularly for square root singularities. This can be effective, but it is limited in generality. A third technique is the automatic subdivision of an interval in "splitting on" mode, allowing representation of very general singularities at some cost in speed and smoothness. Detailed in [16], this strategy is similar to the *hp*-adaptivity commonly employed in finite element methods.

Nevertheless, despite these features, Chebfun has nothing like the speed and reliability for functions with singularities that it enjoys for smooth ones. For example, though Chebfun can construct short single-piece representations of $p(x) = x^{0.3}$ and $q(x) = x^{0.5}$ on $[0, 1]$, the computation of $p + q$ is problematic.

There is a well-known technology for representing functions with quite general endpoint singularities: expansions in mapped sinc functions. Over the years a number

*Submitted to the journal's Software and High-Performance Computing section February 28, 2011; accepted for publication (in revised form) June 29, 2011; published electronically October 13, 2011. This work was supported by the Engineering and Physical Sciences Research Council (UK), EP/E045847.

http://www.siam.org/journals/sisc/33-5/82594.html

†Oxford University Mathematical Institute, 24–29 St. Giles', Oxford OX1 3LB, UK (mark.richardson@maths.ox.ac.uk, http://people.maths.ox.ac.uk/richardsonm/, trefethen@maths.ox.ac.uk, http://people.maths.ox.ac.uk/trefethen/).

of people have advocated these methods, especially Frank Stenger of the University of Utah, who has written two books on the subject [18, 19], the second serving as a user's guide for the Sinc-Pack MATLAB package. In principle, this technique offers the prospect of treating nearly arbitrary singularities in a uniform manner. The price to be paid is a slowdown in convergence from C^{-N} to $C^{-\sqrt{N}}$ as a function of the number N of sample points, even for analytic functions.

How does it work in practice? Can sinc functions be used as the basis of a Chebfun-like system which quickly and reliably computes with functions on intervals, even when there are endpoint singularities? This paper reports the construction of a Sincfun system, modeled as closely as possible on Chebfun, which was built to try to answer these questions. More precisely, Sincfun approximately duplicates the original "classic" Chebfun of Battles and Trefethen before the introduction of general intervals $[a, b]$, piecewise representations, solution of differential equations, and other enhancements [1]. The result is a working system which can readily be compared with Chebfun in all kinds of respects.

Before describing the design of Sincfun, we give an indication of its capabilities. In Chebfun, the function $f(x) = x \log x$ on $[0, 1]$ is represented by a polynomial of degree more than 30,000:

```
>> f = @(x) x.*log(x);
>> fc = chebfun(f,[0 1]);
   length(fc)
   ans = 32528
```

(Chebfun in "splitting on" mode gets a shorter representation, but it takes longer and lacks smoothness for operations like differentiation.) In Sincfun the number of sample points shrinks by a factor of more than 100:

```
>> fs = sincfun(f,[0 1]);
   length(fs)
   ans = 306
```

Both the Chebfun and Sincfun representations are accurate and effective for further computations, as we illustrate by computing the definite integral of f^2, whose exact value is $2/27$:

```
>> tic, sum(fc.^2), toc
   ans = 0.074074074074074
   Elapsed time is 0.168881 seconds.

>> tic, sum(fs.^2), toc
   ans = 0.074074074074074
   Elapsed time is 0.049372 seconds.
```

Sincfun outperforms Chebfun by a factor of 3, though the difference is not as large as one may have expected, given the size of the initial representations. This discrepancy is due to various inefficiencies present in the Sincfun construction process.

The aim of this paper, then, is to set forth some of the issues that arise in designing a Sincfun system, and to draw conclusions about the advantages and disadvantages of sinc function methods for practical computation. We assume that the reader is familiar enough with Chebfun to understand readily the examples just given. Our conclusions are presented in section 8.

2. Sinc and transplanted sinc interpolants. Throughout this paper we will be concerned with functions on the problem domain $[0,1]$, the x variable, and their transplants to functions on the Fourier domain $\mathbb{R}$, the s variable. To keep the two settings clear we use lowercase letters for the former and uppercase for the latter: $g(x)$ and $G(s)$. A typewriter font such as $\mathtt{g}$ labels computer variables such as the Sincfun representation of g.

Following [18] and [19], we define the basic sinc function by

$$S(s) = \frac{\sin(\pi s)}{\pi s}, \tag{2.1}$$

taking the value 1 at $s = 0$ and 0 at other integers $s \in \mathbb{Z}$. Shifting and scaling to the grid $h\mathbb{Z}$ for some $h > 0$ gives the sinc function

$$S_k^{(h)}(s) = S\left(\frac{s}{h} - k\right) \tag{2.2}$$

for any $k \in \mathbb{Z}$, which takes the value 1 at $s = kh$ and 0 at other points $s \in h\mathbb{Z}$. If G is a function defined on $\mathbb{R}$ that decays sufficiently fast as $|s| \to \infty$, then the series

$$G^{(h)}(s) = \sum_{k=-\infty}^{\infty} G(kh)\, S_k^{(h)}(s) \tag{2.3}$$

converges absolutely and gives an approximation to G. This function goes by various names, including the *Whittaker cardinal function* or the *sinc*, *band-limited*, or *Fourier interpolant* to g. Clearly $G^{(h)}$ interpolates G on $h\mathbb{Z}$, and it can be equivalently defined by Fourier analysis: $G^{(h)}$ is the function obtained if you take the semidiscrete Fourier transform $\hat{G}$ of G on $h\mathbb{Z}$, which is a function defined on $[-\pi/h, \pi/h]$, and then evaluate the inverse transform formula not just for $s \in h\mathbb{Z}$ but for all $s \in \mathbb{R}$. Thus $G^{(h)}$ is band-limited in the sense that it contains energy only at wave numbers $\xi \in [-\pi/h, \pi/h]$. See Chapter 2 of [20].

If G is smooth, then $G^{(h)}$ converges rapidly to G as $h \to 0$. In particular, for $d > 0$, let H_d denote the infinite strip in the complex plane defined by $-d < \operatorname{Im} s < d$. If G is analytic and bounded and decays sufficiently fast as $|s| \to \infty$ in H_d, then

$$\|G - G^{(h)}\| = \mathcal{O}(e^{-\pi d/h}), \qquad h \to 0,$$

where $\|\cdot\|$ is the ∞-norm over $\mathbb{R}$. In other words, sinc interpolants on an infinite grid in $\mathbb{R}$ are *spectrally accurate.* See Chapter 4 of [20].

Next, we transplant these ideas to $x \in [a,b]$ using a function $\varphi : (a,b) \to \mathbb{R}$:

$$s = \varphi(x) = \log\left(\frac{x-a}{b-x}\right), \qquad x = \varphi^{-1}(s) = \frac{a + be^s}{1 + e^s}. \tag{2.4}$$

The function φ^{-1} is essentially a hyperbolic tangent. For any $d < \pi$, it maps the strip H_d in the complex s-plane onto the lens-shaped region L_d in the complex x-plane bounded by two circular arcs meeting with half-angle d at $x = a$ and $x = b$. See Figure 2.1.

Suppose f is a continuous function defined on $[a,b]$. By subtracting off a linear function taking the same values as f at the endpoints, we obtain a function g satisfying $g(a) = g(b) = 0$. To approximate g on $[a,b]$, we transplant it to the function

$$G(s) = g(\varphi^{-1}(s)) \tag{2.5}$$

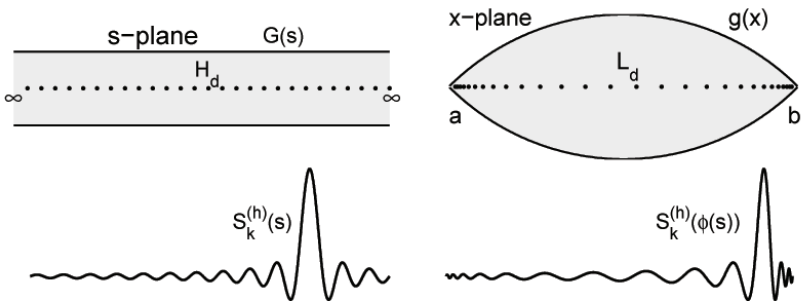

FIG. 2.1. *The function φ^{-1} maps the infinite strip H_d in the s-plane onto the lens-shaped region L_d in the x-plane. Below, a sinc function on the regular grid in $\mathbb{R}$ and its image on the mapped grid in (a, b).*

defined for $s \in \mathbb{R}$. For any $h > 0$, the sinc interpolant $G^{(h)}$ gives an approximation to G, and transplantation back to $[0, 1]$ gives an approximation to g,

$$g^{(h)}(x) = G^{(h)}(\varphi(x)) = \sum_{k\in\mathbb{Z}} g(\varphi^{-1}(kh))\, S_k^{(h)}(\varphi(x)). \tag{2.6}$$

Equations (2.3) and (2.6) represent sinc and transplanted sinc interpolants on an infinite grid. For numerical work, we need a finite grid, and accordingly we truncate the infinite sum. Given integers $m \le n$, typically with $m \ll 0 \ll n$, we define

$$G^{(h,m,n)}(s) = \sum_{k=m}^{n} G(kh)\, S_k^{(h)}(s) \tag{2.7}$$

and the corresponding transplantation to $[a, b]$,

$$g^{(h,m,n)}(x) = G^{(h,m,n)}(\varphi(x)) = \sum_{k=m}^{n} g(\varphi^{-1}(kh))\, S_k^{(h)}(\varphi(x)). \tag{2.8}$$

In this crucial formula we see the two approximations that are made in working with sinc interpolants. On the one hand, there is a sampling error associated with the finite size of h. If g is analytic, this error can be expected to decay exponentially as a function of h^{-1}. On the other hand, there is an error associated with truncating the infinite sum to a sum from m to n. If g satisfies, for some real $\alpha, \beta > 0$,

$$g(x) = \begin{cases} \mathcal{O}(|x-a|^{\alpha}), & x \to a, \\ \mathcal{O}(|x-b|^{\beta}), & x \to b, \end{cases} \tag{2.9}$$

then G will decay exponentially as $s \to \infty$, so this error can be expected to decay exponentially as a function of $|m|$ and n. It is common to pick

$$N = \mathcal{O}(\min(|m|, n)), \qquad h = \mathcal{O}(N^{-1/2}). \tag{2.10}$$

With these choices, one expects the two sources of error to approximately balance, giving a total error of $\mathcal{O}(C^{-\sqrt{N}})$ for some $C > 1$.

We make these ideas precise with the following theorem.

THEOREM 2.1. *Let g satisfy the condition* (2.9) *and be analytic in the lens-shaped region L_d for some $d \in (0, \pi)$. Let also the Fourier transform of the function*

$G(s) = g(\varphi^{-1}(s))$ decay exponentially, satisfying $|\hat{G}(\xi)| = \mathcal{O}(e^{-\gamma|\xi|})$ as $|\xi| \to \infty$ for some $\gamma > 0$. If $g^{(h,m,n)}$ is defined by (2.8), with any choices of h, m, n satisfying

$$h \le C_1 N^{-1/2}, \quad |m| \ge C_2 N^{1/2}, \quad n \ge C_3 N^{1/2}$$

for constants $C_1, C_2, C_3 > 0$, and sufficiently large N defined by (2.10), then

$$\|g - g^{(h,m,n)}\| = \mathcal{O}(C_4^{-\sqrt{N}})$$

as $N \to \infty$ for some $C_4 > 1$.

Proof. See [18, pp. 137–138] or [19, pp. 26–29]. □

The assumptions of this theorem enable the approximation of certain classes of function, and disallow others. For example, the condition (2.9) enables us, in theory, to work with all singularities of the form x^α where $\alpha > 0$ (in practice, very small values of α can be problematic, but this is due to limitations of finite-precision arithmetic, rather than a fundamental mathematical reason). On the other hand, functions that would be disallowed include, for example, those of the explicit form x^α where $\alpha < 0$, and functions that oscillate infinitely frequently near the singularity such as $\sin(1/x)$.

3. Sincfuns and the Sincfun constructor. Mathematically, a sincfun is a finite series (2.8), plus a linear function to accommodate possible nonzero values at the endpoints. Computationally, it is a member of a class in MATLAB which has a number of fields, of which the following are the principal ones. As the variable `g.domain` indicates, sincfuns are defined on arbitrary intervals $[a, b]$, though for simplicity the mathematical formulas discussed in this paper usually refer to $[0, 1]$.

- `g.domain` = vector $[a, b]$ specifying the interval of definition, default $[0, 1]$
- `g.sdomain` = vector $[s_{\text{left}}, s_{\text{right}}]$ outside which $G(s)$ is negligible
- `g.numterms` = integer vector $[|m|, n]$
- `g.vals` = vector of values at the grid points
- `g.endvals` = vector $[g(a), g(b)]$ of values at the endpoints

These data, together with (2.1)–(2.8), fully define the value `g(x)` for all $x \in [a, b]$. The reason for taking the default interval to be $[0, 1]$ rather than $[-1, 1]$ as in Chebfun has to do with accuracy of sinc representations and floating-point arithmetic, matters discussed in sections 7 and 8.

Given a function g to be represented, how does the system decide the values of these parameters? Following the pattern used by Chebfun, the aim is, by sampling g at various points, to determine m, n, h and so on, so that the representation captures the function to high accuracy. In Chebfun, the function would be sampled at $9, 17, 33, \ldots$ Chebyshev points until convergence to machine precision was deemed to have occurred, based on the decay of Chebyshev coefficients.

On the face of it, construction of a sincfun should be more complicated since there are three essential parameters to determine rather than one: m, n, and h. This corresponds to the $\sqrt{N}$ type of convergence suggested by Theorem 2.1. However, we have found that a simple two-step approach is effective. In a first step, we sample G at values $s \ll 0$ and $s \gg 0$. By processes of bisection, we determine values s_{left} and s_{right} such that G appears to be negligible outside $[s_{\text{left}}, s_{\text{right}}]$. From this point on, the construction of the sincfun becomes a matter of Fourier discretization in a fixed interval, converging at a rate governed by N, not $\sqrt{N}$.

Specifically, we have a function $G(s)$ on the interval $[s_{\text{left}}, s_{\text{right}}]$ decaying exponentially to (nearly) 0 at both ends. Because of the exponential decay, it is a good

approximation to regard G as periodic. We now want to represent G by a Fourier series through equispaced points, and the only computational issue is to decide how many such points are needed. This is done in the Chebfun fashion. The function is sampled at $256, 512, 1024, \ldots$ points, and for each of these grids, the Fourier coefficients of the trigonometric interpolant are computed by the fast Fourier transform (FFT). When these decay to a level of about 10^{-15} relative to the scale of G, the grid is deemed to be fine enough. Engineering safeguards are included to minimize the risk of the process being fooled.

The choice of grid sizes beginning at 256 marks a difference from Chebfun, which samples functions on grids of $9, 17, 33, \ldots$ points. The reason is that it is rare for a sincfun to have length less than a few hundred, since one would hardly expect to encounter functions of the form (2.8) for small m and n in the wild. (A special case is a linear function, which the system quickly reduces to length 0—all the content in this case is in the vector `g.endvals`.)

To illustrate sincfun construction, consider the function $f(x) = -\sqrt{x}\log x$. In the initial bisection process, the constructor samples F at the left, for $x \approx 0$, at approximately the following values of s:

$$-88.5,\ -44.3,\ -66.4,\ -77.5,\ -83.0,\ -80.2,\ -81.6,\ -80.9,\ -80.6,\ -80.8,\ -80.9.$$

The last value is $s \approx -80.85335$, corresponding to $x = 7.78 \times 10^{-36}$, where f takes the value 2.24×10^{-16}, which is approximately machine precision. At this point, the constructor has decided that F appears to be negligible for $s < s_{\text{left}}$, and nonnegligible for $s > s_{\text{left}}$.

Next, the constructor bisects on the right, for $x \approx 1$, sampling F at approximately these values of s:

$$36.0,\ 18.0,\ 0,\ 27.0,\ 31.5,\ 33.8,\ 34.9,\ 35.4,\ 35.6,\ 36.0.$$

In this case the final value corresponds to $x \approx 1 - 2.22 \times 10^{-16}$, that is, 1 minus machine precision. This is no coincidence, since f is analytic at $x = 1$.

From this point forward, the approximation interval $[s_{\text{left}}, s_{\text{right}}] \approx [-80.85, 36.04]$ is fixed. Over this interval, $F(s)$ looks as shown in Figure 3.1.

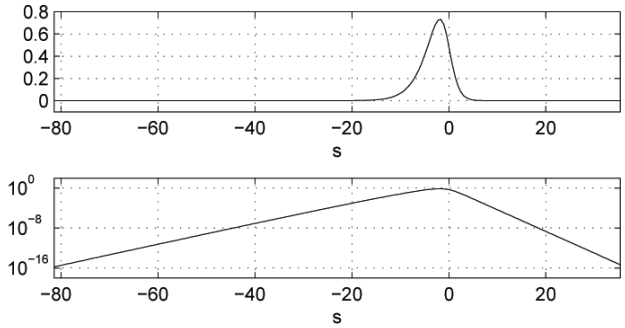

FIG. 3.1. *Example function $F(s)$ corresponding to $f(x) = -\sqrt{x}\log x$, shown on linear and log scales. In the first stage of the construction process, the interval $[s_{\text{left}}, s_{\text{right}}]$ is determined on which $F(s)$ is not negligible.*

Figure 3.2 shows the absolute values of the Fourier coefficients $|\widehat{G(kh)}|$ obtained with the FFT, for grids of size 256, 512, and the final choice, 434. In the first plot,

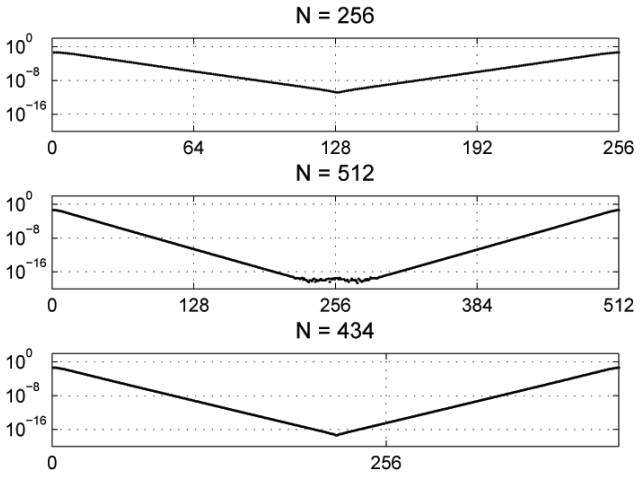

FIG. 3.2. *Fourier coefficients of the function restricted to this interval. In the second stage of the construction, a grid number is determined that resolves the function to machine precision.*

the coefficients have not yet fallen to machine precision: h is too large. In the second, there are many coefficients down at that level: h is too small. For this function (as indeed for many functions) $N = 512$ proves to be the first power of 2 that resolves the function to machine precision.

4. Evaluation and composition of sincfuns. Evaluating a sincfun requires a way of computing the expansion (2.8) for arbitrary $x \in [a, b]$. Perhaps the most obvious and straightforward way of doing so is to evaluate each of the sinc functions $S_k^{(h)}(\varphi(x))$, multiply by the function coefficients $g_k = g(\varphi^{-1}(kh))$, and then sum over k. This is the method used in Sinc-Pack [19] and it involves the evaluation of $|m|+n+1$ sine functions. Computations for vector-valued x follow from using the appropriate MATLAB componentwise vector operators.

Following the analogous technique for evaluating polynomials described in [3], Sincfun uses a barycentric representation to compute (2.8). First documented by Berrut [2], the barycentric formula for sinc functions enables the expansion to be computed without the need to evaluate any sine functions. Our experiments show that using such a representation leads to a computational saving of approximately a factor of 4 compared to evaluating (2.8) directly.

Following [2] we provide a brief derivation of the *weighted barycentric formula* used in Sincfun. Noting first that $\sin\left[\pi\left(\frac{s}{h}-k\right)\right] = (-1)^k \sin\left(\frac{\pi s}{h}\right)$, we see that the sine term may be factored out of the sum in (2.3) in order to write

$$G^{(h)}(s) = \frac{h}{\pi} \sin\left(\frac{\pi s}{h}\right) \sum_{k=-\infty}^{\infty} \frac{(-1)^k}{s-kh} G(kh). \tag{4.1}$$

Though this formula is simple enough in appearance, it is in fact numerically unstable, since large errors are introduced when s is close to one of the interpolation abscissae. In order to combat this, we first represent the constant function 1 using (4.1),

$$1 = \frac{h}{\pi} \sin\left(\frac{\pi s}{h}\right) \sum_{k=-\infty}^{\infty} \frac{(-1)^k}{s-kh}, \tag{4.2}$$

and use this to replace the factors multiplying the sum in (4.1) to obtain

$$G^{(h)}(s) = \frac{\displaystyle\sum_{k=-\infty}^{\infty} \frac{(-1)^k}{s-kh} G(kh)}{\displaystyle\sum_{k=-\infty}^{\infty} \frac{(-1)^k}{s-kh}}. \tag{4.3}$$

The fact that any large terms now appear in both the numerator and the denominator of (4.3) has the effect of eliminating the numerical instability present in (4.1). Moreover, since a constant function is band-limited, (4.2) and (4.3) both provide exact representations. However, upon truncation, the right-hand side of (4.2) converges only linearly towards 1, so unfortunately (4.3) is not of much practical use.

Berrut's solution is to replace the constant function 1 in (4.2) with a rapidly decreasing weight function w whose decay should approximately match that of the transplanted function G. Making this replacement and truncating the sums to $|m| + n + 1$ terms then yields the weighted barycentric formula used in Sincfun:

$$g_w^{(h,m,n)}(x) = w(\varphi(x)) \frac{\displaystyle\sum_{k=m}^{n} \frac{(-1)^k}{s-kh} g_k}{\displaystyle\sum_{k=m}^{n} \frac{(-1)^k}{s-kh} w(kh)}. \tag{4.4}$$

Among the several candidates for w considered in [2], a transformed Gaussian appears to offer the best combination of accuracy and computational flexibility. However, in developing this aspect of Sincfun, we found that it was necessary to make some adjustments to Berrut's techniques in order to deal with asymmetric truncations of the sum. This situation arises when f is singular at an endpoint, requiring Sincfun to demand an asymmetric $[s_{\text{left}}, s_{\text{right}}]$ interval. Typically, if the singularity is at $x = 0$, then $|s_{\text{left}}| \gg s_{\text{right}}$, and using a standard Gaussian will result in slow convergence, since the decay of w will not match the decay of G. Instead, we propose to scale the function $w(s) = e^{-\mathcal{M}(s)^2}$, where $\mathcal{M}$ is a Möbius transformation. The asymmetric decay conditions are met by imposing

$$(1)\ \ w(s_{\text{left}}) = \varepsilon_{\text{m}}, \qquad (2)\ \ w(0) = 1, \qquad (3)\ \ w(s_{\text{right}}) = \varepsilon_{\text{m}},$$

where $\varepsilon_{\text{m}} = 2^{-52} \approx 2.22 \times 10^{-16}$ in IEEE double precision. Solving then yields

$$\mathcal{M}(s) = \frac{s}{p - qs}, \qquad \kappa = \sqrt{-\log_e(\varepsilon_{\text{m}})},$$

$$p = \frac{s_{\text{left}}}{\kappa}\left(\frac{s_{\text{left}} + s_{\text{right}}}{s_{\text{left}} - s_{\text{right}}} + 1\right), \qquad q = \frac{s_{\text{left}} + s_{\text{right}}}{\kappa(s_{\text{left}} - s_{\text{right}})}.$$

A further method of evaluating the sinc interpolant was suggested by Gautschi [11] and involves a simple reordering of (2.8). However, while this preserves the exact representation, the computation does not appear to be possible to vectorize, since Gautschi's rearrangement requires a new index set to be computed for each evaluation. It is therefore our opinion that, at least for an implementation in MATLAB, (4.4) provides the best available method for evaluating the sinc expansion.

Once this has been implemented, further operations on sincfuns become possible. In particular, if `f` is a sincfun, and `op(·)` is a standard MATLAB mathematical function handle or another sincfun, then the composition `op(f)` is well defined computationally. The output is then a further sincfun defined over the same domain as `f`.

As in Chebfun, composition has been implemented by overloading the standard mathematical operators such as `exp`, `log`, and `sin`. In each case, a function handle representing `op(f(x))` is sent to the Sincfun constructor, ensuring that the result is also a uniformly accurate mapped sinc interpolant. The procedure is very similar to the construction process described in section 3. For each evaluation vector `x` of $256, 512, \ldots$ points, the vector `f(x)` is first computed with the barycentric formula. Next, the operator `op` is applied to the vector in order to compute `op(f(x))`, and the Fourier coefficients are computed with the FFT. In the case where `op` is another sincfun, for the computation `op(f)` to make sense, the range of `f` must lie within the domain of `op`. In this case, the barycentric formula is used twice in succession; first to evaluate `f(x)`, and then `op(f(x))`.

5. Rootfinding. We now consider the crucial problem of approximating the zeros of the transplanted functions on $s \in \mathbb{R}$. Provided these can be computed accurately, approximations to the roots of functions on $[a, b]$ may be obtained with the inverse mapping φ^{-1}. These are then the basis of many subsequent operations [1].

In Sincfun, the transplanted functions $G(s) = g(\varphi^{-1}(s))$ decay to zero as $|s| \to \infty$ and are smooth enough to be represented by a convergent Fourier series over $[s_{\text{left}}, s_{\text{right}}]$. In such a setting, the standard technique for approximating roots is to compute the eigenvalues of a companion matrix [17]. On a bounded interval, other approximation bases may be used, and equivalent formulations of the problem exist. For example, in Chebfun [1], roots are approximated by computing the eigenvalues of a colleague matrix, a method based upon expansions in Chebyshev polynomials [12].

Though the Fourier setting is natural for sinc interpolants, we have found that rather than solving a companion matrix eigenvalue problem, it is in fact more economical to reapproximate $F(s)$ using an expansion in Chebyshev polynomials and work with the corresponding colleague matrix. Two observations motivate this.

First, the function $G(s)$ that Sincfun approximates on (a truncation of) $\mathbb{R}$ is not a direct transplant of the function $f(x)$ on $[a, b]$, but rather a transplant of the function $g(x) = f(x) - h(x)$, where $h(x)$ is a linear function satisfying $h(a) = f(a)$ and $h(b) = f(b)$. Though there does not appear to be a trivial relationship between the roots of $G(s)$ on $[s_{\text{left}}, s_{\text{right}}]$ and the roots of $f(x)$ on $[a, b]$, the roots of the function $F(s) = f(\varphi^{-1}(s))$ are directly related to the roots of f by the inverse mapping φ^{-1}. Since in general F does not decay to zero as $|s| \to \infty$, it is not possible to represent F over $[s_{\text{left}}, s_{\text{right}}]$ with a Fourier series without "flipping" the function in order to make it artificially periodic. This of course doubles the length of the representation. Second, contrary to their Fourier counterparts, Chebyshev interpolants do not require periodicity of the underlying function, only smoothness. If one wishes to approximate a function over an arbitrary subset of its initial domain of definition, in general this will not be possible with a Fourier basis, since the function will not be periodic there. These considerations lead us to the idea of expressing $F(s)$ as a finite Chebyshev series on $[s_{\text{left}}, s_{\text{right}}]$ and using an adaptive subdivision scheme to ensure that the size of each individual colleague matrix is never too large.

Sincfun uses a variation of the *recursive subdivision* technique originally proposed by Boyd [8] and now also used in Chebfun. The process involves first sampling

$G(s)$ over the interval $[s_{\text{left}}, s_{\text{right}}]$ at a set of 100 scaled Chebyshev points. These are then transformed into Chebyshev coefficients using an algorithm based on the FFT. If the coefficients decay in such a way that some are below a certain tolerance (typically 10^{-15}), then the unneeded terms are truncated from the expansion, and a colleague matrix eigenvalue problem is constructed and solved to yield the roots over the interval. If on the other hand the coefficients have not decayed sufficiently, then the interval $[s_{\text{left}}, s_{\text{right}}]$ is approximately divided in half, and the process is repeated on each subinterval. This occurs recursively until the function has been resolved over all of $[s_{\text{left}}, s_{\text{right}}]$ by a sequence of polynomials of degree 100 or less. A colleague matrix is formed for each piecewise interpolant, and the results combined to yield approximations to the roots over the entire interval.

Algorithms based upon recursive subdivision typically lead to drastically reduced complexity of the rootfinding problem, often down from $\mathcal{O}(N^3)$ to $\mathcal{O}(N^2)$ [1, 8]. This is because rather than solving a single large eigenvalue problem corresponding to a global interpolant, the computation is instead broken down into several smaller steps, each corresponding to a local interpolant. In Sincfun, we also obtain an $\mathcal{O}(N^2)$ method, and some sample timings displaying this are shown in Figure 5.1.

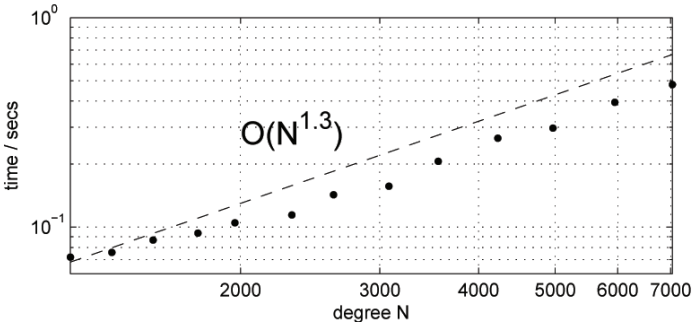

FIG. 5.1. *Timings for the rootfinding algorithm applied to the Sincfun representation of the function* $f_7(x) = \sqrt{x}\cos(k\pi x)$ *on* $[0, 1]$*, where* k *takes successive values from the set* {15, 19, 23, 29, 35, 45, 55, 67, 83, 103, 127, 157, 193}*, and* $N = |m| + n + 1$*. The convergence is better than* $\mathcal{O}(N^2)$*. Though the slope of the displayed dashed line matches the experimental timings for this function, we do not claim that Sincfun's algorithm is in general* $\mathcal{O}(N^{1.3})$*.*

Since the eigenvalues of a colleague matrix are exactly equal to the roots of a corresponding finite Chebyshev series, the errors in Sincfun's computed roots are of approximately the same order of magnitude as the approximation error of the sinc interpolant. Thus, if an adaptive sincfun construction has converged, we may expect the computed roots to be accurate to machine precision. See Table 5.1 for some examples of this involving three oscillatory functions.

TABLE 5.1
Details of sample rootfinding computations for functions approximated on $[0, 1]$*.*

		Sincfun length	Time (s)	$\|\|\cdot\|\|_\infty$ error
f_8	$\sin(4\pi x)$	496	0.065234	8.33×10^{-16}
f_9	$\sin(40\pi x)$	1659	0.121728	2.28×10^{-15}
f_{10}	$\sin(400\pi x)$	10771	1.431303	4.44×10^{-16}

6. Integration and differentiation. Following Chebfun, definite integration in Sincfun has been implemented by overloading the MATLAB method `sum()`. The standard quadrature formula for sinc expansions is beautiful in its simplicity, and we derive it following Stenger [18, pp. 189–190].

We consider first the function

$$\rho(x) = \frac{g(x)}{\varphi'(x)}, \qquad \text{where} \qquad \varphi'(x) = \frac{b-a}{(x-a)(b-x)}. \tag{6.1}$$

If g is analytic in the lens-shaped region L_d, then ρ is analytic there also, and it is possible to construct the following $|m| + n + 1$ term sinc approximation:

$$\rho^{(h,m,n)}(x) = \sum_{k=m}^{n} \rho_k S_k^{(h)}(\varphi(x)), \qquad \rho_k = \frac{g(x_k)}{\varphi'(x_k)}. \tag{6.2}$$

As before, we have that $\|\rho - \rho^{(h,m,n)}\| = O(C_5^{-\sqrt{N}})$ for some $C_5 > 1$, where $N = \min(|m|, n)$. From (6.1) and (6.2), we see that

$$\int_a^b g(x)\mathrm{d}x = \int_a^b \rho(x)\varphi'(x)\mathrm{d}x \approx \sum_{k=m}^{n} \rho_k \int_a^b S_k^{(h)}(\varphi(x))\varphi'(x)\mathrm{d}x.$$

Then, using the identity

$$\int_a^b S_k^{(h)}\left(\varphi(x)\right)\varphi'(x)\mathrm{d}x = h,$$

we arrive at the approximation to the integral,

$$\int_a^b g(x)\mathrm{d}x \approx h \sum_{k=m}^{n} \frac{g(x_k)}{\varphi'(x_k)}. \tag{6.3}$$

One of the attractive features of Chebfun is that all the adaptivity is confined to the initial construction stage. Once a chebfun has been instantiated, evaluating the integral is simply a matter of working with a precomputed vector of numbers consisting of the values of the interpolant at Chebyshev points. In Chebfun, Clenshaw–Curtis quadrature (a method based on integrating the polynomial interpolant) is used to compute definite integrals. In general, the results of such computations are at least as accurate as the interpolant [21], and the same principle holds in Sincfun. If the initial construction process has converged satisfactorily, then the interpolant—and any subsequent integral computation—will be accurate to an approximate relative level of 10^{-15}. Formally, there exists a constant $C_6 > 1$ such that

$$\left| \int_a^b g(x)\mathrm{d}x - h \sum_{k=m}^{n} \frac{g(x_k)}{\varphi'(x_k)} \right| = \mathcal{O}(C_6^{-\sqrt{N}}).$$

For a detailed derivation of this result, see [18, p. 190].

We evaluate the Sincfun approximations to the definite integrals of the functions $f_j(x)$, $j = 1, \ldots, 10$, in Table 6.1. It is interesting to note that, due to the severity of the singularities, Sincfun's adaptive construction procedure did not converge for functions f_5 and f_6, yet the computed integrals are still remarkably accurate.

TABLE 6.1
Approximation errors for definite integral computations in Sincfun.

		$\|\|\cdot\|\|_\infty$ error			$\|\|\cdot\|\|_\infty$ error
f_1	$x\log(x)$	0	f_6	$(1-x)^{1/2}$	5.55**e**−16
f_2	$x^{1/4}\log(x)$	0	f_7	$x^{1/2}\cos(19x)$	3.39**e**−16
f_3	$x^{1/8}\log(x)$	1.11**e**−16	f_8	$\sin(4\pi x)$	3.34**e**−17
f_4	$x^{1/20}\log(x)$	0	f_9	$\sin(40\pi x)$	1.12**e**−16
f_5	$x^{1/30}\log(x)$	3.33**e**−16	f_{10}	$\sin(400\pi x)$	1.11**e**−15

Integrating sincfuns, then, is reasonably straightforward. Differentiating them, however, is far more problematic. There are two factors to consider here. First, and perhaps most obviously, if a function is singular at an endpoint, then its derivative could possibly be unbounded. While the issue of how to represent unbounded functions is not an intractable one (Chebfun has some capability in this regard for example), it is certainly beyond the current capabilities of Sincfun. The second issue is more fundamental and restricts us even from approximating the derivatives of smooth functions. To see why, consider differentiating the expansion (2.8),

$$\frac{\mathrm{d}}{\mathrm{d}x}g^{(h,m,n)}(x) = \varphi'(x)\frac{\mathrm{d}}{\mathrm{d}s}G^{(h,m,n)}(s). \tag{6.4}$$

Recalling that $s = \varphi(x)$, we see that this consists of the approximation to the derivative of the transplanted function multiplied by the derivative of the transplanting map. Though the *absolute* error in the $\frac{\mathrm{d}}{\mathrm{d}s}G^{(h,m,n)}$ term is small for all s, the *relative* error may be large, perhaps even $\mathcal{O}(1)$, for $|s| \gg 0$. Since the φ' term diverges to infinity as x approaches the endpoints of $[a,b]$, any errors present in the $\frac{\mathrm{d}}{\mathrm{d}s}G^{(h,m,n)}$ term are amplified, and the result is a significant loss of accuracy.

7. Performance and accuracy. Sincfun was designed in order to enable Chebfun-like computation with singular functions. For many calculations of this kind, it excels. As an example, consider the following functions with singularities at $x = 0$:

```
>> f = @(x) 3*besselj(0.3,20*x); ff = sincfun(f);
>> g = @(x) 2*sqrt(x).*cos(12*x).*log(x); gg = sincfun(g);
```

The lengths of these representations are 1193 and 776, respectively. As a very basic check of accuracy, we can evaluate the sincfuns and compare the interpolated values to the original functions:

```
>> xx = rand(1000,1); [norm(f(xx)-ff(xx),inf) norm(g(xx)-gg(xx),inf)]
ans =
     3.674838211509268e-14     3.774758283725532e-15
```

Computing the integral of the difference between these functions is straightforward:

```
>> sum(ff-gg)
ans =
     1.082105033952097e-01
```

Similarly, the roots of the difference between the sincfuns will give us the intersections. This computation can be performed, and the results plotted, with just a couple of commands (see Figure 7.1):

```
>> plot(ff), grid on, hold on, plot(gg,'--'), ylim([-1.3 2.4])
>> r = roots(ff-gg); plot(r,ff(r),'.'), hold off
```

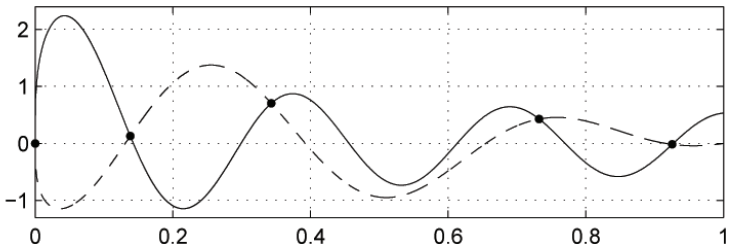

FIG. 7.1. *Plot of the sincfuns* `ff` *and* `gg`*, together with their intersections.*

Theorem 2.1 established that finite sinc expansions converge at the rate $\mathcal{O}(C^{-\sqrt{N}})$. However, from our discussion of the Sincfun construction process in section 3, we recall that in practice the convergence looks like $\mathcal{O}(C^{-N})$ down to the level of machine precision. The constant C here depends upon the length of the interval $[s_{\text{left}}, s_{\text{right}}]$ outside of which $G(s)$ is negligible. We confirm this effect in Figure 7.2, which shows the approximation error of the Sincfun representations for four functions with increasingly extreme singularities but similar oscillatory behavior. Clear exponential convergence is apparent, down to the level of rounding errors.

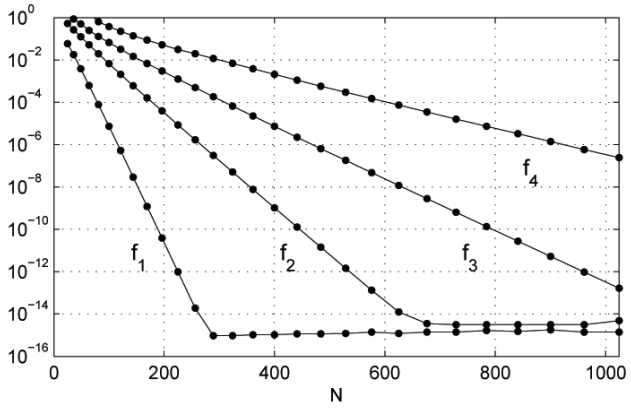

FIG. 7.2. $||f_j - f_j^{(h,m,n)}||$ *for* $f_1(x) = x\log(x)$, $f_2(x) = x^{1/4}\log(x)$, $f_3(x) = x^{1/10}\log(x)$, $f_4(x) = x^{1/20}\log(x)$ *on* $[0,1]$*, where* $N = |m| + n + 1$*. The* ∞*-norm error was estimated by sampling* f_j *and* $f_j^{(h,m,n)}$ *for each* $j = 1,2,3,4$ *at* 2000 *equally spaced points on* $[0,1]$.

In Table 7.1 we display, for the trial functions f_j considered in this paper, the $[s_{\text{left}}, s_{\text{right}}]$ interval, the corresponding $[x_{\text{left}}, x_{\text{right}}]$ interval, the number of terms m, n in the sinc expansion (2.8), and the grid spacing h. These parameters were determined by the adaptive Fourier discretization process described in section 3.

TABLE 7.1

Parameters for ten functions approximated over the default Sincfun interval $[0,1]$*, where* $x_{\text{left}} = \varphi^{-1}(s_{\text{left}})$ *and* $x_{\text{right}} = \varphi^{-1}(s_{\text{right}})$*. The constructor converged for all of the functions except* f_5 *and* f_6*, for which we see that even taking more than* 2^{16} *points is insufficient. The problem is simply that the singularities are too severe to be accurately resolved in double precision. For* f_6*, though we are only dealing with a square root, the fact that it is located at* $x = 1$ *rather than* $x = 0$ *means that Sincfun fails to converge.*

		$-s_{\text{left}}$	s_{right}	x_{left}	$1 - x_{\text{right}}$	$-m$	n	h
f_1	$x\log(x)$	40.53	35.78	2.49**e**−18	3.33**e**−16	162	143	2.502**e**−1
f_2	$x^{1/4}\log(x)$	162.93	35.00	1.73**e**−71	6.66**e**−16	568	122	2.869**e**−1
f_3	$x^{1/8}\log(x)$	325.57	34.43	3.38**e**−142	1.11**e**−15	1069	113	3.047**e**−1
f_4	$x^{1/20}\log(x)$	708.29	33.83	2.46**e**−308	2.00**e**−15	2052	98	3.452**e**−1
f_5	$x^{1/30}\log(x)$	708.29	33.53	2.45**e**−308	2.78**e**−15	62462	2957	1.134**e**−2
f_6	$(1-x)^{1/2}$	36.04	35.97	2.22**e**−16	2.22**e**−16	32792	32728	1.099**e**−3
f_7	$x^{1/2}\cos(19x)$	71.63	35.92	7.81**e**−32	2.22**e**−16	652	327	1.099**e**−1
f_8	$\sin(4\pi x)$	38.44	35.89	2.02**e**−17	2.22**e**−16	256	239	1.502**e**−1
f_9	$\sin(40\pi x)$	40.85	35.94	1.82**e**−18	2.22**e**−16	882	776	4.631**e**−2
f_{10}	$\sin(400\pi x)$	43.15	35.97	1.82**e**−19	2.22**e**−16	5874	4896	7.346**e**−3

In general, severe singularities correspond to large $[s_{\text{left}}, s_{\text{right}}]$ intervals. We observe this effect for the functions $f_1, \ldots, f_4$ in Table 7.1, for which the s_{right} values remain roughly constant, while the s_{left} values increase in magnitude. This behavior is typical for functions that are singular at $x = 0$ and highlights an important limiting factor inherent in the problem.

Suppose a function has a singularity at $x = a$, and that we wish to construct a sinc approximation to within a uniform relative accuracy of 10^{-15} on $[a, b]$. The severity of the singularity that can be coped with at a is constrained by the location of the nearest floating-point number to a in the interval $(a, b]$. This in turn is dependent upon the computational precision. For example, if $a = 0$, then in standard IEEE double-precision arithmetic, the nearest normalized floating-point number to a is $x_{\min} := 2^{-1022} \approx 2.2 \times 10^{-308}$ [14, 15]. The function $f_4(x) = x^{1/20}\log(x)$ is near to the limit of Sincfun's capabilities, and it is no coincidence that the corresponding x_{left} value in Table 7.1 is close to $x_{\min}$, since $f_4(x_{\min}) \approx \varepsilon_{\text{m}}$. In this precision, it would not be possible to fully resolve a function with an even severer singularity, such as $f_5(x) = x^{1/30}\log(x)$, since $f_5(x_{\min}) = \mathcal{O}(10^{-8})$, a value several orders of magnitude larger than ε_{m}.

A further consequence of this observation is that mapped sinc approximation is only fully effective if the singular behavior is located at $x = 0$. To see why, consider the function $f_6(x) = \sqrt{1-x}$, which has a square root at $x = 1$. The closest floating-point number to 1 in double precision is $1 - 2^{-53} = 1 - \varepsilon_{\text{m}}/2$, and as before, $f_6(1 - \varepsilon_{\text{m}}/2) = \mathcal{O}(10^{-8}) \gg \varepsilon_{\text{m}}$. The density of floating-point numbers is again insufficient to capture the function to high precision.

For the most part, composition of sincfuns with mathematical operators is unproblematic. For instance, given the Sincfun representation `fs` of $f(x) = x\log(x)$ from section 1, computations such as `sin(10*fs)` or `exp(5*fs.^3)` converge quickly. However, when the composition may introduce a new singularity, such as in the case of applying `sqrt()`, the construction process is unlikely to converge. The reason is that the Sincfun interpolant cannot be sampled accurately enough close to the singularity

in order to resolve the composition satisfactorily. In other words, the $[s_{\text{left}}, s_{\text{right}}]$ interval does not extend far enough towards $s = -\infty$. For example, we find

```
>> p = @(x) sin(x);
>> ps = sincfun(p,[0 1]);
>> sqps = sqrt(ps);
   Warning: Sincfun did not converge!
```

It is important to note, however, that it would be perfectly possible to construct the desired representation by sampling the function $\sqrt{\sin(x)}$ directly. It is only because the original Sincfun representation of `ps` did not require a large $[s_{\text{left}}, s_{\text{right}}]$ interval that the construction failed.

8. Discussion. We undertook this project to answer a question: Could the Chebyshev techniques that make Chebfun so successful for computation with smooth functions be replaced by sinc function techniques with much greater ability to handle singularities at endpoints, without too great a cost in efficiency? In short, are sinc methods ready for daily use? It seemed to us that the only way to reach an answer with confidence would be to develop a Sincfun software package, following the Chebfun model, since Chebfun incorporates such a wide range of capabilities such as algebraic operations, composition of functions, integration, differentiation, rootfinding, and optimization. We would faithfully attempt to match each such capability by a well-tuned sinc algorithm. (Chebfun also solves differential equations, but we would not attempt to go that far.)

And so we wrote Sincfun, described in this paper, reproducing much of the functionality of Version 1 Chebfun [1]. It is a real pleasure to run Sincfun and watch it handle a function like $x \log x$ without a hiccup. Readers who wish to experiment for themselves may contact the first author for a copy of the software.

Nevertheless, the conclusion we have reached is that sinc methods cannot compete with Chebyshev methods for general use. There are two aspects to this conclusion, one quite focused and essentially a well-known problem, the other more conceptual and open-ended.

The first is the matter of *accuracy difficulties near endpoints.* The whole purpose of sinc function methods is to achieve flexibility in treating singularities at endpoints, yet despite our best efforts, a difficulty persists in Sincfun that has been a well-known problem throughout the history of sinc methods. The changes of variables used by these methods bring sample points extraordinarily close to the endpoints, leading to a need for pointwise accuracy that floating-point arithmetic cannot meet. To minimize this problem, we followed the long tradition in sinc methods of putting one endpoint at $x = 0$ where possible, to take advantage of the scale-invariance of floating-point arithmetic. Even this does not eliminate all problems, however, as illustrated in section 7 by the difficulty of taking the square root of a Sincfun representation of $\sin x$, since that representation lacks the extra accuracy needed to build the square root. As discussed in section 6, the endpoint problem becomes especially acute in the computation of derivatives. For a further approach to this issue, see [6].

So sinc methods' signal advantage, accurate treatment of singularities, remains elusive. To make progress here it seems to us that one might have to go beyond the usual floating-point arithmetic near endpoints. This may be feasible, but it was beyond the scope of this investigation.

To explain the second kind of difficulty, we observe that the name "sinc methods" actually describes algorithms combining two components: (i) the use of sinc function approximations (Fourier interpolants) on the real axis, and (ii) a conformal mapping such as $\varphi^{-1}(s)$, or $\tanh(s)$, from the real axis $\mathbb{R}$ to a finite interval $[a, b]$. Our study has highlighted complications in both of these steps.

Regarding (i), sinc function approximations, we do not see a good reason to use sinc functions at all. The crucial aspect of our experience that leads to this conclusion is the discovery that the best procedure for representing a function $f(x)$ on $[a, b]$ is to map it once and for all to a function $F(s)$ on an appropriate interval $[s_{\text{left}}, s_{\text{right}}]$, and then leave $[s_{\text{left}}, s_{\text{right}}]$ fixed while refining the grid within. In other words, the famous $\sqrt{N}$ balance at the heart of standard sinc methods theory between grid spacing errors and grid extent errors is best treated asymmetrically in practice: fix the grid extent, then adjust the grid spacing. Once you do this, you are working simply with a function F on a fixed interval $[s_{\text{left}}, s_{\text{right}}]$. As has been noted previously [4, 9], the key to the method is the mapping, not the choice of basis. Sincfun faithfully uses sinc functions to represent F, but for this to work it has to subtract a linear function to impose zeros at the ends and rely on the exponential decay, so that F can be regarded as numerically periodic. All this seems weakly motivated; why not simply use a Chebyshev representation on $[s_{\text{left}}, s_{\text{right}}]$? Then no zeros or periodicity are needed, and the cost is in principle no more than a factor of $\pi/2$ in the number of samples [5, 13]. Moreover, as discussed in section 3, for efficient rootfinding one wants to subdivide the interval recursively anyway, and then one is pressed all the more strongly to use Chebyshev technology. For a related approach to constructing spectral methods on unbounded domains, see [7, 10].

Regarding (ii), the conformal maps, we think that the usual $\varphi^{-1}(s)$ and $\tanh(s)$ transforms are wasteful and can be improved. Experience shows that Sincfun requires hundreds of samples to represent virtually any function, whereas Chebfun often succeeds with tens of samples. This may seem like the inevitable price of a move from $\mathcal{O}(C^{-N})$ to $\mathcal{O}(C^{-\sqrt{N}})$ convergence, but we note that over and over again Sincfun seems to use more sample points than "ought" to be necessary. Typically 80% of the points fall in the edge regions, as is evident in Figure 3.1. Moreover, often surprisingly many points per wavelength are used, even in the interior regions. These effects are the consequences of the use of a particular conformal map, which is not the only possible choice. We think this situation can be improved by the use of alternative variable transforms based on a different balance between resolution in the interior and resolution near the endpoints, and we will pursue this idea in future research.

Acknowledgments. The authors would like to thank Andre Weideman and Sheehan Olver for valuable discussions about aspects of this paper.

REFERENCES

[1] Z. Battles and L. N. Trefethen, *An extension of MATLAB to continuous functions and operators*, SIAM J. Sci. Comput., 25 (2004), pp. 1743–1770.

[2] J.-P. Berrut, *Barycentric formulae for cardinal (SINC-)interpolants*, Numer. Math., 54 (1989), pp. 703–718.

[3] J.-P. Berrut and L. N. Trefethen, *Barycentric Lagrange interpolation*, SIAM Rev., 46 (2004), pp. 501–517.

[4] J. P. Boyd, *Polynomial series versus sinc expansions for functions with corner or endpoint singularities*, J. Comput. Phys., 64 (1986), pp. 266–269.

[5] J. P. Boyd, *Chebyshev domain truncation is inferior to Fourier domain truncation for solving problems on an infinite interval*, J. Sci. Comput., 3 (1988), pp. 109–120.

[6] J. P. Boyd, *The asymptotic Chebyshev coefficients for functions with logarithmic endpoint singularities*, Appl. Math. Comput., 29 (1989), pp. 49–67.
[7] J. P. Boyd, *The rate of convergence of Fourier coefficients for entire functions of infinite order with application to the Weideman-Cloot sinh-mapping for pseudospectral computations on an infinite interval*, J. Comput. Phys., 110 (1994), pp. 360–372.
[8] J. P. Boyd, *Computing zeros on a real interval through Chebyshev expansion and polynomial rootfinding*, SIAM J. Numer. Anal., 40 (2002), pp. 1666–1682.
[9] J. P. Boyd, *Chebyshev and Fourier Spectral Methods*, 2nd ed., Dover, Mineola, NY, 2001.
[10] A. Cloot and J. A. C. Weideman, *An adaptive algorithm for spectral computations on unbounded domains*, J. Comput. Phys., 102 (1992), pp. 398–406.
[11] W. Gautschi, *Remark: Barycentric formulae for cardinal (SINC-)interpolants*, Numer. Math., 87 (2001), pp. 791–792.
[12] I. J. Good, *The colleague matrix, a Chebyshev analogue of the companion matrix*, Quart. J. Math., 12 (1961), pp. 61–68.
[13] N. Hale and L. N. Trefethen, *New quadrature formulas from conformal maps*, SIAM J. Numer. Anal., 46 (2008), pp. 930–948.
[14] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed., SIAM, Philadelphia, 2002.
[15] M. L. Overton, *Numerical Computing with IEEE Floating Point Arithmetic*, SIAM, Philadelphia, 2001.
[16] R. Pachón, R. Platte, and L. N. Trefethen, *Piecewise smooth chebfuns*, IMA J. Numer. Anal., 30 (2010), pp. 898–916.
[17] W. Specht, *Die Lage der Nullstellen eines Polynoms*. IV, Math. Nachr., 21 (1960), pp. 201–222.
[18] F. Stenger, *Numerical Methods Based on Sinc and Analytic Functions*, Springer, New York, 1993.
[19] F. Stenger, *Handbook of Sinc Numerical Methods*, CRC Press, Boca Raton, FL, 2010.
[20] L. N. Trefethen, *Spectral Methods in MATLAB*, SIAM, Philadelphia, 2000.
[21] L. N. Trefethen, *Approximation Theory and Approximation Practice*, manuscript in preparation.
[22] L. N. Trefethen et al., *Chebfun software package*, www.maths.ox.ac.uk/chebfun/.
[23] L. N. Trefethen et al., *Chebfun Guide*, www.maths.ox.ac.uk/chebfun/guide/.