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, spectral method

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

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 [16].

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 [17, chap. 8]. 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. Another is the use of variable transformations, particularly for square root singularities. A third is the automatic subdivision of an interval in "splitting on" mode, allowing representation of very general singularities at some cost in speed and smoothness. Nevertheless, despite these features, Chebfun has nothing like the speed and reliability for functions with singularities that it enjoys for smooth ones. For example, Chebfun can construct short representations of $f(x) = x^{0.3}$ and $g(x) = x^{0.5}$ on $[0, 1]$, but the computation of $f + g$ is problematic.

There is a well-known technology for representing functions with quite general endpoint singularities: expansions in sinc functions. Over the years a number of people have advocated these methods, especially Frank Stenger of the University of Utah, who has written two books on the subject [11, 12], the second serving as a user's guide for the Sinc-Pack Matlab package. In principle, sinc expansions offer 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

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

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

"classic" Chebfun of Battles and Trefethen for the interval $[-1,1]$, 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.
```

The aim of this paper 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 lower case letters for the former and upper case for the latter: $g(x)$ and $G(s)$. A typewriter font such as g labels computer variables such as the Sincfun representation of g.

Following [11] and [12], 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 [14].

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 [14].

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

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

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

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

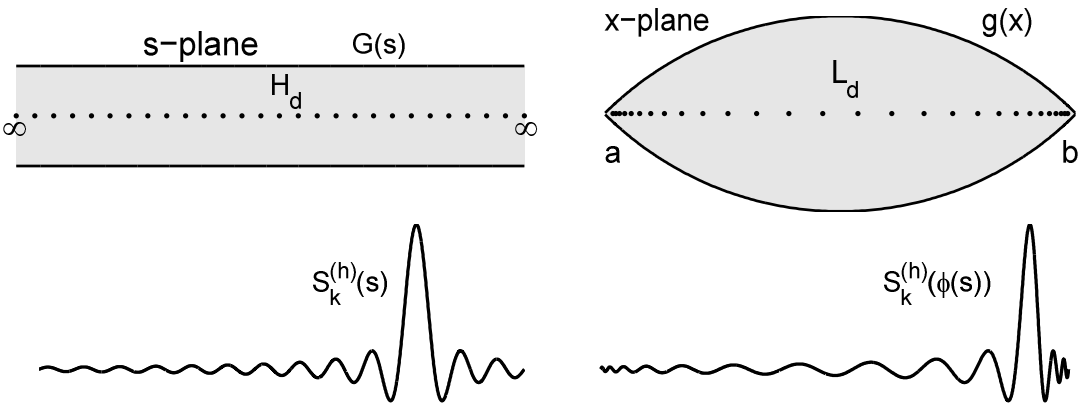

FIG. 2.1. *The function φ^{-1} maps the infinite strip H_d in the s-plane onto the lens-shaped region L_d in the x-plane. Below, a sinc function on the regular grid in $\mathbb{R}$ and its image on the mapped grid in (a, b).*

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 $C_4 > 0$.

Proof. See [11] pp. 137–138 or [12] pp. 26–29. □

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 always 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 end points

These data, together with equations (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 & 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 non-negligible 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: it has happened since f is nonsingular 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.

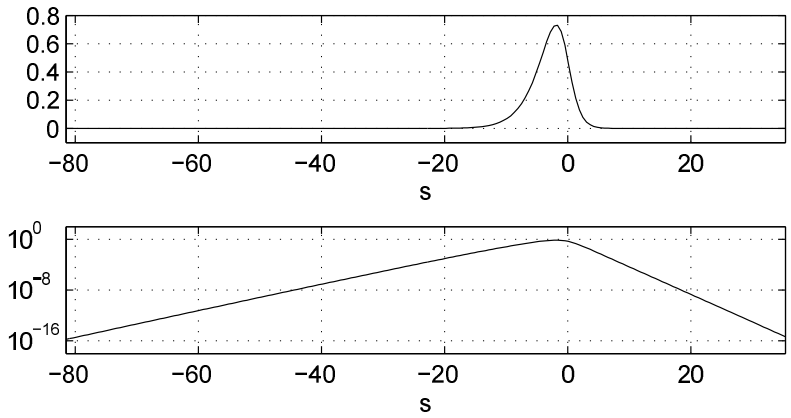

FIG. 3.1. *Example function $F(s)$ corresponding to $f(x) = -\sqrt{x}\log x$, shown on linear and log scales. In the first stage of the construction process, the interval $[s_{left}, s_{right}]$ is determined on which $F(s)$ is not negligible.*

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, 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 straightforward way of doing this is to evaluate each of the sinc functions $S_k^{(h)}(\varphi(x))$, then multiply by the function coefficients $g_k = g(\varphi^{-1}(kh))$, and sum over k. This is the method used in Sinc-Pack [12] and it involves the evaluation of $|m| + n + 1$ sine functions. Computations for vector valued x follow from using the appropriate Matlab component-wise vector operators.

Following the analogous technique for evaluating polynomials [2], Sincfun uses a barycentric representation to compute (2.8). First documented by Berrut [3], the barycentric formula for sinc functions allows the expansion to be computed without

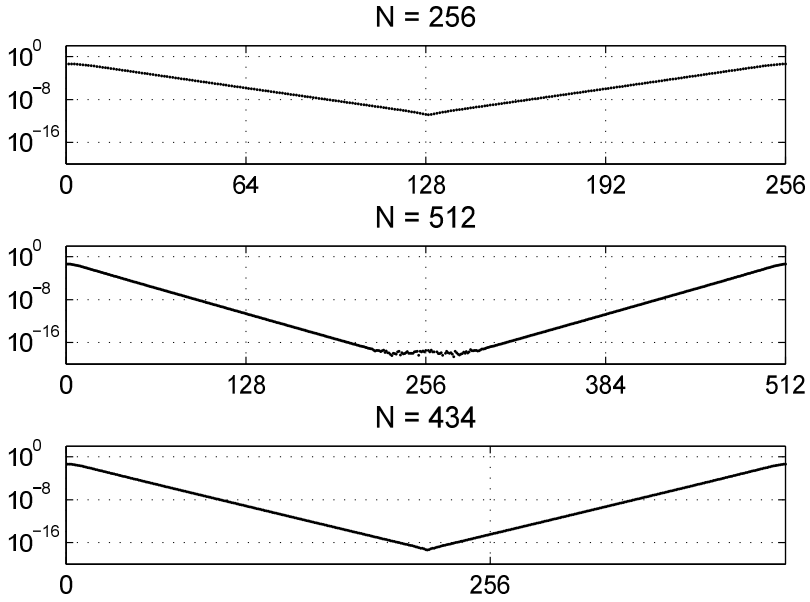

FIG. 3.2. *Fourier coefficients of the function restricted to this interval. In the second stage of the construction, a grid number is determined that resolves the function to machine precision.*

the need to evaluate any sine functions. We have found that using a barycentric representation leads to a saving in computational cost of approximately a factor of 4, compared to evaluating (2.8) directly.

Specifically, Sincfun uses the following *weighted barycentric formula*,

$$g_w^{(h,m,n)}(x) = w(\varphi(x)) \frac{\sum_{k=m}^{n} \frac{(-1)^k}{s-kh} g_k}{\sum_{k=m}^{n} \frac{(-1)^k}{s-kh} w(kh)}, \tag{4.1}$$

where w is a function which must decay on $\mathbb{R}$, at a rate comparable to that of the transplanted function [3]. To accommodate the general case, Sincfun uses a Möbius transformation to scale a standard Gaussian to obtain a weight function with the required decay properties. This is achieved by setting $w(s) = e^{-\mathcal{M}(s)^2}$, where

$$\mathcal{M}(s) = \frac{s}{p - qs}, \qquad p = \frac{s_{\text{left}}}{6}\left(\frac{s_{\text{left}} + s_{\text{right}}}{s_{\text{left}} - s_{\text{right}}} + 1\right), \qquad q = \frac{s_{\text{left}} + s_{\text{right}}}{6(s_{\text{left}} - s_{\text{right}})}.$$

A further method of evaluating the Sinc interpolant was suggested by Gautschi [5] and involves a simple re-ordering of (2.8). However, it is does not appear to be possible to vectorize this computation, since Gautschi's rearrangement requires a new index set to be computed for each evaluation. Thus for a Matlab-based system of evaluating sinc expansions, (4.1) appears to be the best choice.

Once evaluation has been properly defined, 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 operation `op(f)` is well defined computationally. This process is called composition, and the output is a further sincfun defined over the domain of `f`.

As in Chebfun, composition has been implemented by overloading many of 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`. If this is the case, then the barycentric formula is used twice in succession to evaluate first `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 [10]. 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 [6].

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 re-approximate $F(s)$ using an expansion in Chebyshev polynomials and work with the corresponding colleague matrix. Two observations motivate this.

Firstly, 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. Secondly, 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 proposed by Boyd [4] and now also used in Chebfun. The process involves sampling $G(s)$ over the interval $[s_{\text{left}}, s_{\text{right}}]$, at a set of 100 scaled Chebyshev points which are then transformed into Chebyshev coefficients using an algorithm based on the FFT. If these 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 then 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 repeated on each sub-interval. 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 of the eigenvalue computations are combined to give the roots over the entire interval.

A further computational saving can be achieved by exploiting the exponential decay. Instead of sampling F over the whole of $[s_{\text{left}}, s_{\text{right}}]$, it suffices to focus in on the middle part of the curve, where all the important oscillatory behavior is located. To achieve this, Sincfun searches the vector of numbers stored in the `f.vals` field in order to identify the first and last sign changes in the equispaced function values. For an illustration of this, see Figure 5.1.

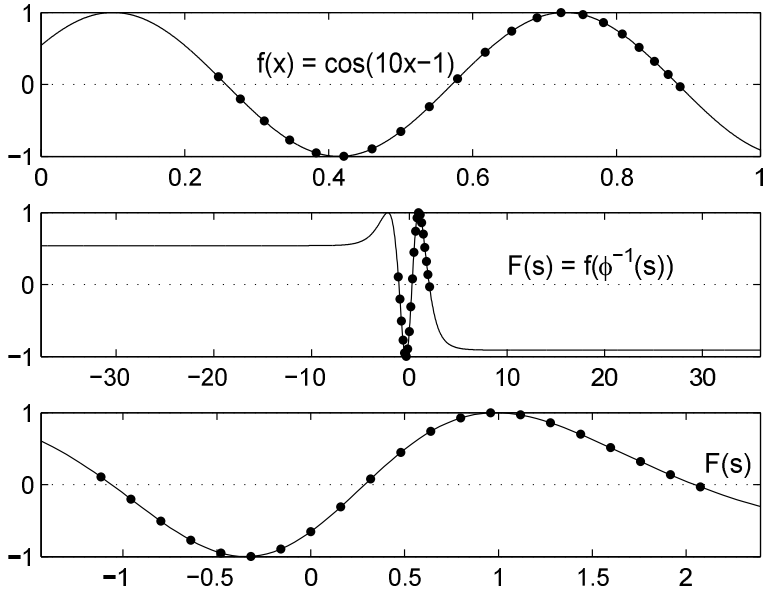

FIG. 5.1. *When computing roots, Sincfun normally samples only the middle part of the transplanted function $F(s)$. Above: A simple oscillatory function on $[0,1]$. Middle: The transplanted function plotted in $[s_{left}, s_{right}]$. Below: Sincfun zooms in on the oscillatory part of the curve by detecting the first and last sign changes in the* `f.vals` *Sincfun field. F is then sampled over this sub-interval, rather than over the whole of $[s_{left}, s_{right}]$. The dots are the corresponding subset of function values at the points $\varphi^{-1}(kh)$.*

If no sign change is detected in the `f.vals` vector, we assume that f itself does not change sign over $[a, b]$, so in this case no eigenvalue computation is necessary. It is possible however, that f has roots at the interval endpoints that do not get detected by looking for sign changes in the function values. To avoid this problem, Sincfun performs a manual check to see if $f(a) = 0$ or $f(b) = 0$. Any further roots identified in this step are then appended to the output vector.

Algorithms based upon recursive subdivision often lead to drastically reduced complexity of the rootfinding problem, typically down from $\mathcal{O}(N^3)$ to $\mathcal{O}(N^2)$ [1, 4]. This is due to the fact that instead of solving a single large eigenvalue problem corresponding to a global interpolant, the computation is broken down into several smaller steps, each corresponding to a local interpolant. In Sincfun, we also obtain an $\mathcal{O}(N^2)$ method, and the sign-change heuristic contributes a significant saving for lower degree sincfuns. Figure 5.2 shows the timings of our algorithm both with and

without the heuristic being applied.[1]

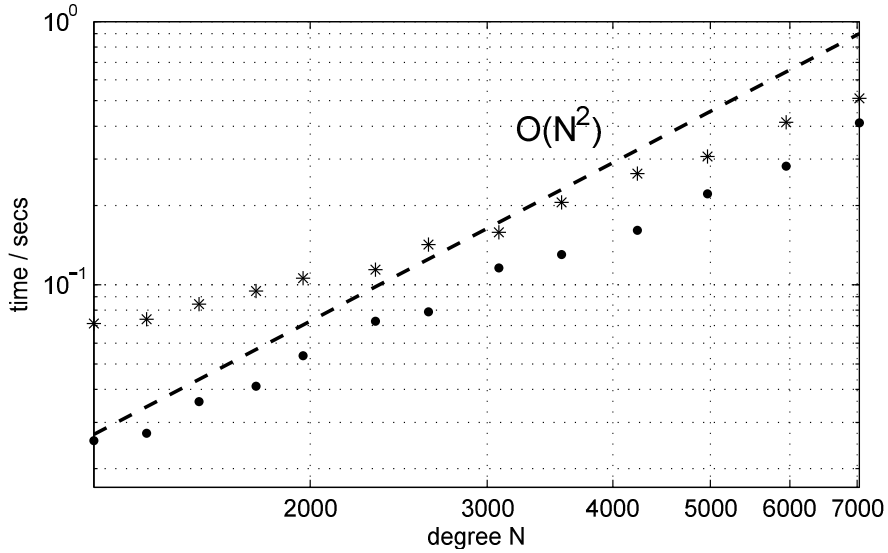

FIG. 5.2. *Timings for the rootfinding algorithm applied to the Sincfun representation of the function* $f_7(x) = \sqrt{x}\cos(k\pi x)$ *on* $[0,1]$, *where* k *takes successive values from the set* $\{15, 19, 23, 29, 35, 45, 55, 67, 83, 103, 127, 157, 193\}$, *and* $N = |m| + n + 1$. *The dots correspond to the heuristic being applied, whilst the stars correspond to sampling over the entire* $[s_{left}, s_{right}]$ *interval. In each case, the convergence is at least* $\mathcal{O}(N^2)$.

Since the eigenvalues of a colleague matrix are exactly equal to the roots of the corresponding finite Chebyshev series, the error 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 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.

	Sincfun length	time(s)	$\|\|\cdot\|\|_\infty$ error
$f_8(x) = \sin(4\pi x)$	496	0.010285	8.33×10^{-16}
$f_9(x) = \sin(40\pi x)$	1659	0.051842	2.28×10^{-15}
$f_{10}(x) = \sin(400\pi x)$	10771	1.172530	4.44×10^{-16}

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

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 [11], 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

[1]Users may switch off this feature with the command `sincfunpref('rtsheur',false)`.

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 attractive feature 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 pre-computed 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, and in general, the results of such computations are at least as accurate as the interpolant [15]. 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 $C_6 > 0$ 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 [11], p. 190.

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

Integrating sincfuns, then, is reasonably straightforward. Differentiating them, however, is far more problematic. To see why, consider the derivative of (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 the derivative of $g(x)$ on $[a, b]$ consists of the derivative of the transplanted function $G(s)$ on $\mathbb{R}$ multiplied by the derivative of the transplanting map, φ. Inevitably, the $\varphi'(x)$ term diverges to infinity as x approaches the endpoints of $[a, b]$. When this is multiplied by any errors in the derivative of $G(s)$ as in (6.4), significant accuracy near $x = a, b$ is lost.

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

TABLE 6.1
Approximation errors for definite integral computations in Sincfun.

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
```

If we wish to, say, find the integral of the difference between these functions, this is not a problem:

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

Similarly, the roots of the difference between the Sincfuns will give us the intersections. We can perform this computation and plot the results with just a couple of commands:

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

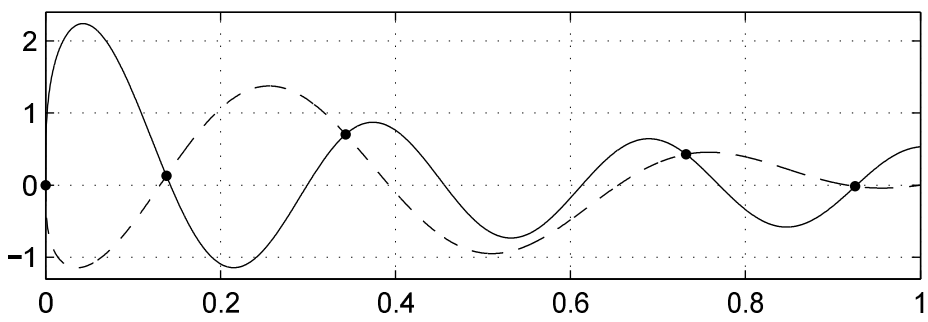

FIG. 7.1. *Plot of the sincfuns* `ff` *and* `gg`*, together with their intersections.*

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 it is possible to recover $\mathcal{O}(C^{-N})$ convergence, of a kind, by computing in advance the interval $[s_{\text{left}}, s_{\text{right}}]$, outside of which $G(s)$ is negligible. This is a direct consequence of the fact that our computations are being performed in finite precision. We confirm this effect in Figure 7.2, which shows the approximation error of the Sincfun representations of four functions. Clear exponential convergence is shown, down to the level of the unit roundoff in IEEE double precision.

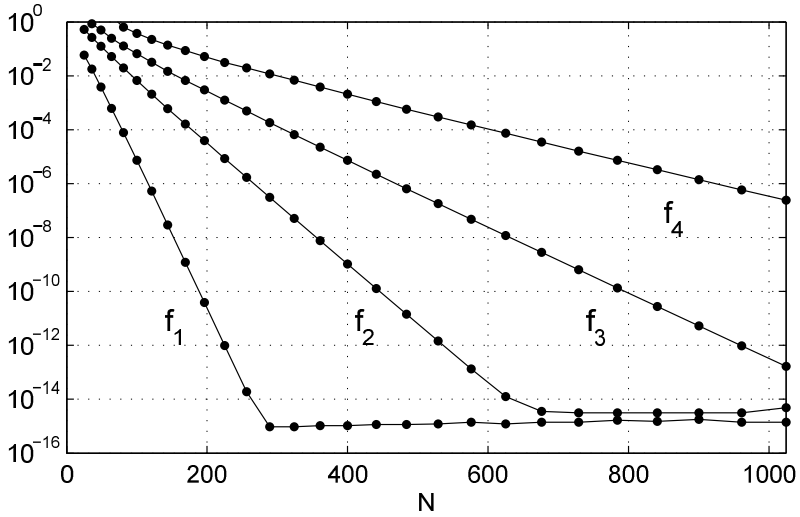

FIG. 7.2. $||f_j - f_j^{(h,m,n)}||$ *for;* $f_1(x) = x\log(x)$, $f_2(x) = x^{1/4}\log(x)$, $f_3(x) = x^{1/10}\log(x)$, $f_4(x) = x^{1/20}\log(x)$*; on* $[0,1]$, *where* $N = |m| + n + 1$. *The* ∞*-norm error was estimated by sampling* f_j *and* $f_j^{(h,m,n)}$ *for each* $j = 1,2,3,4$ *at* 2000 *equally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

Table 7.1 gives the $[s_{\text{left}}, s_{\text{right}}]$ interval, the corresponding $[x_{\text{left}}, x_{\text{right}}]$ interval, and the number of terms m, n in (2.8) for the same four functions as in Figure 7.2. These parameters resulted from the adaptive Fourier discretization process described in Section 3. The s_{left}, s_{right}, and x_{left} values are approximate.

	s_{left}	s_{right}	x_{left}	x_{right}	m	n
$f_1(x) = x\log(x)$	-41	36	2.5×10^{-18}	$1 - \frac{3}{2}\varepsilon_{\text{mach}}$	-162	143
$f_2(x) = x^{1/4}\log(x)$	-160	35	1.7×10^{-71}	$1 - 3\varepsilon_{\text{mach}}$	-568	122
$f_3(x) = x^{1/8}\log(x)$	-330	34	3.4×10^{-142}	$1 - 5\varepsilon_{\text{mach}}$	-1069	113
$f_4(x) = x^{1/20}\log(x)$	-710	34	2.4×10^{-308}	$1 - 9\varepsilon_{\text{mach}}$	-2052	98

TABLE 7.1

Some key properties of the Sincfun representations of the functions from Figure 7.2, where $s_{\text{left}} = \varphi^{-1}(s_{\text{left}})$, $s_{\text{right}} = \varphi^{-1}(s_{\text{right}})$, *and* $\epsilon_{\text{mach}} = 2^{-52} \approx 2.2 \times 10^{-16}$.

In general, stronger singularities correspond to larger $[s_{\text{left}}, s_{\text{right}}]$ intervals. In Table 7.1, we see that the s_{right} values remain roughly constant, whilst 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 a sinc approximation is required to be uniformly accurate to within a relative error of 10^{-15}. The strength of the singularity

at $x = a$ is constrained by the location of the nearest floating point number to $x = a$ in the interval $(a, b]$, and this limit is dependent upon the computational precision. For example with $a = 0$, in IEEE double precision, the smallest normalized floating point number is $2^{-1022} \approx 2.2 \times 10^{-308}$ [9]. 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 x_{left} value in Table 7.1 is close to 2.2×10^{-308}. To machine precision, it would not be possible to approximate a function with an even severer singularity than this, such as $f_5(x) = x^{1/30} \log(x)$, since $|f_5(2.2 \times 10^{-308})| \approx 10^{-8}$, a value several orders of magnitude larger than $\varepsilon_{\text{mach}} \approx 2.2 \times 10^{-16}$.

A further consequence of this observation is that sinc approximation is only fully effective if the singularity is located at $x = 0$. To see why, consider the function $f_6(x) = \sqrt{1 - x}$, which has a square-root singularity at $x = 1$. The closest floating point number to 1 in double precision is $1 - 2^{-53} = 1 - \varepsilon_{\text{mach}}/2$, and as before, $f_6(1 - \varepsilon_{\text{mach}}/2) \approx 10^{-8} \gg \varepsilon_{\text{mach}}$. Thus at locations other than $x = 0$, it is only possible to represent functions with relatively weak singularities.

For the most part, composition of sincfuns with mathematical functions 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 function applied is singular, such as `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. This is another way of saying that 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!
```

However, it is perfectly possible to obtain the desired representation if the sincfun is constructed by sampling $\sqrt{\sin(x)}$ directly. Doing so ensures that the constructor is able to sample the function near to $x = 0$.

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.

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 steps: (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 difficulties 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 $[s_{\text{left}}, s_{\text{right}}]$. 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. The cost is in principle no more than a factor of $\pi/2$ in the number of samples [7]. 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.

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; often surprisingly many points per wavelength are used even in the interior regions, as is evident in Figure 5.1. 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 maps based on a different balance between resolution in the interior and resolution near the endpoints, and we will pursue this idea in future research.

Acknowledgements. The authors would like to thank Andre Weideman and Sheehan Olver for valuable discussions about this paper. This work was supported by the Engineering and Physical Sciences Research Council (UK), EP/E045847.

REFERENCES

[1] Z. Battles and L. N. Trefethen, An extension of Matlab to continuous functions and operators, *SIAM J. Sci. Comp.* 25 (2004), 1743–1770.
[2] J.-P. Berrut and L.N. Trefethen, Barycentric Lagrange interpolation, *SIAM Review* 46 (2004), 501–517.
[3] J.-P. Berrut, Barycentric formulae for cardinal (SINC-)interpolants, *Numer. Math.* 54 (1989), 703–718.
[4] J. P. Boyd, Computing zeros on a real interval through Chebyshev expansion and polynomial rootfinding, *SIAM J. Numer. Anal.* 40 (2002), 1666–1682.
[5] W. Gautschi, Remark: Barycentric formulae for cardinal (SINC-)interpolants, *Numer. Math.* 87 (2001), 791–792.
[6] I. J. Good, The colleague matrix, a Chebyshev analogue of the companion matrix, *Quart. J. Math.* 12 (1961), 61–68.
[7] N. Hale and L.N. Trefethen, New quadrature formulas from conformal maps *SIAM J. Numer. Anal.* 46 (2008), 930–948.
[8] N.J. Higham, *Accuracy and Stability of Numerical Algorithms,* SIAM, 2nd edition, 2002.
[9] M. L. Overton, *Numerical Computing with IEEE Floating Point Arithmetic,* SIAM, 2001.
[10] W. Specht, Die Lage der Nullstellen eines Polynoms. IV, *Math. Nachr.* 21 (1960), 201–222.
[11] F. Stenger, *Numerical Methods Based on Sinc and Analytic Functions,* Springer, 1993.
[12] F. Stenger, *Handbook of Sinc Numerical Methods,* CRC Press, 2010.
[13] M. Sugihara and M. Takayasu, Recent developments of the Sinc numerical methods, *J. Comp. App. Math.* 164–165 (2004), 673–689.
[14] L. N. Trefethen, *Spectral Methods in Matlab,* SIAM, 2000.
[15] L. N. Trefethen, *Approximation Theory and Approximation Practice*, manuscript in preparation.
[16] L. N. Trefethen, et al., Chebfun software package, `www.maths.ox.ac.uk/chebfun/`.
[17] L. N. Trefethen, et al., Chebfun Guide, `www.maths.ox.ac.uk/chebfun/guide/`.