AN ADAPTIVE PARTITION OF UNITY METHOD FOR CHEBYSHEV POLYNOMIAL INTERPOLATION*

KEVIN W. AITON† AND TOBIN A. DRISCOLL†

Abstract. For a function that is analytic on and around an interval, Chebyshev polynomial interpolation provides spectral convergence. However, if the function has a singularity close to the interval, the rate of convergence is near one. In these cases splitting the interval and using piecewise interpolation can accelerate convergence. Chebfun includes a splitting mode that finds an optimal splitting through recursive bisection, but the result has no global smoothness unless conditions are imposed explicitly at the breakpoints. An alternative is to split the domain into overlapping intervals and use an infinitely smooth partition of unity to blend the local Chebyshev interpolants. A simple divide-and-conquer algorithm similar to Chebfun's splitting mode can be used to find an overlapping splitting adapted to features of the function. The algorithm implicitly constructs the partition of unity over the subdomains. This technique is applied to explicitly given functions as well as to the solutions of singularly perturbed boundary-value problems.

Key words. partition of unity, Chebyshev interpolation, Chebfun, overlapping domain decomposition

AMS subject classifications. 65L11, 65D05, 65D25

DOI. 10.1137/17M112052X

1. Introduction. Chebyshev polynomial interpolants provide powerful approximation properties, both in theory and as implemented in practice by the Chebfun software system [2]. Chebfun uses spectral collocation to provide very accurate automatic solutions to differential equations [4]. The method is not fully adaptive, though, since the refinement is limited to the degree of the global interpolant.

Chebfun includes a *splitting* method that creates piecewise polynomial approximations [10]. When splitting is enabled, if a Chebyshev interpolant is unable to represent the function accurately at a specified maximum degree on an interval, the interval is bisected; this process is recursively repeated on the subintervals. Afterwards adjacent subintervals are merged if the new interval allows for a Chebyshev approximation with lower degree. In effect, the method does a binary search for a good splitting location. In [6] it was shown that the splitting locations are roughly optimal based on the singularity structure of the function in the complex plane.

A drawback of Chebfun's splitting approach is that the resulting representation does not ensure anything more than C^0 continuity. Differentiation of the Chebyshev interpolation polynomial of degree n has norm $O(n^2)$, so a jump in the derivative develops across a splitting point and becomes more pronounced for higher derivatives and larger n. In order to solve a boundary-value problem (BVP), Chebfun imposes explicit continuity conditions on the solution to augment the discrete problem. This solution works well in 1D but becomes cumbersome in higher dimensions, particularly if refinements are made nonconformingly.

In this paper we explore the use of Chebyshev interpolants on overlapping do-

mains combined using a *partition of unity*. The resulting approximation has the same accuracy as the individual piecewise interpolants. We use compactly supported weight functions that are infinitely differentiable, so the resulting combined interpolant is also infinitely smooth (though not analytic). We also show that the accuracy of the derivative can be bounded by $\Theta(\delta^{-2})$ for an overlap amount δ, revealing an explicit tradeoff between efficiency (smaller overlap and more like Chebfun splitting) and global accuracy of the derivative. Because the global approximation is smooth, there are no matching conditions needed to solve a BVP, and there are standard preconditioners available that should aid with iterative methods for large discretizations. For example, since we split the interval into overlapping domains we could use the restricted additive Schwarz preconditioner [3].

We describe a recursive, adaptive algorithm for creating and applying a partition of unity, modeled on the recursive splitting in Chebfun but merging adjacent subdomains aggressively in order to keep the total node count low. Even though each node of the recursion only combines two adjacent subdomains, we show that the global approximant is also a partition of unity. We demonstrate that the adaptive refinement is able to resolve highly localized features of an explicitly given function and of a solution to a singularly perturbed BVP.

The use of a partition of unity in our approximation affords us some flexibility: we are able to create approximations which are both efficient and infinitely smooth without matching. Partition of unity schemes have been widely used for interpolation [7, 9, 13] and solving PDEs [8, 12]. In section 3 we introduce the partition of unity method, and we discuss the convergence of the method for a simple split on the interval $[-1, 1]$ in section 4. We describe our adaptive algorithm in section 5. In section 6 we explain how to apply our method to solve BVPs on an interval and perform some experiments with singularly perturbed problems.

2. Chebyshev interpolation. We use Chebyshev interpolants for our partition of unity method because they enjoy spectral convergence. Suppose that $f(x)$ is analytic inside a Bernstein ellipse E_ρ (an ellipse with foci ± 1 and semimajor axis $\rho > 1$). We then have Theorem 6 from [15].

THEOREM 1. *Suppose $f(z)$ is analytic on and inside the Bernstein ellipse E_ρ. Let p_n be the polynomial that interpolates $f(z)$ at $n + 1$ Chebyshev points of the second kind. Then there exists a constant $C > 0$ such that for all $n > 0$,*

$$\|f(x) - p_n(x)\|_\infty \le C\rho^{-n}.$$

If $f(x)$ is Lipschitz continuous on $[-1, 1]$, then

$$(1) \qquad f(x) = \sum_{k=0}^{\infty} a_k T_k(x), \quad a_k = \frac{2}{\pi} \int_{-1}^{1} \frac{f(x)T_k(x)}{\sqrt{1 - x^2}} dx,$$

where T_k denotes the degree k Chebyshev polynomial (and for a_0 we multiply by $\frac{1}{\pi}$ instead of $\frac{2}{\pi}$). Furthermore if $p_n(x)$ is the nth degree Chebyshev interpolant, then

$$(2) \qquad f(x) - p_n(x) = \sum_{k=n+1}^{\infty} a_k \left(T_k(x) - T_m(x)\right),$$

where

$$(3) \qquad m = [(k + n - 1)(\text{mod } 2n) - (n - 1)],$$

implying we can determine the accuracy of the interpolant $p_n(x)$ by inspecting the Chebyshev coefficients [16]. Chebfun's standardChop method determines the minimum required degree by searching for a plateau of low magnitude coefficients [1]. For example, Figure 1 shows the first 128 coefficients of $f(x) = \exp(\sin(\pi x))$. We see that all coefficients after the first 46 have magnitude less than 10^{-15}. In this case, Chebfun determines the ideal degree to be 50.
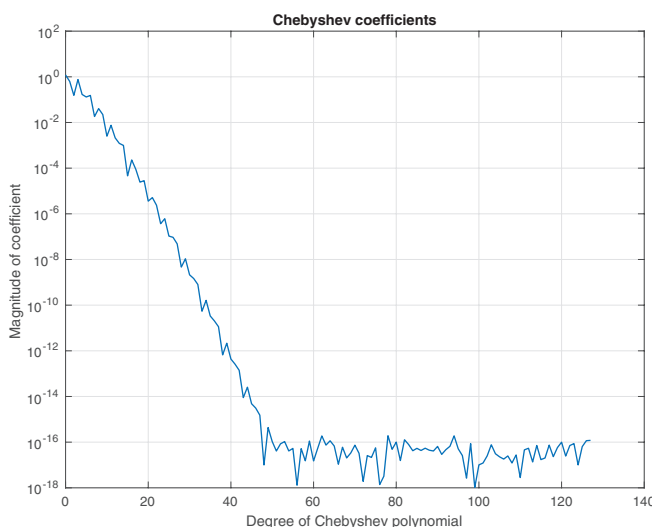

FIG. 1. *Chebyshev coefficients for $f(x) = \exp(\sin(\pi x))$.*

3. Partition of unity formalism. Suppose we have an overlapping covering $\{\Omega_k\}_{k=1}^N$ on a bounded region Ω. A partition of unity (PU) is a collection of real valued functions $\{w_k(x)\}_{k=1}^N$ such that

- $w_k(x)$ has support within Ω_k,
- each $w_k(x)$ is nonnegative,
- for all $x \in \Omega$, $\sum_{k=1}^N w_k(x) = 1$.

The functions $\{w_k(x)\}_{k=1}^N$ are called the *weights* of the partition. Suppose now that $\Omega = [-1, 1]$ and each Ω_k is an interval. We can use the partition of unity $\{w_k(x)\}_{k=1}^N$ to construct an approximating function. Suppose that for $m \geq 0$ we have a function $f \in C^m([-1, 1])$, each weight $w_k(x) \in C^m([-1, 1])$, and for each patch Ω_k we have an approximation $s_k(x)$ of $f(x)$. Then the function

$$(4) \qquad s(x) = \sum_{k=1}^N w_k(x) s_k(x)$$

can be used to approximate $f(x)$ and its derivatives [17].

THEOREM 2. *Suppose $f \in C^m([-1, 1])$ and for each patch Ω_k we have a function $s_k(x)$ such that*

$$\|f^{(\alpha)}(x) - s_k^{(\alpha)}(x)\|_{L_\infty(\Omega_k)} \leq \varepsilon_k(\alpha)$$

for $\alpha \leq m$. Thus for $j \leq m$, if $s(x)$ is the approximation (4), *then*

$$(5) \qquad \left\| f^{(j)}(x) - s^{(j)}(x) \right\|_{L_\infty(\Omega_k)} \leq \sum_{k=1}^{N} \sum_{i=0}^{j} \binom{j}{i} \left\| w_k^{(j-i)}(x) \right\|_{L_\infty(\Omega_k)} \epsilon_k(i).$$

Proof. Since $\sum_{k=1} w_k(x) = 1$, $\sum_{k=1} w_k(x)f(x) = f(x)$. Thus

$$
\begin{aligned}
(6) \quad \frac{d^j}{dx^j}f(x) - \frac{d^j}{dx^j}\sum_{k=1}^{N} w_k(x)s_k(x) &= \frac{d^j}{dx^j}\sum_{k=1}^{N} w_k(x)(f(x) - s_k(x)) \\
&= \sum_{k=1}^{N} \sum_{i=0}^{j} \binom{j}{i} w_k^{(j-i)}(x) \left(f^{(i)}(x) - s_k^{(i)}(x) \right).
\end{aligned}
$$

The result follows from here by the triangle inequality. $\qquad\square$

4. Convergence analysis. In this section we consider a single interval partitioned into two overlapping parts, i.e., $[-1, t],[-t, 1]$, where t is the overlap parameter such that $0 < t < 1$. For the weights, we use Shepard's method [13] based on the compactly supported, infinitely differentiable shape function

$$(7) \qquad \psi(x) = \begin{cases} \exp\left(1 - \frac{1}{1-x^2}\right), & |x| < 1, \\ 0, & |x| \geq 1. \end{cases}$$

We define support functions

$$(8) \qquad \psi_\ell(x) = \psi\left(\frac{x+1}{1+t}\right) \quad \text{and} \quad \psi_r(x) = \psi\left(\frac{x-1}{1+t}\right)$$

to construct the PU weight functions

$$(9) \qquad w_\ell(x) = \frac{\psi_\ell(x)}{\psi_\ell(x) + \psi_r(x)} \quad \text{and} \quad w_r(x) = \frac{\psi_r(x)}{\psi_\ell(x) + \psi_r(x)},$$

where $w_\ell(x), w_r(x)$ have support on the left and right intervals, respectively.

Suppose that $s_\ell(x), s_r(x)$ approximate $f(x)$ on $[-1, t]$, $[-t, 1]$, respectively, and are both infinitely smooth. Let

$$(10) \qquad s(x) = w_\ell(x)s_\ell(x) + w_r(x)s_r(x),$$

where $s(x)$ is the PU approximation. Following Theorem 2 we have for $x \in [-1, 1]$ that

$$
\begin{aligned}
(11) \quad |f(x) - s(x)| &= |w_\ell(x)\left(f(x) - s_\ell(x)\right) + w_r(x)\left(f(x) - s_r(x)\right)| \\
&\leq w_\ell(x)\left|f(x) - s_\ell(x)\right| + w_r(x)\left|f(x) - s_r(x)\right|.
\end{aligned}
$$

We conclude that

$$(12)$$
$$\|f(x) - s(x)\|_{L_\infty[-1,1]} \leq \max\left(\|f(x) - s_\ell(x)\|_{L_\infty[-1,t]}, \|f(x) - s_r(x)\|_{L_\infty[-t,1]} \right).$$

This implies that the partition of unity method (PUM) preserves the accuracy of its local approximants. We also have that $s(x)$ is infinitely smooth. For the first derivative we have

$$
\begin{aligned}
(13) \quad |f'(x) - s'(x)| &\leq |w_\ell(x)\left(f'(x) - s_\ell'(x)\right)| + |w_r(x)\left(f'(x) - s_r'(x)\right)| \\
&\quad + |w_\ell'(x)\left(f(x) - s_\ell(x)\right)| + |w_r'(x)\left(f(x) - s_r(x)\right)|,
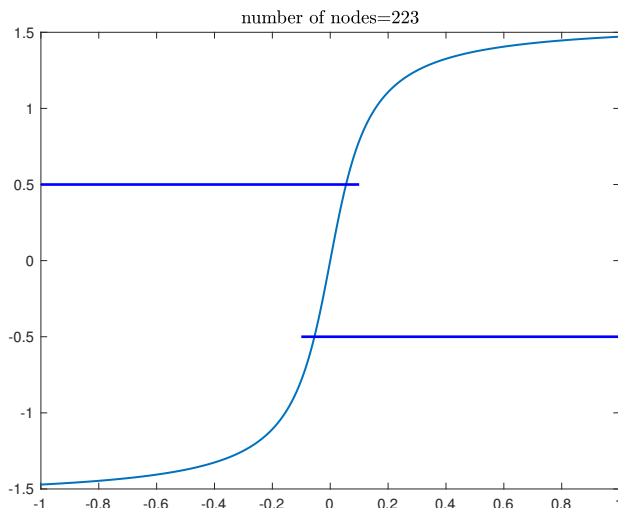\end{aligned}
$$

FIG. 2. *Plot of the PU approximation with overlap parameter $t = 0.1$ for $f(x) = \arctan(x/0.1)$, where the thick lines represent the domains of the left and right approximation. Here $\|f(x) - s(x)\|_{L_\infty[-1,1]} = 2.4\text{e-}15$ and $\|f'(x) - s'(x)\|_{L_\infty[-1,1]} = 1.7\text{e-}13$.*

giving us
(14)
$$\|f'(x) - s'(x)\|_{L_\infty[-1,1]} \le \max\left(\|f'(x) - s'_\ell(x)\|_{L_\infty[-1,t]}, \|f'(x) - s'_r(x)\|_{L_\infty[-t,1]}\right)$$
$$+ \|w'_\ell(x)\|_{L_\infty[-t,t]} \|f(x) - s_\ell(x)\|_{L_\infty[-t,t]}$$
$$+ \|w'_r(x)\|_{L_\infty[-t,t]} \|f(x) - s_r(x)\|_{L_\infty[-t,t]},$$

since the derivatives of the weights have support only on the overlap. For $t \ll 1$, the weights steepen to become nearly step functions. This causes the derivatives of the weights to be large in magnitude, resulting in an increase in the error for the derivative.

Since $w'_\ell(x) = -w'_r(x)$, from (14) we can infer
(15)
$$\|f'(x) - s'(x)\|_{L_\infty[-1,1]} \le \max\left(\|f'(x) - s'_\ell(x)\|_{L_\infty[-1,t]}, \|f'(x) - s'_r(x)\|_{L_\infty[-t,1]}\right)$$
$$+ \|w'_\ell(x)\|_{L_\infty[-t,t]} \max\left(\|f(x) - s_\ell(x)\|_{L_\infty[-t,t]}, \|f(x) - s_r(x)\|_{L_\infty[-t,t]}\right).$$

We have that $x = 0$ is a critical point of $w'_\ell(x)$, and for $t < 0.4$ it can be shown that the maximum of $|w'_\ell(x)|$ occurs at $x = 0$. Since

(16)
$$w'_\ell(0) = -\frac{(1+t)^2}{t^2(2+t)^2},$$

we can infer that $\|w'_\ell(x)\|_{L_\infty[-t,t]} = \Theta(t^{-2})$ as $t \to 0$. The norm of the Chebyshev differentiation operator is $\Theta(n^2)$ (for n nodes), implying that the two terms on the right-hand side of (15) are balanced if $t^{-2} = \Theta(n^2)$, or equivalently $t = \Theta\left(\frac{1}{n}\right)$. A simple example of a split can be seen in Figure 2.

5. Recursive algorithm. In order to allow for adaptation to specific features of $f(x)$, we next describe a recursive bisection algorithm that works similarly to

Chebfun's splitting algorithm [6] and to that of [14]. Suppose we want to construct a PU approximation $s_{[a,b]}(x)$ on the interval $[a,b]$ using Chebyshev interpolants on the patches. If $f(x)$ can be resolved by a Chebyshev interpolant $s(x)$ of length $n_{\max}$ on $[a,b]$, then

$$(17) \qquad s_{[a,b]}(x) = s(x).$$

Otherwise we split the interval into two overlapping domains and blend the results as in (10):

$$(18) \qquad s_{[a,b]}(x) = w_\ell(x)s_{[a,a+\delta]}(x) + w_r(x)s_{[b-\delta,b]}(x),$$

where w_ℓ, w_r are the PU weight functions defined in (9) (but defined for $[a,b]$, and $\delta = (1+t)\left(\frac{a+b}{2}\right)$).

We define a binary tree T with each node ν having the following properties:
- interval(ν) := the domain of the patch.
- $c_0(\nu)$,$c_1(\nu)$:= respective left and right subtrees of ν (if split).
- $w_0(\nu)$,$w_1(\nu)$:= respective left and right weights of ν (if split).
- interpolant(ν) := Chebyshev interpolant on interval(ν) if ν is a leaf.
- values(ν) := values of the function we are approximating at the Chebyshev points of ν.

We define root(T) as the root node of T. In Algorithm 1 we formally describe how we refine our splitting; the merge method is described in section 5.1.

We first initialize the tree T with a single node ν where interval(ν) = $[a,b]$. Next we repeatedly call the splitleaves method until each leaf of T has a Chebyshev interpolant that can resolve $f(x)$ with degree less than $n_{\max}$, as seen in Algorithm 2. For each leaf ν of T, sample(T,$f(x)$) sets values(ν) using $f(x)$. For a leaf ν, we determine if a Chebyshev interpolant can resolve $f(x)$ using Chebfun's standardChop method with values(ν) (as described in section 2). Using T we can evaluate $s_{[a,b]}(x)$ recursively as demonstrated in Algorithm 3.

As a simple example, we approximate the function $f(x) = \arctan\left(\frac{x-0.25}{0.001}\right)$ with $n_{\max} = 128$. In order to resolve to machine precision, a global Chebyshev interpolant on the interval $[-1,1]$ requires 25743 nodes, while our method requires 523. Chebfun with nonoverlapping splitting requires 381 nodes. Overlapping splittings will typically require more total nodes while offering the benefit of global smoothness. The result can be seen in Figure 3.

We can deduce from (12) that $s_{[a,b]}(x)$ will approximate $f(x)$. Moreover, our method implicitly creates a PU on the leaves of the tree through the product of the weights at each level.

THEOREM 3. *Let an approximation $s_{[a,b]}(x)$ be as in (10). Then the tree that represents $s_{[a,b]}(x)$ implicitly defines a PU $\{w_k(x)\}_{k=1}^{M}$, where $w_k(x)$ has compact support over the kth leaf.*

Proof. Suppose that on the domain $[a,b]$ we have PUs $\{w_{\ell k}(x)\}_{k=1}^{M_\ell}$, $\{w_{rk}(x)\}_{k=1}^{M_r}$ for the leaves of the left and right child, respectively. We claim that

$$(19) \qquad \{w_\ell(x)w_{\ell k}(x)\}_{k=1}^{M_\ell} \cup \{w_r(x)w_{rk}(x)\}_{k=1}^{M_r}$$

forms a PU over the leaves of the tree. We first observe that $w_\ell(x)w_{\ell k}(x)$ will have support in supp $(w_{1k}(x))$, the domain of the respective leaf. This is similarly true for $w_r(x)w_{rk}(x)$.

Algorithm 1 splitleaves(ν,$n_{\max}$,t)

---

if ν is a leaf and $f(x)$ cannot be resolved by interpolant(ν) **then**
    Define new nodes ν_0, ν_1.
    $[a,b] :=$ interval(ν)
    $\delta := \frac{b-a}{2}(1+t)$
    interval(ν_0) $:= [a, a+\delta]$
    interval(ν_1) $:= [b-\delta, b]$
    for $k = 0, 1$ **do**
        $c_k(\nu) := \nu_k$
    end for
    $w_0(\nu)$,$w_1(\nu) :=$ weights in (9) defined for $[a, a+\delta]$,$[b-\delta, b]$
else if ν is a leaf and $f(x)$ can be resolved by a Chebyshev interpolant with degree
less than $n_{\max}$ **then**
    interpolant(ν) := minimum degree interpolant $f(x)$ can be resolved by
          as determined by Chebfun
else
    for $k = 0, 1$ **do**
        splitleaves($c_k(\nu)$,$n_{\max}$,t)
    end for
    merge(ν,$n_{\max}$)
end if

Algorithm 2 $T =$ refine($n_{\max}$,t,$f(x)$)

Define T as a tree with a single node.
while T has unresolved leaves **do**
    sample(T,$f(x)$)
    splitleaves(root(T),$n_{\max}$,t)
end while

Algorithm 3 $val =$ eval(ν,x)

---

if ν is a leaf **then**
    $p :=$ interpolant(ν)
    $val := p(x)$
else
    $val_0, val_1 := 0$
    $w_0 := w_0(\nu)$
    $w_1 := w_1(\nu)$
    for $k = 0, 1$ **do**
        if $x \in$ interval($c_k(\nu)$) **then**
            $val_k :=$ eval($c_k(\nu)$,x)
        end if
    end for
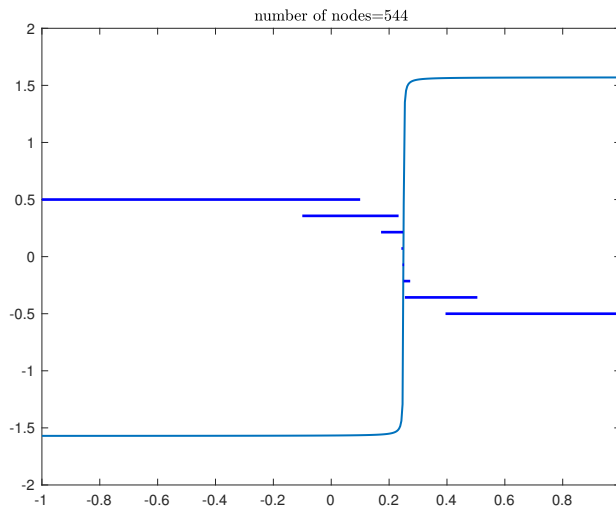 $val := w_0(x)val_0 + w_1(x)val_1$
end if

FIG. 3. *Plot of the PU approximation with overlap parameter $t = 0.1$ for $f(x) = \arctan\left((x - 0.25)/0.001\right)$, where the solid horizontal lines represent the patches.*

Next suppose that $x \in \sup\left(w_\ell(x)\right) \cap \sup\left(w_r(x)\right)^C$. Then $w_\ell(x) = 1$ and $w_r(x) = 0$, so

$$(20) \qquad \sum_{k=1}^{M_\ell} w_\ell(x) w_{\ell k}(x) + \sum_{k=1}^{M_r} w_r(x) w_{rk}(x) = \sum_{k=1}^{M_\ell} w_{\ell k}(x) = 1,$$

since $\{w_{\ell k}(x)\}_{k=1}^{M_\ell}$ is a PU. This is similarly true if $x \in \sup\left(w_\ell(x)\right)^C \cap \sup\left(w_r(x)\right)$. Finally if $x \in \sup\left(w_\ell(x)\right) \cap \sup\left(w_r(x)\right)$, then

$$(21) \qquad \sum_{k=1}^{M_\ell} w_\ell(x) w_{\ell k}(x) + \sum_{k=1}^{M_r} w_r(x) w_{rk}(x) = w_\ell(x) \sum_{k=1}^{M_\ell} w_{\ell k}(x) + w_r(x) \sum_{k=1}^{M_r} w_{rk}(x)$$
$$= w_\ell(x) + w_r(x) = 1.$$

Thus by induction we have that the product of weights through the binary tree for (10) implicitly creates a PU over the leaves. □

5.1. Merging. As we create the tree we opportunistically merge leaves for greater efficiency. If a particular location in the interval requires a great deal of refinement, the recursive splitting essentially performs a binary search for that location (as was noted about Chebfun splitting in [6]). The intermediate splits are not necessarily aiding with resolving the function; they are there just to keep the binary tree full. In Chebfun the recursive splitting phase is followed by a merging phase that discards counterproductive splits. We describe a similar merging operation here, but we allow these merges to take place whenever a leaf splits while its sibling does not, in order to keep the number of leaves from unnecessarily growing exponentially.

In Figure 4 we illustrate how we merge leaves; the interval $[a, b_1]$ is merged with $[a_2, b_{21}]$. Here we decide to merge if $f(x)$ can be resolved with an interpolant with degree less than $n_{\max}$ on the interval $[a, b_{21}]$. For the new tree we define the left
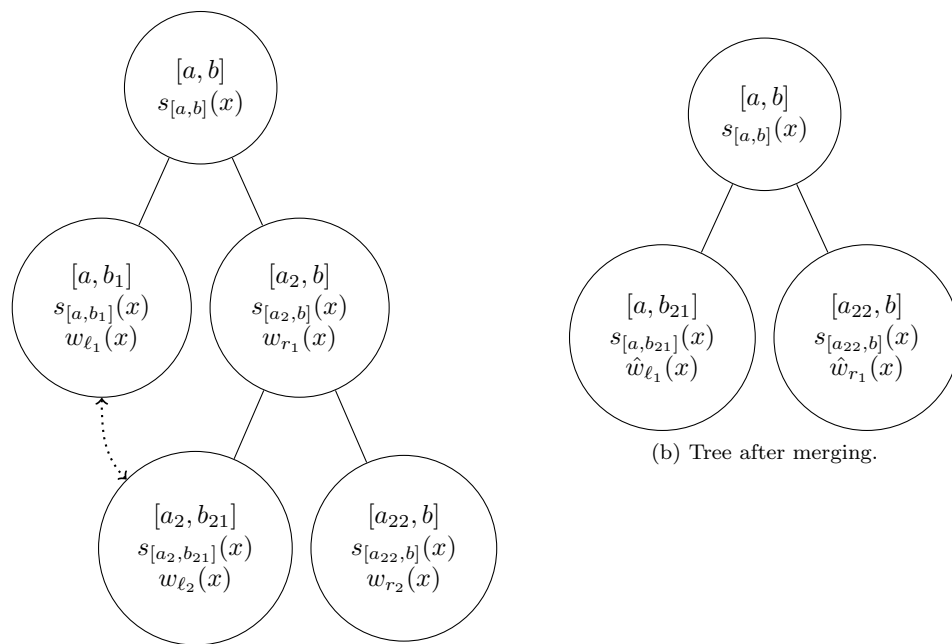
(a) Tree before merging. Here $a_2 < b_1$ and $a_{22} < b_{21}$.

(b) Tree after merging.

FIG. 4. *An example of how leaves are merged, where each node is labeled with its domain, PU approximation, and weight.*

weight $\hat{w}_{\ell_1}(x)$ in Figure 4 as

$$(22) \qquad \hat{w}_{\ell_1}(x) = \begin{cases} 1, & x < a_{22}, \\ w_{\ell_2}(x) & \text{otherwise.} \end{cases}$$

Since $w_{\ell_2}(x) = 1$ for $x < a_{22}$, $\hat{w}_{\ell_1}(x)$ is smooth. For the right weight we use $\hat{w}_{r_1}(x) = w_{r_2}(x)$; these new weights form a PU. The PU approximation

$$(23) \qquad \hat{s}(x) = w_{\ell 1}(x)s_{[a,b_1]}(x) + w_{r1}(x)s_{[a_2,b_{21}]}(x)$$

can be used to approximate $f(x)$ on $[a, b_{21}]$ since $f(x)$ is resolved at the leaves. In this case $s_{[a,b_{21}]}(x)$ is computed from sampling $\hat{s}(x)$. If the degree of $s_{[a,b_{21}]}(x)$ after Chebfun's chopping is less than $n_{\max}$, we decide to merge. We explain in more detail the merging in Algorithm 4; here extend($w(x),[a,b]$) piecewise extends the weight $w(x)$ in $[a,b]$ as in (22). We show the results for merging in Figure 5 with $f(x) = \frac{1}{x-1.001}$.

5.2. Differentiation matrices. Next we demonstrate how to construct a first derivative matrix; higher derivative matrices can be similarly constructed. Suppose we have constructed a splitting represented by the tree T. For each node ν of the tree, we add the following methods:

- points(ν) := provides the Chebyshev points of the leaves of ν.
- leafpoints(ν) := provides the Chebyshev points of T in interval(ν), i.e., points(root(T)) $\cap$ interval(ν).
- pointindex(ν) := gives the index of points(ν) with respect to the points of the parent of ν (if ν is a child).

Algorithm 4 merge(ν,$n_{\max}$)

---

if $c_0(\nu)$ and $c_0(c_1(\nu))$ (child and grandchild of ν) are leaves and both of the intervals of the leaves can be resolved on **then**

    Define a new leaf ν_0.

    $p_0(x) := \text{interpolant}(c_0(\nu))$

    $p_1(x) := \text{interpolant}(c_1(c_0(\nu)))$

    $w_0(x) := w_0(\nu)$

    $w_1(x) := w_1(\nu)$

    $\hat{s}(x) := w_0(x)p_0(x) + w_1(x)p_1(x)$

    if $\hat{s}(x)$ can be resolved by a Chebyshev interpolant $p(x)$ with degree less than $n_{\max}$ **then**

        $\text{interval}(\nu_0) := \text{interval}(c_0(\nu)) \cup \text{interval}(c_0(c_1(\nu)))$

        $\text{interpolant}(\nu_0) := p(x)$

        $\text{points}(\nu_0) := \text{Chebyshev grid of length } \deg(p(x)) \text{ on } [a_0, b_1]$

        $\hat{w}_0(x) := w_0(c_1(\nu))$

        $\hat{w}_1(x) := w_1(c_1(\nu))$

        $w_0(\nu) := \text{extend}(\hat{w}_0(x), \text{interval}(\nu_0))$

        $w_1(\nu) := \hat{w}_1(x)$

        $c_0(\nu) := \nu_0$

        $c_1(\nu) := c_1(c_1(\nu))$

    end if

else if $c_1(\nu)$ is a leaf and $c_1(c_0(\nu))$ is a leaf (and exists) **then**

    $\text{inv}(\text{merge}(\nu))$ (i.e., apply the algorithm, except swap 0 and 1)
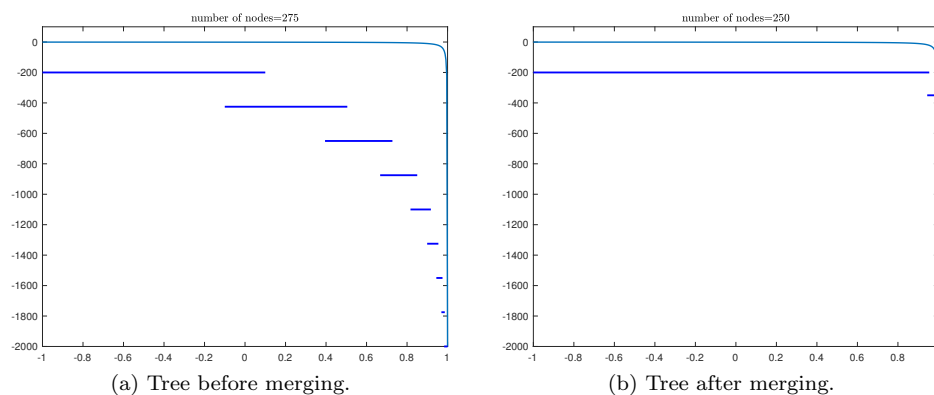
end if

(a) Tree before merging.   (b) Tree after merging.

FIG. 5. *An example of how the PUM with $t = 0.08$, $n_{max} = 128$ resolves $f(x) = \frac{1}{x - 1.0005}$* (a) *without merging and* (b) *after merging.*

    • leafpointindex(ν) := gives the index of leafpoints(ν) with respect to the leafpoints of the parent of ν (if ν is a child).

    Let $[\alpha, \beta] = \text{interval}(\nu)$. We want to construct matrices M, D such that

(24)
$$M \left. f(x) \right|_{\text{points}(\nu)} = \left. s_{[\alpha,\beta]}(x) \right|_{\text{leafpoints}(\nu)},$$
$$D \left. f(x) \right|_{\text{points}(\nu)} = \left. \frac{d}{dx} s_{[\alpha,\beta]}(x) \right|_{\text{leafpoints}(\nu)}.$$

Algorithm 5 $[M, D] = \text{diffmatrix}(\nu)$

---

  if ν is a leaf **then**
    $M :=$ the Chebyshev barycentric matrix from points(ν) to leafpoints(ν)
    $D_x :=$ Chebyshev differentiation matrix with grid points(ν)
    $D := MD_x$
  else
    $M, D := \text{zeros}(\text{length}(\text{leafpoints}(\nu)), \text{length}(\text{points}(\nu)))$
    for $k = 0, 1$ **do**
      $[M_k, D_k] := \text{diffmatrix}(c_k(\nu))$
      $M(\text{leafpointindex}(c_k(\nu)), \text{pointindex}(c_k(\nu))) =$
        $\hookrightarrow \text{diag}\left(\left.w_k\right|_{\text{leafpoints}(c_k(\nu))}\right) * M_k;$
      $D(\text{leafpointindex}(c_k(\nu)), \text{pointindex}(c_k(\nu))) =$
        $\hookrightarrow \text{diag}\left(\left.w_k\right|_{\text{leafpoints}(c_k(\nu))}\right) * D_k +$
        $\hookrightarrow \text{diag}\left(\left.\frac{d}{dx} w_k\right|_{\text{leafpoints}(c_k(\nu))}\right) * M_k;$
    end for
  end if

Let $I_k = \text{interval}(c_k(\nu))$, $w_k(x) = w_k(\nu)$ for $k = 0, 1$. Then

$$
(25) \quad \left.s_{[\alpha,\beta]}(x)\right|_{\text{leafpoints}(\nu)} = \sum_{k=0}^{1} w_k(x) s_{I_k}(x)\big|_{\text{leafpoints}(\nu)},
$$

$$
\left.\frac{d}{dx} s_{[\alpha,\beta]}(x)\right|_{\text{leafpoints}(\nu)} = \sum_{k=0}^{1} \left.\left(w_k(x)\frac{d}{dx} s_{I_k}(x) + \frac{d}{dx} w_k(x) s_{I_k}(x)\right)\right|_{\text{leafpoints}(\nu)}.
$$

Thus we can recursively build up the differentiation matrix through the tree T. Due to the support of the weights, for each term in (25) we only need evaluate the approximation $s_{I_k}(x)$ (or its derivative) for leafpoints$(\nu) \cap I_k$, i.e., leafpoints$(c_k(\nu))$. We describe how to construct the differentiation recursively in Algorithm 5, using MATLAB notation for matrices. At each leaf the interpolation matrix M has entries given by the barycentric interpolation formula based on second-kind Chebyshev points, as produced by the Chebfun command `barymat` [5].

For $x \in [\alpha, \beta]$ we only need to evaluate the local approximations for the patches x belongs to; this implies that the differentiation matrices will be inherently sparse. For example, Figure 6 shows the sparsity of the first derivative matrix for the tree generated in Figure 3. In this case, we have a sparsity ratio of around 76%.

6. PUM for boundary-value problems. Our method can be applied to solve linear and nonlinear BVPs. For instance, consider a simple Poisson problem with zero boundary conditions:

$$
(26) \quad \begin{aligned} u''(x) &= f(x) \text{ for } -1 < x < 1, \\ u(-1) &= 0, u(1) = 0. \end{aligned}
$$

Suppose that we have differentiation and interpolation matrices D_{xx} and M from section 5.2, X is the set of Chebyshev points over all the leaves, and X_I, X_B are the respective interior and boundary points of X. Let E_I and E_B be the matrices that map a vector to its subvector for the interior and boundary indices, respectively. Let
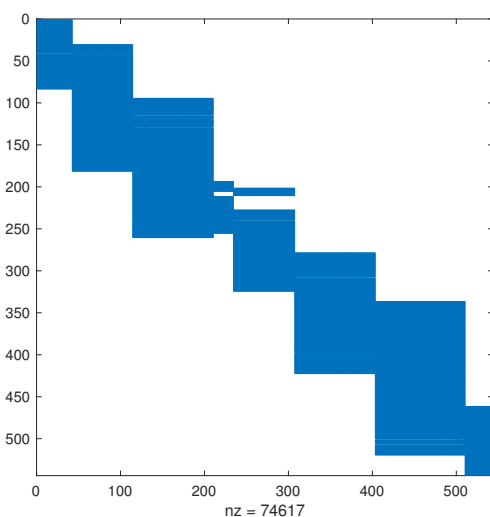
FIG. 6. *Sparsity of the first derivative matrix for the tree generated for Figure* 3.

F be the vector of values used for the local interpolants (i.e., if we had only two leaves whose interpolants used values F_1, F_2, we set $F = [F_1^T F_2^T]^T$). In order to find a PUM approximation $s(x)$ that approximates (26) we find F by solving the following linear system:

$$(27) \qquad \begin{bmatrix} E_I D_{xx} \\ E_B M \end{bmatrix} \begin{bmatrix} E_I F \\ E_B F \end{bmatrix} = \begin{bmatrix} f|_{X_I} \\ 0 \end{bmatrix}.$$

Algorithm 6 builds an adaptive solution for the BVP. We first construct a PU approximation $s(x)$ by solving the discretized system in (27). Sampling with $s(x)$, we use Algorithm 2 to determine if the solution is refined and split leaves that are determined to be unrefined. Here we also allow merging for a node with resolved left and right leaves (i.e., the left and right leaves can be merged back together).

Algorithm 6 $T = \text{refineBVP}(n_{\max}, t, \text{BVP})$

Define T as a tree with a single node with the domain of the BVP.
while T has unrefined leaves **do**
    Find values for the interpolants F of the leaves of T by solving a discretized
        system defined by the interpolation and differentiation matrices of T.
    sample(T,F)
    $s(x) = \text{eval}(\text{root}(T), x)$ (the PU approximation)
    sample($T,s(x)$)
    splitleaves(root(T),$n_{\max}$,t)
end while

6.1. BVP examples. We solve the stationary Burgers equation on the interval $[0, 1]$ with Robin boundary conditions [11]:

$$(28) \qquad \begin{aligned} \nu u''(x) - u(x)u'(x) &= 0, \\ \nu u'(0) - \kappa(u(0) - \alpha) &= 0, \\ \nu u'(1) + \kappa(u(1) + \alpha) &= 0, \end{aligned}$$

which has nontrivial solution

$$(29) \qquad u(x) = -\beta \tanh\left(\frac{1}{2}\beta\nu^{-1}\left(x - \frac{1}{2}\right)\right),$$

where β satisfies

$$(30) \qquad -\frac{1}{2}\beta^2\mathrm{sech}^2\left(\frac{1}{4}\beta\nu^{-1}\right) + \kappa\left[\alpha - \beta\tanh\left(\frac{1}{4}\beta\nu^{-1}\right)\right] = 0.$$

We choose $\nu = 5 \times 10^{-3}, \alpha = 1$, and $\kappa = 2$. We use `fsolve` in MATLAB to solve the BVP, supplying the Jacobian of the discretized nonlinear system. Starting with a linear guess $u(x) = 0$, we update the solution from the latest solve (i.e., if the solution $s(x)$ from Algorithm 6 is determined to be unresolved, we use it as the next initial guess). For this problem we set the Chebfun chopping tolerance to 10^{-10}. Our solution was resolved to the tolerance we set after four nonlinear solves; as seen in Figure 7, the final approximation had 298 nodes and the absolute error was less than 10^{-4}. On a machine with a 2.6 GHz Intel Core i5 processor, the solution was found in 1.3 seconds.
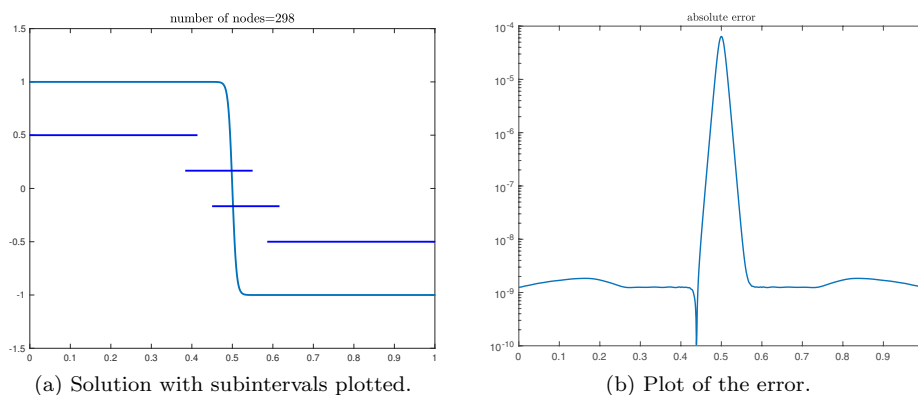

(a) Solution with subintervals plotted. (b) Plot of the error.

FIG. 7. *Numerical solution using the PUM and residual for the BVP* (28) *with* $\nu = 5 \times 10^{-3}$, $t = 0.1$, *and* $n_{max} = 128$.

We performed a similar experiment but instead used global Chebyshev interpolants. We adapt by increasing the degree of the polynomial from n to floor$(1.5n)$, starting with $n = 128$. We stop when we have a solution that is refined to the tolerance 10^{-10} (same as before). Both the solution and residual are shown in Figure 8; here we have that the absolute error is higher at 1.8e-2. The solution took 3.2 minutes on the same machine. There are two main reasons why the global solution performs much slower. First, in order to resolve the true solution with the tolerance 10^{-10}, the global Chebyshev solution requires 766 nodes, versus 300 for the PU approximation. Second, when adapting with the PUM, if a leaf is determined to be refined, the number of nodes is reduced as dictated in Algorithm 1 and the leaf is not split in further iterations. This keeps the total number of nodes lower while adapting.

7. Discussion. Our method offers a simple way to adaptively construct infinitely smooth approximations of functions that are given explicitly or that solve BVPs. By recursively constructing the PU weights with the binary tree, we avoid the need to
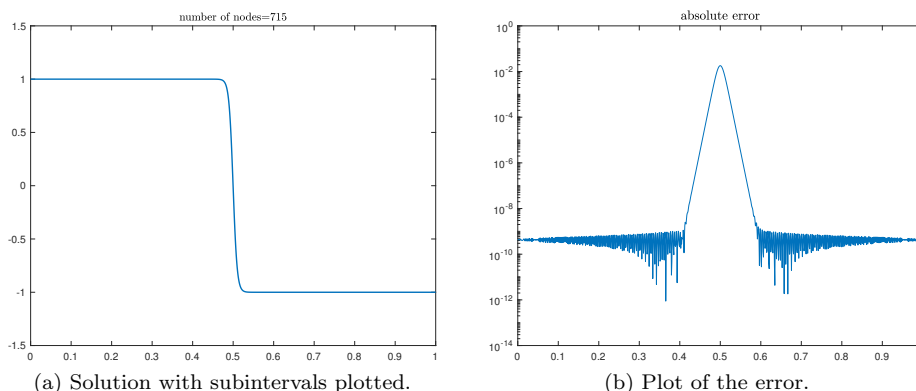
(a) Solution with subintervals plotted.

(b) Plot of the error.

FIG. 8. *Numerical solution using the global Chebyshev method and residual for the BVP* (28) *with* $\nu = 5 \times 10^{-3}$.

determine the neighbors of each patch (as would be needed with the standard Shepard's PU weights). While this is not a serious issue in one dimension, the complexity of how the patches overlap increases with higher dimension. For example, in 2D we could build a similar method on a box where we use tensor product Chebyshev approximations. We would refine by splitting the box into two overlapping parts (either in x or y) and recursively building a binary tree. We similarly would define PUs for each of the splits. If we used infinitely smooth weights at the splits, the 2D PU approximation would be infinitely smooth as well.

Our method leaves room for improvement. For instance, while merging helps reduce the number of nodes, in cases where we have a singularity right above the split the PU method overresolves in the overlap; this can be seen in Figure 3. The source of the problem is that patches may be adjacent in space but not in the tree. This could be resolved by a more robust merging algorithm. Alternatively we could determine an optimal splitting location through a Chebyshev–Padé approximation as in [6], but the PU adds a layer of complexity since we must optimize not just for the splitting location but for the size of the overlap.

Additionally it is possible to construct weights that are not C^∞ but have smaller norms in their derivatives. For instance,

$$(31) \qquad w_\ell(x) = \begin{cases} 1, & x \leq -t, \\ \frac{1}{4t^3}x^3 - \frac{3}{4t}x + \frac{1}{2}, & -t \leq x \leq t, \\ 0, & x > t, \end{cases}$$
$$w_r(x) = 1 - w_\ell(x)$$

defines a $C^1[-1, 1]$ piecewise cubic partition of unity, where $\|w_\ell'(x)\|_\infty = \frac{3}{4t}$. If a BVP requires higher smoothness, we could similarly construct a higher degree polynomial for the weights.

REFERENCES

[1] J. L. AURENTZ AND L. N. TREFETHEN, *Chopping a Chebyshev series*, ACM Trans. Math. Softw., 43 (2017), pp. 33:1–33:21, https://doi.org/10.1145/2998442.

[2] Z. BATTLES AND L. N. TREFETHEN, *An extension of MATLAB to continuous functions and operators*, SIAM J. Sci. Comput., 25 (2004), pp. 1743–1770, https://doi.org/10.1137/S1064827503430126.

[3] X.-C. CAI AND M. SARKIS, *A restricted additive Schwarz preconditioner for general sparse linear systems*, SIAM J. Sci. Comput., 21 (1999), pp. 792–797, https://doi.org/10.1137/S106482759732678X.

[4] T. A. DRISCOLL, F. BORNEMANN, AND L. N. TREFETHEN, *The chebop system for automatic solution of differential equations*, BIT, 48 (2008), pp. 701–723.

[5] T. A. DRISCOLL AND N. HALE, *Rectangular spectral collocation*, IMA J. Numer. Anal., 36 (2015), pp. 108–132.

[6] T. A. DRISCOLL AND J. A. C. WEIDEMAN, *Optimal domain splitting for interpolation by Chebyshev polynomials*, SIAM J. Numer. Anal., 52 (2014), pp. 1913–1927, https://doi.org/10.1137/130919428.

[7] R. FRANKE AND G. NIELSON, *Smooth interpolation of large sets of scattered data*, Internat. J. Numer. Methods Engrg., 15 (1980), pp. 1691–1704.

[8] M. GRIEBEL AND M. A. SCHWEITZER, *A particle-partition of unity method for the solution of elliptic, parabolic, and hyperbolic PDEs*, SIAM J. Sci. Comput., 22 (2000), pp. 853–890, https://doi.org/10.1137/S1064827599355840.

[9] D. H. McLAIN, *Two dimensional interpolation from random data*, Comput. J., 19 (1976), pp. 178–181.

[10] R. PACHÓN, R. B. PLATTE, AND L. N. TREFETHEN, *Piecewise-smooth chebfuns*, IMA J. Numer. Anal., 30 (2010), pp. 898–916.

[11] L. G. REYNA AND M. J. WARD, *On the exponentially slow motion of a viscous shock*, Comm. Pure Appl. Math., 48 (1995), pp. 79–120.

[12] A. SAFDARI-VAIGHANI, A. HERYUDONO, AND E. LARSSON, *A radial basis function partition of unity collocation method for convection–diffusion equations arising in financial applications*, J. Sci. Comput., 64 (2015), pp. 341–367.

[13] D. SHEPARD, *A two-dimensional interpolation function for irregularly-spaced data*, in Proceedings of the 23rd ACM National Conference, ACM, 1968, pp. 517–524.

[14] I. TOBOR, P. REUTER, AND C. SCHLICK, *Reconstructing multi-scale variational partition of unity implicit surfaces with attributes*, Graphical Models, 68 (2006), pp. 25–41.

[15] L. N. TREFETHEN, *Spectral Methods in MATLAB*, Software Environ. Tools 10, SIAM, Philadelphia, 2000, https://doi.org/10.1137/1.9780898719598.

[16] L. N. TREFETHEN, *Approximation Theory and Approximation Practice*, SIAM, Philadelphia, 2013.

[17] H. WENDLAND, *Scattered Data Approximation*, Cambridge University Press, 2004.