Report no. 08/20

Barycentric-Remez algorithms for best polynomial approximation in the chebfun system

Ricardo Pachón and Lloyd N. Trefethen

Variants of the Remez algorithm for best polynomial approximation are presented based on two key features: the use of the barycentric interpolation formula to represent the trial polynomials, and the setting of the whole computation in the chebfun system, where the determination of local and global extrema at each iterative step becomes trivial. The new algorithms make it a routine matter to compute approximations of degrees in the hundreds, and as an example, we report approximation of $|x|$ up to degree 10,000. Since barycentric formulas can also represent rational functions, the algorithms we introduce may also point the way to new methods for computing best rational approximations.

Oxford University Computing Laboratory
Numerical Analysis Group
Wolfson Building
Parks Road
Oxford, England OX1 3QD

December, 2008

1 Introduction

Consider the following best polynomial approximation problem: Given a continuous function f defined on an interval $I = [a, b]$, find a function p^* in the space $\mathcal{P}_n$ of polynomials of degree less or equal to n such that

$$\|f - p^*\| \leq \|f - p\| \quad \text{for all } p \in \mathcal{P}_n, \tag{1.1}$$

where $\|\cdot\|$ is the supremum norm $\|\phi\| = \max_{x\in I} |\phi(x)|$. The approximation p^* exists and is unique, and is known as the best, uniform, Chebyshev or minimax approximation to f. Discussions of this problem can be found in almost every book on approximation theory [8, 10, 17, 20, 21, 25, 30].

Starting with Chebyshev himself, the best approximation problem was studied from the second half of the 19th century to the early 20th century, and by 1915 the main results had been established [34]. A second wave of interest came in the 1950s and 1960s when computational aspects were investigated. The focus of much of this work was the algorithm introduced by Evgeny Yakovlevich Remez in a series of three papers published in 1934 [27, 28, 29], and in this period were developed a deep understanding of its theoretical properties as well as numerous variations for its practical implementation. In the 1970s the Remez algorithm also became a fundamental tool of digital signal processing, where it was introduced by Parks and McClellan in the context of filter design [24].

In the present work we consider the use of the barycentric interpolation formula in the Remez algorithm. Specifically, we represent p^* as an interpolant through $n+1$ points instead of expressing it, for example, as an expansion in the monomial basis. The main benefits obtained are a considerable simplification, both practically and conceptually, and numerical robustness.

We have two aims in this paper. The first is to present the barycentric-Remez algorithm for best polynomial approximation. We study the changes in the classical Remez algorithm, the theoretical implications of this variation and the main features of its implementation. In particular, we use the new algorithm to compute p^* for the example $f(x) = |x|$ with n in the thousands.

Parks and McClellan [24] seem to have been the first to have implemented the Remez algorithm with the barycentric interpolation formula. Looking for an efficient filter design procedure, they found that they required a robust interpolation method to evaluate the error of trial polynomials, as they recall in [18]:

> *"Once the program started to be employed for extremely long filters, it was somewhat surprising that the method continued to work. Much of that success can be attributed to the Barycentric interpolation formula."*

A similar barycentric strategy has been proposed more recently for the computation of orthonormal wavelets with the Remez algorithm [31]. Nevertheless, in this paper we examine aspects of the barycentric-Remez algorithm that, to our knowledge, have not been mentioned before, including a computation of barycentric weights stable enough

to work for very large degrees and the use of barycentric formulas in the polynomial as opposed to trigonometric polynomial case.

Our second aim is to present implementations of barycentric-Remez algorithms in the chebfun system. Chebfun is a software system developed in MATLAB that allows the user to manipulate continuous functions numerically in a manner analogous to what is done for discrete vectors. The algorithms presented here take the chebfun of a continuous function as input and return the chebfun of its best approximation. Chebfun greatly simplifies the implementation of these algorithms and proves to be a most convenient tool for the Remez algorithm due to its powerful capabilities for locating global and local extrema. The reader can find expositions of the chebfun system in [3] and [23] and at `www.comlab.ox.ac.uk/chebfun`.

We give a comprehensive presentation of the classical Remez algorithm in Section 2, emphasizing its two main steps: the computation of a trial reference (Section 2.1) and a trial polynomial (Section 2.2). Section 2.3 presents a compact implementation of the classical Remez algorithm in chebfun and some numerical examples. Section 3 introduces the barycentric-Remez algorithm, together with its chebfun implementation. The codes presented in this paper make it easy to explore properties of best approximations. In Section 4 we show some of these possibilities. We compare the convergence of best approximants computed with chebfuns to Jackson-type bounds for continuous and Lipschitz continuous functions. Our short and efficient implementation allows us to replicate with little effort some of the computations associated with the disproof given by Varga and Carpenter in 1985 [39] of a conjecture formulated by Bernstein in 1914 for polynomial approximation of $|x|$. We conclude in Section 5 with comments on the problem of extending the barycentric-Remez algorithm to rational approximations.

2 Classical Remez algorithms in the chebfun system

Since the best approximation is unique, we can define the operator that assigns to each continuous function its best polynomial approximation of fixed degree. It is well known that this operator, although continuous, is nonlinear (for an example see [17, p. 33]), and so we need iterative methods to compute p^*. The Remez algorithm is one such method. Other important algorithms are the differential correction algorithms, which rely on ideas of linear programming [26]. We will not mention these methods further in this paper.

We begin our discussion of the Remez algorithm by recalling two theorems that are essential to it. The first was first proved by Borel in 1905 [6] [8, p. 75] [25, p. 77].

Theorem 1 (Equioscillation Property) *A polynomial $p \in \mathcal{P}_n$ is the best approximation to f (that is, $p = p^*$) if and only if there exists a set of $n+2$ distinct points $\{x_i\}_{i=0}^{n+1}$ such that*

$$f(x_i) - p(x_i) = \lambda \sigma_i \|f - p^*\|, \quad i = 0, \ldots, n+1, \tag{2.1}$$

where $\sigma_i := (-1)^i$ and $\lambda = 1$ or $\lambda = -1$ is fixed.

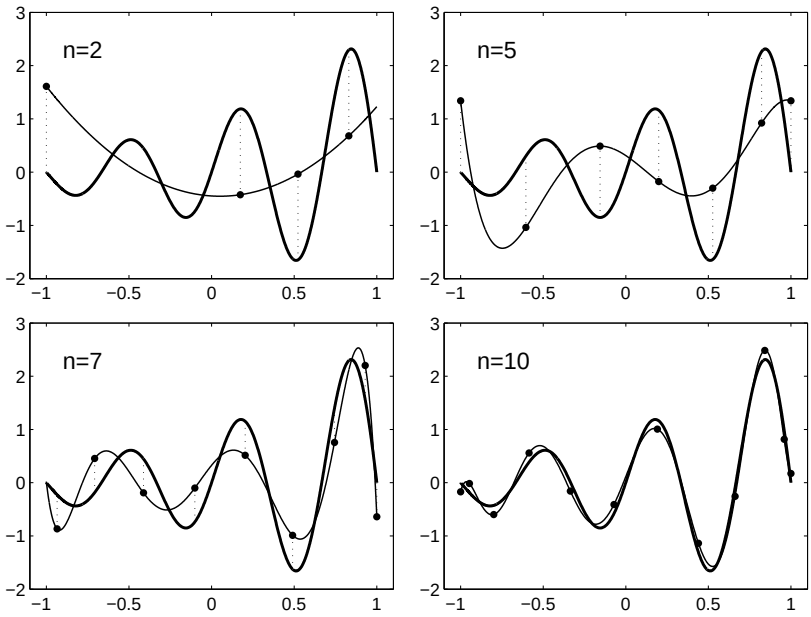

Figure 1: Best polynomial approximations (thin lines) of degree 2, 5, 7 and 10 to $f(x) = \sin(3\pi x)\exp(x)$ (bold lines) on $[-1,1]$. The dots show the polynomial at the reference and the dotted vertical bars the corresponding errors, of equal lengths and alternating in orientation.

A set of points $A^* := \{x_i\}_{i=0}^{n+1}$ that satisfies (2.1) is called a *reference*. Figure 1 shows p^* for $f(x) = \sin(3\pi x)\exp(x)$, $I = [-1,1]$ and $n = 2, 5, 7$ and 10, and the references of 4, 7, 9 and 12 points respectively where $f - p^*$ equioscillates. Analogous properties hold for other types of approximations such as best rational, CF and Padé [37].

Theorem 1 can be generalized for approximations that satisfy the "Haar condition" [25, p. 77], of which polynomials are a special case. This allows us to look for best approximations in other sets of functions, for example trigonometric polynomials, which are the ones used for the Parks-McClellan algorithm. This paper works only with polynomials, but we believe that our methods can be carried over to the trigonometric case.

The second theorem, proved by de la Vallée Poussin in 1910 [11], establishes an inequality between the alternating error of a trial polynomial and the error of the best approximation [8, p. 77][25, Thm. 7.7].

Theorem 2 (de la Vallée Poussin) *Let $p \in \mathcal{P}_n$ and $\{y_i\}_{i=0}^{n+1}$ be a set of $n+2$ distinct points in I such that $\operatorname{sign}\big(f(y_i) - p(y_i)\big) = \lambda\sigma_i$, $i = 0, \ldots, n+1$, with σ_i and λ defined as in Theorem 2.1. Then, for every $q \in \mathcal{P}_n$,*

$$\min_i |f(y_i) - p(y_i)| \le \max_i |f(y_i) - q(y_i)|, \tag{2.2}$$

and in particular,

$$\min_i |f(y_i) - p(y_i)| \le \|f - p^*\| \le \|f - p\|. \tag{2.3}$$

Theorem 2 asserts that a polynomial $p \in \mathcal{P}_n$ whose error oscillates $n+2$ times is "near-best" in the sense that

$$\|f-p\| \le C\|f-p^*\|, \qquad C = \frac{\|f-p\|}{\min_i |f(y_i)-p(y_i)|} \ge 1.$$

The Remez algorithm constructs a sequence of trial references $\{A_k\}$ and trial polynomials $\{p_k\}$ that satisfy this alternation condition in such a way that $C \to 1$ as $k \to \infty$. At the kth step the algorithm starts with a trial reference A_k and then computes a polynomial p_k such that

$$f(x_i) - p_k(x_i) = \sigma_i h_k, \quad x_i \in A_k, \tag{2.4}$$

where h_k is the *levelled error* (positive or negative), defined as $h_k := f(x_i) - p_k(x_i)$ for all $x_i \in A_k$. Then, a new trial reference A_{k+1} is computed from the extrema of $f - p_k$ in such a way that $|h_{k+1}| \ge |h_k|$ is guaranteed. This monotonic increase of the levelled error is the key observation in showing that the algorithm converges to p^* [25, Thm 9.3]. In Section 2.1 we explain how to compute a trial polynomial and levelled error from a given trial reference, and in Section 2.2 we show how to adjust the trial reference from the error of the trial polynomial.

2.1 From a trial reference to a trial polynomial

We let $\{\phi_j; j = 0, 1, \ldots, n\}$ be a basis of $\mathcal{P}_n$ and express the elements of the latter in the form

$$p(x) = \sum_{j=0}^{n} c_j \phi_j(x).$$

A continuous function f and a set $\{x_i\}_{i=0}^{n+1}$ of $n+2$ points uniquely determine a polynomial p and a levelled error h such that (2.4) is satisfied. The conditions (2.4) amount to a linear system of $n+2$ equations in $n+2$ unknowns: $n+1$ parameters to describe the polynomial, plus the unknown h:

$$\begin{pmatrix} \phi_0(x_0) & \phi_1(x_0) & \cdots & \phi_n(x_0) \\ \phi_0(x_1) & \phi_1(x_1) & \cdots & \phi_n(x_1) \\ \vdots & \vdots & & \vdots \\ \phi_0(x_n) & \phi_1(x_n) & \cdots & \phi_n(x_n) \\ \phi_0(x_{n+1}) & \phi_1(x_{n+1}) & \cdots & \phi_n(x_{n+1}) \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} f(x_0)+\sigma_0 h \\ f(x_1)+\sigma_1 h \\ \vdots \\ f(x_n)+\sigma_n h \\ f(x_{n+1})+\sigma_{n+1} h \end{pmatrix}. \tag{2.5}$$

The chebfun implementation presented in Section 2.3 computes h and the coefficients $\{c_j\}$ simultaneously by solving the $(n+2)\times(n+2)$ system (2.5), having h as one of the unknowns.

The choice of basis $\{\phi_j\}$ is crucial in the numerical solution of (2.5) [12]. The monomial basis, for example, is a terrible choice: the condition number of the resulting Vandermonde matrix grows exponentially in general [14, p. 417]. The use of Chebyshev polynomials will usually improve matters, but can still result in an ill-conditioned system for arbitrary sets of points [1].

2.2 From a trial polynomial to a new trial reference

Suppose that for a trial reference A_k there is a polynomial p_k such that $f(x_i) - p_k(x_i) = \sigma_i h_k$ but $|h_k| < \|f - p^*\|$. The goal is to obtain a new reference $A_{k+1} = \{y_i\}_{i=0}^{n+1}$ where the error of the polynomial $p_{k+1} \in \mathcal{P}_n$ equioscillates with levelled error $|h_{k+1}| > |h_k|$. The key to finding the new reference is the de la Vallée Poussin theorem.

Since $f(y_i) - p_{k+1}(y_i)$ will equioscillate, the right side of (2.2) will be equal to h_{k+1}. Thus, to be sure of increasing the levelled error, the replacement of A_k by A_{k+1} must satisfy

$$|h_k| \le \min_i |f(y_i) - p_k(y_i)|, \ y_i \in A_{k+1}. \tag{2.6}$$

That is, the polynomial p_k must oscillate on A_{k+1} (but not necessarily equioscillate) with amplitude greater than or equal to $|h_k|$. If condition (2.6) is satisfied, it follows from (2.2) that the levelled error increases from p_k to p_{k+1}.

Remez proposed two strategies to achieve this. One is to move one of the points of A_k to the abscissa of the global extremum while keeping the sign alternation; the other is to replace all the points of A_k by $n+2$ oscillating local extrema satisfying (2.6) and to include in A_{k+1} the abscissa of the global extremum. These strategies are known as the first and second Remez algorithms, respectively.

More specifically, the first Remez algorithm constructs A_{k+1} by exchanging a point $x_{\text{old}} \in A_k$ with the global extremum x_{new} of $f - p_k$ in such a way that the alternation of signs of the error is maintained. If $x_0 < x_{\text{new}} < x_{n+1}$, then x_{old} is the closest point in A_k for which the error has the same sign as at x_{new}. If $x_{\text{new}} < x_0$ and the signs of x_{new} and x_0 coincide then x_{old} is x_0; if $x_{\text{new}} < x_0$ but the signs of x_{new} and x_0 are different, then x_{old} is x_{n+1}. Similar rules apply if $x_{\text{new}} > x_{n+1}$.

The second Remez algorithm constructs the set $\tilde{A}^{k+1}$ of points in A^k and local extrema x_r of $f - p_k$ such that $|(f - p_k)(x_r)| > |h_k|$. Then, for each subset of $\tilde{A}^{k+1}$ of consecutive points with the same sign it keeps only one for which $|f - p_k|$ attains the largest value. From the resulting set, A_{k+1} is obtained by choosing $n+2$ consecutive points that include the global extremum of $f - p_k$.

The speed of convergence of the Remez algorithm depends on the choice of the change of reference. Assuming f is twice differentiable, Veidinger [40] showed that the convergence is quadratic for the second algorithm, and Powell [25, Sec. 9.4] proved that the first algorithm converges at an $(n+2)$-step quadratic rate.

2.3 The Remez algorithm in the chebfun system

The problem of replacing the trial reference is one of optimization: on the error function of the trial polynomial p_k, find the global extremum or the alternating local extrema. In the chebfun system this step is easy since one of the main features in chebfun is a very efficient root finder [3] based on ideas of Boyd [7].

The following is our chebfun code that replaces the trial reference A_k using the prescription of the second Remez algorithm. The input arguments are a column vector

`xk` with the trial reference, the chebfun `e` of the error $f - p_k$ and the levelled error `h` $= h_k$.

```
function [xk,norme] = exchange(xk,e,h)
rr = [-1; roots(diff(e)); 1];             % critical pts of the error
pos = find(abs(e(rr))>=abs(h));           % vals above leveled error
[r,m] = sort([rr(pos); xk]);
er = [e(rr(pos));(-1).^(0:length(xk)-1)'*h];
er = er(m);
s = r(1); es = er(1);                     % pts and vals to be kept
for i = 2:length(r)
  if sign(er(i)) == sign(es(end)) &...    % from adjacent pts w/ same sign
          abs(er(i))>abs(es(end))         % keep the one w/ largest val
      s(end) = r(i); es(end) = er(i);
  elseif sign(er(i)) ~= sign(es(end))     % if sign changes, concatenate
      s = [s; r(i)]; es = [es; er(i)];    % pts and vals
  end
end
[norme,idx] = max(abs(es));               % choose n+2 consecutive pts
d = max(idx-length(xk)+1,1);              % that include max of error
xk = s(d:d+length(xk)-1);
```

To implement the first Remez algorithm, we can change lines 2–3 of this code to `[tmp,pos] = max(abs(er)); pos = 1`. Figure 2 shows the number of iterations for both the first and second Remez algorithms when computing p^* for the function in Figure 1.

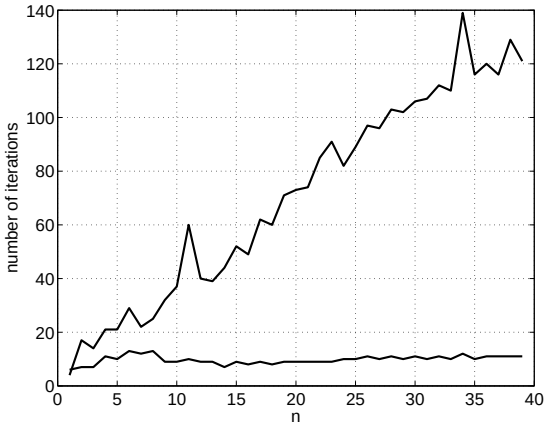

Figure 2: Number of iterations required by the first and second Remez algorithms (upper and lower lines, respectively) to obtain the best polynomial approximation of degree n to the function $f(x) = \sin(3\pi x)\exp(x)$.

The following function uses `exchange` to implement the Remez algorithm. Note that

this entire code, including exchange, is less than 40 lines in MATLAB/chebfun.

```
function p = remez(f,n);                % compute deg n BA to chebfun f
xk = cos(pi*(n+1:-1:0)'/(n+1));         % initial reference
normf = norm(f);
delta = 1;                              % force to enter the loop
while delta/normf > 1e-11
  A = zeros(n+2,n+1);
  for j = 1:n+1                         % assemble matrix of (2.5) using
    t = chebfun((-1).^(j-1:-1:0));      % Chebyshev polynomials
    A(:,j) = t(xk);
  end
  A = [A (-1).^(0:n+1)'];
  c = A\f(xk);                          % solve (2.5)
  h = c(end); c = c(end-1:-1:1);        % leveled error
  p = chebfun(chebpolyval(c));          % chebfun of trial polynomial
  e = f - p;                            % chebfun of the error
  [xk,norme] = exchange(xk,e,h);        % replace reference
  delta = norme - abs(h);               % stopping value
end
```

The implementation just shown takes Chebyshev polynomials as the basis of $\mathcal{P}_n$. The command `chebfun((-1).^(j-1:-1:0))` constructs $T_j(x)$, and the command `chebpolyval(c)`, analogous to MATLAB's `polyval`, uses a vector `c` $= [c_n, c_{n-1} \ldots, c_0]$ to evaluate $\sum_{j=0}^{n} c_j T_j(x)$. As an illustration, these codes were applied to compute the best polynomial approximations of degree 10 of the functions in Table 1. The approximations are plotted in Figure 3.

This combination of Chebyshev polynomials as the basis for $\mathcal{P}_n$ and Chebyshev points as the initial reference provides a reasonably reliable setting for the Remez algorithm. Other choices, for example the monomial basis or equidistant points, often fail for n larger than 50.

3 A barycentric-Remez algorithm in the chebfun system

Even better, however, is a Lagrange basis, which is used by the barycentric-Remez algorithm: Given a set $\{\tilde{x}_j\}_{j=0}^{n}$ of $n+1$ prescribed interpolation points, we express $p(x)$, $x \in I$ as

$$p(x) = \sum_{j=0}^{n} p(\tilde{x}_j)\ell_j(x), \quad \ell_j = \frac{\prod_{\nu=0,\nu\neq j}^{n} 2(x-\tilde{x}_\nu)}{\prod_{\nu=0,\nu\neq j}^{n} 2(\tilde{x}_j-\tilde{x}_\nu)}. \tag{3.1}$$

i	f_i	$\|f_i - p^*\|$
1	$\tanh(x+0.5) - \tanh(x-0.5)$	0.00000030009195
2	$\sin(\exp(x))$	0.00000178623400
3	$\sqrt{x+1}$	0.01978007008380
4	$\sqrt{\lvert x-0.1\rvert}$	0.11467950933306
5	$1 - \sin(5\lvert x-0.5\rvert)$	0.14320591976530
6	$\min\{\operatorname{sech}(3\sin(10x)), \sin(9x)\}$	0.33561414233366
7	$\max\{\sin(20x), \exp(x-1)\}$	0.38723296760148
8	$\operatorname{sech}\big(10(0.5x+0.3)\big)^2 + \operatorname{sech}\big(100(0.5x+0.1)\big)^4 + \operatorname{sech}\big(1000(0.5x-0.1)\big)^6$	0.49987078860779
9	$\log(1.0001+x)$	1.40439492981221

Table 1: Best approximation errors for nine functions by polynomials of degree 10.

The factors of 2 are included to eliminate the risk of underflow (correcting for the fact that the interval $[-1,1]$ has logarithmic capacity $\frac{1}{2}$, not 1). Notice that

$$\ell_j(\tilde{x}_i) = \begin{cases} 1, & j = i \\ 0, & \text{otherwise,} \end{cases} \quad j, i = 0, \ldots, n. \tag{3.2}$$

Notice also that we are now proposing the use of a basis that is not prescribed in advance but depends on the data.

Lagrange interpolation is a fundamental tool in numerical analysis whose success depends on two key choices: the interpolating nodes, and the formula used for implementation. It is well known that the sets of nodes for which Lagrange interpolation is well conditioned are distributed with asymptotic density proportional to $(1-s^2)^{-1/2}$, if s is a rescaling of x to $[-1,1]$ [38, Chap. 5], such as the set of Chebyshev points

$$\tilde{x}_j = \frac{1}{2}(a+b) + \frac{1}{2}(b-a)\cos\Big(\frac{j\pi}{n}\Big), \quad j = 0, 1, \ldots, n. \tag{3.3}$$

Besides being ill-conditioned for certain arrays of grid points, Lagrange interpolation also suffers from numerical instability when implemented improperly. The barycentric formula

$$p(x) = \frac{\displaystyle\sum_{j=0}^{n} \frac{\tilde{w}_j}{x - x_j} p(\tilde{x}_j)}{\displaystyle\sum_{j=0}^{n} \frac{\tilde{w}_j}{x - x_j}}, \tag{3.4}$$

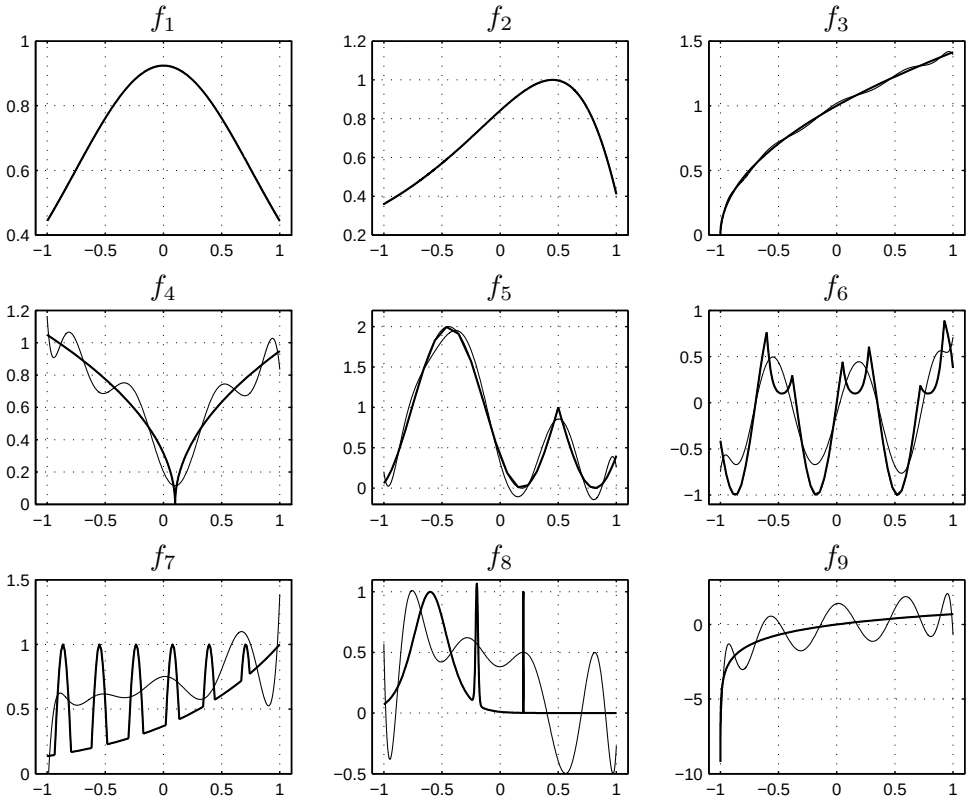

Figure 3: Best polynomial approximations of degree 10 (thin line) to the functions in Table 1 (bold line) computed with the classical Remez algorithm.

where

$$\tilde{w}_j = \frac{1}{\prod_{\nu \neq i} 2(\tilde{x}_i - \tilde{x}_\nu)}, \quad j = 0, \ldots, n, \tag{3.5}$$

are the barycentric weights, is a reformulation of (3.1), and it is stable and fully effective for the evaluation of high-degree polynomials. In particular, Higham has shown that (3.4) is forward stable for point sets with small Lebesgue constant, such as Chebyshev points [15].

Because of the factors of 2 in (3.1) and (3.5), these products are roughly of size $O(1)$ and representable in floating point arithmetic. However, the partial products formed along the way when such a product is evaluated, say, from left to right, may underflow if n is very large (e.g. in the thousands). One solution to this problem would be to order the partial products to compensate, e.g. by a discrete Leja ordering [36]. However in this application we are able to use a simpler solution: rather than multiplying factors,

we sum their logarithms:

$$\tilde{w}_j = \frac{\prod_{\nu\neq j}\operatorname{sign}(\tilde{x}_j - \tilde{x}_\nu)}{\exp\Big(n\log 2 + \sum_{\nu\neq j}\log\big|\tilde{x}_j - \tilde{x}_\nu\big|\Big)}, \quad j = 0, \dots, n.$$

For a review of barycentric interpolation formulas, see [5].

The set of Lagrangian nodes that we use at a fixed step in the barycentric-Remez algorithm is a subset of $n+1$ points from the $(n+2)$-point trial reference $A = \{x_i\}_{i=0}^{n+1}$ omitting, say, the point x_j. From (3.2) it follows that the matrix of (2.5) is the $(n+1)\times(n+1)$ identity except with an additional jth row inserted whose entries are the values of the various Lagrange functions at the particular point x_j. Discarding this row, we end up with the system

$$p(x_i) = f(x_i) + \sigma_i h, \quad i = 0, \dots, n+1,\ i \neq j. \tag{3.6}$$

Since the values $f(x_i)$ are known, the only value left to compute is h. With it, the values $p(x_i)$ can be obtained, and with them, the trial polynomial p can be obtained from the barycentric formula (3.4).

For any basis $\{\phi_j\}$ of $\mathcal{P}_n$ it can be shown that h can be found independently of the coefficients $\{c_j\}$ [25, Thm. 9.1]. For the Lagrange basis we can compute explicitly a closed expression without the values $p(x_i)$. Consider the discarded row of the system (3.6), $p(x_j) + \sigma_j h = f(x_j)$, and use Lagrange interpolation on the remaining set of $n+1$ points to compute $p(x_j)$,

$$\sum_{\substack{i=0\\ i\neq j}}^{n+1} p(x_i)\ell_i^j(x_j) + \sigma_j h = f(x_j), \quad x_j \in A,\ j = 0, \dots, n+1,\ j \neq i, \tag{3.7}$$

where ℓ_i^j is the ith element of the Lagrange basis that uses A as the Lagrange nodes, except for x_j,

$$\ell_i^j(x) := \prod_{\substack{\nu=0\\ \nu\neq i,j}}^{n+1} \frac{2(x - x_\nu)}{2(x_i - x_\nu)},\ x_i, x_\nu \in A.$$

Notice that

$$\ell_i^j(x_j) = \prod_{\substack{\nu=0\\ \nu\neq i,j}}^{n+1} \frac{2(x_j - x_\nu)}{2(x_i - x_\nu)} = \frac{\prod_{\nu\neq j} 2(x_j - x_\nu)}{\prod_{\nu\neq i} 2(x_i - x_\nu)} \cdot \frac{x_i - x_j}{x_j - x_i} = -\frac{w_i}{w_j}, \tag{3.8}$$

where $\{w_i\}$ are the barycentric weights we would get if we considered all the points A as the Lagrange nodes, i.e.,

$$w_j = \prod_{\substack{\nu=0\\ \nu\neq j}}^{n+1} \frac{1}{2}(x_j - x_\nu)^{-1}, \quad x_j, x_\nu \in A$$

Inserting (3.8) in (3.7), we obtain

$$-\sum_{\substack{i=0\\ i\neq j}}^{n+1} p(x_i)w_i + \sigma_j w_j h = f(x_j)w_j.$$

Summing over j, and noting that $\sum_{j=0}^{n+1} p(x_j)w_j = 0$, we obtain the compact formula

$$h = \frac{\displaystyle\sum_{j=0}^{n+1} w_j f(x_j)}{\displaystyle\sum_{j=0}^{n+1} \sigma_j w_j}. \tag{3.9}$$

With the values $p(x_j)$ and h we construct the chebfun of the polynomial p given by (3.4), resulting in the chebfun of the trial polynomial. Since we know that the degree of the polynomial is n, we can pass this argument to the constructor and avoid the use of the procedure that automatically computes the degree of the corresponding chebfun:

```
>> chebfun(@(x) bary(x,w,xk,pk), n+1),
```

where `xk` is the trial reference, `pk` are the values (3.6), `w` is a vector with the barycentric weights and `bary` is the barycentric formula, implemented in MATLAB, for example, in [5, p. 510]. The polynomial of degree $n+1$ that interpolates the $n+2$ points of A_k coincides with the trial polynomial p of degree n, and thus the weights w_j used to compute h in (3.9) can be used in the barycentric interpolation to construct p.

The initial guess of the barycentric-Remez algorithm, in principle, can be any set of points. However, to ensure that the first step of the process executes correctly, we begin with a trial reference of $n+2$ Chebyshev points (3.3). If h_0 is below machine precision the error will not equioscillate on this initial reference and all the values $p(x_i)$ will be close to zero. However the next reference can still be computed by following the exchange rules of the second Remez algorithm explained in Section 2.2.

The following code implements the barycentric-Remez algorithm. As an example, we compute with it the best approximation of the function f_8 with $n = 200$. Figure 4 shows p^* for this difficult function.

```
function p = bary_remez(f,n);                % compute deg n BA to chebfun f
xk = cos(pi*(n+1:-1:0)'/(n+1));              % initial reference
sigma = (-1).^(0:n+1)';                      % alternating signs
normf = norm(f);
delta = 1;                                   % force to enter the loop
while delta/normf > 1e-11
   fk = f(xk);                               % function vals
   w = ones(n+2,1);
   for i = 1:n+2                             % compute barycentric weights
```

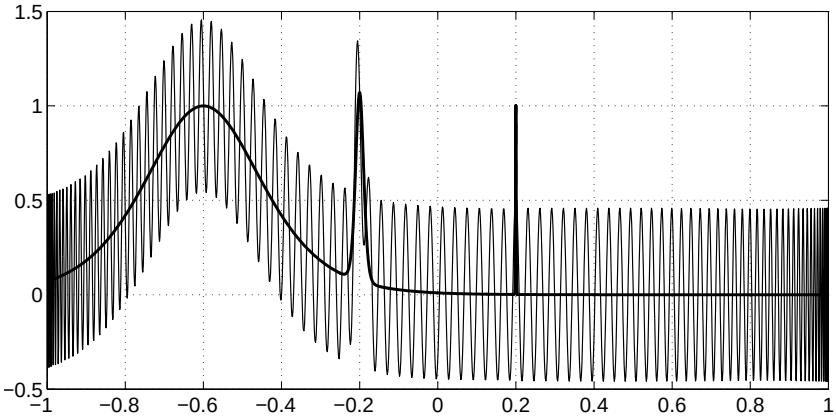

Figure 4: Best polynomial approximation of degree 200 to the function f_8. The large error is consistent with its slow convergence as $n \to \infty$ (see Figure 5).

```
      v = 2*(xk(i)-xk);
      vv = exp(sum(log(abs(v(find(v))))));
      w(i) = 1./(prod(sign(v(find(v))))*vv);
   end
   h = (w'*fk)/(w'*sigma);                % levelled reference error
   pk = fk - h*sigma;
   p = chebfun(@(x) bary(x,xk,pk,w),n+1); % chebfun of trial polynomial
   e = f - p;                             % chebfun of the error
   [xk,norme] = exchange(xk,e,h);         % replace reference
   delta = norme - abs(h);                % stopping value
end
```

From our experiments we have seen that for certain functions and values of n, one of the steps in the barycentric-Remez algorithm constructs a trial polynomial with a very large extremum, located usually near the endpoints, when changing all the points of the reference with the strategy of the second algorithm. The large norm of the polynomial introduces a very large error in h, breaking the computation. A simple way to deal with this problem is to reverse the last step and recompute the trial reference but using the one-point exchange strategy of the first Remez algorithm. This implementation is the one that we used for the experiments in the next section.

4 Numerical experiments

How large can we make n when computing $p^* \in \mathcal{P}_n$ for a given continuous function? The answer has varied with the years. In one of the papers that introduced his algorithm in 1934, Remez gave the polynomial coefficients of p^* for $f(x) = |x|$ for $n = 5, 7, 9, 11$ [28]. Twenty-five years later, Stiefel [35], Curtis and Frank [9] and Murnaghan and Wrench

[22] applied different techniques to compute best approximations of $\sin^{-1} x$, $\tan^{-1}$, $\log x$, 2^x and $|x^5|$ by polynomials of degrees varying between 2 and 18.

The first computer programs for uniform approximation appeared in the 1960s, and included a ALGOL code by Golub and Smith [13] and FORTRAN codes by Barrodale and Phillips [2], and by Simpson [33] based on Schmitt's algorithm [32]. They have been used, for example, for Chebyshev curve fitting with $n > 20$. More recently, Le Bailly and Thiran [16] reported the computation of best approximants of degrees up to 64 as a step to obtaining best approximants on the unit circle in the complex plane. And higher degree approximations have been computed for particular kinds of functions. For example, McClellan and Parks [18] comment on experiments they did thirty years ago in their work with Rabiner [19] involving polynomials of degree about 500 in the context of filter design. Rabiner, at Bell Labs, had some powerful computers to work with:

> *"From our perspective at Rice, it seemed that Larry wanted to set records for the longest optimal filter ever designed. One day we received a printout of the coefficients of a length-*1401 *filter; this probably would have consumed several days of CPU time on our batch machine at Rice."*

In this section we report computations of best polynomial approximations with n in the hundreds and thousands using the methods presented above in seconds or at most minutes.

A collection of results known as the Jackson theorems establish bounds for the error of best polynomial approximation as n increases in terms of the smoothness of f. For example, if f satisfies the Lipschitz condition $|f(x_1)-f(x_2)| \le C|x_1-x_2|$, $x_1, x_2 \in [-1,1]$, then the approximation error is bounded by [25, Thm 16.5]

$$\|f - p^*\| \le \frac{C\pi}{2n+1}. \tag{4.1}$$

The functions f_5, f_6, f_7 and f_8 are Lipschitz continuous. Using the chebfun command `norm(diff(f),inf)` we calculated the Lipschitz coefficients, and Figure 5 shows the best approximation errors for n from 1 and 700 and the bound (4.1) for f_5, f_6 and f_8. The large error of the best approximation in Figure 4 is consistent with the large Lipschitz constant $C_8 \approx 7 \times 10^2$.

For functions that do not satisfy a Lipschitz condition, one can establish the bound [25, Thm 16.5]

$$\|f - p^*\| \le \frac{3}{2}\omega_f\Big(\frac{\pi}{n+1}\Big), \tag{4.2}$$

where ω_f is the modulus of continuity of f, i.e.,

$$\omega_f(\delta) = \sup_{|x_1-x_2|<\delta} |f(x_1) - f(x_2)|, \quad x_1, x_2 \in [-1,1].$$

For both the functions f_3 and f_4, this bound becomes $\frac{3}{2}\sqrt{\pi/(n+1)}$, and in Figure 6 we compare this quantity with the best approximation errors.

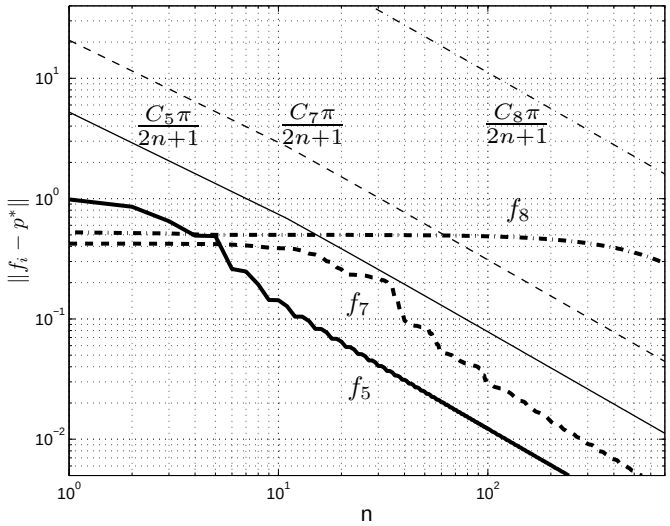

Figure 5: Error of best polynomial approximations for functions f_5, f_7 and f_8 compared with the Jackson bounds (4.1) for Lipschitz continuous functions. The error for f_6, not shown, follows very closely the error for f_7.

The computations of best approximations of $\sqrt{x}$ in $[0,1]$ and $|x|$ in $[-1,1]$ are equivalent. Remez himself used his algorithm and this equivalence to compute the best approximations to $|x|$ by polynomials of odd degrees up to 11 with an accuracy of 10^{-5}. Using the overloaded command `poly(p)`, where `p` is the chebfun of the best polynomial approximation, we can check the coefficients that Remez published in [28]. For example, with $n = 11$, it takes about 0.1 seconds to find that the coefficients c_k in the monomial basis $\{x^k\}$ are:

k	c_k (Remez [28])	c_k (chebfun)
0	0.027837	0.02784511855
2	4.753770	4.75365049278
4	−20.646839	−20.64625015816
6	47.776685	47.77533460523
8	−49.593272	−49.59209097049
10	18.709656	18.70935603064

Evidently Remez's coefficients were accurate to about 4 places.

For this problem of the best approximation of $|x|$, much sharper estimates are available than the general bound (4.1). Bernstein [4] proved that there exists a positive constant β such that

$$\lim_{n\to\infty} 2n\||x| - p^*\| = \beta,$$

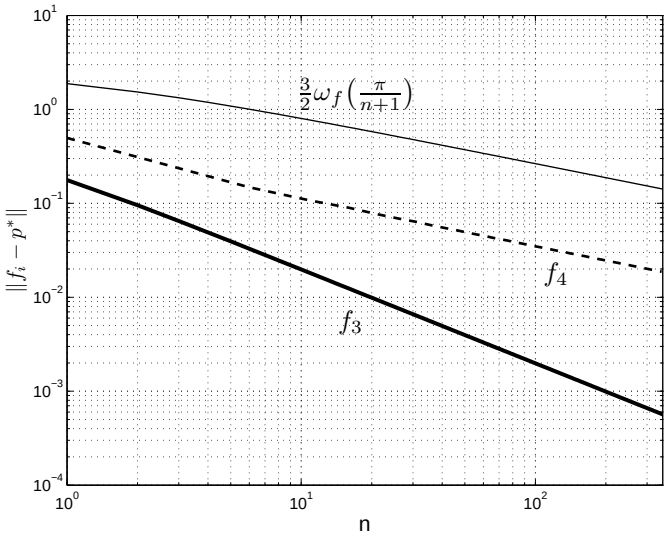

Figure 6: Error of best polynomial approximations for functions f_3 and f_4 compared with Jackson bound (4.2) for continuous functions. For f_4 the bound is asymptotically sharp, but for f_3 it is too pessimistic, since (4.2) does not take into account the difference between singularities of f at the endpoints and in the interior.

and from numerical experiments he conjectured that

$$\beta = \frac{1}{2\sqrt{\pi}} = 0.2820947917\ldots.$$

For seventy years this conjecture was open, until Varga and Carpenter [39] proved that it was false and confirmed this with extensive numerical computations. Among their experiments and results, which included sharper lower and upper bounds for β, they computed $2n\||x| - p^*\|$ for n up to 104, accurate to nearly 95 decimal places (since $|x|$ is an even continuous function, the errors for the best polynomial approximation of degree n and $n+1$ are the same so only even degrees are computed). Using the barycentric-Remez algorithm, we computed the same approximations in 30 seconds on a workstation, obtaining the same 52 errors as Varga and Carpenter to machine precision. We also computed the errors for degrees up to 10,000, and in Figure 7 we compare them with Bernstein's conjectured number, illustrating further that the conjecture was false.

5 Rational approximations

There are also Remez algorithms for best approximation by rational functions of type (m, n), that is, functions of the form $r = p/q$ where p and q are polynomials of degrees less than or equal to m and n respectively. The error equioscillates on a reference with $m+n+2-\delta$ points, where δ is the defect of f, and a de la Vallée Poussin inequality also holds. A Remez algorithm involves the computation of trial references and trial rational

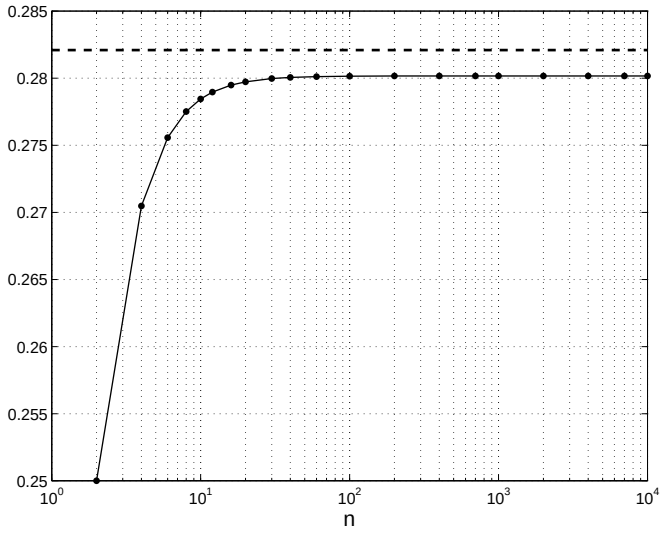

Figure 7: Computed values of $2n\||x| - p^*\|$ (solid) and Bernstein's conjectured number, $\beta = \frac{1}{2\sqrt{\pi}}$ (dashed) for values of n up to 10,000. As shown by Varga and Carpenter, Bernstein's conjecture was false.

functions much as for the polynomial case. The step of obtaining a trial reference of $m+n+2$ points (for the non-degenerate case) from the trial rational function is unaffected and requires one to find the global extremum or the alternating local extrema of the error function. Chebfun, again, makes this step straightforward. However, the computation of the trial rational function from the trial reference is more complex. A barycentric algorithm for this computation will be presented in a forthcoming paper.

Acknowledgments

We are grateful to Rodrigo Platte, Toby Driscoll, Nick Hale and Michael Floater for valuable comments and suggestions.

References

[1] M. Almacany, C. Dunham, and J. Williams, *Discrete Chebyshev approximation by interpolating rationals*, IMA J. Numer. Anal., 4 (1984), pp. 467–477.

[2] I. Barrodale and C. Phillips, *Solution of an overdetermined system of linear equations in the Chebyshev norm*, ACM Trans. Math. Software, 1 (1975), pp. 264–270.

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

[4] S. Bernstein, *Sur la meilleure approximation de $|x|$ par des polynomes de degrés donnés*, Acta. Math., 37 (1914), pp. 1–57.

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

[6] E. Borel, *Leçons sur les fonctions de variables réelles*, Gauthier-Villars, Paris, 1905.

[7] J. A. Boyd, *Computing zeros on a real interval through Chebyshev expansion and polynomial rootfinding*, SIAM J. Numer. Anal., 40 (2002), pp. 1666–1682.

[8] E. W. Cheney, *Introduction to Approximation Theory*, McGraw-Hill, 1966.

[9] P. C. Curtis and W. L. Frank, *An algorithm for the determination of the polynomial of best minimax approximation to a function defined on a finite point set*, J. ACM, 6 (1959), pp. 395–404.

[10] P. J. Davis, *Interpolation and Approximation*, Dover Publications, New York, 1975.

[11] C. J. de la Vallée Poussin, *Sur les polynomes d'approximation et la représentation approchée d'un angle*, Académie Royale de Belgique, Bulletins de la Classe des Sciences, 12 (1910).

[12] C. B. Dunham, *Choice of basis for Chebyshev approximation*, ACM Trans. Math. Softw., 8 (1982), pp. 21–25.

[13] G. H. Golub and L. B. Smith, *Algorithm 414: Chebyshev approximation of continuous functions by a Chebyshev system of functions*, Commun. ACM, 14 (1971), pp. 737–746.

[14] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, SIAM, Philadelphia, 2nd. ed., 2002.

[15] N. J. Higham, *The numerical stability of barycentric Lagrange interpolation*, IMA J. Numer. Anal., 24 (2004), pp. 547–556.

[16] B. Le Bailly and J. P. Thiran, *Computing complex polynomial Chebyshev approximants on the unit circle by the real Remez algorithm*, SIAM J. Numer. Anal., 36 (1999), pp. 1858–1877.

[17] G. G. Lorentz, *Approximation of Functions*, Holt, Rinehart and Winston, 1966.

[18] J. H. McClellan and T. W. Parks, *A personal history of the Parks-McClellan algorithm*, IEEE Signal Processing Magazine, 22 (2005), pp. 82–86.

[19] J. H. McClellan, T. W. Parks, and L. R. Rabiner, *A computer program for designing optimum FIR linear phase digital filters*, IEEE Trans. Audio Electroacoust., 21 (1973), pp. 506–526.

[20] G. Meinardus, *Approximation of Functions: Theory and Numerical Methods*, Springer, Heidelberg, 1967.

[21] H. N. Mhaskar and D. V. Pai, *Fundamentals of Approximation Theory*, Narosa Publishing House, New Delhi, 2000.

[22] F. D. Murnaghan and J. J. W. Wrench, *The determination of the Chebyshev approximating polynomial for a differentiable function*, Math. Tabl. Aids Comput., 13 (1959), pp. 185–193.

[23] R. Pachón, R. Platte, and L. N. Trefethen, *Piecewise smooth chebfuns.* IMA J. Numer. Anal., submitted.

[24] T. W. Parks and J. H. McClellan, *Chebyshev approximation for nonrecursive digital filters with linear phase*, IEEE Trans. Circuit Theory, 19 (1972), pp. 189–194.

[25] M. J. D. Powell, *Approximation Theory and Methods*, Cambridge University Press, Cambridge, UK, 1981.

[26] P. Rabinowitz, *Applications of linear programming to numerical analysis*, SIAM Review, 10 (1968), pp. 121–159.

[27] E. Y. Remez, *Sur la détermination des polynômes d'approximation de degré donnée*, Comm. Soc. Math. Kharkov, 10 (1934).

[28] E. Y. Remez, *Sur le calcul effectif des polynomes d'approximation de Tchebychef*, Compt. Rend. Acad. Sci., 199 (1934), pp. 337–340.

[29] E. Y. Remez, *Sur un procédé convergent d'approximations successives pour déterminer les polynômes d'approximation*, Compt. Rend. Acad. Sci., 198 (1934), pp. 2063–2065.

[30] J. R. Rice, *The Approximation of Functions*, vol. 1, Addison-Wesley, 1964.

[31] O. Rioul and P. Duhamel, *A Remez exchange algorithm for orthonormal wavelets*, IEEE Trans. Circuits Syst. II, 41 (1994), pp. 550–560.

[32] H. Schmitt, *Algorithm 409, discrete Chebychev curve fit*, Comm. ACM, 14 (1971), pp. 355–356.

[33] J. C. Simpson, *Fortran translation of algorithm 409, Discrete Chebychev curve fit*, ACM Trans. Math. Soft., 2 (1976), pp. 95–97.

[34] K.-G. Steffens, *The History of Approximation Theory: From Euler to Bernstein*, Birkhäuser, Boston, 2006.

[35] E. L. Stiefel, *Numerical methods of Tchebycheff approximation*, in On Numerical Approximation, R. Langer, ed., University of Wisconsin Press, Madison, 1959, pp. 217–232.

[36] R. Taylor and V. Totik, *Lebesgue constants for Leja points*, IMA J. Numer. Anal., to appear.

[37] L. N. Trefethen, *Square blocks and equioscillation in the Padé, Walsh, and CF tables*, in Rational Approximation and Interpolation, P. Graves-Morris, E. Saff, and R. Varga, eds., vol. 1105 of Lect. Notes in Math., Springer, 1984.

[38] L. N. Trefethen, *Spectral Methods in* MATLAB, SIAM, Philadelphia, 2000.

[39] R. S. Varga and A. J. Carpenter, *On the Bernstein conjecture in approximation theory*, Constr. Approx, 1 (1985), pp. 333–348.

[40] L. Veidinger, *On the numerical determination of the best approximation in the Chebyshev sense*, Numer. Math., 2 (1960), pp. 99–105.