BIT Numerical Mathematics (2008)46:000-000
DOI:10.1007/s10543-000-0000-x

BARYCENTRIC-REMEZ ALGORITHMS FOR BEST POLYNOMIAL APPROXIMATION IN THE CHEBFUN SYSTEM *

RICARDO PACHÓN[1] and LLOYD N. TREFETHEN[2]

[1] *Computing Laboratory, University of Oxford, Parks Rd., Oxford OX1 3QD, UK. email: ricp@comlab.ox.ac.uk*

[2] *Computing Laboratory, University of Oxford, Parks Rd., Oxford OX1 3QD, UK. email: LNT@comlab.ox.ac.uk*

Abstract.

The Remez algorithm, 75 years old, is a famous method for computing minimax polynomial approximations. Most implementations of this algorithm date to an era when tractable degrees were in the dozens, whereas today, degrees of hundreds or thousands are not a problem. We present a 21st-century update of the Remez ideas in the context of the chebfun software system, which carries out numerical computing with functions rather than numbers. A crucial feature of the new method is its use of chebfun global rootfinding to locate extrema at each iterative step, based on a recursive algorithm combining ideas of Specht, Good, Boyd, and Battles. Another important feature is the use of the barycentric interpolation formula to represent the trial polynomials, which points the way to generalizations for rational approximations. We comment on available software for minimax approximation and its scientific context, arguing that its greatest importance these days is probably for fundamental studies rather than applications.

Key words: Remez algorithm, best polynomial approximation, barycentric interpolation, chebfun system

AMS subject classification (2000): 41A50, 41A10, 65D05

1 Introduction

This article is devoted to a classic problem of best approximation: Given a continuous function f on the interval $I = [a, b]$, find a function p^* in the space $\mathcal{P}_n$ of polynomials of degree $\leq n$ 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 on I. 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 every book on approximation theory [12, 16, 26, 30, 31, 36, 41].

*Received ?. Revised ?. Communicated by ?.

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 [46]. 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 [38, 39, 40], 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 [35].

The 1970s are a long way back, but today's algorithmic techniques and available software for Remez calculations date mainly to that era. Concerning software, some items of note are

- ACM Algorithm 318, by Boothroyd in 1967 [8]
- ACM Algorithm 414, by Golub and Smith in 1971 [18]
- ACM Algorithm 604, by Sauer in 1983 [42]
- The Remes-difcor algorithm, by Kaufman, Leeming and Taylor [24]
- RATCH/DRATCH from the IMSL library [23]
- E02ACF from the NAG library [33]
- REMES and FIRPM from Matlab's Signal Processing Toolbox [27]

Most of these go back many years and in fact, most of them do not solve the general best approximation problem as posed above but variants involving discrete variables or digital filtering. One can find a few other computer programs in circulation, but overall, it seems that there is no widely-used program at present for computing best approximations. Thus this is a topic that is very widely known among numerical mathematicians and engineers, but without much of a computational footprint.

If one considers applications, apart from signal processing, it is clear that in the early decades of computers minimax approximations played a role in approximation of special functions. For example, both polynomial and rational minimax approximations were used in the design of FUNPACK in the 1970s [13], the predecessor of SPECFUN [14]. This kind of application, however, appears to have faded with the years as computers have become more powerful and desiderata of scope and portability have grown more important than getting the last percentage point of efficiency from an approximation.

We believe that even if best approximations are not at the heart of computational science nowadays, they are such a fundamental idea, so basic a tool of approximation theory, that is desirable to be able to compute them easily. In the present article we present an algorithm that achieves this, making it possible to compute a best aproximation with a single command. For example, here we find the degree 100 best approximation to $e^{|x|}$ in less than a second on a workstation:

```
>> f = chebfun('exp(abs(x))');
>> tic, p = remez(f,100); toc
  Elapsed time is 0.715704 seconds.
>> norm(f-p,inf)
  ans = 0.002801440898864
```

The further command `plot(f-p)` shows a beautiful curve with 103 points of equioscillation.

Our algorithm has five crucial features:

- Formulation within the chebfun system
- Barycentric Lagrange representation of polynomials
- Analytic formula for the levelled error
- Scaling barycentric weights by capacity
- Location of extrema by recursive Chebyshev rootfinding

The use of chebfun is new and brings many new angles to the problem, as we shall describe. (The reader can find expositions of the chebfun system in [4], [34] and `www.comlab.ox.ac.uk/chebfun`.) A barycentric representation was introduced previously by Parks and McLellan [35] for trigonometric approximations but may be new for algebraic polynomials, where the new feature of scaling by the logarithmic capacity of the approximation interval becomes crucial for robustness. Finally the method of localization of extrema at each iterative step is the most novel algorithmic feature of our method. In the past, parabolic and other approximations have been used which are fast but may miss extrema. The new approach relies instead on global representations of each error curve and global zerofinding, one of the hallmarks of the chebfun system. This makes possible an exceptionally high degree of reliability.

Section 2 presents the classical Remez algorithm, emphasizing its two main steps: the computation of a trial reference (Section 2.1) and a trial polynomial (Section 2.2). Section 3 introduces the new barycentric-Remez algorithm together with its chebfun implementation. The codes presented in this section 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 [52] 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 approximation.

2 Classical Remez algorithm

Since the best approximation is unique, we can define the operator that assigns to each continuous function its best polynomial approximation p^* of fixed degree. It is well known that this operator, although continuous, is nonlinear (for an example see [26, 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 and are used to solve the discrete version of this problem [2, 24, 37, 44]. 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 [9] [12, p. 75] [36, p. 77].

THEOREM 2.1 (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 $\{x_i\}_{i=0}^{n+1}$ in I such that*

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

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

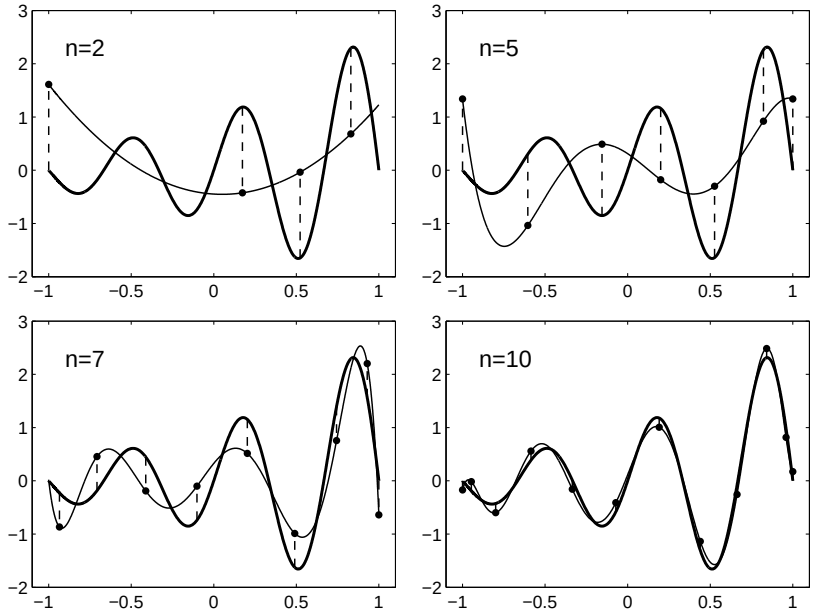

Figure 2.1: Best polynomial approximations (thin lines) of degrees 2, 5, 7 and 10 to $f(x) = \sin(3\pi x)\exp(x)$ (bold lines) on $[-1, 1]$. The dots show the polynomial at the reference and the dashed vertical bars the corresponding errors, of equal lengths and alternating in orientation.

A set of points $A^* := \{x_i\}_{i=0}^{n+1}$ that satisfies (2.1) is called a *reference*. Figure 2.1 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 equioscillation properties hold for best rational, CF and Padé approximations [49].

Theorem 2.1 can be generalized for approximations that satisfy the "Haar condition" [36, 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 [51], establishes an inequality between the alternating error of a trial polynomial and the error of the best approximation [12, p. 77][36, Thm. 7.7].

THEOREM 2.2 (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 2.2 asserts that a polynomial $p \in \mathcal{P}_n$ whose error oscillates $n+2$ times is "near-best" in the sense that

$$\|f - p\| \le C\|f - p^*\|, \quad C = \frac{\|f - p\|}{\min_i |f(y_i) - p(y_i)|} \ge 1. \tag{2.4}$$

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

$$f(x_i) - p_k(x_i) = \sigma_i h_k, \quad x_i \in A_k, \tag{2.5}$$

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^* [36, 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 (2.5) is satisfied. The conditions (2.5) amount to a linear system of $n+2$ equations in $n+2$ unknowns: $n+1$ parameters to describe the polynomial, plus the unknown h:

(2.6)
$$\begin{pmatrix} \phi_0(x_0) & \phi_1(x_0) & \cdots & \phi_n(x_0) \\ \phi_0(x_1) & \phi_1(x_1) & \cdots & \phi_n(x_1) \\ \vdots & \vdots & & \vdots \\ \phi_0(x_n) & \phi_1(x_n) & \cdots & \phi_n(x_n) \\ \phi_0(x_{n+1}) & \phi_1(x_{n+1}) & \cdots & \phi_n(x_{n+1}) \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} f(x_0) + \sigma_0 h \\ f(x_1) + \sigma_1 h \\ \vdots \\ f(x_n) + \sigma_n h \\ f(x_{n+1}) + \sigma_{n+1} h \end{pmatrix}.$$

We emphasize that this is the classical formulation. In the next section we shall recommend different methods for representing p and determining h.

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 Theorem 2.2.

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

$$|h_k| \le \min_i |f(y_i) - p_k(y_i)|, \ y_i \in A_{k+1}. \tag{2.7}$$

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 (2.7) 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 (2.7) 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 $f - p_k$.

Assuming f is twice differentiable, the error of the Remez algorithm decays at a quadratic rate if we measure error at every $n + 2$ steps in the case of the first Remez algorithm [36, Sec. 9.4] and at every step in the case of the second [53].

3 The new Remez algorithm and its chebfun implementation

We present our algorithm in six subsections. Each of the first five focuses on one of the main features that distinguishes it from previous work, and in the last one we discuss the possibility of using CF approximation as initial guess.

3.1 Formulation within the chebfun system

A major fact about this work is that our algorithm is not a standalone product but part of the chebfun system, which was introduced in its original form by Battles and Trefethen in 2004 [4] and has subsequently developed into a widely capable software system for computing with functions of one real variable. The principle of the chebfun system is that it works with functions themselves rather than with numbers, operating upon them with overloads of familiar Matlab commands for dealing with vectors such as `sum`, `abs`, `max` or `diff`. Each chebfun is represented by one or more polynomial interpolants through Chebyshev points on subintervals, enabling the system to compute rapidly and to nearly machine precision with almost any picewise smooth function on a bounded interval. See `http://www.comlab.ox.ac.uk/chebfun` for more information.

From a user's point of view, this framework changes the nature of possible Remez explorations fundamentally. The command `remez` in Figure 3.1, included in the chebfun system since version 2.0501, produces the chebfun `p` of the best polynomial approximation of degree `n` of a chebfun `f`. A single line of code like `f = chebfun('tanh(10*x)'); plot(f-remez(f,50))` produces an equioscillating error curve in a faction of a second. It is an easy matter quickly to compare best approximants on the fly of dozens of different degrees, or of dozens of functions.

The generality of the function representation in the chebfun system makes it possible to compute best approximations to even very complicated functions. The output `p` is of course a polynomial, and it is interesting that in this framework the input `f` is also a polynomial or a piecewise polynomial. This fact of implementation has important consequences for our method of finding extrema discussed in subsection 3.5.

```
function [p,err] = remez(f,n);        % compute deg n BA to chebfun f
iter = 1; delta = 1; deltamin = delta;
[a,b] = domain(f);
xk = chebpts(n+2); xo = xk;           % initial reference
sigma = (-1).^[0:n+1]';               % alternating signs
normf = norm(f);
while (delta/normf > 1e-14) & iter <= 20
  fk = feval(f,xk);                   % function values
  w = bary_weights(xk);               % compute barycentric weights
  h = (w'*fk)/(w'*sigma);             % levelled reference error
  if h==0, h = 1e-19; end             % perturb error if necessary
  pk = fk - h*sigma;                  % polynomial vals in the reference
  p=chebfun(@(x)bary(x,pk,xk,w),n+1); % chebfun of trial polynomial
  e = f - p;                          % chebfun of the error
  [xk,err] = exchange(xk,e,h,2);      % replace reference
  if err/normf > 1e5                  % if overshoot, recompute with one-
      [xk,err] = exchange(xo,e,h,1);  % point exchange
  end
  xo = xk;
  delta = err - abs(h);               % stopping value
  if delta < deltamin,                % store poly with minimal norm
      deltamin = delta;
      pmin = p; errmin = err;
  end
  iter = iter + 1;
end
p = pmin; err = errmin;
```

Figure 3.1: Code of the Remez algorithm in the chebfun system (slightly but only slightly simplified). The input arguments are a chebfun `f` and the degree `n` of the polynomial to be computed and the output arguments are a chebfun `p` of the best polynomial approximation to `f` and the error `err`.

3.2 Barycentric Lagrange representation of polynomials

The choice of basis $\{\phi_j\}$ for $\mathcal{P}_n$ is crucial in the numerical solution of (2.6) [17] for the computation of the trial polynomials. The monomial basis, for example, is a terrible choice: the condition number of the resulting Vandermonde matrix grows exponentially in general [21, 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].

A much better choice, however, is a Lagrange basis: Given a set $\{\tilde{x}_j\}_{j=0}^n$ of $n+1$ prescribed interpolation points, our representation of $p(x)$, $x \in I$, is mathematically equivalent to

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

Notice that

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

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 node distributions for which Lagrange interpolation is well conditioned have asymptotic density proportional to $(1 - s^2)^{-1/2}$, if s is a rescaling of x to $[-1, 1]$ [50, 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. Thus it is crucial that we actually work with (3.1) through the representation of the barycentric formula

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

where

$$\tilde{w}_j = \prod_{\nu \neq j} (\tilde{x}_j - \tilde{x}_\nu)^{-1}, \quad j = 0, \ldots, n,$$

are the barycentric weights, a formulation that 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 [22]. For a review of barycentric interpolation formulas, see [6].

3.3 Analytic formula for the levelled error

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 (2.6) 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

$$p(x_i) = f(x_i) + \sigma_i h, \quad i = 0, \ldots, n+1, \ i \neq j. \tag{3.5}$$

The formula (3.9) that we derive below allows one to compute the levelled error h independently of the values $p(x_i)$. Hence, we can compute the values $p(x_i)$ in (3.5) which in turn are used to construct the chebfun of the trial polynomial p with the barycentric formula. Since the barycentric-Remez algorithm does not have to solve system (2.6) but just to compute the values $p(x_i)$ from (3.5), ill-conditioning is not a problem as it may result when working with other basis for $\mathcal{P}_n$, even Chebyshev polynomials.

For any basis $\{\phi_j\}$ of $\mathcal{P}_n$ it can be shown that h can be found independently of the coefficients $\{c_j\}$ [36, 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 (3.5), $p(x_j)+\sigma_j h = f(x_j)$, and use Lagrange interpolation on the remaining set of $n+1$ points to compute $p(x_j)$,

$$\sum_{\substack{i=0\\ i\neq j}}^{n+1} p(x_i)\ell_i^j(x_j) + \sigma_j h = f(x_j), \quad x_j \in A,\ j = 0,\dots,n+1,\ j \neq i, \tag{3.6}$$

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

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

Notice that

$$\ell_i^j(x_j) = \prod_{\substack{\nu=0\\ \nu\neq i,j}}^{n+1} \frac{(x_j-x_\nu)}{(x_i-x_\nu)} = \frac{\prod_{\nu\neq j}(x_j-x_\nu)}{\prod_{\nu\neq i}(x_i-x_\nu)} \cdot \frac{x_i-x_j}{x_j-x_i} = -\frac{w_i}{w_j}, \tag{3.7}$$

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

$$w_j = \prod_{\substack{\nu=0\\ \nu\neq j}}^{n+1} (x_j-x_\nu)^{-1}, \quad x_j, x_\nu \in A \tag{3.8}$$

Inserting (3.7) in (3.6), 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 that is used to compute the levelled error in our algorithm,

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

It follows from (3.5) and (3.6) that by using this value of h, the trial polynomial p of degree n coincides with the polynomial interpolant through the $n+2$-point trial reference A. The barycentric weights of this polynomial are (3.8), the ones used to compute h in (3.9). In code listing of Figure 3.1, the chebfun of the trial polynomial is computed in the line

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

The chebfun command `bary` computes the interpolant at the point `x` using the barycentric formula for the vectors `pk`, `xk` and `w` of length $n+2$ with values $f(x_i) - \sigma_i h$, the trial reference $A = \{x_i\}_{i=0}^{n+1}$, and the barycentric weights (3.8), respectively.

3.4 Scaling barycentric weights by capacity

Our barycentric-Remez algorithm can not only compute p^* with very high degrees but also allows one to compute best approximations in the interval $[-10^6, 10^6]$ as easily as in the interval $[-1, 1]$. These features require the appropriate scaling of the barycentric weights w_j.

The scale of w_j in (3.8) for an arbitrary set of n nodes in an interval of length $4C$ grows at least at the rate C^{-n}, where the value C is known as the *logarithmic capacity* of the interval [6, p. 509]. If the length of the interval is 4, the standard formula for the barycentric weights can be used directly, even for very large n. However, when working with high degrees in intervals where the logarithmic capacity is far from one (for example when $n > 500$ and $\mathcal{I} = [-1, 1]$), the large values of w_j will cause overflow or underflow. To eliminate this risk, we multiply each difference in (3.8) by C^{-1}, scaling the barycentric weights roughly to size $O(1)$ and making them representable in floating point arithmetic.

When n is even larger (for example when $n > 5000$ and $\mathcal{I} = [-1, 1]$), the computation of w_j encounters a new problem: The partial products formed along the way when the products are evaluated, for example from left to right, may overflow or underflow. One solution to this problem would be to order the partial products to compensate, e.g. by a discrete Leja ordering [48]. However in this application we are able to use a simpler solution: rather than multiplying factors, we sum their logarithms. Following these two observations, the barycentric

weights (3.8) can be equivalently represented as

$$w_j = \frac{\displaystyle\prod_{\nu\neq j}\operatorname{sign}(x_j - x_\nu)}{\exp\Big(n\log C^{-1} + \displaystyle\sum_{\nu\neq j}\log\big|x_j - x_\nu\big|\Big)}, \quad j = 0,\dots,n.$$

This is the formula used by the chebfun command `bary_weights` to compute these values in an arbitrary interval $[a, b]$ of length $4C$.

3.5 Location of extrema by recursive Chebyshev rootfinding

The problem of updating the trial reference, as explained in Section 2.2, 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 context of Remez algorithms, a standard technique to compute these extrema [7] consists of constructing a parabola for each node x_k in the reference that interpolates the error at x_k and two other points. The vertex of the parabola is then exchanged with the lowest of the three values and the process starts again. Golub and Smith [18], for example, use this strategy in their Remez algorithm. Though they claim that the old nodes in the reference provide a good initial guess for the extrema, they acknowledge the possibility that the parabola method may not yield acceptable values, in which case they switch to a crude search method. They write *"most of the programming effort is involved in the locating the extrema of the error function $\epsilon(x)$"* [18].

In the chebfun system this step is handled completely differently. One of the main features in chebfun is the very efficient root finder `roots`, originally implemented by Battles [3]. Finding the roots of a smooth chebfun is equivalent to locating the roots of a polynomial in its Chebyshev form, which in turn can be done by computing the eigenvalues of a "colleague" matrix, the elements of which are Chebyshev coefficients. This method is well-conditioned and was discovered independently by Specht [45] and Good [19].

Although this strategy benefits from translating the problem to an eigenvalue problem for which very robust and fast algorithms exist, it is preferred to work with low-degree matrices arising from low-degree polynomials. Following the key idea by Boyd in [10], the `roots` command in chebfun recursively splits the polynomial into smaller subintervals until pieces of degree less or equal to one hundred are obtained. Then, the roots of each piece are computed as described above and collected with the roots of all the other pieces. For piecewise smooth chebfuns the strategy is the same, computing the roots of every smooth part and bookkeeping possible roots at breakpoints [34]. The subfunction `exchange` in Figure 3.2 replaces a trial reference using the command `roots` and the rules presented in Section 2.2.

In the chebfun system, locating the global extremum of the error function requires the same computational effort as locating all the local extrema, both of them using the `roots` command. Hence, our implementation of the second

```
function [xk,norme] = exchange(xk,e,h,method)
rr = [-1; roots(diff(e)); 1];           % critical pts of the error
if method == 1                          % one-point exchange
    [tmp,pos] = max(abs(feval(e,rr))); pos = pos(1);
else                                    % full exchange
    pos = find(abs(feval(e,rr))>=abs(h)); % vals above leveled error
end
[r,m] = sort([rr(pos); xk]);
er = [feval(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);
```

Figure 3.2: Code of the exchange algorithm in the chebfun system (again slightly simplified). The input arguments are a column vector `xk` with the trial reference A_k, the chebfun `e` of the error $f - p_k$, the levelled error `h` and a number `method`, with values 1 or 2 prescribing the use of the first or the second Remez algorithm respectively. The output arguments are the modified vector `xk` with the new trial reference A_{k+1} and the norm `norme` of the new associated error.

Remez algorithm is usually much faster than the first algorithm since the later clearly needs many more iterations. However, we have seen that for certain functions and values of n, one of the steps in the second Remez algorithm constructs a trial polynomial with a very large extremum, located usually near the endpoints. The large norm of the polynomial introduces a very large error in h, breaking the computation. A simple way to solve this is to reverse the last step and recompute the trial reference but using the one-point exchange of the first Remez algorithm. We have seen from our experiments that this strategy usually solves the problem.

3.6 CF approximation for initial guess

Since the Remez algorithm is nonlinear, the question arises as to a suitable initial guess: an initial choice of the reference x_k. One interesting possibility here is to start by computing a near-best approximation to f by the Carathéodory-Fejer (CF) method and derive x_k from this. The CF method, based on the singular value analysis of a Hankel matrix of Chebyshev coefficients, goes back to 1982 [20] and is implemented in a command `cf` in the chebfun system, so such

a variation is easy to implement. Since the CF method is not iterative apart from the singular value computation, it is up to 100 times faster than the Remez computation.

For smooth functions f and values of n bigger than 1 or 2, CF approximants are often so close to best approximants that there may be no point in executing the Remez algorithm at all. In the `remez` code, however, we have decided against using the CF method to generate an initial guess. Instead we rely on the initial guess that has been the standard choice throughout the history of the Remez algorithm: the Chebyshev points (3.3). This choice ensures that the first step of the process executes correctly and that only between 5 and 10 Remez steps are typically needed before the difference $\|f-p_k\|-|h_k|$ becomes sufficiently small, so any improvement from introducing a more complicated CF initial guess would at best be rather modest.

Notice that 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 as explained in Section 2.2.

4 Numerical experiments

We can use the function `remez` to easily compute best polynomial approximations in the chebfun system. As an illustration, we use it to compute $p^* \in \mathcal{P}_{10}$ for the functions in Table 4.1. Numbers are given to 14 digits in the Table to aid readers who may wish to make comparisons with our results. The approximations are plotted in Figure 4.1.

Table 4.1: Best approximation errors for nine functions by polynomials of degree 10.

i	f_i	$\|f_i-p_0\|$	$\|f_i-p^*\|$
1	$\tanh(x+0.5)-\tanh(x-0.5)$	0.00000058780531	0.00000030009195
2	$\sin(\exp(x))$	0.00000386118470	0.00000178623400
3	$\sqrt{x+1}$	0.04212512276261	0.01978007008380
4	$\sqrt{\lvert x-0.1\rvert}$	0.30512512446096	0.11467954016268
5	$1-\sin(5\lvert x-0.5\rvert)$	0.40947166876230	0.14320591977421
6	$\min\{\mathrm{sech}(3\sin(10x)),\sin(9x)\}$	0.71216404197963	0.33561414233366
7	$\max\{\sin(20x),\exp(x-1)\}$	0.77453305461326	0.38723296760148
8	$\mathrm{sech}\big(10(0.5x+0.3)\big)^2+$ $\mathrm{sech}\big(100(0.5x+0.1)\big)^4+$ $\mathrm{sech}\big(1000(0.5x-0.1)\big)^6$	1.08706818322313	0.49987078860783
9	$\log(1.0001+x)$	2.98370118052234	1.40439492981387

Table 4.1 also includes the error of the polynomial interpolant p_0 through 11 Chebyshev points, which served as the initial trial polynomial for the Remez algorithm, and the error of the best approximation p^*. Although the former is

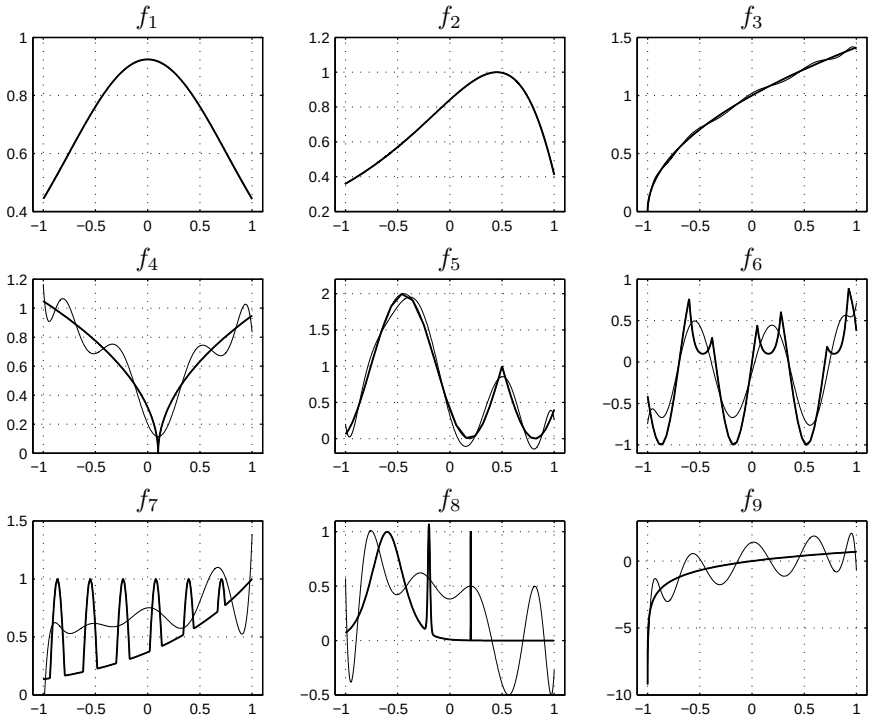

Figure 4.1: Best polynomial approximations of degree 10 (thin lines) to the functions in Table 4.1 (bold lines).

obviously always larger than the later, we can see in these examples that the improvement due to the Remez algorithm is less than a factor of three, and in general, for any continuous function and $n = 10$, it will always be less than a factor of 3.47. This is a consequence of the fact that interpolants in n Chebyshev points are "near best" as in (2.4) with $C = 2 + \frac{2}{\pi} \log n$ [11].

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 $n = 5, 7, 9, 11$ [39]. Twenty-five years later, Stiefel [47], Curtis and Frank [15] and Murnaghan and Wrench [32] applied different techniques to compute best approximations of $\sin^{-1} x$, $\tan^{-1} x$, $\log x$, 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 [18] and FORTRAN codes by Barrodale and Phillips [2], and by Simpson [44] based on Schmitt's algorithm [43]. They have been used, for example, for Chebyshev curve fitting with $n > 20$. More recently, Le Bailly and Thiran [25] 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. Most remarkably for its era, McClellan and Parks [28] comment on experiments they did thirty years ago in their work with Rabiner [29] 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 in seconds or at most minutes, using the barycentric-Remez algorithm. Compared to other implementations of the Remez algorithm where the construction of the system (2.6) would not be feasible, for example using Chebyshev polynomials as the basis for $\mathcal{P}_n$, the barycentric-Remez algorithm never has conditioning problems and allows one to make experiments of these very high degrees.

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 $|f(x_1) - f(x_2)| \leq C|x_1 - x_2|$, $x_1, x_2 \in [-1, 1]$, then the approximation error is bounded by [36, Thm. 16.5]

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

The functions f_5, f_6, f_7 and f_8 in Table 4.1 (among others) are Lipschitz continuous. Using the chebfun command `norm(diff(f),inf)` we calculated the Lipschitz coefficients, and Figure 4.2 shows the best approximation errors for n between 1 and 10,000 and the bound (4.1) for f_5 and f_8. The large error of the best approximation to f_8 in Figure 4.2 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 [36, Thm. 16.5]

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

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 4.3 we compare this quantity with the best approximation errors for n from 1 to 1,000.

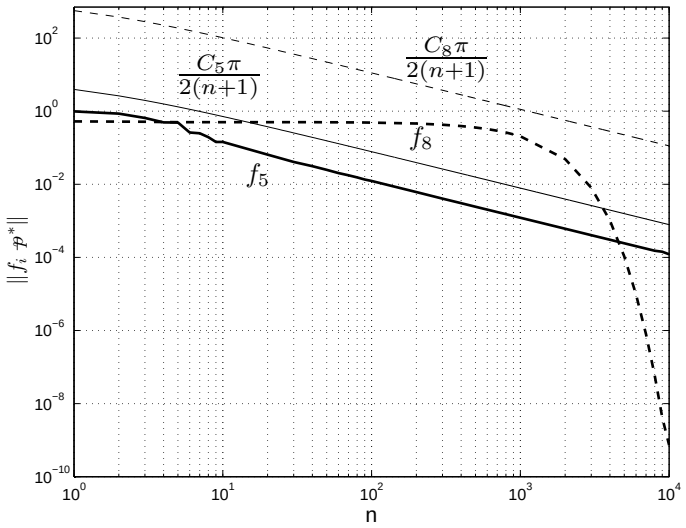

Figure 4.2: Error of best polynomial approximations for functions f_5 and f_8 (bold lines) compared with the Jackson bounds (4.1) for Lipschitz continuous functions (thin lines). The errors for f_6 and f_7, not shown, follow closely the error for f_5.

Remez used his algorithm to compute the best approximations to $|x|$ in $[-1, 1]$ by polynomials of degrees up to 11 with an accuracy of 10^{-5}. As reported in one of his 1934 papers [39], the computations were carried out by three female students of the University of Kiev and used the equivalence between best approximants to $|x|$ and $\sqrt{x}$. 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 odd degrees were computed. Using the overloaded command `poly(p)`, where `p` is the chebfun of the best polynomial approximation, we can check the coefficients published in that paper. 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 [39])	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 best approximation of $|x|$, much sharper estimates are

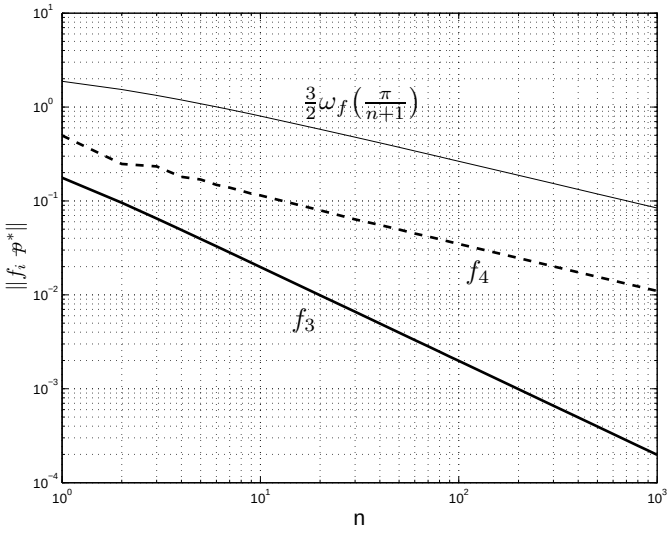

Figure 4.3: Error of best polynomial approximations for functions f_3 and f_4 (bold lines) compared with Jackson bound (4.2) for continuous functions (thin lines). For f_3 the bound is too pessimistic, since (4.2) does not take into account the difference between singularities of f at the endpoints and in the interior.

available than the general bound (4.1). Bernstein [5] proved that there exists a positive constant β such that

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

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 [52] 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 $n\|\,|x| - p^*\|$ for n up to 104, accurate to nearly 95 decimal places.

The method of Varga and Carpenter requires the use of extended precision. Using the barycentric-Remez algorithm, we were able to compute the same approximations in 30 seconds on a workstation in ordinary IEEE arithmetic, obtaining the same 52 errors as Varga and Carpenter to 15 digits. We also computed the polynomial approximations for degrees up to 1,500 and confirmed the first seven of the fifty digits of β that Varga and Carpenter computed by Richardson extrapolation:

$$\beta \approx 0.2801694\ldots$$

We can compute best approximants of higher degrees using the barycentric-Remez algorithm, and in Figure 4.4 we compare the errors with Bernstein's conjectured number up to $n = 10{,}000$, illustrating further — as if further ilustration were needed! — that the conjecture was false. However, as we mentioned in Section 3, polynomial interpolation in arbitrary sets of nodes, including the references in the barycentric-Remez algorithm when these are far from Chebyshev points, may not be well-conditioned. For large n this causes equation (3.5) to have a significant error. In Figure 4.4, the values for $n > 1500$ are accurate to 4 decimal places.

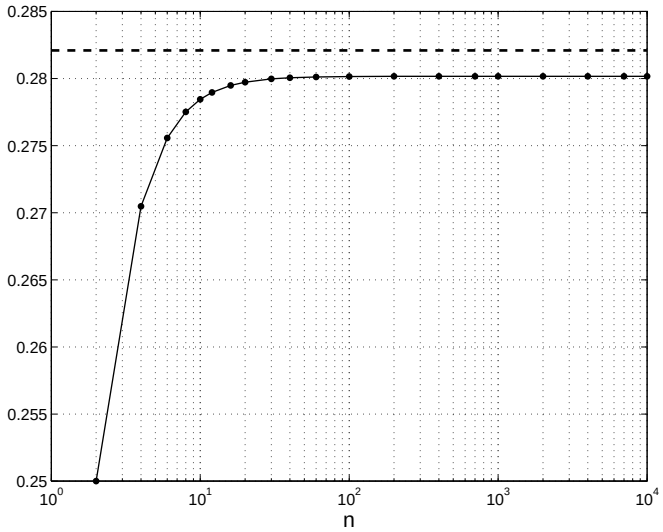

Figure 4.4: Computed values of $n\||x|-p^*\|$ (solid) and Bernstein's conjectured number, $\beta = \frac{1}{2\sqrt{\pi}}$ (dashed) for values of n up to 10,000. As shown by Varga and Carpenter, Bernstein's conjecture was false.

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 a number known as 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 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 Toby Driscoll, Michael Floater, Nick Hale, Michael Overton, Rodrigo Platte and Ian Sobey for valuable comments and suggestions.

REFERENCES

1. Almacany, M., Dunham, C., Williams, J.: Discrete Chebyshev approximation by interpolating rationals. IMA J. Numer. Anal. **4**, 467–477 (1984)
2. Barrodale, I., Phillips, C.: Solution of an overdetermined system of linear equations in the Chebyshev norm. ACM Trans. Math. Software **1**, 264–270 (1975)
3. Battles, Z.: Numerical linear algebra for continuous functions. Ph.D. thesis, University of Oxford (2005)
4. Battles, Z., Trefethen, L.N.: An extension of MATLAB to continuous functions and operators. SIAM J. Sci. Comput. **25**(5), 1743–1770 (2004)
5. Bernstein, S.: Sur la meilleure approximation de $|x|$ par des polynomes de degrés donnés. Acta. Math. **37**, 1–57 (1914)
6. Berrut, J.P., Trefethen, L.N.: Barycentric Lagrange interpolation. SIAM Review **46**, 501–517 (2004)
7. de Boor, C., Rice, J.R.: Extremal polynomials with application to Richardson iteration for indefinite linear systems. SIAM J. Sci. Stat. Comput **3**, 47–57 (1982)
8. Boothroyd, J.: Algorithm 318: Chebyschev curve-fit. Commun. ACM **10**(12), 801–803 (1967)
9. Borel, E.: Leçons sur les fonctions de variables réelles. Gauthier-Villars, Paris (1905)
10. Boyd, J.A.: Computing zeros on a real interval through Chebyshev expansion and polynomial rootfinding. SIAM J. Numer. Anal. **40**(5), 1666–1682 (2002)
11. Brutman, L.: Lebesgue functions for polynomial interpolation — a survey. Ann. Numer. Math. **4**, 111–128 (1997)
12. Cheney, E.W.: Introduction to Approximation Theory. McGraw-Hill (1966)
13. Cody, W.J.: The FUNPACK package of special function subroutines. ACM Trans. Math. Softw. **1**(1), 13–25 (1975)
14. Cody, W.J.: Algorithm 715: SPECFUN – a portable FORTRAN package of special function routines and test drivers. ACM Trans. Math. Softw. **19**(1), 22–30 (1993)
15. Curtis, P.C., Frank, W.L.: An algorithm for the determination of the polynomial of best minimax approximation to a function defined on a finite point set. J. ACM **6**, 395–404 (1959)

16. Davis, P.J.: Interpolation and Approximation. Dover Publications, New York (1975)
17. Dunham, C.B.: Choice of basis for Chebyshev approximation. ACM Trans. Math. Softw. **8**(1), 21–25 (1982)
18. Golub, G.H., Smith, L.B.: Algorithm 414: Chebyshev approximation of continuous functions by a Chebyshev system of functions. Commun. ACM **14**(11), 737–746 (1971)
19. Good, I.J.: The colleague matrix, a Chebyshev analogue of the companion matrix. Quart. J. Math. **12**, 61–68 (1961)
20. Gutknecht, M.H., Trefethen, L.N.: Real polynomial Chebyshev approximation by the Carathéodory-fejér method. SIAM J. Numer. Anal. **19**, 358–371 (1982)
21. Higham, N.J.: Accuracy and Stability of Numerical Algorithms, 2nd. edn. SIAM, Philadelphia (2002)
22. Higham, N.J.: The numerical stability of barycentric Lagrange interpolation. IMA J. Numer. Anal. **24**, 547–556 (2004)
23. IMSL Numerical Libraries: Technical documentation. Visual Numerics Inc., Houston, Texas
24. Kaufman, Jr., E.H., Leeming, D.J., Taylor, G.D.: Uniform rational approximation by differential correction and Remes-differential correction. Internat. J. Numer. Methods Engrg. **17**, 1273–1278 (1981)
25. Le Bailly, B., Thiran, J.P.: Computing complex polynomial Chebyshev approximants on the unit circle by the real Remez algorithm. SIAM J. Numer. Anal. **36**, 1858–1877 (1999)
26. Lorentz, G.G.: Approximation of Functions. Holt, Rinehart and Winston (1966)
27. MATLAB: User's Guide. The MathWorks Inc., Natick, Massachusetts
28. McClellan, J.H., Parks, T.W.: A personal history of the Parks-McClellan algorithm. IEEE Signal Processing Magazine **22**, 82–86 (2005)
29. McClellan, J.H., Parks, T.W., Rabiner, L.R.: A computer program for designing optimum FIR linear phase digital filters. IEEE Trans. Audio Electroacoust. **21**, 506–526 (1973)
30. Meinardus, G.: Approximation of Functions: Theory and Numerical Methods. Springer, Heidelberg (1967)
31. Mhaskar, H.N., Pai, D.V.: Fundamentals of Approximation Theory. Narosa Publishing House, New Delhi (2000)
32. Murnaghan, F.D., J. W. Wrench, J.: The determination of the Chebyshev approximating polynomial for a differentiable function. Math. Tabl. Aids Comput. **13**, 185–193 (1959)
33. NAG: Library Manual. The Numerical Algorithms Group Ltd., Oxford, UK

34. Pachón, R., Platte, R., Trefethen, L.N.: Piecewise smooth chebfuns. IMA J. Numer. Anal., to appear

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

36. Powell, M.J.D.: Approximation Theory and Methods. Cambridge University Press, Cambridge, UK (1981)

37. Rabinowitz, P.: Applications of linear programming to numerical analysis. SIAM Review **10**, 121–159 (1968)

38. Remes, E.: Sur la détermination des polynomes d'approximation de degré donnée. Comm. Soc. Math. Kharkov **10** (1934)

39. Remes, E.: Sur le calcul effectif des polynomes d'approximation de Tchebychef. Compt. Rend. Acad. Sci. **199**, 337–340 (1934)

40. Remes, E.: Sur un procédé convergent d'approximations successives pour déterminer les polynomes d'approximation. Compt. Rend. Acad. Sci. **198**, 2063–2065 (1934)

41. Rice, J.R.: The Approximation of Functions, vol. 1. Addison-Wesley (1964)

42. Sauer, F.W.: Algorithm 604: A FORTRAN program for the calculation of an extremal polynomial. ACM Trans. Math. Softw. **9**(3), 381–383 (1983)

43. Schmitt, H.: Algorithm 409, discrete Chebychev curve fit. Comm. ACM **14**, 355–356 (1971)

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

45. Specht, W.: Die Lage der Nullstellen eines Polynoms. IV. Math. Nachr. **21**, 201–222 (1960)

46. Steffens, K.G.: The History of Approximation Theory: From Euler to Bernstein. Birkhäuser, Boston (2006)

47. Stiefel, E.L.: Numerical methods of Tchebycheff approximation. In: R. Langer (ed.) On Numerical Approximation, pp. 217–232. University of Wisconsin Press, Madison (1959)

48. Taylor, R., Totik, V.: Lebesgue constants for Leja points. IMA J. Numer. Anal., to appear

49. Trefethen, L.N.: Square blocks and equioscillation in the Padé, Walsh, and CF tables. In: P. Graves-Morris, E. Saff, R. Varga (eds.) Rational Approximation and Interpolation, *Lect. Notes in Math.*, vol. 1105. Springer (1984)

50. Trefethen, L.N.: Spectral Methods in MATLAB. SIAM, Philadelphia (2000)

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

52. Varga, R.S., Carpenter, A.J.: On the Bernstein conjecture in approximation theory. Constr. Approx **1**, 333–348 (1985)

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