A robust and scalable implementation of the Parks-McClellan algorithm for designing FIR filters

Silviu-Ioan Filip

A ROBUST AND SCALABLE IMPLEMENTATION OF THE PARKS-MCCLELLAN ALGORITHM FOR DESIGNING FIR FILTERS

SILVIU-IOAN FILIP

Abstract. With a long history dating back to the beginning of the 1970s, the Parks-McClellan algorithm is probably the most well-known alternative for designing finite impulse response filters. Despite being a standard routine in many signal processing packages, it is possible to find practical design specifications where existing codes fail to work. Our goal is twofold: (1) examine and present solutions for the practical difficulties related to weighted minimax polynomial approximation problems on multi-interval domains (*i.e.,* the general setting under which the Parks-McClellan algorithm operates), and (2) using them, describe a robust implementation which outperforms existing alternatives.

1. Introduction

Digital filtering operations are essential to many engineering applications, which is why, over the last decades, they have enjoyed a sustained interest from researchers in Computer Science and Electrical Engineering. In general, a filtering framework consists of three major steps:

- specify and determine a mathematical representation of the filter (usually in terms of polynomials/rational functions);
- quantize the values (*i.e.*, coefficients) found at the previous step, using some *imposed* numerical representations (be it fixed-point or floating-point formats) [31];
- synthesize the obtained filter in hardware/software [20].

The Parks-McClellan exchange algorithm [40] solves the first step. It is an iterative procedure converging to the *optimal* polynomial filter with real valued coefficients satisfying some given constraints.

Current implementations of the Parks-McClellan algorithm suffer from robustness issues. A typical instance where codes such as those found in MATLAB's Signal Processing toolbox, GNURadio or Scipy fail to work, is when computing a minimax filter of degree $n = 100$ adhering to the specification from Example 1.2, which will be given shortly. Such a degree is plausible in practice, yet it proves difficult to design. This unsatisfactory situation was the initial motivation of the present study.

The starting point of our work is [39], which describes a robust implementation of the Remez exchange algorithm for computing best polynomial approximations on a *compact interval*, as part of the Chebfun [21] MATLAB library (the `remez` command). The main elements that allow Chebfun `remez` to perform well in practice are: (1) the use of Chebyshev nodes (see Section 5.2.2 for a definition) as starting reference, (2) barycentric Lagrange interpolation [7] and (3) a Chebyshev-proxy root-finding method [12, 57] based on the recursive subdivision of the approximation domain. The Parks-McClellan algorithm computes minimax approximations in a *weighted multi-interval* context and can therefore be viewed as an extension of the case addressed by [39]. Unfortunately, adapting this implementation to our context is not at all straightforward and raises several significant issues. The present work addresses all of them and, to the best of our knowledge, it describes the first fast, robust and scalable implementation of the Parks-McClellan algorithm. Note also that, despite a central focus on filtering applications (the traditional setting for the Parks-McClellan algorithm), our code can compute more general weighted minimax polynomial approximations over a multi-interval compact subset of $\mathbb{R}$ (see Section 7.3).

We begin with an overview of how minimax approximation algorithms are related to digital filter design (Section 2) and practical difficulties in using them (Section 3). The main problem in writing an efficient implementation in our context is finding a *suitable* initial reference. In particular, there is no *simple*, *closed-form* equivalent for Chebyshev nodes in this setting (in the sense that interpolation at those points is *close* to the optimal approximation). Fast practical convergence of the exchange algorithm is very much dependent on placing the right number of reference points inside each interval of the domain. In Section 4, we describe two *complementary* heuristic solutions to this initialization problem. Our experiments show that they are quite satisfactory in practice. Section 5 looks at how initialization choices affect execution. We also identify the possible sources of numerical instability at runtime, argue how they are related to each other and how to *counter* them. Central to our study is the *Lebesgue constant* (see Section 4.1 for a definition) that we use, in an apparently novel way, as a tool for detecting numerical problems.

Another important difference with the Chebfun `remez` code is the fact that we eliminate recursion during root-finding by using problem-specific information about the root positions. Major advantages are a smaller number of computations compared to a recursive search and the possibility of introducing parallelism (see Sections 6 and 7), further speeding up execution on multicore systems.

The resulting code[1] allows one to successfully address problematic instances in a scalable manner (see the examples from Sections 7.2 and 7.3). We offer three versions of our implementation. The first one uses IEEE-754 double-precision floating-point arithmetic, while the second one requires `long double` operations. On x86 machines and compilers, this format corresponds to 80-bit floating-point numbers. The other version uses `MPFR` [26] multiple precision formats and serves mostly as a verification tool for our `double` and `long double` versions. All of the numerical tests in the sequel (and many others) are included with the source code. The double-precision version is sufficient for *most* practical uses. In our tests, only computations involving Example 1.4 required `long double` operations to obtain an accurate result.

Throughout the text, we will be frequently using four filter design problems to highlight our approach. They are taken and/or adapted from the literature and prove to be hard or impossible to solve using current *de facto* minimax implementations. The theory needed to understand them is briefly reviewed in the next section.

We start with two examples used in [44].

Example 1.1. A unit weight type I lowpass filter with $[0, 0.4\pi]$ passband and $[0.5\pi, \pi]$ stopband. Multiple degrees n are considered.

Example 1.2. A type I, uniformly weighted, bandstop filter with passbands $[0, 0.2\pi]$, $[0.6\pi, \pi]$ and stopband $[0.3\pi, 0.5\pi]$. Several degrees n are used.

The next problem comes from [60] and is related to the design of equiripple comb FIR filters.

Example 1.3. A unit weight type I linear phase filter with passband $[0, 0.99\pi]$, stopband centered at π and degree $n = 520$.

Our last example is derived from [1]. There, the exchange algorithm acts as an intermediary step in designing efficient wideband channelizers. The lengths of the filters they use are proportional to the number of desired channels. In order to design a 8192-subchannel system, we can use:

Example 1.4. A degree $n = 13 \cdot 4096 = 53248$ unit weight type II lowpass filter with passband $\left[0, \frac{1}{8192}\pi\right]$ and stopband $\left[\frac{3}{8192}\pi, \pi\right]$.

2. An overview of minimax FIR filter design

A finite impulse response (FIR) filter with coefficients $\{b_k\}_{k=-n_1}^{n_2}$ is generally designed using a trigonometric polynomial $H(\omega)$ (the *frequency response* of the filter) of the form:

$$H(\omega) = \sum_{k=-n_1}^{n_2} b_k e^{-ik\omega} \tag{1}$$

Many problems in signal processing (for instance in data transmission, audio and image processing) require the b_k's to have real values and be (anti)symmetric around the index $k = 0$ (so-called *linear-phase* filters). Depending on the type of symmetry used, there are 4 major types (I to IV) of such filters [38, Ch. 5.7.3]. Without any loss of generality, we will focus on type I filters, since type II to IV instances can be designed analogously [6, Ch. 15.8–15.9].

2.1. Optimal FIR filters and weighted polynomial approximation.

Type I filters are characterized by $n_1 = n_2 = n$ in (1), where $b_{-k} = b_k, k = 1, \ldots, n$. We then have the equivalent form

$$H(\omega) = \sum_{k=0}^{n} h_k \cos(\omega k), \tag{2}$$

where $h_0 = b_0$ and $h_k = 2b_k, 1 \leqslant k \leqslant n$.

We address the following:

Problem 2.1 (Equiripple (or minimax) FIR filter design)**.** *Let Ω be a compact subset of $[0, \pi]$ and $D(\omega)$ an ideal frequency response, continuous on Ω. For a given **filter degree** $n \in \mathbb{N}$, we want to determine $H(\omega) = \sum_{k=0}^{n} h_k \cos(\omega k)$ such that the weighted error function $E(\omega) = W(\omega)\left(D(\omega) - H(\omega)\right)$ has **minimum** uniform norm*

$$\|E(\omega)\|_{\infty,\Omega} = \sup_{\omega\in\Omega} |E(\omega)|,$$

where the weight function W is continuous and strictly positive over Ω.

[1] See https://github.com/sfilip/firpm for the accompanying C++ code

Remark 2.2. By the change of variable $x = \cos(\omega)$, $H(\omega)$ is a polynomial [38, Ch. 7.7] in x of degree at most n. Problem 2.1 is thus equivalent to weighted minimax polynomial approximation over a compact subset of $[-1, 1]$.

The solution has a *qualitative* description:

Theorem 2.3 (Alternation theorem)**.** *A necessary and sufficient condition for $H(\omega)$ to be the **unique** transfer function of degree at most n that minimizes the weighted approximation error $\delta_\Omega = \|E(\omega)\|_{\infty,\Omega}$ is that $E(\omega)$ exhibit **at least** $n+2$ equioscillating extremal frequencies over Ω;* i.e.*, there exist at least $n+2$ values ω_k in Ω such that $\omega_0 < \omega_1 < \ldots < \omega_{n+1}$ and*

$$E(\omega_k) = -E(\omega_{k+1}) = \lambda(-1)^k \delta_\Omega, \qquad k = 0, \ldots, n,$$

where $\lambda \in \{\pm 1\}$ is fixed.

Proof. See [17, Ch. 3.4]. □

2.2. **The Parks-McClellan algorithm.** Together with a result due to de La Vallée Poussin [17, Ch. 3.4], Theorem 2.3 has been used by Remez [46] to derive iterative procedures which converge to the minimax result [17, Ch. 3.8] [42, Ch. 8–10]. Parks and McClellan [40] were the first to give a successful Remez algorithm implementation for solving Problem 2.1:

Parks-McClellan (Remez) exchange algorithm

Input: the filter degree n, the ideal frequency response D, the weight function W, the frequency set of interest Ω, convergence parameter threshold ε_t

Output: the $(h_k)_{0 \leqslant k \leqslant n}$ coefficients of the final frequency response H

(1) **Initialization**: Choose a reference vector of frequencies $\boldsymbol{\omega} = (\omega_k)_{0 \leqslant k \leqslant n+1}$.

(2) **Finite Set Approximation & Interpolation**: Find the current frequency response $H(\omega)$ and its associated alternating error $\delta_{\boldsymbol{\omega}}$, which correspond to solving Problem 2.1 on the elements of $\boldsymbol{\omega}$.

(3) **Extrema Search**: Determine the current error function $E(\omega)$ and find its local extrema over Ω, where $|E(\omega)| \geqslant \delta_{\boldsymbol{\omega}}$.

(4) **Reference Set Update**: Compute a new reference vector $\boldsymbol{\omega'} = (\omega'_k)_{0 \leqslant k \leqslant n+1}$ from the set of potential extremas found at Step 3 by picking $n+2$ that include the global extrema of $E(\omega)$ and for which the error alternates in sign.

(5) **Convergence Parameter Test**: If $\frac{\max|E(\omega'_k)| - \min|E(\omega'_k)|}{\max|E(\omega'_k)|} \leqslant \varepsilon_t$, return the filter characterized by the set of equioscillating frequencies $\boldsymbol{\omega'}$. If not, **go to** Step 2, with $\boldsymbol{\omega} = \boldsymbol{\omega'}$ as the new reference vector.

Remark 2.4. Such a routine is usually called a multipoint exchange algorithm (or second Remez algorithm). Remarks about its convergence are made in [35], while in [6, Ch 15.4] it is mentioned that, in practice, up to 12 iterations (Steps 2 to 4) are usually required for designing two and three-band FIR filters. In particular, [35] states that convergence is *ensured* if Ω is a finite set of points. This assumption fits the setting of effectively implementing the algorithm, since Ω will technically be the set of all floating-point values included in the actual frequency intervals of interest. Since we do not use this detail in the rest of the text, we will consider Ω as a finite union of closed intervals.

Remark 2.5. Because of the equivalence to polynomial approximation, for much of the paper we will be using the set $\mathbf{x} = (x_k)_{0 \leqslant k \leqslant n+1} = (\cos(\omega_k))_{0 \leqslant k \leqslant n+1}$ in place of $\boldsymbol{\omega}$. We denote with X, where $X \subseteq [-1, 1]$, the transformed frequency bands in this case. When talking about x, it will be as $x = \cos(\omega)$. Also, we consider $\delta_{\mathbf{x}} = \delta_{\boldsymbol{\omega}}$ and $\delta_X = \delta_\Omega$.

2.3. **Implementations.** The first implementation of the Parks-McClellan algorithm [36], was written in Fortran. It works on a modified version of the filter design problem. At each iteration, the new extremal set $\boldsymbol{\omega'}$ from Step 5 is chosen from a dense grid G of uniformly spaced points in Ω. The size of G is taken large enough (the initial value [40] was $20n$), so that the obtained solution will be close to the one over Ω. Computing $\boldsymbol{\omega'}$ is done by an exhaustive evaluation of $E(\omega)$ over G.

This early Fortran program is a starting point for current codes, like the `firpm` function from MATLAB's Signal Processing Toolbox and the variations from [3], or similar routines from Scilab, SciPy and GNURadio.

2.4. **The role of transition bands.** A typical use case for the exchange method is the design of lowpass filters, like the one in Figure 1, with $\Omega = [0, \omega_p] \cup [\omega_s, \pi]$,

$$D(\omega) = \begin{cases} 1, & 0 \leqslant \omega \leqslant \omega_p, \\ 0, & \omega_s \leqslant \omega \leqslant \pi, \end{cases} \quad \text{and} \quad W(\omega) = \begin{cases} \frac{\delta_2}{\delta_1}, & 0 \leqslant \omega \leqslant \omega_p, \\ 1, & \omega_s \leqslant \omega \leqslant \pi, \end{cases}$$

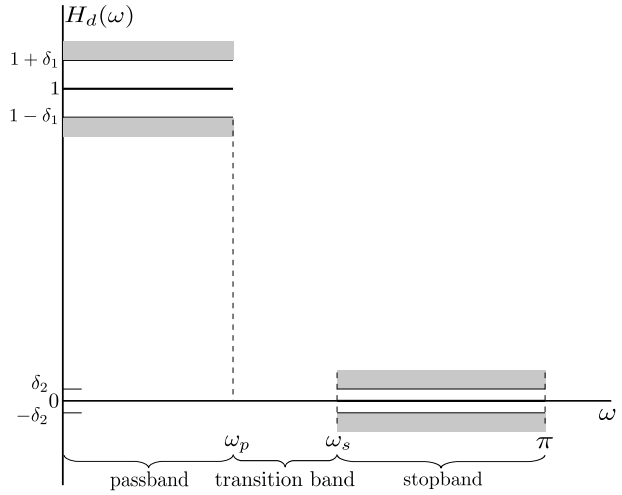

FIGURE 1. Tolerance scheme and ideal frequency response for a lowpass filter

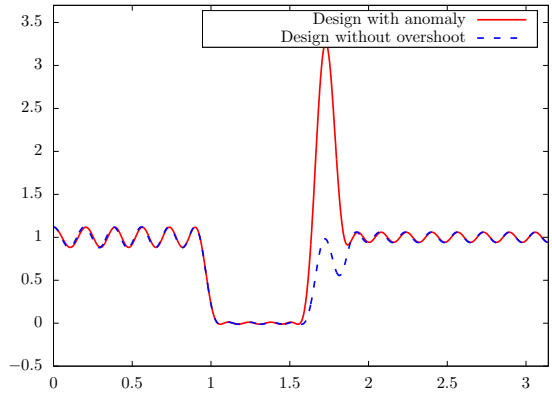

FIGURE 2. Design example showcasing a transition band anomaly and how it can be removed.

where δ_1 and δ_2 denote the maximum errors allowed over $[0, \omega_p]$ (the *passband*) and $[\omega_s, \pi]$ (the *stopband*). Between pass and stopbands, we find transition bands. They are crucial here because their absence incurs jump discontinuities in the functions to approximate (at $\omega = \omega_p = \omega_s$), giving rise to the unwanted Gibbs-Wilbraham phenomenon (see [57, Ch. 9] for a detailed description).

In practice, the frequency response of a minimax filter inside transition regions is required to be *monotonic* and/or not present *overshoots* [45]. Such restrictions are implicitly assured in most practical cases, even though they do not hold in general [45]. Figure 2 shows a three band, degree $n = 38$, type I minimax FIR filter with passband $[0, 0.3\pi]$, stopband $[0.33\pi, 0.5\pi]$, passband $[0.6\pi, \pi]$ and corresponding weights 1, 10 and 2. A huge spike is clearly visible in the transition region between the stopband and the second passband. If we only need to contain it, we can adapt the initial specification using mild constraints on the problematic transition region [3, Sec. 3] and get the second minimax filter from Figure 2, where the transition response is confined between 0 and 1. There is only a minor loss in quality with respect to the initial design (the minimax error δ_Ω grows from 0.1172 to 0.1205). If monotonicity is required, extensions of the exchange method that ensure it [53] exist. The cost is again, a larger δ_Ω.

3. Practical problems

Over the years, certain difficulties with implementing multi-interval exchange methods have been discussed in the literature. We give short accounts on the most important ones, previous solutions and where we *address* them in the sequel.

3.1. Robustness issues: initialization is a key step.

The need for high-order filters, where $n = 100 \dots 500$ or even larger, is explored in [23, 41]. Applications include the design of transmultiplexers and filter banks for high resolution spectral analysis.

A good choice for the starting reference vector $\boldsymbol{\omega} = (\omega_k)_{0 \leqslant k \leqslant n+1}$ is critical for practical convergence. The default method employed in practice [6, Ch. 15.3.1] is to take the elements of $\boldsymbol{\omega}$ *uniformly* from $\Omega \in [0, \pi]$. An empirical justification for this comes from a parallel with how one initializes the Remez algorithm on $[-1, 1]$. There, good choices are usually Chebyshev nodes of the second kind (see Section 4.1), which, according to Remark 2.2, map to a uniform grid on $[0, \pi]$.

For a large number of high-degree problems this classic choice does not work well. Several alternatives are available. One approach is to pick reference sets according to some empirically computed distributions [2, 41, 53]. Another tactic [44] is to compute a least-squares optimal filter with the same band and weight constraints as the minimax problem and take the local extrema of its approximation error as the initial reference set $\boldsymbol{\omega}$. Using Carathéodory-Fejer (CF) near-best approximations [39, Sec. 3.6] is also an option[2]. Nevertheless, the major shortcoming of these approaches is that they have been shown to work only in some cases (for example, uniform weighted filters for CF approximation).

Our solution: In the next section we introduce two new heuristics for initializing the exchange algorithm. They are quite general and effective in practice, often leading to significant speedups when compared to existing

[2] http://www.chebfun.org/examples/approx/FiltersCF.html

implementations. Moreover Section 5.2 introduces useful *analysis tools* for *debugging* numerical problems, which are, in particular, frequent for high-degree designs.

3.2. **Scalability issues: speeding up the extrema search.** For most inputs, especially high degree ones, the computational bottleneck of the exchange algorithm is the search for potential extrema. Articles that concentrate on this aspect are [4,5,8]. The common denominator of all three approaches is that they use information pertaining to the first and/or second derivative of the error function $E(\omega)$ for accelerating the extrema search.

Our solution: Section 6 describes a search strategy based on Chebyshev root-finding. Compared to [39], our domain subdivision routine is not recursive, making it very easy to introduce parallelism.

4. Two new heuristics for choosing the initial reference

The first step of the algorithm we stated in Section 2.2 is to choose the initial reference vector $\boldsymbol{\omega}$ (or equivalently $\mathbf{x}$).

4.1. **A reminder on Lebesgue constants, Chebyshev nodes and Chebyshev polynomials.** Consider a compact set $K \subset \mathbb{R}$ and the finite-dimensional space of weighted polynomials $w\mathbb{P}_n(\mathbb{R}) = \operatorname{span}_{\mathbb{R}}\{w(x), w(x)x, \dots, w(x)x^n\}$ of degree at most n, where $w \in \mathcal{C}(K)$ is a positive weight function. If $T = \{t_0, \dots, t_n\} \subset K$ and $\{\phi_0, \dots, \phi_n\}$ is a basis of $w\mathbb{P}_n(\mathbb{R})$, we introduce the Vandermonde-like matrix

$$V(t_0, \dots, t_n) = [v_{ij}] := [\phi_j(t_i)].$$

The Lagrange interpolation operator at the nodes T has basis functions

$$\ell_i(x) = \frac{\det V(t_0, \dots, t_{i-1}, x, t_{i+1}, \dots, t_n)}{\det V(t_0, \dots, t_{i-1}, t_i, t_{i+1}, \dots, t_n)}, \qquad i = 0, \dots, n \tag{3}$$

and is defined as

$$\mathcal{L}_T f(x) = \sum_{i=0}^{n} f(t_i)\ell_i(x).$$

Its operator norm (also called *Lebesgue constant*), satisfies [42, Thm. 4.3]

$$\Lambda_{K,T,w} = \|\mathcal{L}_T\| = \max_{x \in K} \sum_{i=0}^{n} |\ell_i(x)|$$

and offers a *quantitative* way to measure the *quality* of a family of points for doing interpolation over K. Indeed, for every $f \in \mathcal{C}(K)$ we have [42, Thm. 3.1]

$$\|f - \mathcal{L}_T f\|_K \leqslant (1 + \Lambda_{K,T,w})\operatorname{dist}_K(f, w\mathbb{P}(\mathbb{R})). \tag{4}$$

For us, Lebesgue constants over the compact X play a major role in the next subsection, while in Section 5, Lebesgue constants over $[-1, 1]$ with weight function $w(x) = 1$ are used to justify the numerical stability of barycentric Lagrange interpolation inside the exchange algorithm.

Classic examples of good interpolation points with small Lebesgue constants over $[-1, 1]$ are the Chebyshev nodes. To define them, we need Chebyshev polynomials.

The n-th Chebyshev polynomial of the first kind is

$$T_n(\cos(\omega)) = \cos(n\omega), \forall \omega \in [0, \pi],$$

or recursively $T_0(x) = 1, T_1(x) = x, T_{n+2}(x) = 2xT_{n+1}(x) - T_n(x), \forall n \in \mathbb{N}$. In similar fashion, Chebyshev polynomials of the second kind are given by

$$U_0(x) = 1, U_1(x) = 2x, U_{n+2}(x) = 2xU_{n+1}(x) - U_n(x), \forall n \in \mathbb{N}.$$

Going back to formula (2), we see that the transfer function $H(\omega)$ is in fact a linear combination of Chebyshev polynomials, *i.e.*,

$$H(\omega) = \sum_{k=0}^{n} h_k \cos(k\omega) = \sum_{k=0}^{n} h_k T_k(\cos(\omega)).$$

The Chebyshev nodes of the first kind are

$$\mu_k = \cos\left(\frac{(2k+1)\pi}{2(n+1)}\right), \qquad k = 0, \dots, n,$$

TABLE 1. Number of reference values in each band for the bandstop specification of Example 1.2 before the first iteration and at the end of execution.

	$n = 50$	$n = 100$	$n = 200$
$[0, 0.2\pi]$	14/13	26/26	51/51
$[0.3\pi, 0.5\pi]$	14/15	26/31	51/59
$[0.6\pi, \pi]$	24/24	50/45	100/92

and correspond to the roots of T_{n+1}, while the Chebyshev nodes of the second kind are the local extrema of T_n over $[-1, 1]$. They can be given analytically as

$$\nu_k = \cos\left(\frac{k\pi}{n}\right), \qquad k = 0, \ldots, n.$$

4.2. **An approximate Fekete points approach.** Equation (4) suggests that point sets with small Lebesgue constants over X are good choices for starting the exchange algorithm. Fekete points are a natural choice. In the language of Section 4.1, they maximize the Vandermonde determinant in the denominators of the Lagrange basis functions (3), resulting in a value bounded by the dimension of the interpolation space.

As noted in [55,57], their exact computation is very challenging in general. Despite this, recent articles [9,10,54,55] explore the use of a greedy algorithm for computing *approximate* Fekete points. It is a QR-based factorization routine which extracts large volume submatrices from rectangular Vandermonde-type matrices defined on an *appropriate* discretization of X. Numerical examples of weighted polynomial interpolation problems using this approach are discussed in [55]. Here, we only give a very brief statement of this algorithm when applied to our context.

For a degree n minimax approximation in the form of Problem 2.1, if

$$Y = \{y_i\} \subset X, \quad 0 \leqslant i < N, \quad N > n + 2,$$

is a suitable discretization of $X \subset [-1, 1]$, we construct the $N \times (n + 2)$ matrix

$$V(y_0, \cdots, y_{N-1}) = [v_{ij}] := [W(\arccos(y_i)) \cos(j \cdot \arccos(y_i))], \quad 0 \leqslant i < N, \quad 0 \leqslant j \leqslant n + 1$$

and extract from it a maximum volume $(n + 2) \times (n + 2)$ submatrix. Although this problem is known to be NP-hard [16], the following linear algebra-based greedy algorithm gives good results in practice:

AFP (Approximate Fekete Points) algorithm
Input: appropriate discretized subset $Y = \{y_0, \ldots, y_{N-1}\} \subset X, N > n + 2$
Output: $Y_\star = \{y_0^\star, \ldots, y_{n+1}^\star\} \subset Y$ s.t. the $(n + 2) \times (n + 2)$ Vandermonde-like submatrix constructed from the elements of $Y_\star$ has a large volume

(1) **Initialization**: $V = V(y_0, \ldots, y_{N-1}), b \in \mathbb{R}^{n+2}, b = (1, \ldots, 1)^t$
(2) **QR-based Linear System Solver**: Find $w \in \mathbb{R}^N$, which solves the system $b = V^t w$. This *must* be done using a column pivoting-based QR factorization [14].
(3) **Subset Selection**: Take $Y_\star$ as the set of elements from Y whose corresponding terms inside w are different from zero, that is, if $y_i \in Y$ and $w_i \neq 0$, then $y_i \in Y_\star$.

Appealing choices for Y are, so-called, (weakly) admissible meshes [15]. This is due to the fact that approximate Fekete points chosen from admissible meshes are known to have the same asymptotic behavior as true Fekete points. Using this theory, we can construct very simple weakly admissible meshes for X. For instance, if $X = \bigcup_{k=1}^m X_k, m \in \mathbb{N}^*$ is a finite disjoint union of closed intervals, then $Y = \bigcup_{k=1}^m Y_k$, where Y_k corresponds to the $(n + 1)$-th order Chebyshev nodes of the first kind mapped to X_k, is a suitable discretization of X. Also, it has a relatively small size: $O(n)$ elements if we consider m to be a small constant.

Despite being very robust, the only major downside of this approach is that, because of the $O(n^3)$ linear solver, it does not scale particularly well to huge degrees. Hence, in practice, we suggest limiting its use up to degree $n = 500$ problems and combine it, if needed, with the following *complementary* approach.

4.3. **A reference scaling idea.** The asymptotics of (weighted) minimax polynomial approximation problems is an important subject of Potential Theory [32]. Of particular interest to us is the fact that, *asymptotically*, the final reference sets of minimax approximations follow a certain distribution over X, called the *equilibrium distribution* of X (see, for instance, [52, Sec. 5] for the necessary theoretical results). Computing it, in all but the simplest setting (the interval), is an involved procedure [24, 52], cumbersome to carry out in practice.

TABLE 2. Iteration count comparison for uniform/reference scaling/ AFP-based initialization

Example 1.1		Example 1.2		Example 1.3		Example 1.4	
Degree	Iterations	Degree	Iterations	Degree	Iterations	Degree	Iterations
50	11/4/6	50	14/14/4				
80	8/3/4	80	13/3/12	520	12*/3/1	53248	NC/3/NC
100	9/8/3	100	23*/18/16				

An important element developed in the aforementioned articles, confirmed experimentally in our tests, is that the proportion of reference values for minimax polynomial approximants inside each interval of X is asymptotically constant with the degree. As an example, we consider the design specification from Example 1.2 and look at what happens when uniform initialization is used within the exchange algorithm. The a/b cells in Table 1 show the number of reference points inside each band before the first iteration (the a value) and upon convergence (the b value), for three different degrees. As expected, the b values tend to be proportional with n, while for the second and third band, the corresponding a and b become further apart as we increase the degree.

This behavior allowed us to develop a *relatively fast* and *very simple* to implement heuristic for choosing an initial reference. The core of the approach is that, if, for a degree n approximation, we first compute the minimax result of degree $\lfloor n/2 \rfloor$, then we have a very good idea on the number of reference values to put inside each band for the degree n problem. The values of the new reference set $\mathbf{x}$ are established by taking the $\lfloor n/2 \rfloor + 2$ final references of the smaller result and adding the remaining $n - \lfloor n/2 \rfloor$ points uniformly between them, thus ensuring a *similar* distribution of the references inside X for the two degrees. If needed, this strategy can be applied recursively.

Since it is based on asymptotic results, this idea works best for high degree designs, where the AFP-based strategy becomes expensive.

4.4. **Numerical examples.** Performance-wise, Table 2 shows some results when considering Examples 1.1 to 1.4. In the Iterations cells, the first value corresponds to the uniform initialization technique, while the second and third ones represent our scaling and AFP approaches. We can observe that, in many cases, there is a considerable reduction in the number of iterations required for convergence. This frequently translates to smaller execution times as well, despite a more involved setup of the starting references. Still, the biggest advantage of these heuristics is that often, they allow us to address, in a simple manner, filter design problems where uniform initialization gives rise to numerical instability issues and the implementation does not converge (NC). We also tried the strategies from [44, 53], but encountered numerical problems on the larger designs of Examples 1.3 and 1.4.

In the case of Example 1.3 we applied reference scaling two times, starting from a degree $n = 130$ filter and then moving up to $n = 260$ and finally $n = 520$. We only needed $4, 3$ and 3 iterations for convergence. AFP initialization performed even better, only requiring one iteration. For Example 1.4, we again applied reference scaling two times, for a number of $6, 3$ and 3 iterations, whereas AFP proved to be too costly.

Note that, actually, convergence with uniform initialization for Examples 1.2 and 1.3 (the two $*$-ed cases) occurs in spite of numerical problems: this behavior is fortunate and does not generally happen in such a bad numerical context. See also Remark 5.1 for more insight.

Inside the test files, we have provided code for the design of over 70 different filters that are considered high degree (with n in the order of hundreds and thousands). While the reduction in number of iterations, when computable, was very variable (savings from 0% to over 70% compared to uniform initialization were observable on most of the examples), reference scaling *always* ensured convergence, whereas for more than 30% of the considered filters, uniform initialization failed to converge. The AFP-based idea also performed very well, with only a small percentage (around 5%) of the tests failing. The examples where this happened had degree $n \geqslant 1000$ and were much better suited for the scaling approach.

5. Computing the current interpolation function

Next, we need to determine the value of the current alternating error δ and derive a formula for computing $H(\omega)$ (Step 2). If we apply the Alternation Theorem on the current reference set, our unknowns satisfy the following equations:

$$E(\omega_i) = W(\omega_i)\,[D(\omega_i) - H(\omega_i)] = (-1)^i \delta'_{\mathbf{x}}, \qquad i = 0, \ldots, n+1,$$

where D denotes the ideal filter we are approximating, W is the corresponding weight function for the error and $\delta_{\mathbf{x}} = |\delta'_{\mathbf{x}}|$. By taking $H(\omega)$ in its cosine expansion from equation (2), we get the following linear system of equations

in $(h_k)_{0\leqslant k\leqslant n}$ and $\delta'_{\mathbf{x}}$:

$$\begin{bmatrix} 1 & \cos(\omega_0) & \cos(2\omega_0) & \cdots & \cos(n\omega_0) & \dfrac{1}{W(\omega_0)} \\ 1 & \cos(\omega_1) & \cos(2\omega_1) & \cdots & \cos(n\omega_1) & \dfrac{-1}{W(\omega_1)} \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 1 & \cos(\omega_{n+1}) & \cos(2\omega_{n+1}) & \cdots & \cos(n\omega_{n+1}) & \dfrac{(-1)^{n+1}}{W(\omega_{n+1})} \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \\ \vdots \\ h_n \\ \delta'_{\mathbf{x}} \end{bmatrix} = \begin{bmatrix} D(\omega_0) \\ D(\omega_1) \\ \vdots \\ D(\omega_n) \\ D(\omega_{n+1}) \end{bmatrix}$$

This system has a unique solution, since the $(n+2)\times(n+2)$ matrix is non-singular (see [58] for a proof idea). According to [6, Ch. 15.3.3] and [38, Ch. 7.7.3], solving it is not encouraged, since it is computationally inefficient and susceptible to numerical ill-conditioning.

In practice, one deduces $\delta'_{\mathbf{x}}$ analytically and uses barycentric Lagrange interpolation to determine $H(\omega)$ (*i.e.* the original choice of [40]). The entirety of this section focuses on the numerical stability of this approach and of computing $\delta'_{\mathbf{x}}$.

5.1. **Barycentric Lagrange interpolation.** In recent years, barycentric interpolation has become an active research topic in Numerical Analysis. This is mostly due to the nice numerical properties of this interpolation scheme [7, 25].

The basic setting is the following: if $f:[x^-,x^+]\to\mathbb{R}$ and $n\in\mathbb{N}$, let $\mathbf{x}\in\mathbb{R}^{n+2}$ be a vector of distinct interpolation points $x_k\in[x^-,x^+], k=0,\ldots,n+1$, given in increasing order, together with $\mathbf{y}\in\mathbb{R}^{n+2}$, where $y_k=f(x_k), k=0,\ldots,n+1$. We want to find a polynomial p with real coefficients of degree at most $n+1$ which interpolates f at $\mathbf{x}$ (*i.e.*, $p(x_k)=y_k, k=0,\ldots,n+1$).

According to [47], the barycentric forms of p are given by

$$p(x)=\ell(x)\sum_{k=0}^{n+1}\frac{w_k}{x-x_k}y_k, \tag{5}$$

known as the *first* barycentric interpolation formula, and

$$p(x)=\frac{\displaystyle\sum_{k=0}^{n+1}\frac{w_k}{x-x_k}y_k}{\displaystyle\sum_{k=0}^{n+1}\frac{w_k}{x-x_k}}, \tag{6}$$

the *second* (or *proper*) barycentric formula, where $\ell(x)=\prod_{i=0}^{n+1}(x-x_i)$ and

$$w_k=\frac{1}{\ell'(x_k)}=\frac{1}{\prod_{i\neq k}(x_k-x_i)},\qquad k=0,\ldots,n+1. \tag{7}$$

If the barycentric weights w_k have been precomputed, both (5) and (6) can be evaluated with only $O(n)$ arithmetic operations at an arbitrary point x.

To use barycentric interpolation for the evaluation of $H(\omega)$, we first compute $\delta'_{\mathbf{x}}$ using the formula

$$\delta'_{\mathbf{x}}=\frac{\displaystyle\sum_{k=0}^{n+1}w_kD(\omega_k)}{\displaystyle\sum_{k=0}^{n+1}\frac{(-1)^kw_k}{W(\omega_k)}} \tag{8}$$

(see [39, Sec. 3.3] for a proof idea).

We have

$$H(\omega)=\frac{\displaystyle\sum_{k=0}^{n+1}\frac{w_k}{x-x_k}c_k}{\displaystyle\sum_{k=0}^{n+1}\frac{w_k}{x-x_k}} \tag{9}$$

where $x = \cos(\omega), c_k = D(\omega_k) - (-1)^k \dfrac{\delta'_{\mathbf{x}}}{W(\omega_k)}$ and $\mathbf{w} \in \mathbb{R}^{n+2}$ is the weight vector whose elements are computed according to (7). Such a second barycentric formula is usually preferred for computing the current frequency response. We will elaborate on why this is a good idea in Section 5.2.2.

5.2. **Numerical stability issues.** The evaluation of (8) and (9) is split into the computation of the barycentric weights (done once for each new reference vector $\mathbf{x}$) and of the sums in the numerators and the denominators. For a numerically robust evaluation of the w_k's ($O(n^2)$ operations), we use the formulas and ideas of [7, Sec. 7] and [39, Sec. 3.4]. Notice that inside (9) we are using $n+2$ points to interpolate a polynomial of degree at most n. This is intentional. We could have easily picked only a subset of $n+1$ elements of $\mathbf{x}$, but leaving out one element can pose numerical problems, especially if that point is the smallest or largest one from $\mathbf{x}$ [59].

5.2.1. *The current leveled error.* Let $p_{\mathbf{x}}$ and $q_{\mathbf{x}}$ be the numerator and denominator values of $\delta'_{\mathbf{x}}$ in formula (8). Unless otherwise stated, all elementary operations are performed in round-to-nearest double-precision arithmetic with unit roundoff $u = 2^{-53}$.

Central to our analysis is the summation condition number [37, Ch. 6.1]: given n real values $a_0, \ldots, a_{n-1}$ whose sum $s_n = \sum_{k=0}^{n-1} a_k$ we want to compute, its corresponding condition number is $C_n = \left(\sum_{k=0}^{n-1} |a_k|\right) / \left|\sum_{k=0}^{n-1} a_k\right|$. If we only have access to perturbed versions $\hat{a}_k = a_k(1+\varepsilon_k), k = 0, \ldots, n-1$ of the input terms, then the relative error E_n of computing $\hat{s}_n = \sum_{k=0}^{n-1} \hat{a}_k$ is upper bounded by $\max_k |\varepsilon_k| \, C_n$. Indeed,

$$E_n = \frac{|\hat{s}_n - s_n|}{|s_n|} = \frac{\left|\sum_{k=0}^{n-1} \varepsilon_k a_k\right|}{\left|\sum_{k=0}^{n-1} a_k\right|} \leqslant \frac{\sum_{k=0}^{n-1} |\varepsilon_k| \, |a_k|}{\left|\sum_{k=0}^{n-1} a_k\right|} \leqslant \max_k |\varepsilon_k| \, \frac{\sum_{k=0}^{n-1} |a_k|}{\left|\sum_{k=0}^{n-1} a_k\right|}.$$

Even if the perturbations are as small as possible with the current arithmetic, meaning $\varepsilon_k = O(u), k = 0, \ldots, n-1$, if there is severe cancellation in computing s_n and $C_n \gg u^{-1}$, then E_n will most likely be greater than 1 and give totally inaccurate results. Since $p_{\mathbf{x}}$ and $q_{\mathbf{x}}$ are obtained using finite precision arithmetic, in certain situations, there is pronounced numerical cancellation when computing $p_{\mathbf{x}}$. This can happen when:

(1) the minimax error at the end of executing the exchange algorithm is very small (in the sense of being relatively close to or smaller than u);
(2) the final minimax error is much larger than u, but the starting leveled error $\delta_{\mathbf{x}}$ is in the order of u or much smaller.

In practice, the second bullet point is more likely to happen than the first one, since filters with minimax error $\delta_X < 10^{-10} (\gg u \sim 10^{-16})$ seldom seem to be required.

When using uniform initialization, such issues are not hard to find, as they sometimes occur in the starting iteration(s) of the exchange method. Consider, for instance, the specification from Example 1.3. Although the final alternating error is around $1.6067 \cdot 10^{-7}$, the first iteration should give $\delta_{\mathbf{x}} \simeq 1.521 \cdot 10^{-21} \ll u$ and a condition number for $p_{\mathbf{x}}$ of about $6.572 \cdot 10^{20} \simeq 72967 \cdot u^{-1}$. Because of this extreme ill-conditioning, a double-precision computation of $\delta_{\mathbf{x}}$ gives a wrong value of $5.753 \cdot 10^{-19}$.

Based on our discussion up to this point, we propose two possible solutions:

(1) use a higher precision arithmetic for doing *all* computations in the current iteration (barycentric weights, $\delta_{\mathbf{x}}$ and local extrema of $E(\omega)$);
(2) pick a *better* starting reference (see initialization strategies from Section 4).

The first alternative is generally the safest, but costlier choice. From our experience, around 150-200 bits of precision are more than enough for the problems discussed here. Hence, the `MPFR`-based version of our code uses 165 bits of precision by default. Once enough iterations have passed and the current condition number for $p_{\mathbf{x}}$ is sufficiently small (*i.e.* smaller than say 10^8), there is a good chance that using double-precision arithmetic for the remaining iterations will still lead to an acceptable result.

For the second strategy, the basic intuition is that the starting $\delta_{\mathbf{x}}$ should be much larger and closer to the minimax one. This consequently implies that the condition number for $p_{\mathbf{x}}$ is much smaller compared to u^{-1} and the summations are more stable. To illustrate this, we again consider Example 1.3, but this time, use the reference scaling strategy from Section 4.3. The initial leveled error is $\delta_{\mathbf{x}} \simeq 1.456 \cdot 10^{-7} \gg u^{-1}$ and the condition number for $p_{\mathbf{x}}$ is around $6.864 \cdot 10^6$. Double-precision computations are again quite viable. Since it usually results in much smaller execution times, we recommend starting with this second approach.

Numerical problems are much less likely to occur when computing $q_{\mathbf{x}}$. Since we assume the interpolation nodes $\mathbf{x}$ are ordered by value, the barycentric weights w_i alternate in sign. This means that all the terms of $q_{\mathbf{x}}$ have the same sign and hence, its condition number has minimum value one.

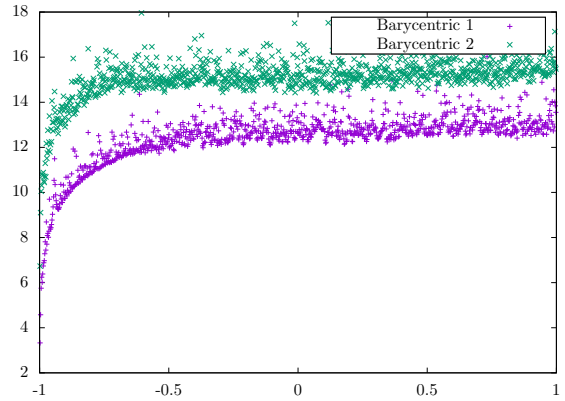

FIGURE 3. Relative errors (correct significant digits) when computing the starting transfer function for Example 1.3 using both types of barycentric formulas. Uniform initialization ($\Lambda_{[-1,1],\mathbf{x}} \simeq 4.24973 \cdot 10^{14}$) is used.

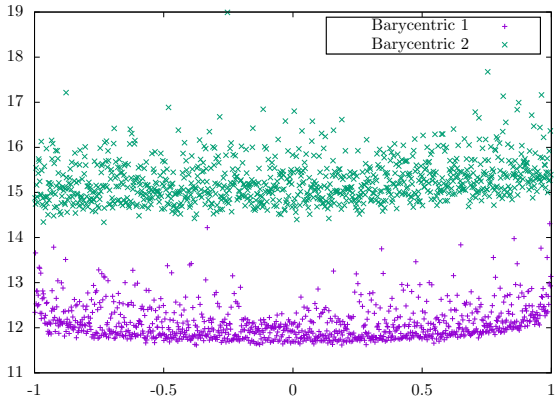

FIGURE 4. Relative errors (correct significant digits) when computing the starting transfer functions for Example 1.3 using both types of barycentric formulas. Reference scaling ($\Lambda_{[-1,1],\mathbf{x}} \simeq 1.29773 \cdot 10^{5}$) is used.

5.2.2. *Lebesgue constants as tools for detecting numerical problems.* The numerical stability of (5) and (6) over $[-1, 1]$ was first looked at in a rigorous manner in [28]. It shows how the modified Lagrange formula (5) is backward stable in general, whereas (6) is shown to be forward stable for vectors of points $\mathbf{x} \in \mathbb{R}^{n+2}$ having a small Lebesgue constant[3]. Despite a less favorable numerical behavior of (6) in general, for sets of points having a small Lebesgue constant, like Chebyshev nodes, using the second barycentric formula is *preferable* [7, 33]. More recently, [34] argues that (6) is backward stable if the relevant Lebesgue constant associated with the interpolation vector $\mathbf{x}$ is small.

An example of a reference set with small Lebesgue constant is showcased in Figures 3 and 4 for the specification given in Example 1.3. Both plots show the relative errors in computed transfer function $H(\omega)$ during the first iteration of the exchange algorithm. The 'exact' computations were performed with `MPFR` using 600 bits of precision (roughly 180 digits). In Figure 3, $\Lambda_{[-1,1],\mathbf{x}}$ is quite large and we see that the loss in accuracy is clearly visible around $x = -1$ (*i.e.* $\omega = \pi$). The results are much better in Figure 4 for a smaller Lebesgue constant, reinforcing our earlier statements.

We make the following *experimental* remark: in general, the Lebesgue constants associated with the references that appear during the execution of the Parks-McClellan algorithm tend to decrease in value, with the final $\Lambda_{[-1,1],\mathbf{x}}$ being not too large.

Typical scenarios of how $\Lambda_{[-1,1],\mathbf{x}}$ changes during the execution of the exchange algorithm are given in Tables 3 to 5 for Examples 1.1 to 1.3. The numerical estimates were computed using the Chebfun routine `lebesgue` mentioned in [57, Ch. 15]. Because of the observed behavior, computations will tend to be more stable during latter iterations. The effect of using the strategies from Section 4 is also clearly visible.

Since a large Lebesgue constant $\Lambda_{[-1,1],\mathbf{x}}$ is symptomatic for a very small $\delta_{\mathbf{x}}$, the solutions for countering ill-conditioning of the previous subsection also apply here.

Remark 5.1. When numerical problems of the types discussed in this section occur (typically because of uniform initialization), our exchange algorithm implementation using double-precision arithmetic can sometimes still converge to the correct result. This may happen even if the computations at certain moments are totally inaccurate. In particular, running the same problem instance with a higher precision, where we assume *all* computations to be accurate, gives a totally different execution trace, despite the output consistency between the two. Nevertheless, as our test cases show, this is a completely random behavior which is not to be expected and, most of all, trusted. One should *definitely* consider using a higher working precision and/or one of the initialization techniques from Section 4.

6. Extrema search

The final two steps of an exchange algorithm iteration consist of computing the new reference set $\boldsymbol{\omega}' \in \mathbb{R}^{n+2}$ (and equivalently $\mathbf{x}' = (\cos(\omega'_k))_{0 \leqslant k \leqslant n+1}$). They require finding the local extrema of the current error function $E(\omega)$. Chebyshev root-finding is reviewed in the next subsection, while Section 6.2 introduces our non-recursive domain

[3]All such constants in this subsection are defined over $[-1, 1], \mathbf{x}$ and unit weight function. Thus, we use the simplified notation $\Lambda_{[-1,1],\mathbf{x}}$ when referring to them.

TABLE 3. Lebesgue constant evolution during the execution of the exchange algorithm (*i.e.*, first, middle and last iteration). The starting reference set is computed using uniform intialization.

Example 1.1 $n = 100$		Example 1.2 $n = 100$		Example 1.3 $n = 520$	
Iteration	$\Lambda_{[-1,1],\mathbf{x}}$	Iteration	$\Lambda_{[-1,1],\mathbf{x}}$	Iteration	$\Lambda_{[-1,1],\mathbf{x}}$
1	$3.72896 \cdot 10^{10}$	1	$3.00467 \cdot 10^{15}$	1	$4.24973 \cdot 10^{14}$
5	$8.31678 \cdot 10^{6}$	12	$7.45988 \cdot 10^{7}$	6	$6.11242 \cdot 10^{5}$
9	$4.00568 \cdot 10^{6}$	23	$1.35889 \cdot 10^{7}$	12	$1.38665 \cdot 10^{5}$

TABLE 4. Lebesgue constant evolution when the reference scaling approach described in Section 4.3 is used.

Example 1.1 $n = 100$		Example 1.2 $n = 100$		Example 1.3 $n = 520$	
Iteration	$\Lambda_{[-1,1],\mathbf{x}}$	Iteration	$\Lambda_{[-1,1],\mathbf{x}}$	Iteration	$\Lambda_{[-1,1],\mathbf{x}}$
1	$5.98098 \cdot 10^{6}$	1	$4.12072 \cdot 10^{8}$	1	$1.29773 \cdot 10^{5}$
4	$4.68392 \cdot 10^{6}$	9	$1.04020 \cdot 10^{7}$	2	$1.38250 \cdot 10^{5}$
8	$4.00763 \cdot 10^{6}$	18	$1.35728 \cdot 10^{7}$	3	$1.38015 \cdot 10^{5}$

TABLE 5. Lebesgue constant evolution when the AFP-based approach described in Section 4.2 is used.

Example 1.1 $n = 100$		Example 1.2 $n = 100$		Example 1.3 $n = 520$	
Iteration	$\Lambda_{[-1,1],\mathbf{x}}$	Iteration	$\Lambda_{[-1,1],\mathbf{x}}$	Iteration	$\Lambda_{[-1,1],\mathbf{x}}$
1	$4.06753 \cdot 10^{6}$	1	$3.59135 \cdot 10^{8}$		
2	$4.0023 \cdot 10^{6}$	8	$9.90082 \cdot 10^{7}$	1	$1.38015 \cdot 10^{5}$
3	$4.00476 \cdot 10^{6}$	16	$1.35837 \cdot 10^{7}$		

subdivision strategy. For completeness, Sections 6.3 and 6.4 talk about updating the reference set at each iteration and how to retrieve the filter coefficients upon convergence.

6.1. **Chebyshev-proxy root-finding.** We first consider the slightly different problem of determining the zeros of a function $f \in \mathcal{C}([x^-, x^+])$, located inside $[x^-, x^+]$. For simplicity, we again take $[x^-, x^+] = [-1, 1]$, and note that by suitable changes of variable, the following results hold for any closed interval. The idea of the Chebyshev-proxy root-finder (CPR) method [12,57] is to replace $f(x)$ by a degree m polynomial *proxy* $p_m(x)$. If the chosen polynomial is an accurate enough approximation of f, then its zeros will very closely match those of f. It interpolates f at the Chebyshev nodes of the second kind $\nu_k = \cos(k\pi/m), 0 \leqslant k \leqslant m$, and is expressed using the basis of Chebyshev polynomials of the first kind. We get

$$p_m(x) = \sum_{k=0}^{m} a_k T_k(x), \qquad x \in [-1, 1], a_k \in \mathbb{R}, k = 0, \ldots, m,$$

with $p_m(\nu_k) = f(\nu_k), k = 0, \ldots, m$ and

$$a_k = \frac{2}{m} \sum_{i=0}^{m}{}'' f(\nu_i) T_k(\nu_i), \qquad k = 0, \ldots, m, \tag{10}$$

where $\sum''$ denotes that the first and last terms of the sum are to be halved.

The a_i coefficients can be evaluated in a numerically stable way with $O(m^2)$ operations, by using Clenshaw's recurrence relations [43, Ch. 5.4], or faster, in only $O(m \log m)$ operations, by means of the Discrete Cosine Transform [56]. The roots of p_m are then computed as the eigenvalues of a generalized companion matrix [11, 19], also known as a *colleague matrix* [57, Ch. 18]. This approach behaves well in practice [13].

To obtain good approximations of the local extrema of $E(\omega)$, suppose we already have an accurate degree m proxy

$$E_m(\omega) = \sum_{k=0}^{m} a_k T_k(x), \qquad x = \cos(\omega).$$

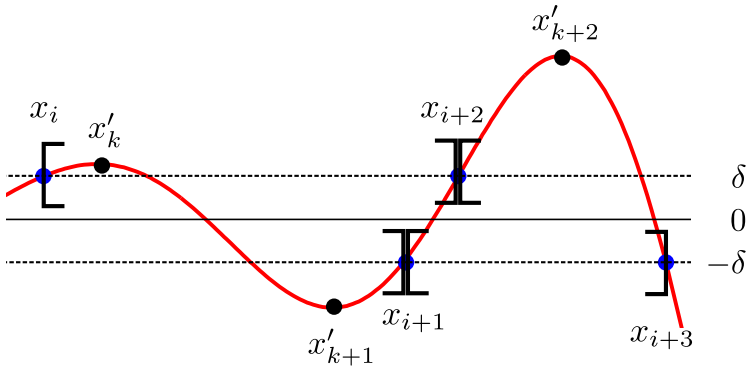

FIGURE 5. Our interval subdivision strategy for computing the extrema of $E(\omega)$.

The value of m should be thought of independently from the degree n of the target minimax response $H(\omega)$. We look for the roots of $e_m(\omega) = \dfrac{\mathrm{d}E_m(\omega)}{\mathrm{d}x}$, whose Chebyshev expansion is

$$e_m(\omega) = \sum_{k=0}^{m-1} b_k T_k(x),$$

where: $b_{k-1} = 2ka_k + b_{k+1}, k = 1, \ldots, m$, and $b_m = b_{m+1} = 0$. Similarly, if we take into account that $\dfrac{\mathrm{d}T_m}{\mathrm{d}x} = mU_{m-1}$, for $m \geqslant 1$, we have

$$e_m(\omega) = \sum_{k=0}^{m-1} c_k U_k(x),$$

where $c_{k-1} = ka_k, k = 1, \ldots, m$. Although the CPR method is introduced in the context of T_m, we found the second formula for e_m more natural to use in our setting.

To find the eigenvalues of a $m \times m$ colleague matrix, for both types of Chebyshev polynomials, we can use a QR/QZ algorithm ($O(m^3)$ operation count). For numerical reasons, we use the colleague matrix form suggested in [57, Ex. 18.3].

This cost can be reduced to $O(m^2)$ computations by dividing the initial domain $[x^-, x^+]$ into several subintervals and taking Chebyshev interpolation polynomials of smaller degree on each of them. The roots can then be determined as the collection of zeros inside all subintervals. More precisely: if $N_{\max} < m$ is the maximum degree on which one is willing to use an eigenvalue algorithm, *recursively* split $[x^-, x^+]$ until there are Chebyshev polynomial interpolants of degree at most $N_{\max}$ on each subinterval, which approximate f accurately.

There are several general criteria that quantify when we have a good enough approximation of $f(x)$ [11, 12]. The [39] Remez algorithm implementation uses such a recursive approach [57, Ch. 18] by means of the Chebfun `roots` command. Note that the intermediary interpolants computed before the final stage subdivisions are not used for the actual root finding.

6.2. **A new subdivision strategy.** We use a different, *non recursive* approach. In general, during each iteration of the Parks-McClellan algorithm, we have estimates about the location and number of extrema of E; if we take a subinterval defined by two consecutive points of the current reference vector $\mathbf{x}$ located inside the same band, say $[x_i, x_{i+1}]$, then we *usually* expect to have between zero and two potential extrema of E inside it. Figure 5 summarizes this scenario. We then use a degree $N_{\max}$ Chebyshev proxy on each subinterval. Values of $N_{\max}$ we frequently used in our experiments are 4 and 8. Dynamic range issues [12, Sec. 7.4] are generally absent because the initial approximations obtained with the approaches from Section 4 are of good quality. Since there is only one subdivision level (no intermediary interpolants), there will be savings in computation time.

At each iteration, we use $O(n)$ subintervals of the form $[x_i, x_{i+i}]$. To cover all of X, we also consider all intervals of the form $[a, x_i] \subseteq X$ and $[x_j, b] \subseteq X$, where a and b are band extremities of X and x_i and x_j are elements of $\mathbf{x}$ which are closest to a and b, respectively. Inside each subinterval, we evaluate the error $N_{\max} + 1$ times in order to construct a suitable proxy for E and its derivative. Because of barycentric interpolation, each evaluation of E will usually require only $O(n)$ operations, while solving each small eigenvalue problem amounts to a constant number of computations. The total operation count will be $O(n^2)$.

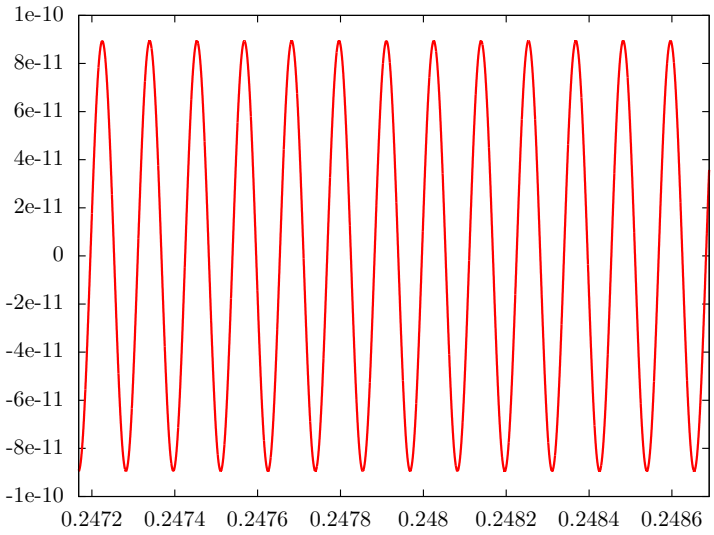

FIGURE 6. The value of the approximation error E on a small interval $[0.247168, 0.24869]$ of X, for the minimax filter satisfying the specification for Example 1.4.

In addition to requiring the evaluation of E a reasonable number of times (in the range of $4n$ to $8n$), this approach can also be very easily parallelized (the computations on each subinterval are *data parallel*). For large degree approximations on multicore systems, the speedup is considerable. We give some examples in Section 7.2.

Remark 6.1. This quadratic extrema search scheme gives very reasonable execution times in practice, even for degrees in the order of thousands, as we will see in Section 7.2. Nevertheless, let us mention that potential speedups are still possible; because of *apriori* knowledge on the *exact* number of evaluations of E involving formula (9), it is feasible to use fast multipole methods for evaluating polynomial interpolants [22] at n arbitrary points. If ε is the numerical precision for our computations, such methods need $O(n \log(1/\varepsilon))$ operations at each iteration. Parallelization is possible here as well; the $O(N_{\max} n)$ polynomial evaluations can be split in $O(N_{\max})$ *independent* applications of the fast multipole method on batches of n points.

6.3. **The new reference set.** Each time we compute a valid eigenvalue y, we add it to the set of potential extrema $\tilde{\mathbf{x}}$ only if $|E(\arccos(y))| \geqslant \delta$. We also add to $\tilde{\mathbf{x}}$ the extremities of each subinterval if the error they yield is larger or equal in absolute value than the current minimax error δ. This allows us to successfully treat cases where band edges of X are in the final reference set, which is always true for type I stopband and passband filters (see [38, Ch. 7.7.1] for a proof).

Assuming the elements of $\tilde{\mathbf{x}}$ are ordered, for each subset of consecutive values where the error has the same sign, we only keep one element with largest absolute error. The new reference $\mathbf{x}'$ is then constructed by taking $n+2$ elements of $\tilde{\mathbf{x}}$ where the error *alternates* in sign and has the *largest* absolute values. A more detailed presentation on how to do this is available in [3, Sec. 4.1]. This strategy is the most robust one we tested. For other discussions, see [6, Ch. 15.3.4] and [39, Sec. 2.2].

6.4. **Retrieving the filter coefficients.** Upon convergence, we compute the coefficients $(h_k)_{0 \leqslant k \leqslant n}$ of the final frequency response $H(\omega)$ (eq. (2)). As we already saw, we can do this in a numerically stable way with a quadratic number of computations by using formula (10) and Clenshaw's algorithm or in $O(n \log n)$ operations with a DCT-based interpolation scheme. Similar to Remark 6.1, the total cost is dictated by how we evaluate the barycentric form of $H(\omega)$ at the n-th order Chebyshev nodes: $O(n^2)$ operations, since we use formula (9) at each node. With the evaluation scheme of [22], we would require $O(n \log(n/\varepsilon))$ operations (fast polynomial evaluation + DCT-based Chebyshev interpolation).

Figure 6 shows a small sample of the minimax approximation error E, for Example 1.4, *after* the filter coefficients have been computed using Clenshaw's recurrence relations. Although the degree $n = 53248$ is quite large, the computations are *numerically accurate*, with the equioscillations mentioned inside Theorem 2.3 clearly visible.

7. IMPLEMENTATION DETAILS

In this section we give practical information on how someone can use our routines and compare them to equivalent ones available in widely used signal processing packages.

To summarize the previous three sections, each iteration of the exchange algorithm takes $O(n^2)$ operations with our code (see https://github.com/sfilip/firpm). Choosing the starting reference takes $O(n)$ operations if uniform initialization is used, but this cost can jump to $O(n^2)$ with the approach from Section 4.3 or $O(n^3)$ operations for Section 4.2. Since we limit the use of AFP-based initialization to moderate degrees, the cubic cost is not problematic in practice. As discussed in Section 4.4, these more expensive strategies are most of the time worthwhile and sometimes even necessary. Also, the computational bottleneck of our implementation (the extrema search of Section 6.2) is *embarrassingly parallel*. Exploiting it is very effective in practice (see Table 7).

7.1. **User interface.** Our code requires a small number of external libraries in order to work. Besides calling `MPFR` inside the multiple precision version, we also use the `Eigen` library [27] to perform all the eigenvalue computations when searching for a new reference set. Because it is designed with template metaprogramming techniques, the code calling `Eigen` requires little to no changes between the different versions. To parallelize the extrema search in Section 6.2, we use OpenMP [18].

All three versions of our code have an almost identical interface, inspired by the style used for the MATLAB implementation of the Parks-McClellan algorithm. As such, we will focus on describing only the double-precision version.

The function for designing type I and II filters has the following prototype:

```
PMOutput firpm(std::size_t N, std::vector<double>const& f,
               std::vector<double>const& a, std::vector<double>const& w,
               double epsT = 0.01, int Nmax = 4);
```

where

- `N`+1 is the number of coefficients of the designed filter;
- `f` is the vector of the frequency band edges of $\Omega \subseteq [0, \pi]$, normalized to $[0, 1]$, and given in increasing order;
- `a` contains the desired amplitudes at each of the points from `f`;
- `w` is the vector of weights on each band specified by `f` (its size is half that of `f` and `a`);
- `epsT` convergence parameter threshold from Step 5 in the Algorithm from Section 2.2. The default value is taken from [6, Ch. 15.3].
- `Nmax` designates the degree of the Chebyshev interpolants used for the extrema search from Section 6.2, with a default value of 4.

For instance, designing a degree $n = 100$, type I filter adhering to the specification given in Example 1.1, results in the following, valid C++11, function call:

```
PMOutput res = firpm(200, {0, 0.4, 0.5, 1}, {1, 1, 0, 0}, {1, 1});
```

The returned `PMOutput` object `res` has several member variables that can be useful to a filter designer. The vector `res.h` corresponds to the final coefficients of the filter transfer function from equation (1), while `res.x` is the final reference vector, belonging to X. The number of iterations required for convergence are stored in the variable `res.iter`, while the value of the final reference error δ_X is denoted with `res.delta`.

The `firpm` functions use uniform initialization by default. To use reference scaling or approximate Fekete points, we supply the functions `firpmRS` and `firpmAFP`. The `firpmRS` versions allow the user to choose between uniform and AFP-based initialization at the lowest level.

7.2. **Timings.** There are multiple implementations of the Parks-McClellan algorithm available, so we compared our approach to those we believe are the most widely used and/or robust in practice. Some results are given in Table 6. Because they are written in different languages, there are bound to be some differences in terms of execution times. To make matters as *fair* as possible, we *disabled parallelization* of the extrema search in our code and went for *uniform initialization* in all the test cases. Default grid size parameters (see the discussion from the beginning of Section 3) were used for the routines in the last four columns and a default value of $N_{\max} = 4$ for our implementation.

The two optimized routines from [3] are at heart efficient rewritings of the original code of [36], the only difference between the two being how the reference set gets updated at each iteration. Together with our implementation, they were the only ones that were able to converge on 7 out of the 8 test cases. Even so, our code is the most robust one in terms of execution time. We also tested the corresponding routine from Scilab 5.5.1 (written in Fortran), but did not achieve convergence for any of the test cases.

The effects of starting with a better reference (via *scaling* or *approximate Fekete points*) and *parallelization* of the extrema search are showcased in Table 7. As we already emphasized, the greatest gain is for the last two examples, where the filter degrees are larger. In the case of Example 1.4, for which *only* our routine converged, enabling parallelization on the quad core machine we used for testing, resulted in our code running 3.3 times faster than the purely sequential version.

TABLE 6. Runtime (real time) comparisons of our IEEE-754 double-precision implementation of the Parks-McClellan algorithm with those available in GNURadio 3.7.8.1 (also written in C++), MATLAB R2014b, the one from SciPy 0.16.1 (python code) and two other MATLAB implementations. The same machine, a quad core 3.6 GHz 64-bit Intel Xeon(R) E5-1620 running Linux 4.1.12 (15.9), was used for all the tests. The average execution times (in seconds) of running each piece of code 50 times, are given. When a routine did not converge to the minimax result, NC was used in place of the execution time. The a/b cells in the last column correspond to the two implementations described in [3].

Example (degree)	Uniform (sequential)	GNURadio	MATLAB	SciPy	[3]
1.1 ($n = 50$)	0.0034	**0.0029**	0.0532	0.0711	0.0384/0.0098
1.1 ($n = 80$)	**0.0045**	0.0068	0.1174	0.2587	0.0416/0.0316
1.1 ($n = 100$)	**0.0073**	NC	0.1491	0.3511	0.0451/0.0396
1.2 ($n = 50$)	**0.0022**	0.0023	0.0681	0.0971	0.0395/0.0174
1.2 ($n = 80$)	**0.0069**	NC	0.2002	0.3492	0.0761/0.0293
1.2 ($n = 100$)	**0.0301**	NC	NC	NC	0.2599/0.0521
1.3 ($n = 520$)	**0.2375**	NC	NC	NC	1.1691/2.7011
1.4 ($n = 53248$)	NC	NC	NC	NC	NC/NC

TABLE 7. Timings showing the effect of running our code with different options. As for Table 6, the numerical values represent the averages in seconds over 50 executions. For the first seven lines, the double-precision version of our routine was used, while for the last one `long double` 80-bit operations were carried out. In all cases, our code was compiled using `g++5.2.0` with `-O3 -DNDEBUG` level optimizations.

Example (degree)	Uniform (sequential)	Uniform (parallel)	Scaling (sequential)	Scaling (parallel)	AFP (sequential)	AFP (parallel)
1.1 ($n = 50$)	0.0034	0.0031	0.0021	0.0018	0.0021	0.0011
1.1 ($n = 80$)	0.0045	0.0035	0.0029	0.0012	0.0029	0.0015
1.1 ($n = 100$)	0.0073	0.0059	0.0099	0.0041	0.0035	0.0023
1.2 ($n = 50$)	0.0022	0.0019	0.0024	0.0015	0.0026	0.0014
1.2 ($n = 80$)	0.0069	0.0064	0.0058	0.0041	0.0029	0.0028
1.2 ($n = 100$)	0.0301	0.0151	0.0123	0.0066	0.0113	0.0071
1.3 ($n = 520$)	0.2375	0.1613	0.1632	0.0725	0.0731	0.0693
1.4 ($n = 53248$)	NC	NC	537.8	162.6	NC	NC

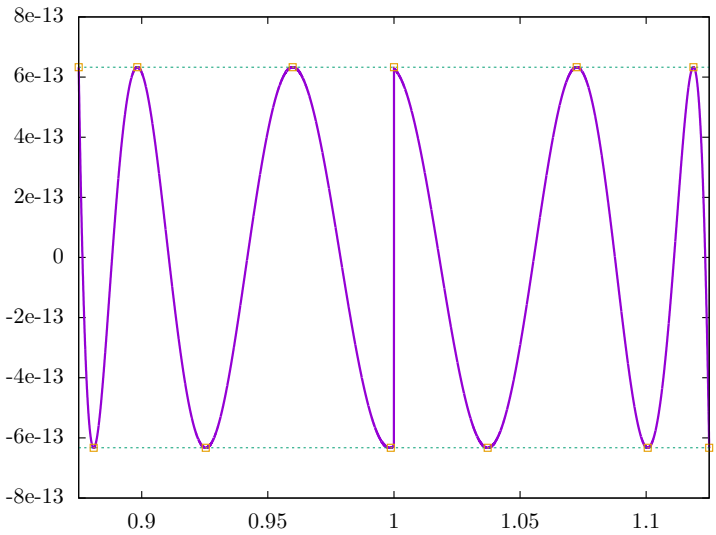

FIGURE 7. Degree $n = 10$ minimax approximation of $f(x) = \ln(x)e^{\sin(x)}$ in terms of relative error.

7.3. **Other applications.** Despite the focus on filter design problems, the routines we provide are more general. They can be used for arbitrary weighted minimax polynomial approximation problems involving multi-interval domains. Such a scenario arises, for instance, when minimizing the *relative error* of approximation for a function with zero(s).

As an example, consider the function $f(x) = \ln(x)e^{\sin(x)}$. We want to find the polynomial $p^\star$ of degree at most $n = 10$, which *best* approximates f over $[1 - 2^{-3}, 1 + 2^{-3}]$ in terms of relative error (corresponding weight function $w(x) = 1/|f(x)|$). Since f has a zero at $x = 1$, in a double-precision context, we can split the domain into two parts:

$$[1 - 2^{-3}, 0.99999999999999988897769753748434595763683319091796875]$$

and

$$[1.00000000000000022204460492503130808472633361816406 25, 1 + 2^{-3}],$$

that is, remove all the values between the largest double-precision value smaller than 1 and the smallest double-precision value greater than 1.

Figure 7 shows the minimax error $(f(x) - p^\star(x))/|f(x)|$ obtained by running our implementation of the exchange algorithm on this new two interval domain, together with the $n + 2 = 12$ equioscillation points around the extremal values $\pm 6.32824 \cdot 10^{-13}$.

8. Conclusion

In this article we have presented several ideas that aid in the development of weighted minimax polynomial approximations, with a central focus on optimal linear-phase FIR filters. These contributions amount to the following:

- introduce two new initialization strategies (Sections 4.2 and 4.3); they ensure the convergence of our routine in cases where all other implementations fail to work, while frequently incurring significant speedups as well (consequence of a small number of iterations required for convergence);
- present pertinent analysis tools which help diagnose when the exchange algorithm is prone to numerical problems in practice (Section 5.2);
- introduce a variation of a well-established root-finding approach [12,57] which allows one to design FIR filters in an efficient way (Section 6.2).

Equally noteworthy is the fact that this study has helped us develop an efficient and highly parallel software library. As we saw throughout all of the examples we considered in this text, our routine outperforms other existing codes in terms of scalability and numerical accuracy.

Since at the end of the filter synthesis framework, the final coefficients are quantized, the result of the Parks-McClellan algorithm needs to be adequately *modified*. We are currently working on efficient ways of solving this problem. We also hope to investigate if similar ideas to the ones we presented here can be applied to other digital signal processing tasks, like those requiring FIR filters with complex coefficients [29, 30] or IIR filters [48–51].

References

[1] Abu-Al-Saud, W. A., and Stüber, G. L. Efficient wideband channelizer for software radio systems using modulated PR filterbanks. *IEEE Transactions on Signal Processing, 52*, 10 (Oct 2004), 2807–2820.

[2] Adams, J. W., and Wilson, A. N., J. On the fast design of high-order FIR digital filters. *IEEE Transactions on Circuits and Systems 32*, 9 (Sep 1985), 958–960.

[3] Ahsan, M., and Saramäki, T. Two Novel Implementations of the Remez Multiple Exchange Algorithm for Optimum FIR Filter Design. http://www.intechopen.com/download/pdf/39374, 2012.

[4] Antoniou, A. Accelerated procedure for the design of equiripple nonrecursive digital filters. *IEE Proceedings G, Electronic Circuits and Systems 129*, 1 (February 1982).

[5] Antoniou, A. New Improved Method for the Design of Weighted-Chebyshev, Nonrecursive, Digital Filters. *IEEE Transactions on Circuits and Systems 30*, 10 (Oct 1983), 740–750.

[6] Antoniou, A. *Digital Signal Processing: Signals, Systems, and Filters*. McGraw-Hill Education, 2005.

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

[8] Bonzanigo, F. Some improvements to the design programs for equiripple FIR filters. In *IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP '82.* (May 1982), vol. 7, pp. 274–277.

[9] Bos, L. P., Calvi, J.-P., Levenberg, N., Sommariva, A., and Vianello, M. Geometric weakly admissible meshes, discrete least squares approximations and approximate Fekete points. *Mathematics of Computation 80*, 275 (2011), 1623–1638.

[10] Bos, L. P., and Levenberg, N. On the calculation of approximate Fekete points: the univariate case. *Electronic Transactions on Numerical Analysis 30* (2008), 377–397.

[11] Boyd, J. P. Computing the zeros, maxima and inflection points of Chebyshev, Legendre and Fourier series: solving transcendental equations by spectral interpolation and polynomial rootfinding. *Journal of Engineering Mathematics 56*, 3 (2006), 203–219.

[12] Boyd, J. P. Finding the Zeros of a Univariate Equation: Proxy Rootfinders, Chebyshev Interpolation, and the Companion Matrix. *SIAM Review 55*, 2 (2013), 375–396.

[13] Boyd, J. P., and Gally, D. H. Numerical experiments on the accuracy of the Chebyshev-Frobenius companion matrix method for finding the zeros of a truncated series of Chebyshev polynomials. *Journal of Computational and Applied Mathematics 205*, 1 (2007), 281 – 295.

[14] Businger, P., and Golub, G. H. Linear least squares solutions by Householder transformations. *Numerische Mathematik 7*, 3 (1965), 269–276.

[15] Calvi, J.-P., and Levenberg, N. Uniform approximation by discrete least squares polynomials. *Journal of Approximation Theory 152*, 1 (May 2008), 82–100.

[16] Çivril, A., and Magdon-Ismail, M. On selecting a maximum volume sub-matrix of a matrix and related problems. *Theoretical Computer Science 410*, 47-49 (Nov. 2009), 4801–4811.

[17] Cheney, E. W. *Introduction to Approximation Theory*. AMS Chelsea Publishing Series. AMS Chelsea Pub., 1982.

[18] Dagum, L., and Menon, R. OpenMP: an industry standard API for shared-memory programming. *IEEE Computational Science Engineering 5*, 1 (Jan. 1998), 46–55.

[19] Day, D., and Romero, L. Roots of Polynomials Expressed in Terms of Orthogonal Polynomials. *SIAM Journal on Numerical Analysis 43*, 5 (2005), 1969–1987.

[20] De Dinechin, F., Istoan, M., and Massouri, A. Sum-of-product architectures computing just right. In *2014 IEEE 25th International Conference on Application-specific Systems, Architectures and Processors (ASAP)* (June 2014), pp. 41–47.

[21] Driscoll, T. A., Hale, N., and Trefethen, L. N. *Chebfun Guide*. Pafnuty Publications, Oxford, 2014.

[22] Dutt, A., Gu, M., and Rokhlin, V. Fast algorithms for polynomial interpolation, integration, and differentiation. *SIAM Journal on Numerical Analysis 33*, 5 (1996), 1689–1711.

[23] Ebert, S., and Heute, U. Accelerated design of linear or minimum phase FIR filters with a Chebyshev magnitude response. *IEE Proceedings G, Electronic Circuits and Systems 130*, 6 (December 1983), 267–270.

[24] Embree, M., and Trefethen, L. N. Green's Functions for Multiply Connected Domains via Conformal Mapping. *SIAM Review 41*, 4 (Dec. 1999), 745–761.

[25] Floater, M. S., and Hormann, K. Barycentric rational interpolation with no poles and high rates of approximation. *Numerische Mathematik 107*, 2 (2007), 315–331.

[26] Fousse, L., Hanrot, G., Lefèvre, V., Pélissier, P., and Zimmermann, P. MPFR: A Multiple-precision Binary Floating-point Library with Correct Rounding. *ACM Trans. Math. Softw. 33*, 2 (2007).

[27] Guennebaud, G., Jacob, B., et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

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

[29] Karam, L. J., and McClellan, J. H. Complex Chebyshev approximation for FIR filter design. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing 42*, 3 (Mar 1995), 207–216.

[30] Karam, L. J., and McClellan, J. H. Design of optimal digital FIR filters with arbitrary magnitude and phase responses. In *ISCAS '96., IEEE International Symposium on Circuits and Systems* (May 1996), vol. 2, pp. 385–388.

[31] Kodek, D. M. LLL Algorithm and the Optimal Finite Wordlength FIR Design. *IEEE Transactions on Signal Processing 60*, 3 (Mar. 2012), 1493–1498.

[32] Levin, E., and Saff, E. B. Potential theoretic tools in polynomial and rational approximation. In *Harmonic Analysis and Rational Approximation*, J.-D. Fournier, J. Grimm, J. Leblond, and J. R. Partington, Eds., vol. 327 of *Lecture Notes in Control and Information Science*. Springer Berlin Heidelberg, 2006, pp. 71–94.

[33] Mascarenhas, W. F. The stability of barycentric interpolation at the Chebyshev points of the second kind. *Numerische Mathematik 128*, 2 (2014), 265–300.

[34] Mascarenhas, W. F., and Camargo, A. On the backward stability of the second barycentric formula for interpolation. *Dolomites Research Notes on Approximation 7* (2014), 1–12.

[35] McClellan, J. H., and Parks, T. W. A personal history of the Parks-McClellan algorithm. *IEEE Signal Processing Magazine 22*, 2 (March 2005), 82–86.

[36] McClellan, J. H., Parks, T. W., and Rabiner, L. A computer program for designing optimum FIR linear phase digital filters. *IEEE Transactions on Audio and Electroacoustics 21*, 6 (Dec 1973), 506–526.

[37] Muller, J. M., Brisebarre, N., de Dinechin, F., Jeannerod, C. P., Lefèvre, V., Melquiond, G., Revol, N., Stehlé, D., and Torres, S. *Handbook of Floating-Point Arithmetic*. Birkhäuser Boston, 2009.

[38] Oppenheim, A. V., and Schafer, R. W. *Discrete-Time Signal Processing*. Prentice-Hall Signal Processing Series. Prentice Hall, 2010.

[39] Pachón, R., and Trefethen, L. N. Barycentric-Remez algorithms for best polynomial approximation in the Chebfun system. *BIT Numerical Mathematics 49*, 4 (2009), 721–741.

[40] Parks, T. W., and McClellan, J. H. Chebyshev Approximation for Nonrecursive Digital Filters with Linear Phase. *IEEE Transactions on Circuit Theory 19*, 2 (March 1972), 189–194.

[41] Pelloni, D., and Bonzanigo, F. On the design of high-order linear phase FIR filters. *Digital Signal Processing 1* (1980), 3–10.

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

[43] Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. *Numerical Recipes: The Art of Scientific Computing*, 3 ed. Cambridge University Press, 2007.

[44] Psarakis, E. Z., and Moustakides, G. V. A robust initialization scheme for the Remez exchange algorithm. *IEEE Signal Processing Letters 10*, 1 (Jan 2003), 1–3.

[45] Rabiner, L., Kaiser, J., and Schafer, R. W. Some Considerations in the Design of Multiband Finite-Impulse-Response Digital Filters. *IEEE Transactions on Acoustics, Speech and Signal Processing 22*, 6 (Dec 1974), 462–472.

[46] Remes, E. Sur le calcul effectif des polynômes d'approximation de Tchebichef. *Comptes rendus hebdomadaires des séances de l'Académie des Sciences*, 199 (1934), 337–340.

[47] Rutishauser, H. *Vorlesungen über Numerische Mathematik*. Birkhäuser Basel, Stuttgard, 1976. English translation, 1990. *Lectures on Numerical Mathematics*. Walter Gautschi, ed., Birkhäuser Boston.

[48] Saramäki, T. An efficient Remez-type algorithm for the design of optimum IIR filters with arbitrary partially constrained specifications. In *ISCAS '92., IEEE International Symposium on Circuits and Systems* (May 1992), vol. 5, pp. 2577–2580.

[49] Saramäki, T. Generalizations of classical recursive digital filters and their design with the aid of a Remez-type algorithm. In *ISCAS '94., IEEE International Symposium on Circuits and Systems* (May 1994), vol. 2, pp. 549–552.

[50] Selesnick, I. W. New exchange rules for IIR filter design. In *ICASSP-97., IEEE International Conference on Acoustics, Speech, and Signal Processing* (Apr 1997), vol. 3, pp. 2209–2212.

[51] Selesnick, I. W., Lang, M., and Burrus, C. S. Magnitude squared design of recursive filters with the Chebyshev norm using a constrained rational Remez algorithm. In *Sixth IEEE Digital Signal Processing Workshop,* (Oct 1994), pp. 23–26.

[52] Shen, J., Strang, G., and Wathen, A. J. The Potential Theory of Several Intervals and Its Applications. *Applied Mathematics and Optimization 44*, 1 (Jan. 2001), 67–85.

[53] Shpak, D. J., and Antoniou, A. A generalized Remez method for the design of FIR digital filters. *IEEE Transactions on Circuits and Systems 37*, 2 (Feb 1990), 161–174.

[54] Sommariva, A., and Vianello, M. Computing approximate Fekete points by QR factorizations of Vandermonde matrices. *Computers & Mathematics with Applications 57*, 8 (Apr. 2009), 1324–1336.

[55] Sommariva, A., and Vianello, M. Approximate Fekete points for weighted polynomial interpolation. *Electronic Transactions on Numerical Analysis 37* (2010), 1–22.

[56] Strang, G. The discrete cosine transform. *SIAM Review 41*, 1 (1999), 135–147.

[57] Trefethen, L. N. *Approximation Theory and Approximation Practice*. Society for Industrial and Applied Mathematics, 2013.

[58] Veidinger, L. On the numerical determination of the best approximation in the Chebyshev sense. *Numerische Mathematik 2*, 1 (1960), 99–105.

[59] Webb, M., Trefethen, L. N., and Gonnet, P. Stability of Barycentric Interpolation Formulas for Extrapolation. *SIAM J. Scientific Computing 34*, 6 (2012).

[60] Zahradnik, P., and Vlcek, M. Robust analytical design of equiripple comb FIR filters. In *IEEE International Symposium on Circuits and Systems, 2008. ISCAS 2008* (May 2008), pp. 1128–1131.