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 such codes fail to work. In this paper, we introduce a new implementation of this algorithm. It is based on three main ingredients: (1) a new heuristic initialization strategy that generally improves the convergence properties of the Parks-McClellan routine, (2) numerically stable barycentric Lagrange interpolation formulas, and (3) colleague matrix-based rootfinding algorithms. We argue that our approach is very robust in practice, even for hard to design problems. The result, an open source C++ library, is capable of constructing filters where the final degree is more than 50000, outperforming other implementations.

1. Introduction

Digital filtering operations are vital 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);
- synthesize the obtained filter in hardware/software.

The Parks-McClellan exchange algorithm [35] is concerned with the first step. Its output is a set of real valued coefficients which describes a filter that is, in general, very close to a *theoretically optimal* one. Because all signal processing hardware makes use of numeric formats with a limited representation range, the quantization step proves to be necessary for effectively using the coefficients. Fixed-point arithmetic is the usual choice, mostly because it can be implemented more efficiently in hardware and amounts to lower circuit costs, than say, floating-point arithmetic. The last step deals with how arithmetic operations performed by the filter are actually executed in hardware/software.

Our interest in the Parks-McClellan filter design method began while the author was looking into efficient algorithms for solving the quantization problem with fixed-point coefficients. We feel that there are still pertinent questions, not fully answered, when talking about the Parks-McClellan algorithm. The most important ones we are trying to address with this paper are *when* and *why* it behaves well in practice.

The answers are given by *how* the main computational tasks of the algorithm are carried out. It will become obvious from Sections 3.1 and 4 that an inspired choice for initializing the exchange algorithm is maybe the most crucial element for practical convergence to the final result. The second major ingredient is polynomial interpolation. Barycentric Lagrange interpolation has been the preferred choice since the beginning [35]. Its use is encouraged for families of interpolation points with a small Lebesgue constant (for a definition see Section 5.2), where barycentric interpolation has excellent numerical stability properties [7,25,29]. The last ingredient is fast and accurate root-finding algorithms. In Section 6, we *adapt* to the FIR filter design problem, in a highly scalable way, so-called Chebyshev-proxy root-finding methods [11,50]. This choice is influenced by the success that Chebyshev-proxy methods enjoy in the Chebfun [19] MATLAB library, and in particular in the `remez` command, a robust implementation of the Remez exchange algorithm for computing best polynomial approximations on a compact interval [34]. Note that barycentric Lagrange interpolation is also one of the cornerstones of Chebfun.

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. We chose them because they prove to be hard or impossible to solve using current *de facto* implementations of the exchange algorithm. All the theoretical elements needed to understand these specifications are reviewed in the next section.

We start with two examples used in [40].

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 [53] 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 try using:

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 $\left[\frac{3}{8192}\pi, \pi\right]$.

An important byproduct of our research is an open source implementation of the Parks-McClellan algorithm, written in C++[1]. We offer three versions. The first one uses IEEE-754 double-precision floating-point arithmetic, while the second one requires `long double` operations, which are specified in the C11/C++11 language standards. On x86 machines and compilers, this format corresponds to 80-bit extended-precision floating-point numbers. The other version uses `MPFR` [22] multiple precision formats. It allows one to tackle a wider range of problems, but since multiple precision arithmetic in `MPFR` is implemented in software, it comes at the cost of much slower execution times. All of the numerical tests in the sequel are available together with the source code.

We believe the double-precision version of our routine to be sufficient for *most* practical uses. Nevertheless, in very particular instances, having larger precision alternatives can prove to be a necessity, which is why we provide them. For us, this was the case with the tests involving Example 1.4, where we had to use at least `long double` arithmetic in order to obtain an accurate final result. All the other test scenarios we considered were well in the reach of IEEE-754 double-precision arithmetic.

The next section opens with a condensed presentation of some important filtering concepts we will be using throughout the text. We then state the general form of the Parks-McClellan algorithm. In Section 3, we give an overview of some of the advances and practical issues that have been attributed to this method over the years. This will help us in Sections 4 to 6, where we present our approach for each step of the algorithm. A large emphasis is given to the practical implications of our decisions. We stress the fact that our focus was to prioritize numerically stable results wherever possible, while also taking into account performance needs. Section 7 compares the computational speed of our routine to other existing implementations, while concluding remarks are made in Section 8.

2. Theoretical background and problem statement

Digital filters are computational blocks operating on complex valued sequences, called discrete-time signals. If we denote such an input signal with $x[n], n \in \mathbb{Z}$, then the output of a causal linear time-invariant filter acting on it can be expressed by

$$y[n] = \sum_{k=0}^{N} b_k x[n-k] - \sum_{k=1}^{M} a_k y[n-k], \tag{1}$$

where $M, N \in \mathbb{N}$. We make the assumption that a_k, b_k are real numbers.

Algebraically, digital filters can be understood by means of the z-transform. For a given sequence x, its z-transform is defined as

$$X(z) = \mathcal{Z}\{x[n]\} = \sum_{n=-\infty}^{\infty} x[n] z^{-n}, z \in \mathbb{C}.$$

Applying it to (1), we obtain

$$Y(z) = \sum_{k=0}^{N} b_k z^{-k} X(z) - \sum_{k=1}^{M} a_k z^{-k} Y(z) = \frac{\sum_{k=0}^{N} b_k z^{-k}}{1 + \sum_{k=1}^{M} a_k z^{-k}} X(z) = H(z) X(z), \tag{2}$$

where $X(z)$ and $Y(z)$ are the corresponding z-transforms of $x[n]$ and $y[n]$. $H(z)$ is called the *transfer function* of the filter. Based on the form of $H(z)$ we can talk about two large families of filters:

(1) **FIR** (finite impulse response): the transfer function is a polynomial in z^{-1} (*i.e.*, all the a_k coefficients are zero);
(2) **IIR** (infinite impulse response): $H(z)$ is a rational fraction in z^{-1}.

[1] available at https://github.com/sfilip/firpm

TABLE 1. FIR generalized linear phase filter types

Type	Length $(N+1)$	Symmetry	Delay (d)	Phase (α)
I	odd $(2m+1)$	symmetrical	integer (m)	0
II	even $(2m)$	symmetrical	fractional $(m-1/2)$	0
III	odd $(2m+1)$	antisymmetrical	integer (m)	$\pi/2$
IV	even $(2m)$	antisymmetrical	fractional $(m-1/2)$	$\pi/2$

The frequency domain representation is also an important measure for the behavior of a signal/filter. It is given by the Discrete-Time Fourier Transform (DTFT), which is just the restriction of the z-transform to the unit circle (*i.e.*, we take $z = e^{i\omega}, \omega \in [-\pi, \pi]$ and i is the imaginary unit). $H(e^{i\omega})$ is known as the *frequency response* of the filter. Because we view $H(e^{i\omega})$ as a function in ω, for the rest of this paper we will make the following abuse of notation and use $H(\omega)$ when talking about DTFT representations.[2]

There are various reasons as to why one would want to use a type of filter over the other (FIR versus IIR) and we refer the reader to [38, Ch. 7.1.1] for an introductory discussion on the subject. Here, our focus will be on FIR filters, and more particularly those that have generalized linear phase (*i.e.*, $H(\omega) = |H(\omega)|\, e^{-i\omega d + i\alpha}, d, \alpha \in \mathbb{R}$). Linear phase is an important property for many applications in data transmission and audio signal processing. It is very easy to have with FIR filters (it can be intrinsic to their representation), while for IIR filters it has to be imposed as a constraint during the design process.

Usually, four combinations of d and α, which are summarized in Table 1, are of interest. The third column on symmetry is to be interpreted in terms of the middle coefficient(s) of the transfer function. For simplicity, we will focus our presentation mostly on type I filters, but the developed theory will work just as well for types II to IV. See [6, Ch. 15.8-15.9] for more details on all four filter types.

2.1. **Optimal FIR filters and polynomial approximation.** In the case of type I filters, we have $H(z) = \sum_{k=0}^{2n} b_k z^{-k}$, where $b_{n-k} = b_{n+k}, k = 1, \ldots, n$. Taking the delay into account, we can write $H(z) = z^{-n} H_d(z)$, with

$$H_d(z) = \sum_{k=-n}^{n} c_k z^k = c_0 + \sum_{k=-n}^{-1} c_k z^k + \sum_{k=1}^{n} c_k z^k = c_0 + \sum_{k=1}^{n} c_k (z^{-k} + z^k), \tag{3}$$

and $c_k = b_{n+k}, k = -n, \ldots, n$. The corresponding frequency response is

$$H_d(\omega) = \sum_{k=0}^{n} h_k \cos(\omega k), \tag{4}$$

where $h_0 = c_0$ and $h_k = 2c_k, 1 \leqslant k \leqslant n$. With respect to the initial coefficients of $H(\omega)$, we have $b_n = h_0$ and $b_{n-k} = b_{n+k} = h_k/2$, for $1 \leqslant k \leqslant n$. Similar relations between the coefficients of $H(\omega)$ and those of a corresponding $H_d(\omega)$ can be derived for filters of types II to IV [6, Ch. 15.8].

We can now state the problem we are trying to solve in terms of $H_d(\omega)$.

Problem 2.1 (Equiripple (or minimax) FIR filter design)**.** *Let Ω be a closed 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 $H_d(\omega) = \sum_{k=0}^{n} h_k \cos(\omega k)$ such that the weighted error function $E(\omega) = W(\omega)\,(D(\omega) - H_d(\omega))$ 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.

A typical example of such a problem is the design of lowpass filters, like the one in Figure 1, with $\Omega = [0, \omega_p] \cup [\omega_s, \pi]$,

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

where δ_1 and δ_2 denote the maximum errors allowed over $[0, \omega_p]$ and $[\omega_s, \pi]$. For a discussion on which of these parameters $(\omega_p, \omega_s, \delta_1, \delta_2)$ is fixed or unknown during the design process, the reader is referred to [33, Ch. 7.7].

Stopbands represent frequency ranges that we want eliminated from the output, while passbands are those components that we want to let pass unaltered through the filter. Between them, we find transition bands. While in theory we are not usually concerned with what value $H_d(\omega)$ takes in such regions, we will see in Section 3.3 that this should not always be the case in practice.

[2]The variable names (ω and z) should make it clear in a particular context whether we are talking about a Fourier or a z-transform.

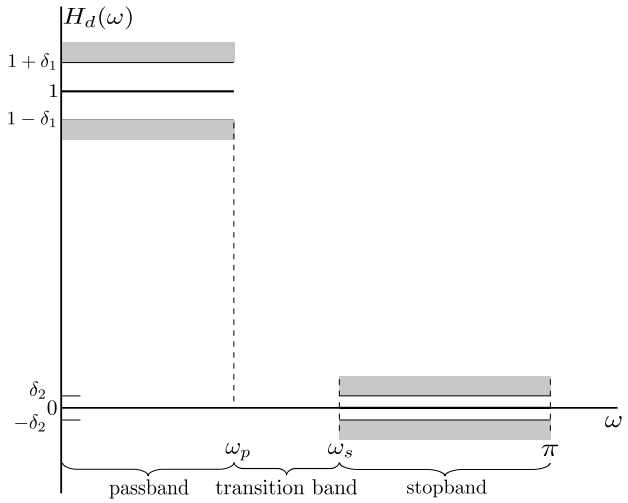

FIGURE 1. Tolerance scheme and ideal frequency response for a lowpass filter

Transition bands are crucial because their absence incurs jump discontinuities in the functions to approximate, potentially giving rise to the so-called Gibbs-Wilbraham phenomenon. In such a setting, the error function $E(\omega)$ exhibits large overshoots around the discontinuity, which do not disappear, even by increasing the degree of $H_d(\omega)$. The specifics of the Gibbs-Wilbraham phenomenon are an interesting subject and they are relevant in many application scenarios. We point the reader to [50, Ch. 9] for an introduction and further references.

By applying the change of variable $x = \cos(\omega)$ in (4), we can view $H_d(\omega)$ as a polynomial in x of degree at most n (see [33, Ch. 7.7] for the details). Minimax (or Chebyshev) polynomial approximation problems like the one we just stated are a well studied topic in Approximation Theory [15, 37, 50]. Parks and McClellan were aware of this when they introduced their approach for tackling Problem 2.1 [31]. The starting point of their research is the following *qualitative* description of the solution.

Theorem 2.2 (Alternation theorem)**.** *In the context of Problem 2.1, a necessary and sufficient condition for $H_d(\omega)$ to be the **unique** transfer function of degree at most n that minimizes the weighted approximation error $\delta = \|E(\omega)\|_{\infty,\Omega}$ 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*

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

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

Proof. See [15, Ch. 3.4]. □

2.2. **The Parks-McClellan algorithm.** Together with a result due to de La Vallée Poussin (see for example [15, Ch. 3.4] for a precise statement), Theorem 2.2 can be used to derive iterative procedures which converge to the minimax result. Much of this work was done during the 1930s by Evgeny Remez [42]. Comprehensive accounts on the algorithms Remez introduced can be found in [15, Ch. 3.8] and [37, Ch. 8-10]. Parks and McClellan are credited with implementing a Remez exchange algorithm in the context of solving Problem 2.1. Their approach can be summarized as follows:

Parks-McClellan (Remez exchange) algorithm

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

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

(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 $H_d(\omega)$ and its associated alternating error δ, 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 $|E(\omega)| \geqslant \delta$.
(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(s) of $E(\omega)$ and for which the error alternates in sign.

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

Such a routine corresponds to what is frequently called a multipoint exchange algorithm (or second Remez algorithm). Some remarks about its convergence are made in [31], while in [6, Ch 15.4] it is stated that up to 12 iterations are usually required for designing two and three-band FIR filters.

How one deals with computing the current filter $H_d(\omega)$ and its error function $E(\omega)$ at Steps 2 and 3 is critical for making the exchange method *practical*. Parks and McClellan opted to use a barycentric form of classic Lagrange polynomial interpolation for this task. As we shall see in Section 5, this was a wise decision on their part.

3. Bibliographical study and state of the art

The first implementation of the Parks-McClellan algorithm that lets one design all four linear phase FIR filter types from Table 1 is presented in [32] and was written in Fortran. It works on a discretized 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 (*i.e.*, the default value is $16n$), in the hope that a solution for the discrete formulation is very close to the continuous one over Ω.

In this case, searching for the elements of $\boldsymbol{\omega'}$ is reduced to a relatively straightforward process. One only needs to evaluate $E(\omega)$ on the grid and choose from those points which are local extrema of $E(\omega)$ over G. This early Fortran code serves as starting point for current implementations, like the `firpm` function from MATLAB's Signal Processing Toolbox, or similar routines from Scilab, SciPy and GNURadio.

Over the years, certain shortcomings of the Parks-McClellan algorithm have been discussed in the literature. For the remainder of this section, we give an account on the most important ones. At the end of each subsection we also specify where that particular problem is *addressed* in the present article. For the sake of completeness, we also mention a problem we do not *explicitly* treat in the sequel.

3.1. Initialization issues: high-degree linear phase filters.

An interesting aspect regarding FIR filter design, which started to gain traction in the beginning of the 1980s, is the need of obtaining high-order filters, with $n = 200 \ldots 500$ or even larger. Examples of such application scenarios (along with references) are given in [20, 36], and include the design of transmultiplexers and filter banks for high resolution spectral analysis.

Crucial when dealing with such problems is how one picks the starting reference vector $\boldsymbol{\omega} = (\omega_k)_{0\leqslant k\leqslant n+1}$. The default method employed in practice is to assume that the elements of $\boldsymbol{\omega}$ are *uniformly* placed inside each band belonging to Ω. For details, we refer the reader to [6, Ch. 15.3.1].

Theoretically, the location of these points does not affect the convergence of the exchange algorithm, but in practice, their position has a huge impact on the number of required iterations. When dealing with large degrees, reducing the number of iterations can prove to be pivotal not only in terms of computation time, but also for the numerical precision needed to perform all the computations. Numerical stability in such scenarios is the focal point of Section 5.3.

For high-degree problems, uniform initialization does not usually work well. Based on experimental observations (see [20] for example), good initialization choices seem to be those where the extrema are taken to be more densely packed near the interior edges of the pass and stop bands. Articles like [2, 36, 48] try to compute such distributions.

Another interesting initialization tactic is presented in [40]. The idea 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 $\boldsymbol{\omega}$. For computing this filter, one has to solve a Toeplitz-plus-Hankel linear system. The main reason for choosing such a least-squares approach is that it behaves very well with respect to the minimax error measure (see [13, 14] for design methodologies). We have tested this approach on MATLAB R2014b, and when it works, it significantly reduces the number of required iterations, compared to the `firpm` routine from MATLAB's Signal Processing Toolbox. Issues tended to show up when the degrees got larger than $n = 200$ and the numerical Toeplitz-plus-Hankel solver ceased to behave in a stable way.

Our approach: We will address the initialization problem in Section 4.1 and present a new heuristic for reducing the number of iterations of the Parks-McClellan algorithm. As can be seen throughout the examples we consider in Sections 4 to 7, our tactic works very well for designing high degree filters. The main impact of this initialization choice is that it usually *improves* the numerical behavior of barycentric Lagrange interpolation. We shall see what we mean by this in Section 5.3.

3.2. Scalability issues: speeding up the extrema search.

For many inputs, the most computationally intensive part of the exchange algorithm proves to be the search for potential extrema. Articles that concentrate on this aspect

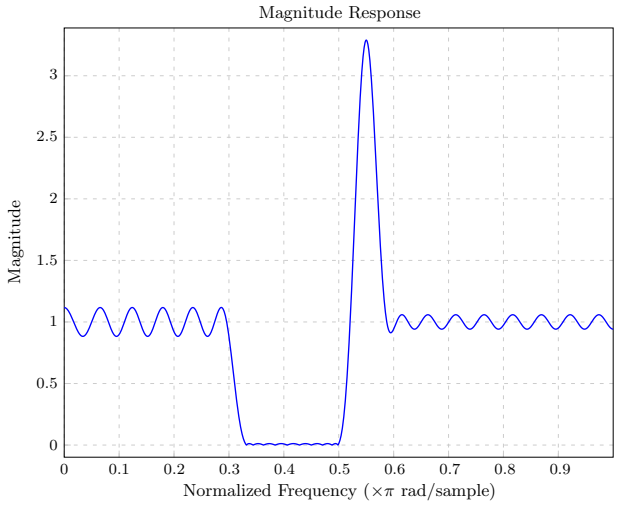

FIGURE 2. Multiband FIR filter exhibiting transition band anomalies

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 speeding up the extrema search.

Our approach: The author of [4, 5] admits that, although being fast, the approaches described in those papers come at the cost of slightly more convergence failures to the minimax result, than when using the code from [32]. As evidenced by the results presented in Section 6 and 7.2, we show how Chebyshev-proxy root-finding methods can be used, to great effect, inside the Parks-McClellan algorithm.

3.3. **Transition band anomalies.** Although the frequency response of a filter inside transition regions is not usually a design parameter, in practice, most of the time, we want it to be *monotonic*. When applying the algorithm we gave in the previous section, such a behavior is in many cases implicitly assured. Unfortunately, [41] remark that this does not hold in general. An example is shown in Figure 2. It denotes a three band, degree $n = 38$, type I 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.

Several extensions of the exchange algorithm are devoted to solving this issue. For example, [16] proposes placing constraints on the transition bands as well, forcing a design model on $[0, \pi]$, while [23, 30] suggest imposing lower and upper bound functions for the frequency response. Another interesting idea is presented in [48]. The design methods described there allow for the construction of filters where the error function equioscillates more times than the minimum number required by the Alternation Theorem.

Since our focus with the present paper is the *core* Parks-McClellan algorithm, we have not tried to handle transition band problems. Extending our ideas to methods like the ones mentioned in the previous paragraph should be the object of future work.

Over the course of the next three sections we go into the details of implementing each step of our Parks-McClellan routine.

4. A new heuristic for choosing the initial reference

Once a design specification is given, the first step of the algorithm we stated in Section 2.2 is to choose the initial reference vector $\boldsymbol{\omega}$. Since uniform initialization is the common choice, we also adhere to this strategy by default, using it when the filters to be designed are of *moderate* degree, say $n < 200$.

We should mention that, for the rest of the paper, we will mostly 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 also denote with X, where $X \subseteq [-1, 1]$, the transformed frequency bands in this case. When talking about x, it will be as $x = \cos(\omega)$. The results we are about to present are dependent on this trigonometric change of variable.

4.1. **A reference scaling idea.** We now introduce another initialization strategy which we found to work well for designing equiripple FIR filters of high degree. The basic idea is really simple and it relies on the following empirical observation: in many cases, the extrema distribution for minimax FIR filters that follow a certain specification tends to remain relatively invariant to the length of the filter. What we mean by this is that the number of extrema inside each band tends to scale well with the degree.

TABLE 2. Number of extrema in each band for the bandstop specification of Example 1.2

	$n = 50$	$n = 100$	$n = 200$
$[0, 0.2\pi]$	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

TABLE 3. Iteration count comparison for uniform initialization and reference scaling

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

Considering uniform initialization, let's look at what happens for Example 1.2. Table 2 uses a/b cells to show the number of reference points inside each band in the first iteration (the a value) and upon convergence (the b value), for three different degrees. We notice that, consistent with our previous statement, 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.

What this problem basically tells us is that, if, for a degree n approximation, we first compute the minimax filter 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 transfer function. The values of the new reference set $\mathbf{x}$ are established by taking the $\lfloor n/2 \rfloor + 2$ final references of the smaller filter and adding the remaining $n - \lfloor n/2 \rfloor$ points uniformly between them.

If needed, this strategy can be applied recursively until the smaller degree filter can be successfully computed using uniform initialization.

4.2. **Numerical examples.** Performance-wise, Table 3 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 one represents the scaling approach. We can observe that, apart from two exceptions, there is a considerable reduction in the number of iterations required for convergence. This frequently translates to smaller execution times as well, even though a filter of half the required degree is also computed. Still, as can be seen for Example 1.4, the biggest advantage of this heuristic is that it allows us to address, in a simple manner, filter design problems where uniform initialization gives rise to numerical instability issues and the algorithm does not converge (NC).

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. We did the same for Example 1.4, and ended up having to execute $6, 3$ and 3 iterations. In contrast, when we tried the least-squares approach from [40], it worked very well for the filters corresponding to Examples 1.1 and 1.2, but failed to produce any results for Examples 1.3 and 1.4. This is due to the fact that the quadratic solver used for the Toeplitz-plus-Hankel matrix ceased to behave in a stable manner for those degrees.

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.

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 $H_d(\omega)$ (Step 2). If we apply the Alternation Theorem on the current reference set, our unknowns satisfy the following equations:

$$E(\omega_i) = W(\omega_i)\left[D(\omega_i) - H_d(\omega_i)\right] = (-1)^i \delta', \qquad i = 0, \ldots, n+1, \tag{5}$$

where D denotes the ideal filter we are approximating, W is the corresponding weight function for the error and $\delta = |\delta'|$. By taking $H_d(\omega)$ in its cosine expansion from equation (4), we get the following linear system of equations in

$(h_k)_{0\leqslant k\leqslant n}$ and δ':

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

A unique solution to this system always exists, since the $(n+2)\times(n+2)$ matrix is non-singular (see [51] for a proof idea). According to [6, Ch. 15.3.3] and [33, Ch. 7.7.3], solving this system directly is not encouraged, since it can be computationally inefficient and is susceptible to numerical ill-conditioning.

In practice, one deduces δ' analytically and uses barycentric Lagrange interpolation to determine $H_d(\omega)$. For the rest of this section we will focus on presenting some important results related to this approach. What we believe is new are the comments referring to the numerical stability of barycentric Lagrange interpolation in the case of designing FIR filters.

5.1. **Barycentric Lagrange interpolation.** During the last twelve years, barycentric interpolation has become an active research topic in Numerical Analysis. This is mostly due to articles like [7,21], which describe the nice numerical properties of this interpolation scheme. Of note is the fact that [34] uses barycentric interpolation to implement a robust version of the Remez exchange algorithm inside Chebfun with the `remez` command. They use it to approximate continuous functions over a closed real interval with polynomials. The efforts of the present article with respect to the FIR filter design problem are in part an extension of their work.

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 $x_k\in[x^-,x^+], k=0,\ldots,n+1$, 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 [43], the barycentric forms of p are given by

$$p(x;\mathbf{x},\mathbf{y},\mathbf{w}) := p(x) = \ell(x)\sum_{k=0}^{n+1}\frac{w_k}{x-x_k}y_k, \tag{6}$$

known as the *first* barycentric interpolation formula, and

$$p(x;\mathbf{x},\mathbf{y},\mathbf{w}) = \frac{\displaystyle\sum_{k=0}^{n+1}\frac{w_k}{x-x_k}y_k}{\displaystyle\sum_{k=0}^{n+1}\frac{w_k}{x-x_k}}, \tag{7}$$

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

$$w_k = \frac{1}{\ell'(x_k)} = \frac{1}{\prod_{i\neq k}(x_k-x_i)}, \qquad k=0,\ldots,n+1. \tag{8}$$

If the barycentric weights w_k have been precomputed, both (6) and (7) can be evaluated with only $O(n)$ arithmetic operations at an arbitrary point x.

In our case, we are able to use barycentric interpolation to evaluate $H_d(\omega)$ because δ' can be computed beforehand using the formula

$$\delta_{\mathbf{x},\boldsymbol{\omega},\mathbf{w}} := \delta' = \frac{\displaystyle\sum_{k=0}^{n+1} w_k D(\omega_k)}{\displaystyle\sum_{k=0}^{n+1}\frac{(-1)^k w_k}{W(\omega_k)}} \tag{9}$$

(see [34, Sec. 3.3] for a proof idea).

The second barycentric formula is usually preferred for computing the current frequency response. We will justify why this is a good idea in the next subsections.

We have

$$H_d(x;\mathbf{x},\mathbf{c},\mathbf{w}) := H_d(\omega) = \frac{\displaystyle\sum_{k=0}^{n+1}\frac{w_k}{x-x_k}c_k}{\displaystyle\sum_{k=0}^{n+1}\frac{w_k}{x-x_k}} \tag{10}$$

where $x = \cos(\omega), c_k = D(\omega_k) - (-1)^k\dfrac{\delta_{\mathbf{x},\boldsymbol{\omega},\mathbf{w}}}{W(\omega_k)}$ and $\mathbf{w} \in \mathbb{R}^{n+2}$ is the weight vector whose elements are computed according to (8).

The evaluation of (9) and (10) can be 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 refer the reader to the formulas and ideas of [7, Sec. 7] and [34, Sec. 3.4]. The reader will notice that inside (10) we are using $n+2$ points to interpolate a polynomial of degree at most n. This is not by accident. 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 $\mathbf{x}$ [52].

5.2. **Numerical stability issues.** The numerical stability of (6) and (7) was first looked at in a rigorous manner in [25]. That article shows how the modified Lagrange formula (6) is backward stable in general, whereas (7) is shown to be forward stable for vectors of points $\mathbf{x} \in \mathbb{R}^{n+2}$ having a small Lebesgue constant, which we define shortly.

We recall that interpolation at a reference vector $\mathbf{x}$, with elements belonging to $[x^-, x^+]$, creates a correspondence between a function $f \in \mathcal{C}([x^-, x^+])$ and a polynomial p. This induces a linear projection operator $L_{\mathbf{x}}$ onto the space of polynomials with real coefficients of degree at most $n+1$. The norm of this operator

$$\Lambda_{x^-,x^+,\mathbf{x}} = \sup_{f\in\mathcal{C}([x^-,x^+])}\frac{\|L_{\mathbf{x}}f\|_{\infty,[x^-,x^+]}}{\|f\|_{\infty,[x^-,x^+]}} \tag{11}$$

is the Lebesgue constant associated with the reference vector $\mathbf{x}$. It offers a *quantitative* way to measure the *quality* of a family of points for doing polynomial interpolation.

Well-known choices of interpolation points which provide small Lebesgue constants are the Chebyshev nodes of the first and second kind. They are usually given in correspondence with the Chebyshev polynomials of the first and second kind. The n-th Chebyshev polynomial of the first kind is defined on $[-1,1]$ and is characterized by the equation

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

A recursive definition is also possible, and we have

$$T_0(x) = 1, T_1(x) = x, T_{n+2}(x) = 2xT_{n+1}(x) - T_n(x), \forall n \in \mathbb{N}.$$

In similar fashion, the Chebyshev polynomials of the second kind are given by

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

If we go back to formula (4), we see that the transfer function $H_d(\omega)$ is in fact a linear combination of Chebyshev polynomials, *i.e.*,

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

The Chebyshev nodes of the first kind are

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

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

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

A good, up to date presentation on the subject of Chebyshev interpolation can be found in [50]. For us, this topic is important, mainly because of its central role in the extrema search strategy we use in Section 6.

Although Higham's paper shows that the error analysis for formula (7) is in general not as favorable as the one for (6), for sets of points having a small Lebesgue constant, like Chebyshev nodes, using the second barycentric formula is *preferable* [7, 28].

5.3. **Barycentric interpolation works well in practice.** The main idea we are trying to reinforce is that one should first consider using (7) when doing polynomial interpolation.

5.3.1. *A general result.* In support of this statement, we also mention [29], where a new stability analysis for the second barycentric formula, complementing that of [25], is presented. They consider the more general setting of rational interpolation and argue that (7) is backward stable if the relevant Lebesgue constant $\Lambda_{x^-,x^+,\mathbf{x}}$, corresponding to the interpolation vector $\mathbf{x}$, is small.

We make this explicit by recalling a slightly simplified version of the main result from [29]. It gives an upper bound on the backward error for the second barycentric formula and allows us to take a closer look at the effect of using formula (10) inside the Parks-McClellan algorithm.

The following analysis takes into account the relative errors in the length of the intervals $[x_i, x_j]$, which are computed as

$$\eta_{kk} := \eta_{kk}(\mathbf{x}, \hat{\mathbf{x}}) := 0 \qquad \text{and} \qquad \eta_{ij} := \eta_{ij}(\mathbf{x}, \hat{\mathbf{x}}) := \frac{x_i - x_j}{\hat{x}_i - \hat{x}_j} - 1, i \neq j,$$

where the hatted variables represent (possibly perturbed) floating-point representations of the values without a hat. We have similar quantities with respect to the interval endpoints

$$\eta_k^- := \eta_k^-(x^-, \mathbf{x}, \hat{x}^-, \hat{\mathbf{x}}) := \frac{x^- - x_k}{\hat{x}^- - \hat{x}_k} - 1,$$

$$\eta_k^+ := \eta_k^+(x^+, \mathbf{x}, \hat{x}^+, \hat{\mathbf{x}}) := \frac{x^+ - x_k}{\hat{x}^+ - \hat{x}_k} - 1,$$

with $\eta_k^- = 0$ if $\hat{x}_k = \hat{x}^-$, and $\eta_k^+ = 0$ when $\hat{x}_k = \hat{x}^+$. Globally, these errors are quantized by

$$\eta := \max_{0 \leqslant i,j \leqslant n} \left\{ \left|\eta_i^-\right|, \left|\eta_{ij}\right|, \left|\eta_i^+\right| \right\} \tag{12}$$

To measure the relative errors with respect to the computed weights $\hat{\mathbf{w}}$ they use

$$\zeta_k := \zeta_k(\mathbf{w}, \hat{\mathbf{w}}) := \frac{w_k - \hat{w}_k}{\hat{w}_k}, \tag{13}$$

under the sensible assumption that w_k and $\hat{w}_k$ are different from 0.

Theorem 5.1. *In the context of formula (7), with the elements of* $\mathbf{x}$ *given in increasing order, we let* ϵ *be the machine precision used in our computations and take* n *such that* $(2n+7)\epsilon < 1$. *We define*

$$Z = \frac{\|\boldsymbol{\zeta}(\mathbf{w}, \hat{\mathbf{w}})\|_\infty + (n+3)\epsilon}{1 - (n+3)\epsilon}. \tag{14}$$

If, for η *in (12)*

$$(\eta + Z)\Lambda_{x^-,x^+,\mathbf{x}} + Z < 1, \tag{15}$$

and $\hat{x} \in [\hat{x}^-, \hat{x}^+]$ *is a floating-point number, then the computed value* $\mathrm{fl}(p(\hat{x}; \hat{\mathbf{x}}, \mathbf{y}, \hat{\mathbf{w}}))$ *is equal to* $p(x; \mathbf{x}, \tilde{\mathbf{y}}, \mathbf{w})$, *for some* $x \in [x^-, x^+]$ *such that*

$$|x - \hat{x}| \leqslant \max\left\{ \|\mathbf{x} - \hat{\mathbf{x}}\|_\infty, \left|\hat{x}^- - x^-\right|, \left|\hat{x}^+ - x^+\right| \right\}, \tag{16}$$

$$\tilde{y}_k = y_k(1 + \alpha_k)(1 + \nu_k) \qquad \text{where} \qquad \|\boldsymbol{\nu}\|_\infty \leqslant \frac{(2n+7)\epsilon}{1 - (2n+7)\epsilon}, \tag{17}$$

and

$$\|\boldsymbol{\alpha}\|_\infty \leqslant \frac{(1 + \Lambda_{x^-,x^+,\mathbf{x}})(\eta + Z)}{1 - Z - (\eta + Z)\Lambda_{x^-,x^+,\mathbf{x}}}. \tag{18}$$

Proof. See [29]. □

5.3.2. *The evolution of* $\Lambda_{x^-,x^+,\mathbf{x}}$ *inside the Parks-McClellan algorithm.* At each iteration of the exchange algorithm a new reference vector $\mathbf{x}$ is computed. The elements of $\mathbf{x}$ are given as machine representable numbers, so we can suppose $\mathbf{x} = \hat{\mathbf{x}}$. Similarly, we take the approximation interval such that $[x^-, x^+] = [\hat{x}^-, \hat{x}^+] = [-1, 1]$. This gives $\eta = 0$ in (12) and eliminates one of the error sources. Also, 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 decrease in value, with the final $\Lambda_{x^-,x^+,\mathbf{x}}$ being not too large.

We will look at some examples shortly, but for now let us assume that the Lebesgue constants we encounter in practice are such that $\Lambda_{-1,1,\mathbf{x}} \leqslant 10^7$, with $8 < n < 1000000$. Taking into account the numerical formats we use in our code, where $\epsilon \leqslant 2.3 \times 10^{-16}$, Theorem 5.1 gives us:

TABLE 4. Lowpass filter design with unit weight passband $[0, 0.49\pi]$ and weight 10 stopband $[0.5\pi, \pi]$. The table shows how the value of the Lebesgue constant evolves during the execution of the Parks-McClellan algorithm for degree 500 and 1000 FIR filters. δ corresponds to the reference error obtained at the last iteration.

$n = 500$ $\delta = 0.164524 \cdot 10^{-3}$		$n = 1000$ $\delta = 0.466115 \cdot 10^{-7}$	
Iteration	$\Lambda_{-1,1,\mathbf{x}}$	Iteration	$\Lambda_{-1,1,\mathbf{x}}$
1	$1.03385 \cdot 10^{9}$	1	$8.35034 \cdot 10^{19}$
10	$2.91177 \cdot 10^{6}$	6	$8.01172 \cdot 10^{6}$
15	2683.07	12	$6.91968 \cdot 10^{6}$

Corollary 5.2. *Under the assumptions of the previous two paragraphs and those of Theorem 5.1, if $\hat{x} \in [-1, 1]$ is a floating point number, then there exists $\boldsymbol{\beta} \in \mathbb{R}^{n+2}$ which verifies*

$$\|\boldsymbol{\beta}\|_{\infty} < 32 \cdot 10^{6} \epsilon n \tag{19}$$

and the vector $\tilde{\mathbf{c}} \in \mathbb{R}^{n+2}$ whose elements are $\tilde{c}_k = (1 + \beta_k)c_k$, such that

$$\mathrm{fl}(H_d(\hat{x}; \mathbf{x}, \mathbf{c}, \hat{\mathbf{w}})) = H_d(\hat{x}; \mathbf{x}, \tilde{\mathbf{c}}, \mathbf{w}). \tag{20}$$

Proof. From the proof of Corollary 2.3 in [29] we know that in (13)

$$|\zeta_k| \leqslant \frac{2\epsilon(n+1)}{1 - 2\epsilon(n+1)} \leqslant \frac{2\epsilon(n+1)}{1 - 2 \times 10^{6} \times 2.3 \times 10^{-16}} \leqslant 2.00001\epsilon(n+1).$$

Using this inequality inside (14), we have

$$Z \leqslant \frac{2.00001\epsilon(n+1) + (n+3)\epsilon}{1 - (10^{6} + 2) \times 2.3 \times 10^{-16}} < 3.0001\epsilon(n+1) < 10^{-9}.$$

We can also show that $\Lambda_{-1,1,\mathbf{x}} > 2.1$ (see Theorem 15.2 from [50] for $n + 1 = 10$). This allows us to bound $\|\boldsymbol{\alpha}\|_{\infty}$ in (18)

$$\|\boldsymbol{\alpha}\|_{\infty} \leqslant \frac{(1 + \Lambda_{-1,1,\mathbf{x}})Z}{1 - (1 + \Lambda_{-1,1,\mathbf{x}})Z} < \frac{(1 + 10^{7}) \cdot 3.0001\epsilon(n+1)}{1 - 3.1 \cdot Z}$$

$$< \frac{10^{7} \cdot 3.001\epsilon(n+1)}{1 - 3.1 \cdot 3.0001\epsilon(n+1)} < \frac{10^{7} \cdot 3.001\epsilon(n+1)}{1 - 10\epsilon(n+1)} < 31 \cdot 10^{6}\epsilon(n+1).$$

For (17), we get

$$\|\boldsymbol{\nu}\|_{\infty} \leqslant \frac{(2n+7)\epsilon}{1 - (2n+7)\epsilon} \leqslant \frac{(2n+7)\epsilon}{1 - (2 \times 10^{6} + 7) \times 2.3 \times 10^{-10}}$$

$$< 1.0005(2n+7)\epsilon < 4.6025 \times 10^{-10},$$

which means that $\beta_k := \nu_k + (1 + \nu_k)\alpha_k$ satisfies

$$|\beta_k| < 1.0005(2n+7)\epsilon + (1 + 4.6025 \times 10^{-10}) \times 31 \cdot 10^{6}\epsilon(n+1) < 32 \cdot 10^{6}\epsilon n.$$

□

5.3.3. *Comments and examples.* The previous result tells us that, under reasonable assumptions, we can expect formula (10) to behave in a numerically stable way. In general, the degrees used in practice do not grow past the $n = 100000$ mark, so the bound on $\Lambda_{-1,1,\mathbf{x}}$ can be, in most cases, relaxed. This is helpful for problems where uniform initialization is used and the starting Lebesgue constants are larger.

Typical examples of how $\Lambda_{-1,1,\mathbf{x}}$ evolves during the execution of the exchange algorithm, when uniform initialization is used, are given in Tables 4, 5 and 6. The Lebesgue constant estimates were numerically computed using the Chebfun routine `lebesgue` mentioned in [50, Ch. 15].

Because $\Lambda_{-1,1,\mathbf{x}}$ usually decreases, it means that, based on our previous statements, computations will tend to be more stable during latter iterations.

TABLE 5. Lebesgue constant evolution when using the Parks-McClellan algorithm to design a bandpass filter with unit weight passband $[0.25\pi, 0.6\pi]$, stopbands $[0, 0.15\pi]$, $[0.7\pi, \pi]$ with weights 10 and 5. The degrees considered range from 60 to 100.

$n = 60$ $\delta = 0.33217 \cdot 10^{-6}$		$n = 70$ $\delta = 0.614085 \cdot 10^{-7}$		$n = 80$ $\delta = 0.876614 \cdot 10^{-8}$		$n = 100$ $\delta = 0.430204 \cdot 10^{-9}$	
Iteration	$\Lambda_{-1,1,\mathbf{x}}$	Iteration	$\Lambda_{-1,1,\mathbf{x}}$	Iteration	$\Lambda_{-1,1,\mathbf{x}}$	Iteration	$\Lambda_{-1,1,\mathbf{x}}$
1	93904.4	1	$3.17826 \cdot 10^7$	1	$2.84902 \cdot 10^9$	1	$3.28558 \cdot 10^{10}$
3	10743.5	5	$1.55826 \cdot 10^5$	13	$5.01386 \cdot 10^5$	4	$8.71543 \cdot 10^6$
6	10280.6	10	$6.22509 \cdot 10^4$	25	$1.36643 \cdot 10^6$	8	$6.47116 \cdot 10^6$

TABLE 6. The evolution of the Lebesgue constant during the execution the Parks-McClellan algorithm for a multiband specification with unit weight passbands $[0.2\pi, 0.4\pi]$, $[0.57\pi, 0.65\pi]$, $[0.77\pi, 0.85\pi]$, weight 10 stopbands $[0, 0.18\pi]$, $[0.42\pi, 0.55\pi]$, $[0.67\pi, 0.75\pi]$, $[0.87\pi, \pi]$ and degrees ranging from 50 to 300.

$n = 50$ $\delta = 0.216717$		$n = 100$ $\delta = 0.36463 \cdot 10^{-1}$		$n = 200$ $\delta = 0.102731 \cdot 10^{-2}$		$n = 300$ $\delta = 0.408778 \cdot 10^{-4}$	
Iteration	$\Lambda_{-1,1,\mathbf{x}}$	Iteration	$\Lambda_{-1,1,\mathbf{x}}$	Iteration	$\Lambda_{-1,1,\mathbf{x}}$	Iteration	$\Lambda_{-1,1,\mathbf{x}}$
1	$5.7809 \cdot 10^7$	1	$1.0949 \cdot 10^8$	1	$9.8061 \cdot 10^{11}$	1	$2.4826 \cdot 10^{17}$
8	808.204	13	$2.1083 \cdot 10^4$	10	$8.8301 \cdot 10^5$	15	$6.8935 \cdot 10^7$
17	596.372	26	60.006	21	3349.64	30	16103.6

TABLE 7. The Lebesgue constants for the starting iterations (with uniform initialization and reference scaling) on the two problematic examples from Tables 4 and 6. The large overall decrease in value is clearly visible.

	$n = 1000$ $\delta = 0.466115 \cdot 10^{-7}$	$n = 300$ $\delta = 0.408778 \cdot 10^{-4}$
Initialization strategy	$\Lambda_{-1,1,\mathbf{x}}$	$\Lambda_{-1,1,\mathbf{x}}$
Uniform	$8.35034 \cdot 10^{19}$	$2.4826 \cdot 10^{17}$
Reference scaling	$1.20561 \cdot 10^7$	$8.0966 \cdot 10^{10}$

Remark 5.3. Two peculiar cases which warrant further discussion are the degree 1000 filter from Table 4 and the degree 300 one from Table 6. The starting Lebesgue constants are rather large for those examples, and based on our discussion up to now, one expects numerical problems if IEEE-754 double-precision arithmetic is used. Surprisingly, our `double` routine with uniform initialization converged for both filters, while MATLAB's `firpm` routine worked only for the second one. Unfortunately, the convergence of uniform initialization in such cases where $\Lambda_{-1,1,\mathbf{x}}$ is very large is not to be expected in general, since numerical problems are plenty. In the provided test files, there are a lot of examples where we do not have convergence with uniform initialization. We verified the presence of numerical instability in such cases by comparing the execution of our `double` code with that of our `MPFR` version. The *exact same* algorithm, executed with 53 bits of precision for `double` evaluations and with 165 bits for our multiple precision routine, behaved in *radically different* ways. For example, in the case of the $n = 1000$ filter, the δ corresponding to the first iteration differed between the two routines by a factor of 10^9! By contrast, when we used our reference scaling approach from Section 4.1 for the two codes (which, as can be seen from Table 7, drastically decreased the value of the starting $\Lambda_{-1,1,\mathbf{x}}$), their execution traces were *almost identical*. This translates to a much stable behavior for the `double` version when reference scaling is used.

Similar observations regarding the decrease in value of $\Lambda_{-1,1,\mathbf{x}}$ can be made for Examples 1.1 to 1.4. We also note that, by knowing the value of n and initial $\Lambda_{-1,1,\mathbf{x}}$ for a particular problem, we can use Theorem 5.1 to derive a suitable working precision ϵ which ensures an accurate final result.

6. Extrema search

Determining the reference set $\boldsymbol{\omega'} \in \mathbb{R}^{n+2}$ (or equivalently $\mathbf{x}' = (\cos(\omega'_k))_{0 \leqslant k \leqslant n+1}$), which takes its values from the local extrema of the error function $E(\omega)$, represents the next step in the Parks-McClellan exchange algorithm.

6.1. A Chebyshev-proxy root-finding method.

We begin our discussion by considering the slightly different problem of determining the zeros of a function $f \in \mathcal{C}([x^-, x^+])$, located inside $[x^-, x^+]$. For simplicity, we will again

take $[x^-, x^+] = [-1, 1]$, and note that by suitable changes of variable, the following results hold up for any closed interval. The idea of the Chebyshev-proxy root-finder (CPR) method [9–11, 50] 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 the zeros of p_m will very closely match those of f. Just like barycentric Lagrange interpolation, this global root-finding approach has long been on display as an integral part of Chebfun (the `roots` command) and its implementation of the Remez algorithm [34], of which it is a central ingredient. Here, we present our own variation, which we consider to be very well suited for designing FIR filters.

The chosen proxy, p_m, 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. This gives us

$$p_m(x) = \sum_{k=0}^{m} a_k T_k(x), \qquad x \in [-1, 1], a_k \in \mathbb{R}, k = 0, \ldots, m, \tag{21}$$

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

$$a_k = \frac{2}{m} \sum_{i=0}^{m}{}'' f(\nu_i) T_k(\nu_i), \qquad k = 0, \ldots, m, \tag{22}$$

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

Computing the a_i coefficients in the previous formula can be done in a numerically stable way with $O(m^2)$ operations, by using Clenshaw's recurrence relations [39, Ch. 5.4], or faster, in only $O(m \log_2 m)$ operations, by means of the Discrete Cosine Transform [49]. Because it is simpler to implement and it does not affect the overall complexity of our routine, we use Clenshaw's algorithm.

The roots of p_m are usually computed as the eigenvalues of a generalized companion matrix [10, 18], also known as a *colleague matrix* (see [50, Ch. 18] for further information and details). Such an approach is known to behave very well in practice [12].

With these remarks, let us go back to our original problem of determining the local extrema of the error function $E(\omega)$ and see how we would apply the CPR method in this scenario. Suppose that we have already computed a degree m proxy

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

which accurately approximates $E(\omega)$. The value of m should be thought of independently of the degree n for the target minimax response $H_d(\omega)$. Finding good approximations of the local extrema of $E_m(\omega)$ would then amount to looking for the roots of $e_m(\omega) = \dfrac{\mathrm{d}E_m(\omega)}{\mathrm{d}x}$. To translate this to an eigenvalue problem, we need to compute the Chebyshev expansion of $e_m(\omega)$. If we consider it to be

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

we get the recurrence: $b_{k-1} = 2ka_k + b_{k+1}, k = 1, \ldots, m$, where $b_m = b_{m+1} = 0$. 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 Boyd introduces the CPR method in the context of T_m, we found the second formula for e_m more natural to compute in our setting.

Finding the eigenvalues of a $m \times m$ colleague matrix, for both Chebyshev polynomials of the first and second kind, can be done by applying a QR/QZ algorithm with an $O(m^3)$ operation count. This cost can be taken down to $O(m^2)$ by adaptively dividing the initial interval $[x^-, x^+]$ 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. A general way of performing the subdivision is the following: if we set $N_{\max} < m$ to be the maximum degree on which we are willing to use an eigenvalue algorithm, we recursively split $[x^-, x^+]$ until we have Chebyshev polynomial interpolants of degree at most $N_{\max}$ on each subinterval, which approximate f accurately.

There are several general criteria that try to quantify when we have a good enough approximation of f (see for example [10, 11]). These adaptive approaches amount to doing extra computations each time a subinterval is split, since we will not use the interpolant we computed before the subdivision.

Our subdivision strategy is different and consists of noticing that, in general, during each iteration of the Parks-McClellan algorithm, we have an idea about the location and number of extrema of E for certain subintervals. What

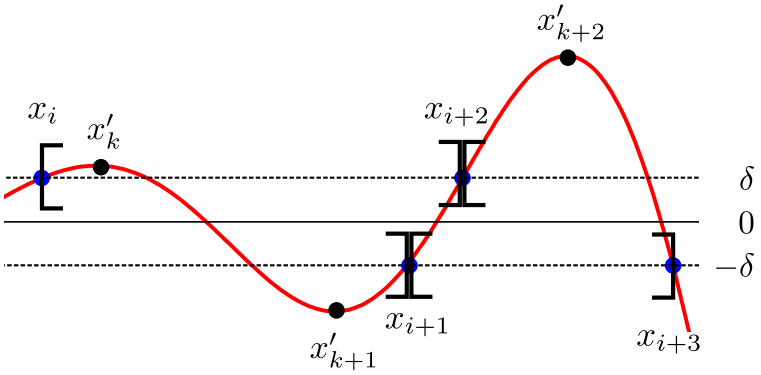

FIGURE 3. Interval subdivision strategy for computing the extrema of $E(\omega)$

we mean by this is that, if we take the subinterval defined by two consecutive points of the current reference vector $\mathbf{x}$ of X located inside the same band, say $[x_i, x_{i+1}]$, then we can *usually* expect to have between zero and two potential extrema of E inside it. Figure 3 summarizes this scenario. We can then use a degree $N_{\max}$ Chebyshev proxy on each such subinterval. Values of $N_{\max}$ we frequently used in our experiments are 4 and 8. This allows us to avoid subdivision tests and consequently require less computation time.

If we partition X into subintervals of the form $[x_i, x_{i+i}]$, we can expect to have in total $O(n)$ such intervals. To cover all of X, we also need to consider 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 we use 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. This results in a total operation count of $O(n^2)$.

We think that such a CPR-based variation is very robust for determining the extrema of E in practice. Our reasoning behind this claim is based on three important remarks:

(1) Chebyshev interpolation on each subinterval is very *accurate* in practice;
(2) E is evaluated a *small* and *constant* number of times on each subinterval;
(3) The computations on each subinterval are *data parallel*.

The consequence is that, in addition to requiring the evaluation of E a reasonable number of times, our approach can also be very easily parallelized.

Because the amount of work on each subinterval is the same, the scheduling cost for parallel execution should be minimal. If we also take into account the fact that the extrema search corresponds to the majority of the execution time of the Park-McClellan algorithm, impressive speed-ups can be achieved on multicore systems for large degree filters. We give some examples in the next section.

6.2. **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 their error 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 [33, 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. Other discussions on how to update the reference set can be found in [6, Ch. 15.3.4] and [34, Sec. 2.2].

6.3. **Computing the filter coefficients.** All of the techniques we introduced during the last three sections amount to each iteration (Step 2 to Step 5) of the exchange algorithm being done using a total of $O(n^2)$ arithmetic operations. Upon convergence, we also have to retrieve the coefficients $(h_k)_{0 \leqslant k \leqslant n}$ from (4) of the final frequency response $H_d(\omega)$. As we already argued, we can do this in a numerically stable way with a quadratic number of computations by using formula (22) and Clenshaw's algorithm.

Figure 4 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 2.2 clearly visible.

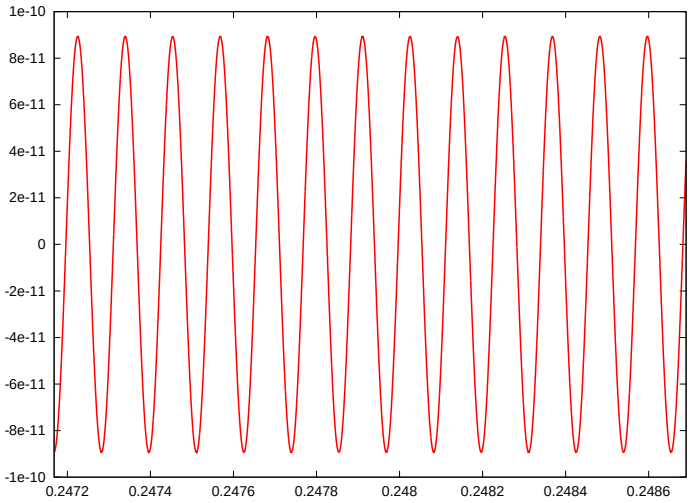

FIGURE 4. The value of the approximation error E on a small interval $[0.247168, 0.24869]$ of X, for the minimax filter satisfying the specification for Example 1.4.

7. Implementation details

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

7.1. **User interface.** Our code requires a small number of external libraries in order to work. Besides the use of `MPFR` when we do multiple precision arithmetic, we also use the `Eigen` library [24] to perform all the eigenvalue computations related to the CPR method. 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 6.1, we use two `#parallel for` OpenMP [17] pragmas.

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 CPR method in Section 6.1, 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 $H(z) = \sum_{k=0}^{N} b_k z^{-k}$ from equation (2), 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 is denoted with `res.delta`.

Type III and IV differentiators and Hilbert transformers [33, Ch. 12.4] are computed in a similar way, the only difference being that the `firpm` function in this case has an extra parameter of type `ftype`, which can take the values `FIR_DIFFERENTIATOR` and `FIR_HILBERT`.

TABLE 8. Runtime comparisons of our IEEE-754 double-precision implementation of the Parks-McClellan algorithm with those available in GNURadio 3.7.5.1 (also written in C++), MATLAB R2014b, the one from SciPy 0.15.1 (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 3.14.29 (Manjaro 0.8.11), 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 ($n = 50$)	0.0103	**0.0029**	0.0532	0.0711	0.0384/0.0098
1.1 ($n = 80$)	**0.0071**	0.0073	0.1174	0.2587	0.0416/0.0316
1.1 ($n = 100$)	**0.0112**	NC	0.1491	0.3511	0.0451/0.0396
1.2 ($n = 50$)	0.0059	**0.0031**	0.0681	0.0971	0.0395/0.0174
1.2 ($n = 80$)	**0.0079**	NC	0.2002	0.3492	0.0761/0.0293
1.2 ($n = 100$)	**0.0395**	NC	NC	NC	0.2599/0.0521
1.3 ($n = 520$)	**0.3519**	NC	NC	NC	1.1691/2.7011
1.4 ($n = 53248$)	NC	NC	NC	NC	NC/NC

TABLE 9. Timings showing the effect of running our code with different options. As for Table 8, 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++4.9.2` with `-O3 -DNDEBUG` level optimizations.

Example (degree)	Uniform (sequential)	Uniform (parallel)	Scaling (sequential)	Scaling (parallel)
1.1 ($n = 50$)	0.0103	0.0051	0.0029	0.0018
1.1 ($n = 80$)	0.0071	0.0047	0.0042	0.0027
1.1 ($n = 100$)	0.0112	0.0073	0.0147	0.011
1.2 ($n = 50$)	0.0059	0.0045	0.0075	0.0046
1.2 ($n = 80$)	0.0079	0.0062	0.0048	0.0031
1.2 ($n = 100$)	0.0395	0.0274	0.0339	0.0275
1.3 ($n = 520$)	0.3519	0.2251	0.0982	0.0716
1.4 ($n = 53248$)	NC	NC	537.8	162.6

The `firpm` functions use uniform initialization by default. To use reference scaling, we supply the function `firpmRS`. For more general designs, we also make available the function `exchange`, which, among its parameters, takes a vector `x`, corresponding to the initial reference from X, used inside the exchange algorithm.

More details are provided in the documentation of the libraries, while examples of how to use the `firpmRS` and `exchange` methods can be found in the `test` directory of each version.

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 8. 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 [32], 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 *reference scaling* and *parallelization* of the extrema search are showcased in Table 9. 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.

8. Conclusions

In this article we have presented several ideas that aid in the development of optimal linear-phase FIR filters. These contributions amount to the following:

- introduce a new initialization technique for high degree FIR filters (Section 4.1), which we found to greatly improve the numerical behavior of the exchange method in the case of high degree designs;
- offer a pertinent analysis about when and why the exchange algorithm for FIR filter design behaves well in practice (Sections 5.2 and 5.3);
- present a variation of a well-established root-finding approach [11, 50] which allows one to design FIR filters in a very efficient way (Section 6.1).

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.

The problem of finding a suitable initial reference can also be potentially addressed as suggested in [34, Sec. 3.6] with the help of Carathéodory-Fejer (CF) near-best approximations (see [50, Ch. 20] for an introduction and further references). Some remarks on using CF approximations for designing FIR filters are made by Trefethen on the Chebfun website[3]. Since CF approximations are constructed on closed intervals, the biggest issue with such an approach would be how to introduce suitable constraints on the transition bands. A good starting point could be [16]. How weights are incorporated to this approach can also prove to be an interesting point of investigation.

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 extended to approaches that address transition band anomalies (see Section 3.3) or be applied to other digital signal processing tasks, like those requiring FIR filters with complex coefficients [26, 27] or IIR filters [44–47].

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] Boyd, J. P. Computing Zeros on a Real Interval through Chebyshev Expansion and Polynomial Rootfinding. *SIAM Journal on Numerical Analysis 40*, 5 (2002), 1666–1682.

[10] 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.

[11] 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.

[12] 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.

[13] Burrus, C. S. Multiband least squares FIR filter design. *IEEE Transactions on Signal Processing 43*, 2 (Feb 1995), 412–421.

[14] Burrus, C. S., Soewito, A. W., and Gopinath, R. A. Least squared error FIR filter design with transition bands. *IEEE Transactions on Signal Processing 40*, 6 (Jun 1992), 1327–1340.

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

[16] Cortelazzo, G., Lightner, M. R., and Jenkins, W. K. An alternate technique for min-max design of multiband finite impulse response digital filters. *Circuits, Systems and Signal Processing 2*, 3 (1983), 285–309.

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

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

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

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

[20] 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.
[21] Floater, M. S., and Hormann, K. Barycentric rational interpolation with no poles and high rates of approximation. *Numerische Mathematik 107*, 2 (2007), 315–331.
[22] 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).
[23] Grenez, F. Design of linear or minimum-phase FIR filters by constrained Chebyshev approximation. *Signal Processing 5*, 4 (1983), 325–332.
[24] Guennebaud, G., Jacob, B., et al. Eigen v3. http://eigen.tuxfamily.org, 2010.
[25] Higham, N. J. The numerical stability of barycentric Lagrange interpolation. *IMA J. Numer. Anal 24* (2004), 547–556.
[26] 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.
[27] 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.
[28] Mascarenhas, W. F. The stability of barycentric interpolation at the Chebyshev points of the second kind. *Numerische Mathematik 128*, 2 (2014), 265–300.
[29] 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.
[30] McCallig, M., and Leon, B. Constrained ripple design of FIR digital filters. *IEEE Transactions on Circuits and Systems 25*, 11 (Nov 1978), 893–902.
[31] 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.
[32] 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.
[33] Oppenheim, A. V., and Schafer, R. W. *Discrete-Time Signal Processing*. Prentice-Hall Signal Processing Series. Prentice Hall, 2010.
[34] 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.
[35] 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.
[36] Pelloni, D., and Bonzanigo, F. On the design of high-order linear phase FIR filters. *Digital Signal Processing 1* (1980), 3–10.
[37] Powell, M. J. D. *Approximation Theory and Methods*. Cambridge University Press, 1981.
[38] Prandoni, P., and Vetterli, M. *Signal Processing for Communications*. Taylor & Francis, 2008.
[39] 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.
[40] 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.
[41] 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.
[42] Remes, E. Sur le calcul effectif des polynomes d'approximation de Tchebichef. *Comptes rendus hebdomadaires des séances de l'Académie des Sciences*, 199 (1934), 337–340.
[43] Rutishauser, H. *Vorlesungen über Numerische Mathematik*. Birkhäuser Basel, Stuttgard, 1976. English translation, 1990. *Lectures on Numerical Mathematics*. Walter Gautschi, ed., Birkhäuser Boston.
[44] 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.
[45] 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.
[46] 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.
[47] 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.
[48] 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.
[49] Strang, G. The discrete cosine transform. *SIAM Review 41*, 1 (1999), 135–147.
[50] Trefethen, L. N. *Approximation Theory and Approximation Practice*. Society for Industrial and Applied Mathematics, 2013.
[51] Veidinger, L. On the numerical determination of the best approximation in the Chebyshev sense. *Numerische Mathematik 2*, 1 (1960), 99–105.
[52] Webb, M., Trefethen, L. N., and Gonnet, P. Stability of Barycentric Interpolation Formulas for Extrapolation. *SIAM J. Scientific Computing 34*, 6 (2012).
[53] 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.