Two Novel Implementations of the Remez Multiple Exchange Algorithm for Optimum FIR Filter Design

Muhammad Ahsan and Tapio Saramäki

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/46477

1. Introduction

One of the main advantages of the linear-phase FIR filters over their IIR counterparts is the fact that there exist efficient algorithms for optimizing the arbitrary-magnitude FIR filters in the minimax sense. In case of IIR filters, the design of arbitrary-magnitude filters is usually time-consuming and the convergence to the optimum solution is not always guaranteed. The most efficient method for designing optimum magnitude linear-phase FIR filters with arbitrary-magnitude specifications is the Remez algorithm and the most frequent method to implement this algorithm is the one originally proposed by Parks and McClellan. Initially, they came up with the design of conventional low-pass linear phase FIR filters, whose impulse response is symmetric and the order is even [5]. Later on, along with Rabiner they extended the original design technique in [7] such that the generalized algorithm is applicable to the design of all the four linear-phase FIR filter types with arbitrary specifications. Due to the initial work of Parks and McClellan for the extended algorithm, this algorithm is famously known as the Parks-McClellan (PM) algorithm.

The PM algorithm was generated in the beginning of 1970 by using FORTRAN. During that era, the computer resources were quite limited. When people applied this algorithm in practice for high-order filters, they failed to achieve the optimum results. This gave rise to two main doubts about the PM algorithm. The first doubt was that the algorithm is not properly constructed and is valid for only low-order filters design. However, after noticing that the maximum number of iterations in the algorithm implementation is set to only 25 which is quite low to achieve the optimum solutions for high order filters, it became clear that the first doubt is superfluous.

The second doubt was concerned with the implementation of the PM algorithm's search strategy for the "real" extremal points of the weighted error function, which is formed based on the "trial" extremal points. While mimicking the search technique included in the PM algorithm in various Remez-type algorithms for designing recursive digital filters [15, 16, 18, 19], it was observed that some of the them are quite sensitive to the selection of the initial set of the "trial" extremal points. Moreover, they suffered from certain issues which prevented them to converge at the optimum solution in some cases. For algorithms described in [15], and [18], the convergence issue was solved by increasing the maximum number of iterations in the above-mentioned manner. However, the algorithms described in [16] and [19] have still remained sensitive to the selection of the initial set of the "trial" extremal points. This sensitivity motivated the authors of this contribution to figure out how the search for the "true" extremal points is really carried out in the core discrete Remez algorithm part in the FORTRAN implementation of the PM algorithm. After a thorough study of the FORTRAN code, it was observed that this code utilizes almost twenty interlaced "go to" statements which are quite redundant for locating the "real" extremal points. Meticulous investigation of the code revealed that it is possible to decompose the one large chunk of the search technique into two compact search techniques referred to as *Vicinity Search* and *Endpoint Search* in such a manner that the same optimum solution can be achieved in an efficient manner as follows.

In *Vicinity Search*, the candidate "real" extremal point is located in the vicinity of each "trial" extremal point, which is bounded by the preceding and the following "trial" extremal points with the exception of the first (last) point for which the lower (upper) bound is the first (last) grid point in use. *Endpoint Search*, in turn, checks whether before the first (after the last) local extremum found by *Vicinity Search* there is an additional first (last) local extremum of opposite sign. If one or both of such extrema exist, then their locations are considered as candidate "real" extremal points of the overall search consisting of *Vicinity Search* and *Endpoint Search*. In this case, there are one or two more candidate "real" extremal points as *Vicinity Search* already provides the desired number of "real" extremal points. In the PM algorithm, the desired final "real" extremal points are determined according to the following three options:

Option 1: The additional last extremum exists such that its absolute value is larger than or equal to those of the first extremum of *Vicinity Search* and the possible additional first extremum.

Option 2: The additional first extremum exists such that its absolute value is larger than or equal to that of the first extremum of the *Vicinity Search* and larger than that of the possible additional last extremum.

Option 3: The conditions in *Options 1* and *2* are not valid.

For *Option 3*, the final "real" extremal points are the ones obtained directly from the *Vicinity Search*, whereas for *Option 1* (*Option 2*), these points are obtained by omitting the first (last) point based on *Option 3* and by replacing the last (first) point with the one found by *Endpoint Search*. Based on the above-mentioned facts, an extremely compact translation of the original FORTRAN implementation into a corresponding MATLAB implementation has been reporeted in [2].

The above study on how the PM algorithm performs the search for the "real" extremal points indicates that mimicking this search principle in the Remez-type algorithms proposed in [16], [19] does not give the flexibility to transfer two extremal points between the two consecutive bands, for example, from a passband to a stopband or vice versa, which is a necessary prerequisite for convergence to the optimum solution in certain cases. Most significantly, the search technique included in the PM algorithm does not follow the fundamental idea of the Remez multiple exchange (RME) algorithm when the approximation interval is a union of three or more disjoint intervals [8, 11, 20]. That is, if there are more candidate "real" extremal points than required, then the desired points should be selected in such a way that the ones corresponding to the largest absolute values of the weighted error functions are retained subject to the condition that the sign of the weighted error function alternates at the consecutive points. An efficient MATLAB based implementation following the above mentioned fundamental notion of the RME algorithm has been reported in [3] and provides significant improvements in designing the multiband FIR filters.

In the beginning, the main purpose of the authors of this contribution was to modify the core discrete Remez part of the PM algorithm in FORTRAN such that it follows the fundamental principle of the RME algorithm and can ultimately be incorporated in the algorithms proposed in [16] and [19]. However, during the course of modifications, it was observed that a modified MATLAB implementation mimicking the original FORTRAN implementation is quite effective and superior to already available MATLAB implementation of the algorithm in the function **`firpm`** [21]. Based on the above discussion, this chapter describes two novel MATLAB based implementations of the Remez algorithm within the PM algorithm. Implementation I is an extremely fast and compact translation of the Remez algorithm part of the original FORTRAN code to the corresponding MATLAB code and is valid for general purpose linear-phase FIR filters design [2]. It is worth noting that Implementation I imitates the implementation idea of the Remez algorithm presented in PM algorithm. Implementation II is based on the fundamental notion of the Remez algorithm as described in [20] and provides significant improvements in designing the multiband FIR filters [3]. It is important to note that this chapter emphasizes on the practical MATLAB based implementation aspects of the Remez algorithm. In order to get an overview of the theoretical aspects of the Remez algorithm, please refer to [10, 17]. The organization of this chapter is as follows. Section (2) formally states the problem under consideration, Section (3) describes the Implementation I in detail, Section (4) discusses the Implementation II in detail, and finally, the concluding remarks are presented in Section (5).

2. Problem statement

After specifying the filter type, the filter order, and the filter specifications such that the problem is solvable using the RME algorithm, the essential problem in the PM algorithm is the following:

***Core Discrete Approximation Problem*:** Given $nz - 1$ [1], the number of unknowns $a[n]$ for $n = 0, 1, \ldots, nz-2$, and the grid points $grid(k)$ included in the vector **grid** of length $ngrid$, which contains values between 0 and 0.5 [2], along with the vectors **des** and **wt** of the same length $ngrid$, the entries of which carry the information of the desired and weight values, respectively, at the corresponding grid points of the vector **grid**, find the unknowns $a[n]$ to minimize the following quantity:

$$\varepsilon = \max_{1 \le k \le ngrid} |E(k)|, \tag{1a}$$

[1] In this contribution, $nz - 1$ is chosen to be the number of adjustable parameters in both the FORTRAN and the MATLAB implementations of the PM algorithm because in this case nz stands for the number of extrema at which *Alternation Theorem* should be satisfied in order to guarantee the optimality of the solution.

[2] In the original PM algorithm, this range is the baseband for the so-called normalized frequencies from which the corresponding angular frequencies are obtained by multiplying these frequencies by 2π [17].

where

$$E(k) = \mathbf{wt}(k)\,[G(k) - \mathbf{des}(k)] \tag{1b}$$

and

$$G(k) = \sum_{n=0}^{nz-2} a[n]\cos[2\pi n \cdot grid(k)]. \tag{1c}$$

According to *Alternation (characterization) Theorem* [4, 12, 13], $G(k)$ of the form of (1c) is the best unique solution minimizing ϵ as given by (1a) if and only if there exists a vector $\boldsymbol{\ell}_{\mathbf{opt}}$ that contains (at least) nz entries $\boldsymbol{\ell}_{\mathbf{opt}}(1), \boldsymbol{\ell}_{\mathbf{opt}}(2), \ldots, \boldsymbol{\ell}_{\mathbf{opt}}(nz)$ having the values of k within $1 \leq k \leq ngrid$ such that

$$\begin{aligned}
&\boldsymbol{\ell}_{\mathbf{opt}}(1) < \boldsymbol{\ell}_{\mathbf{opt}}(2) < \ldots < \boldsymbol{\ell}_{\mathbf{opt}}(nz-1) < \boldsymbol{\ell}_{\mathbf{opt}}(nz)\\
&E[\boldsymbol{\ell}_{\mathbf{opt}}(m+1)] = -E[\boldsymbol{\ell}_{\mathbf{opt}}(m)] \text{ for } m = 1, 2, \ldots, nz-1\\
&\left|E[\boldsymbol{\ell}_{\mathbf{opt}}(m)]\right| = \varepsilon \text{ for } m = 1, 2, \ldots, nz.
\end{aligned}$$

It is worth mentioning that the core discrete approximation problem is the same for both the implementations I and II as defined in the Introduction.

3. Implementation I

This section discusses Implementation I in detail as follows. First, the theoretical formulation of the algorithm is described so that the reader can grasp the very essence of the MATLAB code snippet provided later on in this section. Second, during this inclusion, it is emphasized that instead of using approximately 15 nested loops and around 300 lines of code, only 3 looping structures and approximately 100 lines of code are required by Implementation I. Third, it is shown, by means of four examples, that the overall CPU execution time required by the proposed implementation to arrive practically in the same manner at the optimum FIR filter designs is only around one third in comparison with the original implementation. Fourth, in the last two examples, there are unwanted peaks in the transition bands. In order to suppress these peaks to acceptable levels, two methods of including the transition bands in the original problem are introduced.

3.1. Theoretical formulation

The theoretical formulation of the proposed algorithm is roughly classified into the initialization phase and the iteration phase. The initialization phase performs the necessary initializations for the algorithm, whereas the iteration phase carries out the actual Remez exchange loop. In order to explain why Implementation I is a compact and efficient MATLAB based routine, the iteration phase is further decomposed into four well-defined primary segments. Each segment is constructed in such a way that before the start of the basic steps, there is a thorough explanation on the benefits of carrying out the segment under consideration with the aid of the proposed basic steps.

3.1.1. Initialization phase

The overall implementation starts with the following initializations:

- Initialize the element values of "trial" vector $\boldsymbol{\ell}_{\textbf{trial}}$ of length nz for $\boldsymbol{\ell}_{\textbf{opt}}$ as $\boldsymbol{\ell}_{\textbf{trial}}(m) = 1 + (m-1)\lfloor (ngrid-1)/nz \rfloor$ for $m = 1, 2, \ldots, nz-1$ and $\boldsymbol{\ell}_{\textbf{trial}}(nz) = ngrid$. Here, $\lfloor x \rfloor$ stands for the integer part of x.
- Initialize the iteration counter as $niter = 1$ and set the maximum number of iterations as $itrmax = 250$.

3.1.2. Iteration phase

The Remez exchange loop iteratively locates the desired optimum vector $\boldsymbol{\ell}_{\textbf{trial}}$ having as its entries the nz values of k within $1 \le k \le ngrid$, at which *Alternation Theorem* is satisfied as follows. In the first loop, the vector $\boldsymbol{\ell}_{\textbf{trial}}$ found in the initialization phase is the first "trial" vector for being the desired optimum vector $\boldsymbol{\ell}_{\textbf{opt}}$. As this is extremely unlikely to happen, **Segment 1**, based on the vector $\boldsymbol{\ell}_{\textbf{opt}}$, generates the weighted error vector **wei_err**(k) for $1 \le k \le ngrid$ corresponding to $E(k)$, as given by (1b), in such a way that the $nz-1$ unknowns $a[0], a[1], \ldots, a[nz-2]$ as well as dev [3] are implicitly found so that the following nz equations

$$\textbf{wei\_err}(\boldsymbol{\ell}_{\textbf{trial}}(m)) = (-1)^{m+1} dev \text{ for } m = 1, 2, \ldots, nz \tag{2}$$

are satisfied. When concentrating only on the values of k being the enteries of the "trial" vector $\boldsymbol{\ell}_{\textbf{trial}}$, this solution is the best one according to *Alternation Theorem*. However, when considering all the values of k within $1 \le k \le ngrid$, this solution is not the best one.

The efficiency of Implementation I in comparison with the original MATLAB function **`firpm`**, in terms of significant reduction in the code compactness and a considerable reduction in the CPU execution time for obtaining practically in the same manner the best linear-phase FIR solutions are mostly based on the following two novel facts. First, the steps under **Segment 1** are accomplished by employing the efficient MATLAB vectorization operations whenever possible and, most importantly, by avoiding the call for one subroutine by replacing this call with highly efficient matrix operations available in MATLAB. Second, as already mentioned in the introduction, the lengthy search technique involved in the function **`firpm`** for locating the true extremal points based on the weighted error function can be compressed into *Vicinity Search* and *Endpoint Search*. In the sequel, **Segment 2** and **Segment 3** will take care of *Vicinity Search* and *Endpoint Search*, respectively. More detail can be found in the actual implementations of Segments 1, 2, and 3.

Finally, **Segment 4** checks whether $\boldsymbol{\ell}_{\textbf{real}} \equiv \boldsymbol{\ell}_{\textbf{trial}}$ or not. If this equivalence is established, then the best solution has been found as in this case $\boldsymbol{\ell}_{\textbf{opt}} \equiv \boldsymbol{\ell}_{\textbf{real}} \equiv \boldsymbol{\ell}_{\textbf{trial}}$. Otherwise, the whole process is repeated by using the "real" vector $\boldsymbol{\ell}_{\textbf{real}}$ of the present iteration as a "trial" vector for the next iteration. This exchange of the vectors is continued until $\boldsymbol{\ell}_{\textbf{real}}$ and $\boldsymbol{\ell}_{\textbf{trial}}$ coincide or the maximum allowable number of the iterations is exceeded, which is extremely unlikely to occur.

Segment 1: After knowing the "trial" vector $\boldsymbol{\ell}_{\textbf{trial}}$ that contains the nz trial values of k in the ascending order for $1 \le k \le ngrid$ in the present iteration, this first segment guarantees that

[3] It should be noted that the value of dev is either positive or negative.

Alternation Theorem is satisfied when concentrating only on those nz values of k being involved in the vector $\boldsymbol{\ell}_{\mathbf{trial}}$. For this purpose, it generates the weighted error vector $\mathbf{wei\_err}(k)$ for $1 \le k \le ngrid$ such that the following system of nz equations:

$$\mathbf{wt}(\boldsymbol{\ell}_{\mathbf{trial}}(m)) \left[\sum_{m=0}^{nz-2} a(n) \cos\left(2\pi n \cdot grid(\boldsymbol{\ell}_{\mathbf{trial}}(m))\right) - \mathbf{des}(\boldsymbol{\ell}_{\mathbf{trial}}(m)) \right] = (-1)^{m+1} dev \text{ for } m = 1, 2, \ldots, nz \tag{3}$$

is implicitly solved for the $nz - 1$ unknowns $a[0], a[1], \ldots, a[nz-2]$ as well as for dev.[4] For this purpose, similar to the function **`firpm`**, for a given "trial" vector $\boldsymbol{\ell}_{\mathbf{trial}}$, the value of $\mathbf{wei\_err}(k)$ at $k = \boldsymbol{\ell}_{\mathbf{trial}}(1)$, denoted by dev, the corresponding abscissa vector $\mathbf{x}$, the ordinate vector $\mathbf{y}$, and the coefficient vector $\mathbf{ad}$, each of which are of length nz, are determined. These vectors are required to express the zero-phase frequency response when using the Lagrange interpolation formula in the barycentric form at each value of k for $1 \le k \le ngrid$, thereby making the implementation of the Remez loop very accurate and efficient.

In comparison with many scattered scalar operations in the original function **`firpm`**, the MATLAB code snippet, which is available in the following subsection, is computationally efficient and is highly compact due to the above-mentioned vectors. In addition to that, the time consuming subroutine of "**`remezdd`**" is replaced with simple and highly efficient matrix operations. Further improvements are obtained by using the vector **grid**, which contains the grid points under consideration, as well as **des** and **wt**, which carry information of the desired values and weights at these grid points, respectively. With the above-mentioned data, the weighted error function is generated only once during each iteration and is a single vector **wei_err**. This vector plays a pivotal role in the implementations of *Vicinity Search* in **Segment 2** and *Endpoint Search* in **Segment 3**.

This segment is performed by using the following ten steps:

Step 1: Determine the entries of the vectors **x** and **ad** as

$$\mathbf{x}(m) = \cos[2\pi \cdot \boldsymbol{\ell}_{\mathbf{trial}}(m)] \text{ for } m = 1, 2, \ldots, nz \tag{4}$$

and

$$\mathbf{ad}(m) = 1 \Bigg/ \prod_{\substack{k=1 \\ k \ne m}}^{nz} [\mathbf{x}(m) - \mathbf{x}(k)] \text{ for } m = 1, 2, \ldots, nz, \tag{5}$$

respectively, as well as the corresponding deviation value as

$$dev = -\frac{\sum_{m=1}^{nz} \mathbf{ad}(m)\mathbf{des}[\boldsymbol{\ell}_{\mathbf{trial}}(m)]}{\sum_{m=1}^{nz} (-1)^{m-1} \mathbf{ad}(m)\mathbf{wt}[\boldsymbol{\ell}_{\mathbf{trial}}(m)]}. \tag{6}$$

[4] It is worth emphasizing that implicit solutions for the calculation of $a[n]$'s are not required for the intermediate iterations. The explicit solution for the calculation of $a[n]$'s is needed only after achieving the convergence to the best solution.

Step 2: Determine the entries of the vector **y** as

$$\mathbf{y}(m) = \mathbf{des}[\boldsymbol{\ell}_{\mathbf{trial}}(m)] + (-1)^{m-1}\mathbf{ad}[\boldsymbol{\ell}_{\mathbf{trial}}(m)]/\mathbf{wt}[\boldsymbol{\ell}_{\mathbf{trial}}(m)] \;\text{ for }\; m = 1, 2, \ldots, nz. \tag{7}$$

Step 3: Generate the entries of the abscissa vector **x_all** covering all the entries in the vector **grid** as

$$\mathbf{x\_all}(k) = \cos[2\pi \cdot \mathbf{grid}(k)] \;\text{ for }\; k = 1, 2, \ldots, ngrid. \tag{8}$$

Step 4: Select the entries of the vectors **err_num** and **err_den** of length *ngrid* to be zero valued, set $m = 1$, and go to the next step.

Step 5: Generate the entries of the vector **aid** as

$$\mathbf{aid}(k) = \mathbf{ad}(m)/[\mathbf{x\_all}(k) - \mathbf{x}(m)] \;\text{ for }\; k = 1, 2, \ldots, ngrid \tag{9}$$

and update $\mathbf{err\_num}(k)$ and $\mathbf{err\_den}(k)$ for $k = 1, 2, \ldots, nz$ as $\mathbf{err\_num}(k) = \mathbf{err\_num}(k) + \mathbf{y}(m)\mathbf{aid}(k)$ and $\mathbf{err\_den}(k) = \mathbf{err\_den}(k) + \mathbf{aid}(k)$, respectively.

Step 6: Set $m = m + 1$. If $m > nz$, then go to the next step. Otherwise, go to the previous step.

Step 7: Generate the entries of the weighted error function **wei_err** for $k = 1, 2, \ldots, ngrid$ as

$$\mathbf{wei\_err}(k) = [\,\mathbf{err\_num}(k)/\mathbf{err\_den}(k) \;-\mathbf{des}(k)]\mathbf{wt}(k). \tag{10}$$

The resulting **wei_err** contains undefined values at the entries of $\boldsymbol{\ell}_{\mathbf{trial}}$ due to the use of the Lagrange interpolation formula in the barycentric form. The undefined values can be conveniently filled based on the fact that at $\boldsymbol{\ell}_{\mathbf{trial}}(m)$ with m odd (even), the desired value is dev $(-dev)$, where dev is given by (6). Hence, the vector **wei_err** can be completed by using the following three steps:

Step 8: Set $m = 1$ and go to the next step.

Step 9: Update the vector **wei_err** as

$$\mathbf{wei\_err}(\boldsymbol{\ell}_{\mathbf{trial}}(m)) = \begin{cases} +dev & \text{for } m \text{ odd} \\ -dev & \text{for } m \text{ even.} \end{cases} \tag{11}$$

Step 10: Set $m = m + 1$. If $m < nz + 1$, then go to the previous step. Otherwise, go to *Step 1* under **Segment 2**.

Segment 2: This segment explains how to perform *Vicinity Search* based on the values of the weighted error function $\mathbf{wei\_err}(k)$ for $1 \le k \le ngrid$, which has been generated at **Segment 1**, and the "trial" vector $\boldsymbol{\ell}_{\mathbf{trial}}$, which is under consideration in the present iteration. The key idea in *Vicinity Search* is to determine the mth entry of the "real" vector $\boldsymbol{\ell}_{\mathbf{real}}$, denoted by $\boldsymbol{\ell}_{\mathbf{real}}(m)$ for $m = 1, 2, \ldots, nz$, to be the value of k in the close vicinity of $k = \boldsymbol{\ell}_{\mathbf{real}}(m)$, where a local extremum of $\mathbf{wei\_err}(k)$ with the same sign occurs. The location of these nz entries are simplified as follows.

In the first phase, the search of both local minima and maxima is reduced to that of local maxima by multiplying the values of $\mathbf{wei\_err}(k)$ for $1 \le k \le ngrid$ by $\text{sign}[\mathbf{wei\_err}(\boldsymbol{\ell}_{\mathbf{trial}}(m))]$ as in this case the values of the resulting signed weighted function $\text{sign}[\mathbf{wei\_err}(\boldsymbol{\ell}_{\mathbf{trial}}(m))] \times$

$\mathbf{wei\_err}(k)$ become positive at $k = \boldsymbol{\ell}_{\mathbf{trial}}(m)$ as well as in its proximity. In the second phase, the proper location of each $\boldsymbol{\ell}_{\mathbf{real}}(m)$ for $m = 1, 2, \ldots, nz$ can be obtained conveniently based on the following facts. First **Segment 1** guarantees that for $m > 1$ $[m < nz]$, the "signs" of $\mathbf{wei\_err}(k)$ at both $k = \boldsymbol{\ell}_{\mathbf{trial}}(m-1)$ and $k = \boldsymbol{\ell}_{\mathbf{trial}}(m+1)$ is opposite to that of $k = \boldsymbol{\ell}_{\mathbf{trial}}(m)$, or correspondingly, at $k = \boldsymbol{\ell}_{\mathbf{real}}(m)$. Second, during the course of the present search, $\boldsymbol{\ell}_{\mathbf{real}}(m-1)$ is located before $\boldsymbol{\ell}_{\mathbf{real}}(m)$ in such a way that the sign of $\mathbf{wei\_err}(k)$ at $k = \boldsymbol{\ell}_{\mathbf{real}}(m-1)$ is opposite to that of the sign of $\mathbf{wei\_err}(k)$ at $k = \boldsymbol{\ell}_{\mathbf{real}}(m+1)$. The above mentioned facts together with the reasoning that the lowest value of k for locating $\boldsymbol{\ell}_{\mathbf{real}}(1)$ is 1 and the highest value of k for locating $\boldsymbol{\ell}_{\mathbf{real}}(nz)$ is $ngrid$ inherently lead to carrying out *Vicinity Search* by using the following three steps:

Step 1: Set $m = 1$ and go to the next step.

Step 2: Find the mth element, denoted by $\widetilde{\boldsymbol{\ell}}_{\mathbf{real}}(m)$, at which the vector

$$\mathbf{err\_vicinity} = \text{sign}[\mathbf{wei\_err}(\boldsymbol{\ell}_{\mathbf{trial}}(m))]\mathbf{wei\_err}(low : upp), \tag{12a}$$

where

$$low = \begin{cases} 1 & \text{for } m = 1 \\ \max\{\boldsymbol{\ell}_{\mathbf{trial}}(m-1)+1, \boldsymbol{\ell}_{\mathbf{real}}(m-1)+1\} & \text{for } 2 \le m \le nz \end{cases} \tag{12b}$$

and

$$upp = \begin{cases} \boldsymbol{\ell}_{\mathbf{trial}}(m+1) - 1 & \text{for } 1 \le m \le nz-1 \\ ngrid & \text{for } m = nz \end{cases} \tag{12c}$$

achieves the maximum value. Generate $\boldsymbol{\ell}_{\mathbf{real}}(m) = \widetilde{\boldsymbol{\ell}}_{\mathbf{real}}(m) + low - 1$. If $m = 1$ $[m = nz]$, then store the corresponding maximum value as $\mathbf{err\_vic}(1)$ $[\mathbf{err\_vic}(nz)]$ to be used in *Endpoint Search* at **Segment 3** . Update $\boldsymbol{\ell}_{\mathbf{real}}(m)$ as $\boldsymbol{\ell}_{\mathbf{real}}(m) = \widetilde{\boldsymbol{\ell}}_{\mathbf{real}}(m) + low - 1$.

Step 3: Set $m = m + 1$. If $m < nz + 1$, then go to the previous step. Otherwise, go to *Step 1* under **Segment 3**.

Segment 3: This segment explains how to perform *Endpoint Search*. After *Vicinity Search*, the role of *Endpoint Search* is to check whether the weighted error function $\mathbf{wei\_err}(k)$ contains an additional local extremum before $k = \boldsymbol{\ell}_{\mathbf{real}}(1)$ [after $k = \boldsymbol{\ell}_{\mathbf{real}}(nz)$] such that its sign is opposite to that of occurring at $k = \boldsymbol{\ell}_{\mathbf{real}}(1)$ $[k = \boldsymbol{\ell}_{\mathbf{real}}(nz)]$. It is worth emphasizing that in order to take into account all the candidate extrema, *Endpoint Search* is necessary to be used after *Vicinity Search* as *Vicinity Search* totally omits the existence of these possible additional extrema.

The appearance of the additional first local extremum implies that [5]

$$upp_{\mathbf{end}} = \min\{\boldsymbol{\ell}_{\mathbf{trial}}(1) - 1, \boldsymbol{\ell}_{\mathbf{real}}(1) - 1\} \tag{13}$$

[5] *Vicinity Search* automatically guarantees that the sign of the weighted error function $\mathbf{wei\_err}(k)$ is same at both $k = \boldsymbol{\ell}_{\mathbf{trial}}(1)$ and $k = \boldsymbol{\ell}_{\mathbf{real}}(1)$. Hence, $upp_{\mathbf{end}}$ is the smallest value of k, where the sign of $\mathbf{wei\_err}(k)$ can be opposite before these values. Similarly, the sign of $\mathbf{wei\_err}(k)$ is same at both $k = \boldsymbol{\ell}_{\mathbf{trial}}(nz)$ and $k = \boldsymbol{\ell}_{\mathbf{real}}(nz)$ and $low_{\mathbf{end}}$ as given by (14) is the largest value of k, where the sign of $\mathbf{wei\_err}(k)$ can be opposite after these values.

is larger than or equal to 1. If this fact holds true, then the largest entry in the sub-vector $-\text{sign}[\textbf{wei\_err}(\boldsymbol{\ell}_{\textbf{real}}(1))] \cdot \textbf{wei\_err}(1 : upp_{\textbf{end}})$ should be positive. Similarly, the existence of the additional last local extremum implies that

$$low_{\textbf{end}} = \max\{\boldsymbol{\ell}_{\textbf{trial}}(nz) + 1, \boldsymbol{\ell}_{\textbf{real}}(nz) + 1\} \tag{14}$$

is smaller than or equal to $ngrid$. If this fact holds true, then the largest entry in the subvector $-\text{sign}[\textbf{wei\_err}(\boldsymbol{\ell}_{\textbf{real}}(nz))] \cdot \textbf{wei\_err}(low_{\textbf{end}} : ngrid)$ should be positive.

If no additional extremum exists, then the "final" $\boldsymbol{\ell}_{\textbf{real}}$ is the one found in *Vicinity Search*. Otherwise (that is, one or both the additional extrema exist), the final $\boldsymbol{\ell}_{\textbf{real}}$ is constructed according to the following alternatives:

Alternative 1: The additional last extremum exists such that its absolute value is larger than or equal to those of the first extremum found by *Vicinity Search* and the possible additional first extremum.

Alternative 2: The additional first extremum exists such that its absolute value is larger than or equal to that of the first extremum found by *Vicinity Search* and larger than that of the possible additional last extremum.

If *Alternative 1* (*Alternative 2*) is valid, then the final $\boldsymbol{\ell}_{\textbf{real}}$ is formed such that the first (last) entry of $\boldsymbol{\ell}_{\textbf{real}}$ of *Vicinity Search* is disregarded and the last (first) entry is the value of k for $1 \leq k \leq ngrid$, where the additional last (first) maximum of the signed weighted error function $-\text{sign}[\textbf{wei\_err}(\boldsymbol{\ell}_{\textbf{real}}(nz))] \cdot \textbf{wei\_err}(k)$ $[-\text{sign}[\textbf{wei\_err}(\boldsymbol{\ell}_{\textbf{real}}(1))] \cdot \textbf{wei\_err}(k)]$ occurs.

The above explanation is the key idea to perform *Endpoint Search* in the function **`firpm`**. However, the function **`firpm`** performs *Endpoint Search* in a lengthier manner and in order to exactly follow this strategy, it is carried out by using the following eight steps:

Step 1: Set $endsearch = 0$.

Step 2: Determine $upp_{\textbf{end}}$ *according to* (13). If $upp_{\textbf{end}} = 0$, then set $\textbf{err\_end}(1) = 0$. Otherwise, find the index, denoted by $\boldsymbol{\ell}_{\textbf{end\_real}}(1)$, where the vector

$$\textbf{err\_endpoint} = -\text{sign}[\textbf{wei\_err}(\boldsymbol{\ell}_{\textbf{real}}(1))] \cdot \textbf{wei\_err}(1 : \boldsymbol{\ell}_{\textbf{end}}(1)) \tag{15}$$

achieves the corresponding maximum entry value. Store this maximum entry value as $\textbf{err\_end}(1)$.

Step 3: If $\textbf{err\_end}(1) < \textbf{err\_vic}(nz)$, where $\textbf{err\_vic}(nz)$ has been saved at *Step 2* under **Segment 2**, then go to the next step. Otherwise, set $endsearch = 1$.

Step 4: Determine $low_{\textbf{end}}$ according to (14). If $low_{\textbf{end}} = ngrid + 1$, then go to *Step 6*. Otherwise, find the index, denoted by $\tilde{\ell}_{\textbf{end\_real}}(nz)$, where the vector

$$\textbf{err\_endpoint} = -\text{sign}[\textbf{wei\_err}(\boldsymbol{\ell}_{\textbf{real}}(nz))] \cdot \textbf{wei\_err}(low_{\textbf{end}} : ngrid) \tag{16}$$

achieves its maximum entry value. Set $\boldsymbol{\ell}_{\textbf{end\_real}}(nz) = \tilde{\boldsymbol{\ell}}_{\textbf{end\_real}}(nz) + low_{\textbf{end}} - 1$ and store the corresponding maximum entry value as $\textbf{err\_end}(nz)$.

Step 5: If $\textbf{err\_end}(nz) < \max\{\textbf{err\_end}(1), \textbf{err\_vic}(1)\}$, where $\textbf{err\_vic}(1)$ has been saved at *Step 2* under **Segment 2**, then go to the next step. Otherwise, set $endsearch = 2$.

Step 6: If *endsearch* $= 0$, then go to *Step 1* under **Segment 4**. Otherwise, go to the next step.

Step 7: If *endsearch* = *1*, then set $\ell_{\textbf{real}}(nz + 1 - m) = \ell_{\textbf{real}}(nz - m)$ for $m = 1, 2, \ldots, nz - 1$ and $\ell_{\textbf{real}}(1) = \ell_{\textbf{end\_real}}(1)$ and got to *Step 1* under **Segment 4**. Otherwise, go to the next step.

Step 8: If *endsearch* $= 2$, set $\ell_{\textbf{real}}(m) = \ell_{\textbf{real}}(m + 1)$ for $m = 1, 2, \ldots, nz - 1$ and $\ell_{\textbf{real}}(nz) = \ell_{\textbf{end\_real}}(nz)$. Go to *Step 1* under **Segment 4**.

Segment 4: This concluding segment check the convergence of the Remez exchange loop as follows. If the entries of the vectors $\ell_{\textbf{trial}}$ and $\ell_{\textbf{real}}$ are the same, then stop as in this case the ultimate goal $\ell_{\textbf{opt}} \equiv \ell_{\textbf{real}} \equiv \ell_{\textbf{trial}}$ has been achieved. Otherwise, use the present "real" vector $\ell_{\textbf{real}}$ as the "trial" vector for the subsequent iteration by using the substitution $\ell_{\textbf{trial}} = \ell_{\textbf{real}}$ and go to *Step 1* under **Segment 1**. Continue the Remez loop until $\ell_{\textbf{trial}}$ and $\ell_{\textbf{real}}$ coincide or the value of the iteration counter *niter* exceeds $itrmax = 250$, which is extremely unlikely to occur.

This segment requires only the following two basic steps:

Step 1: If $\ell_{\textbf{trial}}$ and $\ell_{\textbf{real}}$ coincide, then stop. Otherwise, go to the next step.

Step 2: Set $\ell_{\textbf{trial}} = \ell_{\textbf{real}}$ and $niter = niter + 1$. If $niter > itrmax$, then stop. Otherwise, go to *Step 1* under **Segment 1**.

3.2. MATLAB code snippet

The code pasted below has been tested for realizing Implementation I by using MATLAB version 7.11.0.584($R2010b$). In order to embed this code snippet in the MATLAB function **firpm**, edit the function by taking away the code between lines 214 and 514, introduce the function call (the first line of the function of **remez_imp1**) and copy the function at the end of the file. Remember to take the backup copy of the original function. It is worth emphasizing that the implementation of Remez algorithm in the function **firpm** uses approximately 15 nested loops and 300 lines of code, whereas the code snippet provided below requires only 3 looping structures and approximately 100 lines of code to achieve the same optimum solution.

```
function [x,y,ad,dev] = remez_imp1(nz,iext,ngrid,grid,des,wt)
% remez_imp1 implements the Segments 1 - 4 described in the preceding
% section, the function needs to be inserted within the MATLAB function
% firpm. The input argument values come directly from the function firpm
% and the output arguments are required to perform the Inverse Fourier
% transform in order to calculate the filter coefficients. In case of
% any issues send an e-mail to muhammad"dot"ahsan"at"tut "dot" fi.
% Last updated 04.15.2012 4:15 AM (UTC/GMT+2)

% INITIALIZATIONS PHASE
niter = 1;          % Initialize the iteration counter.
itrmax = 250;    % Maximum number of iterations.
l_trial = iext(1:nz)';  % Startup value of l_trial.

% ITERATION PHASE
% REMEZ LOOP FOR LOCATING DESIRED nz INDICES AMONG THE GRID POINTS
while (niter < itrmax)

% SEGMENT 1: BASED ON THE PRESENT 'TRIAL' VECTOR l_trial, GENERATE THE
% WEIGHTED ERROR FUNCTION wei_err(k) AT ALL THE GRID POINTS
     x = cos(2*pi*grid(l_trial));  % Step 1: Lagrange abscissa vector x.
     A = x'*ones(1,nz)-ones(nz,1)*x;
     A(eye(nz)==1) = 1;
     ad = prod(A);
     ad = ad * (-2)^(nz-1);  % Step 1: Lagrange coefficient vector ad...
     ad = 1./ad;  % found efficiently without using the function remezdd.
     add = ones(size(ad));
     add(2:2:nz) = -add(2:2:nz);
     dnum = ad*des(l_trial)';
     dden = add*(ad./wt(l_trial))';
     dev = -dnum/dden;  % Step 1: Current value of deviation.
     % Step 2: Lagrange ordinate vector y
     y = des(l_trial) + dev*add./wt(l_trial);
     % Step 3: Overall abscissa vector x_all
     x_all = cos(2*pi*grid(1:ngrid));
     err_num = zeros(1,ngrid);  % Step 4: Initializations of err_num...
     err_den = err_num;  % and err_den.
     for jj = 1:nz  % Steps 5 and 6: Intermediate evaluations for...
         aid = ad(jj)./(x_all - x(jj)); % obtaining the weighted error...
         err_den = err_den + aid;  % wei_err(k) at all the grid points.
         err_num = err_num + y(jj)*aid;
     end
     err_cy = err_num./err_den;
     wei_err = (err_cy - des).*wt; % Step 7: Generate the vector wei_err.
     dev_vect = ones(size(l_trial));  % Steps 8-10: Fill in the undefined
     dev_vect(2:2:length(l_trial))= -dev_vect(2:2:length(l_trial));
     dev_vect = dev_vect * dev;  % entries of wei_err at l_trial(1:nz)...
     wei_err(l_trial)=dev_vect;   % by using the values of dev (-dev).

% SEGMENT 2: PERFORM VICINITY SEARCH
     for k=1:nz              % Steps 1,2, and 3: Start of Vicinity search.
         if k==1
            low = 1;
            err_vicinity = sign(wei_err(l_trial(k)))*...
            wei_err(low:l_trial(2)-1);
         elseif k==nz
            low = max(l_trial(k-1)+1,l_real(k-1)+1);
            err_vicinity = sign(wei_err(l_trial(k)))*wei_err(low:ngrid);
         else
         low = max(l_trial(k-1)+1,l_real(k-1)+1);
         err_vicinity = sign(wei_err(l_trial(k)))* ...
         wei_err(low:l_trial(k+1)-1);
         end
         [~,ind_vicinity]=max(err_vicinity); % tilde operator does not...
         % work with older MATLAB releases If you are running an older...
         % MATLAB version, considering replacing it with a dummy value.
         l_real(k) = ind_vicinity+low-1;
        if k==1 || k==nz  % Step 3: Find err_vic(1)=wei_err(l_real(1))...
            err_vic(k) = wei_err (l_real(k));
        end  % and err_vic(nz)=wei_err(l_real(nz)) for use at STEP III.
     end                      % Steps 1, 2, and 3: End of Vicinity search.

% SEGMENT 3: PERFORM ENDPOINT SEARCH
     endsearch=0;  % Step 1: Start Endpoint search.
     err_end(1) = 0;  % Step 2: Needed for the case, where upp_end = 0.
     if l_real(1)>1 && l_trial(1)> 1  % Step 2: Find l_end_true(1)...
         upp_end = min(l_real(1)-1,l_trial(1)-1);  % and err_end(1).
         err_endpoint = -sign(wei_err(l_real(1)))*wei_err(1: upp_end);
         [~,ind_endpoint]=max(err_endpoint);
```

```
            l_end_real(1) = ind_endpoint;
            err_end(1) = -sign(wei_err(l_real(1)))*wei_err(l_end_real(1));
            if err_end(1) > abs(err_vic(nz))  % Step 3:Use 'endsearch=1'...
                endsearch=1;  % or not?
            end
        end
        if l_real(nz) < ngrid & l_trial(nz) < ngrid  % Step 4: Find...
            low_end = max(l_real(nz)+1,l_trial(nz)+1);  % l_end_real(nz)...
            err_endpoint = -sign(wei_err(l_real(nz)))*wei_err(low_end:ngrid);
            [~,ind_endpoint]=max(err_endpoint);  % and err_end(nz).
            l_end_real(nz) = ind_endpoint+low_end-1;
            err_end(nz) = -sign(wei_err(l_real(nz)))*wei_err(l_end_real(nz));
            if err_end(nz) > max(abs(err_vic(1)), err_end(1)) % Step 5:...
                endsearch=2;  % Use 'endsearch=2' or not?
            end
        end
        if endsearch == 1   % Step 7: 'endsearch=1' is valid. Form...
            l_real=[l_end_real(1) l_real(1:nz-1)];  %  l_real accordingly.
        elseif endsearch == 2    % Step 8: 'endsearch=2' is true. Form...
            l_real=[l_real(2:nz) l_end_real(nz)];  % l_real accordingly.
        end                                        % End of Endpoint search.

  % SEGMENT 4: TEST CONVERGENCE
        if (l_real == l_trial)  % Step 1: The real and trial vectors...
            break;   % coincide. Hence, stop. Remez loop ended successfully.
        else
            l_trial = l_real;  % Step 2: Otherwise, replace the values of...
            niter = niter + 1;  % l_trial with the values of l_real and...
        end % continue.
    end % END OF THE OVERALL REMEZ LOOP
```

3.3. Performance comparison

This section presents a performance comparison between the Implementation I and the implementation available in the MATLAB function **`firpm`**. The performance measurement criteria is the time taken by both the implementations to design a particular filter and is measured with the help of MATLAB built-in function **`profiler`**. This function indicates the CPU execution time taken by a function and provides the following information:

- Calls – The number of times the function was called while profiling was on.
- Total Time – The total time spent in a function, including all child functions called, in seconds.
- Self Time – The total time taken by an individual function, not including the time for any child functions called, in seconds.

Time measurement was carried out on an IBM ThinkCentre machine equipped with Intel Core 2 Duo processor E6550 running at a speed of 2.33 GHz with a memory of 3 GB.

The following four filters were designed with both implementations. After the examples, the time taken by them will be tabulated in Table 1.

Example 1: It is desired to design a lowpass filter meeting the following criteria:

$$\omega_p = 0.05\pi, \omega_s = 0.1\pi, \delta_p = 0.01, \text{ and } \delta_s = 0.001.$$

The minimum order to meet these criteria is 108 and the relevant MATLAB commands are

```
>> [N,F,A,W] = firpmord([0.05 0.1],[1 0],[0.01 0.001]);
>> firr_coeff = firremez_imp1(N+6,F,A,W);
>> fvtool(firr_coeff); % filter visualization tool
```

The magnitude response of the resulting filter is shown in Fig. 1.

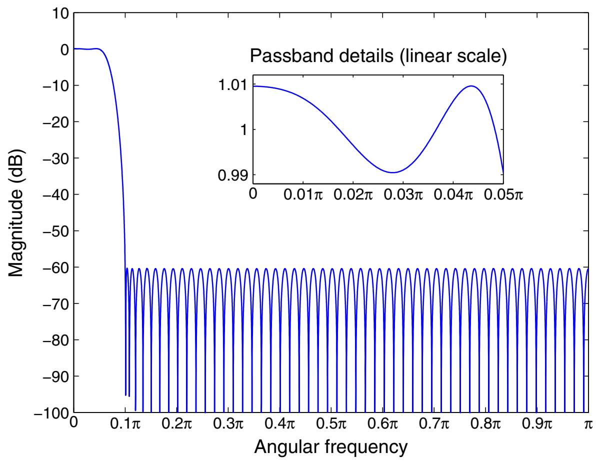

Figure 1. Magnitude response of the lowpass filter of Example 1.

Example 2: It is desired to design a highpass filter meeting the following criteria:

$$\omega_s = 0.02\pi, \omega_p = 0.05\pi, \delta_p = 0.01, \text{ and } \delta_s = 0.001.$$

The minimum order to meet these criteria is 172 and the relevant MATLAB commands are

```
>> [N,F,A,W] = firpmord([0.02 0.05],[0 1],[0.001 0.01]);
>> firr_coeff = firremez_imp1(N-4,F,A,W);
```

The magnitude response of the resulting filter is shown in Fig. 2.

Example 3: It is desired to synthesize a bandpass filter meeting the following criteria:

$$\omega_{s1} = 0.2\pi, \omega_{p1} = 0.25\pi, \omega_{p2} = 0.6\pi, \omega_{s2} = 0.7\pi, \delta_p = \delta_{s2} = 0.01, \text{ and } \delta_{s1} = 0.001.$$

The minimum order required to meet the criteria is 102 and the relevant MATLAB commands are

```
>> [N,F,A,W] = firpmord([0.2 0.25 0.6 0.7],[0 1 0],[0.001 .01 .01]);
>> firr_coeff = firremez_imp1(N,F,A,W);
```

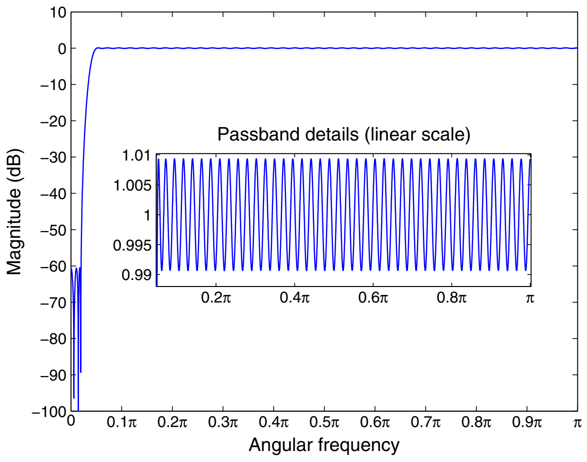

Figure 2. Magnitude response of the highpass filter of Example 2.

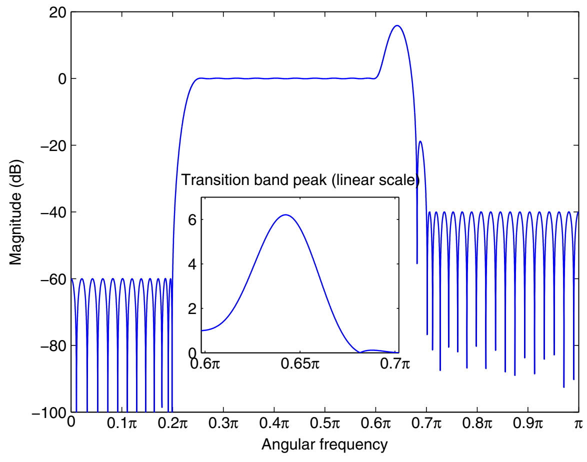

Figure 3. Magnitude response and the highest transition band peak of the bandpass filter of Example 3.

Figure 3 illustrates the magnitude response of the resulting filter. Although the amplitude response is optimal according to *Alternation Theorem*, it is worth noting that this particular filter has an extra peak in the second transition band region of approximately 16 dB. This is because of the fact that the approximation interval is a union of passband and stopband regions and transition bands are considered as "don't care" regions. This assumption works perfectly for filters having bands less than three. However, in case of three or more bands, there is no guarantee that the response is well-behaved in the transition bands, even though it is optimal according to the approximation theory. This fact is especially prominent if any one of the following holds true [9]:

- Transition bandwidths are very large compared to the passband and/or stopband widths.
- Width of transition bands is different; the larger the difference, the greater the problem.

In order to conveniently avoid the appearance of the unwanted transition peaks, consider an original problem stated for linear-phase Type I and Type II filters as follows.

First, there are R interlaced passband and stopband regions as given by

$$\Omega_\kappa = \left[\omega_\kappa^{(low)}, \omega_\kappa^{(upp)}\right] \text{ for } \kappa = 1, 2, \ldots, R. \tag{17a}$$

Such that these regions do not overlap. The lower and upper limits for the zero-phase frequency response in these bands are, respectively, specified as

$$L_\kappa^{(low)} = D_\kappa - \delta_\kappa \text{ and } L_\kappa^{(upp)} = D_\kappa + \delta_\kappa \tag{17b}$$

Here, the D_κ's alternatingly achieve the values of zero and unity such that the first value is zero (unity) if the first band is a stopband (passband). For this original problem, the overall approximation region is

$$\Omega = \Omega_1 \cup \Omega_2 \cup \ldots \cup \Omega_R. \tag{17c}$$

This region can be extended to cover the transition bands as follows:

$$\widehat{\Omega} = \Omega_1 \cup \Omega_1^T \cup \Omega_2 \cup \Omega_2^T \cup \ldots \Omega_{R-1}^T \cup \Omega_R, \tag{17d}$$

where

$$\Omega_\kappa^T = [\omega_\kappa^{(upp)} + \alpha, \omega_\kappa^{(low)} - \alpha] \text{ for } \kappa = 1, 2, \ldots, R-1. \tag{17e}$$

In order to guarantee that $\widehat{\Omega}_\kappa$ is still a closed subset of $[0, \pi]$, α should be a small positive number.[6] There are two natural ways to state the transition band constraints, referred to as *Type A* and *Type B* transition band constraints. For both types, the upper and lower limits for the zero-phase frequency response in the κth transition band Ω_κ^T are specified as follows. For both *Type A* and *Type B*, the upper limit is[7]

$$\widehat{L}_\kappa^{(upp)} = \max\{D_\kappa + \delta_\kappa, D_{\kappa+1} + \delta_{\kappa+1}\}, \tag{17f}$$

whereas the lower limits depend on the type as follows:

$$\widehat{L}_\kappa^{(low)} = \begin{cases} \min\{D_\kappa - \delta_\kappa, D_{\kappa+1} - \delta_{\kappa+1}\}, & \text{for } \textit{Type A} \\ -\widehat{L}_\kappa^{(upp)} & \text{for } \textit{Type B} \end{cases} \tag{17g}$$

The above limits for *Type A* are determined such that if the filter meets the overall criteria, then the maximum (minimum) value of the zero-phase frequency response in each transition band

[6] The only condition for α is that it should be small enough to avoid the extra peaks between the adjacent passbands and the newly formed intervals in the transition band regions.

[7] It is worth emphasizing that the use of $\max\{D_\kappa + \delta_\kappa, D_{\kappa+1} + \delta_{\kappa+1}\}$ implies that the maximum allowable value in the nearest passband is the upper limit. Similarly, in the following equation, $\min\{D_\kappa - \delta_\kappa, D_{\kappa+1} - \delta_{\kappa+1}\}$ means that the lower limit is the one in the nearest stopband.

is less than or equal to the stated upper limit in the nearest passband region (larger than or equal to the stated lower limit in the nearest stopband region). For *Type B*, in turn, the upper limit is the same, whereas the lower limit is obtained from the upper limit by changing its sign, thereby indicating that the magnitude response of the filter is less than or equal to the stated upper limit in the nearest passband region.

The desired value in the κth transition band Ω_κ^T for both types is the average of the corresponding lower and upper limits, whereas the admissible deviation is the difference between the upper limit and the above-mentioned desired value. Hence, in Ω_κ^T for $\kappa = 1, 2, \ldots, R-1$ the desired values, denoted by $\widehat{D}_\kappa$, and the admissible deviations, denoted by $\widehat{\delta}_\kappa$, are as follows:

$$\widehat{D}_\kappa = \begin{cases} [\max\{D_\kappa + \delta_\kappa, D_{\kappa+1} + \delta_{\kappa+1}\} + \min\{D_\kappa - \delta_\kappa, D_{\kappa+1} - \delta_{\kappa+1}\}]/2 & \text{for } \textit{Type A} \\ 0 & \text{for } \textit{Type B}. \end{cases} \tag{17h}$$

and

$$\widehat{\delta}_\kappa = \begin{cases} \max\{D_\kappa + \delta_\kappa, D_{\kappa+1} + \delta_{\kappa+1}\} - \widehat{D}_\kappa & \text{for } \textit{Type A} \\ \max\{D_\kappa + \delta_\kappa, D_{\kappa+1} + \delta_{\kappa+1}\} & \text{for } \textit{Type B}. \end{cases} \tag{17i}$$

The following MATLAB function converts the original design specifications into those ones including either *Type A* or *Type B* transition band constraints as follows. The first three input parameters **F_ori**, **Des_ori**, and **Dev_ori** contain the $2R$ edges of the R bands as a fraction of π as well as the desired values and the admissible deviations from these values in the R bands in the original specifications. . "alpha" corresponds directly to α which is used in (17e), whereas *itype=1* (*itype*=2) means that *Type A* (*Type B*) transition band constraints are in use. The output of this function consists of vectors **F**, **Des**, and **Wt** that are in the desired form when calling the MATLAB function **firpm** in its original form or its modifications referred to as Implementation I or II in this contribution.

```
function [F,Des,Wt]=convert2constrt(F_ori, Des_ori,Dev_ori,alpha,itype)
% This function converts the original deisgn specifications into Type A
% or Type B  transition band constraints compatible specifications.
%
% Input parameters:
% - F_ori contains the edges of the R bands, where R = length(Des_s).
% - Des_ori contains the desired values in the R bands.
% - Dev_ori contains the admissiable deviations in the R bands.
% - alpha is a small positive constant.
% - type=1 (2) generates Type A (B) transition band constraints.
%
%  Output parameters
% - F contains the edges of the 2R-1 bands.
% - Des contains the desired values on all the edges of 2R-1 bands.
% - Wt contains the weights in the 2R-1 bands.

% Check if the input data is correct
 if (alpha < 0)
     error('alpha should be a small positive number.');
 end
R = numel(Des_ori);

```

```
% Generate the output parameters
for k=1:R
    F(4*(k-1)+1)=F_ori(2*k-1); F(4*(k-1)+2)=F_ori(2*k);
    Des(2*(k-1)+1)=Des_ori(k); Dev(2*(k-1)+1)=Dev_ori(k);
end
for k=1:R-1
    F(4*(k-1)+3)=F_ori(2*k)+alpha;F(4*(k-1)+4)=F_ori(2*k+1)-alpha;
    aid1=max(Des_ori(k)+Dev_ori(k),Des_ori(k+1)+Dev_ori(k+1));
    aid2=min(Des_ori(k)-Dev_ori(k),Des_ori(k+1)-Dev_ori(k+1));
    if itype==1
        Des(2*k) = (aid1+aid2)/2;
        Dev(2*k) = aid1-Des(2*k);
    elseif itype==2
        Des(2*k) = 0;
        Dev(2*k) = aid1;
    else
        error('Type should be either 1 or 2.');
    end
end
temp = Des(ones(2,1),:);
F = F';
Des = temp(:);
Wt = (1./Dev)';
```

When using the above MATLAB function with $\alpha = 0.0005$, the ten band-edges of the five bands as fractions of π for both *Type A* and *Type B* transition band constraints become

$$\widehat{\Omega}_{\kappa} = [0, 0.2, 0.2005, 0.2495, 0.25, 0.6, 0.6005, 0.6995, 0.7, 1]\,.$$

The corresponding desired and weight values for *Type A* and *Type B* transition band constraints are, respectively,

$$\begin{aligned} D_A &= [0, 0.5045, 1, 0.5, 0] \\ W_A &= [1000, 1.9782, 100, 1.9608, 100], \end{aligned}$$

and

$$\begin{aligned} D_B &= [0, 0, 1, 0, 0] \\ W_B &= [1000, 0.9901, 100, 0.9901, 100]. \end{aligned}$$

The relevant MATLAB commands are

```
>> % Data for Type A design
>> [F1,A1,W1] = convert2constrt([0 0.2 0.25 0.6 0.7 1],[0 1 0],...
[.001 .01 .01],0.0005,1);
>> N1 = 103; firr_coeff1 = firremez_imp1(N1,F1,A1,W1);
>> % Data for Type B design
>> [F2,A2,W2] = convert2constrt([0 0.2 0.25 0.6 0.7 1],[0 1 0],...
[.001 .01 .01],0.0005,2); firr_coeff2 = firremez_imp1(N1,F2,A2,W2);
```

As seen in Fig. 4, by increasing the original filter’s order from 102 to 103, the transition bands constraints guarantee that the overall response of the filters stays within the desired limits.

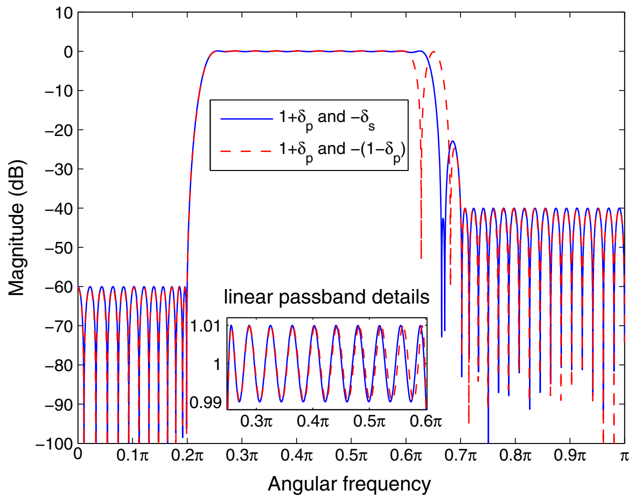

Figure 4. Magnitude response of the two bandpass filters of Example 3.

Example 4: It is required to synthesize a bandstop filter meeting the following criteria:

$$\omega_{p1} = 0.15\pi, \omega_{s1} = 0.3\pi, \omega_{s2} = 0.6\pi, \omega_{p2} = 0.65\pi, \delta_{p1} = \delta_{p2} = 0.01, \text{ and } \delta_s = 0.001.$$

The minimum order required to meet these criteria is 102 and the relevant MATLAB commands are

```
>> [N,F,A,W] = firpmord([0.15 0.3 0.6 0.65],[1 0 1],[0.01 0.001 0.01]);
>> firr_coeff = firremez_imp1(N-4,F,A,W);
```

The magnitude response of the resulting bandstop filter is shown in Fig. 5. This response is the best one according to *Alternation Theorem*, but contains two extra peaks of approximately 33 and 15 dB in the first transition band. By using the technique described above, the transition band peaks can be attenuated to an acceptable level. The relevant MATLAB commands are

```
>> % Data for Type A design
>> [F1,A1,W1] = convert2constrt([0 0.15 0.3 0.6 0.65 1],[1 0 1],...
[0.01 0.001 0.01],0.0005,1);
>>  N1 = 102; firr_coeff1 = firremez_imp1(N1,F1,A1,W1);
>> % Data for Type B design
>> [F2,A2,W2] = convert2constrt([0 0.15 0.3 0.6 0.65 1],[1 0 1],...
[0.01 0.001 0.01],0.0005,2); firr_coeff2 = firremez_imp1(N1,F2,A2,W2);
```

As seen in Fig. 6, the overall response of the filters of the same order as the original one, that is, 102, stay within the desired limits for both *Type A* and *Type B* transition band constraints. Among these two constrained designs, *Type A* constrained design is preferred as for it the undershoot of the zero-phase frequency response is limited to be larger than or equal to $-\delta_s = -0.001$. Furthermore, the response in the first passband remains equiripple.

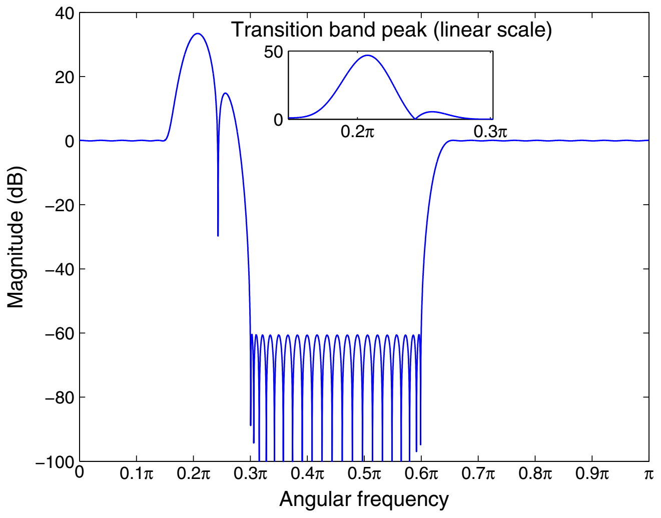

Figure 5. Magnitude response of the bandstop filter of Example 4.

Table 1 indicates the outcomes obtained from the original implementation and the Implementation I of the Remez algorithm, both of which work practically in the same manner. It is evident that the time required by the Implementation I is almost one third of the time taken by the original implementation and illustrates the superiority of the proposed MATLAB implementation of the algorithm.

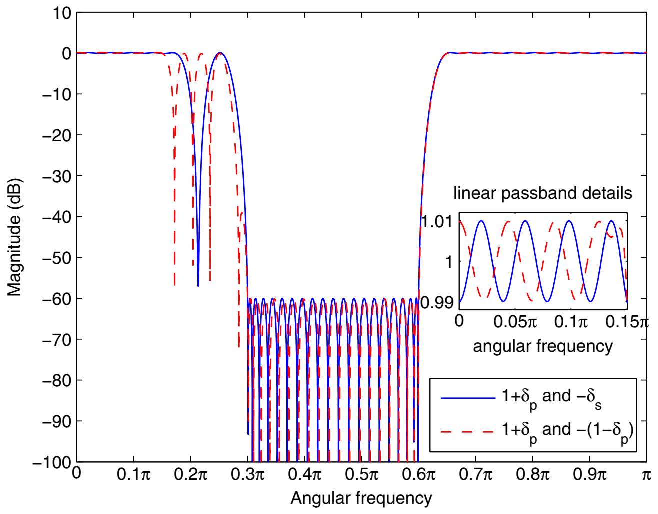

Figure 6. Magnitude response of the two bandstop filters of Example 4.

Example	Original Implementation		Implementation I	
	Total Time*	Self Time	Total Time	Self Time
1	0.152	0.125	0.032	0.032
2	0.235	0.136	0.064	0.064
$3^{\top}$	0.159	0.120	0.032	0.032
$3^{\perp A}$	0.262	0.191	0.056	0.056
$3^{\perp B}$	0.307	0.184	0.061	0.061
$4^{\top}$	0.198	0.138	0.047	0.047
$4^{\perp A}$	0.272	0.169	0.051	0.051
$4^{\perp B}$	0.318	0.186	0.055	0.055

*Time in seconds
$^{\top}$Transition bands are excluded.
$^{\perp A}$ *Type A* transition band constraints are used.
$^{\perp B}$ *Type B* transition band constraints are used.

Table 1. Performance Comparison of Original Implementation and Implementation I

4. Implementation II

This section discusses the Implementation II in detail. First, a theoretical formulation of the algorithm is presented and, then, the corresponding MATLAB code is specified. After that, a detailed comparison shows how this implementation is superior to the original implementation of the Remez algorithm in the function **`firpm`**, in terms of significant reductions in the number of iterations and the CPU execution time required to generate the same optimum solution, especially in multiband cases.

4.1. Theoretical formulation

As mentioned in the introduction, the key difference between Implementations I and II is the search strategies employed for the "real" extremal points of the weighted error function, which is formed based on the "trial" extremal points. Consequently, **Segment 1** and **Segment 4** are same for both the implementations. The remaining **Segment 2** and **Segment 3** along with the accompanying steps are as follows.

Segment 2: Based on the values of $\mathbf{wei\_err}(k)$ for $1 \leq k \leq ngrid$ generated at **Segment 1**, this main step generates the vector $\boldsymbol{\ell}_{\mathbf{real}}^{(\mathbf{start})}$ to include as many entries as possible in the ascending order subject to the following three conditions:

Condition 1: At each entry of $\boldsymbol{\ell}_{\mathbf{real}}^{(\mathbf{start})}$, $\mathbf{wei\_err}(k)$, when regarded as a function of k, achieves a local extremum whose absolute value is larger than or equal to $|dev|$, where $|dev|$ is determined according to (6).

Condition 2: In case of several consecutive local extrema of $\mathbf{wei\_err}(k)$ with the same sign for $k^{(low)} \leq k \leq k^{(upp)}$, only one entry is included in $\boldsymbol{\ell}_{\mathbf{real}}^{(\mathbf{start})}$ and its value is the value of k for $k^{(low)} \leq k \leq k^{(upp)}$, where $|\mathbf{wei\_err}(k)|$ achieves its maximum value.

Condition 3: The sign of **wei_err** as a function of k alternates at the consecutive enteries of $\boldsymbol{\ell}_{\textbf{real}}^{(\textbf{start})}$.

This vector $\boldsymbol{\ell}_{\textbf{real}}^{(\textbf{start})}$ serves as a start-up vector for generating the "real" vector $\boldsymbol{\ell}_{\textbf{real}}$ at **Segment 3**. This segment is carried out using the following four steps:

Step 1: Find all the values of k in the range $1 \leq k \leq ngrid$, where a local extremum of $\textbf{wei\_err}(k)$ occurs. Store these values of k in the ascending order into the vector $\boldsymbol{\ell}_{\textbf{aid1}}$.

Step 2: Extract those entries from $\boldsymbol{\ell}_{\textbf{aid1}}$, where the absolute value of **wei_err** is larger than or equal to $|dev|$, and store these entries into the vector $\boldsymbol{\ell}_{\textbf{aid2}}$.

Step 3: Split the range $1 \leq \kappa \leq n_{\textbf{aid2}}$, where $n_{\textbf{aid2}}$ is the length of $\boldsymbol{\ell}_{\textbf{aid2}}$, into the sub-ranges $\kappa^{(low)}(m) \leq \kappa \leq \kappa^{(upp)}(m)$ for $m = 1, 2, \ldots, n_{\textbf{aid2}}$ in such a way that the signs of $\textbf{wei\_err}(\boldsymbol{\ell}_{\textbf{aid2}}(\kappa))$ in the mth sub-range as given by $\kappa^{(low)}(m) \leq \kappa \leq \kappa^{(upp)}(m)$ are the same.

Step 4: Generate the vector $\boldsymbol{\ell}_{\textbf{real}}^{(\textbf{start})}$ of length $nz_{\textbf{start}}$ such that its mth entry is the value among $\boldsymbol{\ell}_{\textbf{aid2}}(\kappa)$ for $\kappa^{(low)}(m) \leq \kappa \leq \kappa^{(upp)}(m)$, at which $|\textbf{wei\_err}(\boldsymbol{\ell}_{\textbf{aid2}}(\kappa))|$ achieves its maximum value. Go to *Step 1* under **Segment 3**.

Segment 3: Based on the vector $\boldsymbol{\ell}_{\textbf{real}}^{(\textbf{start})}$ generated at **Segment 2**, this main step extracts from its enteries the nz enteries to be included in the "real" vector $\boldsymbol{\ell}_{\textbf{real}}$ such that the largest absolute values of $\textbf{wei\_err}(k)$, when regarded as a function of k, are retained subject to the condition that the maxima and minima alternate at the consecutive extracted enteries. If the length of $\boldsymbol{\ell}_{\textbf{real}}^{(\textbf{start})}$ is nz, then $\boldsymbol{\ell}_{\textbf{real}} \equiv \boldsymbol{\ell}_{\textbf{real}}^{(\textbf{start})}$ and no extraction is required. Otherwise, the extraction is performed using the following steps:

STEP A: Denote the length of $\boldsymbol{\ell}_{\textbf{real}}^{(\textbf{start})}$ by $n_{\textbf{real}}^{(\textbf{start})}$. If $n_{\textbf{real}}^{(\textbf{start})} - nz$ is an odd integer, then the first (last) entry is discarded from $\boldsymbol{\ell}_{\textbf{real}}^{(\textbf{start})}$ if the absolute value of $\textbf{wei\_err}(k)$ at $k = \boldsymbol{\ell}_{\textbf{real}}^{(\textbf{start})}(1)$ is less than or equal (larger) than the corresponding value at $k = \boldsymbol{\ell}_{\textbf{real}}^{(\textbf{start})}$ $(k = n_{\textbf{real}}^{(\textbf{start})})$. Go to the next step.

STEP B: Denote the remaining vector and its length by $\tilde{\boldsymbol{\ell}}_{\textbf{real}}^{(\textbf{start})}$ and $\tilde{n}_{\textbf{real}}^{(\textbf{start})}$, respectively. If $\tilde{n}_{\textbf{real}}^{(\textbf{start})} - nz = 0$, then stop. Otherwise go to the next step.

STEP C: Since $\tilde{n}_{\textbf{real}}^{(\textbf{start})} - nz$ is an even integer, two entries of $\tilde{\boldsymbol{\ell}}_{\textbf{real}}^{(\textbf{start})}$ should be simultaneously discarded. There are altogether $\tilde{n}_{\textbf{real}}^{(\textbf{start})}$ optional pairs such that the first pair consists of the first and last entries of $\tilde{\boldsymbol{\ell}}_{\textbf{real}}^{(\textbf{start})}$ and the remaining ones are $\tilde{n}_{\textbf{real}}^{(\textbf{start})} - 1$ consecutive entry pairs. The pair to be discarded is the pair, where the maximum of two absolute values of $\textbf{wei\_err}(k)$ is the smallest. Go to *STEP C*.

This segment is carried out using the following seven steps:

Step 1: Set $\boldsymbol{\ell}_{\textbf{real}}^{(\textbf{init})} = \boldsymbol{\ell}_{\textbf{real}}^{(\textbf{start})}$. If $nz_{\textbf{real}}^{(\textbf{init})} - nz$, where $nz_{\textbf{real}}^{(\textbf{init})}$ is the length of $\boldsymbol{\ell}_{\textbf{real}}^{(\textbf{init})}$, is zero, then set $\boldsymbol{\ell}_{\textbf{real}} = \boldsymbol{\ell}_{\textbf{real}}^{(\textbf{init})}$ and go to *Step 1* under **Segment 4**. Otherwise, go to the next step.

Step 2: If $nz_{\textbf{real}}^{(\textbf{init})} - nz$ is an even integer, then go to *Step 4*. Otherwise, go to the next step.

Step 3: If $|\textbf{wei\_err}(\boldsymbol{\ell}^{(\text{init})}_{\textbf{real}}(1))| \leq |\textbf{wei\_err}(\boldsymbol{\ell}^{(\text{init})}_{\textbf{real}}(nz^{(\text{init})}_{\textbf{real}}))|$, then discard the first entry from $\boldsymbol{\ell}^{(\text{init})}_{\textbf{real}}$. Otherwise, discard the last entry from $\boldsymbol{\ell}^{(\text{init})}_{\textbf{real}}$. Go to the next step.

Step 4: If $nz^{(\text{init})}_{\textbf{real}} - nz$, where $nz^{(\text{init})}_{\textbf{real}}$ is the length of the remaining vector $\boldsymbol{\ell}^{(\text{init})}_{\textbf{real}}$, is zero, then set $\boldsymbol{\ell}_{\textbf{real}} = \boldsymbol{\ell}^{(\text{init})}_{\textbf{real}}$ and go to *Step 1* under **Segment 4**. Otherwise, go to the next step.

Step 5: Determine the entries of the vector **wei_comp** as follows:

$$\textbf{wei\_comp}(m) = \max\left(|\textbf{wei\_err}(\boldsymbol{\ell}^{(\text{init})}_{\textbf{real}}(m))|, |\textbf{wei\_err}(\boldsymbol{\ell}^{(\text{init})}_{\textbf{real}}(m+1))|\right) \quad \text{for } m = 1, 2, \ldots, nz^{(\text{init})}_{\textbf{real}} - 1. \tag{18}$$

Step 6: If $\max\left(|\textbf{wei\_err}(\boldsymbol{\ell}^{(\text{init})}_{\textbf{real}}(1))|, |\textbf{wei\_err}(\boldsymbol{\ell}^{(\text{init})}_{\textbf{real}}(nz^{(\text{init})}_{\textbf{real}}))|\right)$ is less than or equal to the largest entry of **wei_comp**, then remove the first and last entries from $\boldsymbol{\ell}^{(\text{init})}_{\textbf{real}}$ and go to *Step 4*. Otherwise, go to the next step.

Step 7: Find the entry of **wei_comp** with the smallest value. If this is the mth entry, then remove the mth and the $(m+1)$th entries from $\boldsymbol{\ell}^{(\text{init})}_{\textbf{real}}$. Go to *Step 4*.

4.2. MATLAB code snippet

The code pasted below has been tested and implemented with MATLAB version 7.11.0.584 (R2010b). It can be embedded into the MATLAB function **`firpm`** in a similar fashion to Implementation I.

```matlab
function [x,y,ad,dev] = remez_imp2(nz,iext,ngrid,grid,des,wt)
% remez_imp1 implements the Segments 1 - 4 described in the preceding
% section, the function needs to be inserted within the MATLAB function
% firpm. The input argument values come directly from the function firpm
% and the output arguments are required to perform the Inverse Fourier
% transform in order to calculate the filter coefficients. In case of
% any issues send an e-mail to muhammad"dot"ahsan"at"tut "dot" fi.
% Last updated 04.15.2012 4:54 AM (UTC/GMT+2)

% INITIALIZATIONS PHASE
niter = 1;          % Initialize the iteration counter.
itrmax = 250;    % Maximum number of iterations.
l_trial = iext(1:nz)';  % Startup value of l_trial.

% ITERATION PHASE
% REMEZ LOOP FOR LOCATING DESIRED nz INDICES AMONG THE GRID POINTS
while (niter < itrmax)

% STEP I: BASED ON THE PRESENT 'TRIAL' VECTOR l_trial, GENERATE THE
% WEIGHTED ERROR FUNCTION wei_err(k) AT ALL THE GRID POINTS
     x = cos(2*pi*grid(l_trial));  % Step 1: Lagrange abscissa vector x.
     A = x'*ones(1,nz)-ones(nz,1)*x;
     A(eye(nz)==1) = 1;
     ad = prod(A);
     ad = ad * (-2)^(nz-1);  % Step 1: Lagrange coefficient vector ad...
     ad = 1./ad;  % found efficiently without using the function remezdd.
     add = ones(size(ad));
     add(2:2:nz) = -add(2:2:nz);
     dnum = ad*des(l_trial)';
     dden = add*(ad./wt(l_trial))';
     dev = -dnum/dden;  % Step 1: Current value of deviation.
     % Step 2: Lagrange ordinate vector y
     y = des(l_trial) + dev*add./wt(l_trial);
     % Step 3: Overall abscissa vector x_all
     x_all = cos(2*pi*grid(1:ngrid));
     err_num = zeros(1,ngrid);  % Step 4: Initializations of err_num...
     err_den = err_num;  % and err_den.
     for jj = 1:nz  % Steps 5 and 6: Intermediate evaluations for...
         aid = ad(jj)./(x_all - x(jj)); % obtaining the weighted error...
         err_den = err_den + aid;  % wei_err(k) at all the grid points.
         err_num = err_num + y(jj)*aid;
     end
     err_cy = err_num./err_den;
     wei_err = (err_cy - des).*wt; % Step 7: Generate the vector wei_err.
     dev_vect = ones(size(l_trial));  % Steps 8-10: Fill in the undefined
     dev_vect(2:2:length(l_trial))= -dev_vect(2:2:length(l_trial));
     dev_vect = dev_vect * dev;  % entries of wei_err at l_trial(1:nz)...
     % by alternatingly using the values of dev and -dev.
     wei_err(l_trial)=dev_vect;

% STEP II DETERMINE THE VECTOR l_real_start
    % Step 1: Find l_aid1.
    l_aid1  =  find(diff(sign(diff([0  wei_err  0]))));
    % Step 2: Determine l_aid2.
    l_aid2  =  l_aid1(abs(wei_err(l_aid1))  >=  abs(dev));
    [~,ind]  =  max(sparse(1:length(l_aid2),...       % Step 3
        cumsum([1,(wei_err(l_aid2(2:end))>=0)  ...
        ~=(wei_err(l_aid2(1:end-1))>=0)]),...
        abs(wei_err(l_aid2))));
    l_real_start  =  l_aid2(ind);  % Step 4: Determine l_real_start.

% STEP III DETERMINE THE VECTOR l_real
    l_real_init  =  l_real_start;  % Step 1
    if  rem(numel(l_real_init)  -  nz,2)  == 1 % Step 2: odd difference.
     if  abs(wei_err(l_real_init(1)))  <=  abs(wei_err(l_real_init(end)))
            l_real_init(1)  =  [];  % Step 3: discard the first entry...
     else  % of l_real_init.
            l_real_init(end)  =  [];  % otherwise discard the last entry.
     end
    end
    while  numel(l_real_init)  >  nz   % Step 4
        wei_real=abs(wei_err(l_real_init));  % Start of Step 5
        wei_comp=max(wei_real(1:end-1),...
        wei_real(2:end));                                   % End of Step 5
        if  max(abs(wei_err(l_real_init(1))),...       % Start of Step 6
            abs(wei_err(l_real_init(end))))<=  min(wei_comp)
            l_real_init  =  l_real_init(2:end-1);       % End of Step 6
        else
            [~,ind_omit]=min(wei_comp);                 % Start: Step 7
            l_real_init(ind_omit:ind_omit+1)  =  [];     % End: Step 7
        end
    end
    l_real = l_real_init;

% STEP IV: TEST CONVERGENCE
     if (l_real == l_trial)  % Step 1: The real and trial vectors...
         break;   % coincide. Hence, stop. Remez loop ended successfully.
    else
        l_trial = l_real;  % Step 2: Otherwise, replace the values of...
        niter = niter + 1;  % l_trial with the values of l_real and...
    end % continue.
end % END OF THE OVERALL REMEZ LOOP
```

4.3. Performance comparison

This subsection shows how the proposed Implementation II, following the fundamental principle of the RME algorithms, outperforms the original implementation in terms of significant reductions in both the number of iterations and the CPU execution time required to arrive at the same optimum solution. For this purpose, four practical filter design examples are discussed. In all these examples, the problem is to design a filter having five interlaced passbands and stopbands. In order to achieve the accepted behavior in the transition band regions, the last two examples require the use of the *Type A* or *Type B* transition band constraints described in Subsection 3.3.

Example 5: It is desired to design a five-band filter with two passbands and three stopbands meeting the following specifications:

$$\omega_{s1} = 0.17\pi, \omega_{p1} = 0.23\pi, \omega_{p2} = 0.47\pi, \omega_{s2} = 0.53\pi, \omega_{s3} = 0.67\pi, \omega_{p3} = 0.73\pi,$$
$$\omega_{p4} = 0.82\pi, \omega_{s4} = 0.88\pi, \delta_{s1} = \delta_{s2} = \delta_{s3} = 0.001, \text{ and } \delta_{p1} = \delta_{p2} = 0.01.$$

The minimum order to meet the criteria is 91 and the relevant MATLAB commands are

```
>> [n,f,a,w] = firpmord([.17 .23 .47 .53 .67 .73 .82 .88],...
[0 1 0 1 0], [.001 .01 .001 .01 .001]);
>> firr_coeff = firremez_imp2(n+4,f,a,w);
```

The magnitude response of the resulting filter is shown in Fig. 7.

Example 6: It is desired to design a five-band filter with three passbands and two stopbands with following specifications:

$$\omega_{p1} = 0.1\pi, \omega_{s1} = 0.15\pi, \omega_{s2} = 0.3\pi, \omega_{p2} = 0.35\pi, \omega_{p3} = 0.75\pi, \omega_{s3} = 0.8\pi, \omega_{s4} = 0.85\pi,$$
$$\omega_{p4} = 0.9\pi, \delta_{p1} = \delta_{p2} = \delta_{p3} = 0.01, \text{ and } \delta_{s1} = \delta_{s2} = 0.001.$$

The minimum order to meet the criteria is 106 and the relevant MATLAB commands are

```
>> [n,f,a,w] = firpmord([.1 .15 .3 .35 .75 .8 .85 .9],[1 0 1 0 1],...
[.01 .001 .01 .001 .01]); firr_coeff = firremez_imp2(n,f,a,w);
```

The magnitude response of the resulting filter is shown in Fig. 8.

Example 7: It is desired to design a five-band filter with two passbands and three stopbands with following specifications:

$$\omega_{s1} = 0.15\pi, \omega_{p1} = 0.2\pi, \omega_{p2} = 0.45\pi, \omega_{s2} = 0.55\pi, \omega_{s3} = 0.7\pi, \omega_{p3} = 0.8\pi, \omega_{p4} = 0.85\pi,$$
$$\omega_{s4} = 0.93\pi, \delta_{s1} = \delta_{s2} = \delta_{s3} = 0.001, \text{ and } \delta_{p1} = \delta_{p2} = 0.01.$$

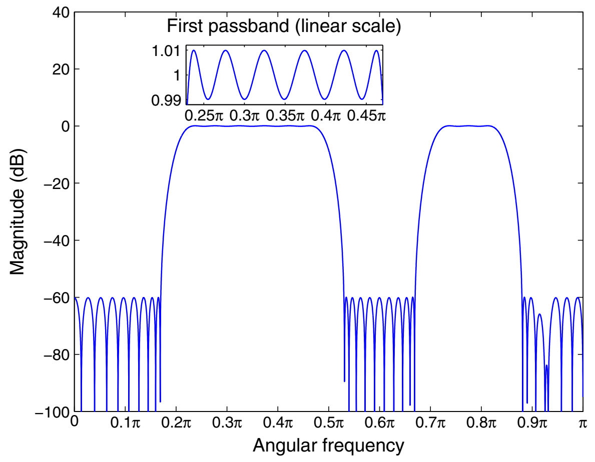

Figure 7. Magnitude response of the five-band filter of Example 5.

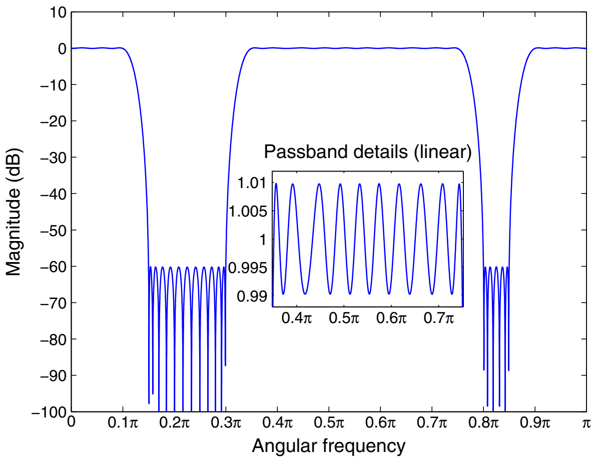

Figure 8. Magnitude response of the five-band filter of Example 6.

The minimum filter order required to meet these specifications is 100. The magnitude response of the resulting filter designed without any constraints in the transition bands is shown by the solid blue line in Fig. 9. It is observed that in the second and third transition bands there are unwanted peaks of approximately 9 dB and 16 dB, respectively. These undesired peaks can be avoided by using *Type A* and *Type B* transition band constraints in the approximation problem according to the discussion of Subsection 3.3. When using $\alpha = 0.0005$, the minimum order to meet the resulting criteria for both *Type A* and *Type B* constraints is 101. The responses of the resulting filters meeting these additional constraints are depicted in Fig. 9 by using a dashed red line and a dot-dashed black line, respectively.

The relevant MATLAB commands for designing the above-mentioned three filters are

```
>> % Filter design without transition band constraints
>> [n,f,a,w] = firpmord([.15 .2 .45 .55 .7 .8 .85 .93],...
[0 1 0 1 0],[.001 .01 .001 .01 .001]);
>> N1 = 101;h = firremez_imp2(n-2,f,a,w);
>> % Filter design with Type A transition band constraints
>> [F1,A1,W1] = convert2constrt([0 .15 .2 .45 .55 .7 .8 .85 .93 1],...
[0 1 0 1 0],[.001 .01 .001 .01 .001],0.0005,1);
>> htbi1 = firremez_imp2(N1,F1,A1,W1);
>> % Filter design with Type B transition band constraints
>> [F2,A2,W2] = convert2constrt([0 .15 .2 .45 .55 .7 .8 .85 .93 1],...
[0 1 0 1 0],[.001 .01 .001 .01 .001],0.0005,2);
>> htbi2 = firremez_imp2(N1,F2,A2,W2);
```

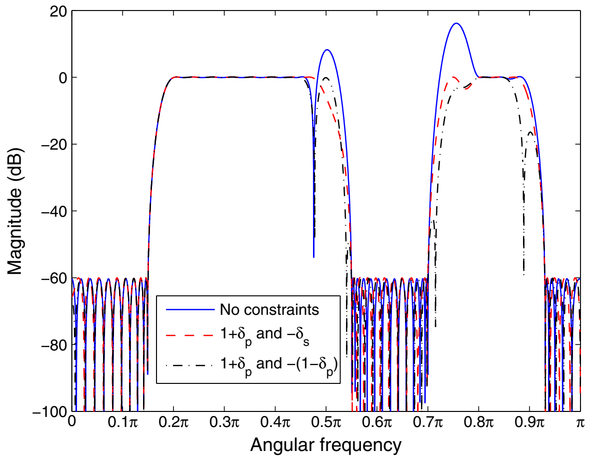

Figure 9. Magnitude responses of the three five-band filters of Example 7.

Example 8: It is desired to design a five bands filter with three passbands and two stopbands with following specifications:

$$\omega_{p1} = 0.17\pi, \omega_{s1} = 0.27\pi, \omega_{s2} = 0.47\pi, \omega_{p2} = 0.52\pi, \omega_{p3} = 0.69\pi, \omega_{s3} = 0.79\pi, \omega_{s4} = 0.87\pi,$$
$$\omega_{p4} = 0.92\pi, \delta_{p1} = \delta_{p2} = \delta_{p3} = 0.01, \text{ and } \delta_{s1} = \delta_{s2} = 0.001.$$

The minimum filter order to meet these specifications is 102. The magnitude response of the resulting filter designed without any constraints in the transition bands is shown by the solid blue line in Fig. 10. It is observed that in the first, second, and third transition bands, there are undesired peaks of approximately 1 dB, 2 dB, and 1.7 dB, respectively. These undesired peaks can be avoided in a manner similar to that used in the previous example. In this example, for *Type A* and *Type B* constraints, the minimum filter orders are 104 and 102, respectively, whereas the responses of the resulting filters are depicted in Fig. 10 using a dashed red line and a dot-dashed black line, respectively.

The relevant MATLAB commands for designing the above-mentioned three filters are

```
>> % Filter design without transition band constraints
>> [n,f,a,w] = firpmord([0.17 0.27 0.47 0.52 0.69 0.79 0.87 0.92],...
[1,0,1,0,1],[0.01,0.001,0.01,0.001,0.01]);
>> h = firremez_imp2(n-4,f,a,w);
>> % Filter design with Type A transition band constraints
>> [F1,A1,W1] = convert2constrt([0 0.17 0.27 0.47 0.52 0.69 0.79,...
0.87 0.92 1], [1,0,1,0,1],[0.01,0.001,0.01,0.001,0.01],0.0005,1);
>> N1 = 104; htbi1 = firremez_imp2(N1,F1,A1,W1);
>> % Filter design with Type B transition band constraints
>> [F2,A2,W2] = convert2constrt([0 0.17 0.27 0.47 0.52 0.69 0.79,...
0.87 0.92 1], [1,0,1,0,1],[0.01,0.001,0.01,0.001,0.01],0.0005,2);
>> N2 = 102; htbi2 = firremez_imp2(N2,F2,A2,W2);
```

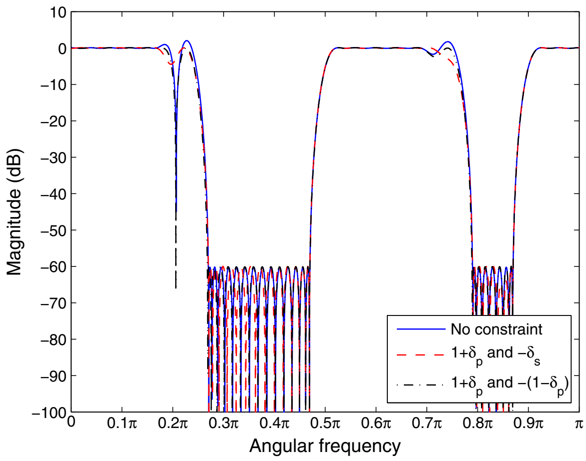

Figure 10. Magnitude response of three five-band filters of Example 8.

Example	Original Implementation		Implementation II		Reduction Percentage(%)	
	Iterations	Time*	Iterations	Time	Iterations	Time
1	34	0.223	7	0.024	79	89
2	35	0.269	16	0.040	54	85
$3^{\top}$	15	0.160	15	0.033	–	79
$3^{\perp A}$	93	0.617	23	0.055	75	91
$3^{\perp B}$	38	0.283	20	0.051	47	82
$4^{\top}$	11	0.146	11	0.029	–	80
$4^{\perp A}$	49	0.360	37	0.082	24	77
$4^{\perp B}$	66	0.398	23	0.057	65	86

*Time in seconds
$^{\top}$Transition bands are excluded.
$^{\perp A}$*Type A* transition band constraints are used.
$^{\perp B}$*Type B* transition band constraints are used.

Table 2. Performance Comparison of Original Implementation and Implementation II.

The number of iterations as well as the CPU execution times required by the original implementation and the proposed second implementation are summarized in Table 2 for synthesizing all the eight filters considered in this section. The hardware and MATLAB versions are the same as used earlier during the comparison of the original and the proposed first implementations. It is seen that the reduction in the numbers of iterations is 79 and 54 percent, respectively, when synthesizing the filters in Example 5 and 6. In case of examples 7 and 8 with the transition band constraints in effect, the reduction in the numbers of iterations is 45 percent (*Type A*), 38 percent (*Type B*) and 18 percent (*Type A*), and 55 percent (*Type B*), respectively. The reduction percentage in the CPU execution time is between 71 and 86 percent for all the eight filter designs under consideration. Hence, the proposed second implementation is highly efficient in the design of multiband filters.

5. Conclusion

This chapter has introduced two novel MATLAB based Remez algorithms for the design of optimum arbitrary-magnitude linear-phase FIR filters. The first algorithm is a highly optimized and compact translation of the PM algorithm from its original FORTRAN code to its MATLAB counterpart in comparison with the existing MATLAB function **`firpm`**. These attractive properties have been achieved by first observing that the PM algorithm's very lengthy search strategy for the "real" extremal points of the weighted error function, which is formed based on the "trial" extremal points, can be compressed into two very compact basic search techniques. Second, the MATLAB vectorization techniques are employed whenever possible. As a result, the CPU execution time is roughly one third to synthesize linear-phase FIR filters practically in the same manner in comparison with the function **`firpm`** being more or less a direct translation from the FORTRAN code. Moreover, the code complexity is reduced to a considerable extent. The original implementation utilizes approximately 15 nested loops and around 300 lines of code whereas the first proposed implementation uses only 3 looping structures and approximately 100 lines of code. Thus, same efficient results are achieved with one fifth of the looping structures and one third of the code lines in the original implementation.

It is, however, important to note that the first technique does not follow the fundamental idea of Remez algorithm as suggested in [20] as it tries to find the new "trial" extremal points in the vicinity of previously found points as well as in the surroundings of the first and last grid points under consideration.

The second implementation obeys the fundamental principle of the Remez multiple exchange algorithm. This means that while searching for the "real" set of extrema, there is a possibility to obtain more than the required points in intermediate calculations. In this situation, the idea is to keep as many extremal points as possible subject to the condition that the corresponding error values are the maximum absolute ones and they obey the sign alternation property. Another prominent feature is that the weighted error function is calculated over the entire grid. This, not only makes sure that no potential extremal frequency point is skipped during a particular iteration, but also enables to transfer the two extremal points between the consecutive bands which, in some cases, is a necessary prerequisite for the algorithms in [16] and [19] to converge. Furthermore, the number of iterations as well as the CPU execution times required by the proposed second implementation to design the linear-phase FIR filters in comparison with the existing MATLAB function **`firpm`**, especially in multi-band cases, are significantly lower. Examples have shown that in most five-band cases with some constraints in the transition bands, the reduction in the number of iteration is more than **50** percent, whereas the reduction in the CPU execution time is around **80** percent.

The quality of the filters designed with the proposed implementations is analogous to that of the PM algorithm with the added advantages of less number of iterations and CPU execution time.

The proposed two implementations have concentrated only on the core discrete Remez part of the PM algorithm. Future work is devoted to explore the possibilities of further improvements in the overall function **`firpm`** and reimplementing the other portions of this function efficiently.

Author details

Muhammad Ahsan and Tapio Saramäki
Tampere University of Technology, Tampere, Finland

6. References

[1] Ahsan, M. (2008). Design of optimum linear phase FIR filters with a modified implementation of the Remez multiple exchange algorithm, In:*Department of Signal Processing, Tampere University of Technology*, Master's thesis, (Sep. 2008), Tampere, Finland, 107 pages.
[2] Ahsan, M. & Saramäki, T. (2009). Significant improvements in translating the Parks-McClellan algorithm from its FORTRAN code to its corresponding MATLAB code, In:*IEEE Symp. Circuits Syst.*, (May 2009), Taipei, Taiwan, pp. 289-292.
[3] Ahsan, M. & Saramäki, T. (2011). A MATLAB Based Optimum Multiband FIR Filters Design Program Following the Original Idea of the Remez Multiple Exchange Algorithm, In:*IEEE Symp. Circuits Syst.*, (May 2011), Rio de Janiero Brazil, pp 137-140.
[4] Cheney, W. E. (1966). *Introduction to Approximation Theory*, AMS Chalsea Publishing.
[5] McClellan, J. H. & Parks, T. W. (1972). A program for the design of linear phase finite impulse response digital filters, In: *IEEE Trans. Audio Electroacoust.*, Vol. AU-20, (Aug. 1972) pp. 195-199.
[6] McClellan, J. H. & Parks, T. W. (1973). A unified approach to the Design of Optimum FIR linear phase digital filters, In: *IEEE Trans. Circuit Theory*, Vol. 20, (Nov. 1973) pp. 697-701.
[7] McClellan, J. H. & Parks, T. W. & Rabiner, L. R. (1973). A computer program for designing optimum FIR linear phase digital filters, In: *IEEE Trans. Audio Electroacoust.*, Vol. 21, (Dec. 1973) pp. 506-526.
[8] Novodvorskii, E. P. & Pinsker, I. S. (1951). The process of equating maxima, In: *Uspekhi Mat. Nauk (USSR)*, Vol. 6, (1951) pp. 174-181 (Engl. transl. by A. Schenitzer).
[9] Rabiner, L. R., Kaiser, J. F. & Schafer, R. W. (1974). Some considerations in the design of multiband finite-impulse-response digital filters, In: *IEEE Trans. Acoust. Speech, Signal Processing*, Vol. 22, (Dec. 1974) pp. 462-472.
[10] Rabiner, R. L., Gold, G. (1975). *Theory and Application of DIGITAL SIGNAL PROCESSING*, Prentice Hall.

[11] Remez, E. (1934). Sur le calcul effectifdes polynomes d'approximation de Tchebychef, In: *Compt. Rend. Acad. Sci*, Vol. 199, (1934) pp. 337-340.

[12] Rice, R. J. (1964). *The Approximation of Functions. Volume 1: Linear Theory*, Addison-Wesley Pub (Sd).

[13] Rivlin, J. T. (2010). *An Introduction to the Approximation of Functions*, Dover Publications.

[14] Saramäki, T. (1981). Design of digital filters requiring a small number of arithmetic operations, In:*Dept. of Electrical Engineering, Tampere University of Technology*, Dr. Tech. Dissertation, Publ. 12, Tampere, Finland, 1981,226 pages.

[15] Saramäki, T. (1987). Efficient iterative algorithms for the design of optimum IIR filters with arbitrary specifications, In:*Proc. Int. Conf. Digital Signal Process.*, Florence,Italy, (Sep. 1987) pp. 32-36.

[16] Saramäki, T. (1992). An efficient Remez-type algorithm for the design of optimum IIR filters with arbitrary partially constrained specifications, In:*IEEE Symp. Circuits Syst.*, Vol. 5, (May 1992) San Diego CA, pp. 2577-2580.

[17] Saramäki, T. (1993). Finite impulse response filter design, In:*Handbook for Digital Signal Processing*, Mitra, S. K. & Kaiser, J. F. , (Eds.), Ch. 4, (1993) New York, NY:John Wiley and Sons, pp. 155-277.

[18] Saramäki, T. (1994). Generalizations of classical recursive digital filters and their design with the aid of a Remez-type Algorithm, In:*IEEE Symp. Circuits Syst.*, Vol. 2, (May 1994), London UK, pp. 549-552.

[19] Saramäki, T. & Renfors, M. (1995). A Remez-type algorithm for designing digital filters composed of all-pass sections based on phase approximations, In:*Proc. 38th Midwest Symp. Circuits Syst.*, Vol. 1, (Aug. 1995), Rio de Janiero Brazil, pp. 571-575.

[20] Temes, G. C. & Calahan, D. A. (1967). Computer-aided network optimization the state-of-the-art, In:*Proc. IEEE*, Vol. 55, (Nov. 1967), pp. 1832-1863. Nov. 1967.

[21] The MathWorks Inc. (2009). Filter design toolbox user's guide, In:*MATLAB Product Help*, Version 4.6, (Sep. 2009), The MathWorks Inc., Natick, MA.