"A MATLAB based optimum multiband FIR filters design program following the original idea of the Remez multiple exchange algorithm"

Conference Paper · May 2011

DOI: 10.1109/ISCAS.2011.5937520 · Source: DBLP

CITATIONS
6

READS
642

2 authors:

Muhammad Ahsan
STAIN Parepare
5 PUBLICATIONS **15** CITATIONS

SEE PROFILE

Tapio Saramaki
Tampere University
331 PUBLICATIONS **5,043** CITATIONS

SEE PROFILE

"A MATLAB Based Optimum Multiband FIR Filters Design Program Following the Original Idea of the Remez Multiple Exchange Algorithm"

Muhammad Ahsan and Tapio Saramäki
Department of Signal Processing, Tampere University of Technology
P.O.Box 553, FI-33101 Tampere, Finland
Email: muhammad.ahsan@tut.fi and ts@cs.tut.fi

***Abstract*— A highly optimized translation of the core discrete Remez part of the Parks-McClellan (PM) algorithm from its original FORTRAN code to its MATLAB counterpart has recently been proposed by the authors. The optimization was achieved by first figuring out that the search for the "real" extremal points of the weighted error function formed based on the "trial" extremal points can be compressed into two compact search techniques and, second, by using the MATLAB strength of vectors and matrices calculations whenever possible. Most importantly, this achievement revealed that the search technique in the original PM algorithms does not follow the fundamental principle of the Remez multiple exchange (RME) algorithm. That is, if there are more candidate "real" extremal points than required, then the desired points should be selected to retain as many largest absolute values of the weighted error function as possible subject the condition that the sign of this function alternates at the consecutive points. This paper modifies the earlier MATLAB implementation of the core discrete Remez part of PM algorithm to exactly follow the above-mentioned search principle. This modification results in a highly optimized MATLAB code which outperforms the very original MATLAB code in, terms of the code compactness, the required number of iterations and CPU execution time, as is illustrated by means of several examples.**

I. Introduction

Parks-McClellan (PM) algorithm is one of the best known methods for the design of optimum magnitude finite-impulse response (FIR) digital filters of all the four types with arbitrary specifications. Park and McClellan proposed the method in [1] for the design of conventional low-pass linear phase FIR filters with symmetric impulse response and even order. Later on, together with Rabiner, they extended the original design technique to cover all the four cases of linear phase FIR filters [2].

The PM algorithm was generated using FORTRAN at the beginning of 1970. The limited computer resources at that time gave rise to two main doubts about the PM algorithm. The first doubt was that the algorithm is not properly constructed as it failed to provide the optimum results especially for filters with very high orders and/or many pass-band and stop-band regions. It was observed that the maximum number of iterations in the FORTRAN implementation was set to very low value of 25. This value was not the optimum one and as such the program did not give any explicit warning about this fact. Thus it became clear that the first doubt about the algorithm is superfluous and by increasing the number of iterations tenfold, the FORTRAN code allowed the design of filters with very high orders and several pass-bands/stop-bands regions. This fact has efficiently been utilized by the MATLAB code which allows the algorithm to iterate 250 times and provides convergence for the filters up to the order of ten thousands and with several pass-band/stop-band regions.

This work was supported by the Academy of Finland, project No. 213462 (Finnish Centre of Excellence program (2006 - 2011))

The second doubt originated from the procedure utilized by the PM algorithm to search for the "real" extremal points of the weighted error function which is formed on the basis of "trial" extremal points. The search strategy included in the PM algorithm did not work as expected when it has been employed in various Remez-type algorithms for designing the recursive digital filters [4]−[8]. Some of these algorithms have been turned out to be very sensitive to the selection of the initial set of "trial" extremal points and have had problems in converging at the optimal solution. For the algorithms described in [5] and [7], the convergence problems have been solved by increasing the maximum number of iterations in the above-mentioned manner. However, the algorithms described in [6] and [8] are still sensitive to the selection 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 performed in the core discrete Remez part of the FORTRAN implementation of the PM algorithm. After a thorough study of the FORTRAN code, it was observed that the search for the "true" extremal points can be compressed into two compact search techniques known as *Vicinity Search* and *Endpoint Search*. Based on these search techniques, a modified implementation of the core discrete Remez algorithm part of PM algorithm is presented in [11] and is analogous to the one used in PM algorithm. It is observed that while employing this search strategy in Remez-type algorithms proposed in [6] and [8], it is not possible to transfer the two extremal points from a pass-band to a stop-band or vice versa which is a necessary requirement for the algorithms to guarantee the convergence. Most importantly, the search technique included in the PM algorithm does not follow the fundamental notion of the Remez multiple exchange algorithm when the approximation interval is a union of several disjoint intervals. That is, if there are more candidate "real" extremal points than required, then the desired points should be selected to retain as many largest absolute values of the weighted error as allowed subject to the condition that the sign of the weighted error function alternates at the successive points [9].

The main purpose of this study of the PM algorithm was to obtain an optimized translation of the core discrete Remez part from its original FORTRAN code to its MATLAB counterpart in such a way that the search technique follows the fundamental principle of the Remez multiple exchange (RME) algorithm so that it can be utilized to make the algorithms in [6] and [8] work properly and efficiently. A surprised side-effect of this effort is the fact that the modified MATLAB code is also superior to the start-up MATLAB code in all aspects and is now the topic of this article.

The organization of this article is as follows. Section II formally states the approximation problem under consideration. Section III provides a detailed description of how to modify the core discrete Remez part of the overall PM algorithm such

that, not only it exactly follows the fundamental principle of the RME algorithm but also is optimized to the best possible extent in the form of a MATLAB code. The resulting code snippet is provided in Section IV. By means of several multiband FIR filter design examples, Section V illustrates how this code snippet outperforms the very original code included in the MATLAB function **`firpm`**. The performance measurements are carried out on the basis of the number of iterations and the CPU execution time taken by the two code snippets. Finally, concluding remarks are given in section VI.

II. 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 contain values between 0 and 0.5[2], along with the vectors **des** and **wt** of the same length, the elements of which carry the information of the desired and weight values, respectively, at the corresponding grid points, find the unknowns $a[n]$ to minimize the following quantity:

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

where

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

and

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

According to *Alternation (Characterization) Theorem* [10] $G(k)$ of the form of (1c) is the best unique solution minimizing ε as given by (1a) and (1b) if and only if there exists an optimal vector $l_{\mathbf{opt}}$ that contains (at least) nz indices $l_{\mathbf{opt}}(1), l_{\mathbf{opt}}(2), \ldots, l_{\mathbf{opt}}(nz)$ for the grid points included in the vector **grid** such that the following three condition are met:
1) $l_{\mathbf{opt}}(1) < l_{\mathbf{opt}}(2) < \ldots < l_{\mathbf{opt}}(nz-1) < l_{\mathbf{opt}}(nz)$.
2) $E[l_{\mathbf{opt}}(m+1)] = -E[l_{\mathbf{opt}}(m)]$ for $m = 1, 2, \ldots, nz-1$.
3) $|E[l_{\mathbf{opt}}(m)]| = \varepsilon$ for $m = 1, 2, \ldots, nz$.

III. Explanation of the Proposed Core RME Algorithm

This section describes how to modify the core Remez algorithm part of the PM algorithm in such a way that not only it follows the fundamental notion of the RME algorithm but also becomes efficiently implementable as a MATLAB code. The efficiency criteria are the CPU execution time, the overall code compactness, and the required number of iterations.

INITIALIZATIONS: After performing the initializations as suggested in ***STEP I*** in [11], the vector $l_{\mathbf{opt}}$, whose elements satisfy the three conditions of the previous section is iteratively sought for with the help of the following algorithm[3]:

STEP I: The first step is to obtain the weighted error vector **wei_err** concurrently over all the grid points included in the vector **grid** subject to the conditions regarding the *nz* elements included in $l_{\mathbf{trial}}$. At these indices, the following system of *nz* linear equations

$$\begin{aligned} E[l_{\mathbf{trial}}(m)] &= \mathbf{wt}[l_{\mathbf{trial}(m)}][G(l_{\mathbf{trial}}(m)) - \mathbf{des}(l_{\mathbf{trial}}(m))] \\ &= (-1)^{m+1} dev \text{ for } m = 1, 2, \ldots, nz \end{aligned} \tag{2}$$

should be satisfied by the $nz-1$ unknowns involved in $G(k)$ and the remaining unknown *dev*. For this purpose, there are certain similarities between the proposed implementation and its corresponding part in MATLAB function **`firpm`** which is more or less a direct translation of the FORTRAN implementation of the PM algorithm. As in function **`firpm`**, the value of *dev* required by the solution to the system of *nz* equations in (2) is determined. After that the abscissa vector **x**, the ordinate vector **y**, and the additional coefficient vector **ad**, each of which is of length *nz*, are determined such that the frequency response corresponding to $G(k)$ can be expressed using the very accurate Lagrange interpolation formula in barycentric form and the system of *nz* equations in (2) is satisfied. Compared to the function **`firpm`**, the crucial difference is the utilization of vectors wherever possible in the MATLAB code to be described in Section IV. According to the MATLAB code description in [11], this main step can be efficiently carried out by using the following eight steps:

Step 1: Determine the elements of the vectors **x** and **ad**

$$\mathbf{x}(m) = \cos\left[2\pi \cdot l_{\mathbf{trial}}(m)\right] \text{ for } m = 1, 2, \ldots, nz \tag{3}$$

and

$$\text{ad}(m) = 1/\prod_{\substack{k=1 \\ k \ne m}}^{nz} \left[\mathbf{x}(m) - \mathbf{x}(k)\right] \text{ for } m = 1, 2, \ldots, nz, \tag{4}$$

respectively, as well as the corresponding deviation value as

$$dev = -\frac{\sum_{m=1}^{nz} \mathbf{ad}(m)\mathbf{des}[l_{\mathbf{trial}}(m)]}{\sum_{m=1}^{nz} (-1)^{m-1}\mathbf{ad}(m)\mathbf{wt}[l_{\mathbf{trial}}(m)]}. \tag{5}$$

Step 2: Determine the elements of the vector **y** for $m = 1, 2, \ldots, nz$ as

$$\mathbf{y}(m) = \mathbf{des}\left[l_{\mathbf{trial}}(m)\right] + (-1)^{m-1}\mathbf{ad}\left[l_{\mathbf{trial}}(m)\right]/\mathbf{wt}\left[l_{\mathbf{trial}}(m)\right]. \tag{6}$$

Step 3: Generate the elements of the abscissa vector **x_all** by converting all the elements in the vector **grid** as

$$\mathbf{x\_all}(k) = \cos\left[2\pi \cdot \mathbf{grid}(k)\right] \text{ for } k = 1, 2, \ldots, ngrid. \tag{7}$$

Step 4: Reset all the elements of the vectors **err_num** and **err_den** of length $ngrid$ to zero, set $m = 1$, and go to next step.

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

$$\mathbf{aid}(k) = \mathbf{ad}(m)\left[\mathbf{x\_all}(k) - \mathbf{x}(m)\right] \text{ for } k = 1, 2, \ldots, ngrid \tag{8}$$

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

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

Step 7: Generate the elements of the vector **wei_err** for $k = 1, 2, \ldots, ngrid$ as,

$$\mathbf{wei\_err}(k) = \left[\mathbf{err\_num}(k)/\mathbf{err\_den}(k) - \mathbf{des}(k)\right]\mathbf{wt}(k). \tag{9}$$

[1] In this contribution, $nz-1$ represents the number of adjustable parameters both in FORTRAN and the MATLAB implementations of the PM algorithm, nz stands for the number of extrema at which *Alternation Theorem*[10] 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[10]

[3] For the sake of simplicity, the algorithm is decomposed into four main steps(bold face) and several sub steps(italic face) such that the role of the main steps is to deliver the core concepts and the role of the sub steps succeeding each main step is to provide the additional information about that particular step.

Step 8: The resulting **wei_err** contains undefined values at the indices of $l_{\mathbf{trial}}$ due to the use of the Lagrange interpolation formula in barycentric form. Fill in the undefined values by utilizing the Steps 10−12 succeeding the ***STEP II*** in [11] and go to ***STEP II***.

STEP II: Based on the values of $\mathbf{wei\_err}(k)$ for $1 \leq k \leq ngrid$ generated at ***STEP I***, this main step generates the vector $l_{\mathbf{real}}^{(\mathbf{start})}$ to be utilized at ***STEP III*** in such a way that it includes as many indices as allowed in the ascending order subject to the following conditions:

1) At each index of $l_{\mathbf{real}}^{(\mathbf{start})}$, the vector **wei_err** achieves a local extremum so that the absolute value is larger than or equal to $|dev|$, where *dev* is determined according to 5.

2) In cases of several consecutive local extrema of **wei_err** with the same sign, only the one with maximum absolute value is selected.

3) The sign of **wei_err** alternates at the consecutive resulting indices in $l_{\mathbf{real}}^{(\mathbf{start})}$.

This main step is carried out with the following three steps:

Step 1: Find all the indices in the range $1 \leq k \leq ngrid$ where a local extremum of $\mathbf{wei\_err}(k)$ occurs. Store all these indices in the ascending order in vector $l_{\mathbf{aid1}}$ in the ascending order.

Step 2: Obtain those indices of the vector $l_{\mathbf{aid1}}(m)$ for $m = 1, 2, \ldots, n_{\mathbf{aid1}}$ at which $|\mathbf{wei\_err}(l_{\mathbf{aid1}}(m))| \geq |dev|$ and store them in the vector $l_{\mathbf{aid2}}$.

Step 3: Split the interval $1 \leq k \leq n_{\mathbf{aid2}}$, where $n_{\mathbf{aid2}}$ is the length of $l_{\mathbf{aid2}}$, into subintervals $k_{low}(m) \leq k \leq k_{up}(m)$ for $m = 1, 2, \ldots, nz_{\mathbf{start}}$ so that in the mth subinterval the signs of $\mathbf{wei\_err}(l_{\mathbf{aid2}}(k))$ are the same. Generate the desired vector $l_{\mathbf{real}}^{(\mathbf{start})}$ of length $nz_{\mathbf{start}}$ such that its mth element has value of $l_{\mathbf{aid2}}(k)$ for $k_{low}(m) \leq k \leq k_{up}(m)$ at which $|\mathbf{wei\_err}(l_{\mathbf{aid2}}(k))|$ achieves the maximum value.

STEP III: This main step is required only when the length of $l_{\mathbf{real}}^{(\mathbf{start})}$ generated at ***STEP II*** is greater than *nz*. Under this circumstance, this step generates the index vector that contains only the nz indices of the above start-up vector such that it retains as many largest absolute values of the vector **wei_err** as possible subject to the condition that the sign of this vector alternates at the remaining consecutive indices according to [9]. After some reasoning, it has turned out that a very straightforward way to achieve the desired index vector is as follows: First, if length of $l_{\mathbf{real}}^{(\mathbf{start})}$ differs from nz by an odd integer, then discard the first or the last index of $l_{\mathbf{real}}^{(\mathbf{start})}$ if the absolute value of **wei_err** is smaller at the first index or at the last one, repectively. Second, if the length of the remaining $l_{\mathbf{real}}^{(\mathbf{start})}$, denoted by $\tilde{l}_{\mathbf{real}}^{(\mathbf{start})}$ is still greater than nz, then its length is gradually decreased by two until it becomes equal to nz by using the following process:

1) For the vector $\tilde{l}_{\mathbf{real}}^{(\mathbf{start})}$ of length $\tilde{n}z_{\mathbf{real}}^{(\mathbf{start})}$, the vector **wei_comp** of length $\tilde{n}z_{\mathbf{real}}^{(\mathbf{start})} - 1$ is generated such that its mth element is the maximum of the values of $|\mathbf{wei\_comp}(\tilde{l}_{\mathbf{real}}^{(\mathbf{start})}(m))|$ and $|\mathbf{wei\_comp}(\tilde{l}_{\mathbf{real}}^{(\mathbf{start})}(m+1))|$.

2) If the maximum of the absolute values of **wei_err** at the first and last elements of $\tilde{l}_{\mathbf{real}}^{(\mathbf{start})}$ is smaller than the minimum of the elements in the vector **wei_comp**, then both the first and last elements of $\tilde{l}_{\mathbf{real}}^{(\mathbf{start})}$ are discarded. Otherwise, the index of **wei_comp** at which the smallest value occurs is located. If the smallest value occurs at the index m, then the mth and $(m+1)$th of $\tilde{l}_{\mathbf{real}}^{(\mathbf{start})}$ are disposed of.

This thrid main step can be carried out using the following seven steps:

Step 1: Set $l_{\mathbf{real}}^{(\mathbf{init})} = l_{\mathbf{real}}^{(\mathbf{start})}$. If $nz_{\mathbf{real}}^{(\mathbf{init})} - nz$, where $nz_{\mathbf{real}}^{(\mathbf{init})}$ is the length of $l_{\mathbf{real}}^{(\mathbf{init})}$, is zero, then set $l_{\mathbf{real}} = l_{\mathbf{real}}^{(\mathbf{init})}$ and go to ***STEP IV***. Otherwise, go to the next step.

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

Step 3: If $|\mathbf{wei\_err}(l_{\mathbf{real}}^{(\mathbf{init})}(1))| \leq |\mathbf{wei\_err}(l_{\mathbf{real}}^{(\mathbf{init})}(nz))|$, then discard the first index of $l_{\mathbf{real}}^{(\mathbf{init})}$. Otherwise, discard the last index of $l_{\mathbf{real}}^{(\mathbf{init})}$. Go to the next step.

Step 4: If $nz_{\mathbf{real}}^{(\mathbf{init})} - nz$, where $nz_{\mathbf{real}}^{(\mathbf{init})}$ is the length of $l_{\mathbf{real}}^{(\mathbf{init})}$, is zero, then set $l_{\mathbf{real}} = l_{\mathbf{real}}^{(\mathbf{init})}$ and go to ***STEP IV***. Otherwise, go to the next step.

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

$$\mathbf{wei\_comp}(m) = max(|\mathbf{wei\_err}(l_{\mathbf{real}}^{(\mathbf{init})}(m))|, |\mathbf{wei\_err}(l_{\mathbf{real}}^{(\mathbf{init})}(m+1))|) \text{ for } m = 1, 2, \ldots, nz_{\mathbf{real}}^{(\mathbf{init})} - 1. \tag{10}$$

Step 6: If $max(|\mathbf{wei\_err}(l_{\mathbf{real}}^{(\mathbf{init})}(1))|, |\mathbf{wei\_err}(l_{\mathbf{real}}^{(\mathbf{init})}(nz_{\mathbf{real}}^{(\mathbf{init})}))|)$ is less than or equal to the largest value in the vector **wei_comp**, then remove the first and last index value from $l_{\mathbf{real}}^{(\mathbf{init})}$ and go to Step 4. Otherwise, go to the next step.

Step 7: Find the mth index of the smallest value in the vector **wei_comp** and remove the corresponding value in the vector $l_{\mathbf{real}}^{(\mathbf{init})}(m)$ along with the value next to it that is $l_{\mathbf{real}}^{(\mathbf{init})}(m+1)$. Go to Step 4.

STEP IV: As suggested in [11], [12], this final main step verifies the convergence of the overall algorithm within the allowed number of 250 iterations.

IV. Code Snippet of the Proposed RME Implementation

An efficient MATLAB code snippet for the implementation of the modified core part of the PM algorithm described in the previous section is shown below:

```
function [x,y,ad,dev]=remez_core(nz,iext,ngrid,grid,des,wt)
% This function efficiently implements the STEPS I - IV
% in the algorithm described in Section III by exploiting
% MATLAB built-in vector and matrix operations.

itrmax = 250;              % START-UP: THREE INITIALIZATIONS
niter = 1;
l_trial = iext(1:nz)';
while (niter < itrmax)             % START OF PRIMARY LOOP
% MAIN STEP I
  x = cos(2*pi*grid(l_trial));     % Step 1;abscissa vector x
  A = x'*ones(1,nz)-ones(nz,1)*x; % get coefficient vector
  A(eye(nz)==1) = 1;               % ad
  ad = prod(A);
  ad = ad * (-2)^(nz-1);
  ad = 1./ad;                      % vector ad ready
  add = ones(size(ad));            % determine current value
  add(2:2:nz) = -add(2:2:nz);      % value of dev
  dnum = ad*des(l_trial)';         % determine ordinate vector
  dden = add*(ad./wt(l_trial))';   % y at Step 2
  dev = -dnum/dden;                % value of dev ready
  y = des(l_trial) + dev*add./wt(l_trial); % vector y ready
  % Step 3;abscissa vector x_all
  x_all = cos(2*pi*grid(1:ngrid));
  err_num = zeros(1,ngrid);        % Step 4
  err_den = err_num;
  for j = 1:nz                     % Step 5
      aid = ad(j)./( x_all - x(j));
      err_den = err_den + aid;
      err_num = err_num + y(j)*aid;
  end
  wei_err_cy = err_num./err_den;
  wei_err = (wei_err_cy - des).*wt; % Step 7
% Fill in the undefined values in wei_err using Steps 8 - 10
% in [11]
  dev_vect = ones(size(l_trial));
  dev_vect(2:2:length(l_trial))=- ...
  dev_vect(2:2:length(l_trial));
  dev_vect = dev_vect * dev;
  wei_err(l_trial) = dev_vect;
% MAIN STEP II
  l_aid1 = find(diff(sign(diff([0 wei_err 0])))~=0);% Step 1
  l_aid2 = l_aid1(abs(wei_err(l_aid1)) >= abs(dev));% Step 2
  [~,ind] = max(sparse(1:length(l_aid2),...   % Start 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);                    % End Step 3
% MAIN STEP III
  l_real = l_real_start;              % Step 1
  if rem(numel(l_real) - nz,2) ==1    % Step 3
       if abs(wei_err(l_real(1))) <= ...
          abs(wei_err(l_real(end)))
           l_real(1) = [];
       else
           l_real(end) = [];
       end                            % End Step 4
    end
    while numel(l_real) > nz
       wei_real=abs(wei_err(l_real));       % Start Step 5
       wei_comp=max(wei_real(1:end-1),...
       wei_real(2:end));                    % End Step 5
       if max(abs(wei_err(l_real(1))),...
          abs(wei_err(l_real(end)))<= min(wei_comp)
           l_real = l_real(2:end-1);        % End Step 6
       else
           [~,ind_omit]=min(wei_comp);
           l_real(ind_omit:ind_omit+1) = [];% End Step 7
       end
    end
  % MAIN STEP IV
  if (l_real == l_trial)% l_real and l_trial are the same.
    niter = niter + 1;
    break;               % Therefore, stop
  else
    l_trial = l_real;  % Otherwise, set l_trial = l_real and
    niter = niter + 1; % continue until l_real and l_trial
  end                  % coincide or niter > itrmax.
end                              % End of the main loop
fprintf('niter is %d.\ n',niter);
```

V. Performance Comparison

In order to verify the efficiency of the proposed method, a performance comparison has been carried out with the code snippet available in MATLAB function **`firpm`**. Both the code snippets are used to design the four multiband FIR filters. The measurement criteria are the number of iterations, and, the execution time taken by them. The filters under consideration are the following:

Example 1: The specifications for a band-pass filter are

$$\omega_{s1} = 0.2\pi, \omega_{p1} = 0.25\pi, \omega_{p2} = 0.6\pi, \omega_{s2} = 0.65\pi$$
$$\delta_{s1} = 0.001, \delta_p = \delta_{s2} = 0.01.$$

The minimum filter order to meet these criteria is 103.

Example 2: The specifications for a band-stop filter are

$$\omega_{p1} = 0.2\pi, \omega_{s1} = 0.35\pi, \omega_{s2} = 0.7\pi, \omega_{p2} = 0.85\pi$$
$$\delta_{p1} = \delta_{p2} = 0.01, \delta_s = 0.001.$$

The minimum order to meet these criteria is 38.

Example 3: The specifications for a five-band filter with three stop-bands and two pass-bands are

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

The minimum filter order to meet these criteria is 91.

Example 4: The specifications for a five-band filter with three pass-bands and two stop-bands are

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

The minimum order to meet these criteria is 106.

Table I indicates the outcomes obtained from both implementations and suggests that the proposed method is quite efficient for designing the multiband filters. The difference in number of iterations is quite evident when designing band-pass and band-stop filters as well as filters having more than three bands. CPU execution time further complements the results by signifying the smaller execution time taken by the proposed method. These measurements were taken on an IBM ThinkCentre machine equipped with intel C2D processor E6550 running at a speed of 2.33GHz and having a 3GB of memory by using MATLAB version 7.11.0.584 (R2010b).

TABLE I
Performance Comparison

	Original Method		Proposed Method	
	Iterations	Time*	Iterations	Time*
Example 1	15	0.166	9	0.03
Example 2	9	0.108	7	0.03
Example 3	34	0.228	18	0.04
Example 4	35	0.275	26	0.06

* Time in seconds

VI. Conclusion

This paper proposed a MATLAB based implementation of the core discrete RME algorithm part in the PM algorithm. The search technique for the "real" extremal points in this implementation obeys the fundamental notion of the RME algorithm as suggested in [9]. 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 "indices" as allowed subject to the condition that the corresponding error values are the maximum absolute ones and they obey the sign alternation property. This not only makes sure that no potential extremal frequency point is skipped during a particular iteration but also allows to transfer the two extremal points between the two bands which is a necessary prerequisite for the algorithms in [6] and [8] to converge. The quality of the filters designed with the proposed method is analogous to that of the PM algorithm with an added advantage of less number of iterations and execution time.

References

[1] J. H. McClellan, T. W. Parks, "A program for the design of linear phase finite impulse response digital filters," *IEEE Trans. Audio Electroacoust.*, vol. AU-20, pp. 195–199, Aug. 1972.

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

[3] *Filter Design Toolbox User's Guide*, Version 4.6, The MathWorks Inc., Natick, MA, 2009.

[4] T. Saramäki, "Design of digital filters requiring a small number of arithmatic operations," in *Dr. Tech. Dissertation*, Dept. of Electrical Engineering, Tampere University of Technology, Tampere, Finland, Publ. 12, 1981,226 pages.

[5] T. Saramäki, "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.

[6] T. Saramäki, "An efficient Remez-type algorithm for the design of optimum IIR filters with arbitrary partially constrained specifications," in *IEEE Symps. Circuits Syst.*, San Diego,CA, May 1992, vol. 5 pp. 2577–2580.

[7] T. Saramäki, "Generalizations of classical recursive digital filters and their design with the aid of a Remez-type Algorithm," in *IEEE Symps. Circuits Syst.*, London,UK, May 1994, vol. 2 pp. 549–552.

[8] T. Saramäki and M. Renfors, "A Remez-tpe algorithm for designing digital filters composed of all-pass sections based on phase approximations," in *Proc. 38th Midwest Symp. Circuits Syst.*, Rio de Janiero,Brazil, Aug. 1995, vol. 1 pp. 571–575.

[9] G. C. Temes and D. A. Calahan, "Computer-aided network optimization the state-of-the-art," *Proc. IEEE*, vol. 55, pp. 1832–1863, Nov. 1967.

[10] T. Saramäki, *Handbook for Digital Signal Processing*. S. K. Mitra and J. F. Kaiser, Eds. New York, NY:John Wiley and Sons, 1993, ch. 4, pp. 155–277.

[11] M. Ahsan and T. Saramäki, "Significant improvements in translating the Parks-McClellan algorithm from its FORTRAN code to its corresponding MATLAB code," in *IEEE Symps. Circuits Syst.*, Taipei, Taiwan, May 2009, pp. 289–292.

[12] M. Ahsan, "Design of optimum linear phase FIR filters with a modified implementation of the Remez multiple exchange algorithm," Master's thesis, Tampere University of Technology, Tampere, 2008.