# Realisation and Simulation of the Mel Log Spectrum Approximation Filter

*Simple4All Internship Report, August – September 2012*
*Aalto University, Finland*

Jiunn Lin Wong
Jacobs University Bremen, Germany

The main tasks of the internship with Professor Paavo Alku were as follows: I. Background reading on speech synthesis techniques, with a focus on hidden Markov model (HMM) – based approach. II. Implementation of a recursive algorithm to obtain truncated mel-cepstral coefficients from linear prediction (LP) coefficients [1]. III. Implementation of the mel log spectrum approximation (MLSA) filter [2].

In this report, a condensed presentation of my interpretation and translation of the given MLSA filter block diagrams in [2] into Matlab algorithms as well as simulation results of the implantation will be given.

## I. Recursive Algorithm For Cepstral Coefficients

The mel-cepstral coefficients $c(m)$ can be calculated from the LP transfer function by

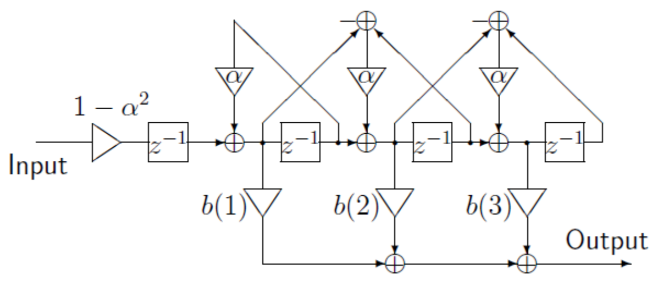
$$\log\left(\frac{K}{1+\sum_{m=1}^{\infty} a(m) z^{-m}}\right) = \sum_{m=1}^{\infty} c(m) z^{-m} = F(z) \qquad (1)$$

where $a(m)$ are the LP coefficients. From this, a recursive algorithm was derived [1]. Here, a Matlab function based on the pseudocode in [1] is realised:

```
function [c] = mel_cepstrum_LP(a,K,N,alpha)
% A recursive algorithm to estimate truncated
mel_cepstral coefficients
% c = [c1 c2 ... cN].' from linear prediction
coefficients a = [a1 a2 ... aM]
% from K.Tokuda,T.Kobayashi and S.Imai,"Recursive
calculation of mel-cepstrum from LP
coefficients,"Nagayo Institute of Technology, 1
April 1994.

% Input:
% a --- linear prediction coefficients a = [a1 a2
... aM]
% K --- all-pole filter coefficient such that
%        log [ K / ( 1+sum(a(m) z^-m))] = sum(c(m)
z^-m)
% N --- desired(truncated)length of 'c'
% alpha --- pole location, such that |alpha| < 1

% Output:
% c --- estimated mel-cepstrum coefficients c = [c1
c2 ... cN].'

M = length(a);
c = zeros(N,1); convo = 0;
% Let v = a~ in literature:
v = zeros(1,N); v(1) = a(1);
for i = -M:-1
    v_old = v;
    v(1) = a(-i)+ alpha*v_old(1);
    v(2) = (1-alpha*alpha)*v_old(1) +
alpha*v_old(2);
    for m = 3:N
        v(m) = v_old(m-1) + alpha*(v_old(m) - v(m-
1));
    end
end

K_tilde = K/v(1);
v = v/v(1);
c(1) = log(K_tilde);
for m = 2:N
    for k = 2:m-1
        temp = (k/m)*c(k)*v(m-k);
        convo = convo + temp;
    end
    c(m) = -v(m) - convo;
end
end
```

## II. The MLSA Filter

To synthesise speech from mel-cepstral coefficients, the exponential transfer function $D(z)$ has to be realised, where $D(z) = \exp(F(z))$ and $F(z) = \sum_{m=1}^{\infty} c(m) z^{-m}$. In [2], Masuko provided an alternative filter structure for $F(z)$ which eliminates the delay-free loop:



(a) Basic filter $F(z)$ $(M = 3)$.

*Figure 1: filter structure for F(z)*

where

$$b = \begin{pmatrix} b(0) \\ \vdots \\ \vdots \\ b(M) \end{pmatrix} = \begin{pmatrix} 1 & (-\alpha) & (-\alpha)^2 & \ldots(-\alpha)^M \\ 0 & 1 & \ddots & (-\alpha)^2 \\ \vdots & \ddots & \ddots & (-\alpha) \\ 0 & \cdots & 0 & 1 \end{pmatrix} \begin{pmatrix} c(0) \\ \vdots \\ \vdots \\ c(M) \end{pmatrix} \qquad (2)$$

The task was to realise the above filter structure in Matlab.

By observation, for an input $x[n]$, and simplifying $\beta = 1-\alpha^2$, the output $y[n]$ for the filter F(z) (by Matlab convention, *i.e.* a vector *v* begins with *v*[1] and not *v*[0] ) is

$$y[2] = b[1] * \beta x[1]$$

$$y[3] = b[2] * \{\overbrace{\beta * x[2] + \alpha\beta * x[1]}^{temp[2]}\}$$
$$+ b[3] * \{\beta * x[1] - \alpha * temp[2]\}$$

$$\vdots$$
$$\vdots$$

$$y[n] = \overbrace{\beta * x[n-1] + \alpha\beta * x[n-2]}^{temp[2]}$$
$$+ \overbrace{\beta * x[n-2] + \alpha\beta * x[n-3] - \alpha * temp[2]}^{temp[3]}$$

$$\vdots$$

$$+ \overbrace{\beta * x[n-(k-1)] + \alpha\beta * x[n-3] - \alpha * temp[n-l]}^{temp[k]}$$
$$+ x[1] - \alpha * temp[k]$$

Note that y[1] is not realisable from the given structure. Hence, we simply use y[1] = c[1]*x[1] .

We also observe that one can first calculate each temporary values *temp*[k] for each y[n] recursively:

$$temp[k] = x[n-(k-1)] + \alpha * x[n-k] - \alpha * temp[k-1], \qquad (3)$$
$$k = 3, 4, 5, \ldots, n-1$$

Finally, sum them up to obtain y[n], for each *n*:

$$y[n] = \beta \begin{bmatrix} 0 \\ temp[2] \\ temp[3] \\ \vdots \\ \vdots \\ temp[n] \end{bmatrix} \times \begin{bmatrix} 0 & b[1] & \cdots\cdots & b[n] \end{bmatrix} \qquad (4)$$

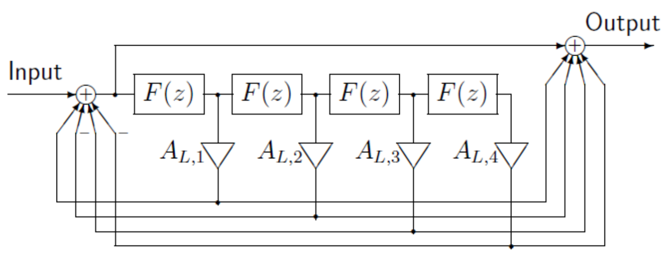
Hence the following Matlab function for the filter F(z):

```
function [y] = mlsa_filter(x, alpha, c)
% Input:
% x     --- input x[n] = [x1 x2 ... x(M+1)]
% alpha --- pole location, such that |alpha|<1
% c     --- mel-cepstral coefficients c = [c1 c2 ...
c(M+1)]

% Output:
% y     --- filtered coeffcients y[n] = f[n]*x[n];
f[n] = MLSA filter
    M = length(x) - 1;
    y = zeros(M+1,1);
    b = zeros(M+1,1);
    temp = zeros(M+1,1);
    beta = 1-alpha*alpha;

    b(M+1)=c(M+1);

    for m = -M:-1
        b(-m) = c(-m)-alpha*b(-m+1);
    end

    %{
    for m = 1:M % There could be mistake in the
paper. Here, b = A'c:
        b(m) = c(m) + alpha*c(m+1);
    end
    %}

    y(1) = c(1)*x(1);
    y(2) = beta*x(1);
    for n = 3:M+1
        temp(2) = x(n-1) + alpha*x(n-2);
        for k = 3:n-1
            temp(k) = x(n-(k-1)) + alpha*x(n-k) -
alpha*temp(k-1);
        end
        temp(n) = x(1) - alpha*temp(n-1);

        y(n) = beta*temp.'*b;
    end
end
```

Having obtained $F(z)$, $D(z) = \exp\left(F(z)\right)$ can be found. However, since $D(z)$ is not a rational function, it can be approximated by Padé approximation:

$$\exp(F(z)) \approx \frac{1 + \sum_{l=1}^{4} A_{4,l} F^l(z)}{1 + \sum_{l=1}^{4} A_{4,l}\left(-F^l(z)\right)^l} = R(z) = \frac{Y(z)}{X(z)} \qquad (5)$$

where the coefficients $A_{L,l}$ have been optimised for $L = 4$ and 5 in [2]. The following MLSA filter was presented in [2] to approximate $D(z)$ from $F(z)$ :



(b) $R_L(F(z)) \simeq \exp F(z) = D(z)$ $(L = 4)$.

*Figure 2: MLSA filter structure for D(z)*

To realise the above filter in algorithm, we can simply expand the Padé approximation

$$\exp(F(z)) \approx \frac{1+\sum_{l=1}^{4} A_{4,l} F^{l}(z)}{1+\sum_{l=1}^{4} A_{4,l}\left(-F^{l}(z)\right)^{l}} = \frac{Y(z)}{X(z)}$$

$$\Rightarrow Y(z) = \overbrace{X(z)+\sum_{l=1}^{4} A_{4,l} F^{l}(z) X(z)}^{\tilde{Y}(z)} - \sum_{l=1}^{4} A_{4,l}\left(-F^{l}(z)\right)^{l} \tilde{Y}(z) \tag{6}$$

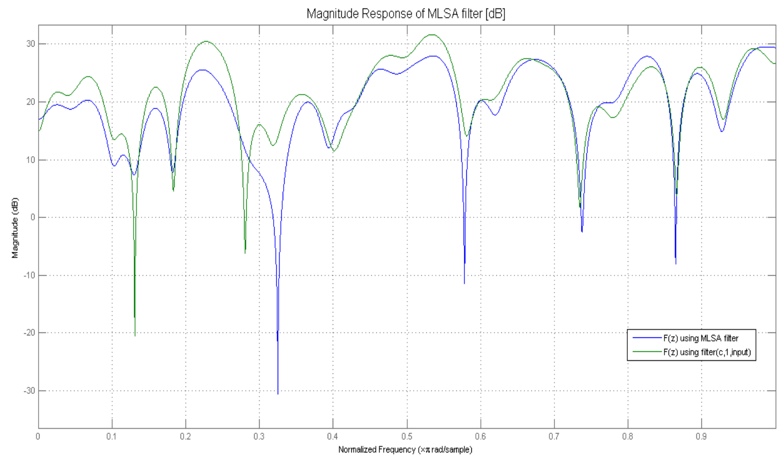
Therefore, the MLSA filter involves cascading the filters $F(z)$ $L$ times, each weighted by the corresponding coefficient $A_{L,l}$, then filter again the above through $F(z)$ $L$ times, finally summing everything with alternating minus sign.

A Matlab function of the MLSA filter is as follows:

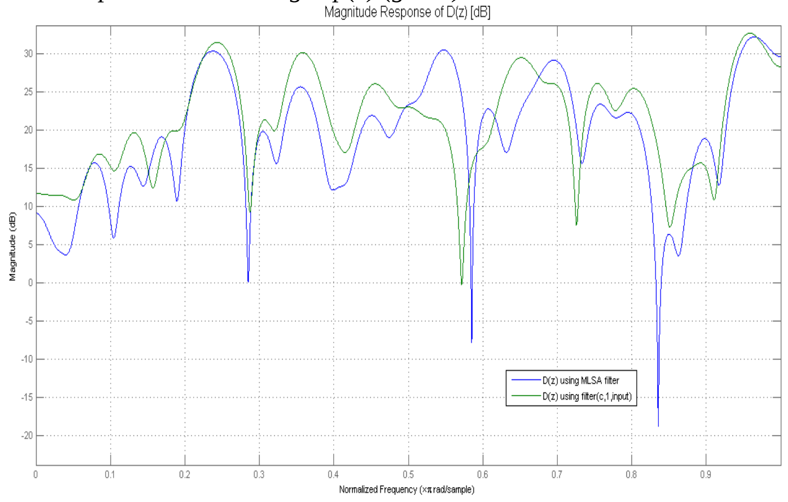
```
function [R] = pade_approx(L, x, c, alpha)
% Obtain R(z)~ exp(F(z)) by Padé approximation

% Input:
% L     --- order of Padé approximation. In the
paper, optimised
%           coefficients were calculated for L = 4
and 5; Here, L = 4 is used.
% x     --- input x[n] = [x1 x2 ... x(M+1)]
% alpha --- pole location, such that |alpha|<1
% c     --- mel-cepstral coefficients c = [c1 c2 ...
c(M+1)]

% Output:
% R     --- R(z) ~ exp(F(z))
    M = length(x)-1;
    R = x;
    F = zeros(M+1,L); % store cascaded filters [F1
F2 ... FL] as columns
    A = [4.999273e-1 1.067005e-1 1.170221e-2
5.656279e-4].';

    F(:,1) = mlsa_filter(x,alpha,c); % Cascade the
filters L times
    for i = 2:4
        F(:,i) = mlsa_filter(F(:,i-1),alpha,c);
    end

    for i = 1:4
        F(:,i) = A(i)*F(:,i); % Multiply by
weighting coefficients A(i)
        R = R + F(:,i); % the FIR part: D = x + F1 +
F2 + F3 + F4
    end

    F(:,1) = mlsa_filter(R,alpha,c); % Cascade the
filters L times again, for D[n]
    for i = 2:4
        F(:,i) = mlsa_filter(F(:,i-1),alpha,c);
    end

R = R + A(1)*F(:,1) - A(2)*F(:,2) + A(3)*F(:,3) -
A(4)*F(:,4);

end
```

In the following simulations, 50 filter tabs were used. The input x[n] and mel-cepstral coefficients c[n] were randomly generated using sinusoidal functions.

*Graph 1: Simulation of F(z) using the above algorithm (blue) versus direct implementation using cepstral coefficients, c (green) :*



*Graph 2: Simulation of D(z) using the MLSA filter (blue) versus direct implementation using exp(c) (green) :*



The MLSA filter appears to approximate the exponential transfer function D(z) reasonably. However, to truly assess the performance the implemented algorithms, real speech data could have been used, and further speech synthesis processes could have been carried out using the filter, which were unfortunately not realisable within the short internship period.

Nonetheless, I am grateful for the opportunity given by the University of Edinburgh and Professor Alku to learn and explore, as the skills and ideas acquired are invaluable for my future research or endeavours.

## REFERENCES

[1] Keiichi Tokuda, Takao Kobayashi, and Satoshi Imai, "Recursive calculation of mel-cepstrum from LP coefficients," *Technical Report of Nagayo Institute of Technology*, April 1994.

[2] Takashi Masuko, "HMM-Based Speech Synthesis and Its Applications," 2002.