Lab 4
Lab report due by 2:00 PM on Monday, March 28, 2016

This is a MATLAB only lab, and therefore each student needs to turn in her/his own lab report and own programs.

---

1 Introduction

This is the first lab on the implementation of the Layer III of MPEGG 1, also known as mp3. In this lab you will implement several subband filters to decompose and reconstruct the original audio signal. There are two types of filtering: one is slicing the frequency axis in critically samples subbands. The other filtering involves the slicing of the time axis overlapping sinusoidal functions.

In this first lab, we implement the polyphase pseudo-QMF that splits the audio signal into 32 frequency band. You will build a code that decompose and a code that reconstruct the audio signal.

1.1 Analysis filter bank

The MP3 encoding scheme first splits the audio signal $x(t)$ into 32 equally spaces frequency bands using a bank of 32 filters h_k,

$$h_k * x[n], \quad k = 1, \ldots, 32. \tag{1}$$

The output of this analysis is critically sampled,

$$s_k[n] = ((h_k * x) \downarrow 32)[n]. \tag{2}$$

As a result, if the incoming signal $x[n]$ contains N samples, the bank of filters yields $N \times 32/32 = N$ samples.

1.2 Synthesis filter bank

The reconstruction is performed by first upsampling the s_k (that is inserting 31 zeros between the samples),

$$(s_k \uparrow 32)[n], \tag{3}$$

and then filtered by g_k

$$(s_k \uparrow 32) * g_k[n], \quad k = 1, \ldots, 32, \tag{4}$$

and finally interleaved to create the reconstructed signal $\tilde{x}[n]$,

$$\tilde{x}[n] = \sum_{k=1}^{32} (s_k \uparrow 32) * g_k[n]. \tag{5}$$

In the case of MP3, the filter bank has nearly (but not exactly) perfect reconstruction: the magnitude of the Fourier transform of the reconstructed signal is slightly attenuated at certain frequencies. In practice, this is not a problem. The advantage of not requiring perfect reconstruction is that the filter design is greatly simplified: the same prototype filter is used for the 32 filters h_k.

2 Cosine modulated pseudo Quadrature Mirror Filter: analysis

In this section, we give the detailed mathematical description of the analysis filter banks. We will manipulate the equations that combine convolution and decimation to arrive at a fast algorithm to compute the output signal s_k for 32 subbands. In section 2.1 we further describe the algorithmic implementation.

Consider the filter h_k, then the output of the combined filtering by h_k and decimation is given by

$$s_k[n] = \sum_{m=0}^{511} h_k[m] x[32n - m], \tag{6}$$

where

$$h_k[n] = p_0[n] \cos\left(\frac{(2k+1)(n-16)\pi}{64}\right) \quad k = 0, \ldots, 31, \quad n = 0, \ldots, 511, \tag{7}$$

and p_0 is a prototype lowpass filter (see Fig. 1). The role of h_k is clear: the modulation of p_0 by $\cos\left(\frac{(2k+1)(n-16)\pi}{64}\right)$ shifts the lowpass filter around the frequency $(2k+1)\pi/64$, and thus h_k will become a bandpass filter that selects frequencies around $(2k+1)\pi/64$, $k = 0, \ldots 31$, with a nominal bandwidth of $\pi/32$, (see Fig. 1).

Equation (6) requires $32 \times 512 = 16,384$ combined multiplications and additions to compute the 32 outputs $s_1, \ldots, s_{32}$ for each block of 32 samples of the incoming signal x. In the following, we develop a faster algorithm to compute this operation.

We start with (6) and we have

$$s_k[n] = \sum_{m=0}^{511} h_k[m] x[32n - m] = \sum_{r=0}^{63} \sum_{q=0}^{7} h_k[64q + r] x[32n - 64q - r] \tag{8}$$

Now, we observe that

$$\cos\left(\frac{(2k+1)(64q+r-16)\pi}{64}\right) = \cos\left(\frac{(2k+1)(r-16)\pi}{64} + q\pi\right) \tag{9}$$

$$= \begin{cases} \cos\left(\frac{(2k+1)(r-16)\pi}{64}\right) & \text{if } \lfloor n/64 \rfloor \text{ is even} \\ -\cos\left(\frac{(2k+1)(r-16)\pi}{64}\right) & \text{otherwise} \end{cases} \tag{10}$$

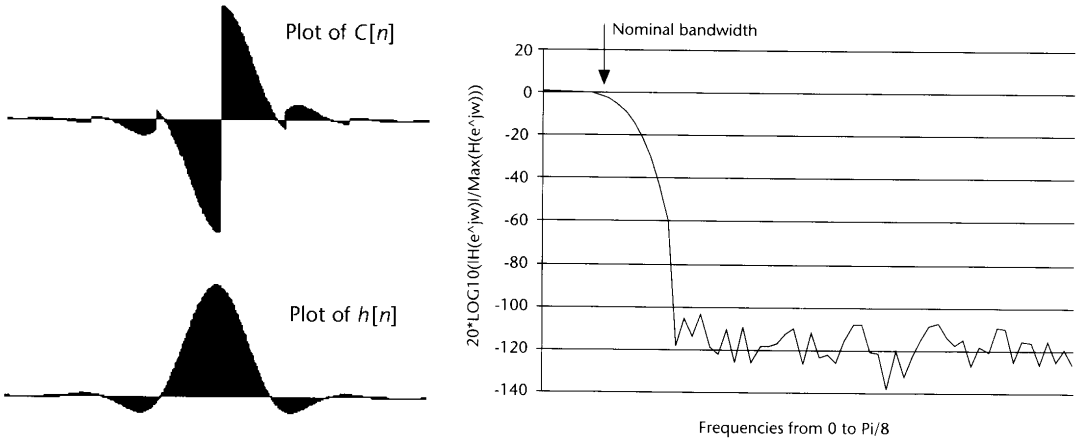

Figure 1: Left: prototype filter $p_0[n]$, called $h[n]$ in [Pan, 1995]. Right: frequency response of the prototype filter p_0. Figures taken from [Pan, 1995].

Let us define (see Fig. 1),

$$c[n] = \begin{cases} -p_0[n] & \text{if } \lfloor n/64 \rfloor \text{ is odd} \\ p_0[n] & \text{otherwise,} \end{cases} \tag{11}$$

then

$$h_k[64q + r] = c[64q + r] \cos\left(\frac{(2k+1)(r-16)\pi}{64}\right) \tag{12}$$

Using the notations of the standard, we further define

$$\boxed{M_{k,r} = \cos\left(\frac{(2k+1)(r-16)\pi}{64}\right), \quad k = 0, \ldots, 31, \ r = 0, \ldots, 63,} \tag{13}$$

then

$$h_k[64q + r] = c[64q + r] M_{k,r}, \tag{14}$$

and

$$s_k[n] = \sum_{r=0}^{63} \sum_{q=0}^{7} c[64q + r] M_{k,r} x[32n - 64q - r] \tag{15}$$

$$= \sum_{r=0}^{63} M_{k,r} \sum_{q=0}^{7} c[64q + r] x[32n - 64q - r] \tag{16}$$

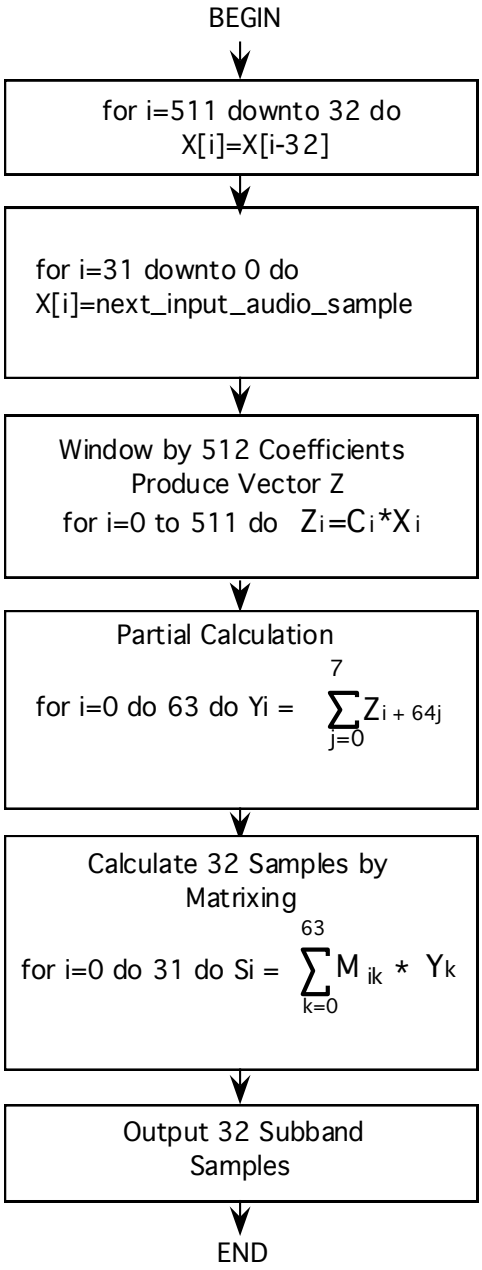

Figure 2: Encoding flow chart: the buffer X receives 32 new audio data, and 32 subband coefficients $S_0, \ldots, S_{31}$ are computed.

In summary, for every integer $m = 32n$, multiple of 32, the convolution (6) can be computed quickly using the following three steps,

$$z[64q + r] = c[64q + r]x[m - 64q - r], \quad r = 0, \ldots, 63, q = 0, \ldots, 7 \tag{17}$$

$$y[r] = \sum_{q=0}^{7} z[64q + r], \quad r = 0, \ldots, 63; \tag{18}$$

$$s[k] = \sum_{r=0}^{63} M_{k,r}\, y[r], \quad k = 0, \ldots, 31. \tag{19}$$

This algorithm is described in the block diagram in Fig. 2. In its most naive implementation, this algorithm requires $512 + 32 \times 64 = 2,560$ multiplications, and $64 \times 7 + 32 \times 63 = 2,464$ additions. Further speedup can be obtained by using a fast DCT algorithm to compute the matrix-vector multiplication in 19.

Note that the output $s_k[n]$ is maximally decimated: for every 32 new input samples, the filter bank generates 32 outputs $s_1[n], \ldots s_{32}[n]$. In other words, for each band k, there is a single output for each new 32 input samples.

2.1 Implementation of the analysis filter bank

Figure 2 shows the flow chart of the filtering of the audio data. The filtering is based on the following ideas.

1. Each processing cycle works on a packet of 32 audio samples.
2. The filtering is performed on a buffer X of size 512, using equations (17)-(13).
3. During each processing cycle, the buffer X is shifted to the right, and $X(480 : 511)$ are discarded. This leaves 32 empty slots that are filled with new data: the 32 new incoming audio data that are saved in $X(0 : 31)$.
4. 32 subband coefficients $s[i], i = 0, \ldots 31$ are computed using (19).
5. For the first 480 audio samples, the buffer X is filled up with zeros. After having processed 512 audio samples, the buffer X is always full.

We assume that we only process monophonic audio. Each frame contains $576 = 18 \times 32$ audio samples. The output of a frame will be a sequence of 18 vectors of subband coefficients of size 32. The following MATLAB code provides a skeleton for the subband filtering in layer III.

```
[audio, fs] = wavread (filename);
...
frameSize = 576;
nFrame = floor (length (audio)/frameSize);

for frame = 1:nFrame                % chunk the audio into blocks of 576 samples
    offset = (frame - 1)* frameSize;       % absolute address of the frame
    for index = 1:18                  % 18 non overlapping blocks of size 32

        % process a block of 32 new input samples
        % see flow chart in Fig. 2
        ...
        ...

        % Frequency inversion
        if (mod (index,2) == 1)
            channel = 1:2:32;
            S(channel) = -S(channel);
        end

end
end
```

The 512 coefficients of the filter c (see Fig. 3) are assigned using the function `loadwindow.m` that can be downloaded from D2L.

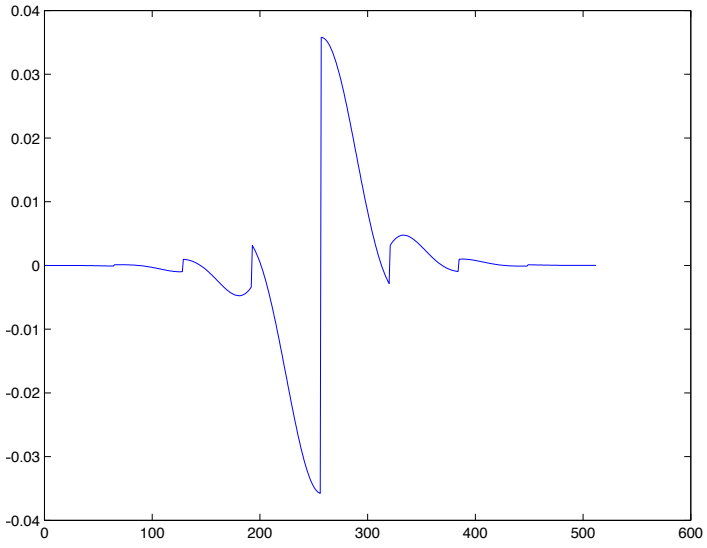

Figure 3: Analysis window c.

2.2 Frequency inversion

After computing the 32 coefficients $s[i]$, we need to perform a frequency inversion correction to order the coefficients in by increasing frequency. Indeed, the filter's output are scrambled. The unscrambling, involves multiplying by -1 every even coefficients of every even subband of the polyphase filterbank.

This step makes it possible to create very smooth output in the coefficients, as a function of time, and leads the way for further transformation using the MDCT.

Assignment

1. Write the MATLAB function `pqmf` that implements the analysis filter bank described in equations (17-13). The function will have the following template:

   ```
   [coefficients] = pqmf (input)
   ```

   where `input` is a buffer that contains an integer number of frames of audio data. The output array `coefficients` has the same size as the buffer `input`, and contains the subband coefficients.

   The array coefficients should be organized in the following manner:

   $$\texttt{coefficients} = \begin{bmatrix} S_0[0] & \dots & S_0[N_S-1] & \dots S_{31}[0] \dots S_{31}[N_S-1] \end{bmatrix} \tag{20}$$

   where $S_i[k]$ is the coefficient from subband $i = 0, \dots, 31$ computed for the packet k of 32 audio samples. Also N_S is the total number of packets of 32 samples:

   $$N_S = \frac{\texttt{nSamples}}{32} = 18 * \texttt{nFrames} \tag{21}$$

   The organization of `coefficients` is such that the low frequencies come first, and then the next higher frequencies, and so on and so forth. Figure 4 displays the array coefficients for the first 5 seconds of the piano sonata in `sample1.wav`. You notice that the absence of very low frequencies, and few high frequencies. In comparison, figure 5 displays the array coefficients for the first 5 seconds of the dance floor piece in `sample2.wav`. You immediately notice the presence of very low frequencies (the thumping bass), and also significant high frequencies. These high frequencies may be interpreted as noise, and could be removed in an effort to compress the signal (more about that in the rest of the lab).

2. Analyze the first 5 seconds of the following tracks, and display the array `coefficients`, as in Fig. 4,

   - sample1.wav, sample2.wav
   - sine1.wav, sine2.wav
   - handel.wav
   - cast.wav
   - gilberto.wav

   Comment on the visual content of the arrays `coefficients`.

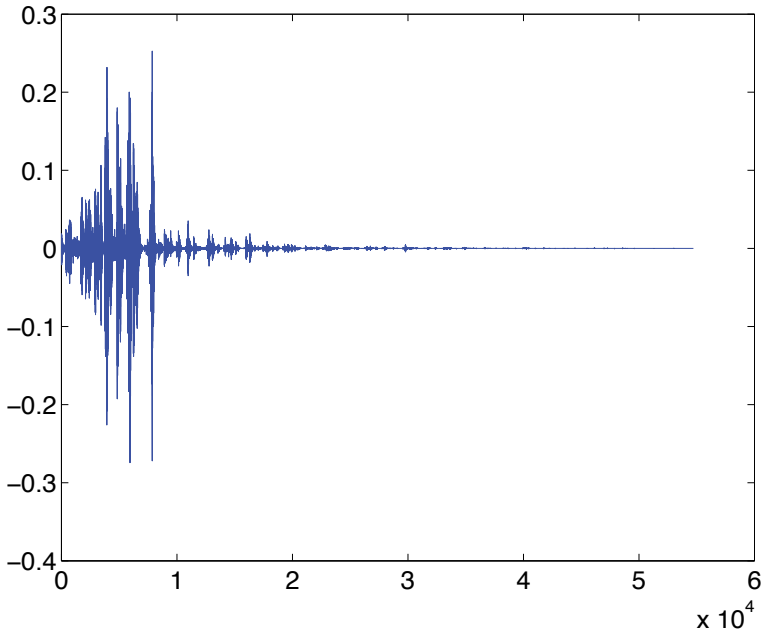

Figure 4: The array `coefficients` for the first 5 seconds of the audio sample 1.

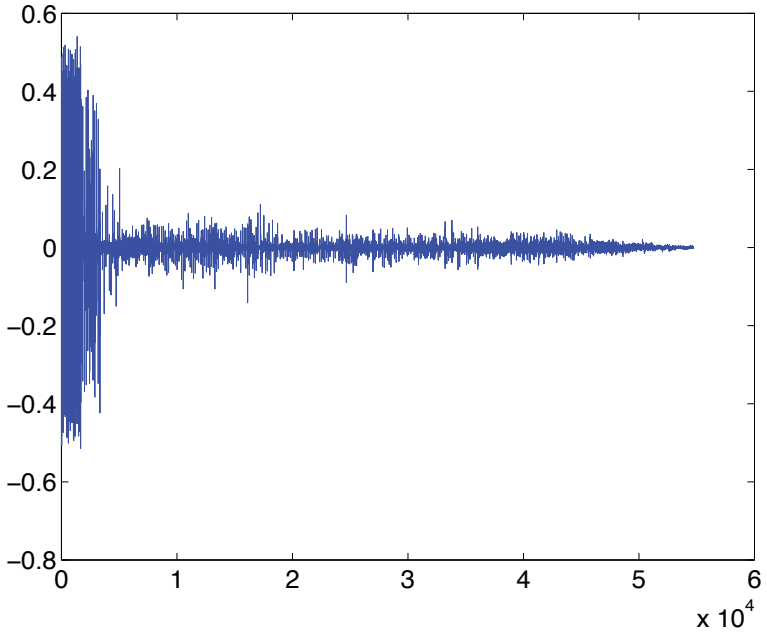

Figure 5: The array `coefficients` for the first 5 seconds of the audio sample 2.

2.3 Synthesis filter bank

The synthesis, or reconstruction, from the coefficients is performed in a very similar manner. The following equations yield the reconstruction of 32 audio samples from 32 subband coefficients

$$\text{for } i = 1023 \text{ down to } 64 \text{ do}$$
$$v[i] = v[i-64] \tag{22}$$

$$\text{for } i = 63 \text{ down to } 0 \text{ do}$$
$$v[i] = \sum_{k=0}^{31} N_{i,k}\; s[k], \tag{23}$$

$$\text{for } i = 0 \text{ to } 7 \text{ do}$$
$$\text{for } j = 0 \text{ to } 31 \text{ do}$$
$$u[64\,i + j] \qquad = v[128\,i + j] \tag{24}$$
$$u[64\,i + j + 32] = v[128\,i + j + 96] \tag{25}$$

$$\text{for } i = 0 \text{ to } 511 \text{ do}$$
$$w[i] = d[i]\; u[i], \tag{26}$$

$$\text{for } j = 0 \text{ to } 31 \text{ do}$$
$$x[j] = \sum_{j=0}^{15} w[j + 32i]. \tag{27}$$

where

$$\boxed{N_{i,k} = \cos\left(\frac{(2k+1)(16+i)\pi}{64}\right), \quad i = 0, \ldots, 63,\ k = 0, \ldots, 31.} \tag{28}$$

Figure 7 shows the flow chart of the filtering of the audio data. The filtering is based on the following ideas.

1. The filtering is performed on a buffer V of size 1024, using equations (23)-(28).
2. Each processing cycle works on a packet of 32 audio samples.
3. During each processing cycle, the buffer V is shifted to the right, and $V(959:1023)$ are discarded (22). This leaves 64 empty slots that are filled with new data: the 64 numbers that are obtained from 32 coefficients using (26).
4. The audio samples are reconstructed using (27).
5. The cosine transform is different from the analysis filter, and is given by (28).

6. The prototype filter for the synthesis is $d[i]$. The 512 coefficients of the synthesis window d (see Fig. 6) are assigned using the function `loadwindow.m`.

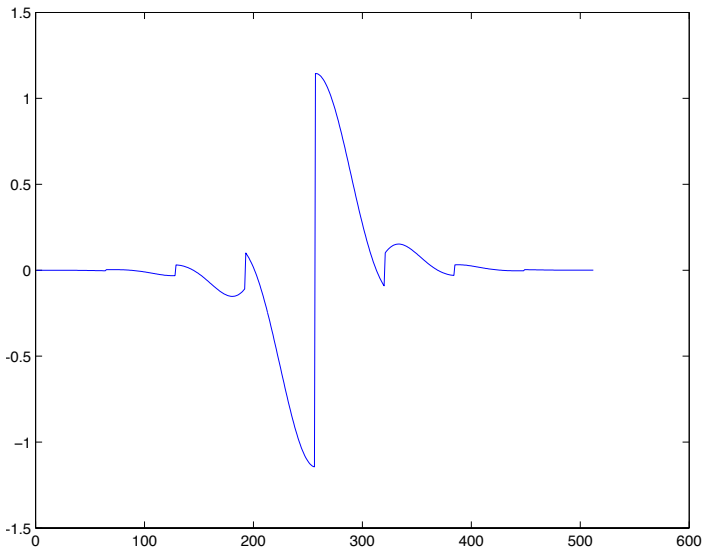

Figure 6: Analysis window d.

2.4 Frequency inversion

Before reconstructing the audio sample, we need to cancel the effect of the frequency inversion created during the analysis. The inversion, involves multiplying by -1 every second coefficients of every second subband of the polyphase filterbank.

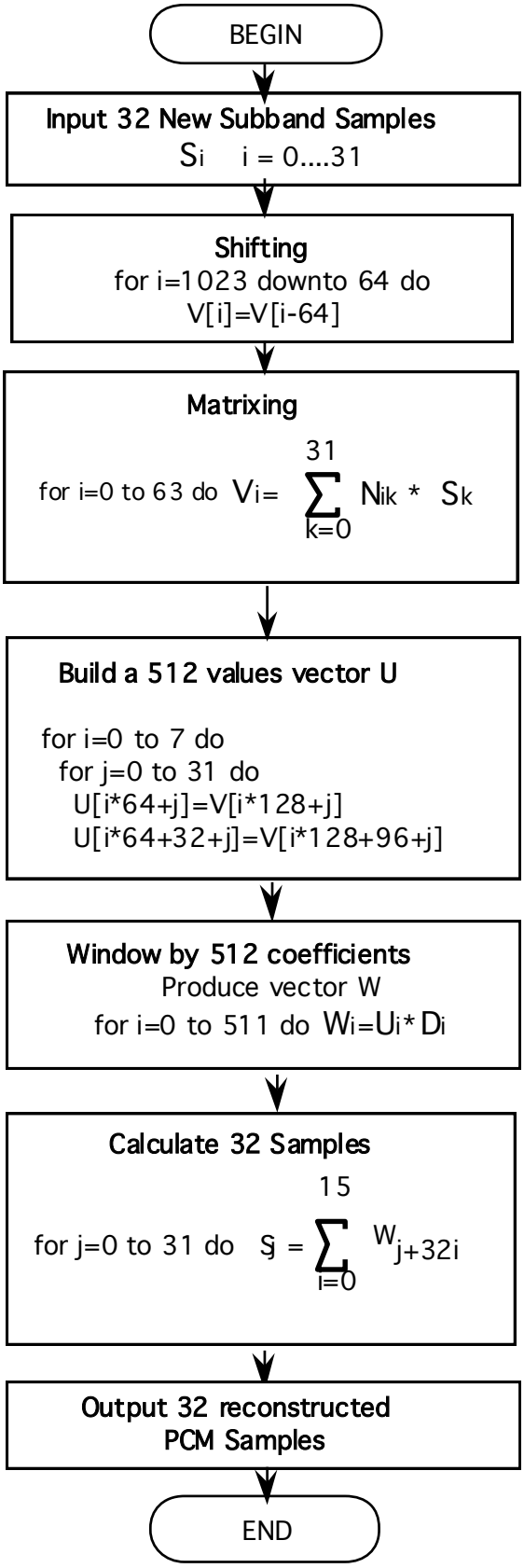

Figure 7: Decoding flow chart: the buffer V receives 32 new PQMF coefficients, $S_0, \ldots, S_{31}$, and 32 new audio samples $X_0, \ldots, X_{31}$ are reconstructed.

Assignment

3. Write the MATLAB function `ipqmf` that implements the synthesis filter bank described in equations (23-(28). The function will have the following template:

   ```
   [recons] = ipqmf (coefficients)
   ```

   where `coefficients` is a buffer that contains the coefficients computed by `pqmf`. The output array `recons` has the same size as the buffer `coefficients`, and contains the reconstructed audio data.

4. Reconstruct the first 5 seconds of the following tracks, and display the signal `recons` on top of the signal `input` as in Fig. 8. You should observe that the reconstructed signal is slightly delayed. This is due to the fact that the processing assumes that a buffer of 512 audio sample is immediately available.

   - sample1.wav, sample2.wav
   - sine1.wav, sine2.wav
   - handel.wav
   - cast.wav
   - gilberto.wav

5. Compute the maximum error between the reconstructed signal and the original, taking into account the delay. The error should be no more than 10^{-5}. Explain how you estimate the delay.

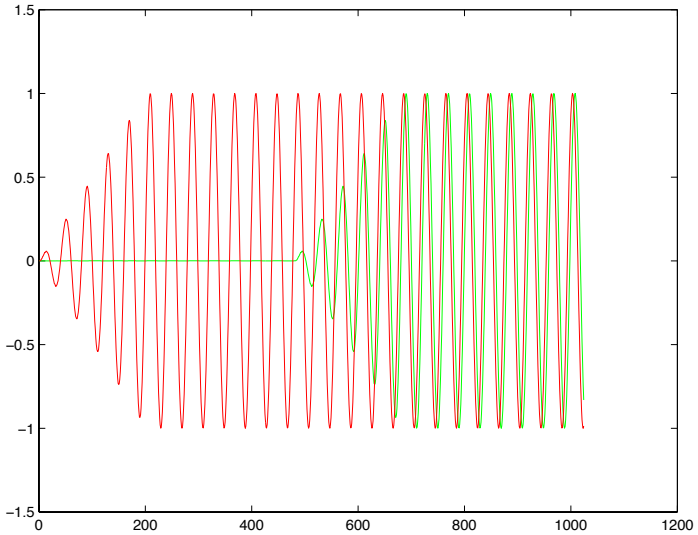

Figure 8: First 1024 samples from the reconstructed signal (green) and the original signal (red), for `sine1.wav`.

Assignment

6. Modify your code to reconstruct an audio signal using only a subset of bands. This is the beginning of compression. Your function prototype should look like this:

   ```
   [recons] = ipqmf (coefficients, thebands)
   ```

   where `thebands` is an array of 32 integers, such that `thebands[i]` =1 if band `i` is used in the reconstruction, and `thebands[i]` = 0 if band `i` is not used.
   Experiment with the files

   - sample1.wav, sample2.wav
   - sine1.wav, sine2.wav
   - handel.wav
   - cast.wav
   - gilberto.wav

   and describe the outcome of the experiments, when certain bands are not used to reconstruct. If you define the compression ratio as

   $$\frac{\text{number of bands used to reconstruct}}{32}, \tag{29}$$

   explain what is a good compression ratio, and a good choice of bands for each audio sample. Note that the psychoacoustic model of MP3 performs this task automatically.

References

[Pan, 1995] Pan, D. (1995). A tutorial on mpeg/audio compression. *IEEE Multimedia magazine*, 2(2):60–74.