Signal Processing for Speech Recognition

Once a signal has been sampled, we have huge amounts of data, often 20,000 16 bit numbers a second! We need to find ways to concisely capture the properties of the signal that are important for speech recognition before we can do much else. We have seen that a spectral representation of the signal, as seen in a spectrogram, contains much of the information we need. We can obtain the spectral information from a segment of the speech signal using an algorithm called the **Fast Fourier Transform**. But even a spectrogram is far too complex a representation to base a speech recognizer on. This section describes some methods for characterizing the spectra in more concise terms.

1. Filter Bank Methods

One way to more concisely characterize the signal is by a filter bank. We divide the frequency range of interest (say 100-8000Hz) into N bands and measure the overall intensity in each band. This could be done using hardware or digital filters directly from the incoming signal, or be computed from a spectral analysis (again derived using hardware or software such as the Fast Fourier Transform). In a *uniform filter bank*, each frequency band is of equal size. For instance, if we used 8 ranges, the bands might cover the frequency ranges

100Hz-1000Hz, 1000Hz-2000Hz, 2000Hz-3000Hz, ... , 7000Hz-8000Hz.

Consider a uniform filter bank representation of an artificially generated spectra similar to that for the vowel IY shown in Figure 1. We can measure the intensity in each band by computing the "area" under the curve. With a discrete set of sample points, we could simply add up all the values in range, or compute a "power" measure by summing the squares of the values. With the signal shown in Figure 1, we would get a representation of the spectra that consists of a vector of

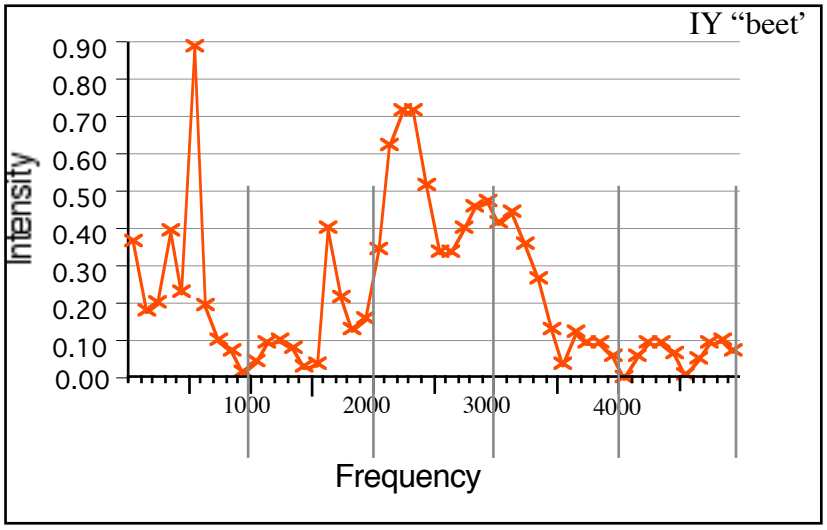


Figure 1: Uniform Filter Bank on Spectra for Vowel IY

eight numbers, in this case

(1.26, .27, 2.61, .62, .05, .04, .03, .02)

How would we know whether this is a good representation? We'd need to compare it to representations of other vowels and see whether the vector reflects differences in the vowels. If we do this, we'll see there are some problems with a uniform filter bank. Specifically, we know that much key information in vowels focuses around the formants, which should show up as intensity peaks in various bands in the filter bank. But if we look back at the frequency ranges of typical vowels in Table 1 we see a problem. The first format of all vowels will always be in range 1 so the frequency differences in the F1 of the words will be reflected in the representation. Then the second formant will almost always in in range 2, and the third formant is typically in range 3. Thus we will get little information to help distinguish the vowels. If we classify each vowel by three numbers indicating the filter banks that their formants fall into using the uniform bank. This encoding separates the vowels into only four classes, with most vowels falling into one class (all with their formants falling in banks 1, 2 and 3 respectively).

A better alternative is to organize the ranges using a logarithmic scale, and this actually agrees better with human perceptual capabilities as well. We set the first range to have the width W, and then subsequent widths are a^n*W. If a = 2 and W is 200 Hz, we get widths of 200 Hz, 400 Hz, 800 Hz, 1600 Hz, and so on. Our frequency ranges would now be

100Hz-300Hz, 300Hz-700Hz, 700Hz-1500Hz, 1500Hz-3100Hz, 3100Hz-6300Hz

With this set of banks, we see the F1 of vowels vary over three ranges, but F2 falls in only two ranges, and F3 in one. If we lower the value of *a* (say to 1.5), we get ranges of widths 200, 300, 450, 675, 1012, 1518, with frequency bands starting at 100Hz, 300 Hz, 600 Hz, 1050 Hz, 1725 Hz, 2737 Hz, and 4255 Hz. With this each of the formants fall across three frequency ranges, giving us good discrimination.

ARPABET	Typical Word	F1	F2	F3	filter #s
IY	beet	270	2290	3010	1, 3, 4
IH	bit	390	1990	2550	1, 2, 3
EH	bet	530	1840	2480	1, 2, 3
AE	bat	660	1720	2410	1, 2, 3
AH	but	520	1190	2390	1, 2, 3
AA	hot	730	1090	2440	1, 2, 3
AO	bought	570	840	2410	1, 1, 3
UH	foot	440	1020	2240	1, 2, 3
UW	boot	300	870	2240	1, 1, 3
ER	bird	490	1350	1690	1, 2, 2

Table 1: Typical formants for vowels and where they fall in a uniform filter bank

The third method is to design a *non-uniform* set of frequency bands that has no simple mathematical characterization but better reflects the responses of the ear as determined from experimentation. One very common design is based perceptual studies to define critical bands in the spectra. A commonly used critical band scale is called the Mel scale which is essentially linear up to 1000 Hz and logarithmic after that. For instance, we might start the ranges at 200 Hz, 400 Hz, 630 Hz, 920 Hz, 1270 Hz, 1720 Hz, 2320 Hz, and 3200 Hz. The frequency bands now look as shown in Figure 2. The Mel scale typically gives good discrimination based on expected formant intensity peaks.

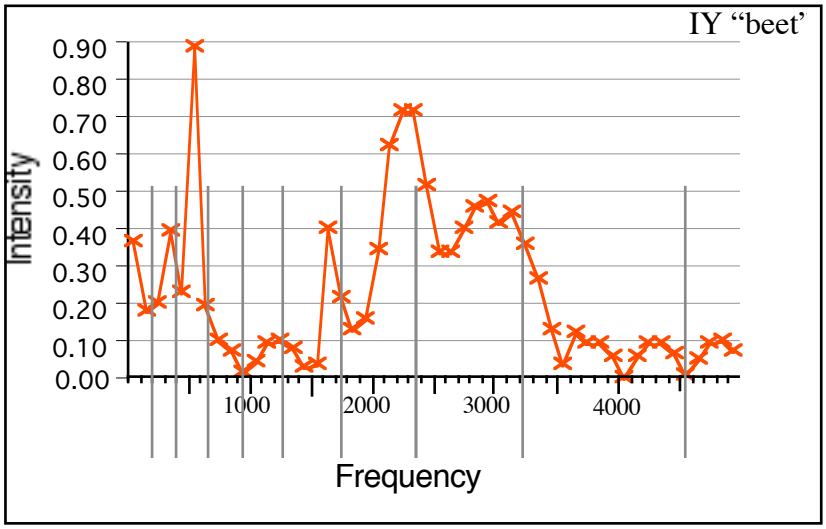


Figure 2: A set of Mel scale frequency ranges

2. Designing a Good "Window" Function

If we simply take the samples as they are in a segment, when we apply a spectral analysis technique like the Fast Fourier Transform, it acts like it is operating on a signal that is zero before the segment starts and then abruptly jumps to the signal during the segment and then back to zero when the segment ends. When the signal is 0 outside of the window and then jumps to the actual values within the window, this introduces significant distortion of the signal, making it appear like there is significant high frequency noise at the beginning and end points of the window.

The typical method to alleviate this problem is to not count all values in a window equally. We apply a function to the samples in the window so that the samples near the beginning and end of the window slowly winnow down to zero. More specifically, if the original signal intensity is s(i) at time i, we can represented the windowed signal as

$$s'(i) = s(i) * w(i)$$

where w(i) is the **window function**. In our segments above, w(i) was a square function that jumped to 1 inside the segment and was 0 outside. Figure 4 shows a signal for a simple sinusoidal curve after applying a window function.

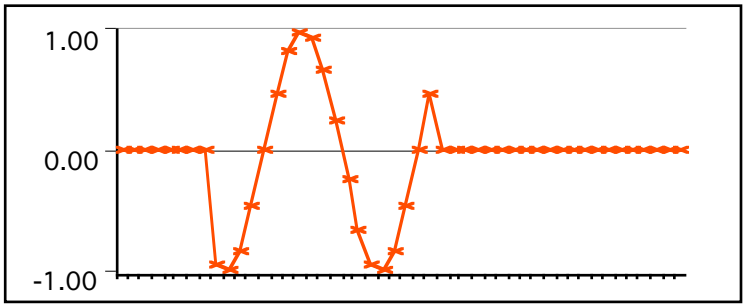


Figure 4: The effect of windowing with a square window function

A common window function that works well is called the **Hamming Window** and is defined by the formula

$$w(i) = .54 - .46 \cos(2pi/(N-1))$$

This function is shown in Figure 5.

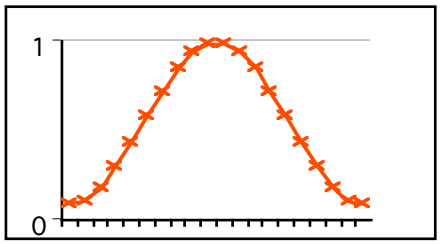


Figure 5: The Hamming Window Function

Applying this to our sinusoidal wave above, it now looks shown in Figure 6. Compare this to Figure 4 and see that the new signal is considerably smother at the ends. This produces a much superior spectral analysis.

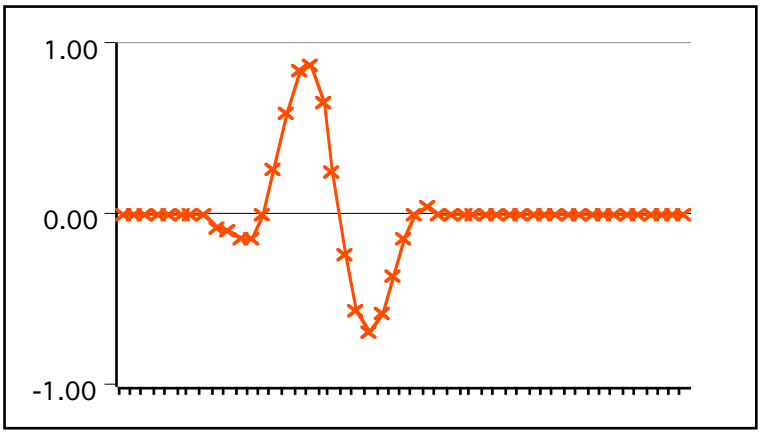


Figure 6: Applying the Hamming Window to the Sinusoidal Function

3. LPC Analysis

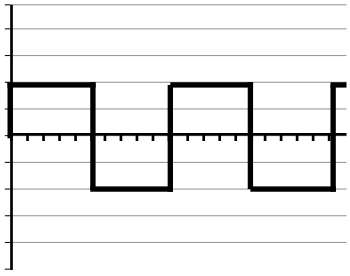
Another method for encoding a speech signal is called **Linear Predictive Coding (LPC)**. LPC is a popular technique because is provides a good model of the speech signal and is considerably more efficient to implement that the digital filter bank approach. With ever faster computers, however, this is becoming less of an issue. The basic idea of LPC is to represent the value of the signal over some some window at time t, s(t) in terms of an equation of the past n samples, i.e.,

$$s(t) = a_1*s(t -1) + a_2*s(t - 2) + ... + a_p*s(t - p)$$

Of course, we usually can't find a set of a_i's that give an exact answer for every sample in the window, so we must settle for the best approximation, s'(t), that minimizes the error. We typically try to measure error by the least-squares difference, i.e.,

$$S_{t=1,w} (s(t) - s'(t))^2$$

If the signal is periodic and hence repeats in a pattern over the window, we can get very good estimates. For instance, consider a 20 Hz square wave over a 100 ms window sampled at 200 Hz. Thus there are 20 samples in a window and the wave contains two cycles (one every 50 ms).



The signal s(t) has the values: s(0) = 0, s(1) = 1, s(2) = 1, s(3) = 1, etc, producing the sequence of values

1, 1, 1, 1, 0, -1, -1, -1, -1, 0, 1, 1, 1, 1, 0, -1, -1, -1, -1, 0

Consider the errors obtained from some simple "displacement" approximations $D_i(t) = s(t - i)$, i.e., $a_i = 1$, and all other $a_j = 0$. The error for the approximation D(1) would be

$$S_{t=1, 20} (s(t) - s(t - 1))^2 = (1 - 0)^2 + (1 - 1)^2 + (1 - 1)^2 + (1 - 1)^2 + (0 - 1)^2 + (-1 - 0)^2 ...$$

Most of these values are zero (a consequence of the exact nature of square waves), but 8 of them are not and give an error weight of 1 each. Thus the error for this approximation is 8. Notice we either need to start i samples into the window or we will need some values of the signal before the window. For the moment, we allow ourselves to "peek" at the signal before the window.

With a displacement of two, we get an even larger error from terms such as at position 6 with value $(-1 -1)^2 = 4$. Table 2 in column 2 below shows the error for the values 1 through 10.

Displacement	Error
1	8
2	24
3	40
4	56
5	64
6	56
7	40
8	24
9	8
10	0

Table 2: Errors for different Displacements

Notice that we get an exact representation of the signal with D10 since s(t) = s(t - 10) (because it 20 Hz signal and thus takes 1/20 of a second, or 10 samples at 200 Hz sampling rate, to repeat.)

Of course, when using the LPC methods, we look for the best approximation with no restriction to simple displacement functions. Typically, all coefficients a_i will be non-zero. Each coefficient will indicate the contributions of frequencies that repeat every i samples.

We haven't discussed what value of p is reasonable? Given the relationship between the displacement and the frequencies it detects, we should pick a p large enough to capture the lowest frequencies we care about. If this is 200 Hz, then each cycle takes 1/200 of a second, or 5 ms. If we sample at 10 kHz, then 1/200 of a second involves 50 samples, so p should be at least 50. In practice, systems using LPC use even fewer parameters, with typical values between 8 and 16.

4. Building Effective Vector Representations of Speech

Whether we use the filter bank approach or the LPC approach, we end up with a small set of numbers that characterize the signal. For instance, if we used the Mel-scale with dividing the spectra into 7 frequency ranges, we have reduced the representation of the signal over the 20 ms segment to a vector consisting of eight numbers. With a 10 ms shift in each segment, we are representing the signal by one of these vectors every 10 ms. This is certainly a dramatic reduction in the space needed to represent the signal. Rather than 10,000 or 20,000 numbers per second, we now represent the signal by 700 numbers a second!

Just using the six spectral measures, however, is not sufficient for large-vocabulary speech recognition tasks. Additional measurements are often taken that capture aspects of the signal not adequately represented in the spectrum. Here are a few additional measurements that are often used:

Power: a measure of the overall intensity. If the segment S_k contains N samples of this signal, s(0),..., s(N-1), then the power power(S_k) is computed using power(S_k) = $\sum_{i=1,N-1} s(i)^2$. An alternative that doesn't create such a wide difference between low and soft sounds

uses the absolute value: $\sum_{i=1,N-1} |s(i)|$. One problem with direct power measurements it that the representation is then very sensitive to how loud the speaker is speaking. To adjust for this, the power can be normalized by an estimate of the maximum power. For instance, if P is the maximum power within the last 2 seconds, the normalized power of the new segment would be power(S_k)/P. The power is an excellent indicator of the voiced/unvoiced distinction, and if the signal is especially noise-free, can be used to separate silence from low intensity speech such as unvoiced fricatives.

Power Difference: The spectral representation captures the static aspects of a signal over the segment, but we have seen that there is much information in the transitions in speech. One way to capture some of this is to add a measure to each segment that reflects the change in power surrounding it. For instance, we could set PowerDiff(S_k)= power(S_{k+1})-power(S_{k-1}). Such a measure would be very useful for detecting stops.

Spectral Shifts: Besides shifts in overall intensity, we saw that frequency shifts in the formants can be quite distinctive, especially with diphongs and in looking at the effects of consonants next to vowels. We can capture some of this information by looking at the difference in the spectral measures in each frequency band. For instance, if we have eight frequency intensity measures for segment S_k, $f_k(1),...,f_k(8)$, then we can define the spectral change for each segment as with the power difference, i.e., $df_k(i) = f_{k-1}(i)-f_{k+1}(i)$

With all these measurements, we would end up with a 18-number vector, the 8 spectral band measures, eight spectral band differences, the overall power and the power difference. This is a reasonable approximation of the types of representations used in current state-of-the-state speech recognition systems. Some systems add another set of values that represent the "acceleration", and would be computed by calculating the differences between the df_k values.

Note that we do not need to explicitly store the delta parameters since they can be computed quickly when needed. For instance, consider the following sequence of vectors

	Power	m1	m2	m3	m4	m5	m6	m7	m8
v_1	(5,	0,	1,	0,	1,	0,	1,	0,	1)
v_2	(3,	0,	0,	1,	0,	0,	1,	1,	1)
v_3	(6,	2,	1,	1,	0,	1,	2,	1,	2)
v_4	(30,	10,	10,	3,	1,	4,	6,	6,	6)
v_5	(50,	15,	15,	5,	3,	8,	12,	12,	13)
v_6	(52,	16,	15,	4,	3,	9,	13,	11,	13)
v_7	(48,	15,	15,	6,	3,	9,	9,	11,	10)

We can expand these out "on the fly" to extend the current vector with the delta coefficients:

Δv_2	(1,	2,	0,	1,	-1,	1,	1,	1,	1)
Δv_3	(27,	10,	10,	2,	1,	4,	5,	5,	5)
Δv_4	(46,	13,	14,	4,	3,	7,	10,	11,	11)
Δv_5	(22,	6,	5,	1,	2,	5,	7,	5,	7)
Δv_6	(-2,	0,	0,	1,	0,	1,	-3,	1,	-3)

This example shows what we would expect to see in a voiced stop - a rapid increase in energy peaking at Δv_4 which then levels out.

5. Improving Vector Representations in Speech Recognition

If the vector representation is to be useful for speech recognition, then we'll need to define a **distance metric** that indicates how similar two vectors sound to each other. Note that there are many possible distance metrics and only some will correspond to perceptual differences in the signal. This section explores some of these issues.

A standard distance metric is a function $d(v_i,v_j)$ with the following properties

The distance between two identical points is zero (i.e., $d(v_i,v_i) = 0$)

The distance between two different points is greater than 0 (i.e., $d(v_i,v_j) > 0$ for all $i \neq j$)

Distances are symmetric (i.e., $d(v_i,v_j) = d(v_j,v_i)$, for all i and j)

Distances satisfy the triangle inequality (i.e., $d(v_i,v_k) \leq d(v_i,v_j) + d(v_j,v_k)$)

A very common distance metric is the Euclidean distance, which is defined as

$d(v_i,v_j) = SQRT(\sum_{x=1,N} (v_i(x)-v_j(x))^2)$. where SQRT is the square root function.

Another common distance measure is the "city block" measure, where distance is measured in terms of straight line distances along the axis. In two dimensions, this means you can only move vertically or horizontally. The metric is

$d(v_i,v_j) = \sum_{x=1,N} |v_i(x)-v_j(x)|$ (i.e., the sum of the absolute values for each dimension)

To enable accurate speech recognition, we would like to have a vector representation and a distance measure that reflects perceptual differences found in humans. The current representation used above falls short in a number of critical areas and would classify different spectra that are perceptually quite similar to humans as quite different. For example, it makes little perceptual difference to a human whether a person is speaking slightly more loudly or softly. A person maybe able to notice the difference but it is not relevant to the speech sounds perceived. But currently our vector quantization measure is quite sensitive to intensity changes. For the sake of keeping the examples simple, lets consider a representation consisting of a 5 element vector indicating the overall magnitude and 4 Mel scale frequency ranges

(30, 22, 10, 13, 5).

If this same sound was uttered but with twice the intensity (say the microphone was turned up slightly), we might get measures that are essentially double, i.e.,

(60, 44, 20, 26, 10).

A Euclidean difference measure between these would give the value 40.96, the same as the difference between the original vector and absolute silence (i.e., (0,0,0,0,0)). Clearly this is not a good measure!

There are several things that can be done to get around this problem. First, we might consider that human perceptual response to signals is more closely related to a logarithmic scale rather than a linear scale. This suggests that we use a **log filter bank** model, which uses a logarithmic scale for the values. One method would be to simply take the log of all values when we construct the vectors. For the above example, our first vector would now be

(1.48, 1.34, 1, 1.11, .7)

and the vector at double the intensity would be

(1.78, 1.68, 1.3, 1.43, 1).

Now the Euclidean distance between the two vectors is .67 compared to the distance of 2.59 between the first vector and absolute silence. So this is a good step in the right direction.

Another method for compensating for differences in intensity is to normalize all intensities relative the mean (or maximum) of intensities found in the signal, like we did with power in the last section. As before, we can estimate the mean by finding the mean of the intensity measures over some previous duration of speech, say 2 seconds. If the mean vector at time i is F_m, and the filter bank values at i are the vector F_i, then our vector representation would be F_i-F_m, i.e.,

($F_i(1)$-$F_m(1)$, ..., $F_i(k)$-$F_m(k)$).

This introduces considerable robustness over differences caused by different speakers, microphones and the normal intensity variations within the speech of a single person over time.

Perceptual studies have found other distortions of the signal that appear to have little perceptual relevance to recognizing what was said, including

Eliminating all the frequencies below the first formant

Eliminating all frequencies above the third formant

Notch filtering: removing any narrow range of frequencies.

On the other hand, other apparently small changes in the spectra can make large perceptual differences. Changing a formant frequency by as little as 5% can change the perceived phoneme. Clearly such a difference is not going to create much distinction in our filter bank representation. On the other hand, the three non-significant changes above could make significant changes to the values in our representation.

There are several ways to compensate for this problem. One is to use a **weighted Euclidean distance measure** (a commonly used measure is called the Mahalanobis distance), where changes in some bands are much more critical than changes in other bands. For example, with our 8 band Mel scale filter bank, we might weight the bands possibly involving the first formant (e.g., 200-440, 400-630, 630-920) the most, and downplay the weight of the band above 3200 Hz. Thus we define distance as between vector v_i and v_x as

$$d(v_i,v_x) = \text{SQRT}(\sum_k w_k * (v_i(k) - v_x(k))^2$$

Of course, pulling the weights w_k out of a hat is unlikely to produce the best performance, and a good set of weights would be derived by empirical means by testing how different weights affect the recognition rate.

Another technique that has proven to be effective in practice is to compute a different set of vectors based on what are called the **Mel Frequency Cepstral Coefficients** (**MFCC**). These coefficients provide a different characterization of the spectra than filter banks and work better in practice. To compute these, we start with a log filter bank representation of the spectra. Since we are using the banks as an intermediate representation, we can use a larger number of banks to get a better representation of the spectra. For instance, we might use a Mel scale over 14 banks (ranges starting at 200, 260, 353, 493, 698, 1380, 1880, 2487, 3192, 3976, 4823, 5717, 6644, and 7595). The MFCC are then computed using the following formula:

$$c_i = \sum_{j=1,14} m_j * \cos(p*i*(j - 0.5)/14) \qquad \text{for } i = 1, N$$

where N is the desired number of coefficients. What this is doing is computing a weighted sum over the filter banks based on a cosine curve. The first coefficient, c_0, is simply the sum of all the filter banks, since i = 0 makes the argument to the cosine function 0 throughout, and cos(0)=1. In essence its an estimate of the overall intensity of the spectrum weighting all frequencies equally. The coefficient c1 uses a weighting that is one half of a cosine cycle, so computes a value that compares the low frequencies to the high frequencies. These first two weighting functions are shown in Figure 7.

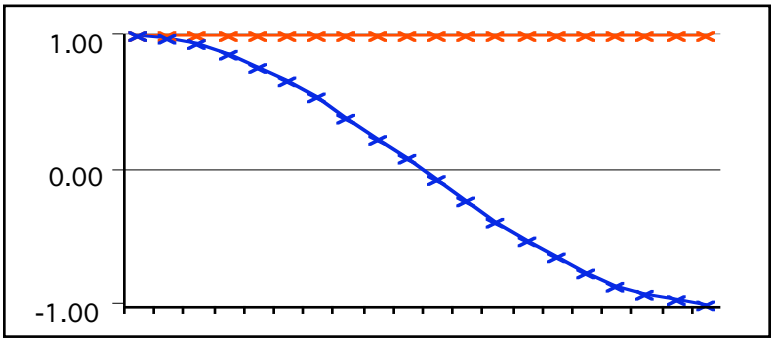


Figure 7: The weighting given to the spectrum for c_0 and c_1

Expanding out the equation, we get

$$c_1 = 0.99*m_1 + 0.94*m_2 + 0.84*m_3 + 0.71*m_4 + 0.53*m_5 + 0.33*m_6 + 0.11*m_7$$
$$-0.11*m_8 - 0.33*m_9 - 0.53*m_{10} - 0.71*m_{11} - 0.85*m_{12} - 0.94*m_{13} - 0.99 * m_{14}$$

The function for c_2 is one cycle of the cosine function, while for c3 it is one and a half cycles, and so on. The functions for c_2 and c_3 are shown in Figure 8.

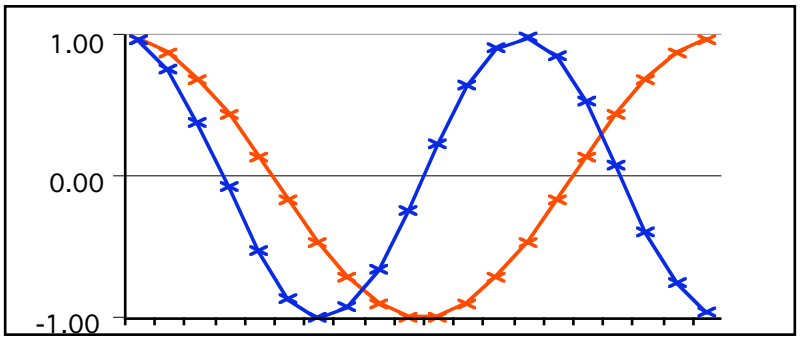


Figure 8: The weighting given to the spectrum for c_2 and c_3

This technique empirically gives superior performance and this representation is used in many state-of-the-art recognition systems. In other words, an 8 element vector based on the MFCC will generally produce better recognition than an 8 element vector based on Mel scale filter banks plus the overall power. We don't need an overall power measure in the MFCC since the power is estimated well by the c_0 coefficient. When using the MFCC we also need to use delta coeeficients as well. These can be computed in the same way by substracting the values of the previous frame from the values of the following frame. Combining the MFCC and its delta coefficients, we end up with a 16 element vector representing each frame of the signal which is quite effective in practice.