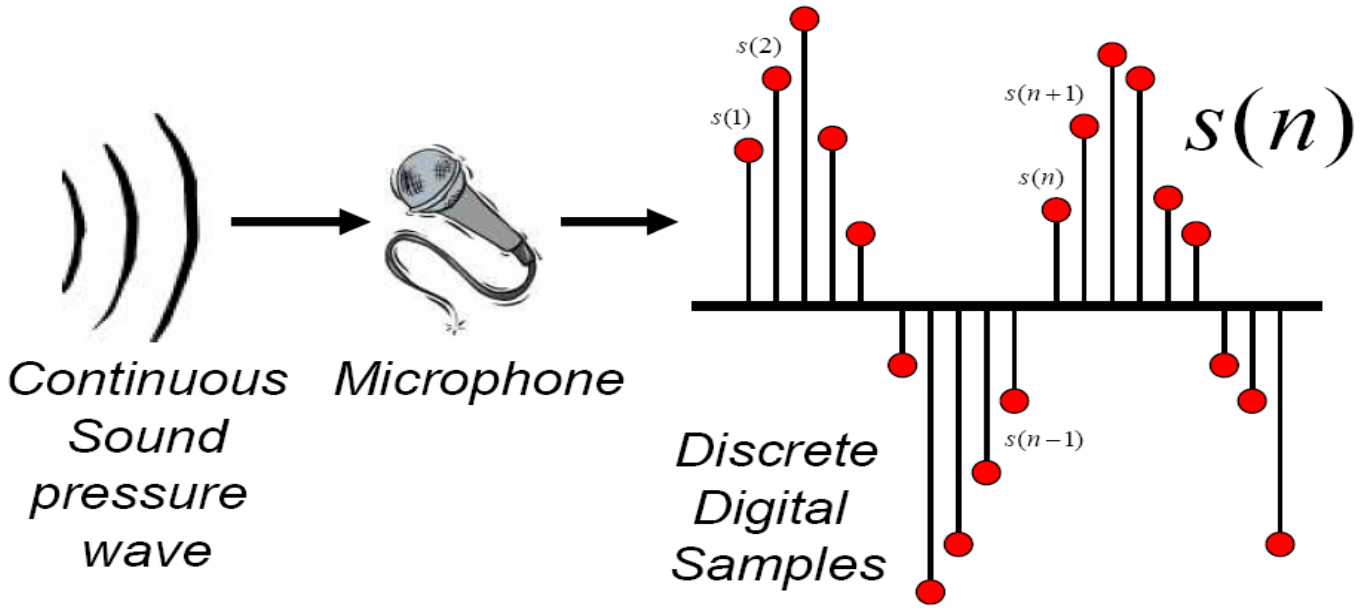
Discrete Representation of Signal

- Represent continuous signal into discrete form.

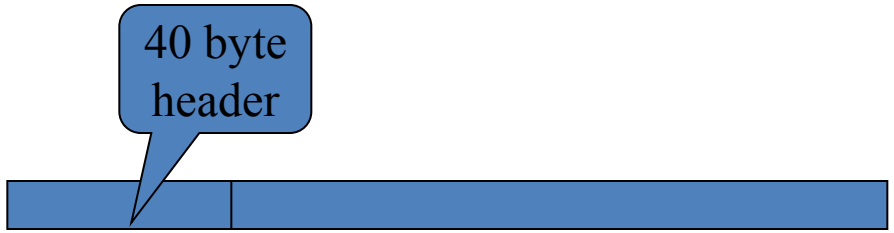


Digitizing the signal (A-D)

- **Sampling**:
  - measuring amplitude of signal at time t
  - 16,000 Hz (samples/sec) Microphone ("Wideband"):
  - 8,000 Hz (samples/sec) Telephone
  - Why?
    - Need at least 2 samples per cycle
    - max measurable frequency is half sampling rate
    - Human speech < 10,000 Hz, so need max 20K
    - Telephone filtered at 4K, so 8K is enough

Digitizing Speech (II)

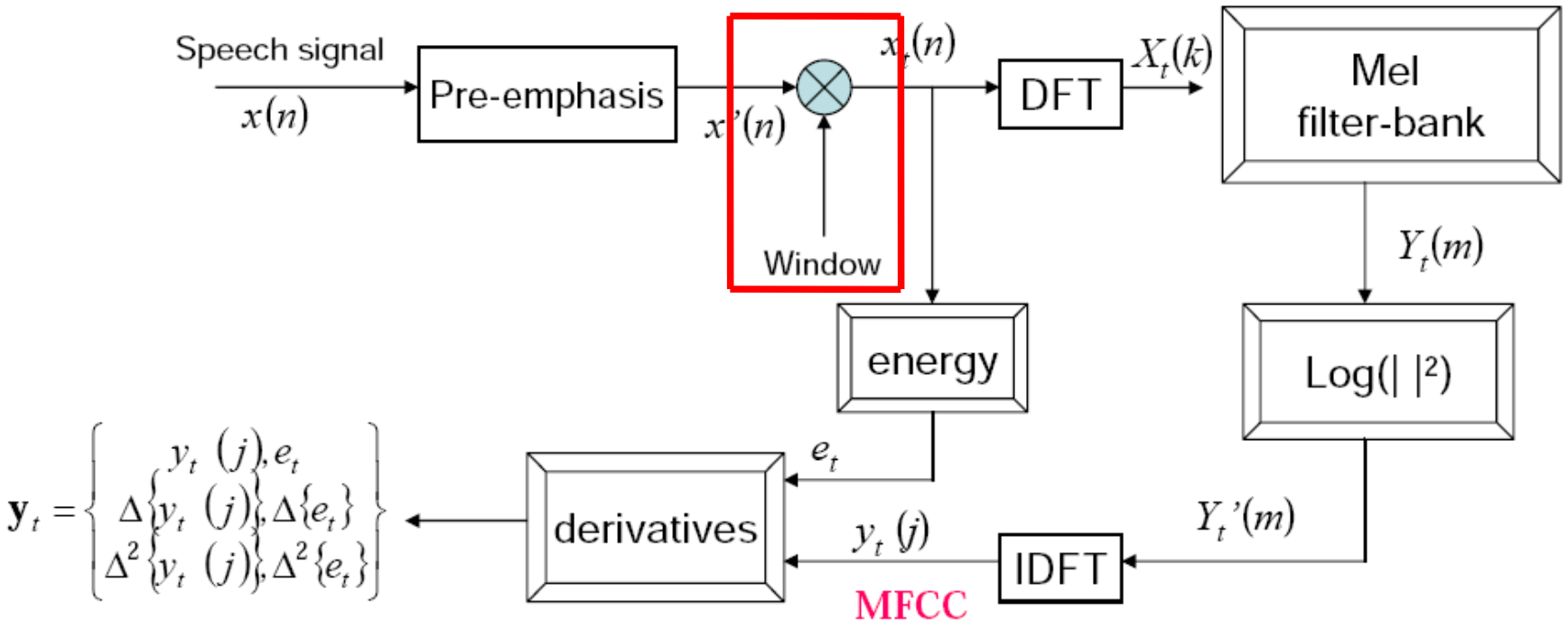
- **Quantization**
  - Representing real value of each amplitude as integer
  - 8-bit (-128 to 127) or 16-bit (-32768 to 32767)
- **Formats**:
  - 16 bit PCM
  - 8 bit mu-law; log compression
- LSB (Intel) vs. MSB (Sun, Apple)
- **Headers**:
  - Raw (no header)
  - Microsoft wav →
  - Sun .au



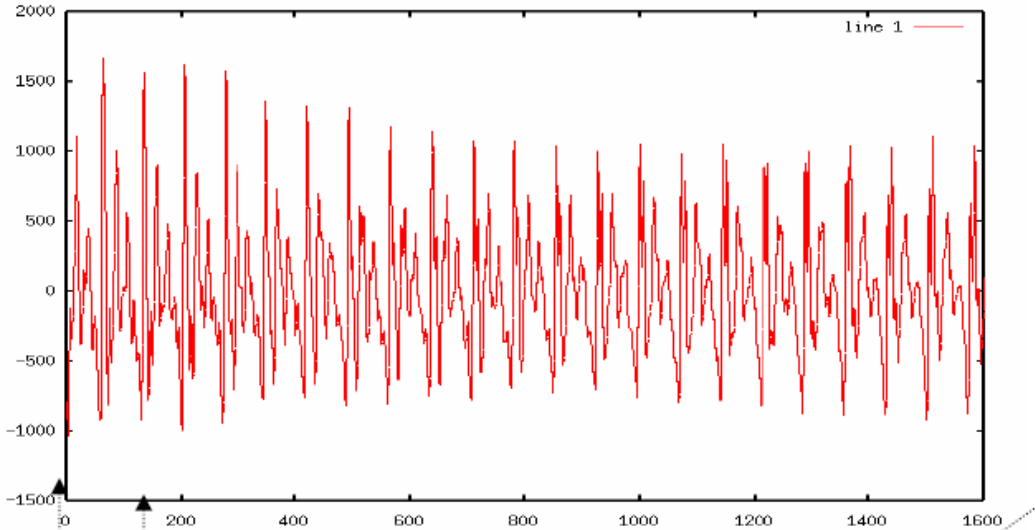
Discrete Representation of Signal

- Byte swapping
  - Little-endian vs. Big-endian
- Some audio formats have headers
  - Headers contain meta-information such as sampling rates, recording condition
  - Raw file refers to 'no header'
  - Example: Microsoft wav, Nist sphere
- Nice sound manipulation tool: sox.
  - change sampling rate
  - convert speech formats

Mel-**F**requency **C**epstral **C**oefficients (MFCC)

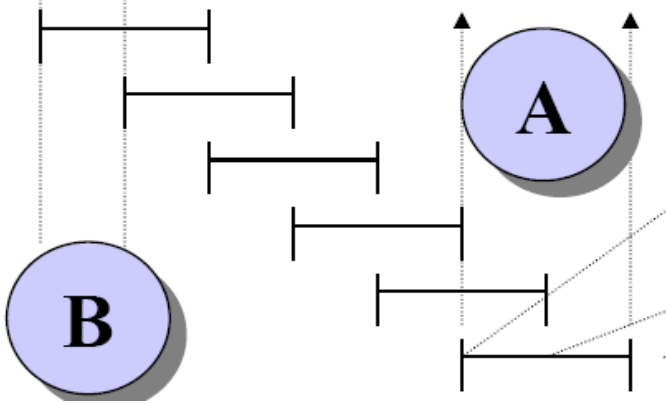


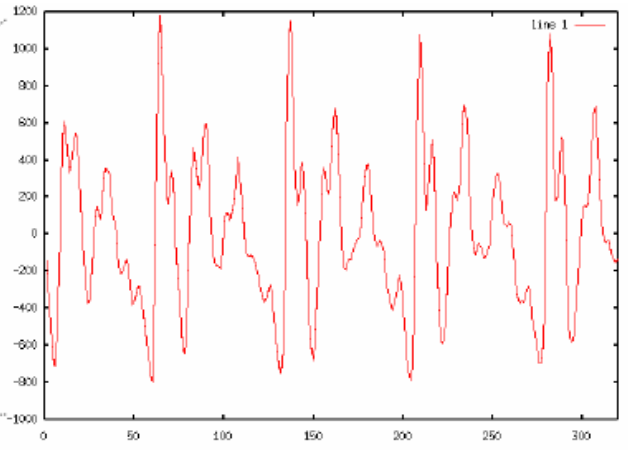
Windowing



A ~ 20 – 25 ms

B ~ 10 ms





Slide from Bryan Pellom

Windowing

- Why divide speech signal into successive overlapping frames?
  - Speech is not a stationary signal; we want information about a small enough region that the spectral information is a useful cue.
- Frames
  - Frame size: typically, 10-25ms
  - Frame shift: the length of time between successive frames, typically, 5-10ms

Common window shapes

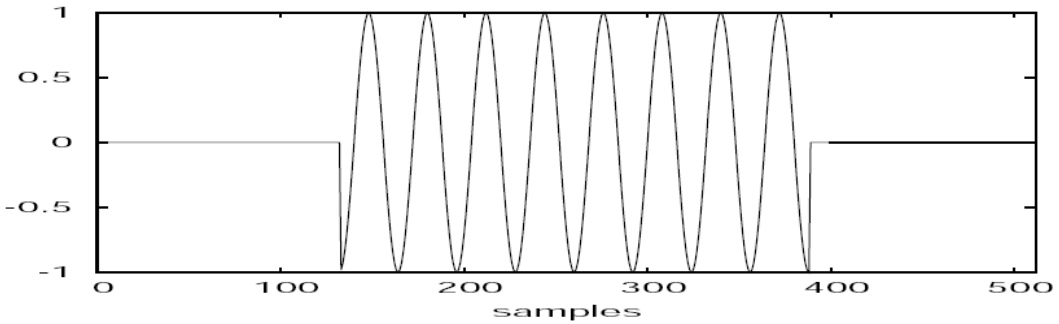
- Rectangular window:

$$w[n] = \begin{cases} 1 & 0 \leq n \leq L-1 \\ 0 & \text{otherwise} \end{cases}$$

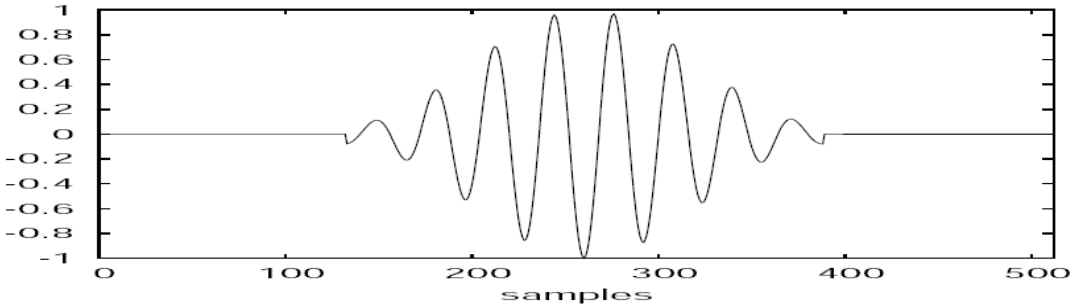
- Hamming window

$$w[n] = \begin{cases} 0.54 - 0.46\cos\left(\frac{2\pi n}{L-1}\right) & 0 \leq n \leq L-1 \\ 0 & \text{otherwise} \end{cases}$$

Window in time domain

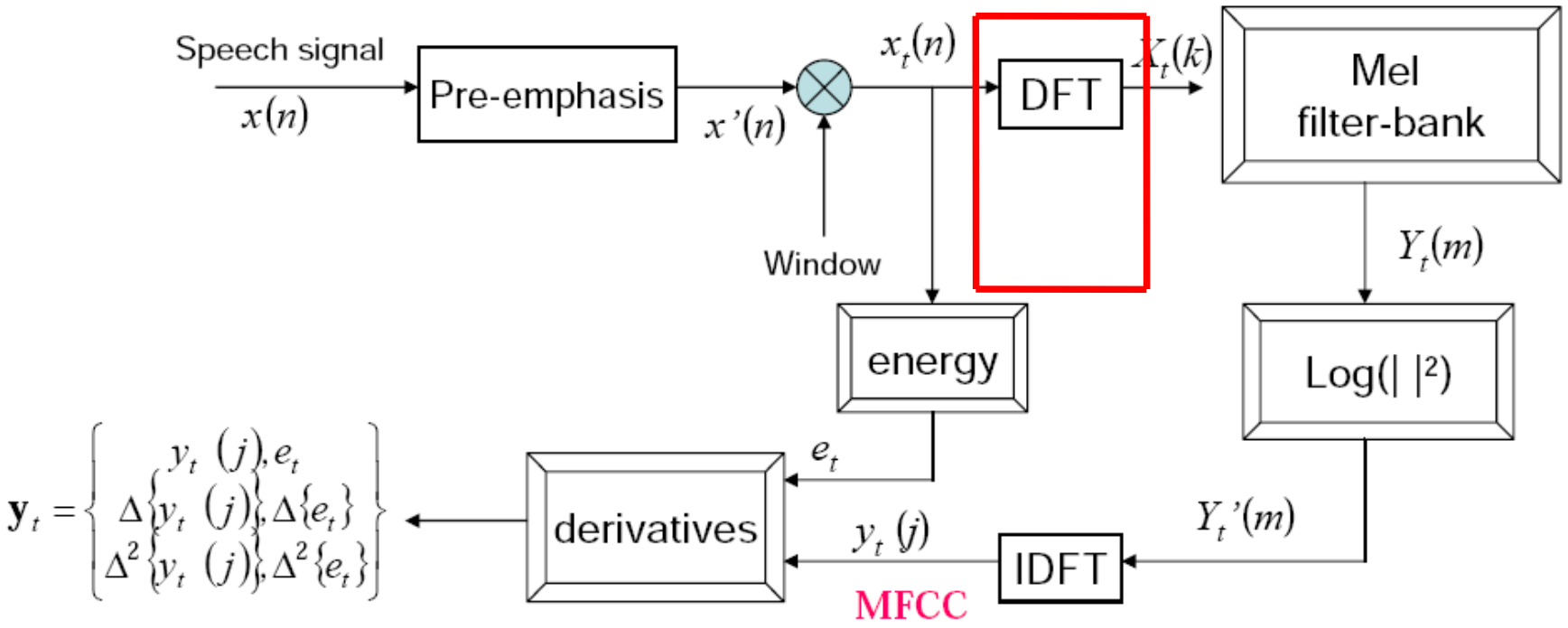


(a) Rectangular window



(c) Hamming window

MFCC



Discrete Fourier Transform

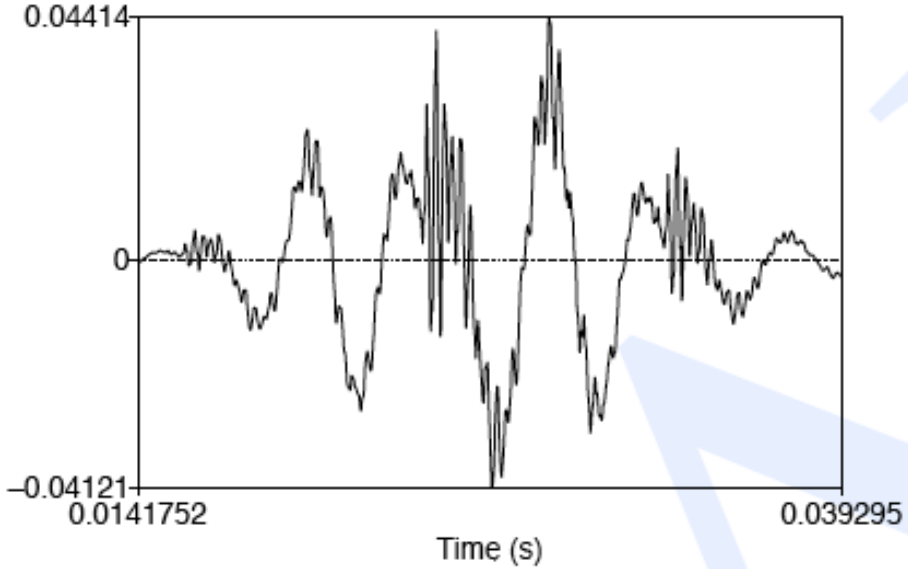
- Input:
  - Windowed signal x[n], n=0 … N-1
- Output:
  - For each of N discrete frequency bands
  - A complex number X[k] representing magnidue and phase of that frequency component in the original signal
- Discrete Fourier Transform (DFT)

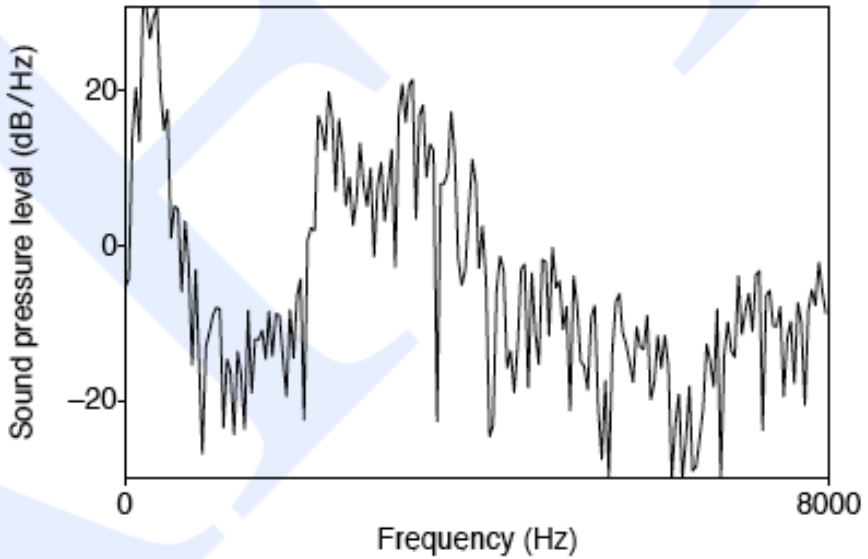
$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\frac{\pi}{N}kn}$$

- Standard algorithm for computing DFT:
  - Fast Fourier Transform (FFT) with complexity N*log(N)
  - In general, choose $N=2^n$, e.g 256, 512 or 1024

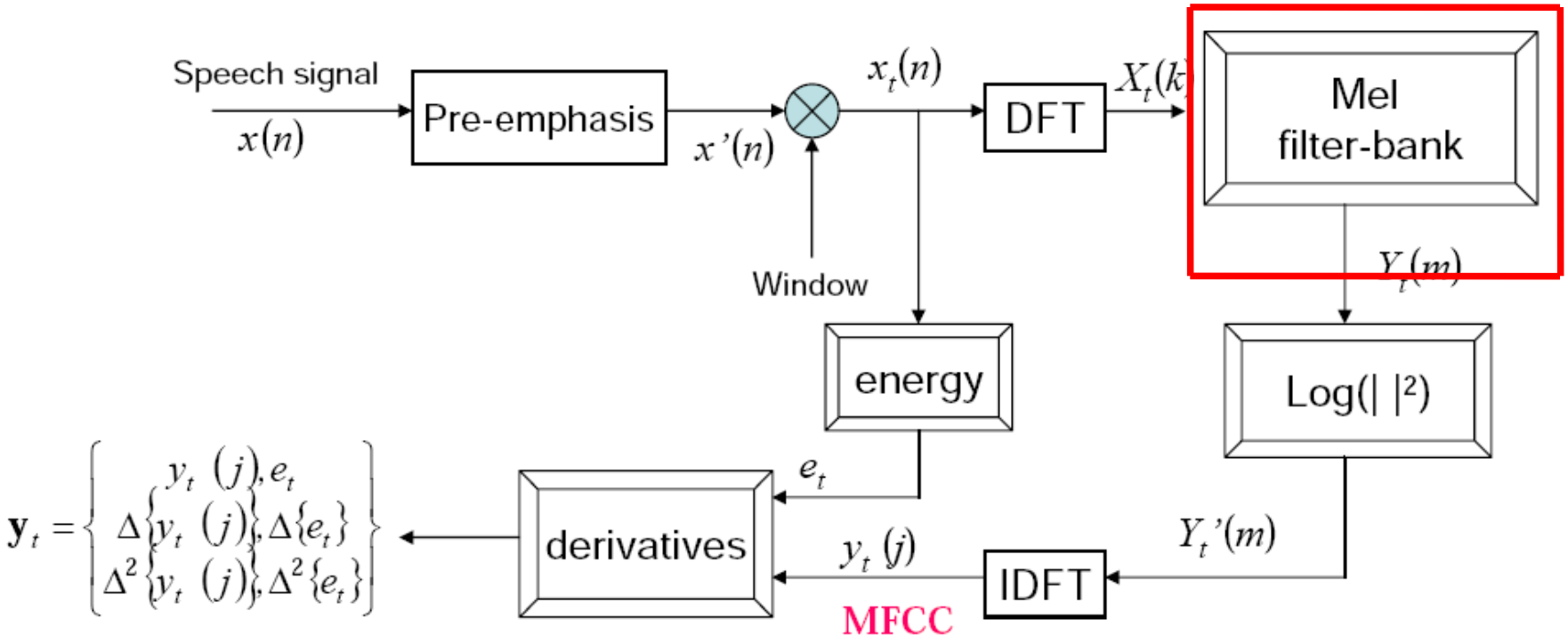
Discrete Fourier Transform computing a **spectrum**

- A 24 ms Hamming-windowed signal
  - And its spectrum as computed by DFT



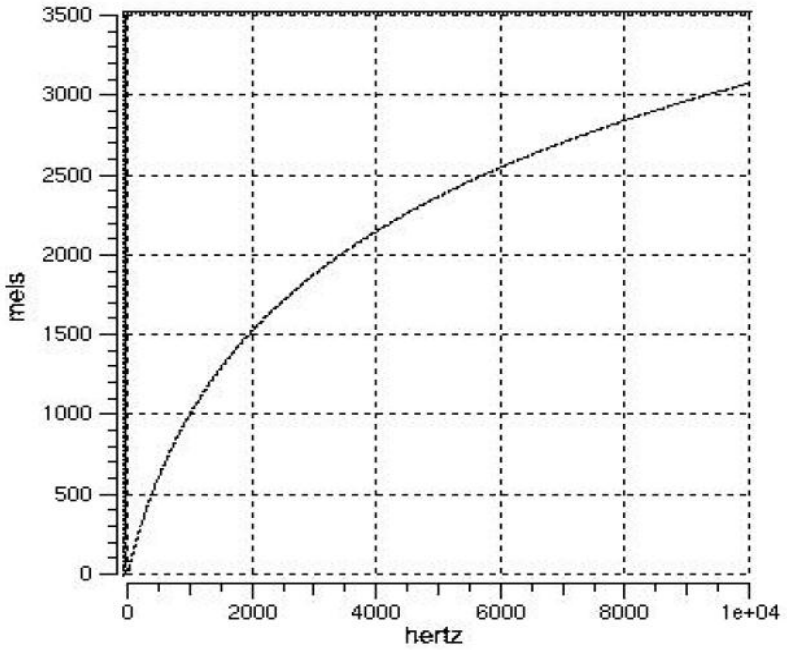


MFCC



Mel-scale

- Human hearing is not equally sensitive to all frequency bands
- Less sensitive at higher frequencies, roughly > 1000 Hz
- I.e. human perception of frequency is non-linear:



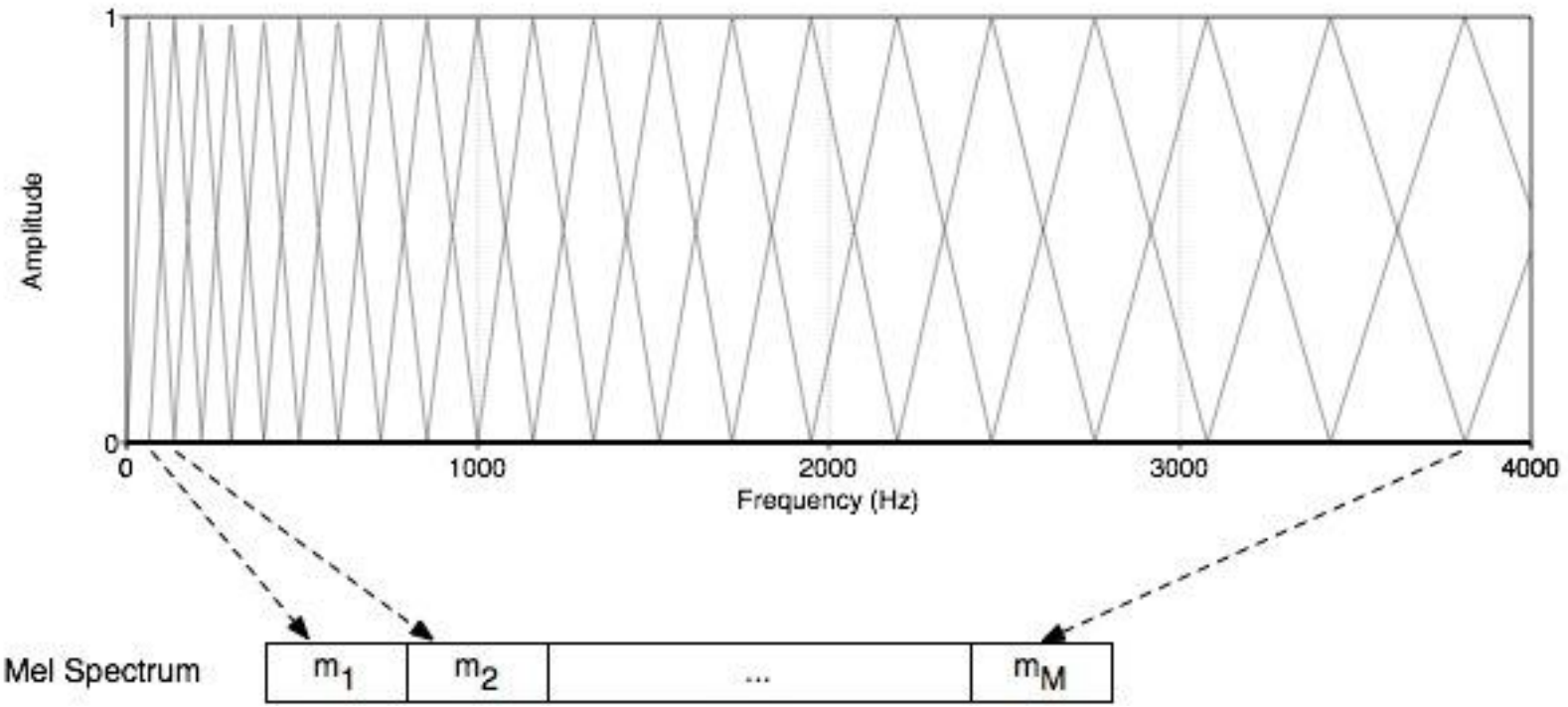
Mel-scale

- Mel-scale is approximately linear below 1 kHz and logarithmic above 1 kHz
- Definition:

$$Mel(f) = 2595 \log_{10}(1 + \frac{f}{700})$$

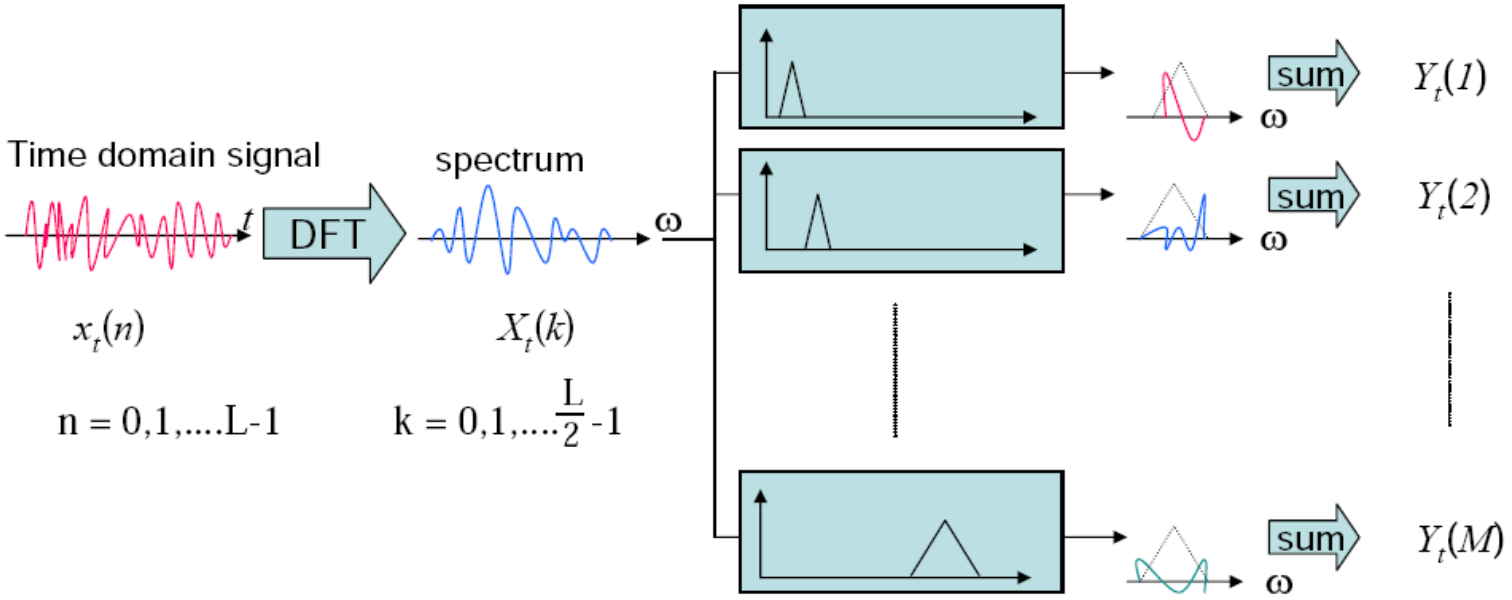
Mel Filter Bank Processing

- Mel Filter bank
  - Uniformly spaced before 1 kHz
  - logarithmic scale after 1 kHz

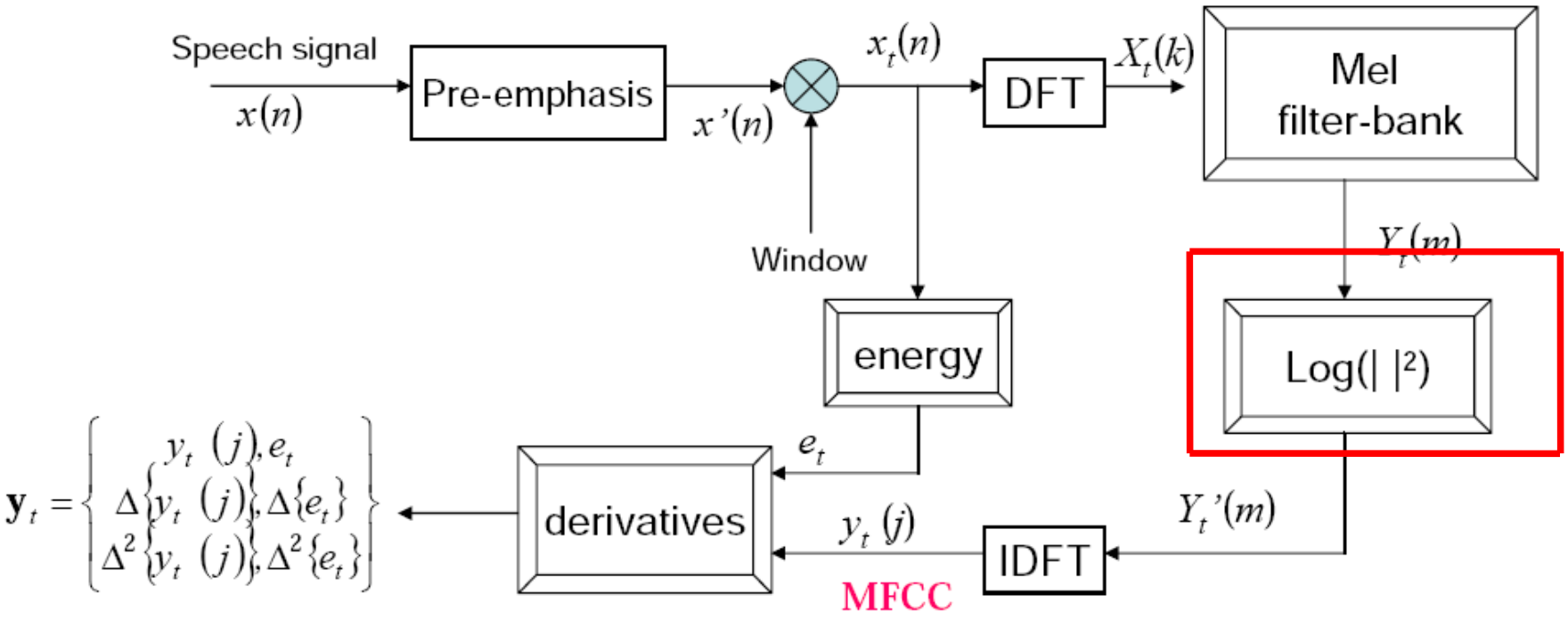


Mel-filter Bank Processing

- Apply the bank of filters according Mel scale to the spectrum
- Each filter output is the sum of its filtered spectral components

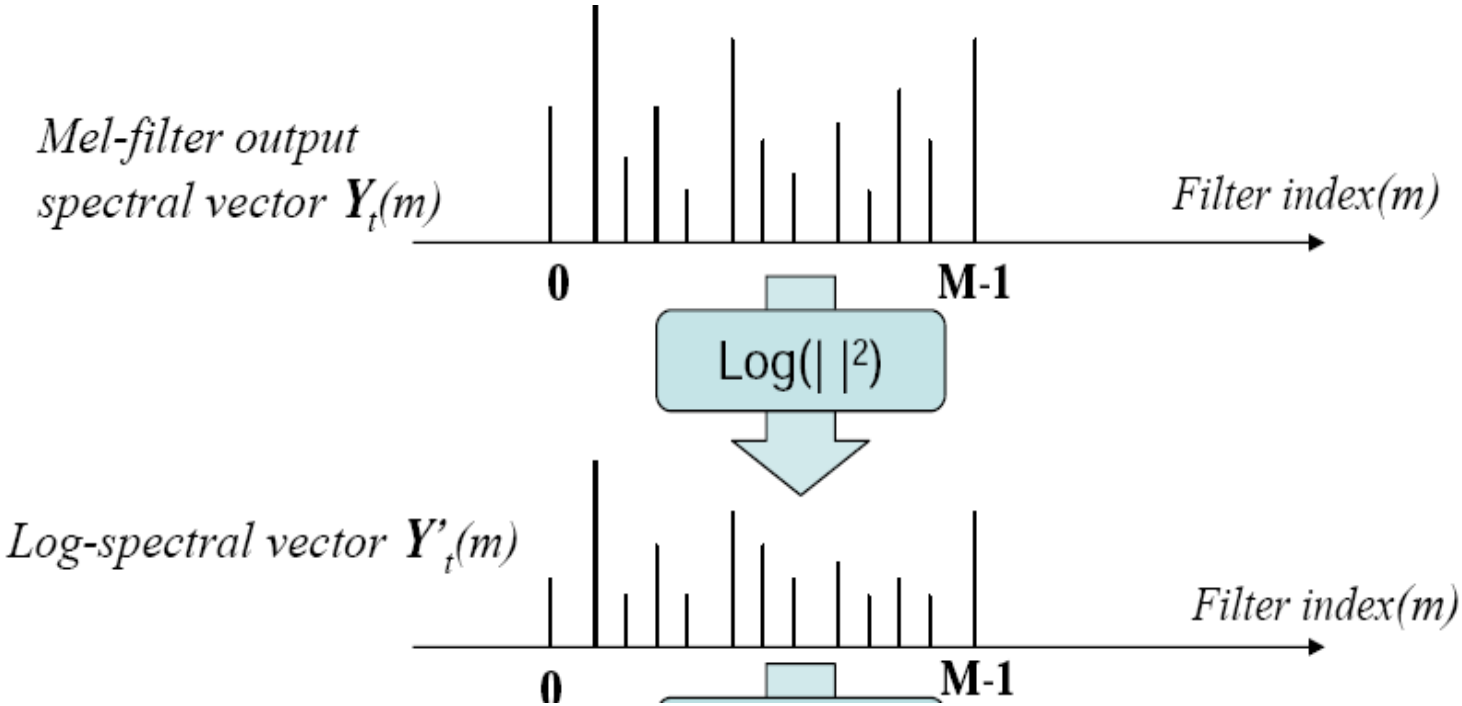


MFCC



Log energy computation

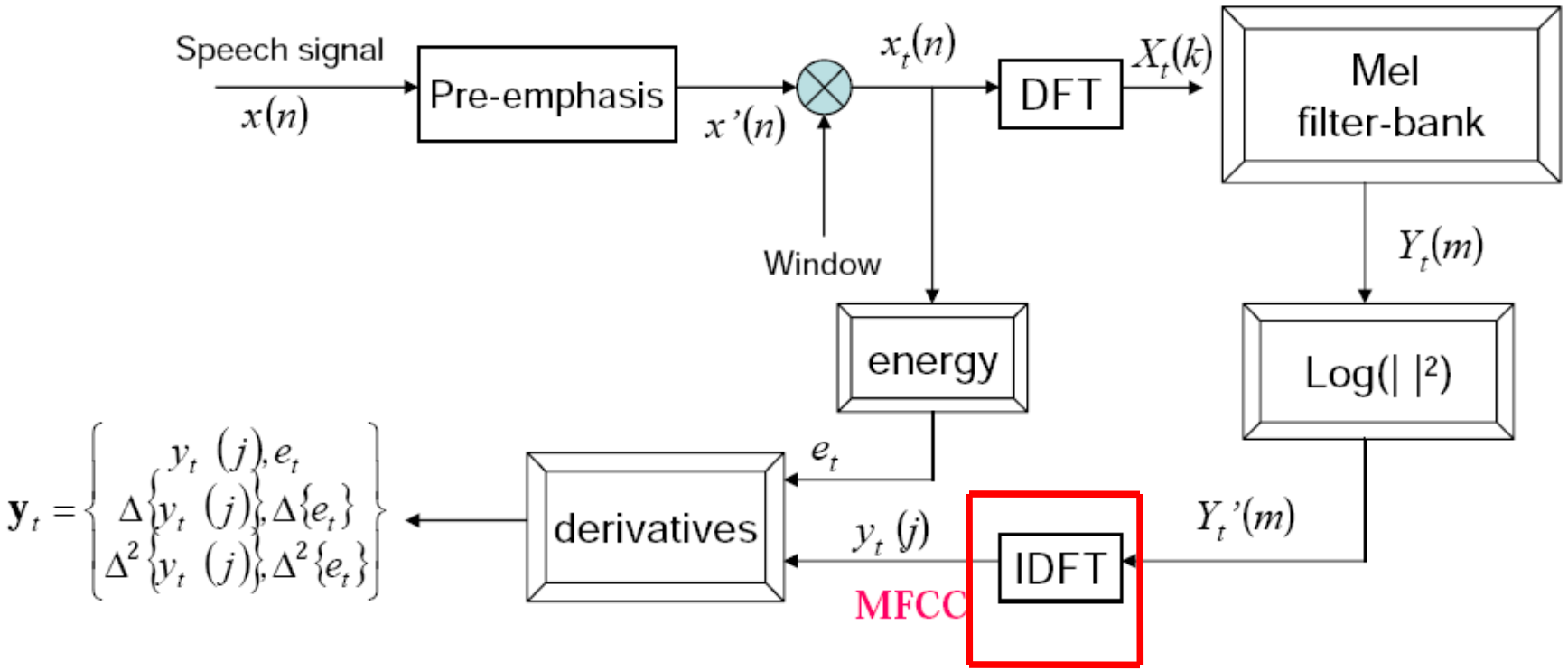
- Compute the logarithm of the square magnitude of the output of Mel-filter bank



Log energy computation

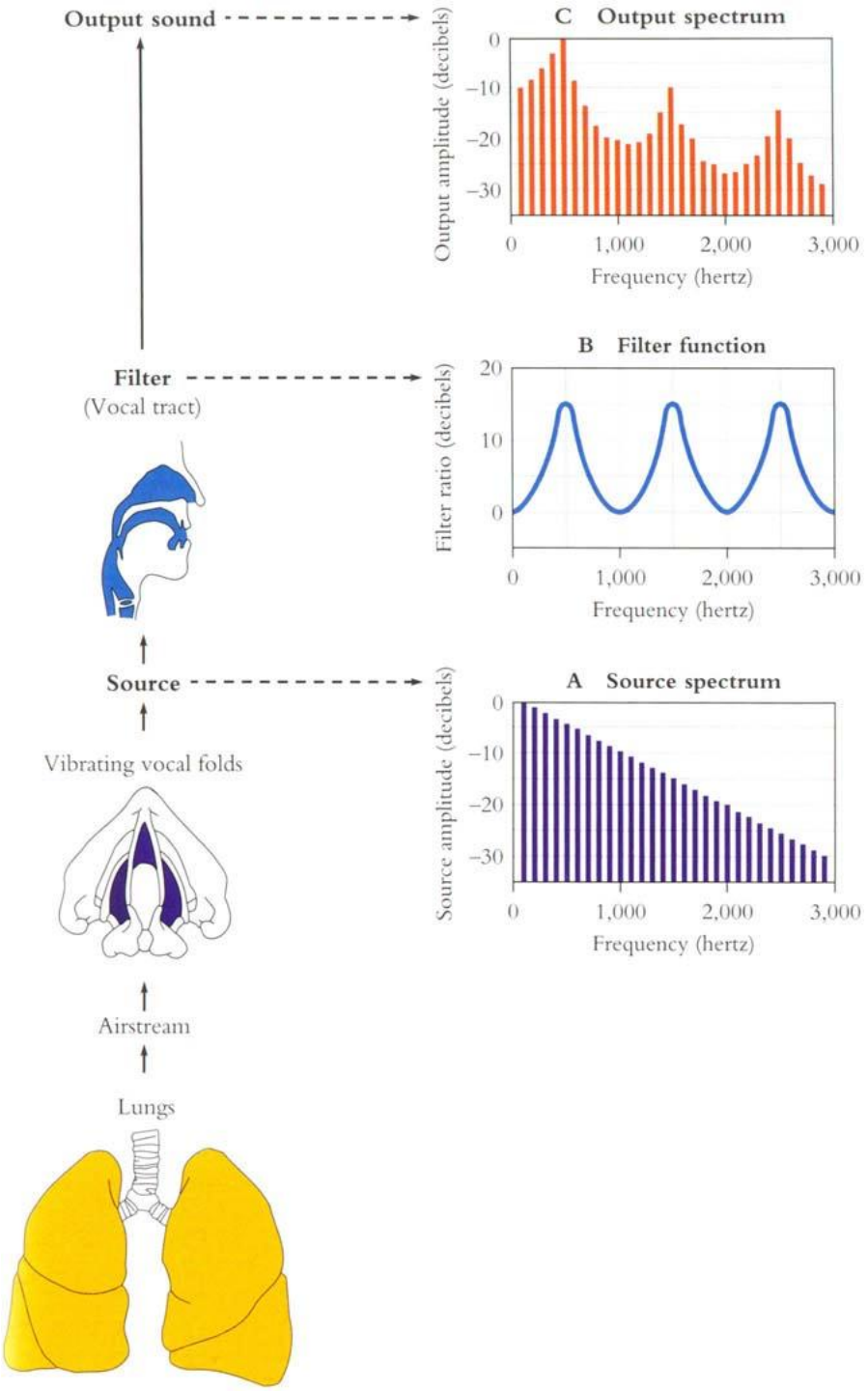
- Why log energy?
  - Logarithm compresses dynamic range of values
    - Human response to signal level is logarithmic
    - humans less sensitive to slight differences in amplitude at high amplitudes than low amplitudes
  - Makes frequency estimates less sensitive to slight variations in input (power variation due to speaker's mouth moving closer to mike)
  - Phase information not helpful in speech

MFCC



The Cepstrum

- One way to think about this
  - Separating the **source** and **filter**
  - Speech waveform is created by
    - A glottal source waveform
    - Passes through a vocal tract which because of its shape has a particular filtering characteristic
- Articulatory facts:
  - The vocal cord vibrations create harmonics
  - The mouth is an amplifier
  - Depending on shape of oral cavity, some harmonics are amplified more than others

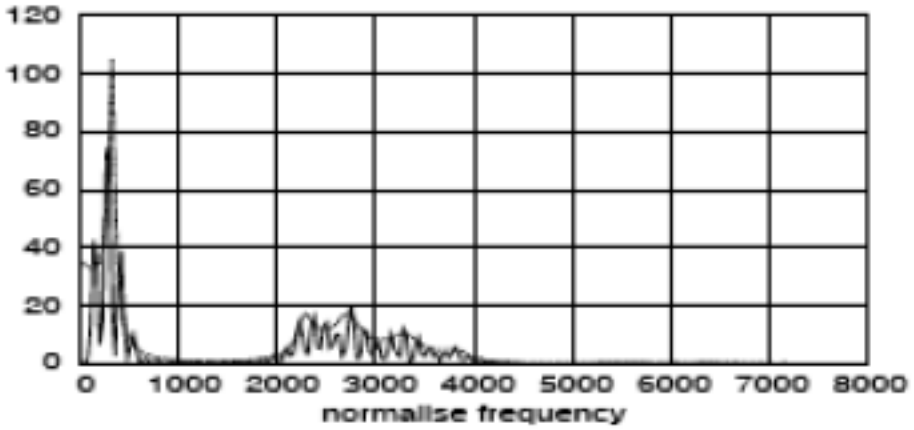


We care about the filter not the source

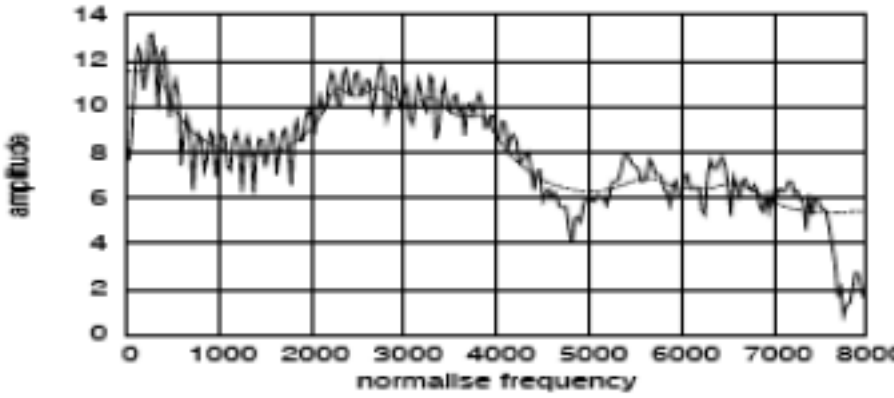
- Most characteristics of the source
  - F0
  - Details of glottal pulse
- Don't matter for phone detection
- What we care about is the **filter**
  - The exact position of the articulators in the oral tract
- So we want a way to separate these
  - And use only the filter function

The Cepstrum

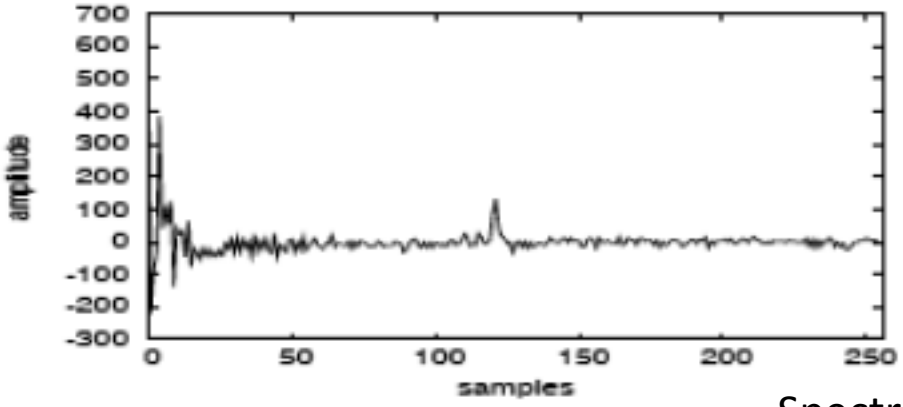
- The spectrum of the log of the spectrum



Spectrum

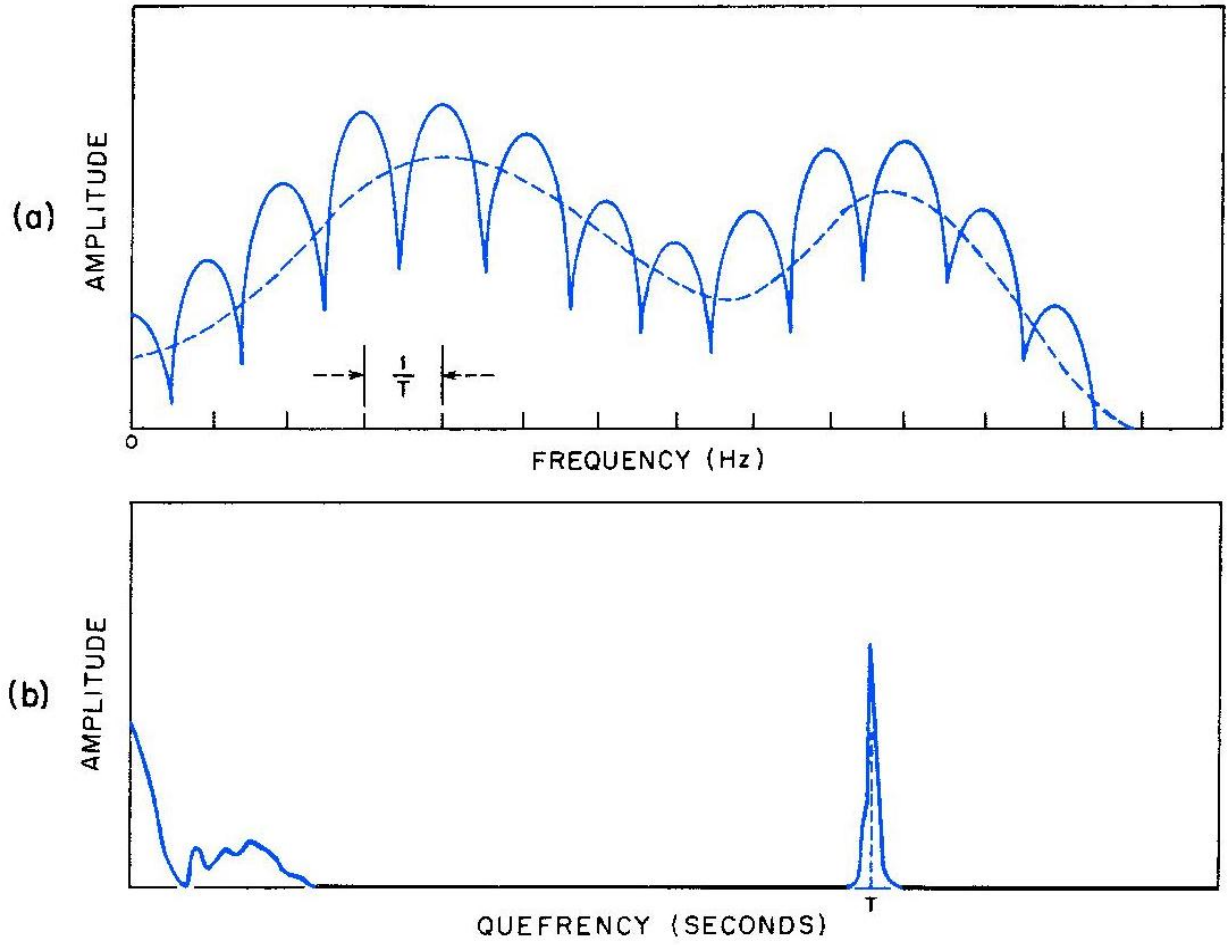


Log spectrum



Spectrum of log spectrum

Thinking about the Cepstrum



Mel Frequency cepstrum

- The cepstrum requires Fourier analysis
- But we're going from frequency space back to time
- So we actually apply inverse DFT

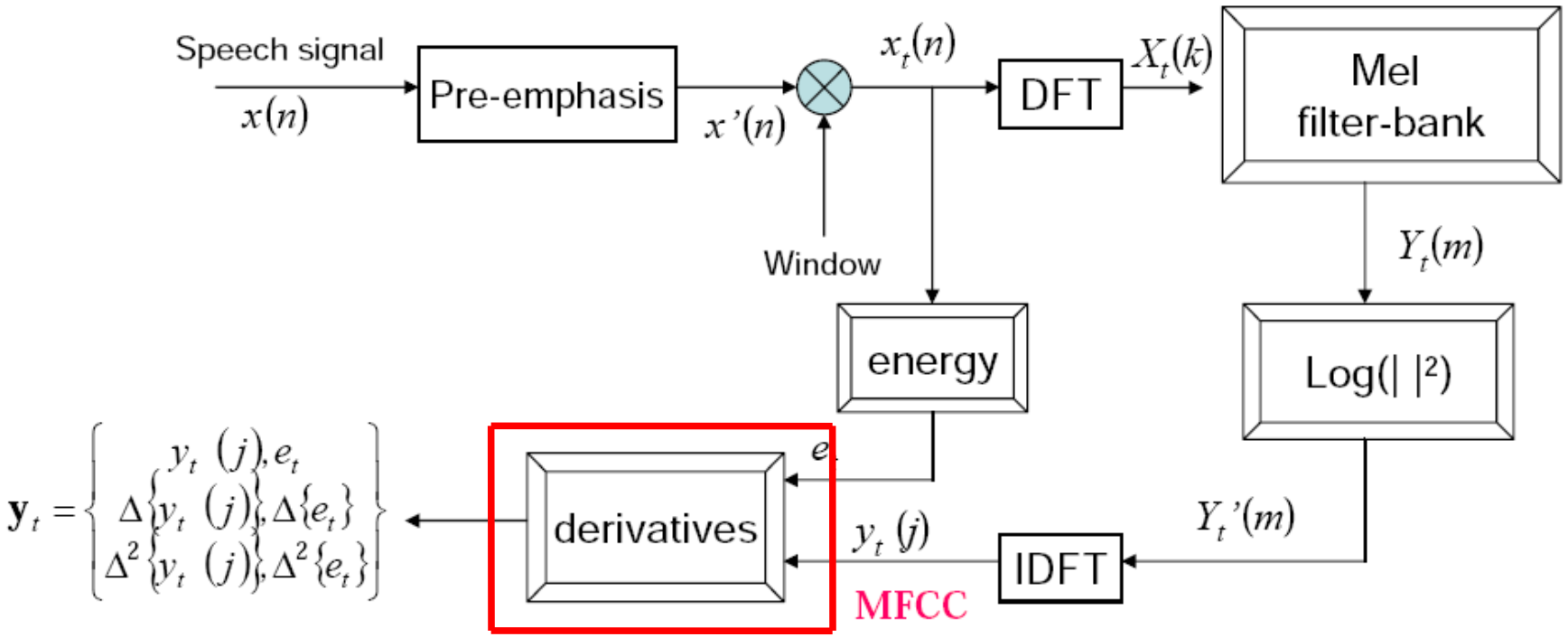
$$y_t[k] = \sum_{m=1}^{M} \log(|Y_t(m)|) \cos(k(m-0.5)\frac{\pi}{M}),\ \text{k=0,...,J}$$

- Details for signal processing gurus: Since the log power spectrum is real and symmetric, inverse DFT reduces to a Discrete Cosine Transform (DCT)

Another advantage of the Cepstrum

- DCT produces highly **uncorrelated** features
- We'll see when we get to acoustic modeling that these will be much easier to model than the spectrum
  - Simply modelled by linear combinations of Gaussian density functions with diagonal covariance matrices
- In general we'll just use the first 12 cepstral coefficients (we don't want the later ones which have e.g. the F0 spike)

MFCC

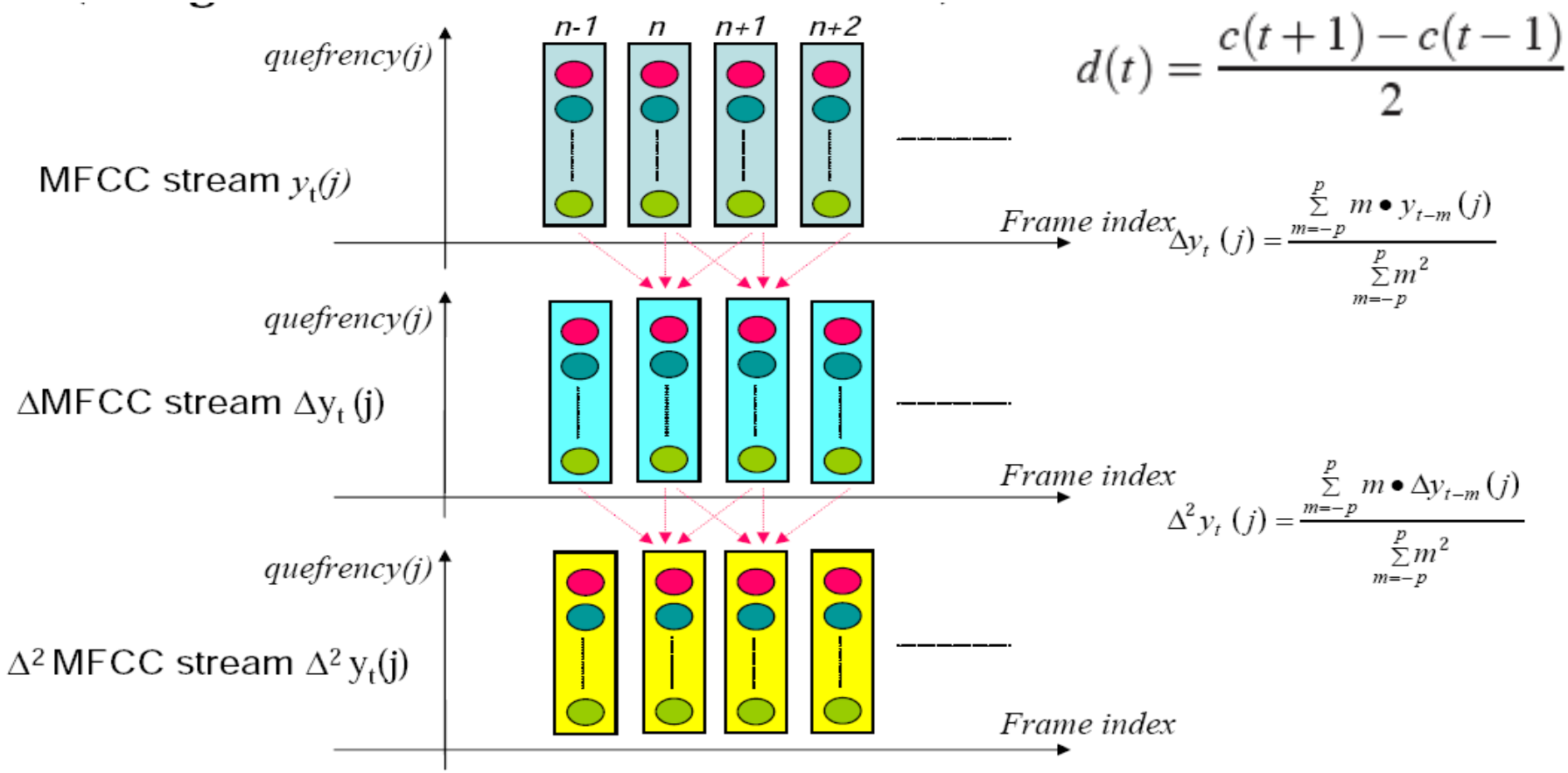


Dynamic Cepstral Coefficient

- The cepstral coefficients do not capture energy
- So we add an energy featur $Energy = \sum_{t=t_1}^{t_2} x^2[t]$
- Also, we know that speech signal is not constant (slope of formants, change from stop burst to release).
- So we want to add the changes in features (the slopes).
- We call these **delta** features
- We also add **double-delta** acceleration features

Delta and double-delta

- Derivative: in order to obtain temporal information



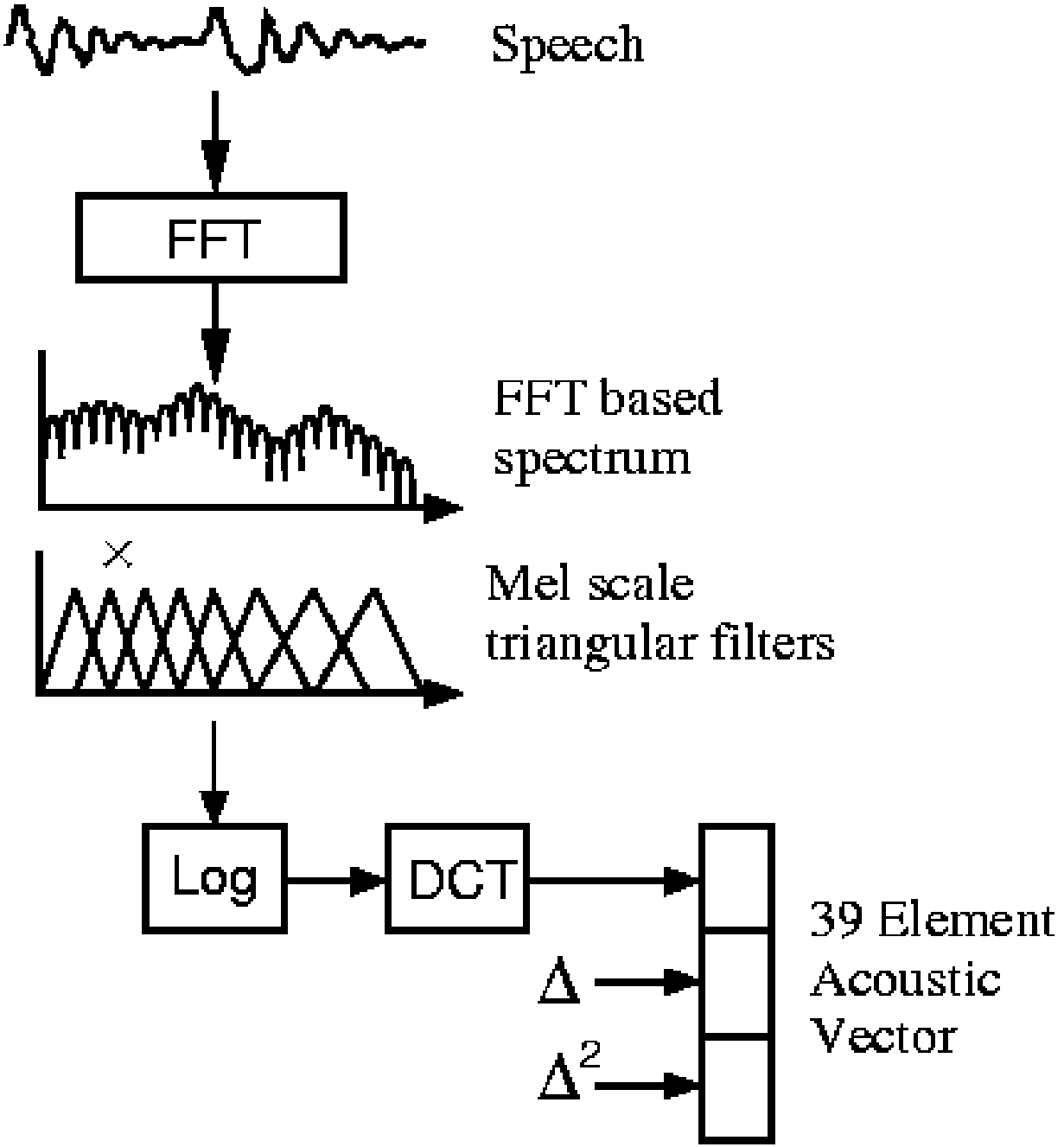
$$d(t) = \frac{c(t+1) - c(t-1)}{2}$$

$$\Delta y_t(j) = \frac{\sum_{m=-p}^{p} m \bullet y_{t-m}(j)}{\sum_{m=-p}^{p} m^2}$$

$$\Delta^2 y_t(j) = \frac{\sum_{m=-p}^{p} m \bullet \Delta y_{t-m}(j)}{\sum_{m=-p}^{p} m^2}$$

Typical MFCC features

- Window size: 25ms
- Window shift: 10ms
- Pre-emphasis coefficient: 0.97
- MFCC:
  - 12 MFCC (mel frequency cepstral coefficients)
  - 1 energy feature
  - 12 delta MFCC features
  - 12 double-delta MFCC features
  - 1 delta energy feature
  - 1 double-delta energy feature
- Total 39-dimensional features

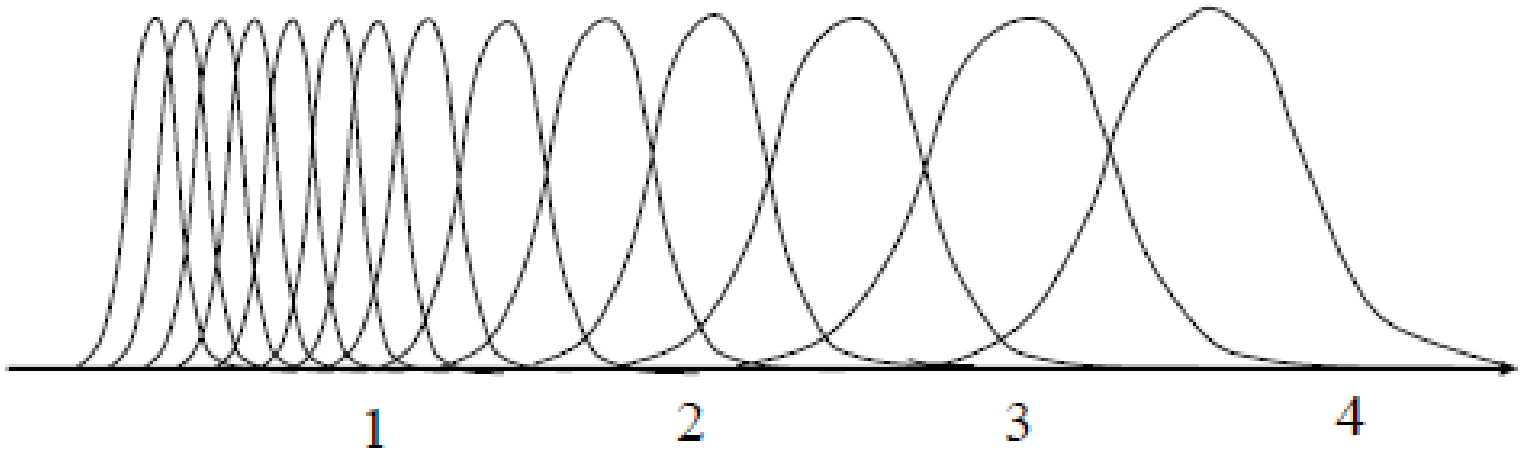


Why is MFCC so popular?

- Efficient to compute
- Incorporates a perceptual Mel frequency scale
- Separates the source and filter
- IDFT(DCT) decorrelates the features
  - Improves diagonal assumption in HMM modeling
- Alternative
  - Linear Prediction
  - Perceptual Linear Prediction (PLP)

Alternative front-end analyses

- Mel-scale filter-bank – replaces DFT, log| |, and 'mel-scale binning'



Speech Processing in Noisy Environment

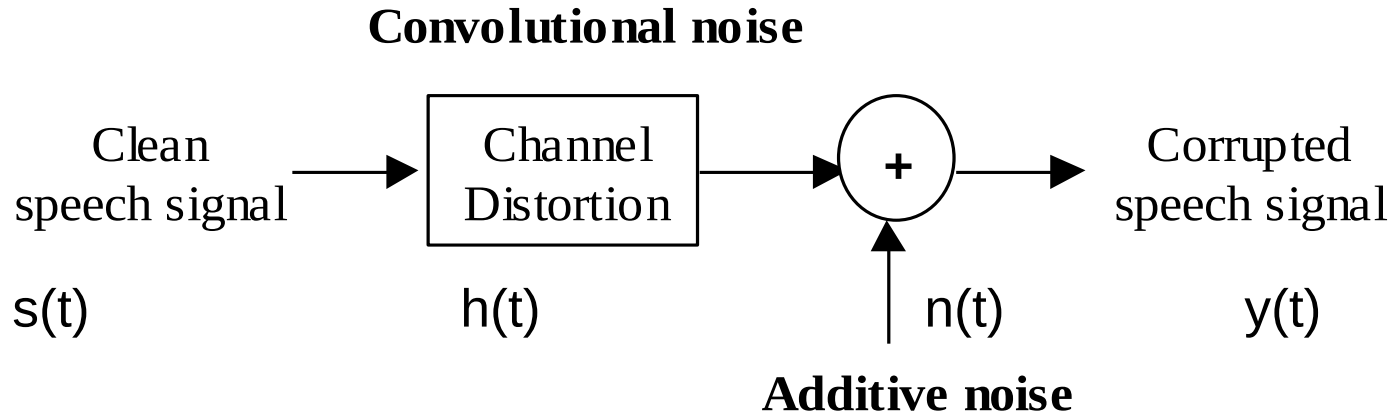
Types of Noise

- Additive Noise
  - Degradation of the signal due to ambient acoustic noise
  - Extent of problem will depend on Signal-to-Noise Ratio (SNR), but also on how ‘speech like’ the noise is
  - Corresponds to addition of speech and noise signals in the time-domain

- Convolutive Noise
  - Changes in the speech signal due to changing room acoustics, changes in microphone, communication channel, etc
  - Corresponds to convolution with a speech signal in the time-domain

Noise Corruption

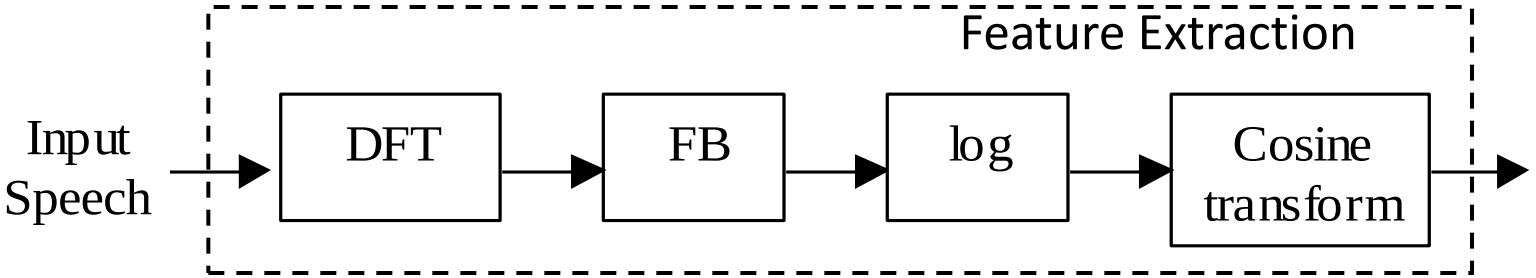
- A block diagram of noise corruption



$$y(t) = s(t) * h(t) + n(t)$$

Interaction with Front-End Processing

- How different types of noise impact on the recognition process ?.. need to understand how they are transformed by front-end processing (i.e. feature extraction block)



Type of Noise	Time Domain	DFT	Filter Bank	log	Cosine Transform
Additive	+	+	~ +	? ~ max	?
Convolutive	*	x	~ x	~ +	~ +

Convolutive Noise Compensation

- Convolutive noise turns out to be the easiest to deal with
- Manifested as addition of constant, or slowly varying, log power spectrum or MFCC vector
- Simplest solution is cepstral mean subtraction (in MFCC/cepstral domain) or spectral mean subtraction (in log power spectrum domain)

Cepstral Mean Subtraction (CMS)

- Cepstral Mean Subtraction (CMS)
  - mean (over a num of frames) subtraction
  - lowpass filtering
  - eliminates communication channel spectral shaping

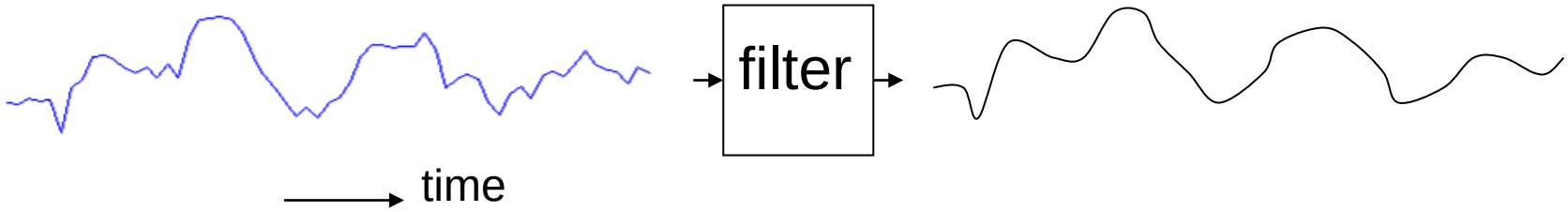
$$c_{CMS}(l;m) = c\ (l;m) - \text{avg}_k(c\ (k;m)), \quad m = 1,\ldots,N$$

RASTA Processing

- Vocal tract cannot move too slow or too fast

trajectory of parameter
as estimated from the signal

corrected trajectory



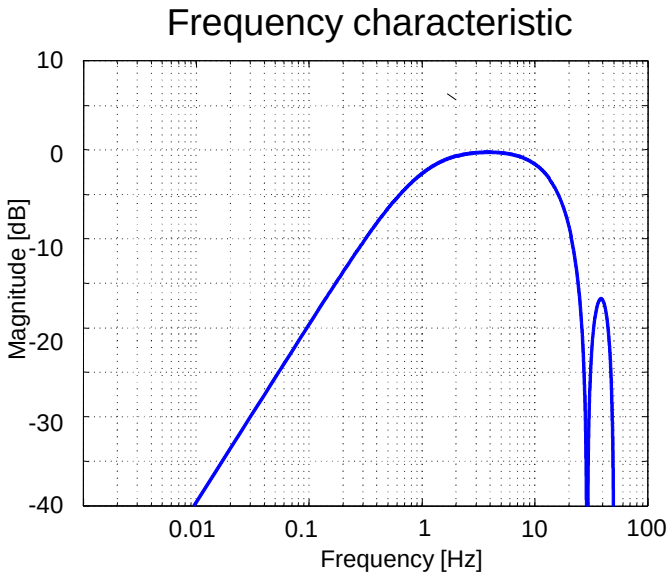
RASTA Processing

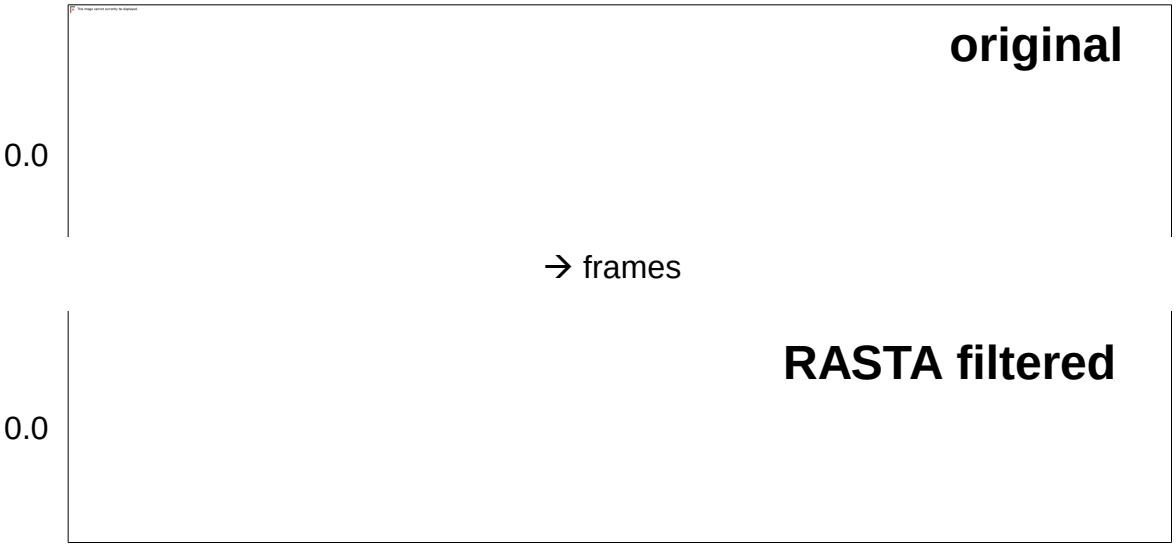
Approximated by IIR Filter:

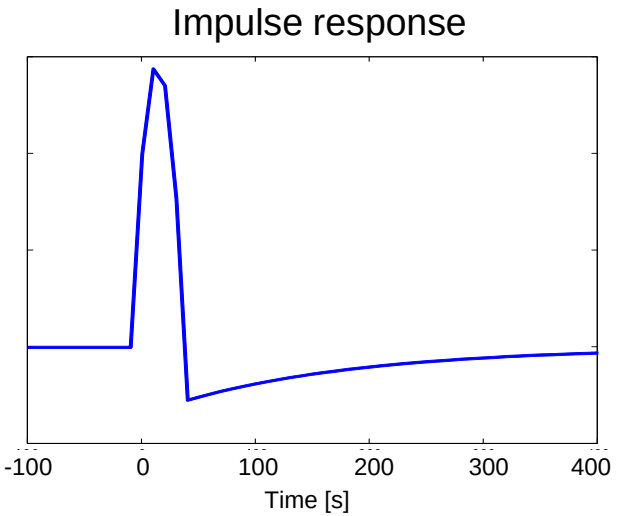
$$H(z) = \frac{0.2 + 0.1z^{-1} - 0.1z^{-3} - 0.2z^{-4}}{1 - 0.98z^{-1}}$$

RASTA filtering

- Filtering log filter bank output (or equivalently cepstral) temporal trajectories by band pass filter
- Remove slow changes to compensate for the channel effect (≈CMS over 0.5 sec. sliding window)
- Remove fast changes (> 25Hz) likely not caused by speaker with limited ability to quickly change vocal tract configuration







Additive Noise Compensation

- In the linear spectral domain, additive noise is still additive

- Therefore, noise compensation can be achieved by subtracting an estimate of the noise spectrum from the noisy-speech spectrum

- Coping with additive noise explicitly in the MFCC domain is more difficult, because of the effect of front-end analysis on ‘addition’

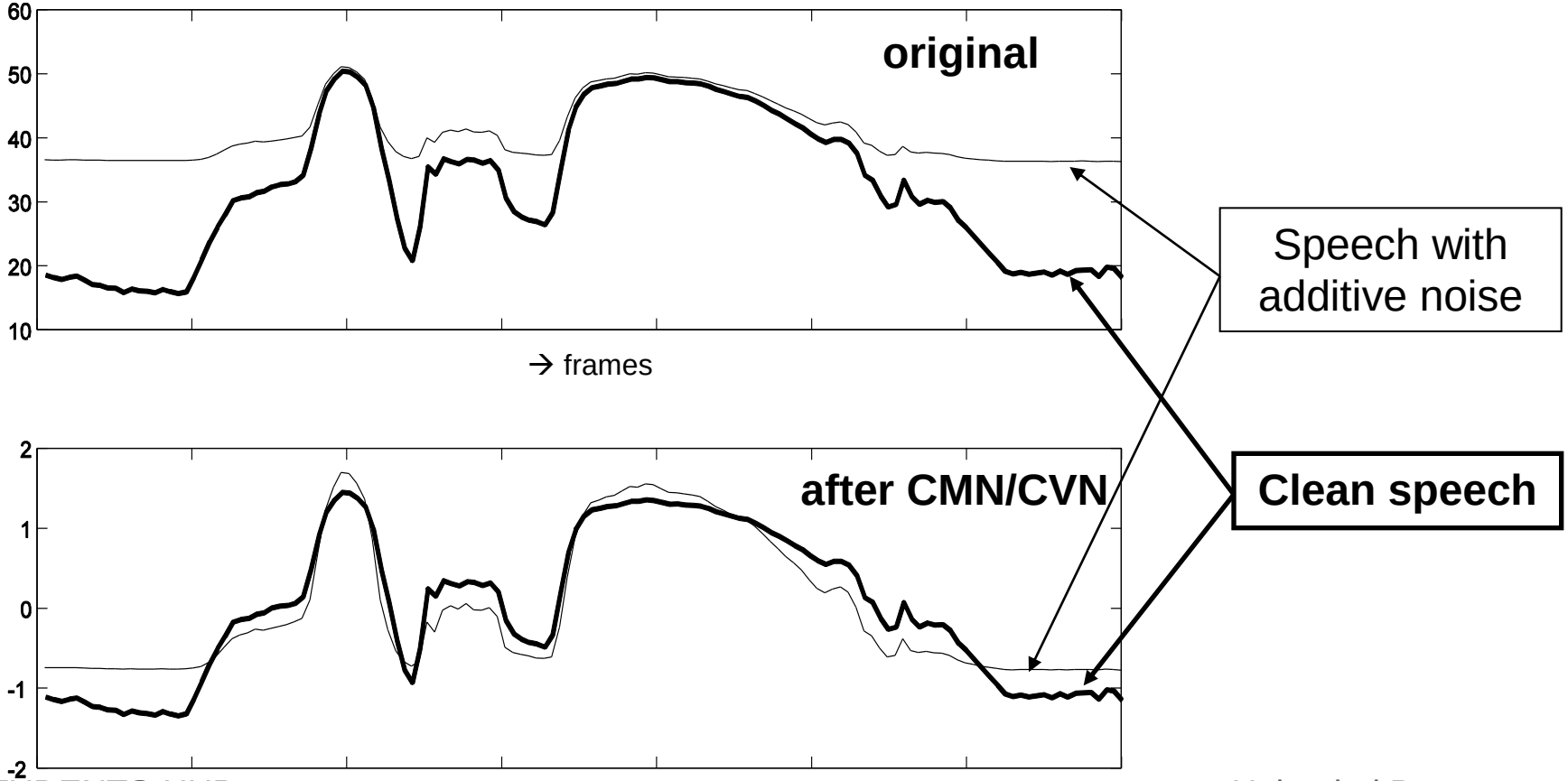
Mean and Variance Normalization

- Estimate mean and variance of cepstral feature vectors of an utterance
- For each cepstral vector (MFCC), subtract mean and divide by variance
- This normalize the features distribution to standard normal distribution (zero mean and unity variance).

Mean and Variance Normalization

•While convolutive noise causes the constant shift of cepstral coeff. temporal trajectories, noise additive in spectral domain fills valleys in the trajectories

•In addition to subtracting mean, trajectory can be normalized to unity variance (i.e. dividing by standard deviation) to compensate for his effect



```
window_size = 20; %20ms
shift = 10; %10ms
nfilts = 22;

[samples,fs,nbits] = wavread('C:\Users\abualsoud\Dropbox\Spoken Language
proc\experiments\sa1_tstDR2mgwt0.wav');

minfreq = 40;
maxfreq = fs/2;
window_size = floor((window_size/1000)*fs); % convert ms to number of samples
shift = floor((shift/1000)*fs);
overlap = window_size - shift;

window = hamming(window_size);



%y = buffer(s,window_size, overlap); % segmentation

num_samples = size(samples,1);

num_frames = floor(num_samples/(shift));
NFFT = 2^(ceil(log(window_size)/log(2)));
half = floor(NFFT/2)+1;
```

```
triangles = fft2melmx(half, fs, nfilts, 1, minfreq, maxfreq,0,1,0);

frame_num =0;
energy = zeros(1,num_frames);
y = zeros(num_frames,half);
fba = zeros(num_frames, nfilts);

  for sample_point =1:shift:num_frames*(shift)
      frame_num = frame_num+1;
      start =  sample_point;
      finish = sample_point+window_size-1;
      if finish > size(samples,1)
         break;
      end
      speech_frame = window .* samples(start:finish);
      %%%%%%%%%%%%
      temp = speech_frame'*speech_frame;
      energy(frame_num) = 10*log10(temp);
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      fftout = fft(speech_frame,NFFT );
      fftout = fftout(1:half);
      fftpower =  abs(fftout).^2;
      %%%%%%%%%
```

```
fftpowlog= log(fftpower);
    y(frame_num, :) = fftpowlog;

    fba(frame_num,:) = triangles*fftpowlog;
  end
  y = y';
  fba = fba';
  pcolor(fba(:,100:330))

  mfcc = DCT(fba,12);
  figure
  pcolor(mfcc(:,100:330))
  fba2 = IDCT(mfcc);
  figure
  pcolor(fba2(:,100:330))
  %%%%%%%%%%%%%%%%

  close all;
  %%%%%%%%%%%%%%%%
  figure
  plot(fba(:,100),'b');
  figure;   plot(fba2(:,100),'r');
  figure;   plot(mfcc(:,100),'k');
```