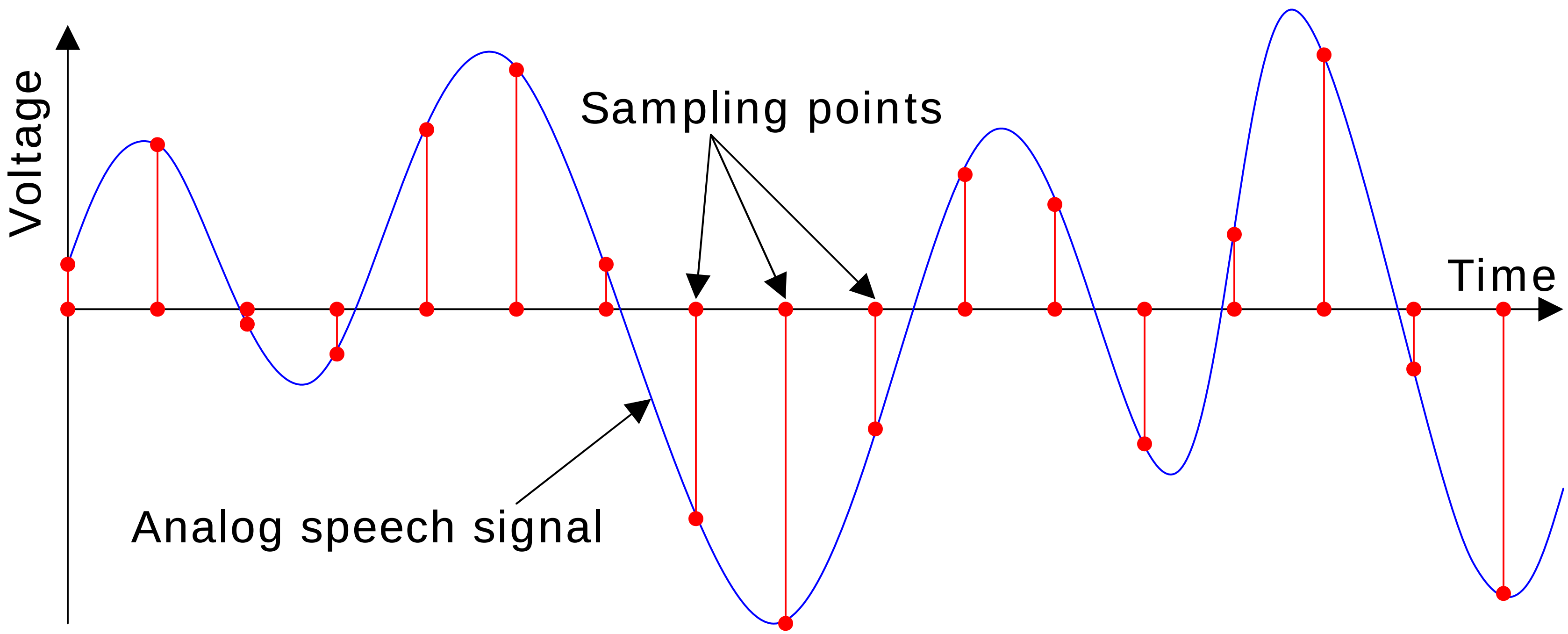
Feature Computation: Representing the Speech Signal

Bhiksha Raj and Rita Singh

A 30-minute crash course in signal processing

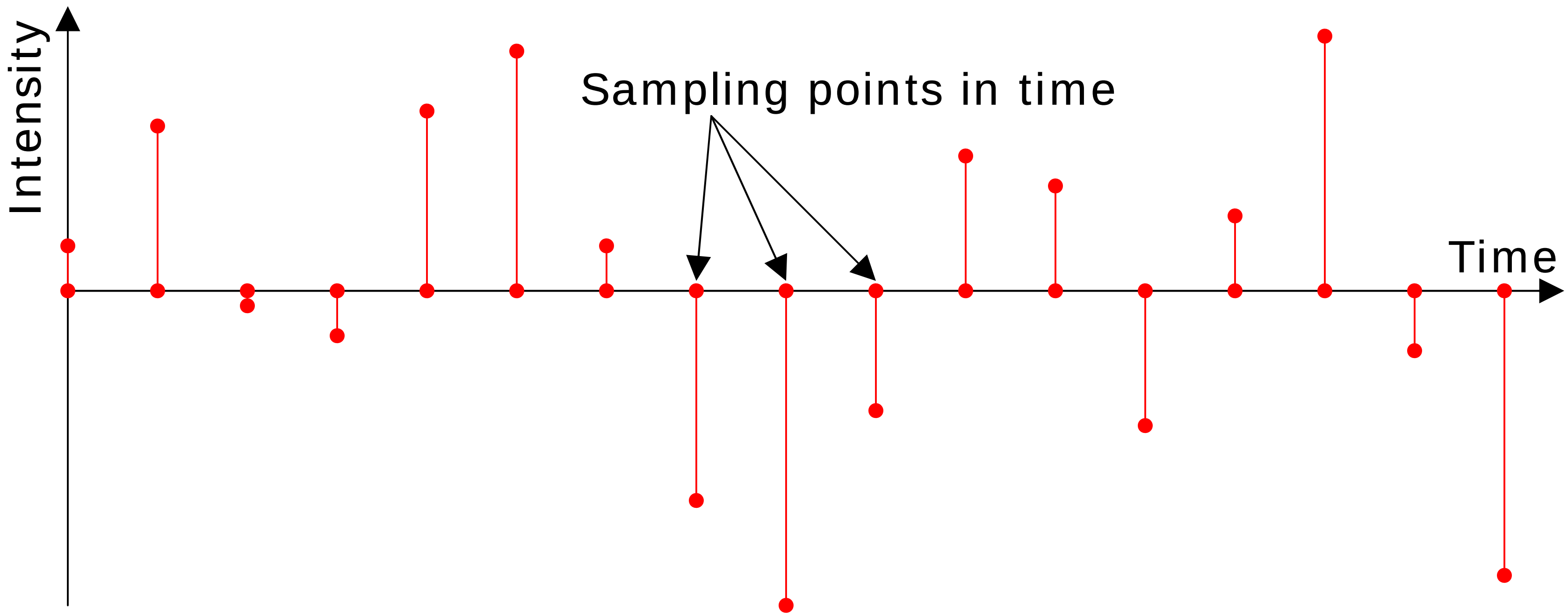
The Speech Signal: Sampling

- The analog speech signal captures pressure variations in air that are produced by the speaker
  - The same function as the ear
- The analog speech input signal from the microphone is *sampled* periodically at some fixed *sampling rate*



The Speech Signal: Sampling

- What remains after sampling is the value of the analog signal at *discrete time points*
- This is the *discrete-time signal*



The Speech Signal: Sampling

- The analog speech signal has many *frequencies*
  - The human ear can perceive frequencies in the range 50Hz-15kHz (more if you're young)
- The information about what was spoken is carried in all these frequencies
- But most of it is in the 150Hz-5kHz range

The Speech Signal: Sampling

- A signal that is digitized at *N* samples/sec can represent frequencies up to *N*/2 Hz only
  – The Nyquist theorem
- Ideally, one would sample the speech signal at a sufficiently high rate to retain all perceivable components in the signal
  – > 30kHz
- For practical reasons, lower sampling rates are often used, however
  – Save bandwidth / storage
  – Speed up computation
- A signal that is sampled at *N* samples per second must first be low-pass filtered at *N*/2 Hz to avoid distortions from "aliasing"
  – A topic we wont go into

The Speech Signal: Sampling

- Audio hardware typically supports several standard rates
  - *E.g.*: 8, 16, 11.025, or 44.1 KHz (*n* Hz = *n* samples/sec)
  - CD recording employs 44.1 KHz per channel – high enough to represent most signals most faithfully

- Speech recognition typically uses 8KHz sampling rate for telephone speech and 16KHz for wideband speech
  - Telephone data is *narrowband* and has frequencies only up to 4 KHz
  - Good microphones provide a *wideband* speech signal
    - 16KHz sampling can represent audio frequencies up to 8 KHz
    - This is considered sufficient for speech recognition

The Speech Signal: Digitization

- Each sampled value is *digitized* (or *quantized* or *encoded*) into one of a set of fixed discrete levels
  - Each analog voltage value is *mapped* to the nearest discrete level
  - Since there are a fixed number of discrete levels, the mapped values can be represented by a number; *e.g.* 8-bit, 12-bit or 16-bit

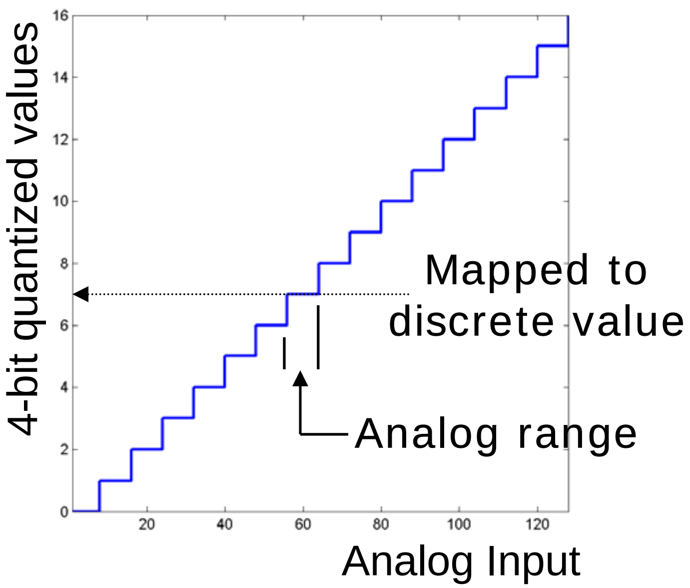
- Digitization can be *linear* (uniform) or *non-linear* (non-uniform)

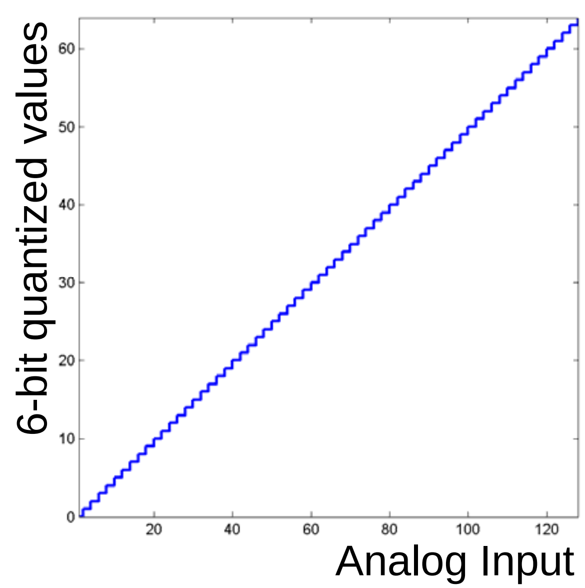
The Speech Signal: Linear Coding

- Linear coding (aka *pulse-code modulation* or PCM) splits the input analog range into some number of uniformly spaced levels
- The no. of discrete levels determines no. of bits needed to represent a quantized signal value; *e.g.*:
  - 4096 levels need a 12-bit representation
  - 65536 levels require 16-bit representation
- In speech recognition, PCM data is typically represented using 16 bits

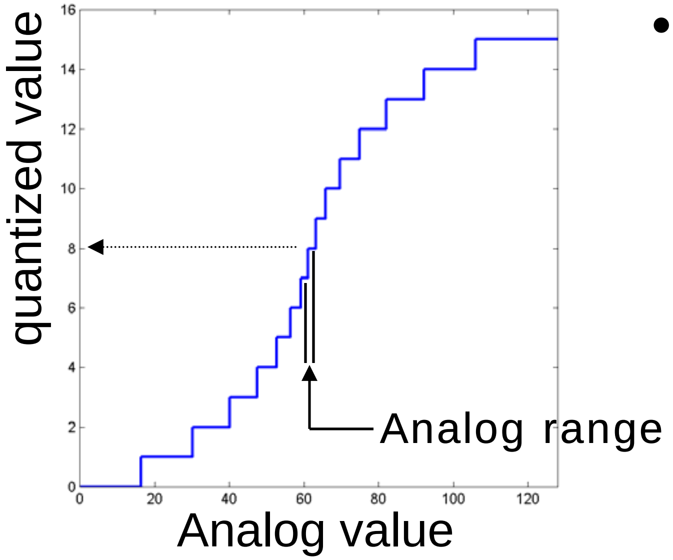
The Speech Signal: Linear Coding

- Example PCM quantizations into 16 and 64 levels:

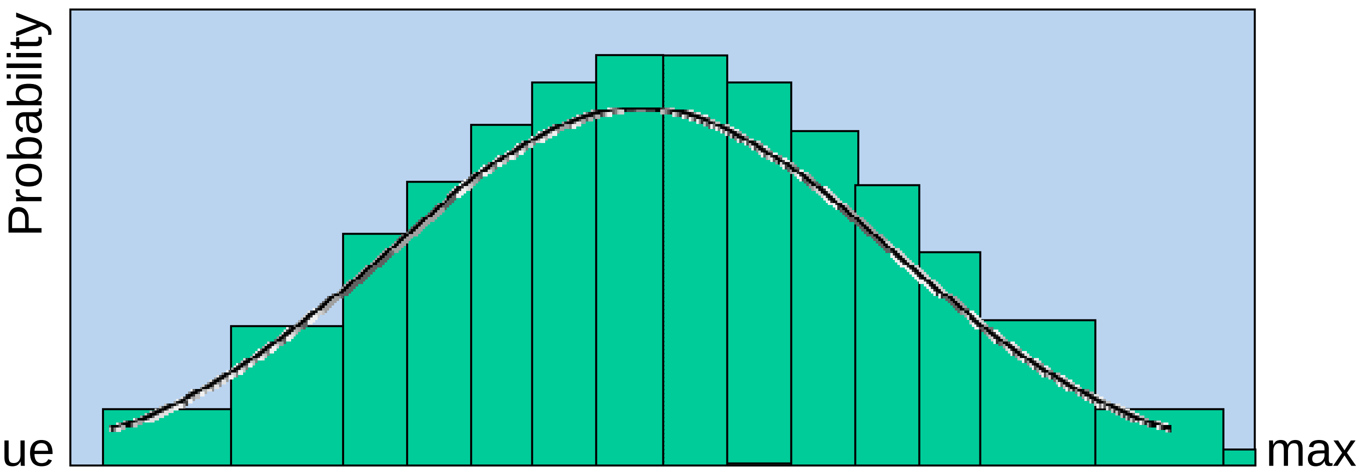




The Speech Signal: Non-Linear Coding



- Converts non-uniform segments of the analog axis to uniform segments of the quantized axis
  - Spacing between adjacent segments on the analog axis is chosen based on the relative frequencies of sample values in that region
  - Sample regions of high frequency are more finely quantized



The Speech Signal: Non-Linear Coding

- Thus, fewer discrete levels can be used, without significantly worsening *average* quantization error
  - High resolution coding around the most probable analog levels
    - Thus, most frequently encountered analog levels have lower quantization error
  - Lower resolution coding around low probability analog levels
    - Encodings with higher quantization error occur less frequently

- *A-law* and *μ-law* encoding schemes use only 256 levels (8-bit encodings)
  - Widely used in telephony
  - Can be converted to linear PCM values via standard tables

- Speech systems usually deal only with 16-bit PCM, so 8-bit signals must first be converted as mentioned above

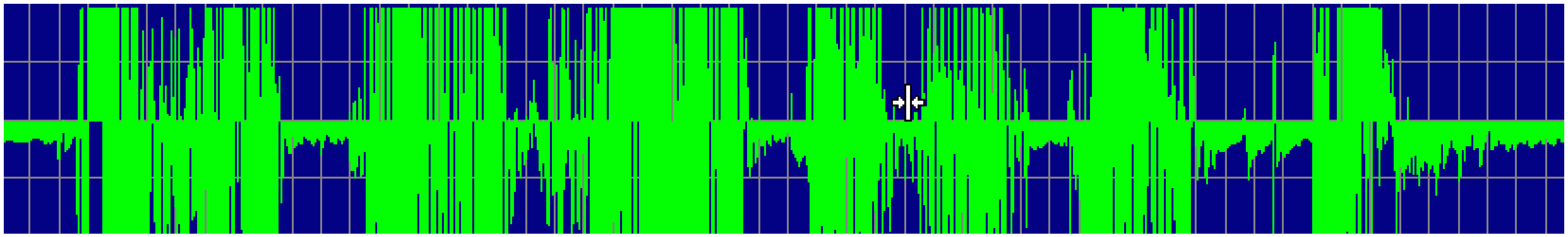
Effect of Signal Quality

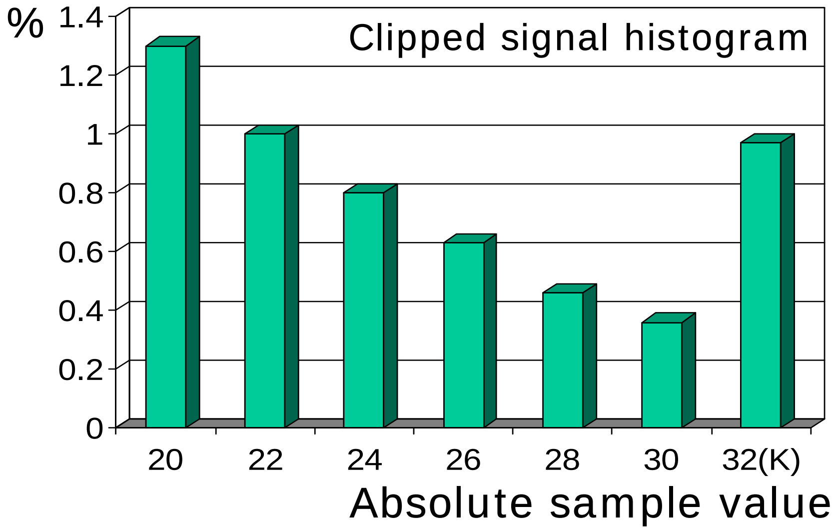
- The quality of the final digitized signal depends critically on all the other components:
  - The microphone quality
  - Environmental quality – the microphone picks up not just the subject's speech, but all other ambient noise
  - The electronics performing sampling and digitization
    - Poor quality electronics can severely degrade signal quality
      - *E.g.* Disk or memory bus activity can inject noise into the analog circuitry
  - Proper setting of the recording level
    - Too low a level underutilizes the available signal range, increasing susceptibility to noise
    - Too high a level can cause *clipping*

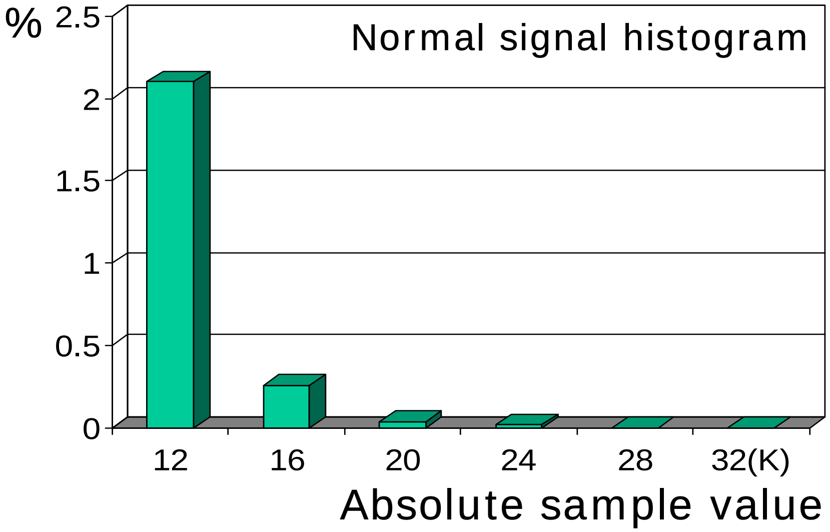
- Suboptimal signal quality can affect recognition accuracy to the point of being completely useless

Digression: Clipping in Speech Signals

- Clipping and non-linear distortion are the most common and most easily fixed problems in audio recording
  – Simply reduce the signal gain (but AGC is not good)







First Step: Feature Extraction

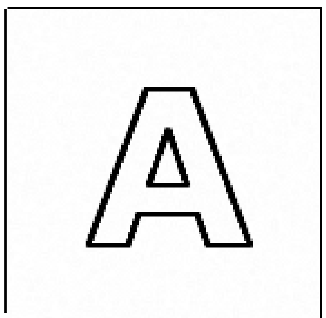
- Speech recognition is a type of pattern recognition problem
- *Q:* Should the pattern matching be performed on the audio sample streams directly? If not, what?
- *A:* Raw sample streams are not well suited for matching
- A visual analogy: recognizing a letter inside a box

  – The input happens to be pixel-wise inverse of the template

- But blind, pixel-wise comparison (*i.e.* on the raw data) shows maximum *dis*-similarity

Feature Extraction (contd.)

- Needed: identification of salient *features* in the images
- E.g. edges, connected lines, shapes
  - These are commonly used features in image analysis
- An *edge detection* algorithm generates the following for both images and now we get a perfect match

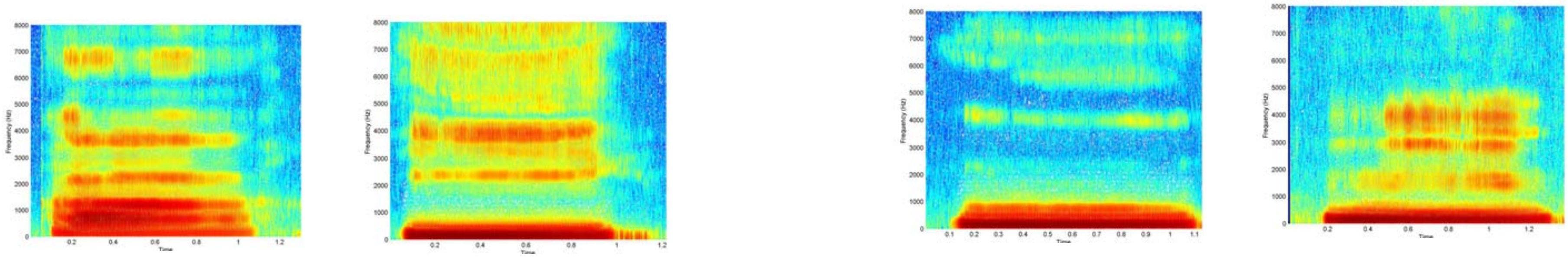


- Our brain does this kind of image analysis automatically and we can instantly identify the input letter as being the same as the template

Sound Characteristics are in Frequency Patterns

- Figures below show energy at various frequencies in a signal as a function of time
  - Called a spectrogram

AA IY UW M



- Different instances of a sound will have the same generic spectral structure
- Features must capture this spectral structure

Computing "Features"

- Features must be computed that capture the *spectral* characteristics of the signal
- Important to capture only the *salient* spectral characteristics of the sounds
  - Without capturing speaker-specific or other incidental structure

- The most commonly used feature is the *Mel-frequency cepstrum*
  - Compute the spectrogram of the signal
  - Derive a set of numbers that capture only the salient apsects of this spectrogram
  - Salient aspects computed according to the manner in which humans perceive sounds

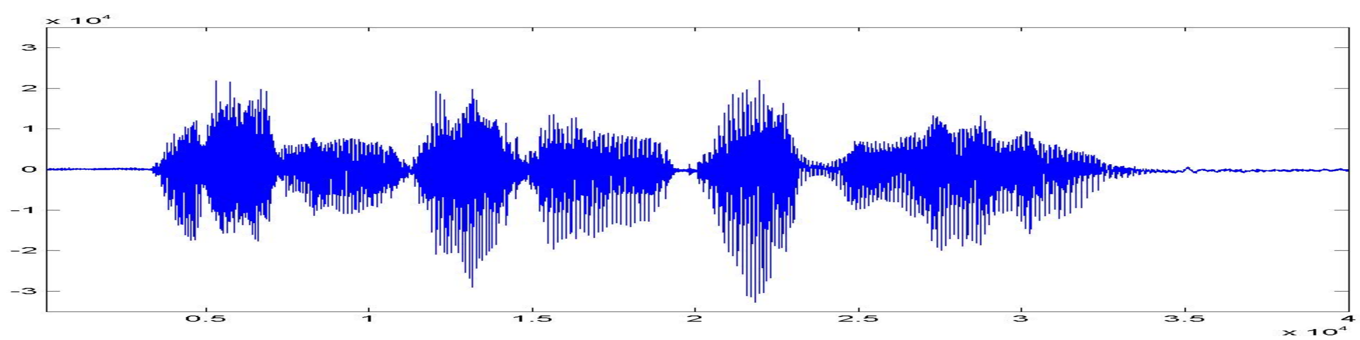
- What follows: A quick intro to signal processing
  - All necessary aspects

Capturing the Spectrum: The discrete Fourier transform

- Transform analysis: Decompose a sequence of numbers into a weighted sum of other time series

- The component time series must be defined
  - For the Fourier Transform, these are complex exponentials

- The analysis determines the weights of the component time series



The complex exponential

- The complex exponential is a complex sum of two sinusoids

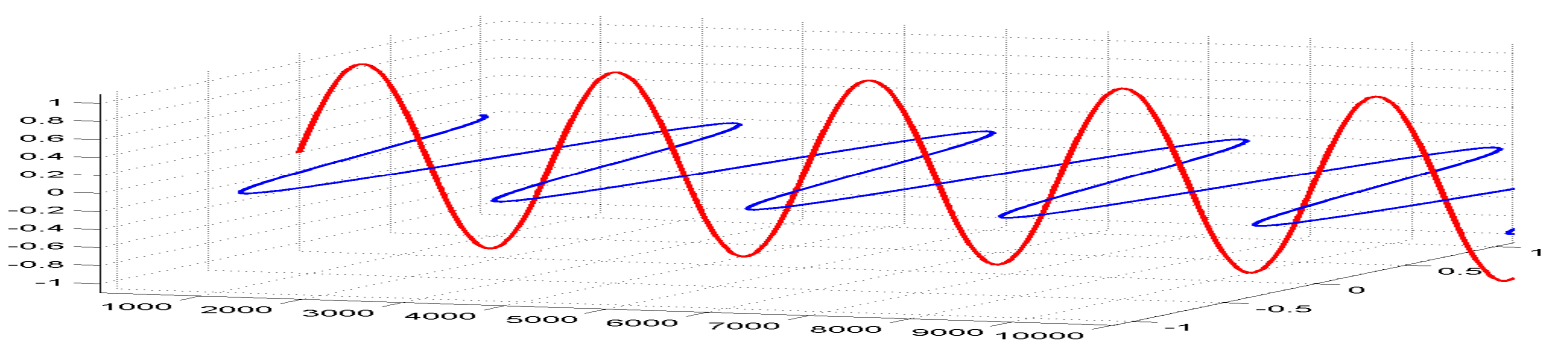
$$e^{j\theta} = \cos\theta + j\sin\theta$$

- The real part is a cosine function
- The imaginary part is a sine function
- A complex exponential time series is a complex sum of two time series

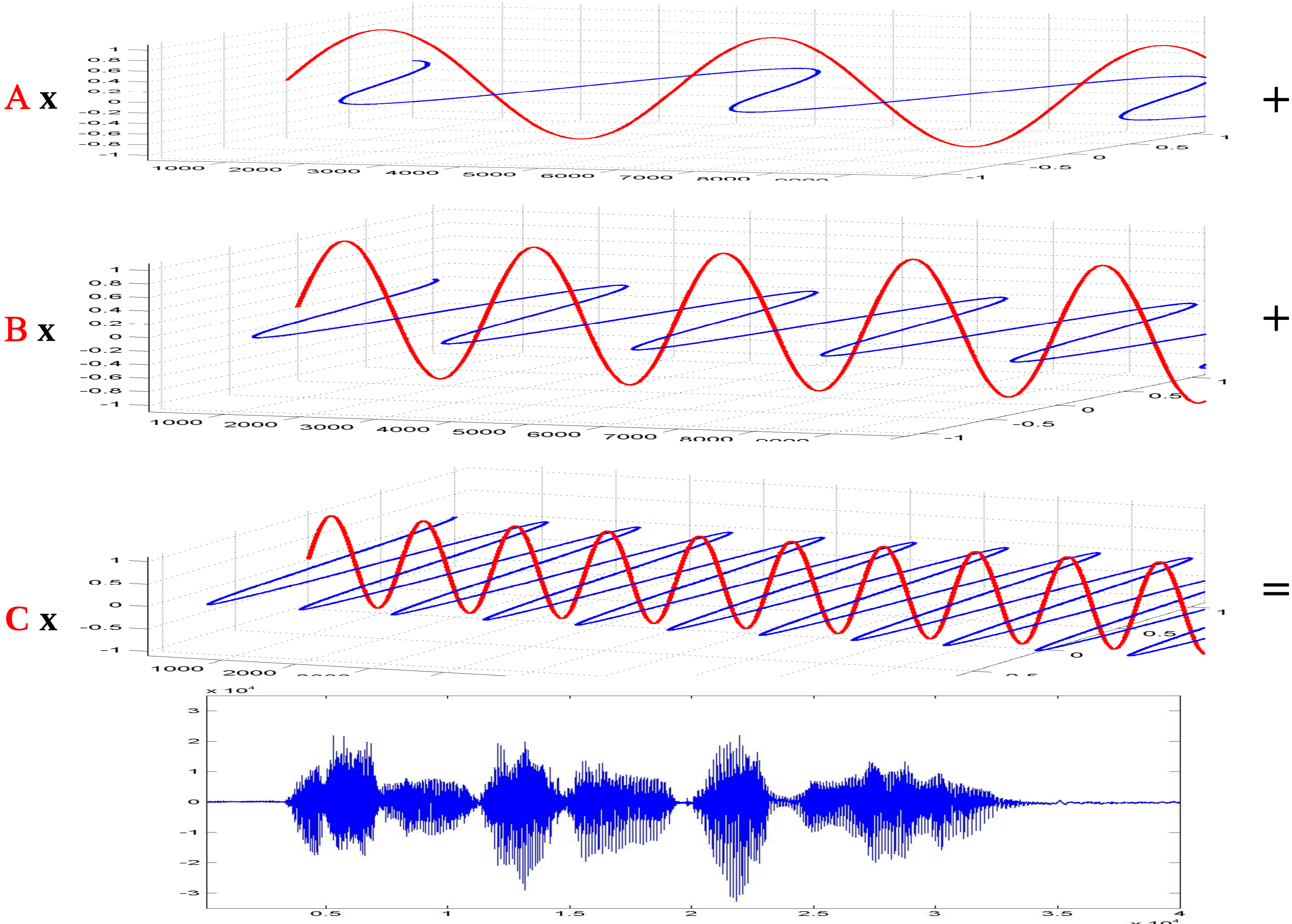
$$e^{j\omega t} = \cos(\omega t) + j\sin(\omega t)$$

- Two complex exponentials of different frequencies are "orthogonal" to each other. i.e.

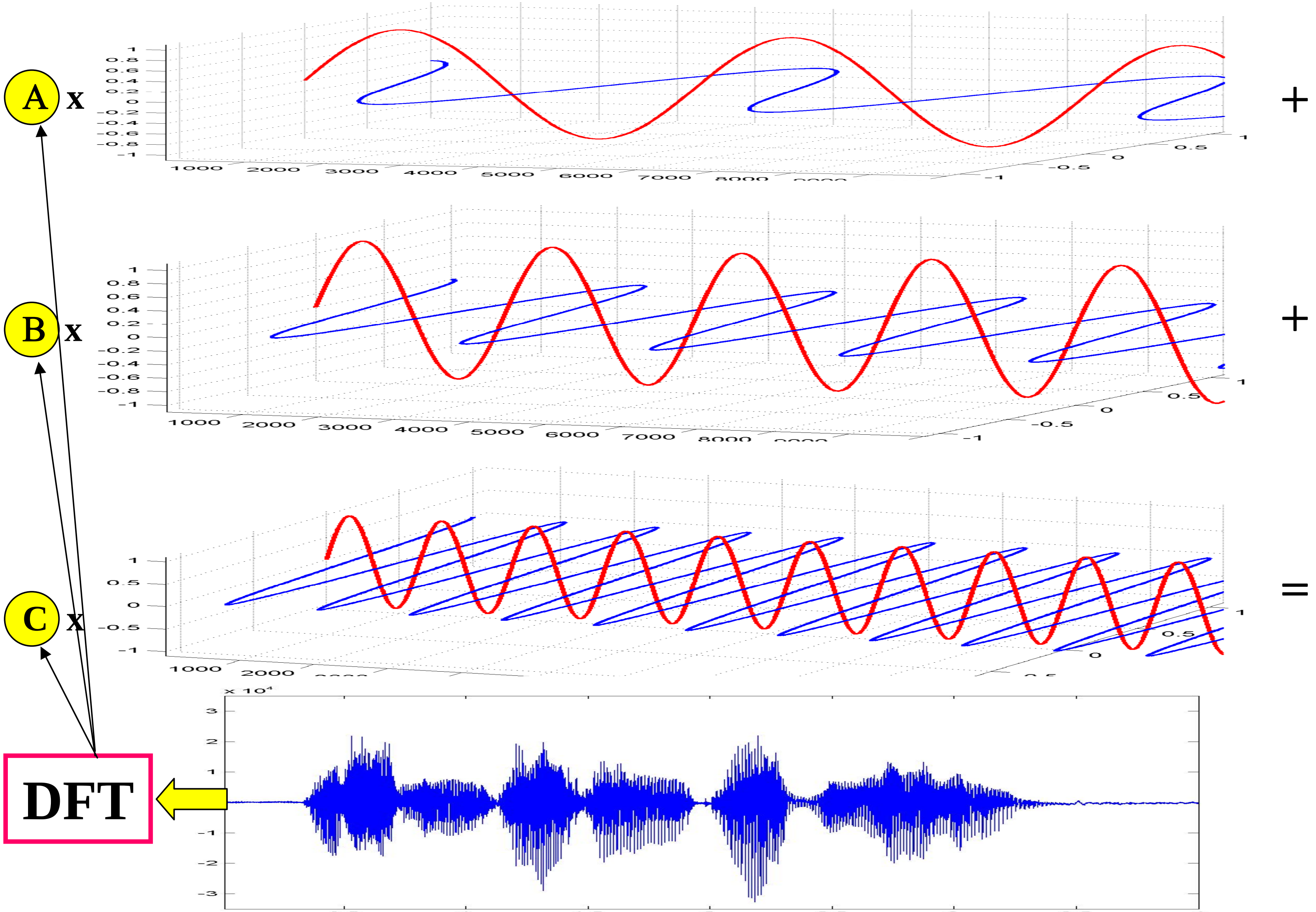
$$\int_{-\infty}^{\infty} e^{j\alpha t} e^{j\beta t} dt = 0 \qquad \text{if } \alpha \neq \beta$$



The discrete Fourier transform



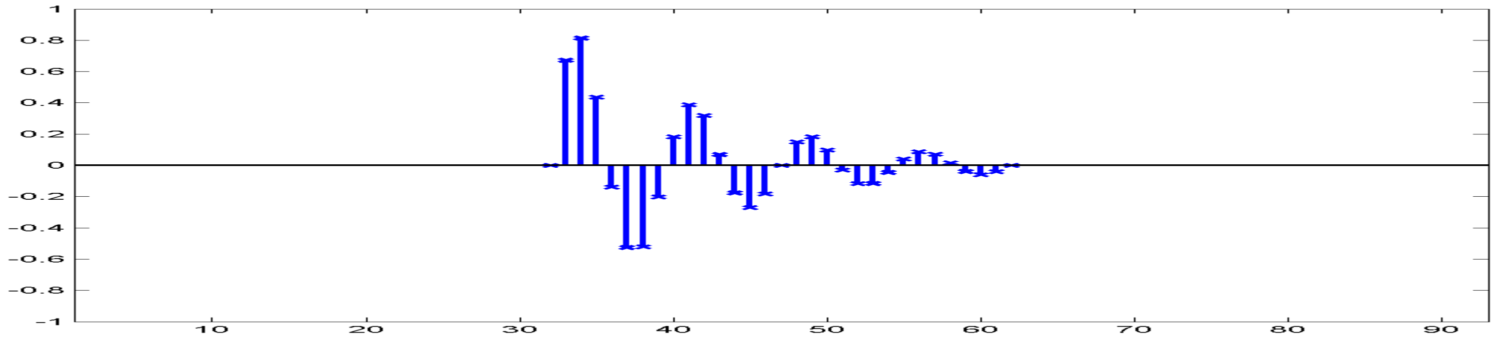
The discrete Fourier transform



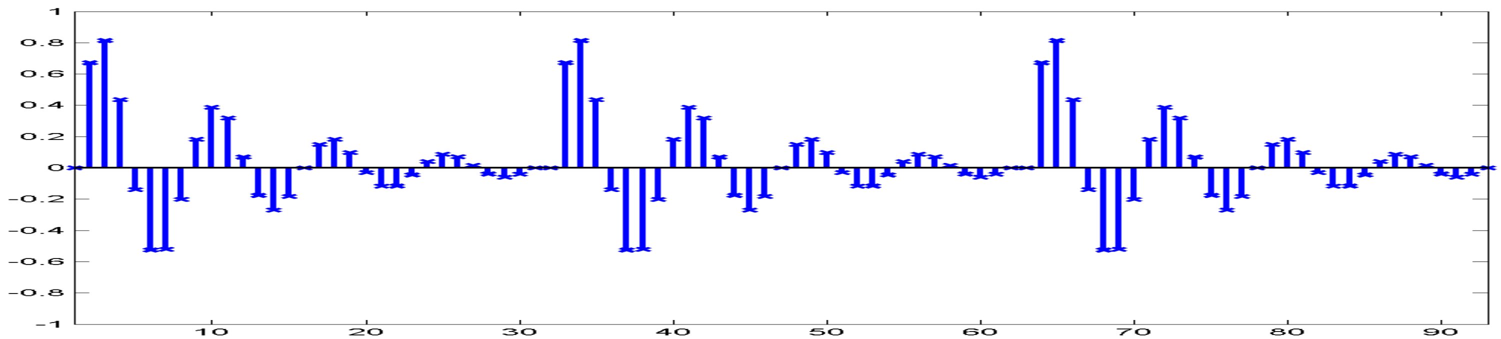
The discrete Fourier transform

- The discrete Fourier transform decomposes the signal into the sum of a finite number of complex exponentials
  - As many exponentials as there are samples in the signal being analyzed
- An aperiodic signal *cannot* be decomposed into a sum of a finite number of complex exponentials
  - Or into a sum of any countable set of periodic signals
- The discrete Fourier transform actually assumes that the signal being analyzed is exactly one period of an infinitely long signal
  - In reality, it computes the Fourier spectrum of the infinitely long periodic signal, of which the analyzed data are one period

The discrete Fourier transform



- The discrete Fourier transform of the above signal actually computes the Fourier spectrum of the periodic signal shown below
  - Which extends from –infinity to +infinity
  - The period of this signal is 31 samples in this example



The discrete Fourier transform

- The k[th] point of a Fourier transform is computed as:

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

  - $x[n]$ is the n[th] point in the analyzed data sequence
  - $X[k]$ is the value of the k[th] point in its Fourier spectrum
  - M is the total number of points in the sequence

- Note that the (M+k)[th] Fourier coefficient is identical to the k[th] Fourier coefficient

$$X[M+k] = \sum_{n=0}^{M-1} x[n] e^{-\frac{j2\pi(M+k)n}{M}} = \sum_{n=0}^{M-1} x[n] e^{-\frac{j2\pi Mn}{M}} e^{-\frac{j2\pi kn}{M}}$$

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

The discrete Fourier transform

- Discrete Fourier transform coefficients are generally complex
  - $e^{j\theta}$ has a real part $\cos\theta$ and an imaginary part $\sin\theta$

$$e^{j\theta} = \cos\theta + j\sin\theta$$

  - As a result, every X[k] has the form

$$X[k] = X_{real}[k] + jX_{imaginary}[k]$$

- A magnitude spectrum represents only the magnitude of the Fourier coefficients

$$X_{magnitude}[k] = \mathrm{sqrt}(X_{real}[k]^2 + X_{imag}[k]^2)$$

- A power spectrum is the square of the magnitude spectrum

$$X_{power}[k] = X_{real}[k]^2 + X_{imag}[k]^2$$

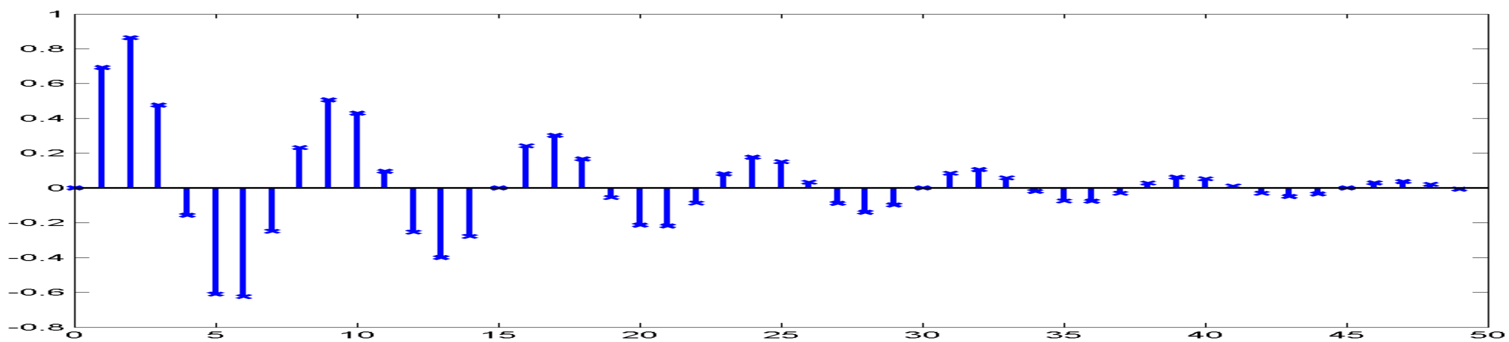
- For speech recognition, we usually use the magnitude or power spectra

The discrete Fourier transform

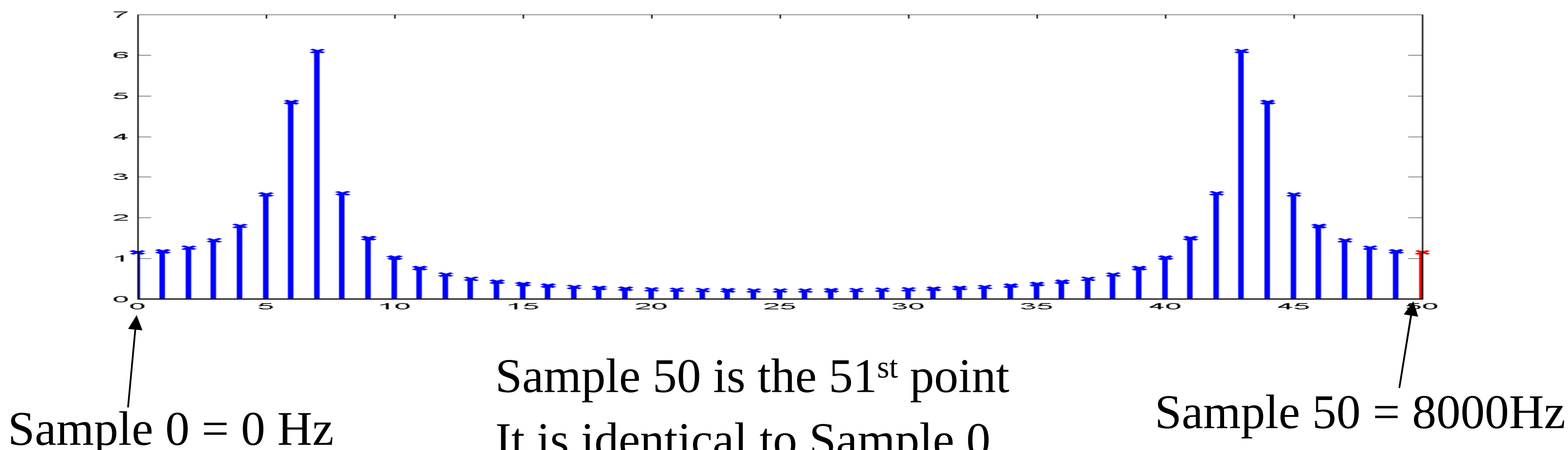
- A discrete Fourier transform of an M-point sequence will only compute M unique frequency components
  - i.e. the DFT of an M point sequence will have M points
  - The M-point DFT represents frequencies in the continuous-time signal that was digitized to obtain the digital signal
- The 0^{th} point in the DFT represents 0Hz, or the DC component of the signal
- The $(M-1)^{th}$ point in the DFT represents (M-1)/M times the sampling frequency
- All DFT points are uniformly spaced on the frequency axis between 0 and the sampling frequency

The discrete Fourier transform

- A 50 point segment of a decaying sine wave sampled at 8000 Hz



- The corresponding 50 point magnitude DFT. The 51st point (shown in red) is identical to the 1st point.



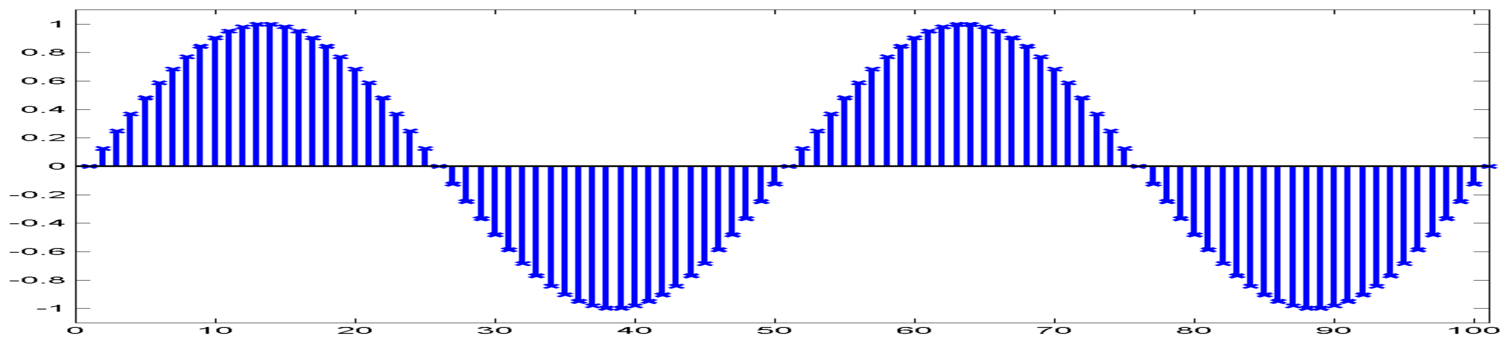
The discrete Fourier transform

- The *Fast Fourier Transform* (FFT) is simply a fast algorithm to compute the DFT
  - It utilizes symmetry in the DFT computation to reduce the total number of arithmetic operations greatly

- The time domain signal can be recovered from its DFT as:

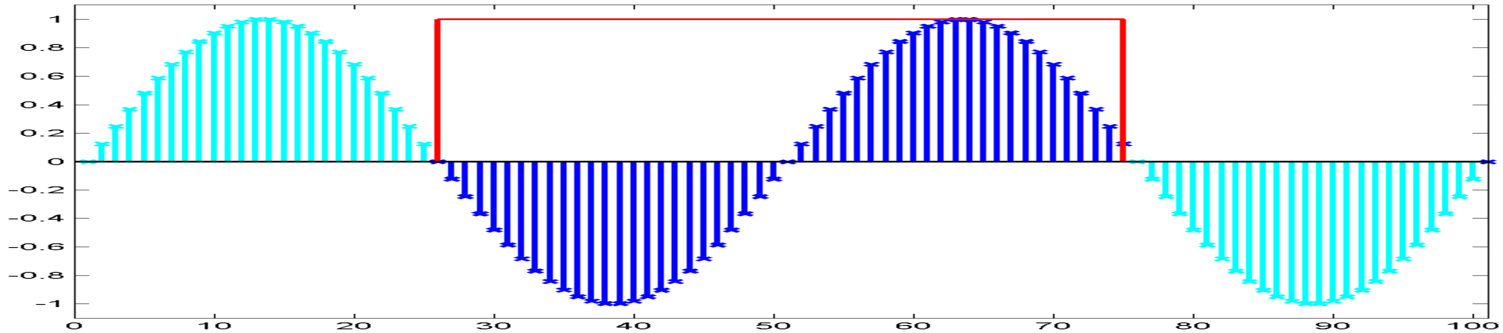
$$x[n] = \frac{1}{M} \sum_{k=0}^{M-1} X[k] e^{\frac{j2\pi kn}{M}}$$

Windowing



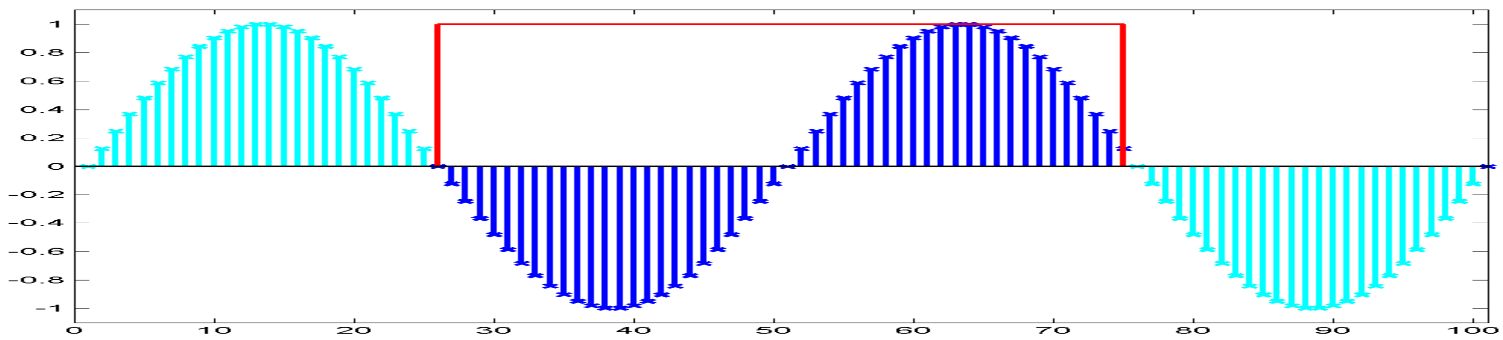
- The DFT of one period of the sinusoid shown in the figure computes the Fourier series of the entire sinusoid from –infinity to +infinity

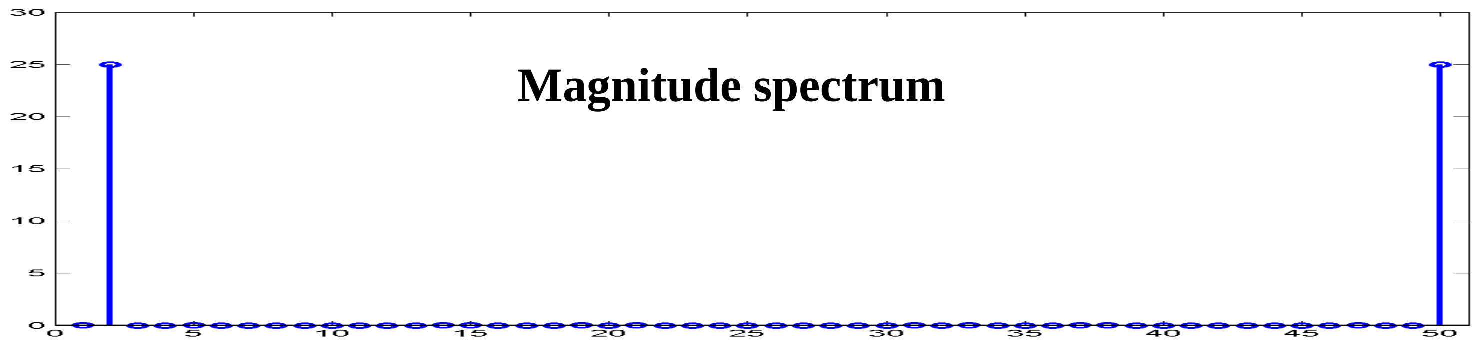
Windowing



- The DFT of one period of the sinusoid shown in the figure computes the Fourier series of the entire sinusoid from –infinity to +infinity

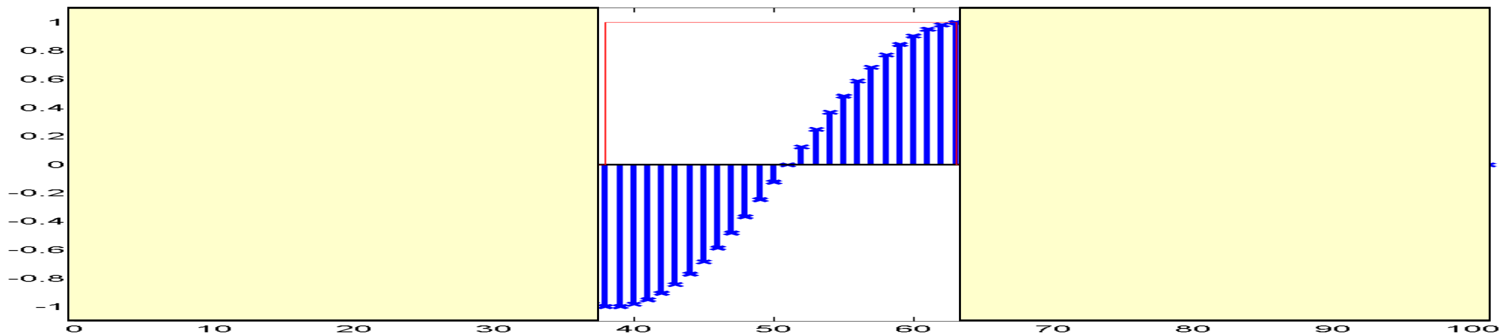
Windowing





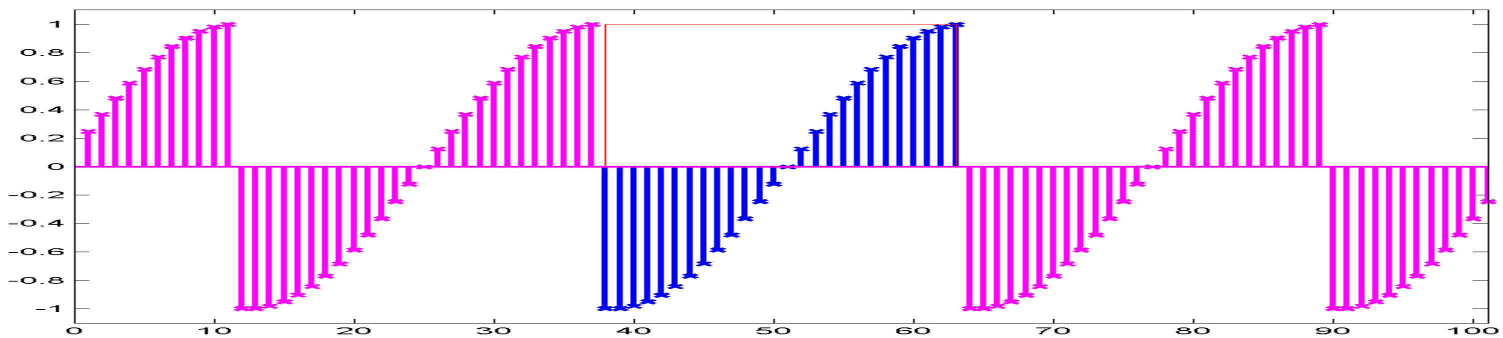
- The DFT of one period of the sinusoid shown in the figure computes the Fourier series of the entire sinusoid from –infinity to +infinity

Windowing



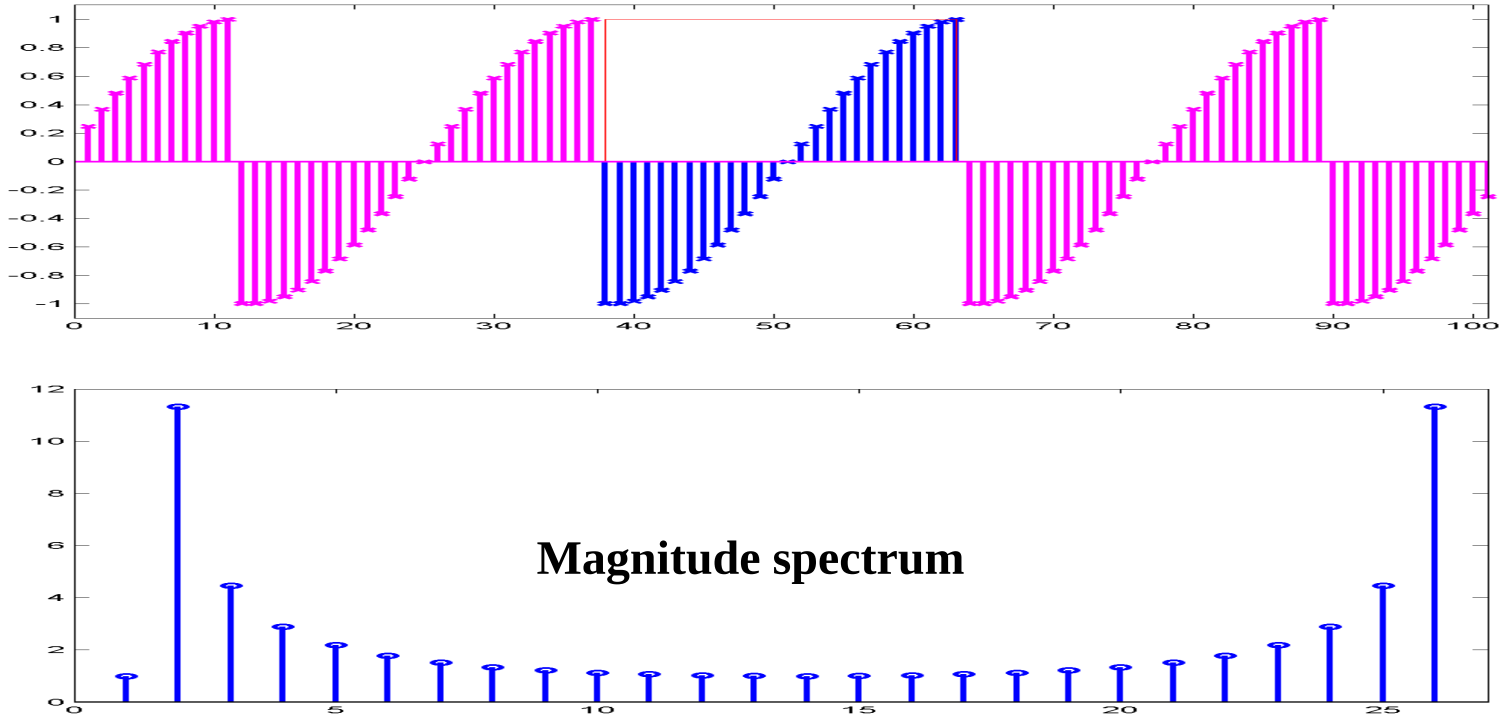
- The DFT of *any* sequence computes the Fourier series for an infinite repetition of that sequence

Windowing



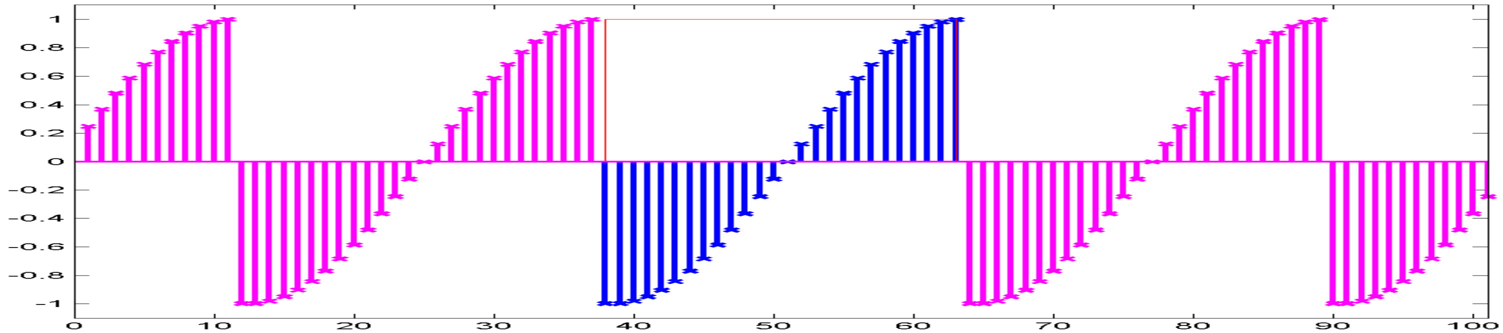
- The DFT of *any* sequence computes the Fourier series for an infinite repetition of that sequence

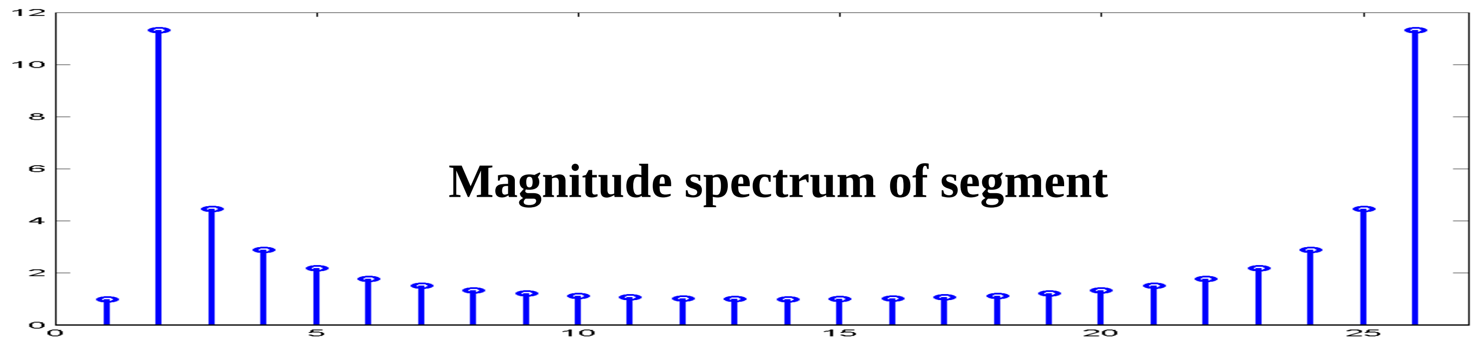
Windowing

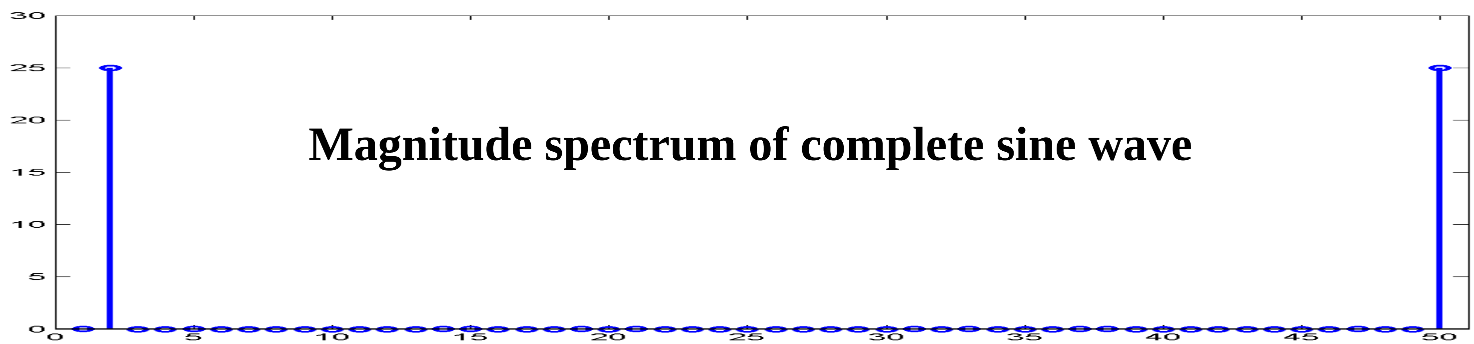


- The DFT of *any* sequence computes the Fourier series for an infinite repetition of that sequence
- The DFT of a partial segment of a sinusoid computes the Fourier series of an inifinite repetition of that segment, and not of the entire sinusoid
- This will not give us the DFT of the sinusoid itself!

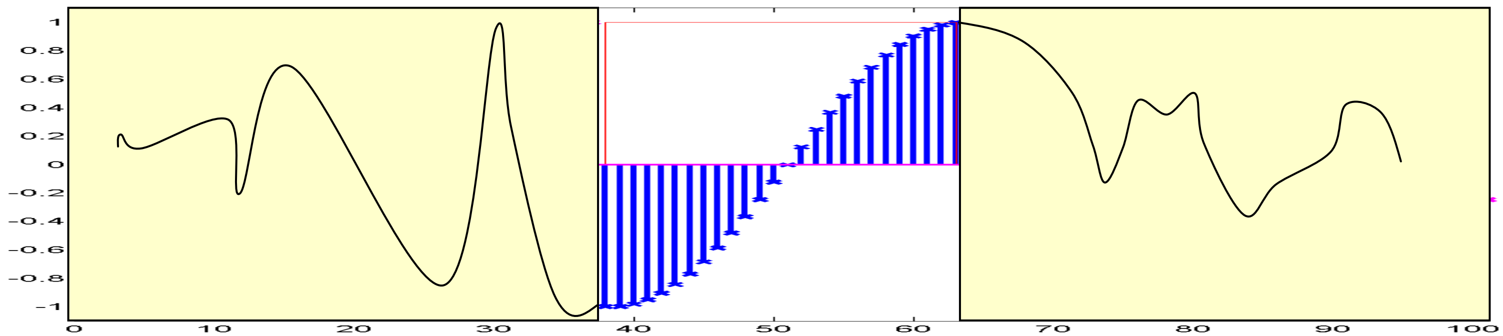
Windowing


1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0 10 20 30 40 50 60 70 80 90 100


12
10
8
6
4
2
0
0 5 10 15 20 25
Magnitude spectrum of segment

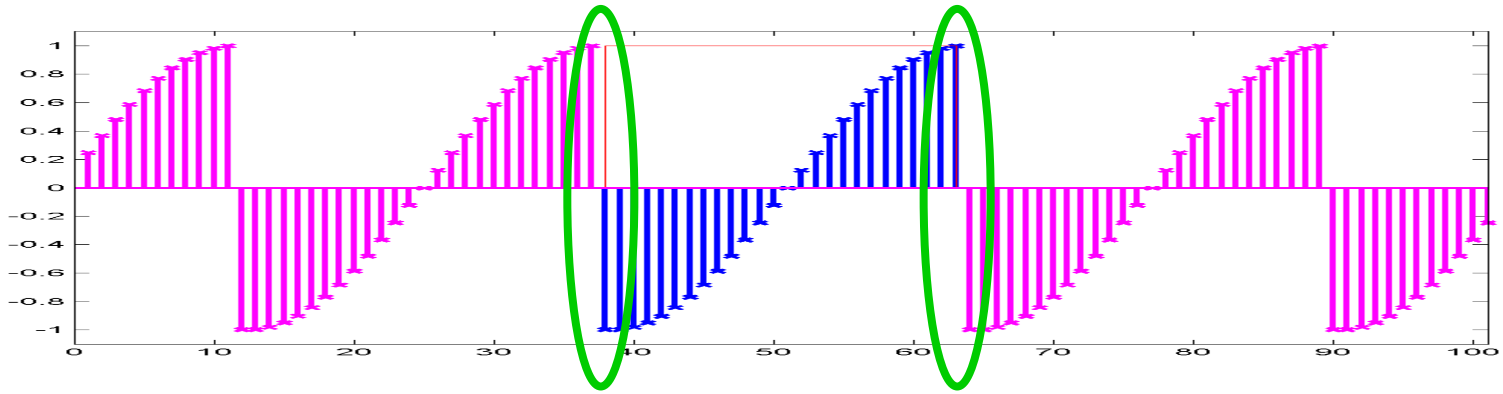

30
25
20
15
10
5
0
0 5 10 15 20 25 30 35 40 45 50
Magnitude spectrum of complete sine wave

Windowing



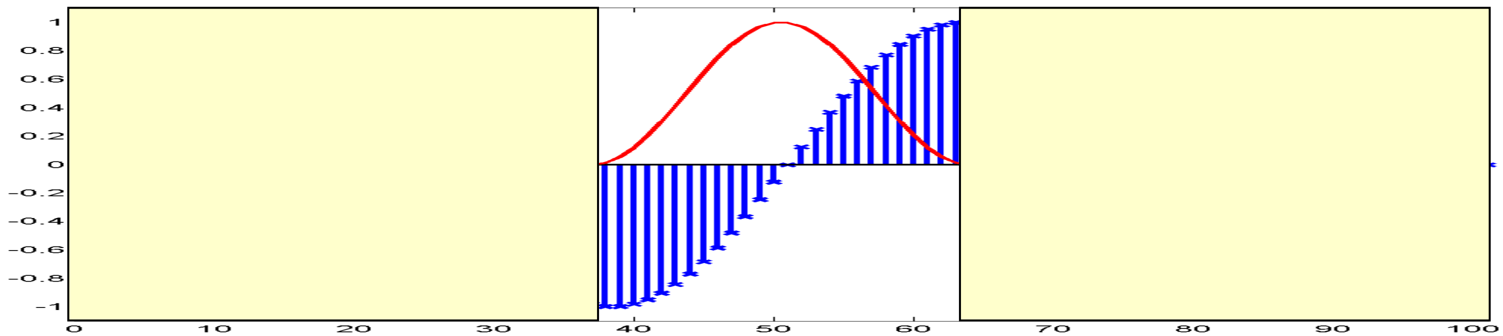
- The difference occurs due to two reasons:
- The transform cannot know what the signal actually looks like outside the observed window
  - We must infer what happens outside the observed window from what happens inside

Windowing



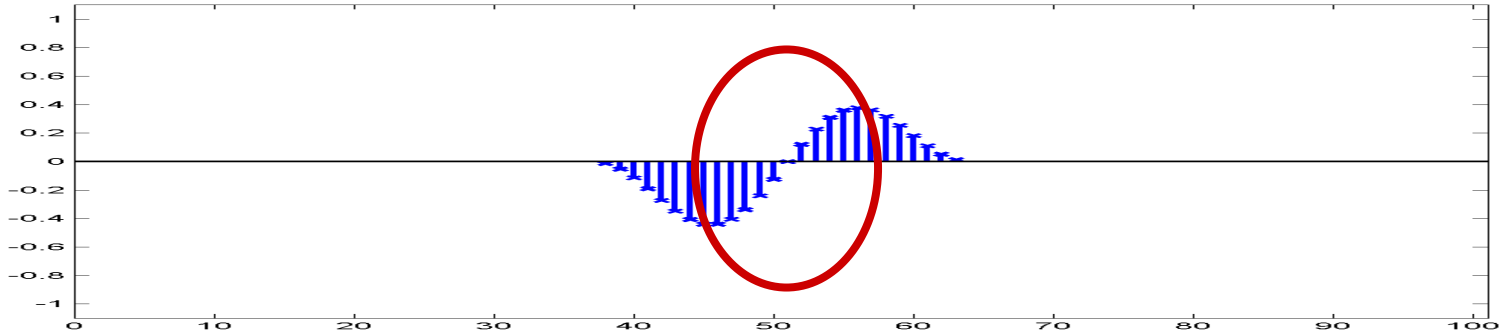
- The difference occurs due to two reasons:
- The transform cannot know what the signal actually looks like outside the observed window
  - We must infer what happens outside the observed window from what happens inside
- The implicit repetition of the observed signal introduces large discontinuities at the points of repetition
  - This distorts even our measurement of what happens at the boundaries of what has been reliably observed
  - The actual signal (whatever it is) is unlikely to have such discontinuities

Windowing



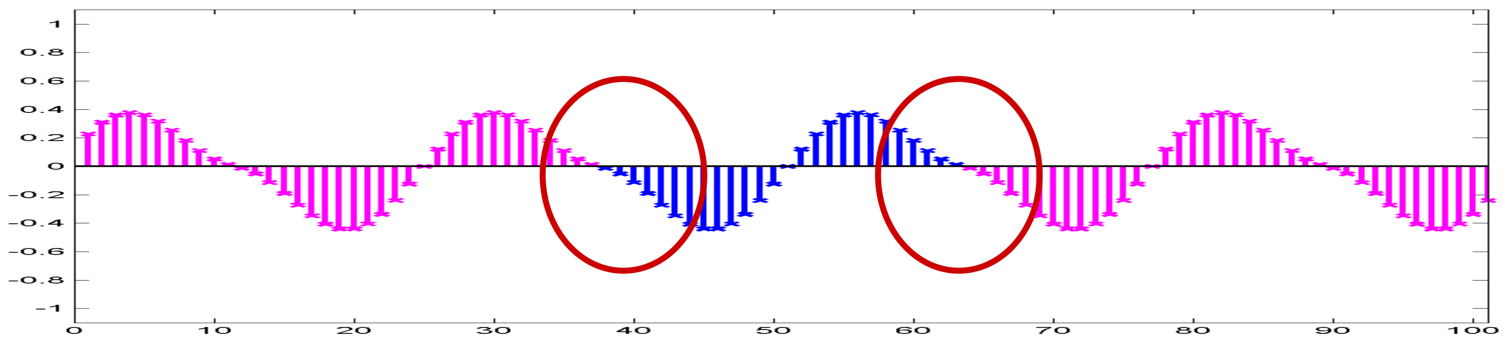
- While we can never know what the signal looks like outside the window, we can try to minimize the discontinuities at the boundaries
- We do this by multiplying the signal with a *window* function
  - We call this procedure windowing
  - We refer to the resulting signal as a "windowed" signal

Windowing



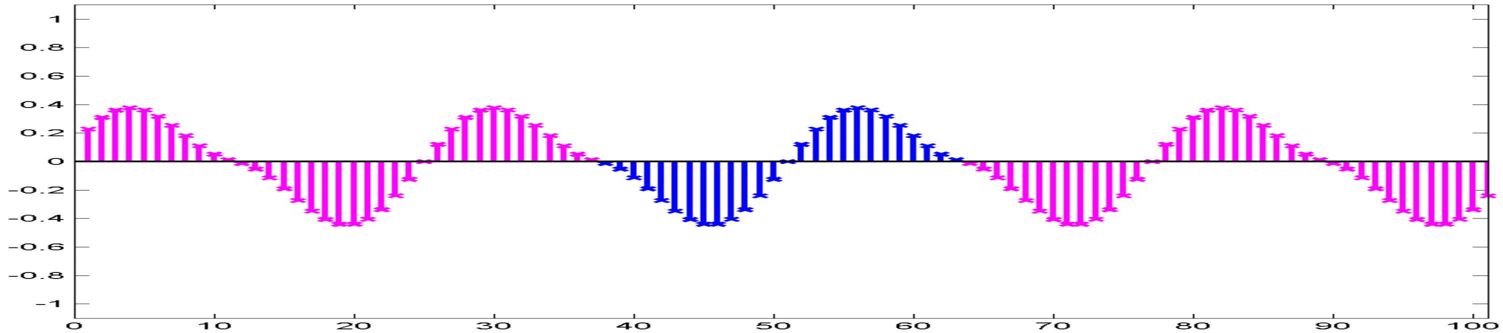
- While we can never know what the signal looks like outside the window, we can try to minimize the discontinuities at the boundaries
- We do this by multiplying the signal with a *window* function
  - We call this procedure windowing
  - We refer to the resulting signal as a "windowed" signal
- Windowing attempts to do the following:
  - Keep the windowed signal similar to the original in the central regions

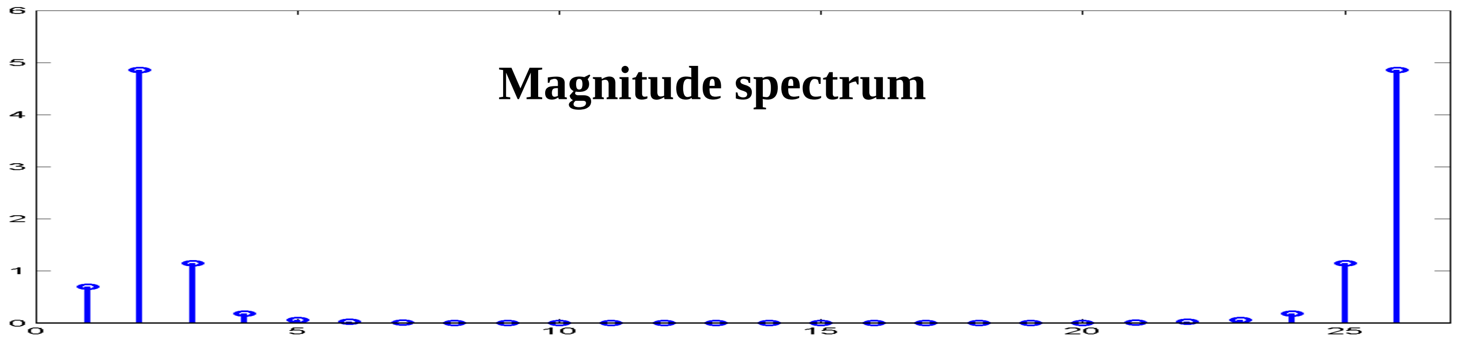
Windowing



- While we can never know what the signal looks like outside the window, we can try to minimize the discontinuities at the boundaries
- We do this by multiplying the signal with a *window* function
  - We call this procedure windowing
  - We refer to the resulting signal as a "windowed" signal
- Windowing attempts to do the following:
  - Keep the windowed signal similar to the original in the central regions
  - Reduce or eliminate the discontinuities in the implicit periodic signal

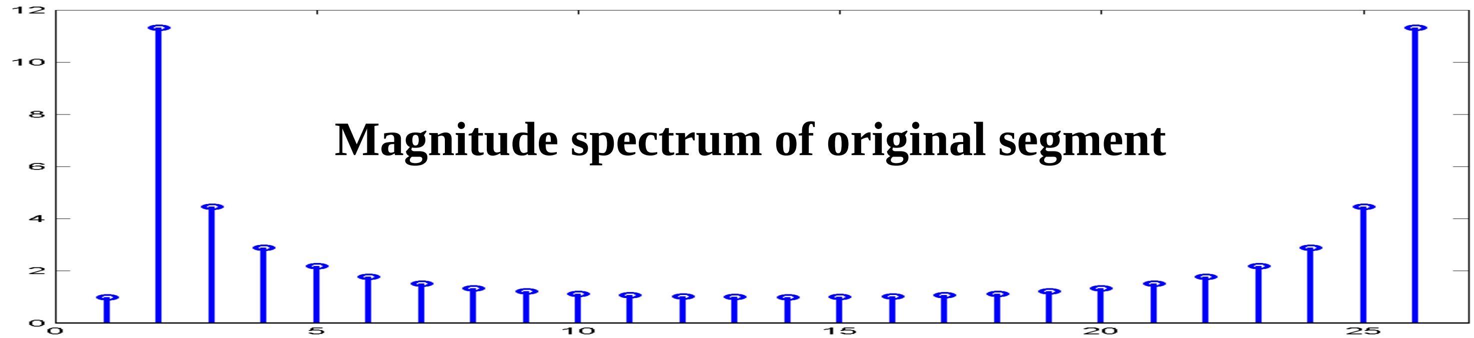
Windowing

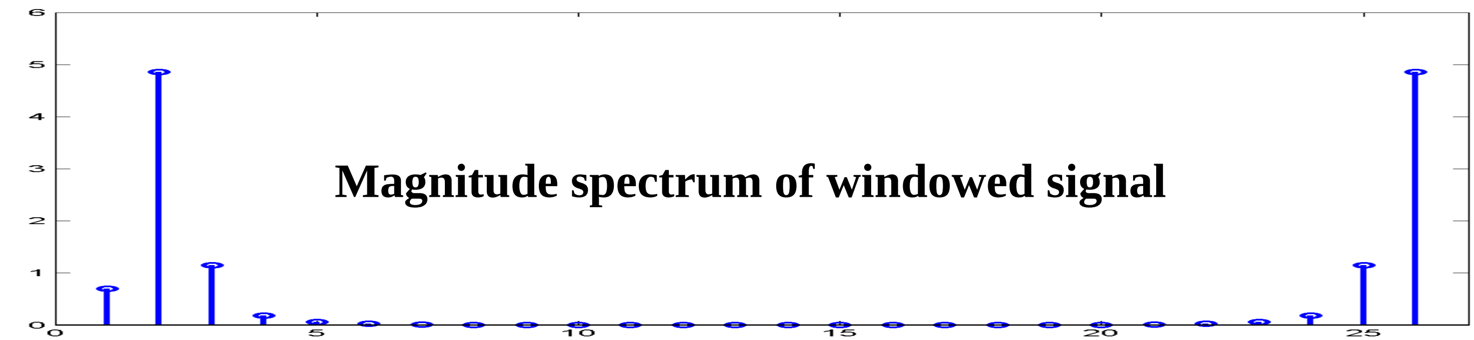


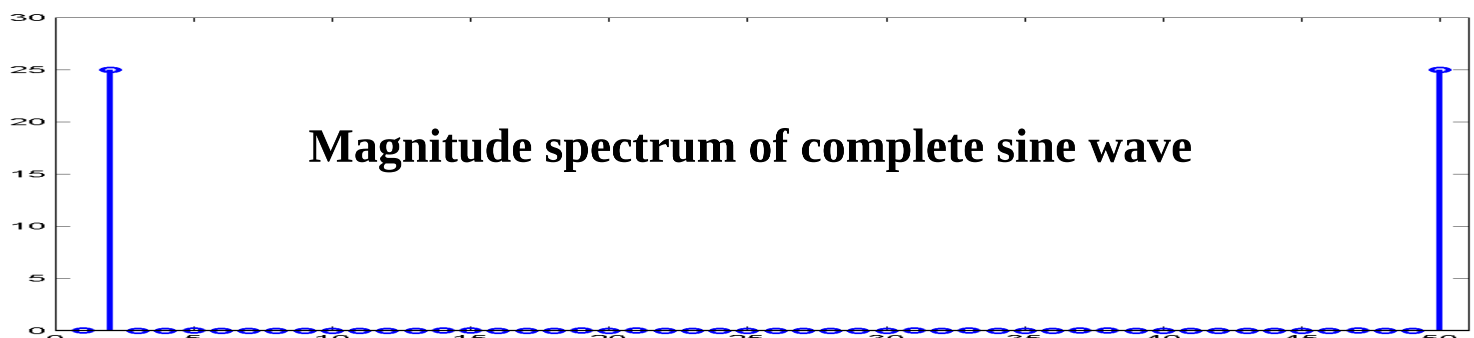


- The DFT of the windowed signal does not have any artefacts introduced by discontinuities in the signal
- Often it is also a more faithful reproduction of the DFT of the complete signal whose segment we have analyzed

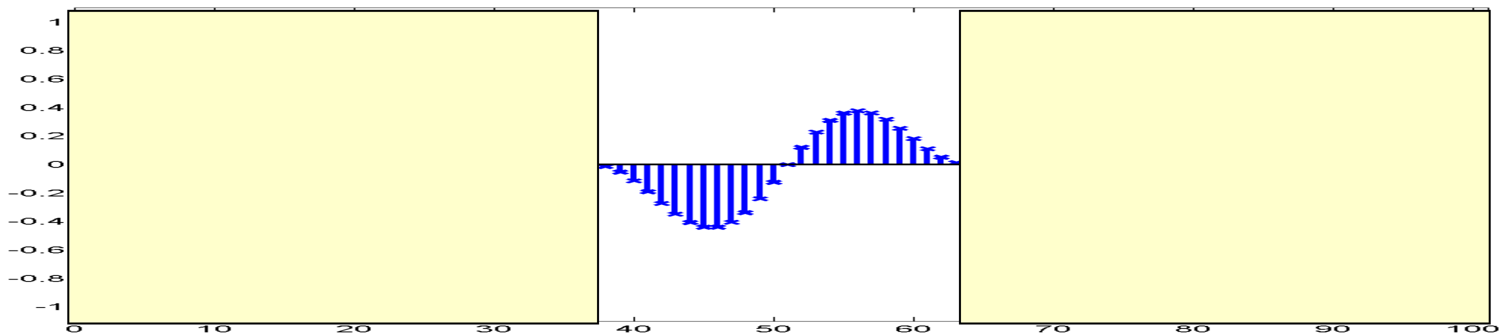
Windowing

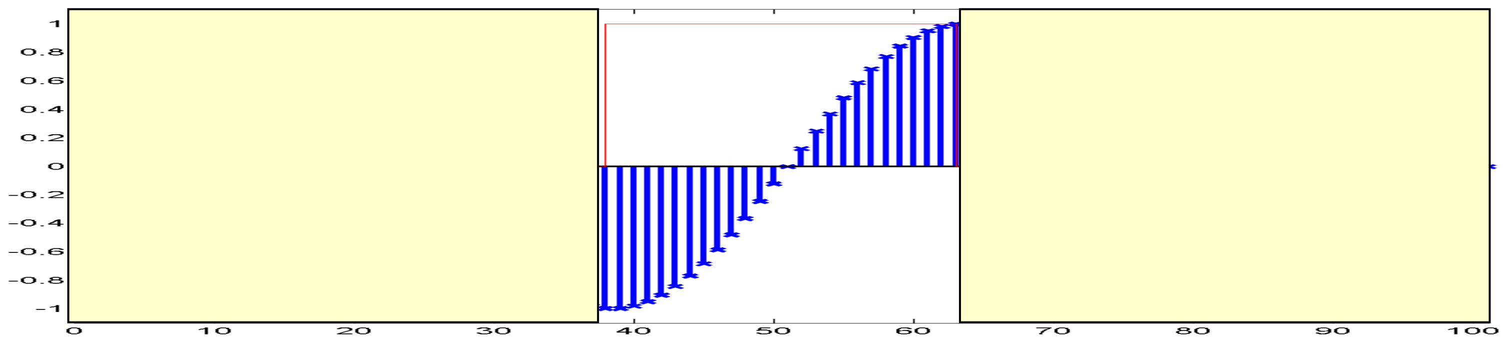

Magnitude spectrum of original segment
0 2 4 6 8 10 12
0 5 10 15 20 25


Magnitude spectrum of windowed signal
0 1 2 3 4 5 6
0 5 10 15 20 25


Magnitude spectrum of complete sine wave
0 5 10 15 20 25 30
0 5 10 15 20 25 30 35 40 45 50

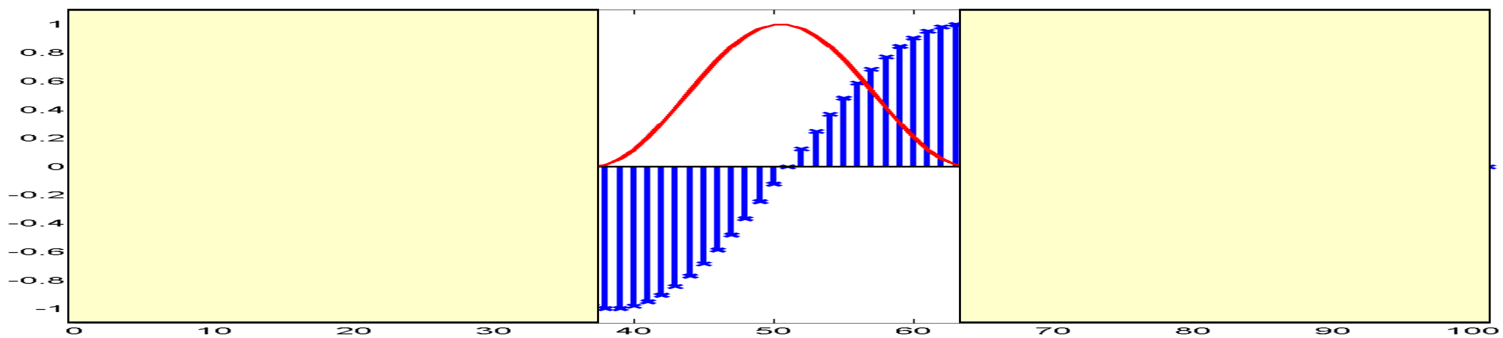
Windowing





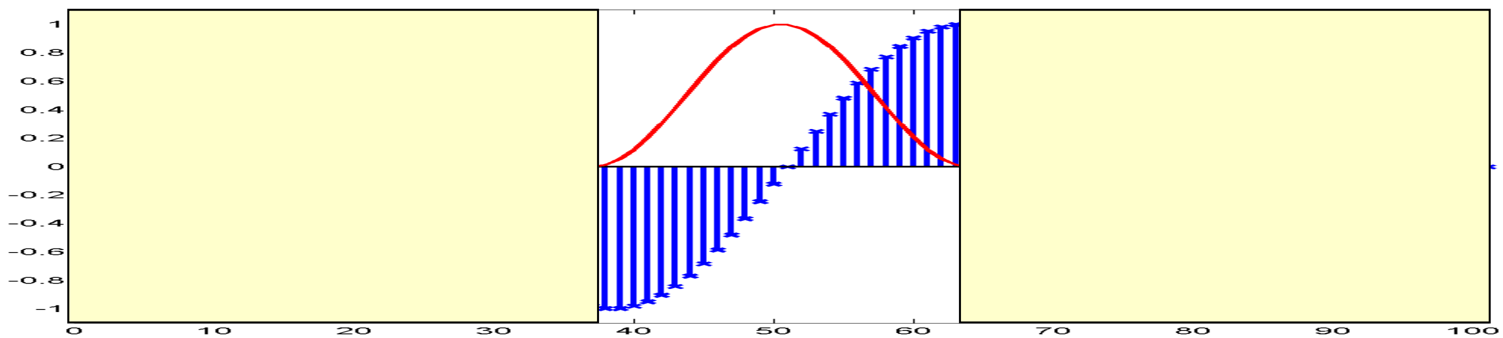
- Windowing is not a perfect solution
  - The original (unwindowed) segment is identical to the original (complete) signal within the segment
  - The windowed segment is often not identical to the complete signal anywhere
- Several windowing functions have been proposed that strike different tradeoffs between the fidelity in the central regions and the smoothing at the boundaries

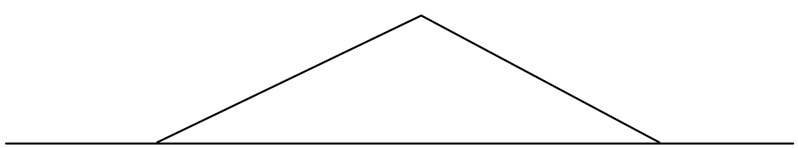
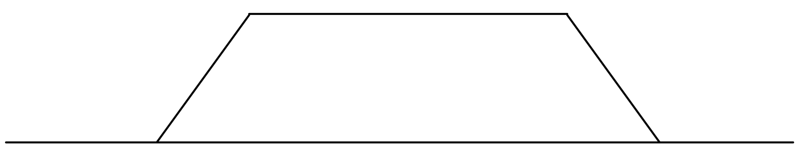
Windowing



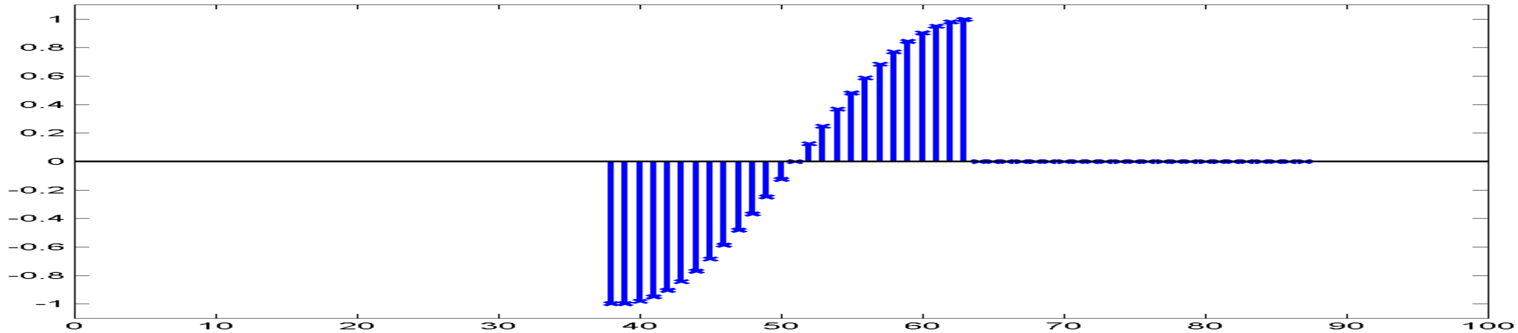
- Cosine windows:
  - Window length is M
  - Index begins at 0
- Hamming: $w[n] = 0.54 – 0.46 \cos(2\pi n/M)$
- Hanning: $w[n] = 0.5 – 0.5 \cos(2\pi n/M)$
- Blackman: $0.42 – 0.5 \cos(2\pi n/M) + 0.08 \cos(4\pi n/M)$

Windowing



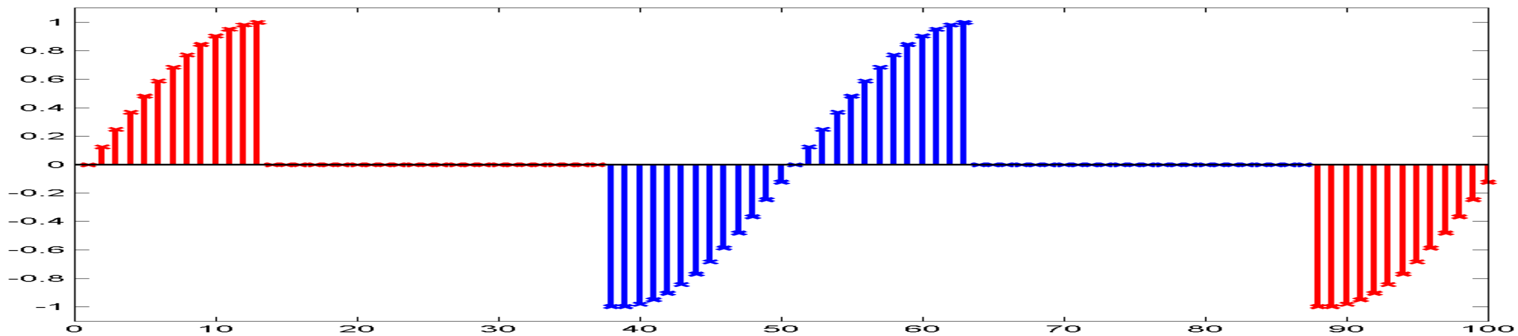
- Geometric windows:
  - Rectangular (boxcar): 
  - Triangular (Bartlett): 
  - Trapezoid: 

Zero Padding



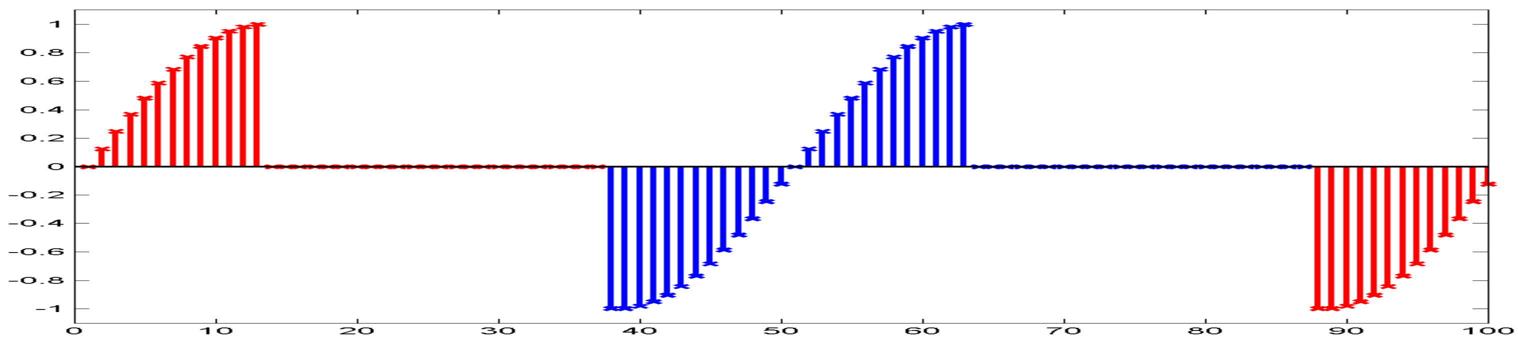
- We can pad zeros to the end of a signal to make it a desired length
  - Useful if the FFT (or any other algorithm we use) requires signals of a specified length
  - E.g. Radix 2 FFTs require signals of length 2^n i.e., some power of 2. We must zero pad the signal to increase its length to the appropriate number

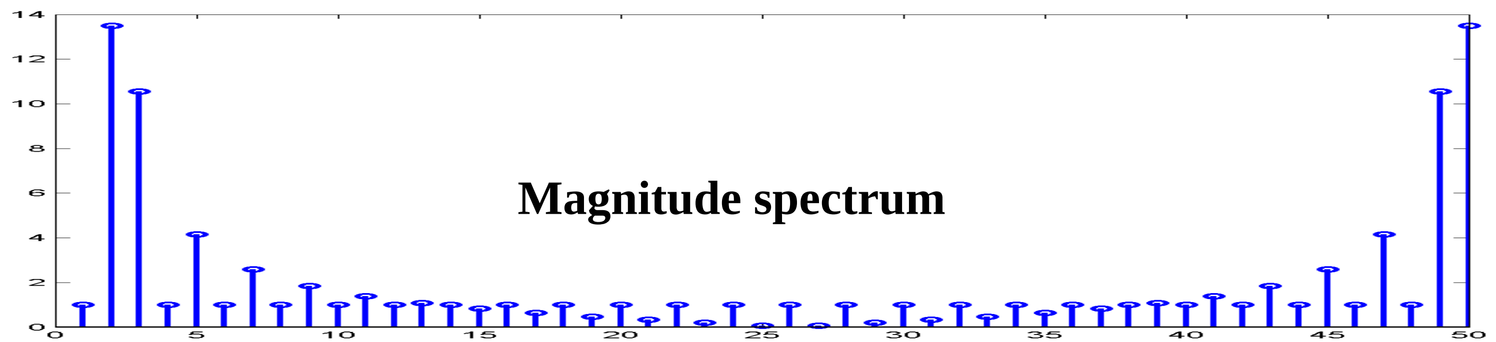
Zero Padding



- We can pad zeros to the end of a signal to make it a desired length
  - Useful if the FFT (or any other algorithm we use) requires signals of a specified length
  - E.g. Radix 2 FFTs require signals of length 2^n i.e., some power of 2. We must zero pad the signal to increase its length to the appropriate number
- The consequence of zero padding is to change the periodic signal whose Fourier spectrum is being computed by the DFT

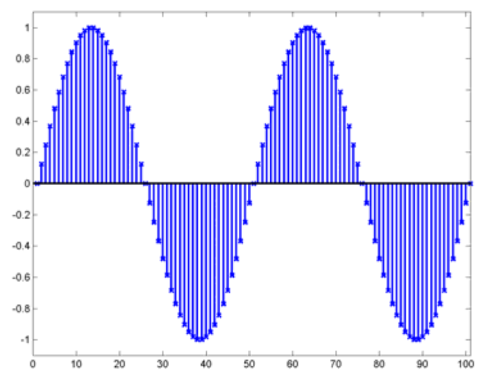
Zero Padding

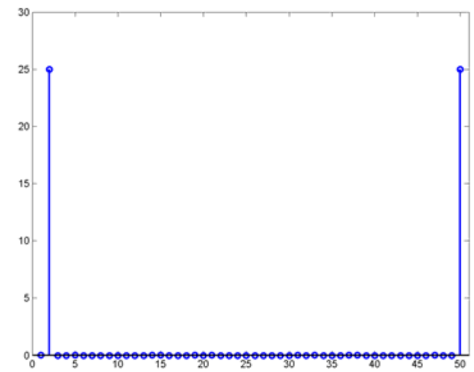


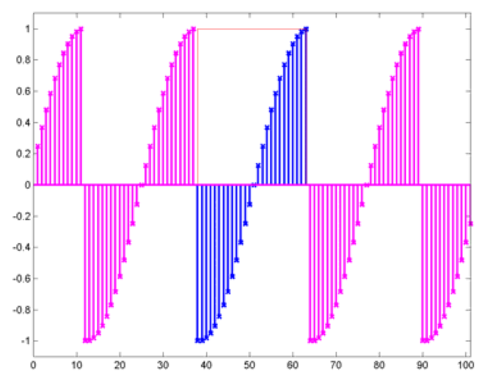


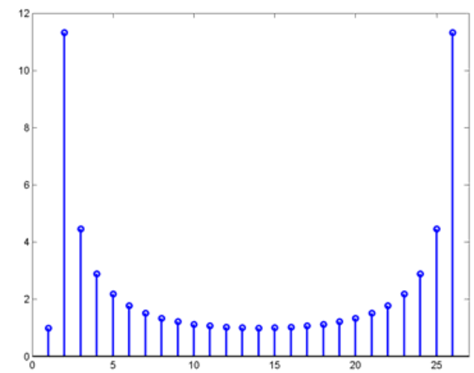
- The DFT of the zero padded signal is essentially the same as the DFT of the unpadded signal, with additional spectral samples inserted in between
  - It does not contain any additional information over the original DFT
  - It also does not contain less information

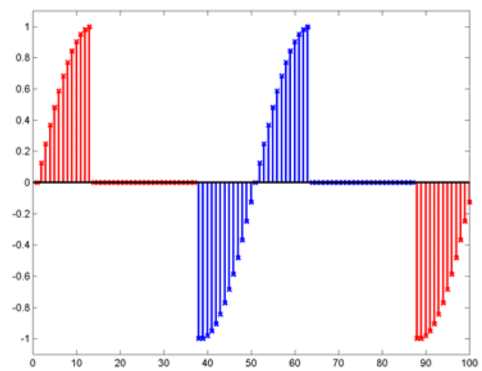
Magnitude spectra

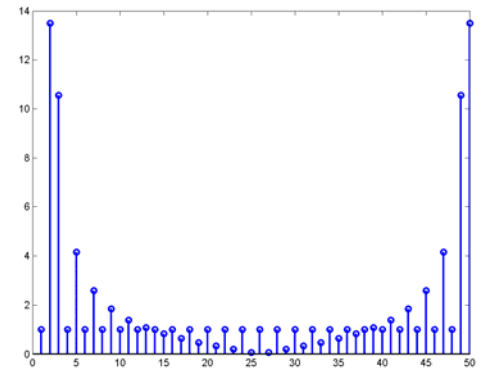




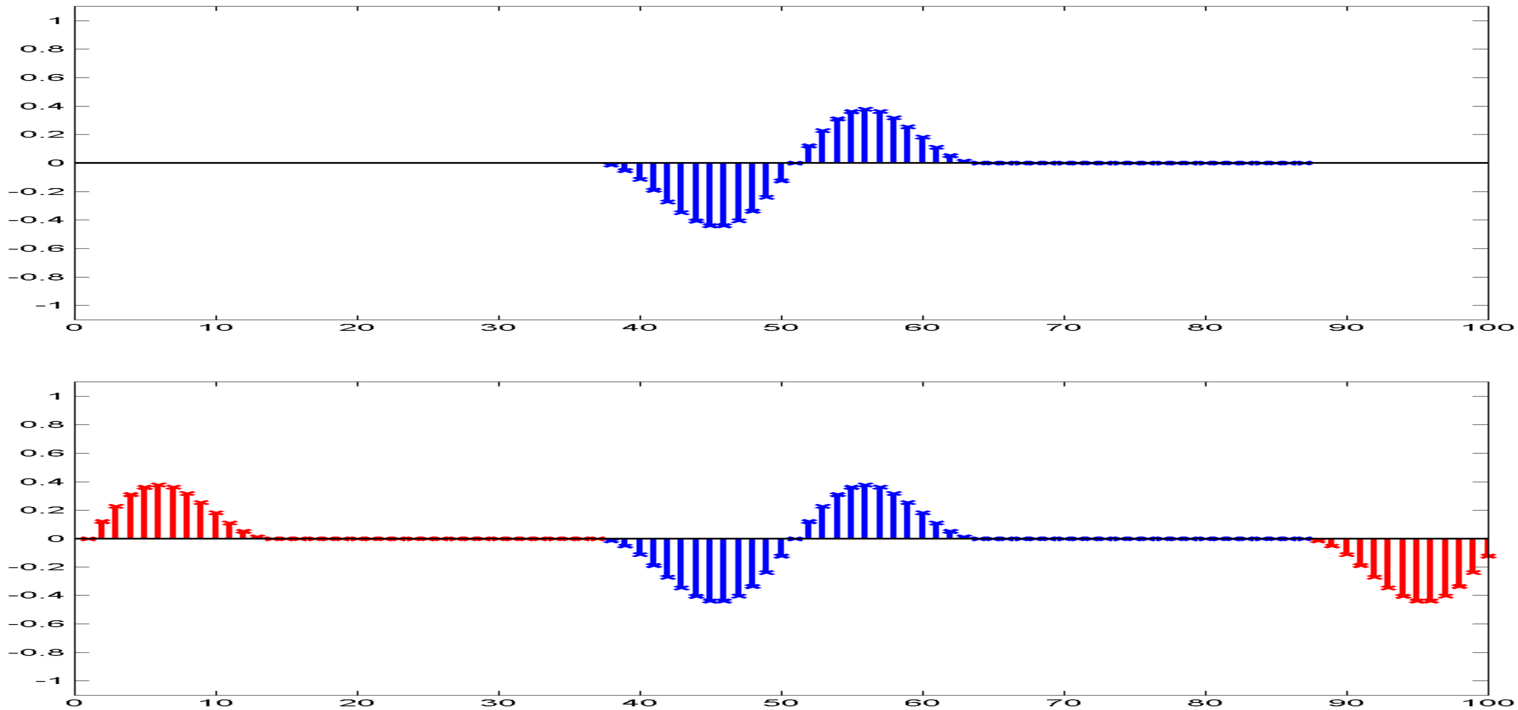






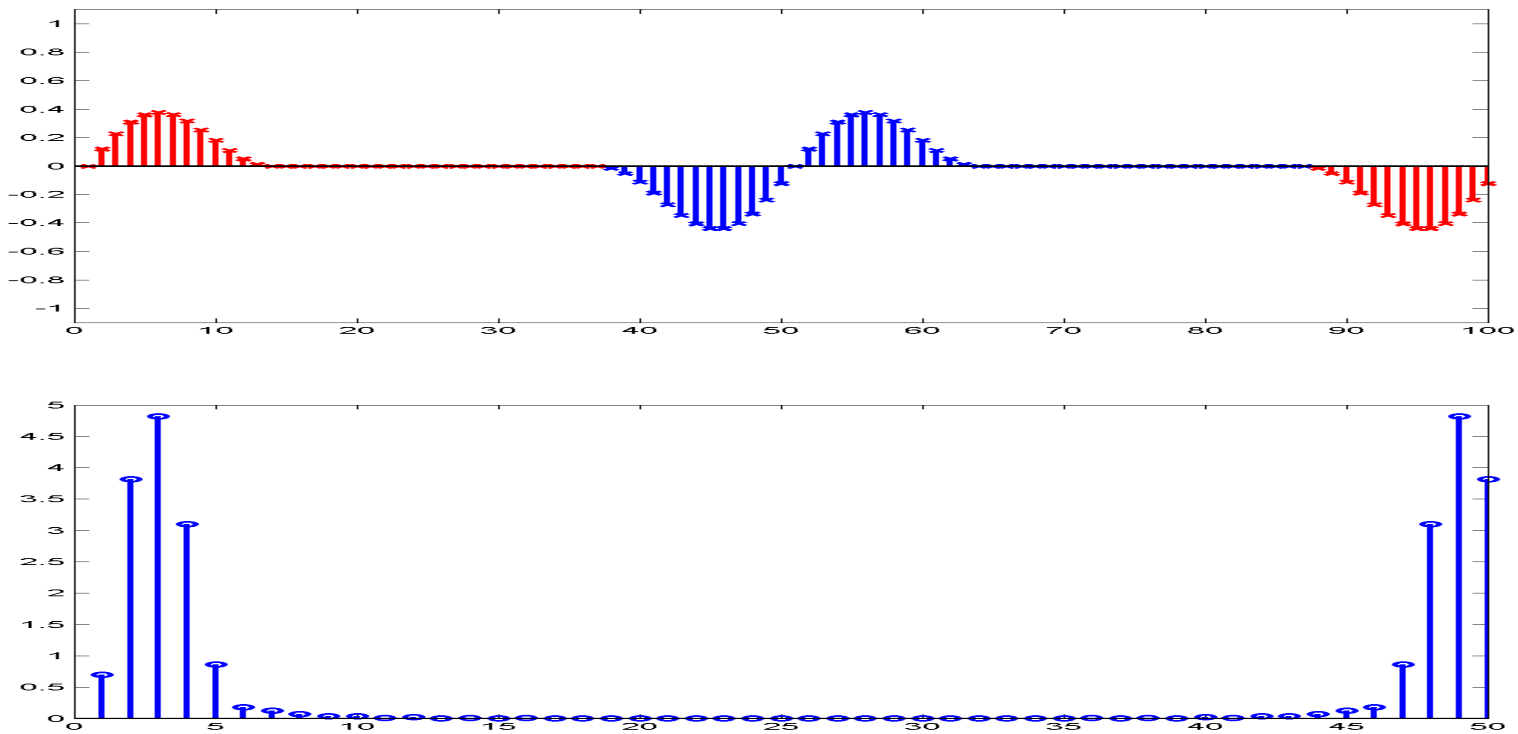


Zero Padding



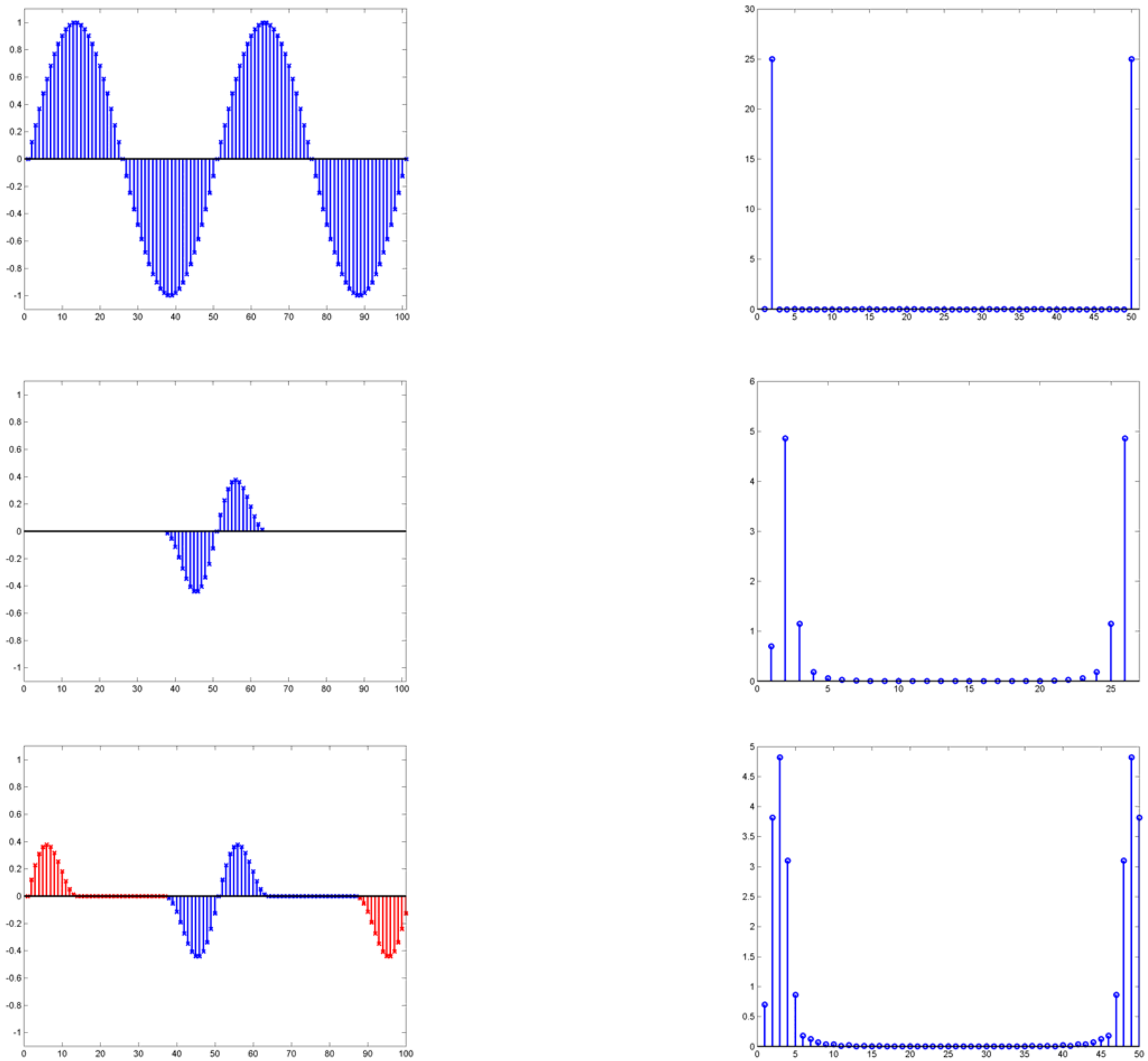
- Zero padding windowed signals results in signals that appear to be less discontinuous at the edges
  - This is only illusory
  - Again, we do not introduce any new information into the signal by merely padding it with zeros

Zero Padding



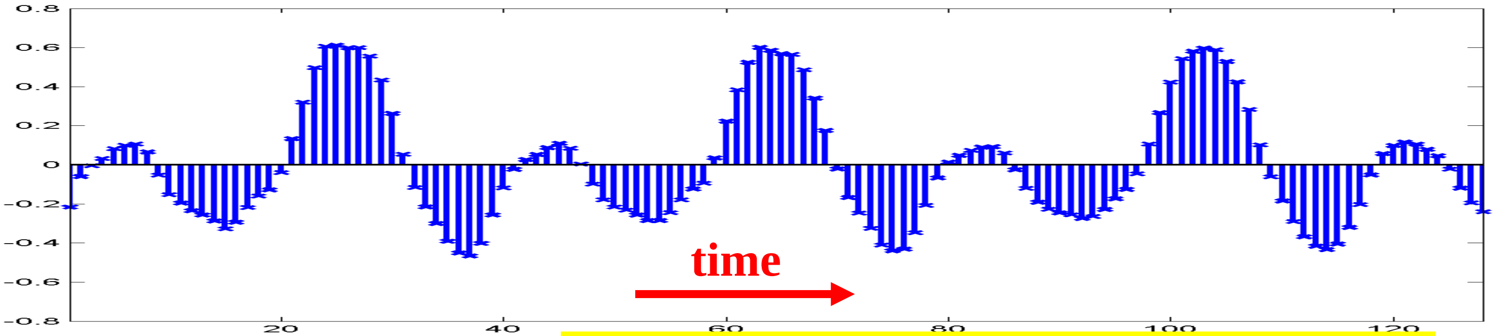
- The DFT of the zero padded signal is essentially the same as the DFT of the unpadded signal, with additional spectral samples inserted in between
  - It does not contain any additional information over the original DFT
  - It also does not contain less information

Magnitude spectra

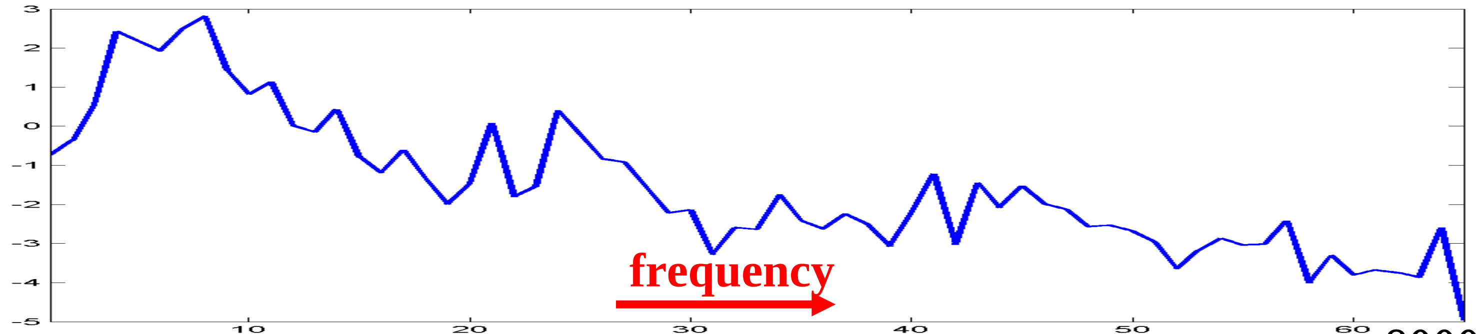


Zero padding a speech signal

128 samples from a speech signal sampled at 16000 Hz

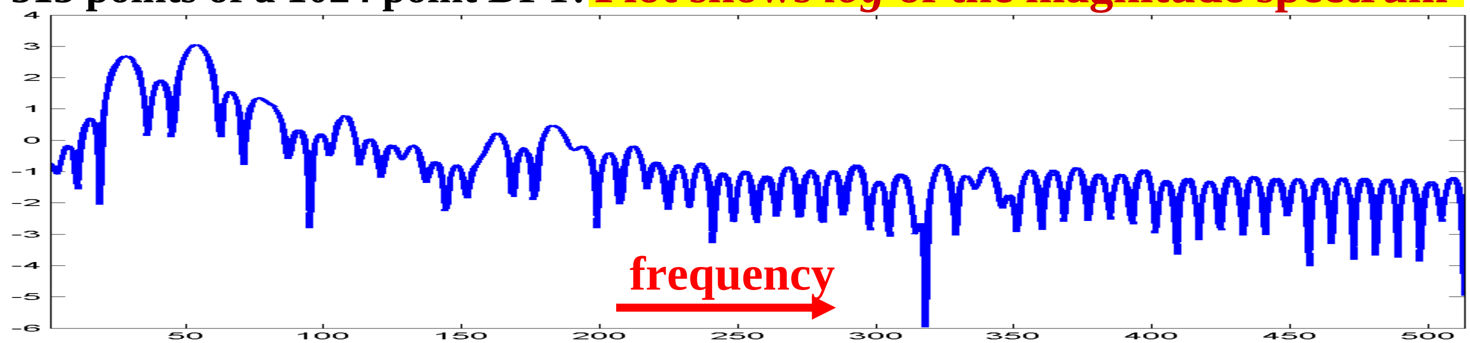


The first 65 points of a 128 point DFT. **Plot shows *log* of the magnitude spectrum**



8000Hz

The first 513 points of a 1024 point DFT. **Plot shows *log* of the magnitude spectrum**



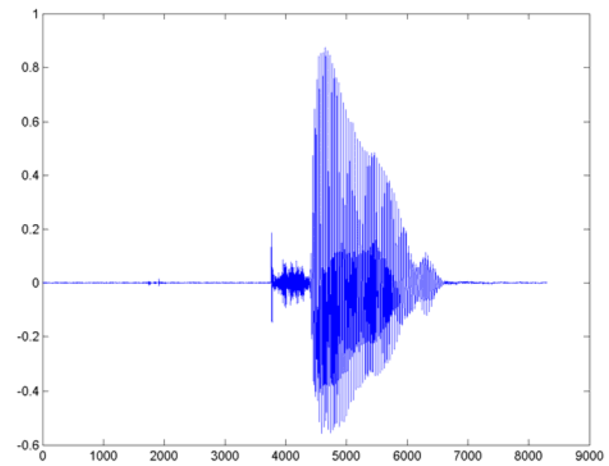
8000Hz

Preemphasizing a speech signal

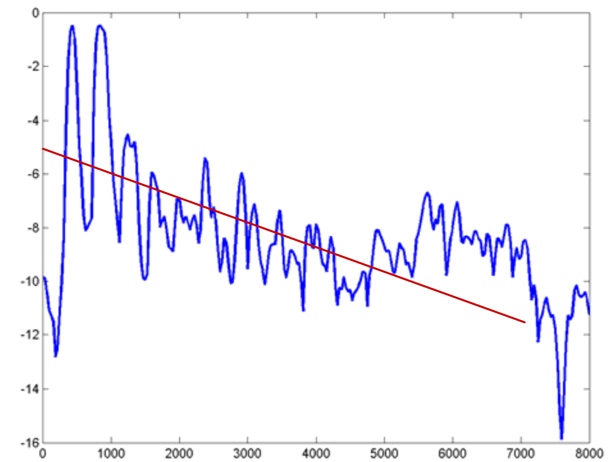
- The spectrum of the speech signal naturally has lower energy at higher frequencies

- This can be observed as a downward trend on a plot of the logarithm of the magnitude spectrum of the signal

- For many applications this can be undesirable
  - E.g. Linear predictive modeling of the spectrum



Log(average(magnitude spectrum))

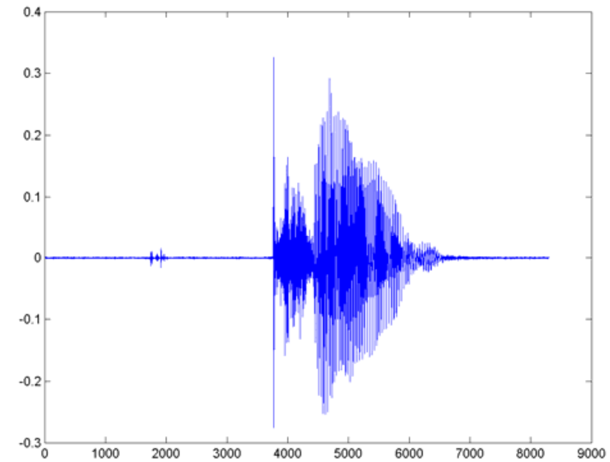


Preemphasizing a speech signal

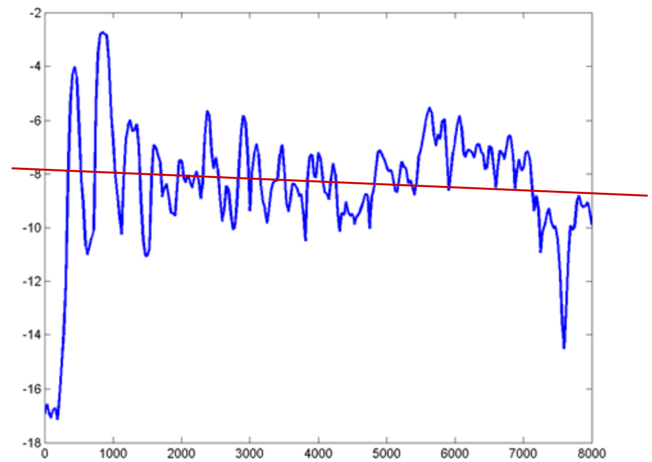
- This spectral tilt can be corrected by preemphasizing the signal
  - $s_{preemp}[n] = s[n] - \alpha * s[n-1]$
  - Typical value of $\alpha = 0.95$

- This is a form of differentiation that boosts high frequencies

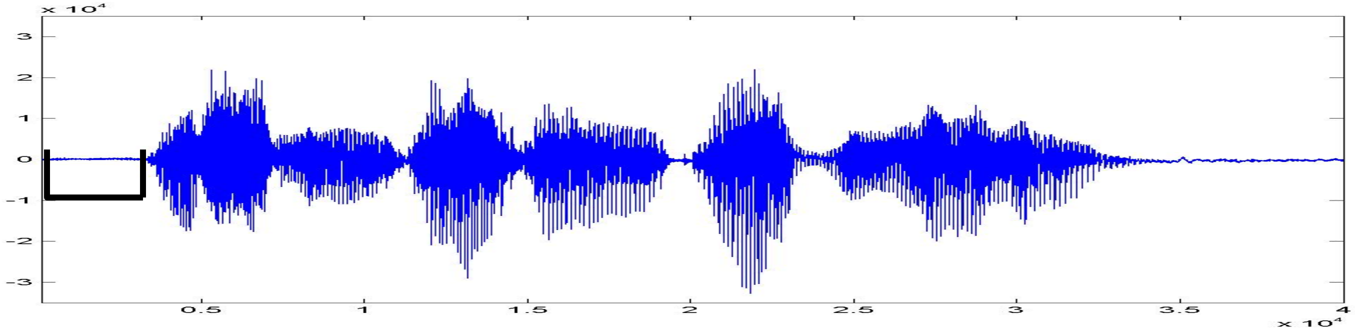
- This spectrum of the preemphasized signal has a more horizontal trend
  - Good for linear prediction and other similar methods



Log(average(magnitude spectrum))

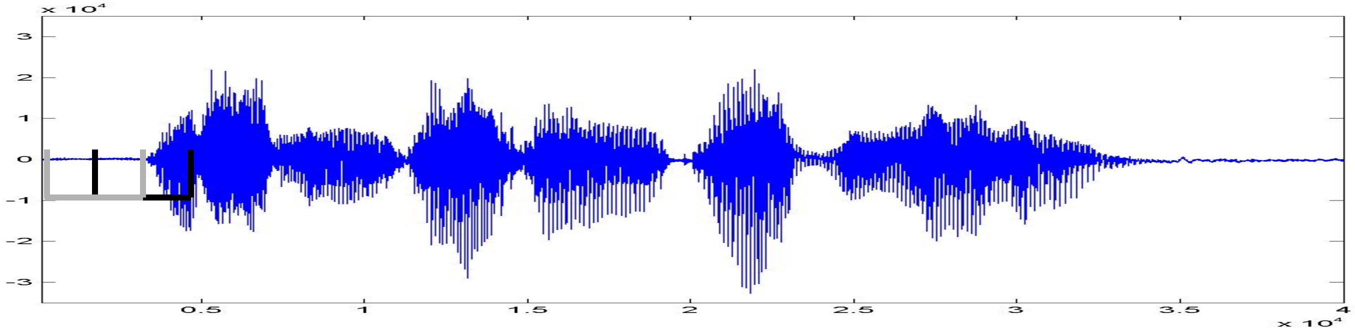


The process of parametrization



The signal is processed in segments.
Segments are typically 25 ms wide.

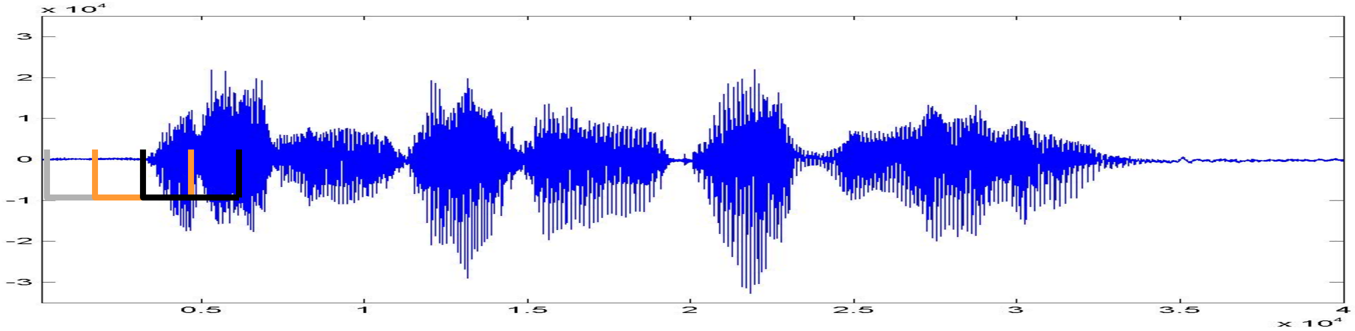
The process of parametrization



The signal is processed in segments. Segments are typically 25 ms wide.

Adjacent segments typically overlap by 15 ms.

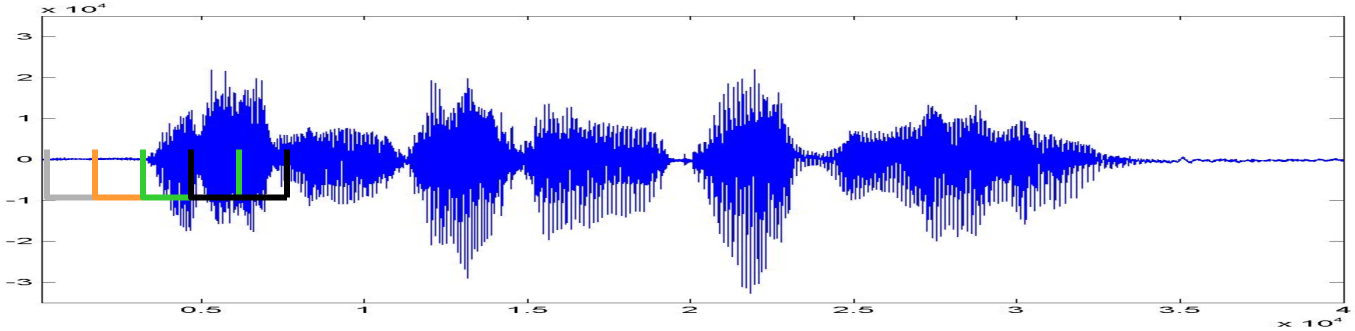
The process of parametrization



The signal is processed in segments.
Segments are typically 25 ms wide.

Adjacent segments typically overlap by 15 ms.

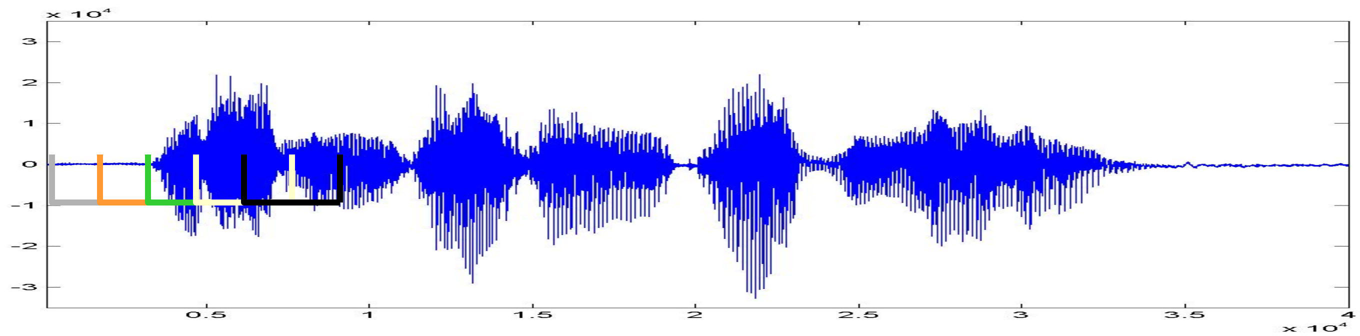
The process of parametrization



The signal is processed in segments. Segments are typically 25 ms wide.

Adjacent segments typically overlap by 15 ms.

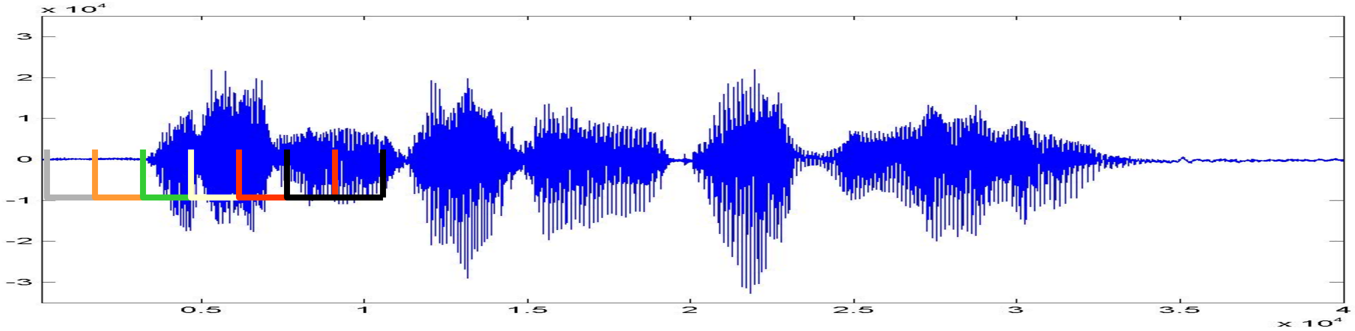
The process of parametrization



The signal is processed in segments. Segments are typically 25 ms wide.

Adjacent segments typically overlap by 15 ms.

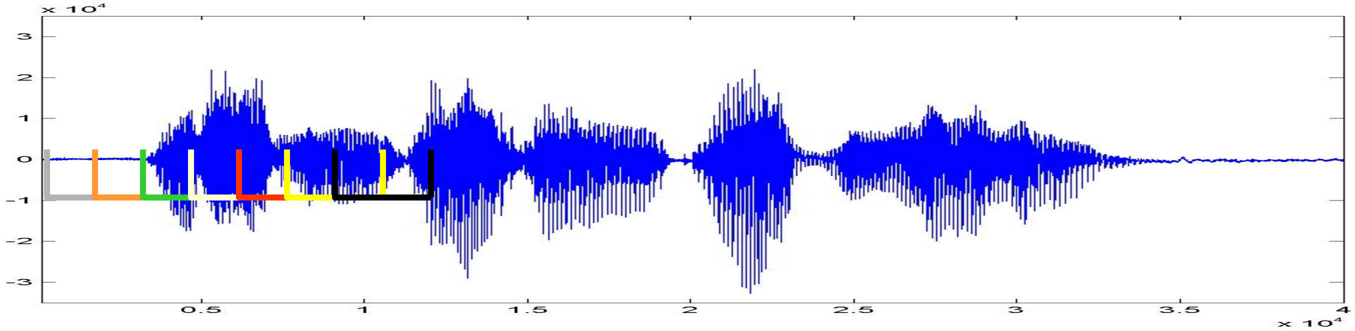
The process of parametrization



The signal is processed in segments. Segments are typically 25 ms wide.

Adjacent segments typically overlap by 15 ms.

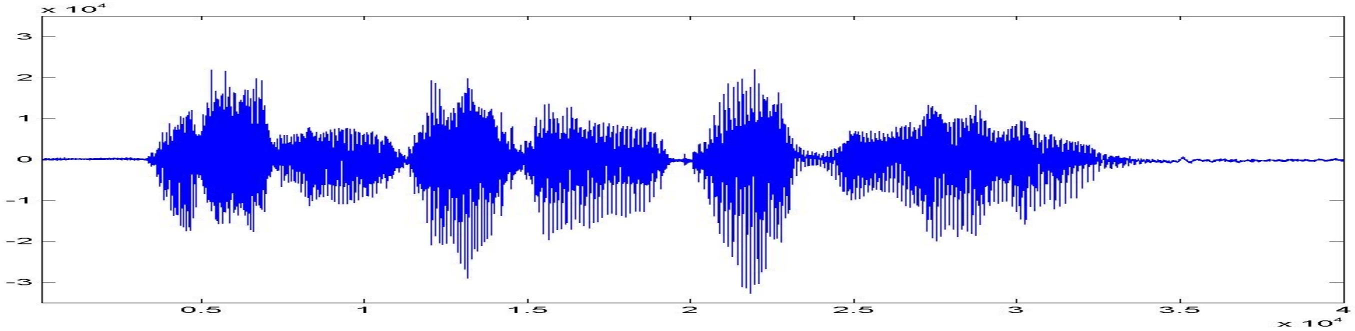
The process of parametrization

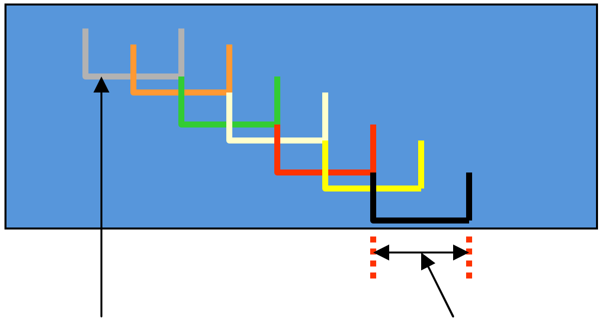


The signal is processed in segments. Segments are typically 25 ms wide.

Adjacent segments typically overlap by 15 ms.

The process of parametrization



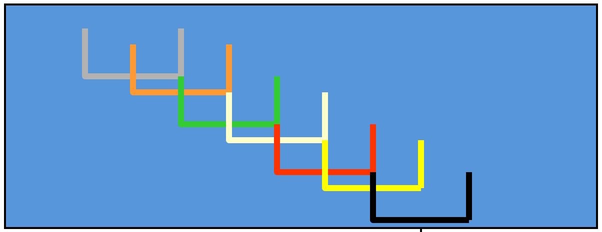


Segments shift every 10 milliseconds

Each segment is typically 20 or 25 milliseconds wide

Speech signals do not change significantly within this short time interval

The process of parametrization



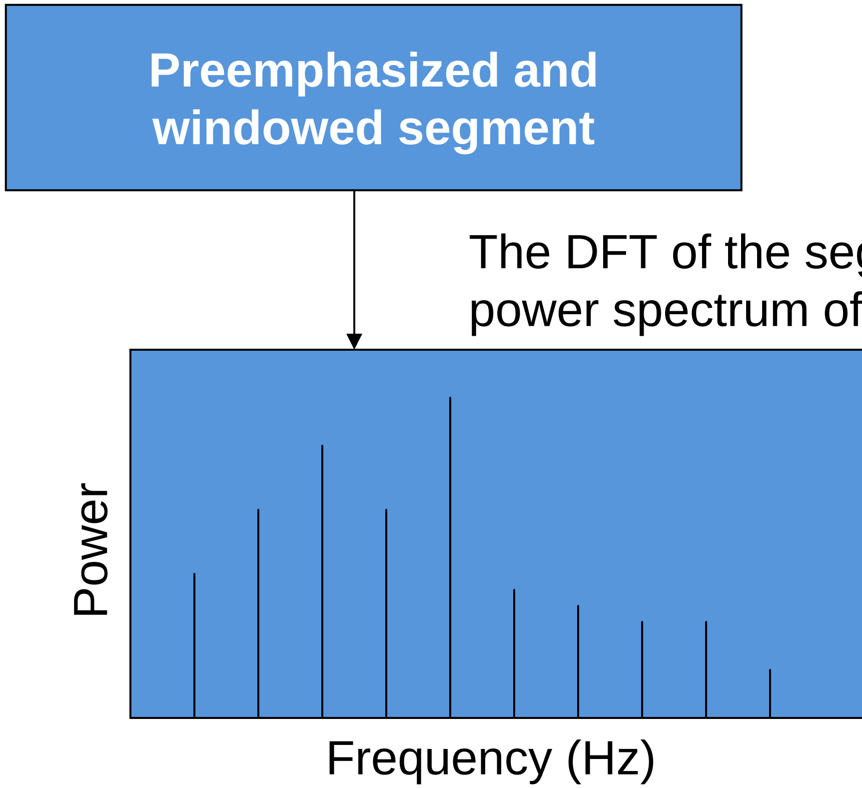
Each segment is preemphasized

Preemphasized segment

The preemphasized segment is windowed

Preemphasized and windowed segment

The process of parametrization



The DFT of the segment, and from it the power spectrum of the segment is computed

= power spectrum

Auditory Perception

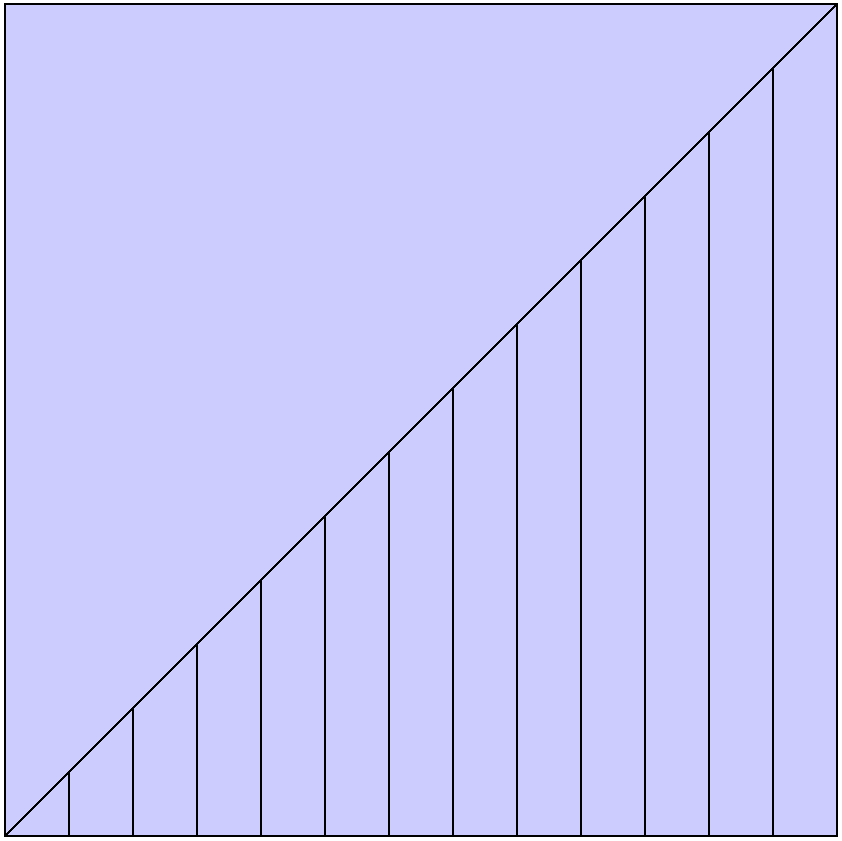
- Conventional Spectral analysis decomposes the signal into a number of linearly spaced frequencies
  - The resolution (differences between adjacent frequencies) is the same at all frequencies
- The human ear, on the other hand, has non-uniform resolution
  - At low frequencies we can detect small changes in frequency
  - At high frequencies, only gross differences can be detected
- Feature computation must be performed with similar resolution
  - Since the information in the speech signal is also distributed in a manner matched to human perception

Matching Human Auditory Response

- Modify the spectrum to model the frequency resolution of the human ear

- *Warp* the frequency axis such that small differences between frequencies at lower frequencies are given the same importance as larger differences at higher frequencies

Warping the frequency axis

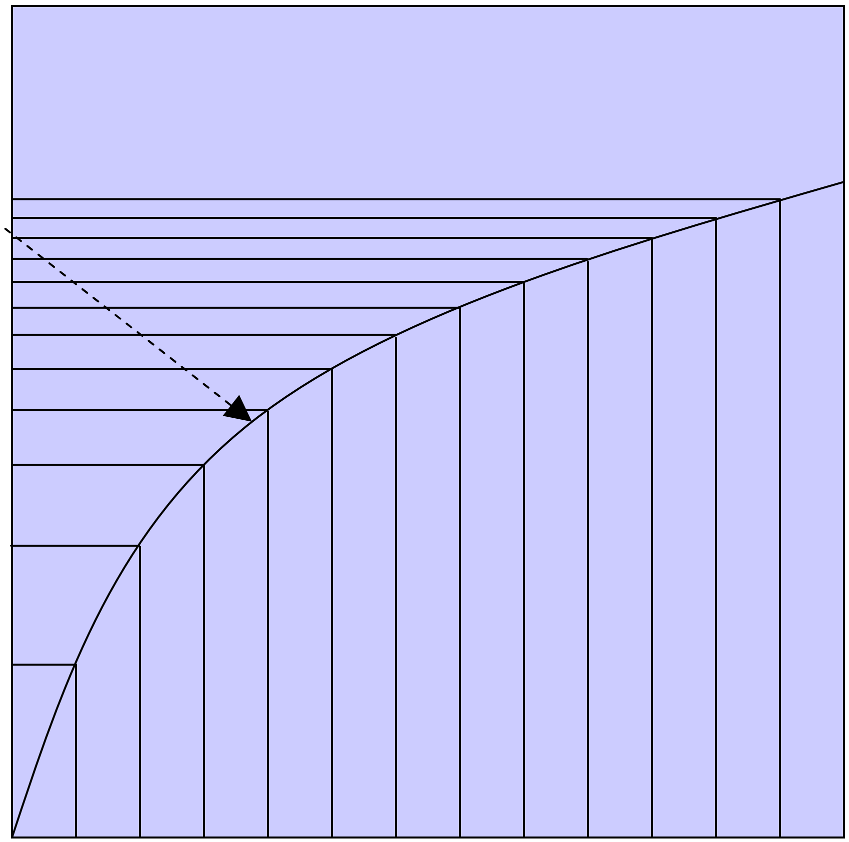


Linear frequency axis: equal increments of frequency at equal intervals

Warping the frequency axis

Warping function
(based on studies of
human hearing)

Warped frequency
axis: unequal increments
of frequency at equal
intervals or conversely,
equal increments of
frequency at unequal
intervals



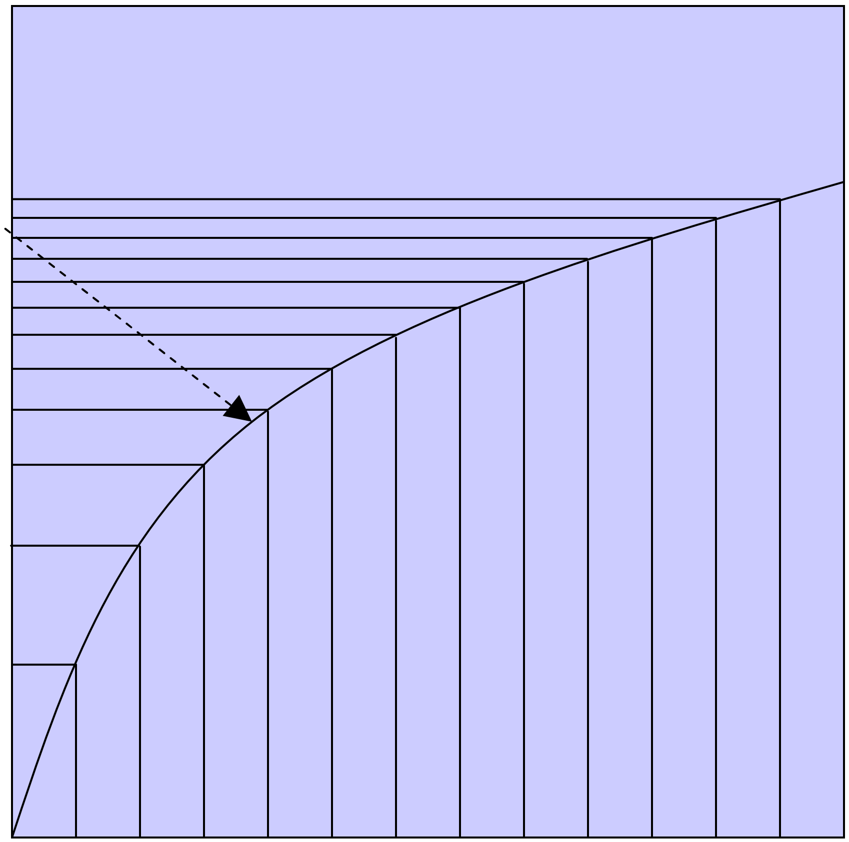
Linear frequency axis:
Sampled at uniform
intervals by an FFT

Warping the frequency axis

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

Warping function (based on studies of human hearing)

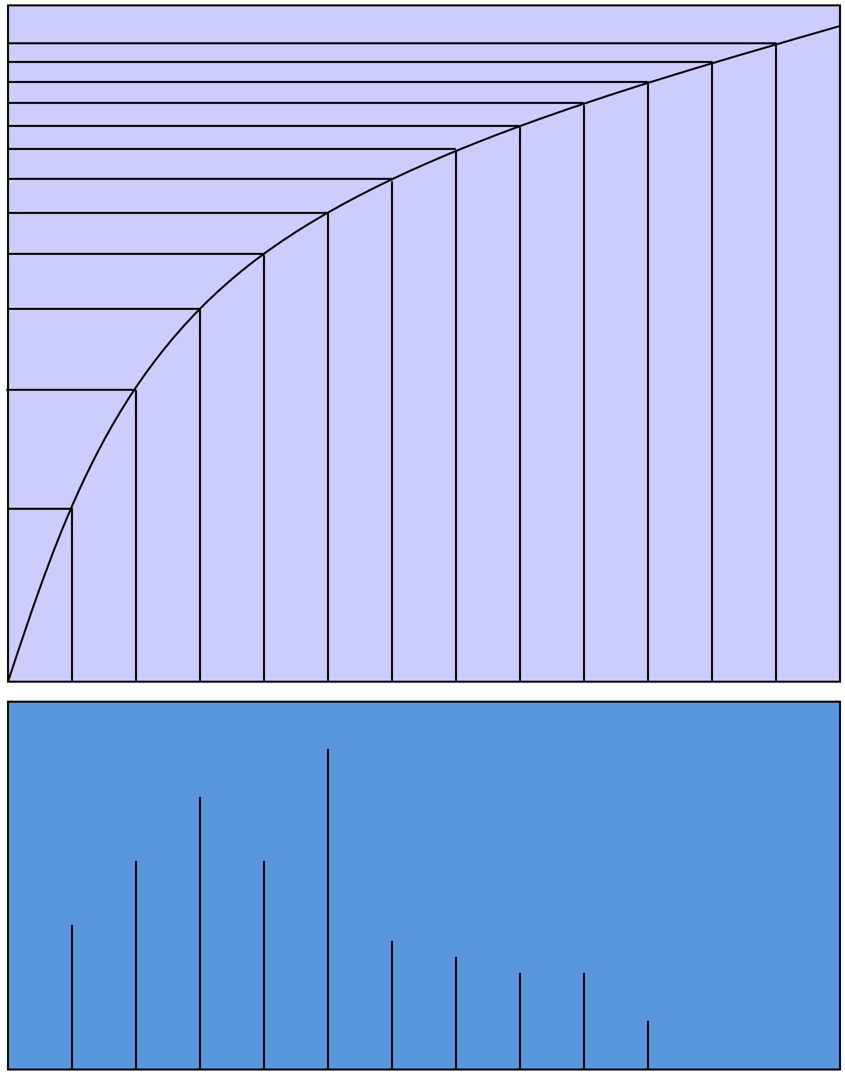
Warped frequency axis: unequal increments of frequency at equal intervals or conversely, equal increments of frequency at unequal intervals



A standard warping function is the Mel warping function

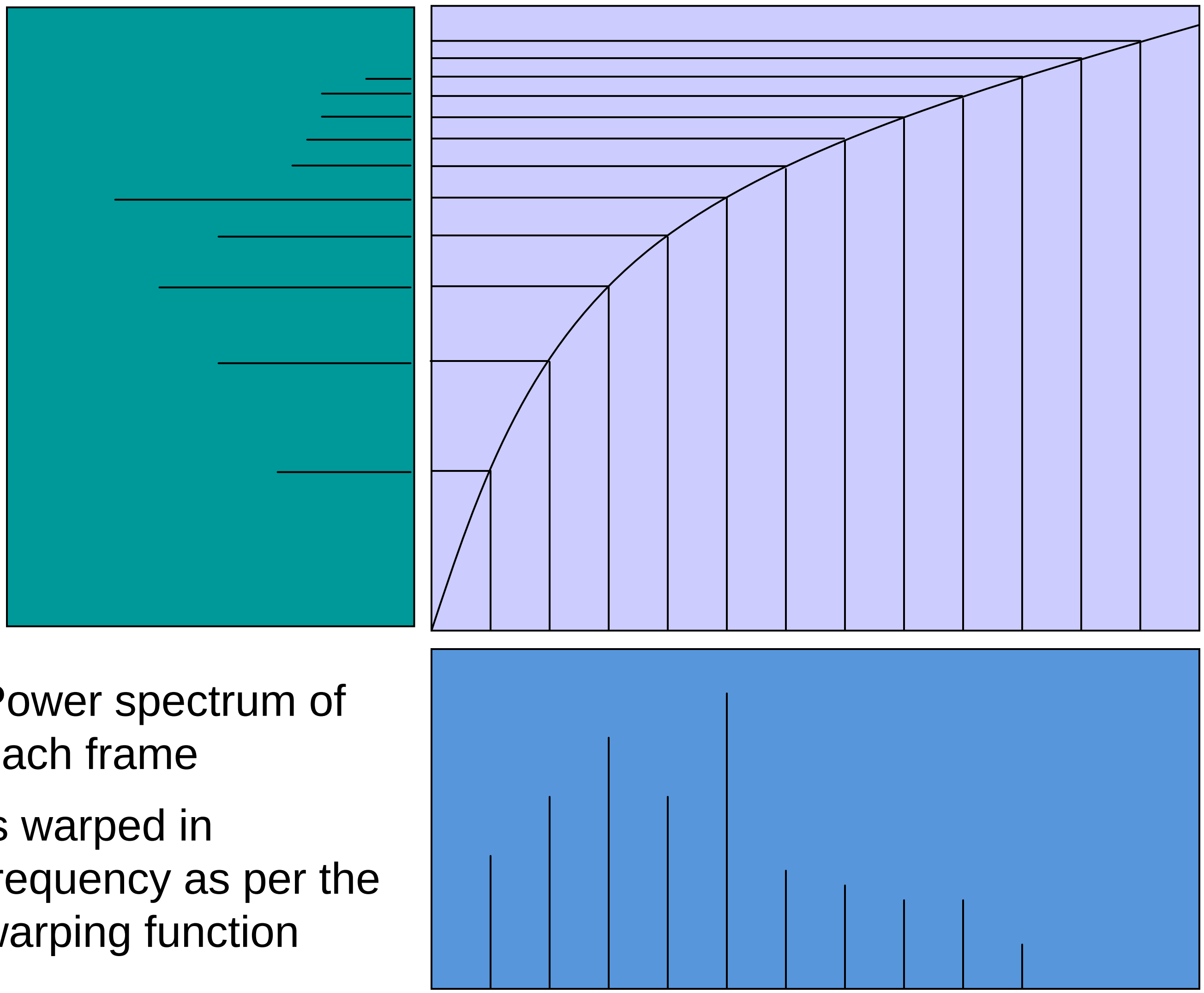
Linear frequency axis: Sampled at uniform intervals by an FFT

The process of parametrization



Power spectrum of each frame

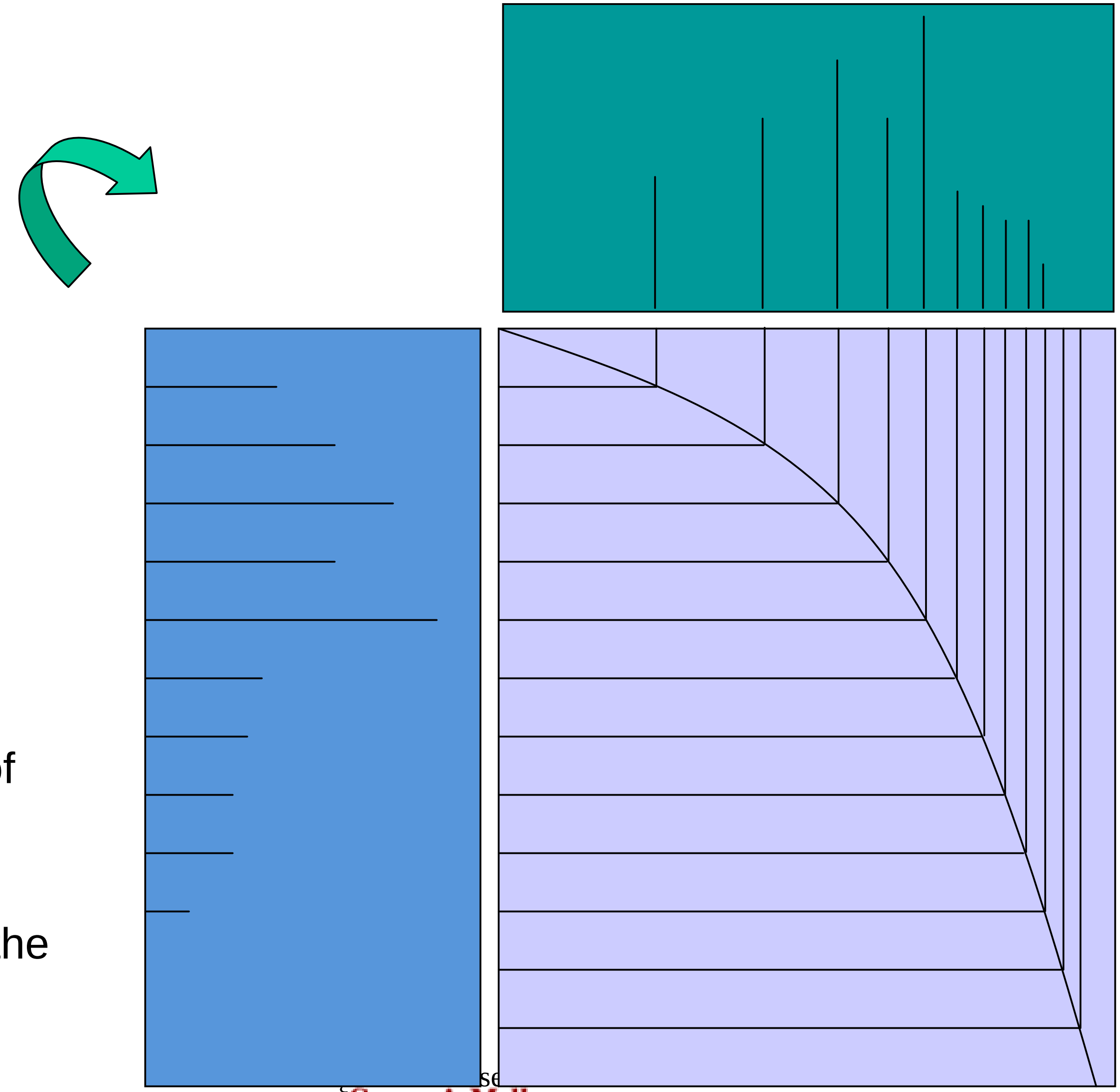
The process of parametrization



Power spectrum of each frame

is warped in frequency as per the warping function

The process of parametrization



Power spectrum of each frame

is warped in frequency as per the warping function

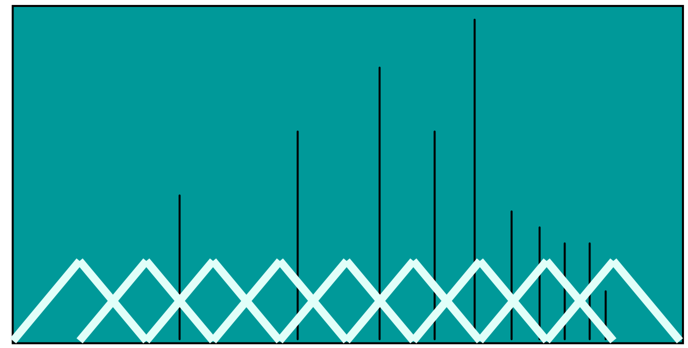
Filter Bank

- Each hair cell in the human ear actually responds to a *band* of frequencies, with a peak response at a particular frequency

- To mimic this, we apply a bank of “auditory” filters
  - Filters are triangular
    - An approximation: hair cell response is not triangular
  - A small number of filters (40)
    - Far fewer than hair cells (~3000)

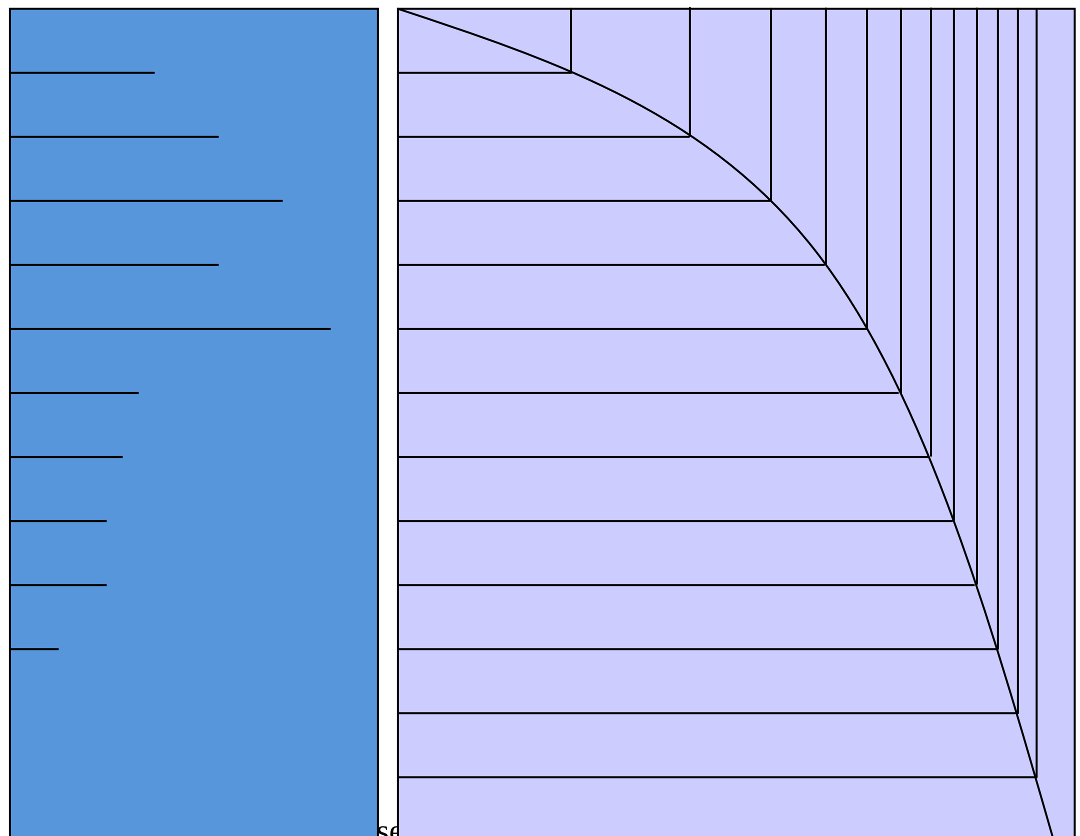
The process of parametrization

Each intensity is weighted by the value of the filter at that frequncy. This picture shows a bank or collection of triangular filters that overlap by 50%

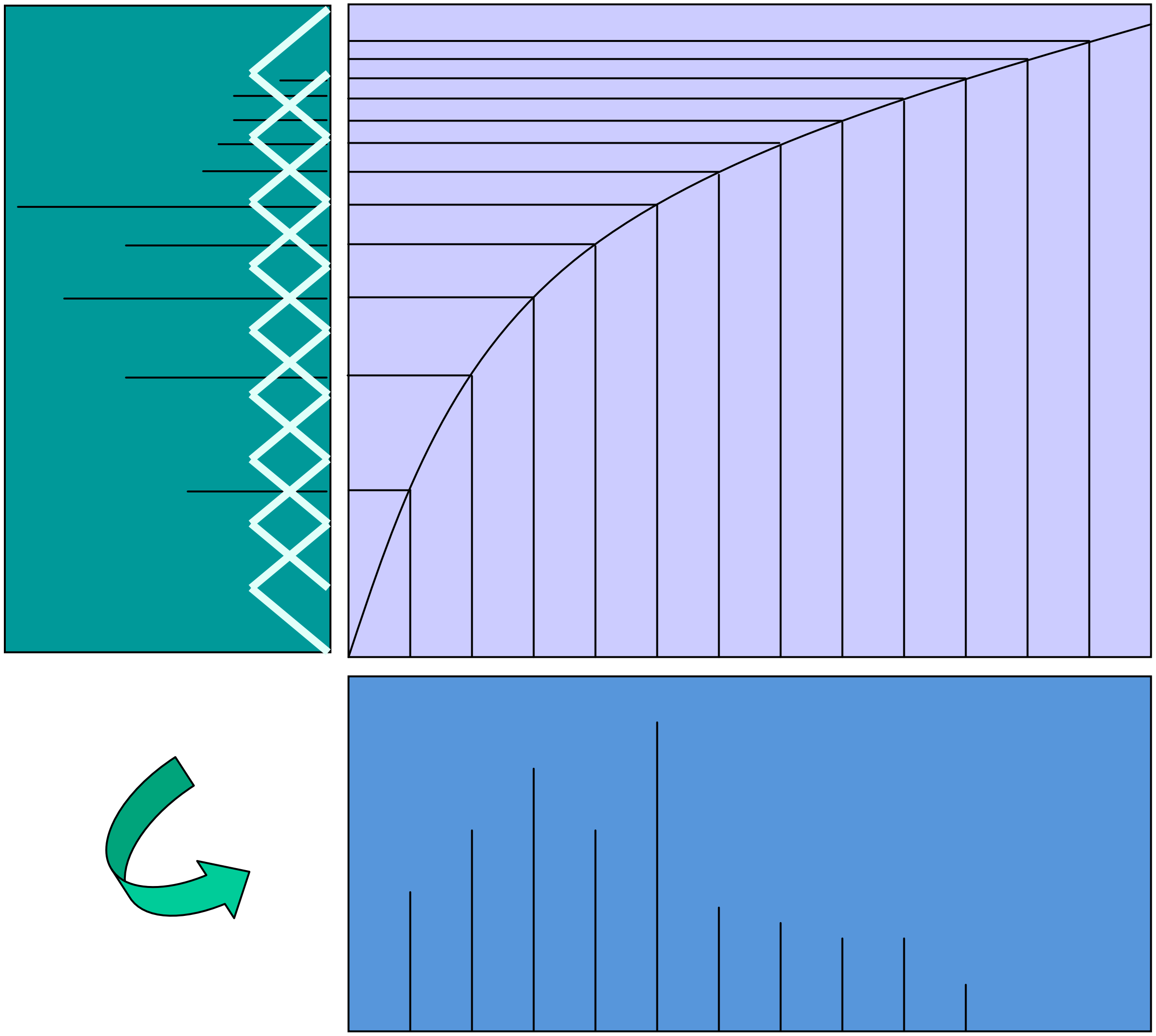


Power spectrum of each frame

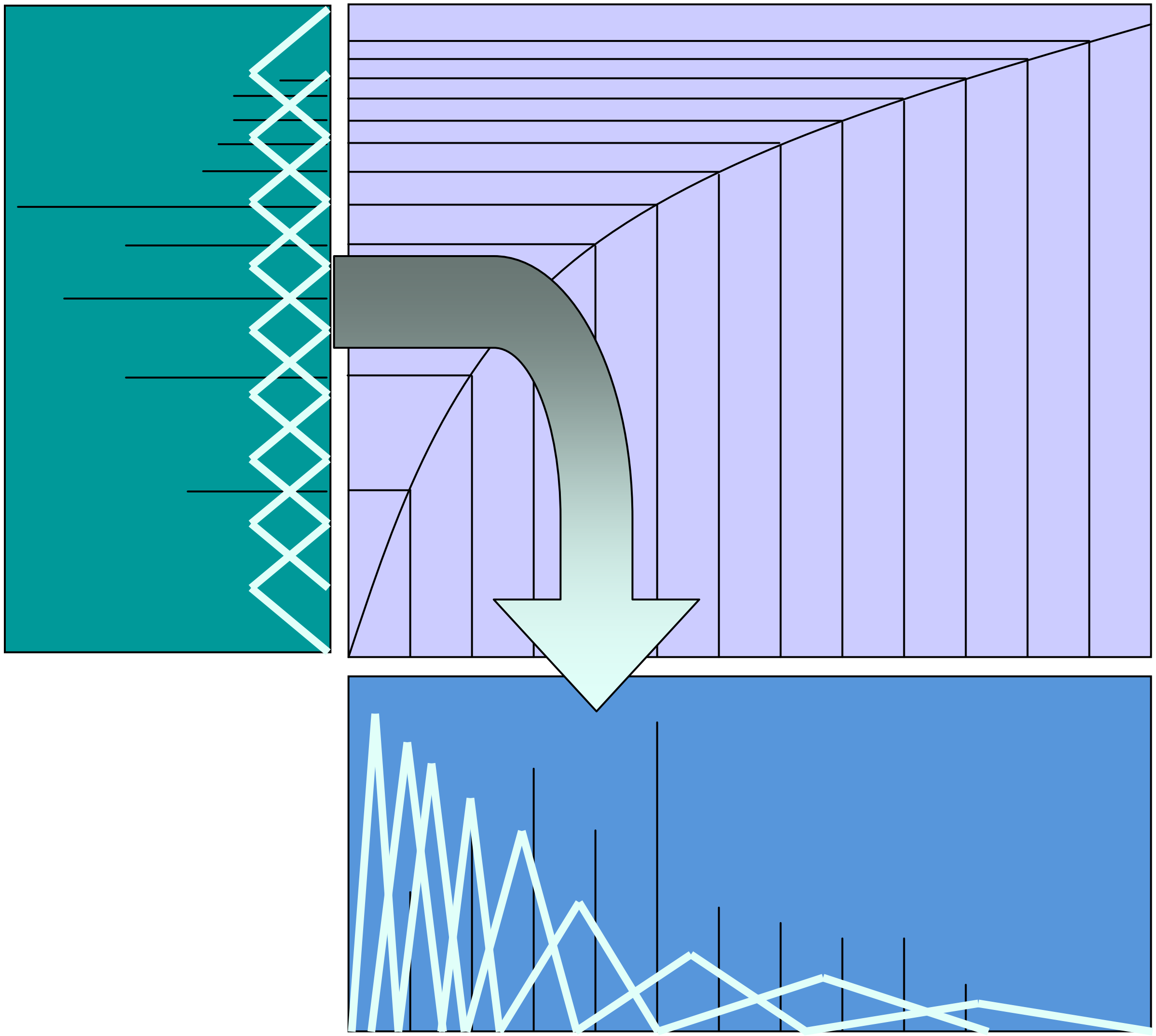
is warped in frequency as per the warping function



The process of parametrization

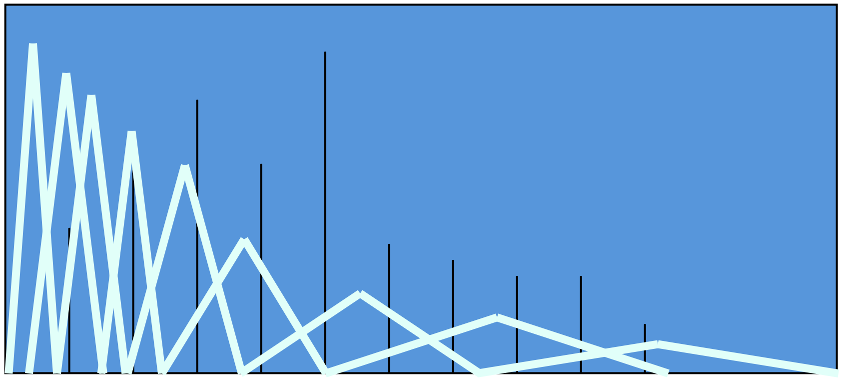


The process of parametrization



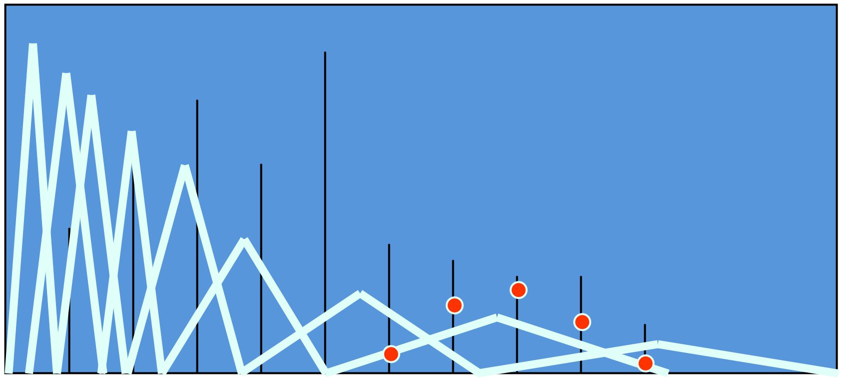
The process of parametrization

For each filter:
Each power spectral value is weighted by the value of the filter at that frequency.

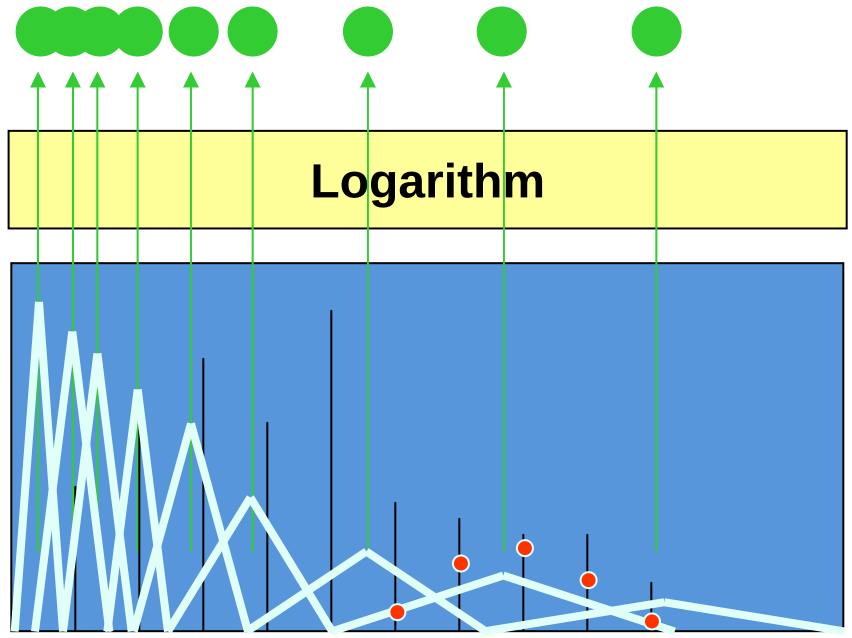


The process of parametrization

For each filter:
All weighted spectral values are integrated (added), giving one value for the filter



The process of parametrization

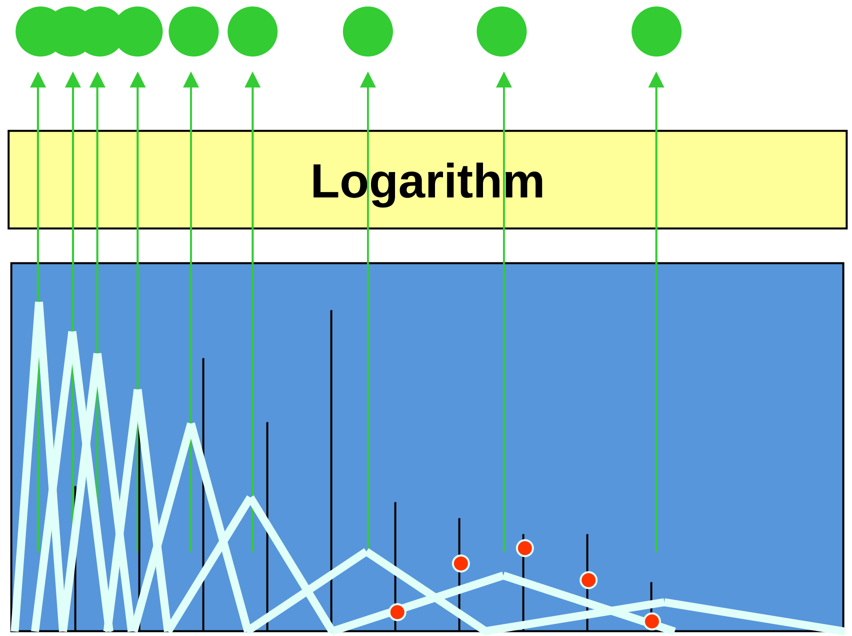


All weighted spectral values for each filter are integrated (added), giving one value per filter

Additional Processing

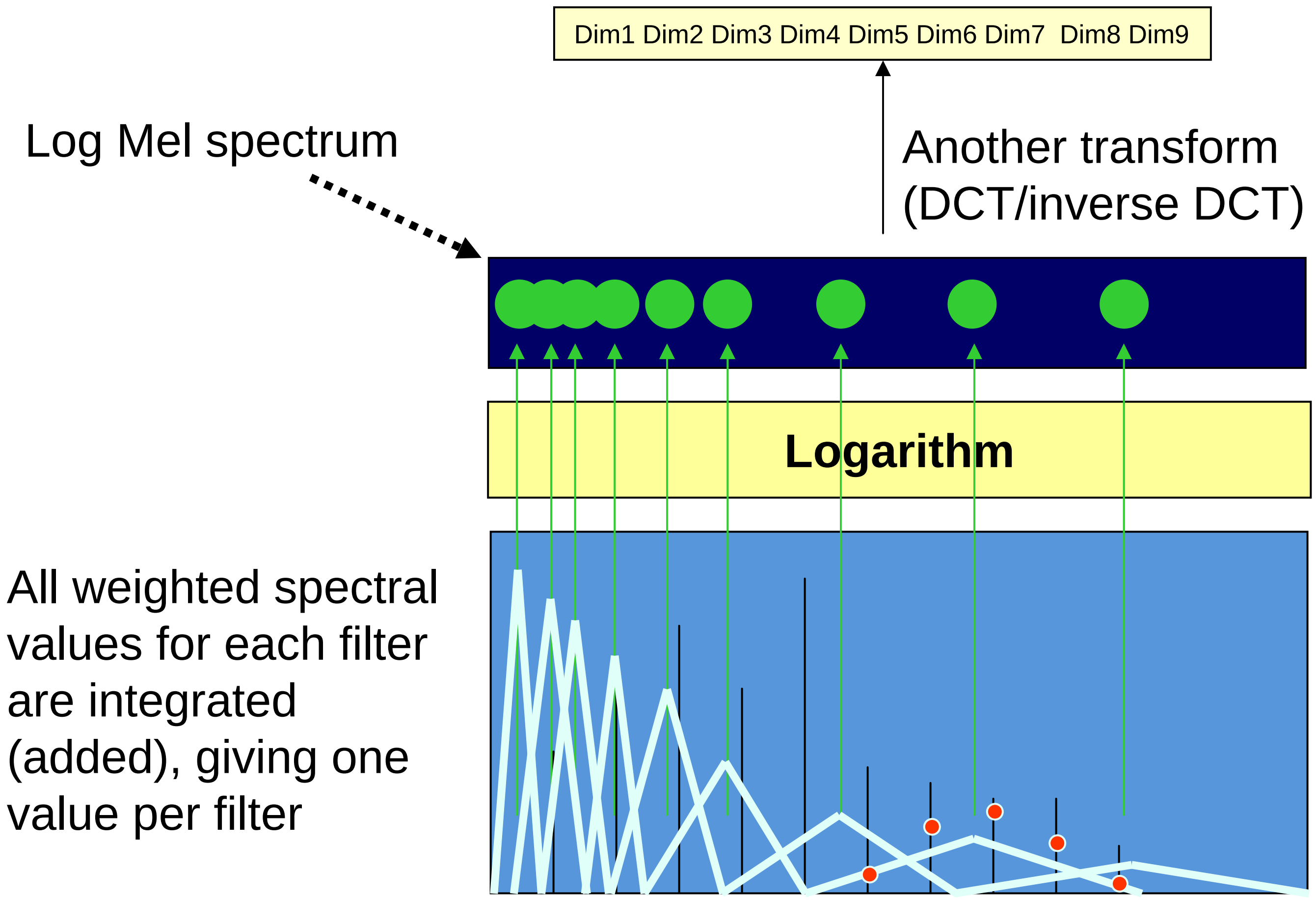
- The Mel spectrum represents energies in frequency bands
  - Highly unequal in different bands
    - Energy and variations in energy are both much much greater at lower frequencies
    - May dominate any pattern classification or template matching scores
  - High-dimensional representation: many filters
- Compress the energy values to reduce imbalance
- Reduce dimensions for computational tractability
  - Also, for generalization: reduced dimensional representations have lower variations across speakers for any sound

The process of parametrization

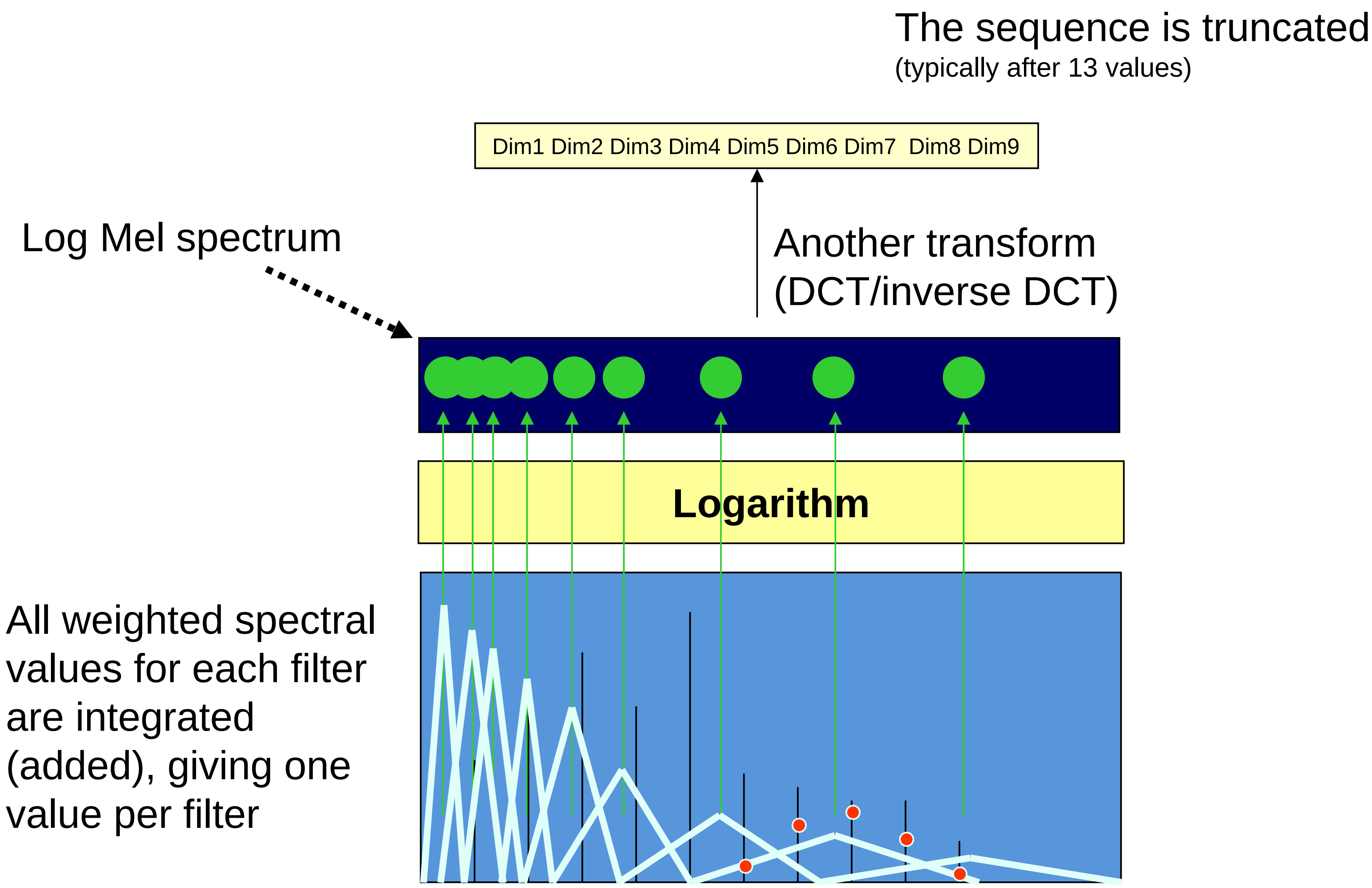


All weighted spectral values for each filter are integrated (added), giving one value per filter

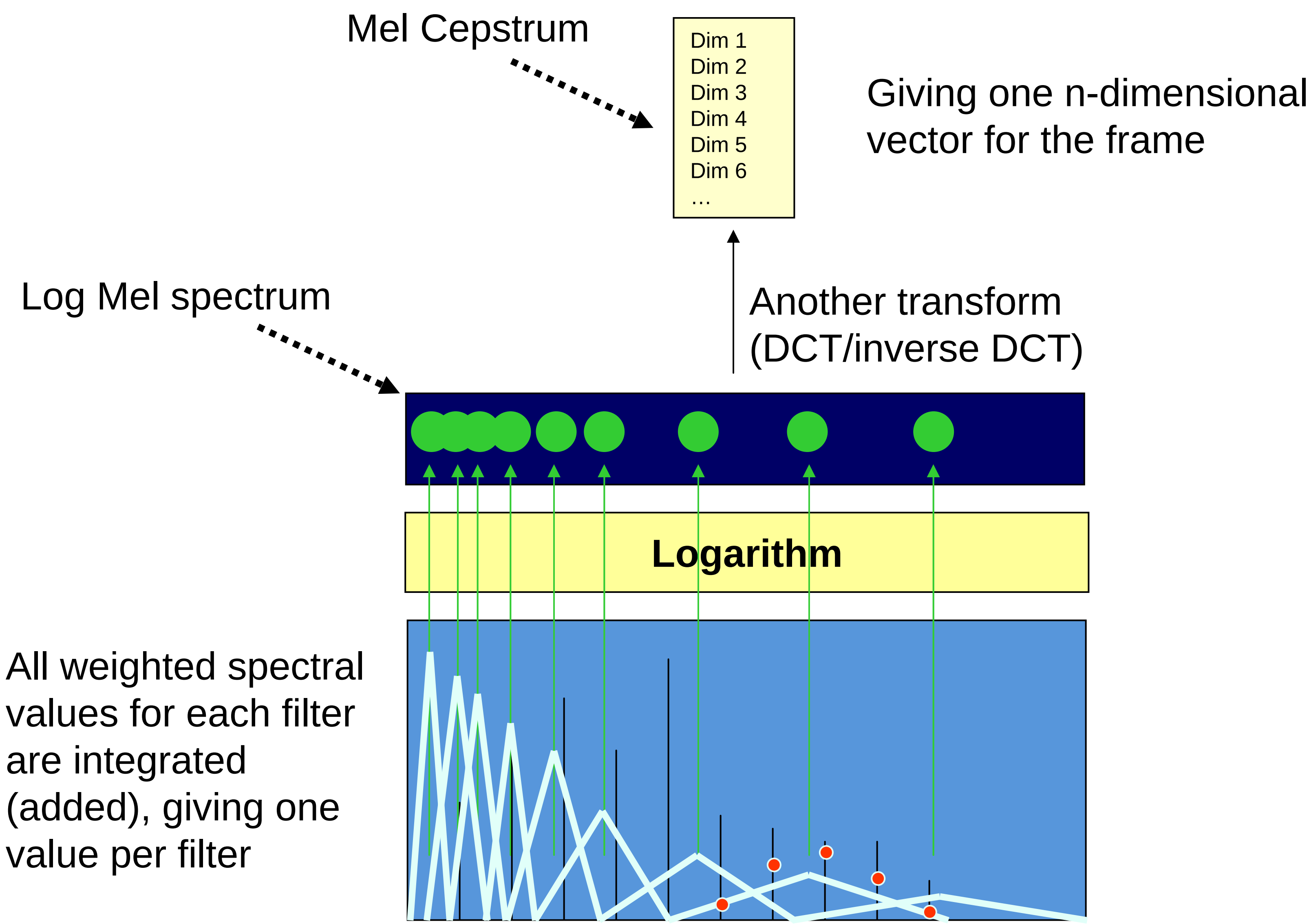
The process of parametrization



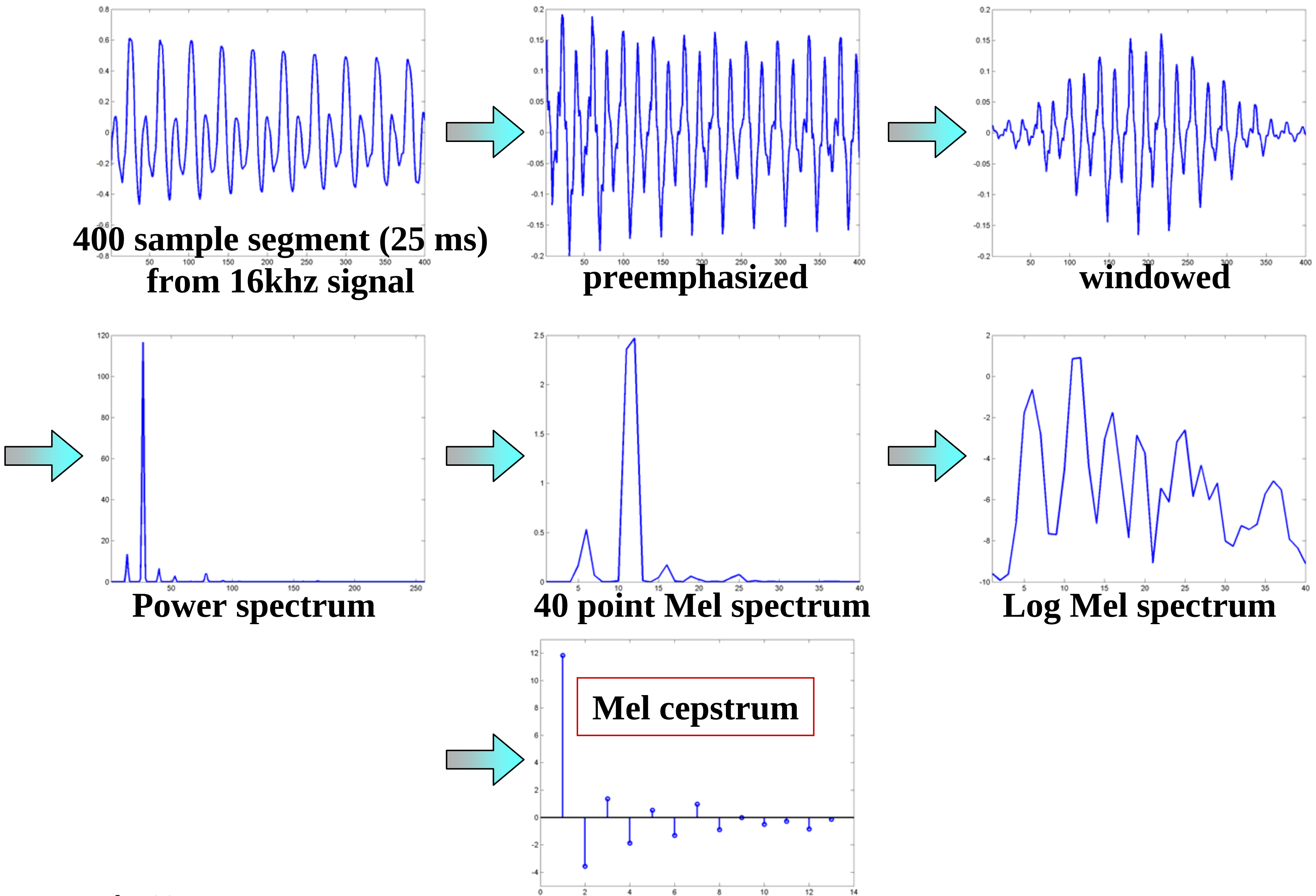
The process of parametrization



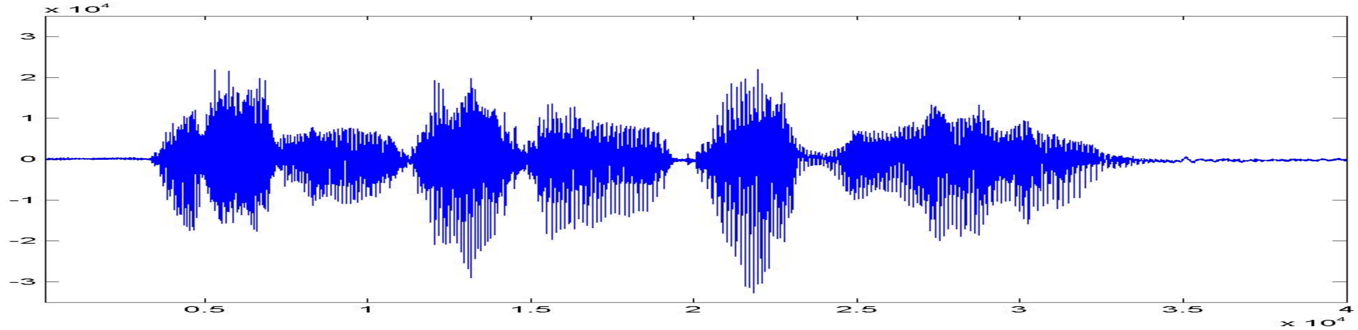
The process of parametrization

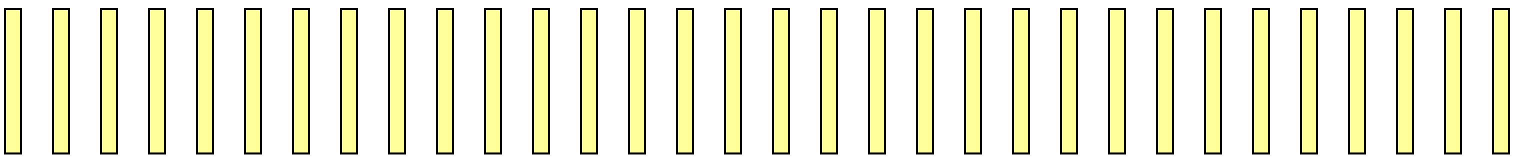


An example segment



The process of feature extraction





The entire speech signal is thus converted into a sequence of vectors. These are cepstral vectors.
There are other ways of converting the speech signal into a sequence of vectors

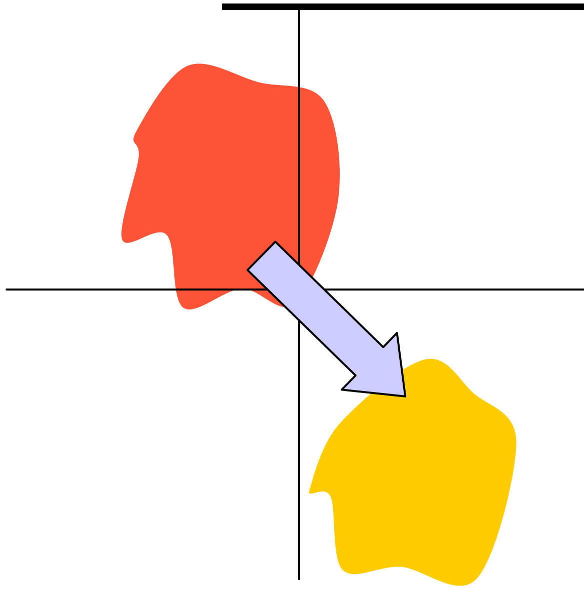
Variations to the basic theme

- Perceptual Linear Prediction (PLP) features:
  - ERB filters instead of MEL filters
  - Cube-root compression instead of Log
  - Linear-prediction spectrum instead of Fourier Spectrum
- Auditory features
  - Detailed and painful models of various components of the human ear

Cepstral Variations from Filtering and Noise

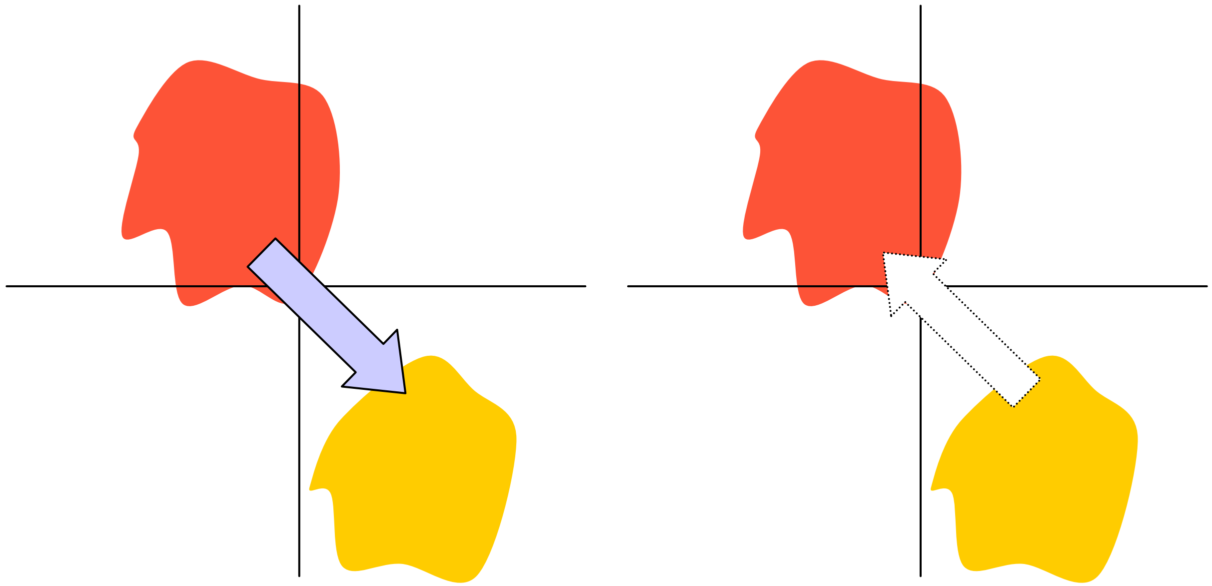
- Microphone characteristics modify the spectral characteristics of the captured signal
  – They change the value of the cepstra
- Noise too modifies spectral characteristics
- As do speaker variations
- All of these change the distribution of the cepstra

Effect of Speaker Variations, Microphone Variations, Noise etc.



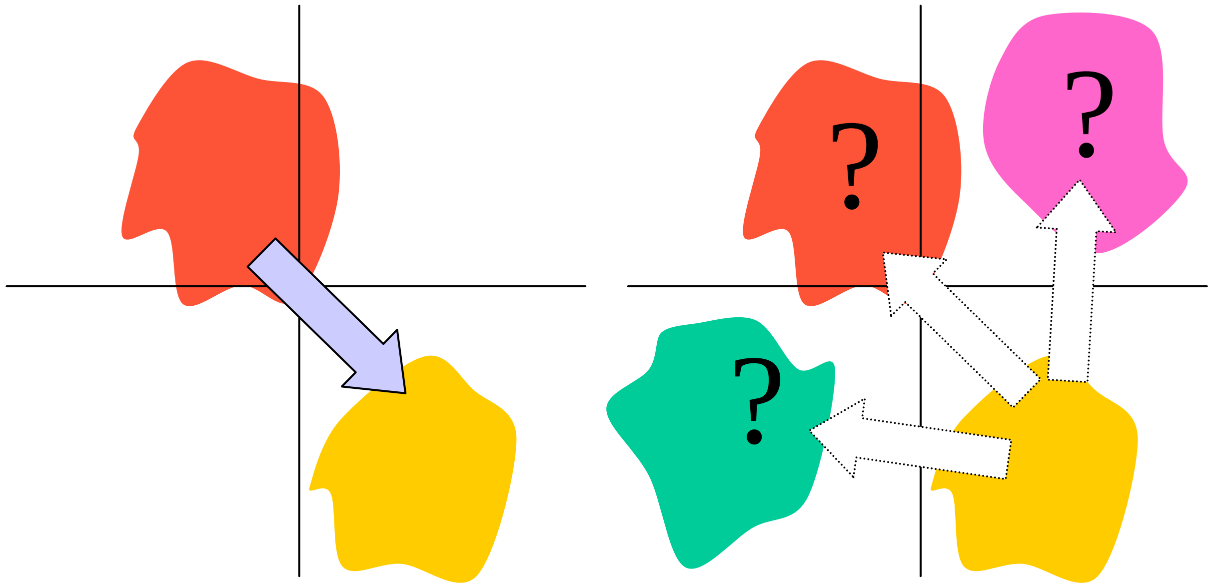
- Noise, channel and speaker variations change the *distribution* of cepstral values

Ideal Correction for Variations



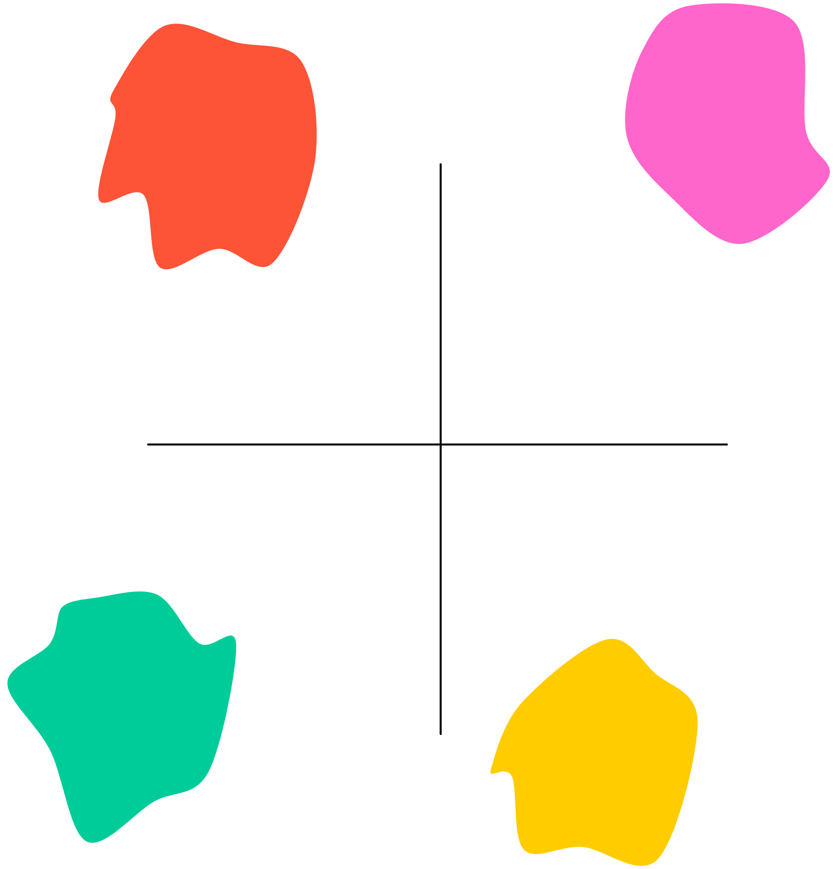
- Noise, channel and speaker variations change the *distribution* of cepstral values
- To compensate for these, we would like to undo these changes to the distribution

Effect of Noise Etc.



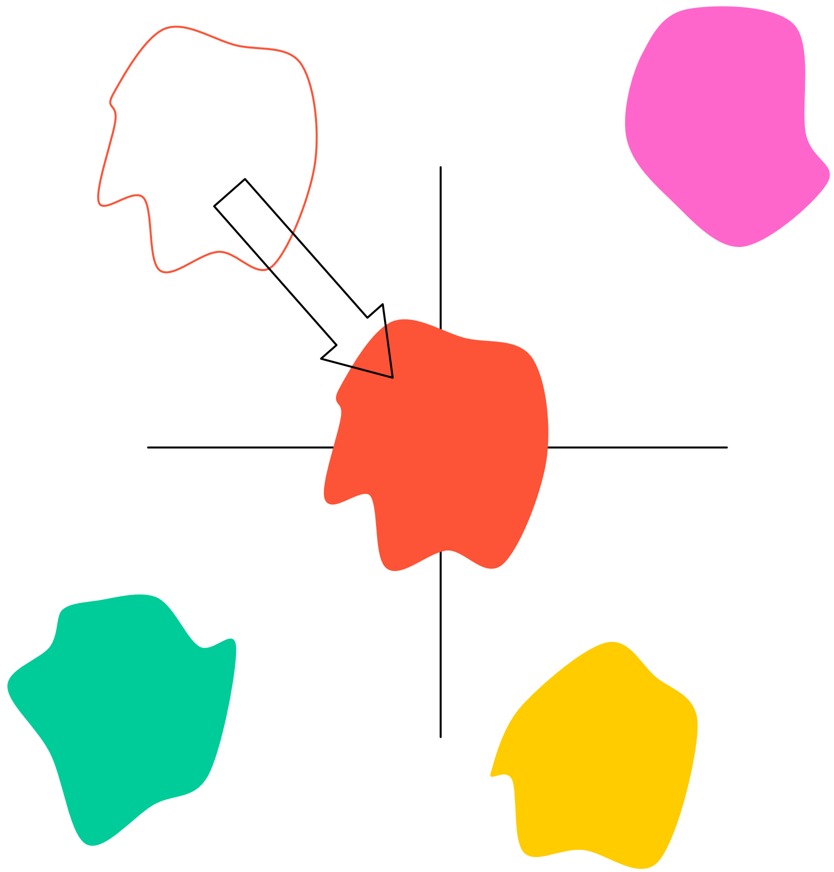
- Noise, channel and speaker variations change the *distribution* of cepstral values
- To compensate for these, we would like to undo these changes to the distribution
- Unfortunately, the precise position of the distributions of the "good" speech is hard to know

Solution: Move all distributions to a "standard" location



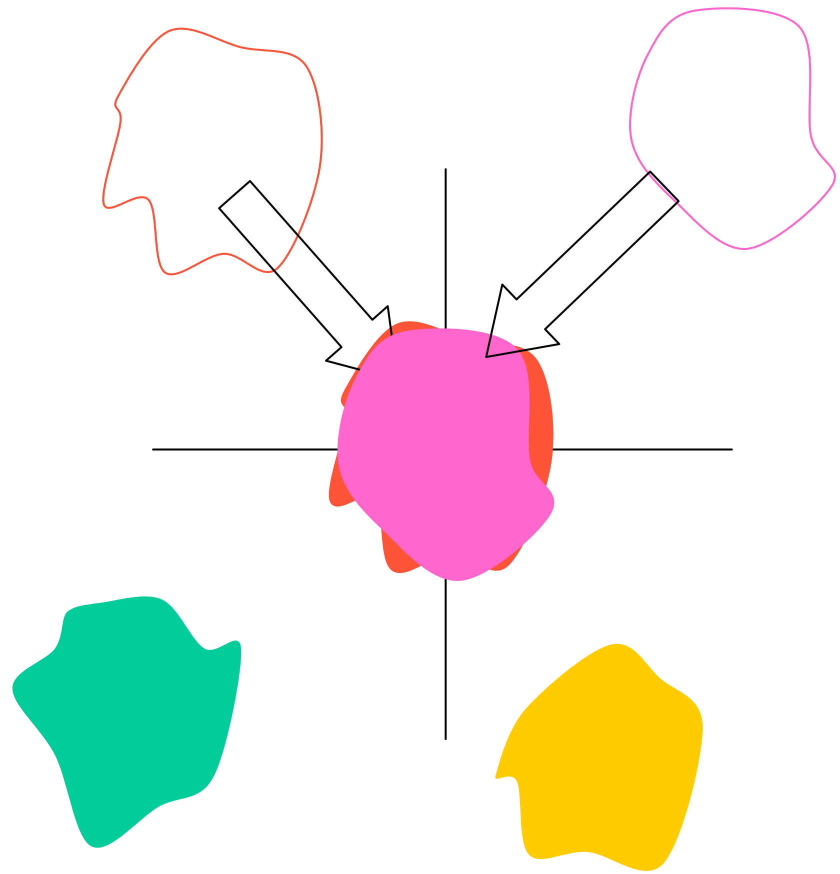
- "Move" all utterances to have a mean of 0
- This ensures that all the data is centered at 0
  - Thereby eliminating *some* of the mismatch

Solution: Move all distributions to a “standard” location



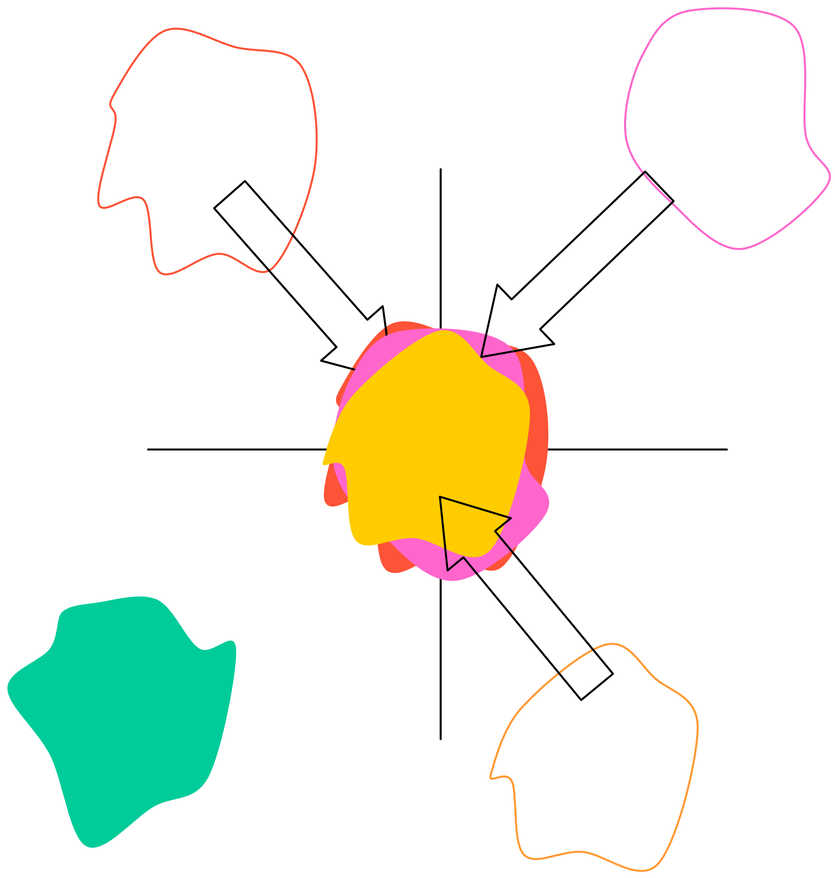
- “Move” all utterances to have a mean of 0
- This ensures that all the data is centered at 0
  - Thereby eliminating *some* of the mismatch

Solution: Move all distributions to a “standard” location



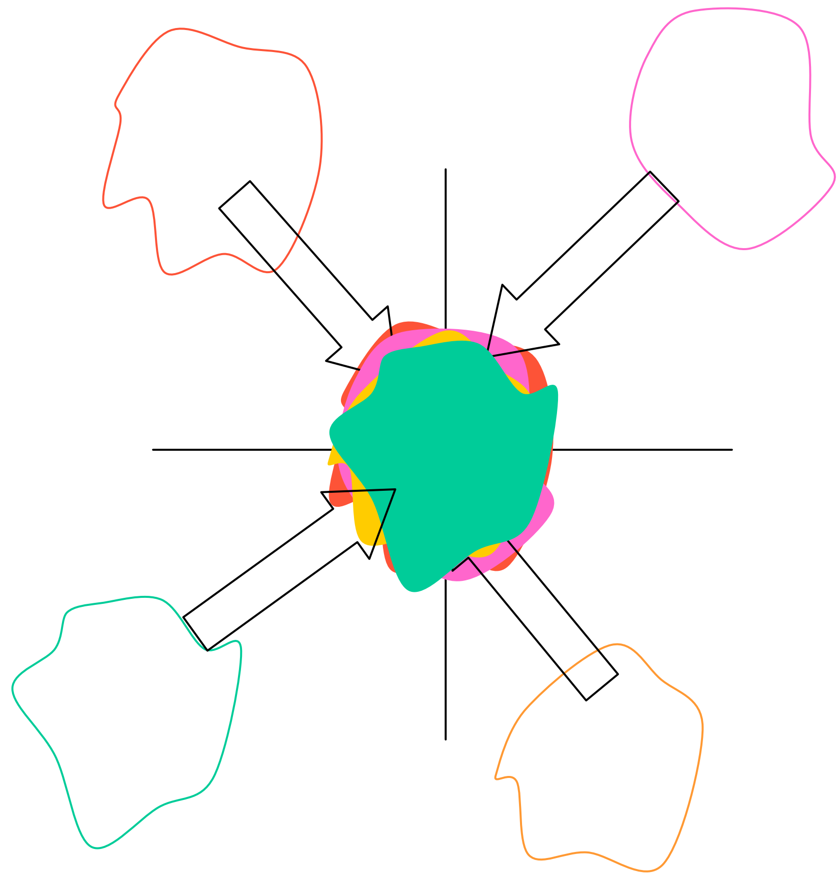
- “Move” all utterances to have a mean of 0
- This ensures that all the data is centered at 0
  - Thereby eliminating *some* of the mismatch

Solution: Move all distributions to a "standard" location



- "Move" all utterances to have a mean of 0
- This ensures that all the data is centered at 0
  - Thereby eliminating *some* of the mismatch

Solution: Move all distributions to a "standard" location



- "Move" all utterances to have a mean of 0
- This ensures that all the data is centered at 0
  - Thereby eliminating *some* of the mismatch

Cepstra Mean Normalization

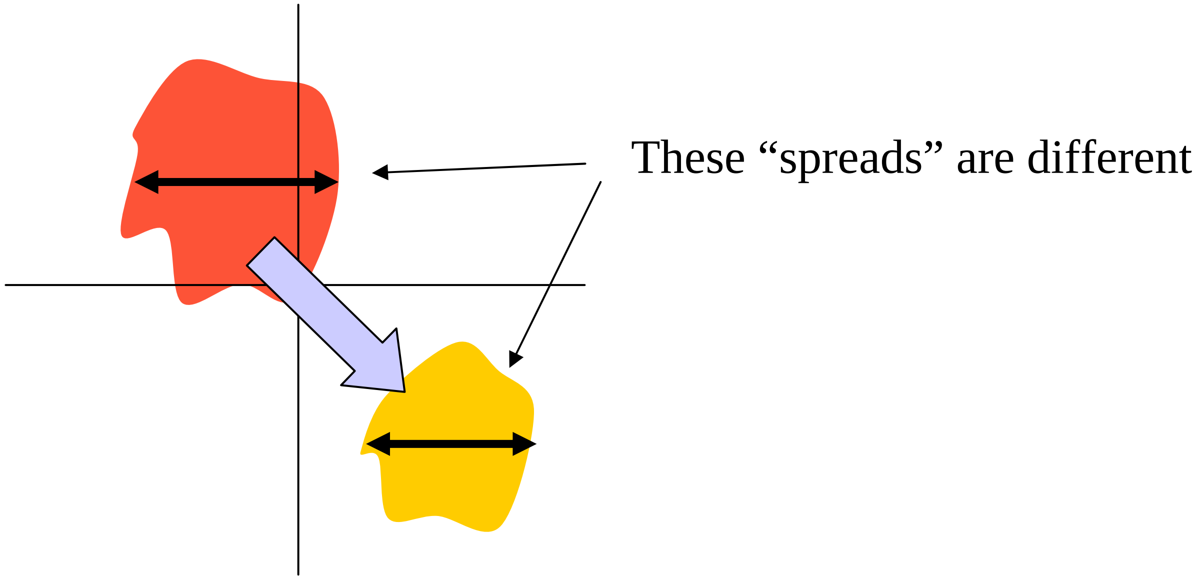
- For each utterance encountered (both in “training” and in “testing”)
- Compute the mean of all cepstral vectors

$$M_{recording} = \frac{1}{Nframes} \sum_t c_{recording}(t)$$

- Subtract the mean out of all cepstral vectors

$$c_{normalized}(t) = c_{recording}(t) - M_{recording}$$

Variance



- The *variance* of the distributions is also modified by the corrupting factors

- This can also be accounted for by variance normalization

Variance Normalization

- Compute the standard deviation of the mean-normalized cepstra

$$sd_{recording} = \sqrt{\frac{1}{Nframes} \sum_t c_{normalized}(t)}$$

- Divide all mean-normalized cepstra by this standard deviation

$$c_{\mathrm{var}\ normalized}(t) = \frac{1}{sd_{recording}} c_{normalized}(t)$$

- The resultant cepstra for any recording have 0 mean and a variance of 1.0

Histogram Normalization

- Go beyond Variances: Modify the entire distribution
- "Histogram normalization" : make the histogram of every recording be identical
- For each recording, for each cepstral value
  - Compute percentile points
  - Find a warping function that maps these percentile points to the corresponding percentile points on a 0 mean unit variance Gaussian
  - Transform the cepstra according to this function

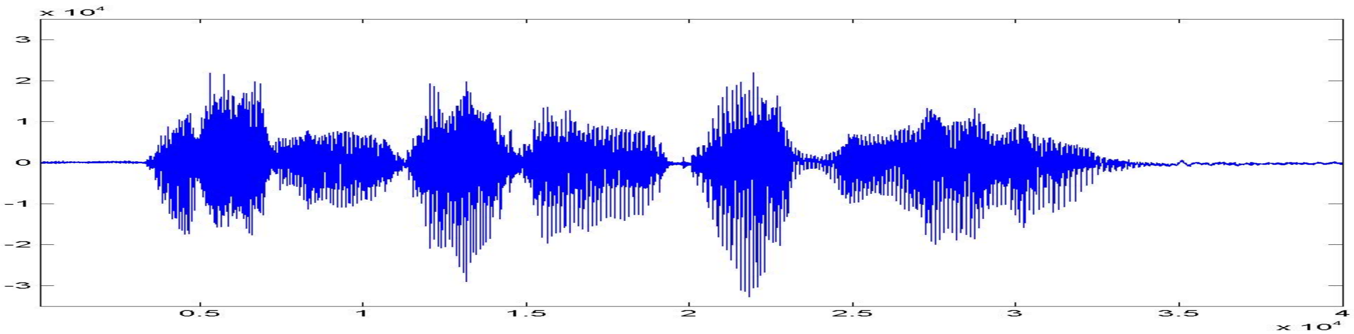
Temporal Variations

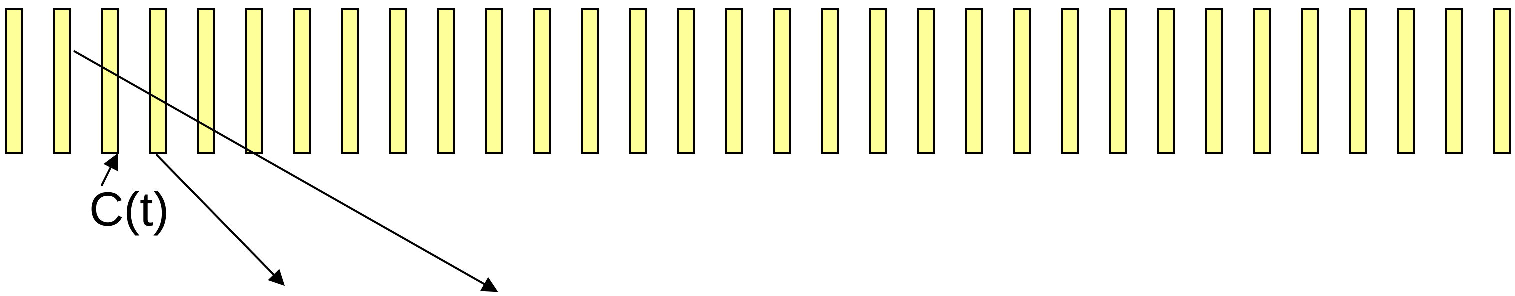
- The cepstral vectors capture instantaneous information only
  - Or, more precisely, current spectral structure within the analysis window
- Phoneme identity resides not just in the snapshot information, but also in the temporal structure
  - Manner in which these values change with time
  - Most characteristic features
    - Velocity: rate of change of value with time
    - Acceleration: rate with which the velocity changes
- These must also be represented in the feature

Velocity Features

- For every component in the cepstrum for any frame
  - compute the difference between the corresponding feature value for the next frame and the value for the previous frame
  - For 13 cepstral values, we obtain 13 “delta” values
- The set of all delta values gives us a “delta feature”

The process of feature extraction



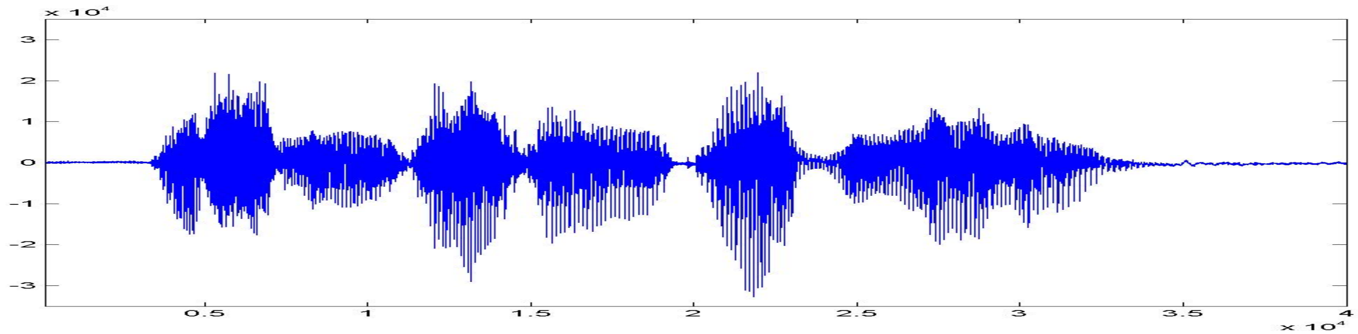


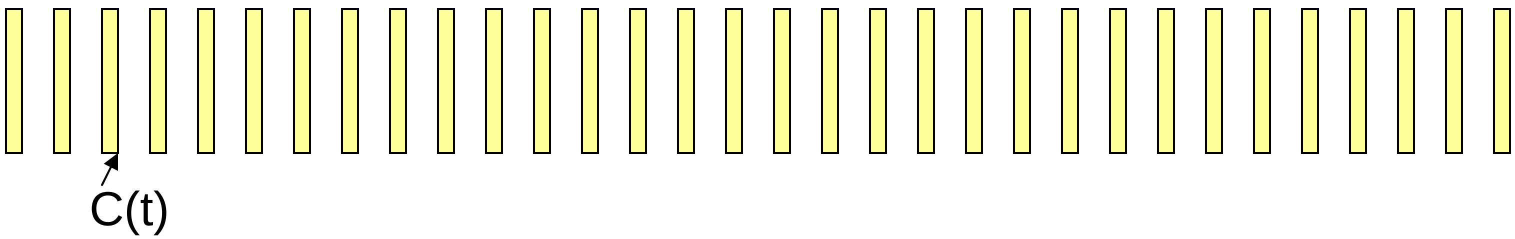
$$\Delta c(t)=c(t+\tau)-c(t-\tau)$$

Representing Acceleration

- The *acceleration* represents the manner in which the velocity changes
- Represented as the derivative of velocity
- The DOUBLE-delta or Acceleration Feature captures this
- For every component in the cepstrum for any frame
  - compute the difference between the corresponding *delta* feature value for the next frame and the *delta* value for the previous frame
  - For 13 cepstral values, we obtain 13 “double-delta” values
- The set of all double-delta values gives us an “acceleration feature”

The process of feature extraction

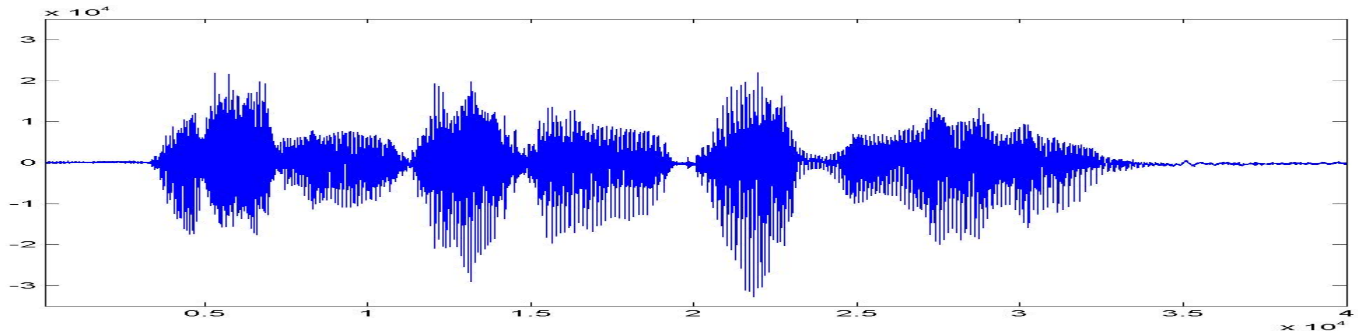


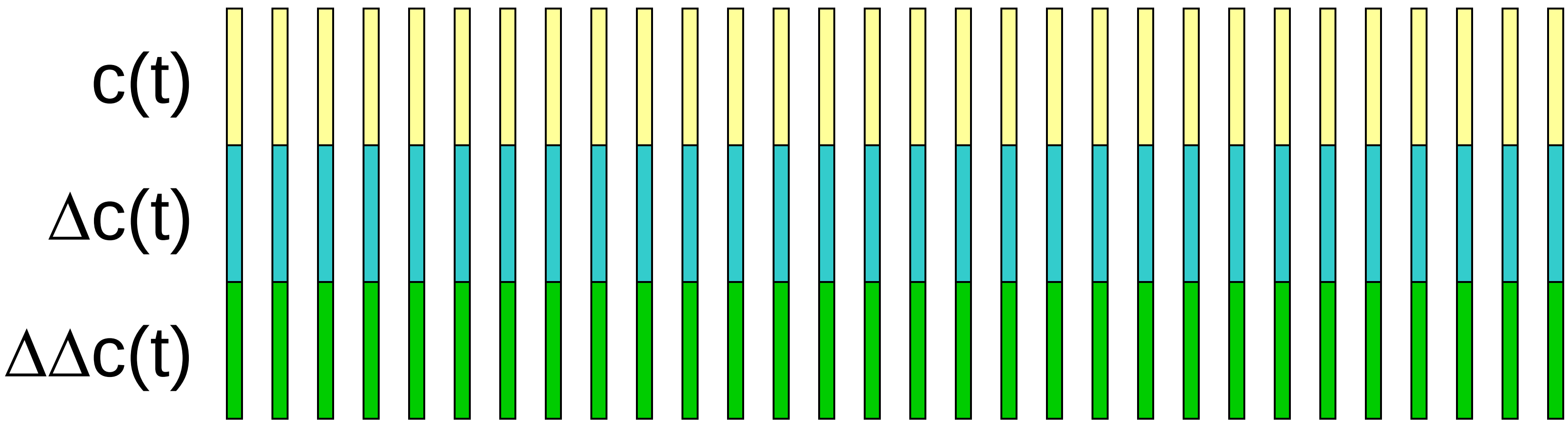


$$\Delta c(t)=c(t+\tau)-c(t-\tau)$$

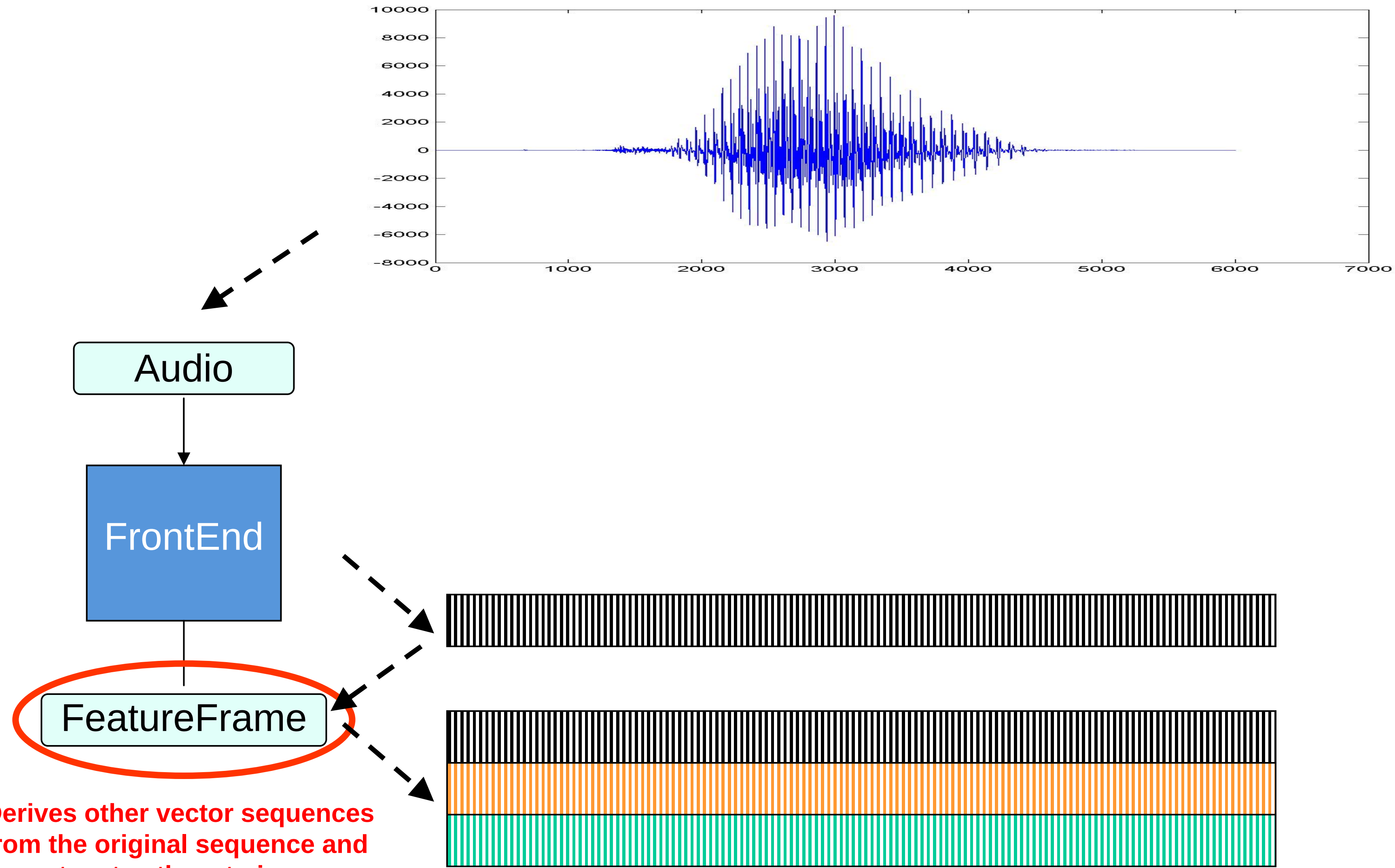
$$\Delta\Delta c(t)=\Delta c(t+\tau)-\Delta c(t-\tau)$$

Feature extraction





Function of the frontend block in a recognizer



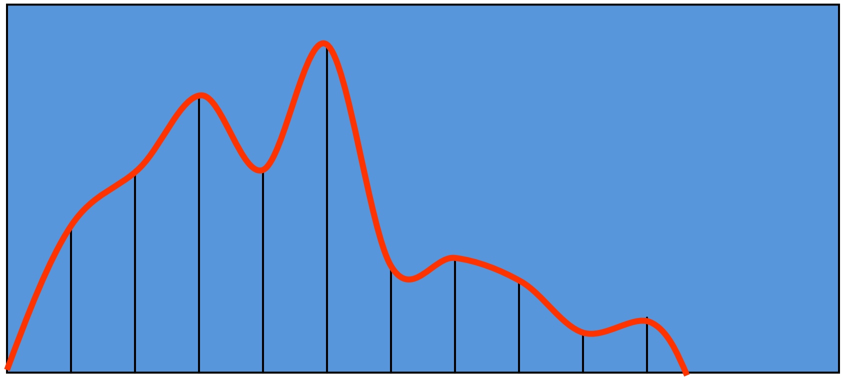
Derives other vector sequences from the original sequence and concatenates them to increase the dimensionality of each vector

This is called feature computation

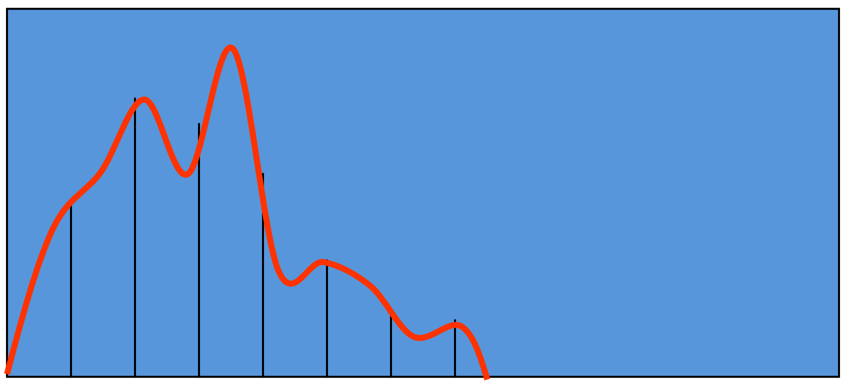
Normalization

- Vocal tracts of different people are different in length
- A longer vocal tract has lower resonant frequencies
- The overall spectral structure changes with the length of the vocal tract

Effect of vocal tract length



- A spectrum for a sound produced by a person with a short vocal tract length



- The same sound produced by someone with a longer vocal tract

Accounting for Vocal Tract Length Variation

- Recognition performance can be improved if the variation in spectrum due to differences in vocal tract length are reduced
  - Reduces variance of each sound class

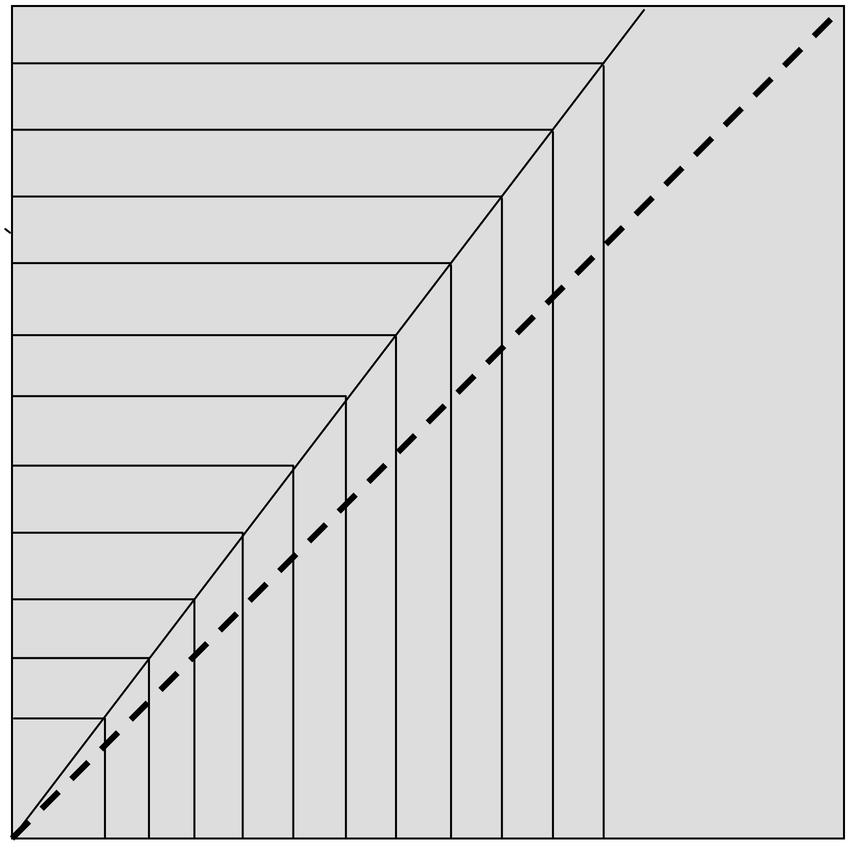
- Way to reduce spectral variation:
  - Linearly "warp" the spectrum of every speaker to a canonical speaker
  - The canonical speaker may be any speaker in the data
  - The canonical speaker may even be a "virtual" speaker

Warping the frequency axis

$f = \alpha f_{canonical}$

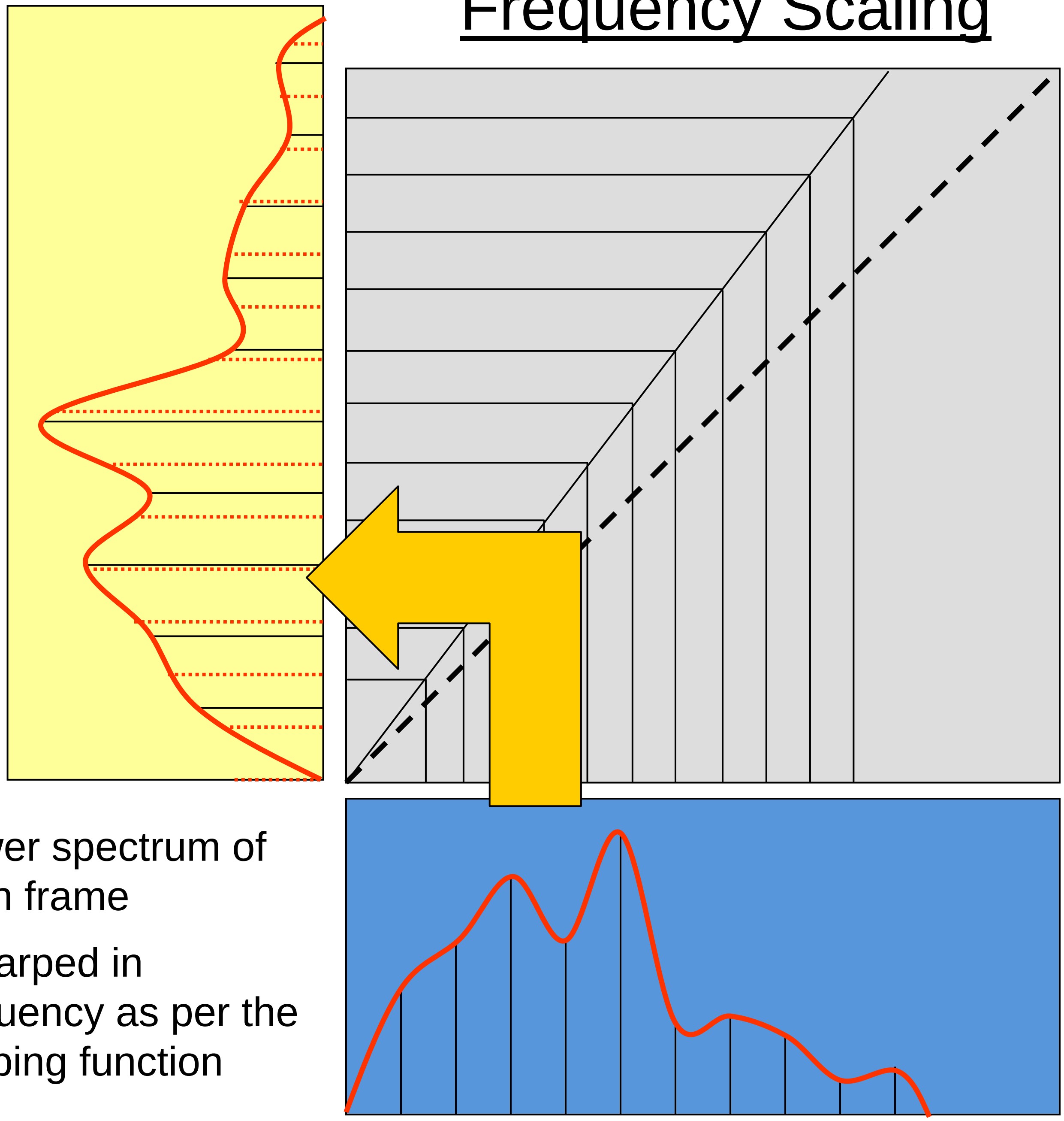
Warping function

Warped frequency axis: frequency difference of *f* in canonical frequency maps to a difference of αf in the warped frequency



Linear frequency axis: Sampled at uniform intervals by an FFT

Frequency Scaling

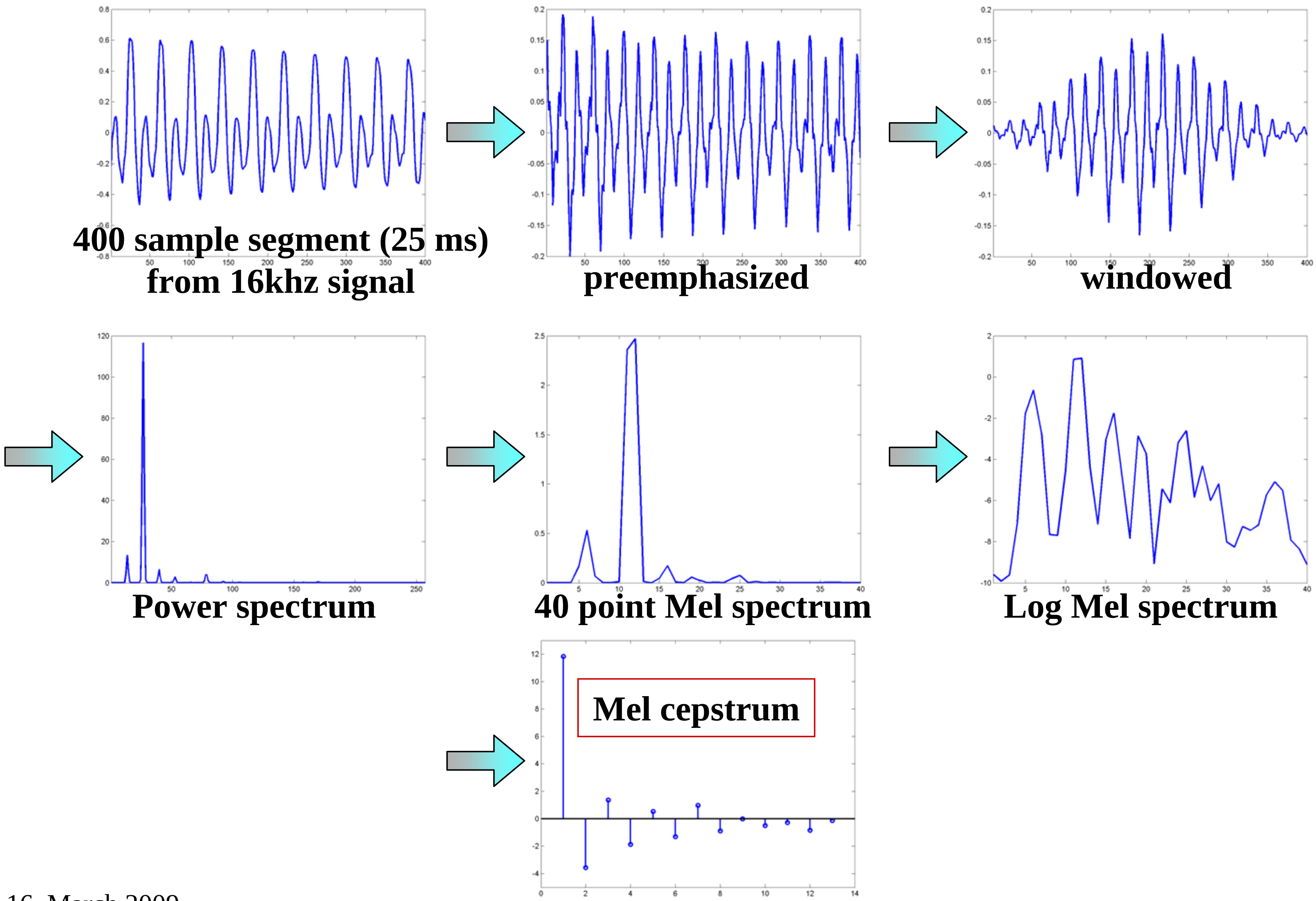


Note: This frequency transform is *separate* from the MEL warping used to compute mel spectra

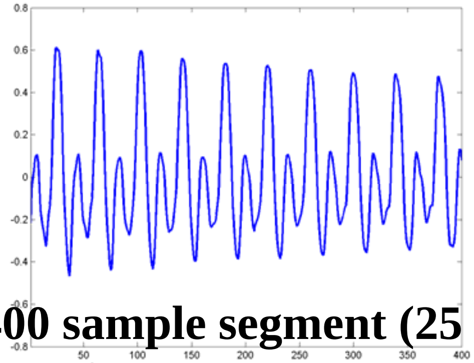
Power spectrum of each frame

is warped in frequency as per the warping function

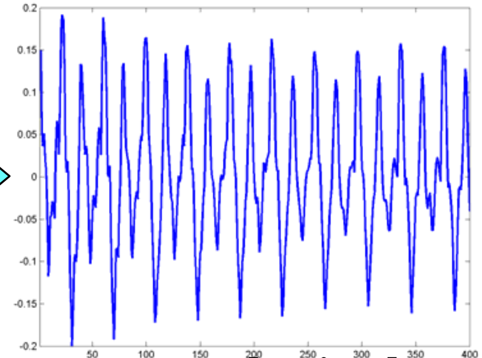
Standard Feature Computation



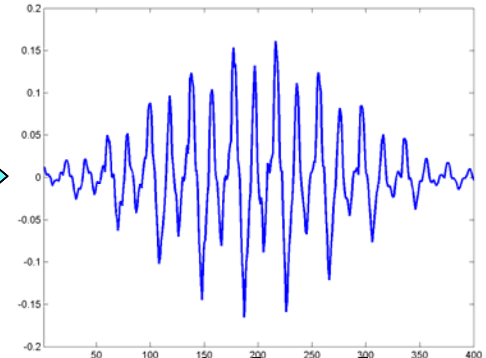
Frequency-warped Feature Comptuation



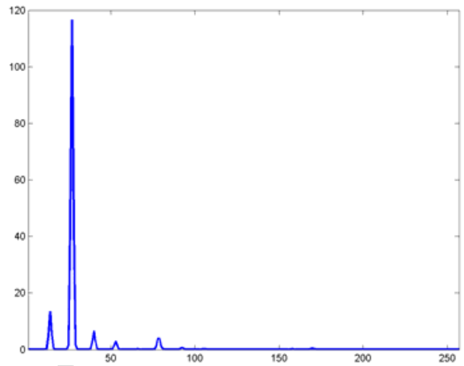
400 sample segment (25 ms) from 16khz signal



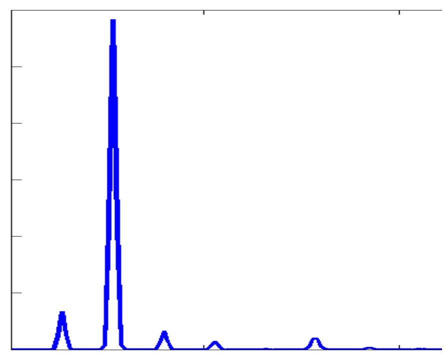
preemphasized



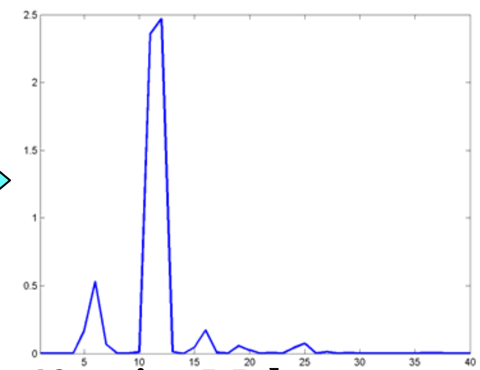
windowed



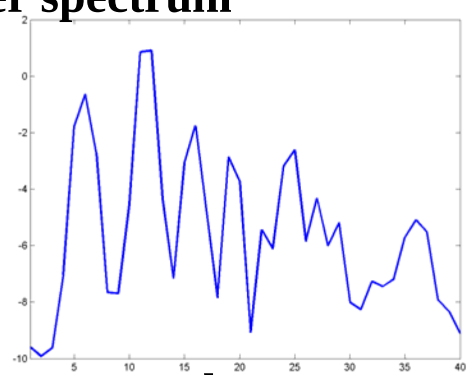
Power spectrum



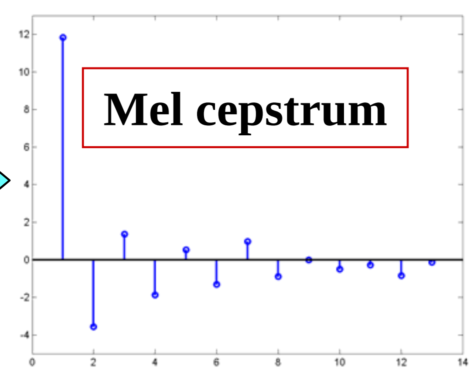
VTLN warping



40 point Mel spectrum



Log Mel spectrum



Mel cepstrum

The process can be shortened

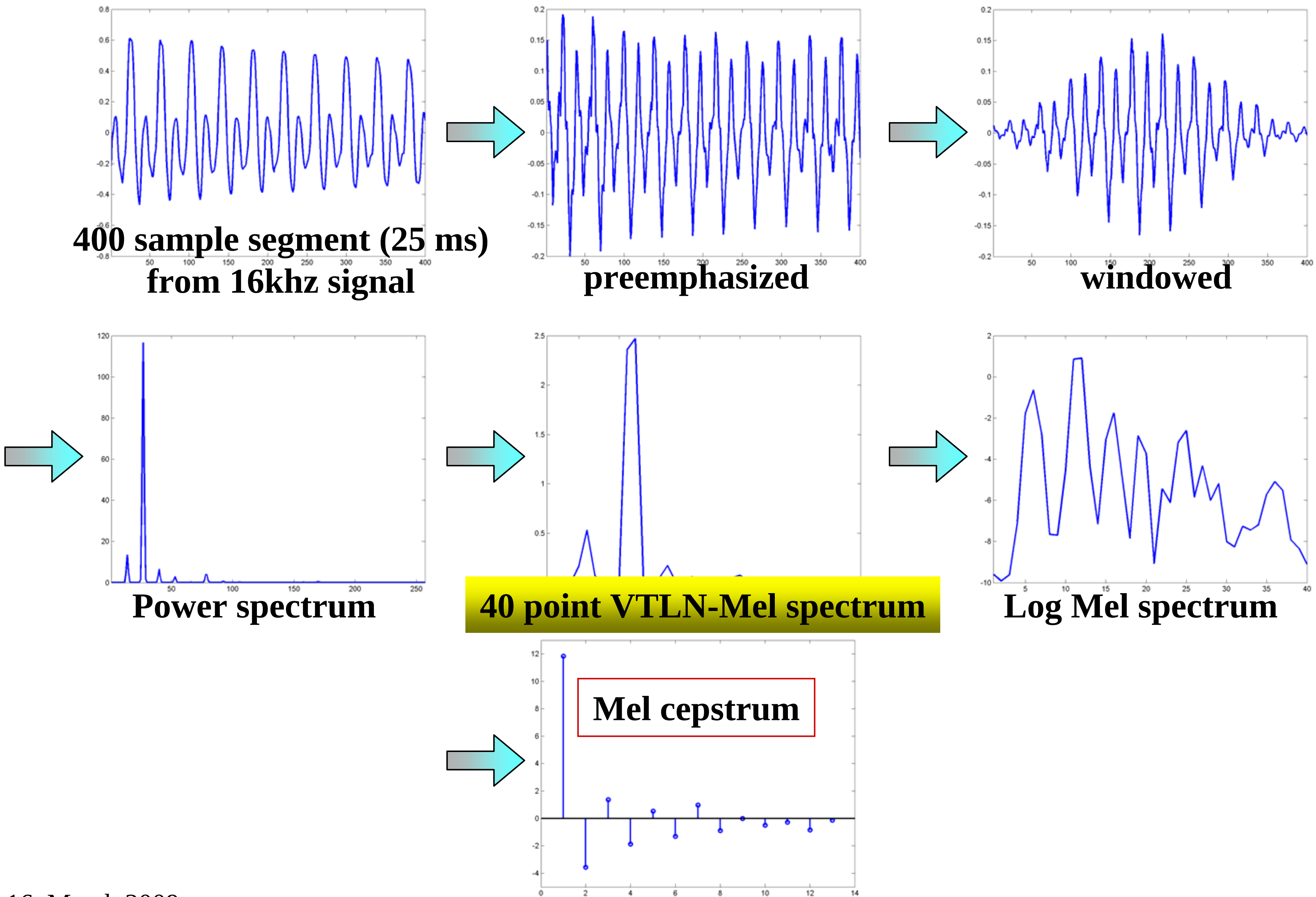
- The frequency warping for vocal-tract length normalization and the Mel-frequency warping can be combined into a single step
  - The MEL frequency warping function changes from:

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

  - To:

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

Modified Feature Computation



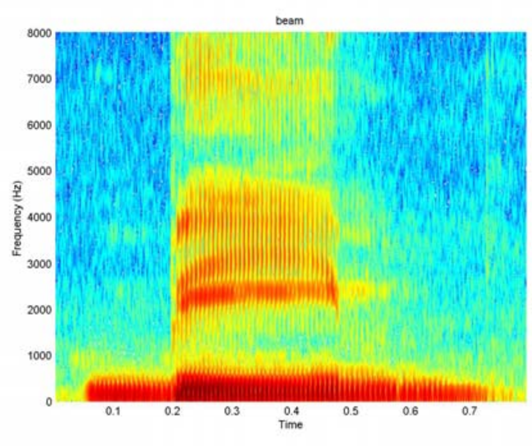
Computing the linear warping

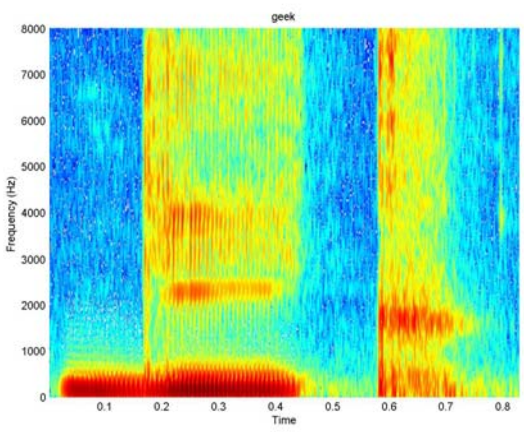
- Based on the spectral characteristics of the signal
  - Linearly scale the frequencies till spectral peaks on the canonical and current speakers match

- Based on statistical comparisons
  - Identify slope of frequency scaling function such that the distribution of features computed from the frequency-scaled data is closest to that of the canonical speaker

Spectral-Characteristic-based Estimation

- Formants are distinctive spectral characteristics
  - Trajectories of peaks in the envelope

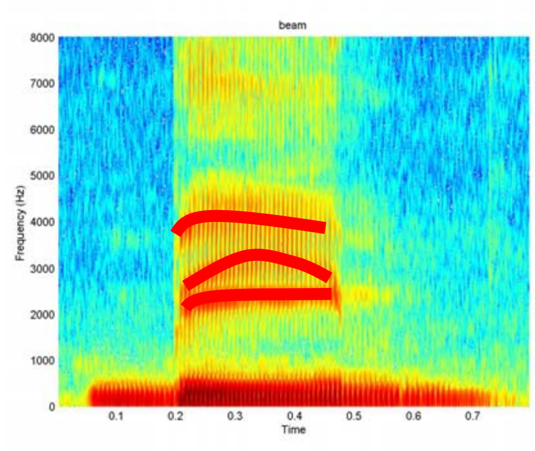


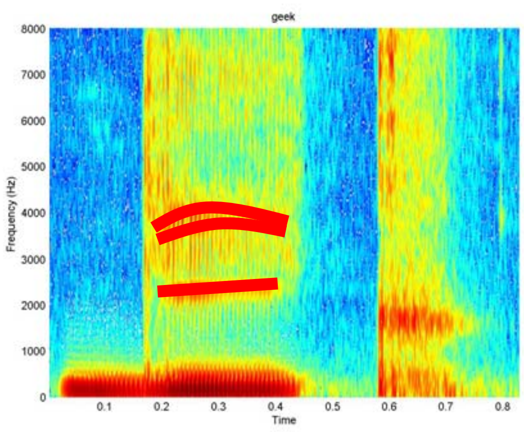


  - These trajectories are similar for different instances of the phoneme
  - But vary in a absolute frequency due to vocal tract length variations

Spectral-Characteristic-based Estimation

- Formants are distinctive spectral characteristics
  - Trajectories of peaks in the envelope





  - These trajectories are similar for different instances of the phoneme
  - But vary in a absolute frequency due to vocal tract length variations

Formants

- Formants are visually identifiable characteristics of speech spectra
- Formants can be estimated for the signal using one of many algorithms
  - Not covering those here
- Formants typically identified as F1, F2 etc. for the first formant, second formant, etc.
  - F0 typically refers to the fundamental frequency – pitch
- The characteristics of phonemes are largely encoded in formant positions

Length Normalization

- To warp a speaker's frequency axis to the canonical speaker, it is sufficient to match formant frequencies for the two
  - i.e. warp the frequency so that F1(speaker) = F1(canonical), F2(speaker) = F2(canonical) etc. on average
- i.e. compute α such that αF1(speaker) = F1(canonical) (and so on) on average

Spectrum-based Vocal Tract Length Normalization

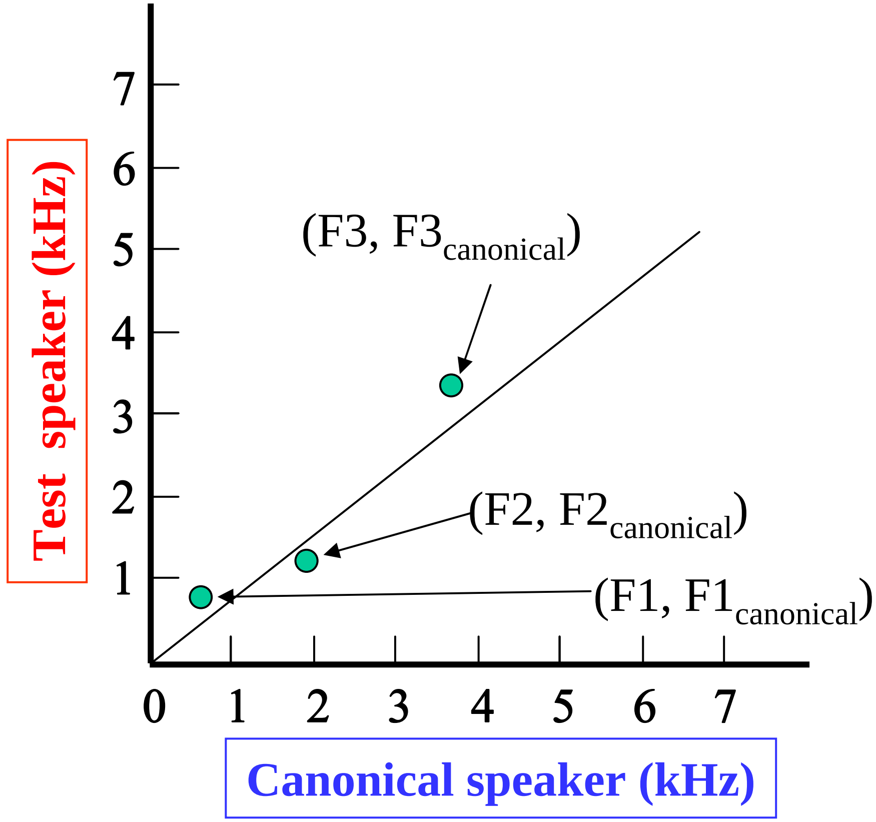
- Compute average F1, F2, F3 for the speaker's speech
  - Run a formant tracker on the speech
    - Returns formants F1, F2, F3.. for each analysis frame
  - Average F1 values for all frames for average F1
    - Similarly compute average F2 and F3.
  - Three formants are sufficient

- Minimize the error:

$$(\alpha F1 - F1_{canonical})^2 + (\alpha F2 - F2_{canonical})^2 + (\alpha F3 - F3_{canonical})^2$$

  - The variables in the above equation are all average formant values
  - This computes a regression between the average formant values for the canonical speaker and those for the test speaker

Spectrum-Based Warping Function



- A is the slope of the regression between (F1, $F1_{canonical}$), (F2, $F2_{canonical}$) and (F3, $F3_{canonical}$)

But WHO is this canonical speaker?

- Simply an average speaker
  - Compute average F1 for all utterances of all speakers
  - Compute average F2 for all utterances of all speakers
  - Compute average F3 for all utterances of all speakers

Overall procedure

- Training:
  - Compute average formant values for all speakers
  - Compute speaker specific frequency warps for each speaker
  - Frequency warp all spectra for the speaker
- Testing:
  - Compute average formant values for the test utterance (or speaker)
  - Compute utterance (or speaker) specific frequency warps
  - Frequency warp all spectra prior to additional processing

Spectra-based VTLN: What sounds to use

- Not useful to use *all* speech
  - No formants in silence regions
  - No formants in fricated sounds (S/SH/H/V/F..)
- Only compute formants from *voiced* sounds
  - Vowels
  - Easy to detect – voicing detection is relatively simple
- Where possible, better to use *a specific* vowel
  - E.g "IY" (very distinctive formant structure)
  - Typically possible where "enrollment" with short utterances is allowed

Distribution-based Estimation

- Compute the distribution of features from the canonical speaker
  - "Features" are Mel-frequency cepstra
  - The distribution is usually modelled as a Gaussian mixture

- For each speaker, identify the warping function such that features computed using it have the highest likelihood on the distribution for the canonical speaker
  - For each of a number of warping functions:
    - Compute features
    - Compute the likelihood of the features on the canonical distribution
    - Select the warping function for which this is highest

Overall Procedure

- The canonical speaker is the *average* speaker
- Overall procedure: Training:
  - Compute the global distribution of all feature vectors for all speakers
  - For each speaker find the warping function that maximizes their likelihood on the global distribution
    - Apply that warping function to the speaker
  - ***Iterate*** (recompute the global distribution etc.)

- The final iteration step is needed since the frequency-warped data for all speakers will have less inherent variability
  - And thereby represent a more consistent canonical speaker

On test data

- For each utterance (or speaker)
  - Find the warping function that maximizes the likelihood for that utterance (or speaker)
  - Apply that warping function

Other Processing: Dealing with Noise

- The incoming speech signal is often corrupted by noise
- Noise may be reduced through spectral subtraction
- Theory:
  - Noise is uncorrelated to speech
  - The power spectrum of noise adds to that of speech, to result in the power spectrum of noisy speech
  - If the power spectrum of noise were known, it could simply be subtracted out from the power spectrum of noisy speech
    - To obtain clean speech

Quick Review

- Discrete Fourier transform coefficients are generally complex
  - $e^{j\theta}$ has a real part $\cos\theta$ and an imaginary part $\sin\theta$

$$e^{j\theta} = \cos\theta + j\sin\theta$$

  - As a result, every X[k] has the form

$$X[k] = X_{real}[k] + jX_{imaginary}[k]$$

- A magnitude spectrum represents only the magnitude of the Fourier coefficients

$$X_{magnitude}[k] = \mathrm{sqrt}(X_{real}[k]^2 + X_{imag}[k]^2)$$

- A power spectrum is the square of the magnitude spectrum

$$X_{power}[k] = X_{real}[k]^2 + X_{imag}[k]^2$$

- For speech recognition, we usually use the magnitude or power spectra

Denoising the speech signal

- The goal is to eliminate the noise from the speech signal itself *before* it is processed any further for recognition
- The basic procedure is as follows:
  - Estimate the noise corrupting the speech signal in any analysis frame (somehow)
  - Remove the noise from the signal

- Problem: The estimation of noise is never perfect
  - It is impossible to estimate the exact noise signal that corrupted the speech signal
  - At best, some average characteristic (e.g. the magnitude or power spectrum) may be estimated
    - Also with significant error

- The noise cancellation technique must be able to eliminate the noise in spite of these drawbacks
  - The noise cancellation may only be expected to improve the noise "on average"

Describing Additive Noise

- Let s(t) represent the speech signal in any frame of speech, and n(t) represent the noise corrupting the signal in that frame
- The observed noisy signal is the sum of the speech and the noise

$$x(t) = s(t) + n(t)$$

- Assumption: The magnitude spectra of the noise and the speech *add* to produce the magnitude spectrum of noisy speech

- In the frequency domain

$$X_{mag}(k) = S_{mag}(k) + N_{mag}(k)$$

Estimating the noise spectrum

- The first step is to obtain an estimate for the noise spectrum
- Problems:
  - The precise noise spectrum varies from analysis frame to analysis frame
  - It is impossible to determine the precise spectrum of the noise that has corrupted a noisy signal
- Assumption: The first few frames of a recording contain only noise
  - The user begins speaking *after* hitting the “record” button
- Assumption: The signal in non-speech regions is all noise
- Assumption: The noise changes slowly
- Observation: The onset of speech is indicated by a sudden increase in signal power

A running estimate of noise

- Initialize (from the first T non-speech frames):

  $N(T,k) = (1/T)\ \Sigma_t\ X(t,k)$

  – k represents frequency band; "t" is the frame index
- Subsequent estimates are obtained as

$$|N(t,f)| = \begin{cases} (1-\lambda)\,|N(t-1,k)| + \lambda\,|X(t,k)| & \text{if } |X(t,k)| < \beta\,|N(t-1,k)| \\ |N(t-1,k)| & \text{otherwise} \end{cases}$$

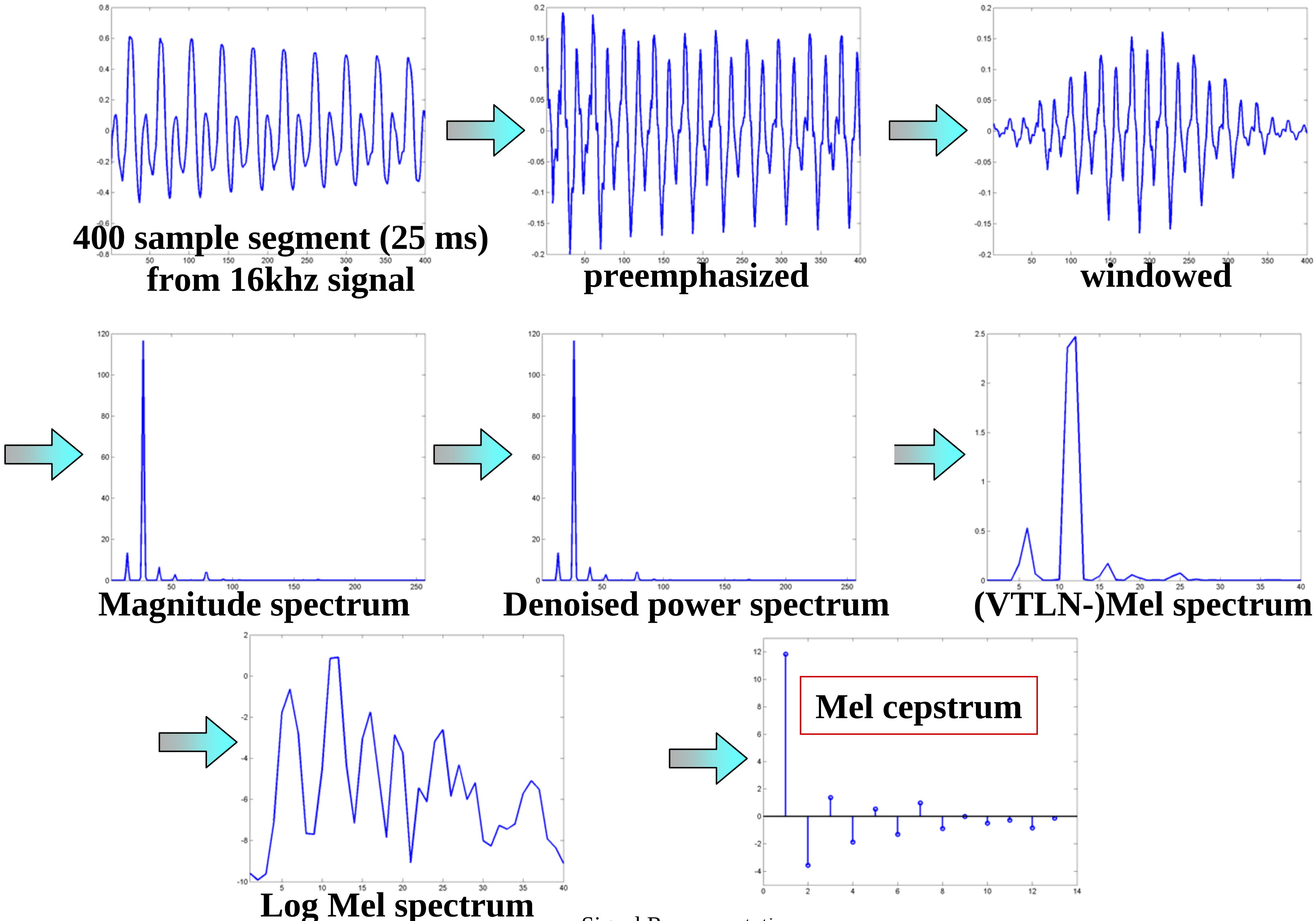
- λ is an update factor, and depends on the rate at which noise changes
  – Typically set to about 0.1
- β is a threshold value: if the signal jumps by this amount, speech has begun

Subtracting the Noise

$$|Y(t,k)| = \begin{cases} |X(t-1,k)| - \alpha|N(t,k)| & \text{if } |X(t,k)| - \alpha|N(t,k)| > \gamma|X(t,k)| \\ \gamma|N(t-1,k)| & \text{otherwise} \end{cases}$$

- α is an oversubtraction factor
  - Typically set to about 5
  - This accounts for the fact that the noise may be underestimated
- γ is a spectral floor
  - This prevents the estimated spectrum from becoming zero or negative
    - The estimated noise spectrum can sometimes be greater than the observed noisy spectrum. Direct subtraction without a floor can result in negative values for the estimated power (or magnitude) spectrum of speech!
  - Typically set to 0.1 or less
- Y(t,k) is used instead of X(t,k) for feature comptuation

Modified Feature Computation



Caveats with Noise Subtraction

- Noise estimates are never perfect
- Subtracting estimated noise will always
  - Leave a little of the real noise behind
  - *Remove some speech*
- The *perceptual quality* of the signal improves, but the *intelligibility* decreases
- Difficult to strike a tradeoff between removing corrupting noise and retaining intelligibility
  - Usually best to simply train on noisy speech with no processing
  - Such data may not be available often, however

Questions

- ?

Wav2feat is a sphinx feature computation tool:

- ./SphinxTrain-1.0/bin.x86_64-unknown-linux-gnu/wave2feat
- [Switch] [Default] [Description]
- -help no Shows the usage of the tool
- -example no Shows example of how to use the tool
- -i Single audio input file
- -o Single cepstral output file
- -c Control file for batch processing
- -nskip If a control file was specified, the number of utterances to skip at the head of the file
- -runlen If a control file was specified, the number of utterances to process (see -nskip too)
- -di Input directory, input file names are relative to this, if defined
- -ei Input extension to be applied to all input files
- -do Output directory, output files are relative to this
- -eo Output extension to be applied to all output files
- -nist no Defines input format as NIST sphere
- -raw no Defines input format as raw binary data
- -mswav no Defines input format as Microsoft Wav (RIFF)
- -input_endian little Endianness of input data, big or little, ignored if NIST or MS Wav
- -nchans 1 Number of channels of data (interlaced samples assumed)
- -whichchan 1 Channel to process
- -logspec no Write out logspectral files instead of cepstra
- -feat sphinx SPHINX format - big endian
- -mach_endian little Endianness of machine, big or little
- -alpha 0.97 Preemphasis parameter
- -srate 16000.0 Sampling rate
- -frate 100 Frame rate
- -wlen 0.025625 Hamming window length
- -nfft 512 Size of FFT
- -nfilt 40 Number of filter banks
- -lowerf 133.33334 Lower edge of filters
- -upperf 6855.4976 Upper edge of filters
- -ncep 13 Number of cep coefficients
- -doublebw no Use double bandwidth filters (same center freq)
- -warp_type inverse_linear Warping function type (or shape)
- -warp_params Parameters defining the warping function
- -blocksize 200000 Block size, used to limit the number of samples used at a time when reading very large audio files
- -dither yes Add 1/2-bit noise to avoid zero energy frames
- -seed -1 Seed for random number generator; if less than zero, pick our own
- -verbose no Show input filenames

Wav2feat is a sphinx feature computation tool:

- ./SphinxTrain-1.0/bin.x86_64-unknown-linux-gnu/wave2feat

[Switch]	[Default]	[Description]
-help	no	Shows the usage of the tool
-example	no	Shows example of how to use the tool

Wav2feat is a sphinx feature computation tool:

```
./SphinxTrain-1.0/bin.x86_64-unknown-linux-gnu/wave2feat
-i                              Single audio input file
-o                              Single cepstral output file
-nist         no                Defines input format as NIST sphere
-raw          no                Defines input format as raw binary data
-mswav        no                Defines input format as Microsoft Wav
-logspec      no                Write out logspectral files instead
                                 of cepstra
-alpha        0.97              Preemphasis parameter
-srate        16000.0           Sampling rate
-frate        100               Frame rate
-wlen         0.025625          Hamming window length
-nfft         512               Size of FFT
-nfilt        40                Number of filter banks
-lowerf       133.33334         Lower edge of filters
-upperf       6855.4976         Upper edge of filters
-ncep         13                Number of cep coefficients
-warp_type    inverse_linear Warping function type (or shape)
-warp_params                     Parameters defining the warping function
-dither       yes               Add 1/2-bit noise to avoid zero energy
   frames
```

Format of output File

- Four-byte integer header
  - Specifies no. of floating point values to follow
  - Can be used to both determine byte order and validity of file

- Sequence of four-byte floating-point values

Inspecting Output

- sphinxbase-0.4.1/src/sphinx_cepview
- [NAME] [DEFLT] [DESCR]
- -b 0 The beginning frame 0-based.
- -d 10 Number of displayed coefficients.
- -describe 0 Whether description will be shown.
- -e 2147483647 The ending frame.
- -f Input feature file.
- -i 13 Number of coefficients in the feature vector.
- -logfn Log file (default stdout/stderr)

Wav2feat Tutorial

- Install SphinxTrain1.0
  - From cmusphinx.sourceforge.net
- Record multiple instances of digits
  - Zero, One, Two etc.
  - Compute log spectra and cepstra using wav2feat
    - No. of features = Num. filters for logspectra
    - No. of features = 13 for cepstra
  - Visualize both using cepview
    - Note similarity in different instances of the same word
  - Modify no. of filters to 30 and 25
    - Patterns will remain, but be more blurry
  - Record data with noise
    - Degradation due to noise may be lesser on 25-filter outputs