Design and Implementation of Speech Recognition Systems

Spring 2013

Class 3: Feature Computation
30 Jan 2013

First Step: Feature Extraction

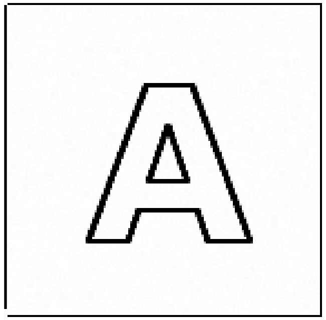
- Speech recognition is a type of pattern recognition problem
- *Q:* Should the pattern matching be performed on the audio sample streams directly? If not, what?
- *A:* Raw sample streams are not well suited for matching
- A visual analogy: recognizing a letter inside a box

  - 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

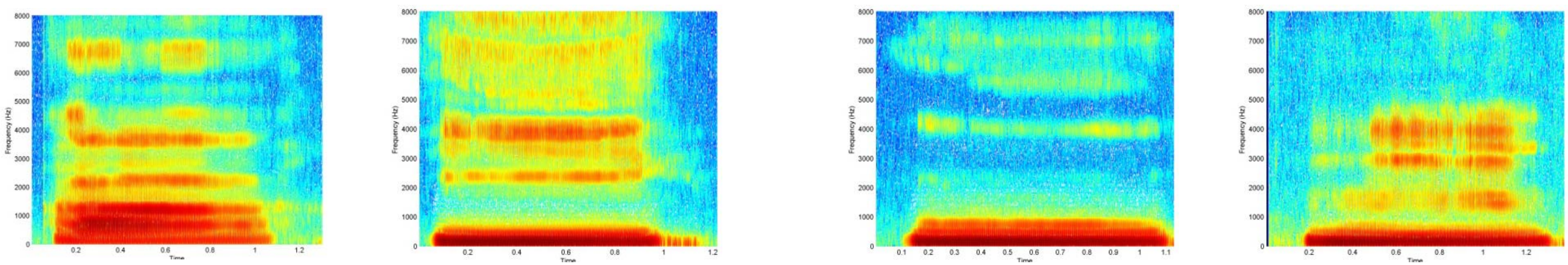


- Our brain does this kind of image analysis automatically and we can instantly identify the input letter as being the same as the template

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"

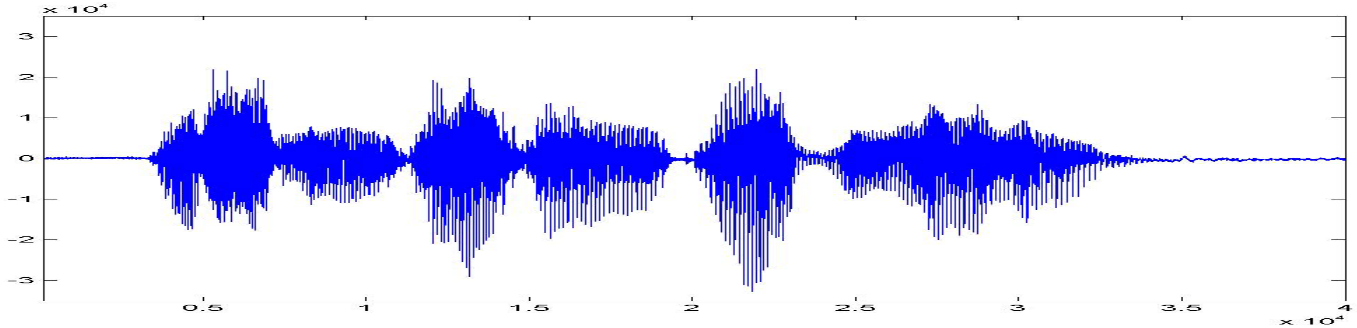
- Features must be computed that capture the *spectral* characteristics of the signal
- Important to capture only the *salient* spectral characteristics of the sounds
  – Without capturing speaker-specific or other incidental structure

- 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 to follow 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:
  $s[n] = A.s_0[n] + B.s_1[n] + C.s_2[n] + \dots$
- 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

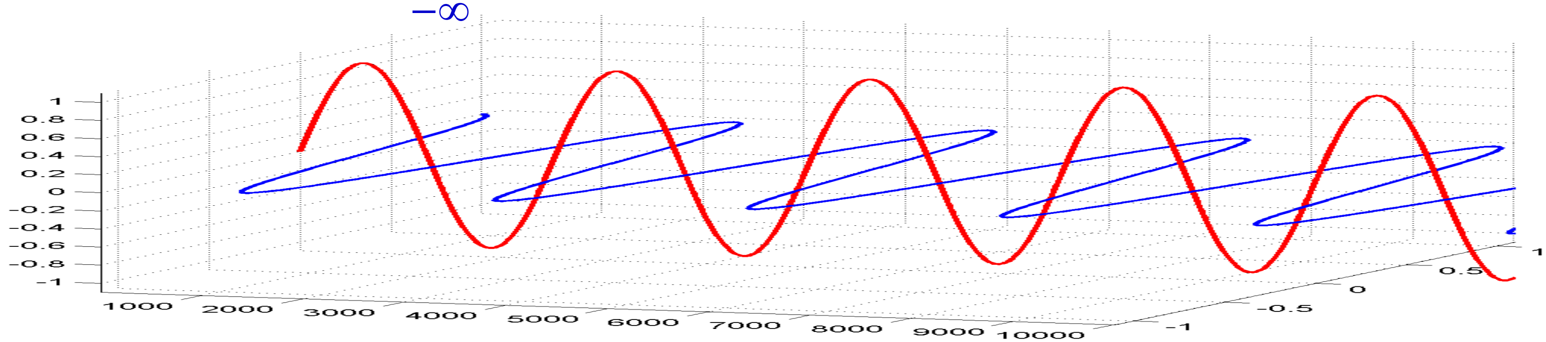


The complex exponential

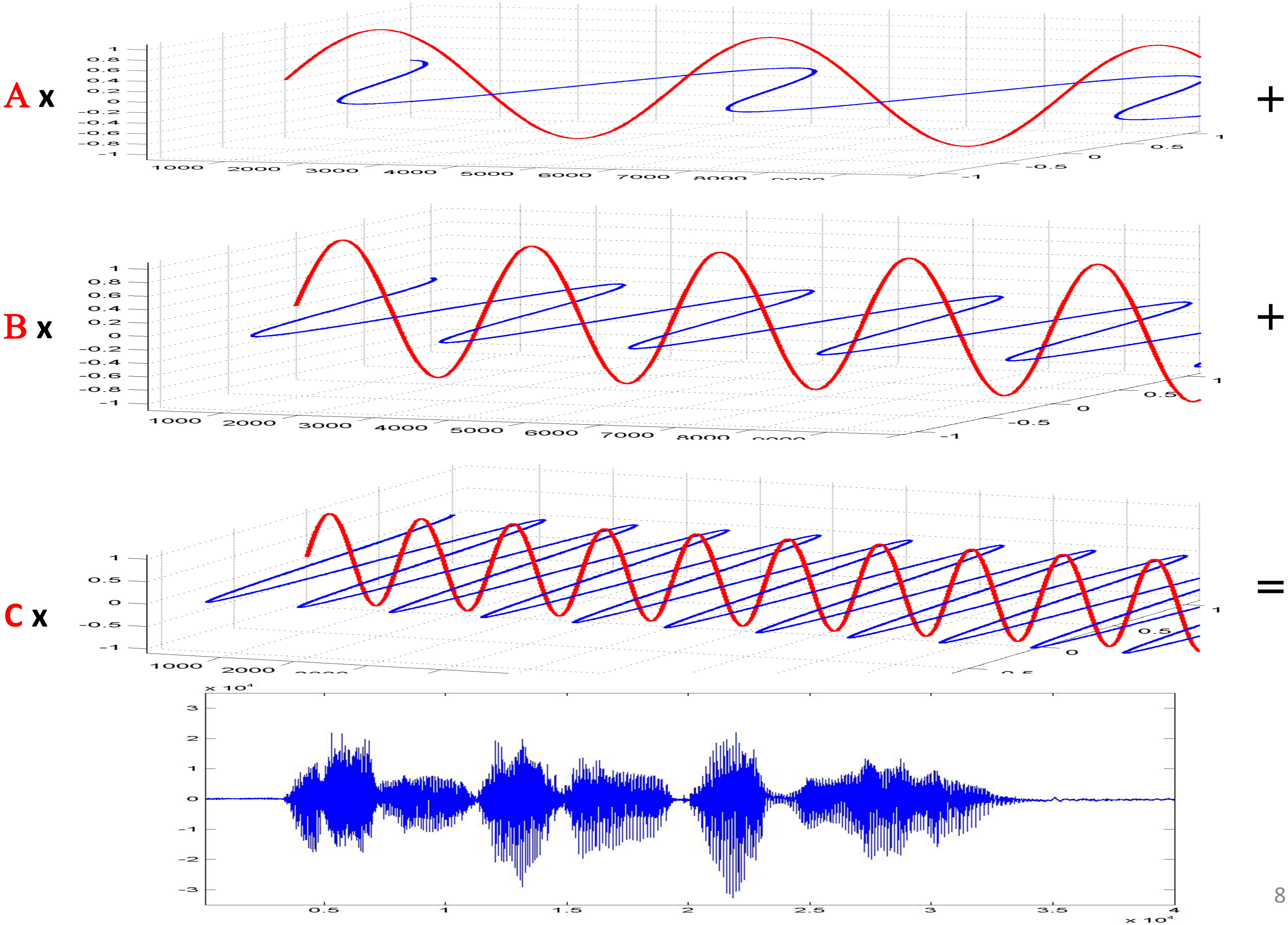
- The complex exponential is a complex sum of two sinusoids
  - $e^{j\theta} = \cos\theta + j\sin\theta$
- The real part is a cosine function
- The imaginary part is a sine function

- 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.

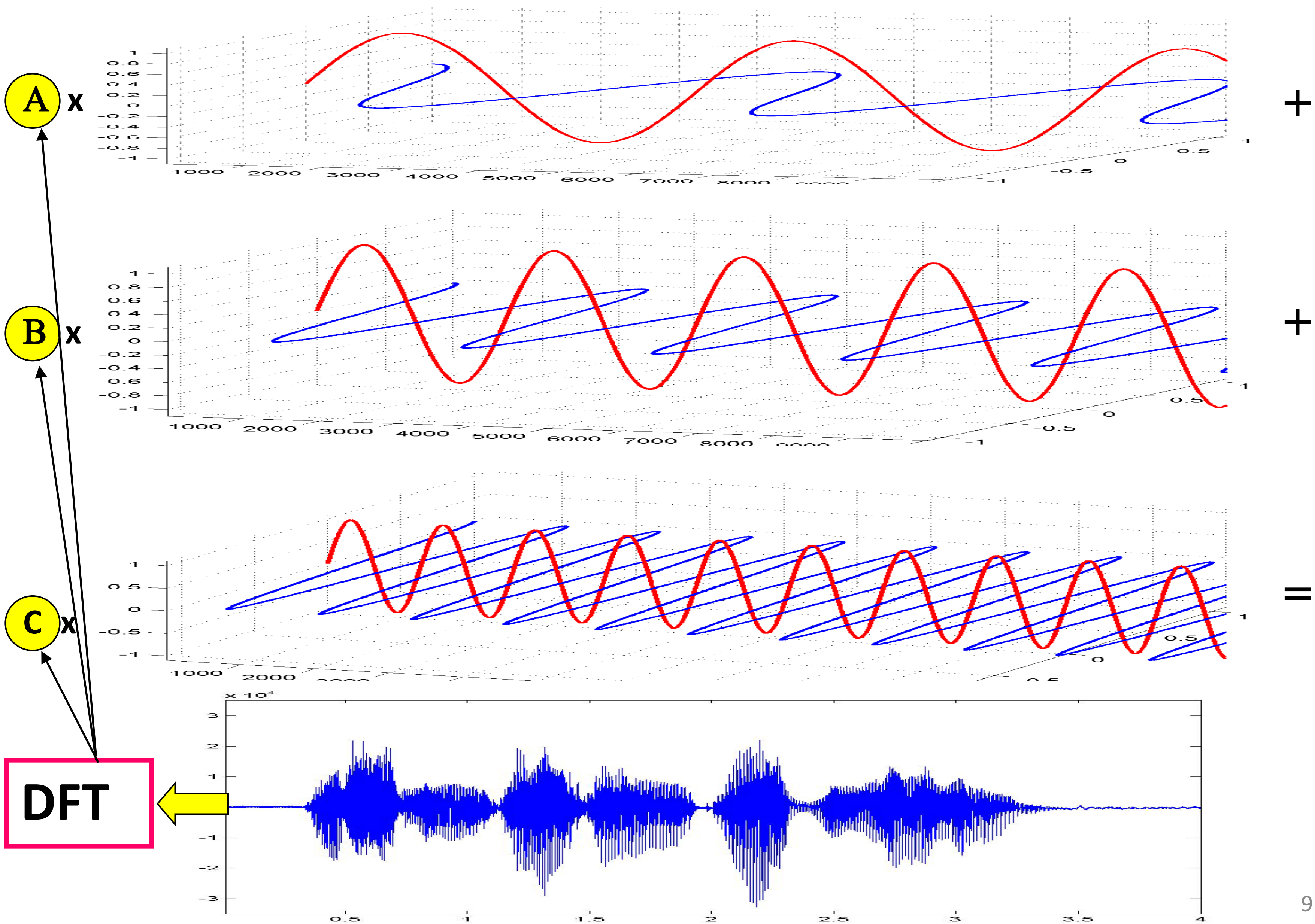
$$\int_{-\infty}^{\infty} e^{j\alpha t} e^{j\beta t}\, dt = 0 \qquad \text{if } \alpha \neq \beta$$



The Discrete Fourier Transform



The Discrete Fourier Transform



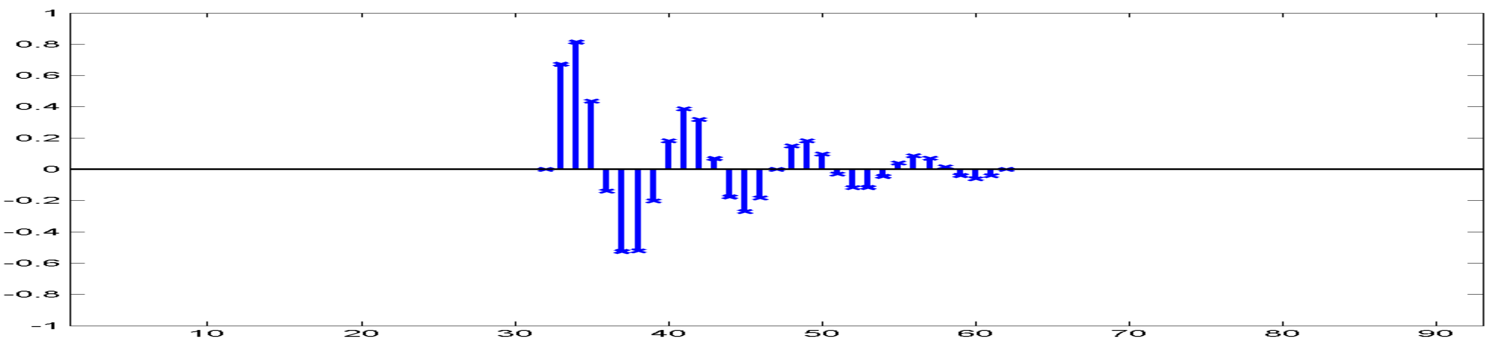
The Discrete Fourier Transform

- 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

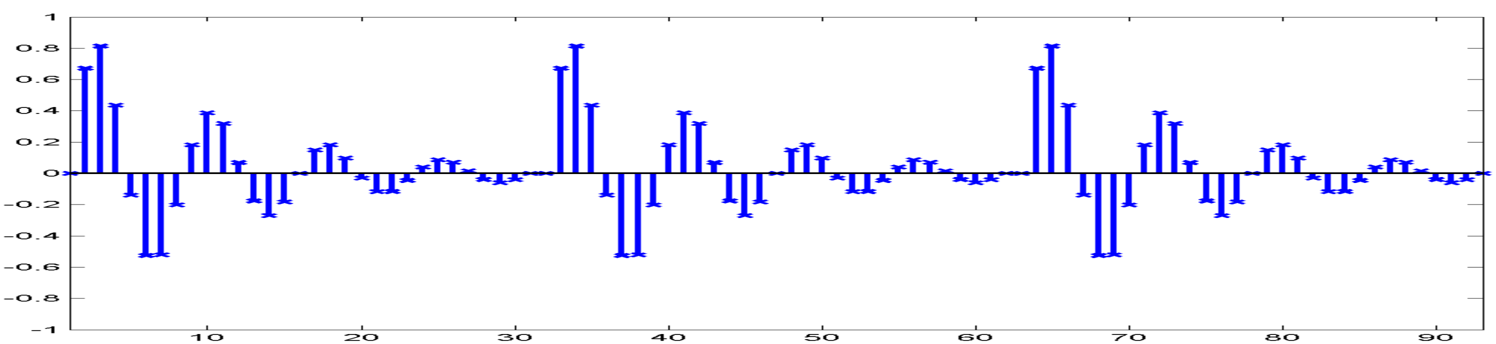
- An aperiodic signal *cannot* be decomposed into a sum of a finite number of complex exponentials
  - Or into a sum of any countable set of periodic signals

- 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 kth 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

- 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
  - The Mth point represents the sampling frequency, which cannot be distinguished from DC
- All DFT points are uniformly spaced on the frequency axis between 0 and the sampling frequency

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

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

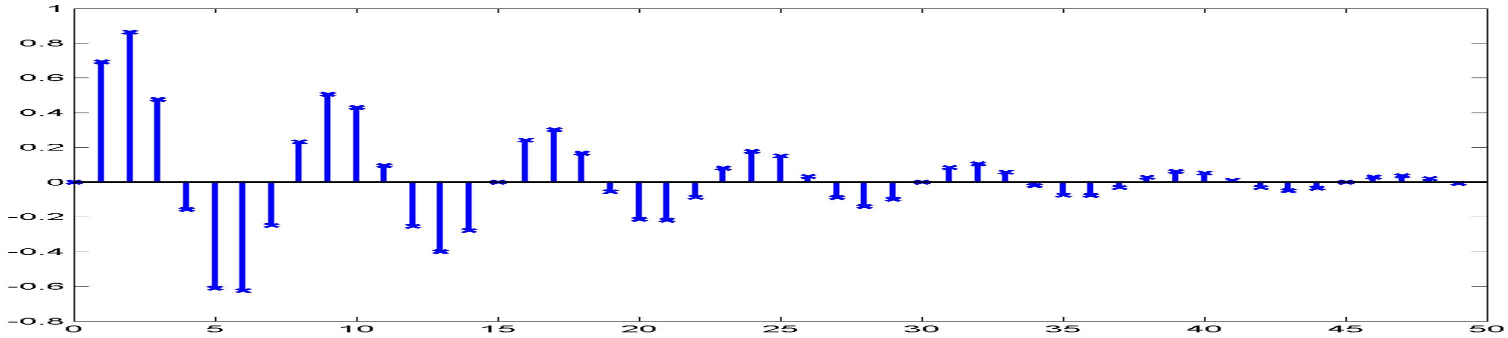
- A power spectrum is the square of the magnitude spectrum

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

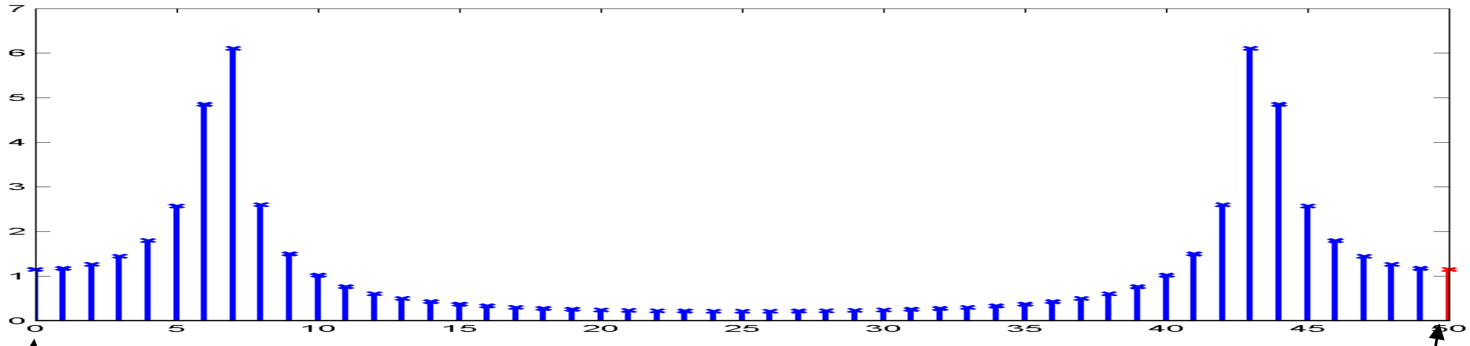
- For speech recognition, we usually use the magnitude or power spectra

The Discrete Fourier Transform

- A 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.



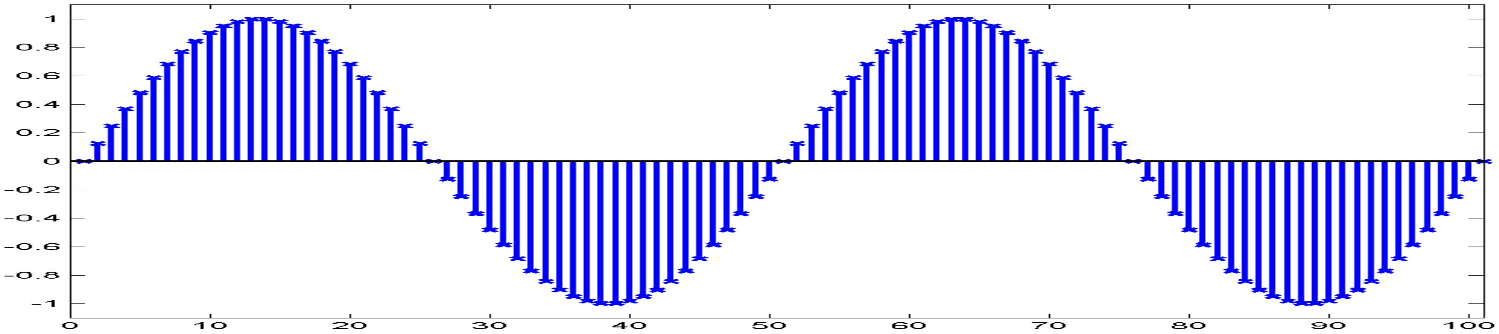
The Discrete Fourier Transform

- 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:

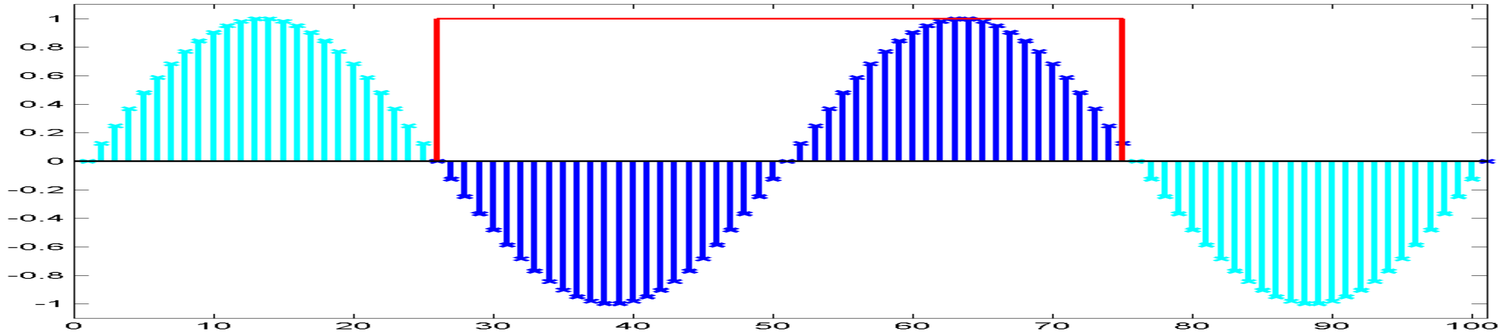
$$x[n] = \frac{1}{M} \sum_{k=0}^{M-1} X[k] e^{\frac{j2\pi kn}{M}}$$

Windowing



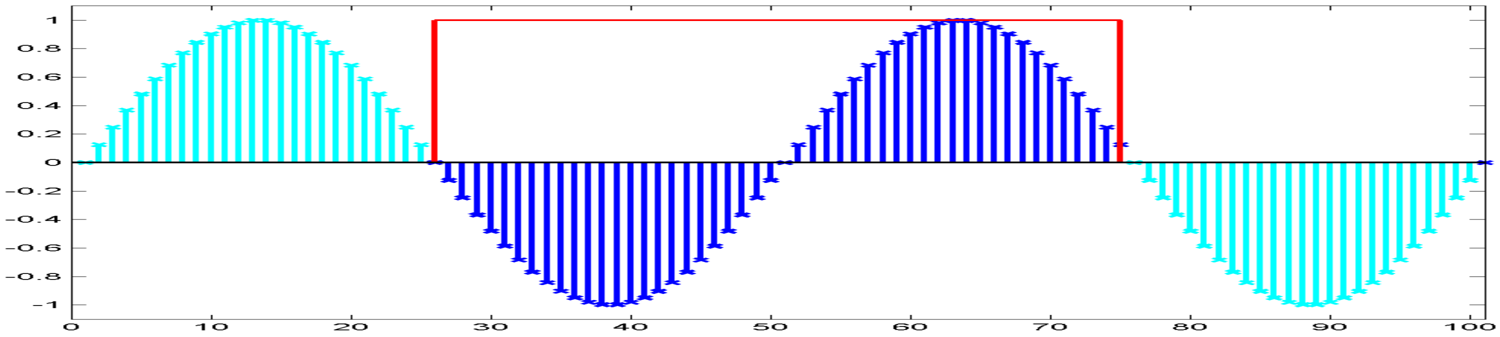
- The DFT of one period of the sinusoid shown in the figure computes the Fourier series of the entire sinusoid from –infinity to +infinity

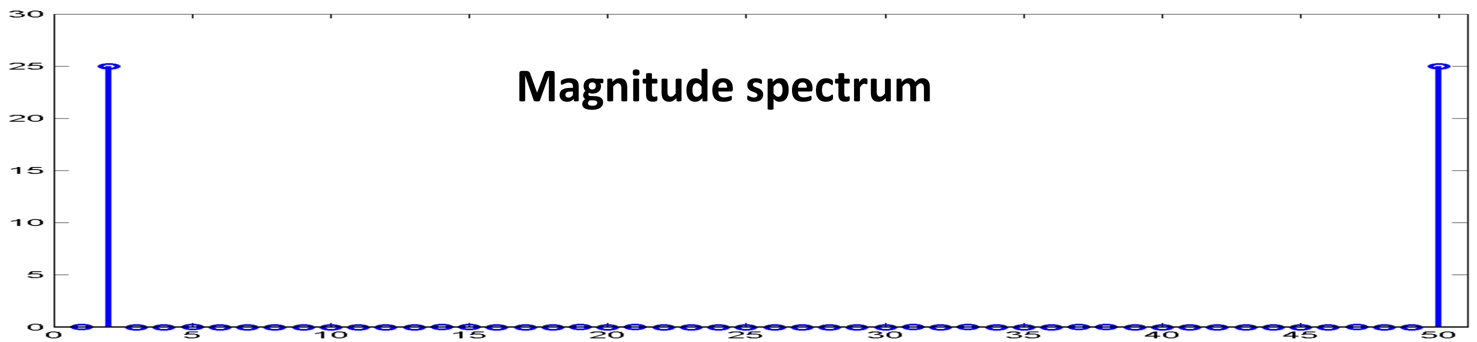
Windowing



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

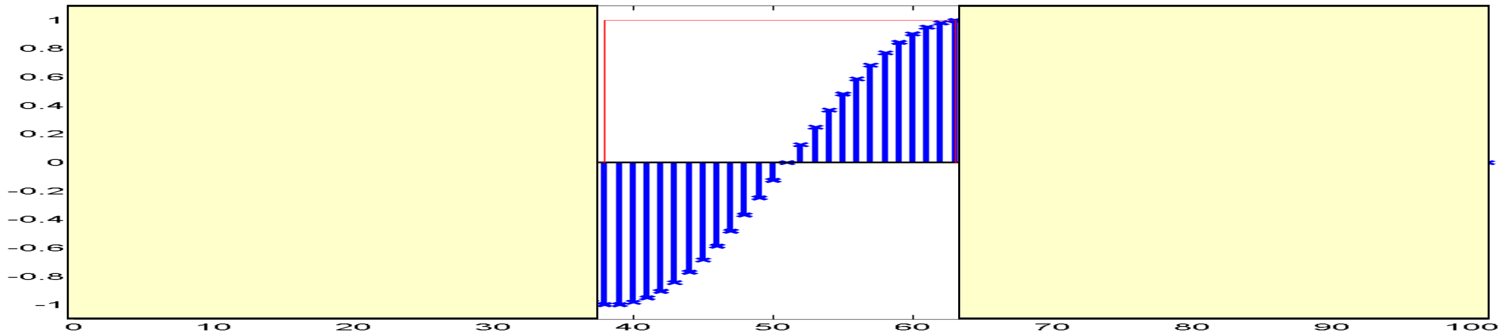
Windowing





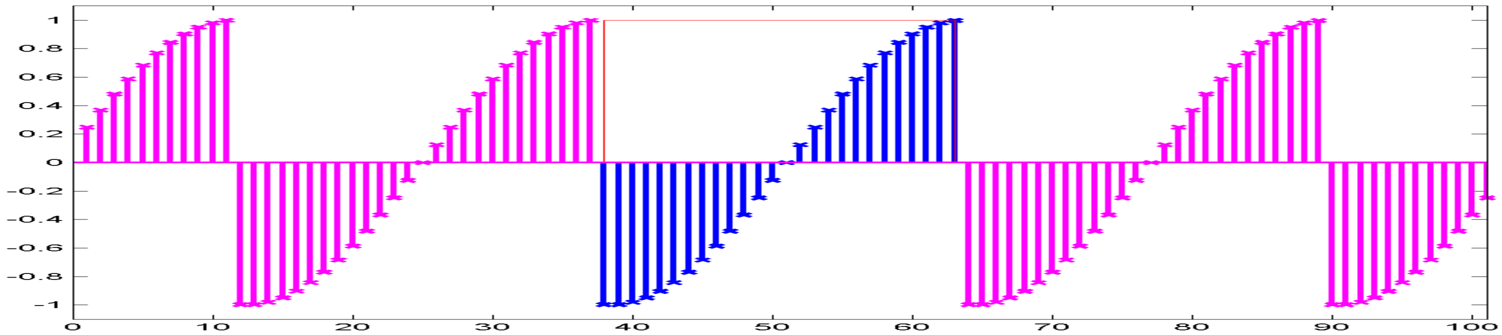
- The DFT of one period of the sinusoid shown in the figure computes the Fourier series of the entire sinusoid from –infinity to +infinity

Windowing



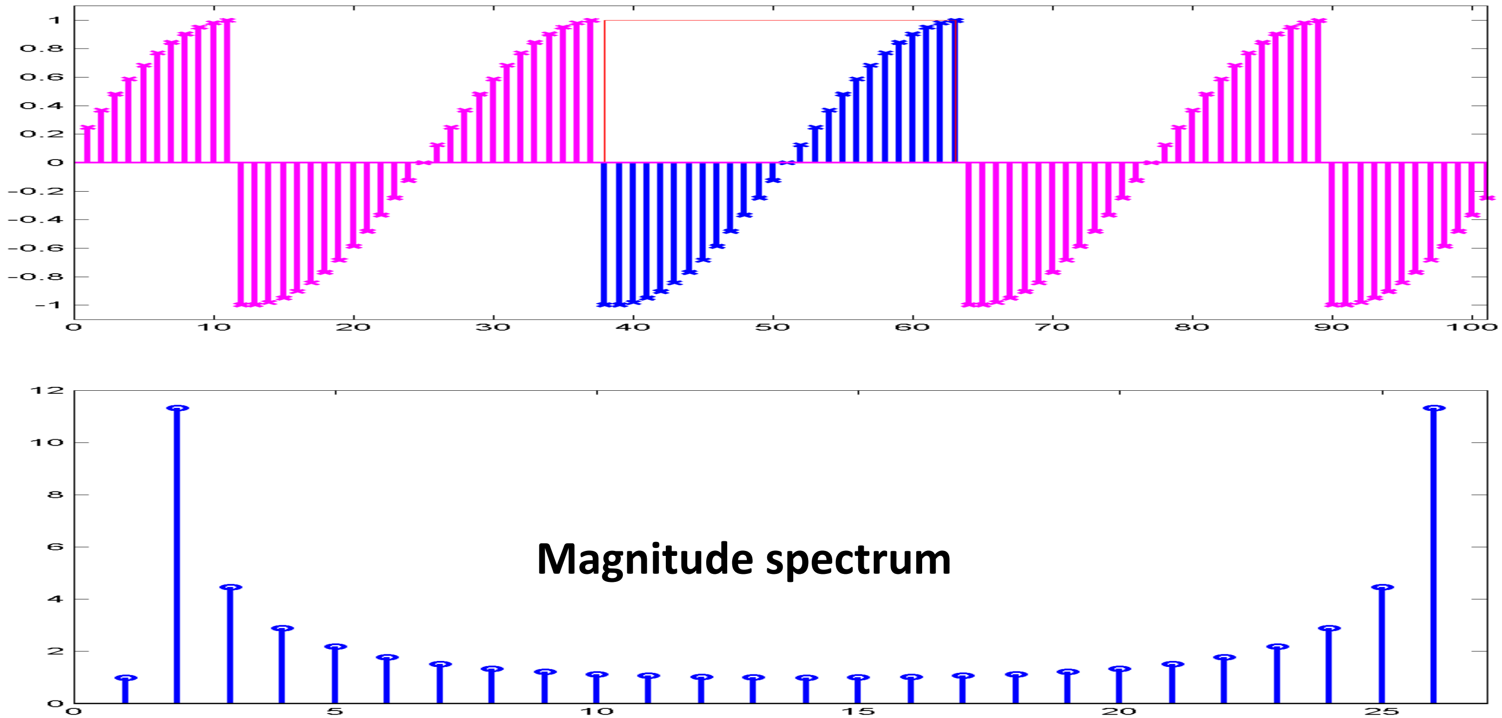
- The DFT of *any* sequence computes the Fourier series for an infinite repetition of that sequence

Windowing



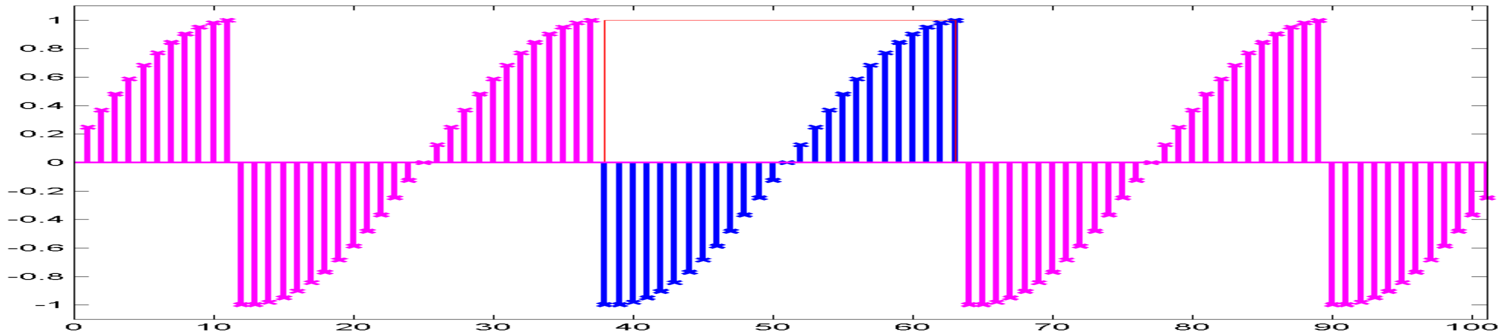
- The DFT of *any* sequence computes the Fourier series for an infinite repetition of that sequence

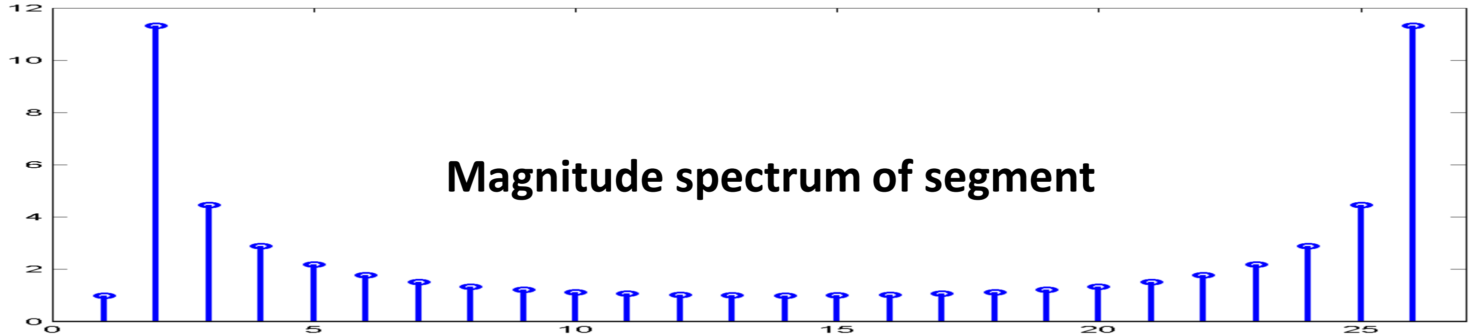
Windowing



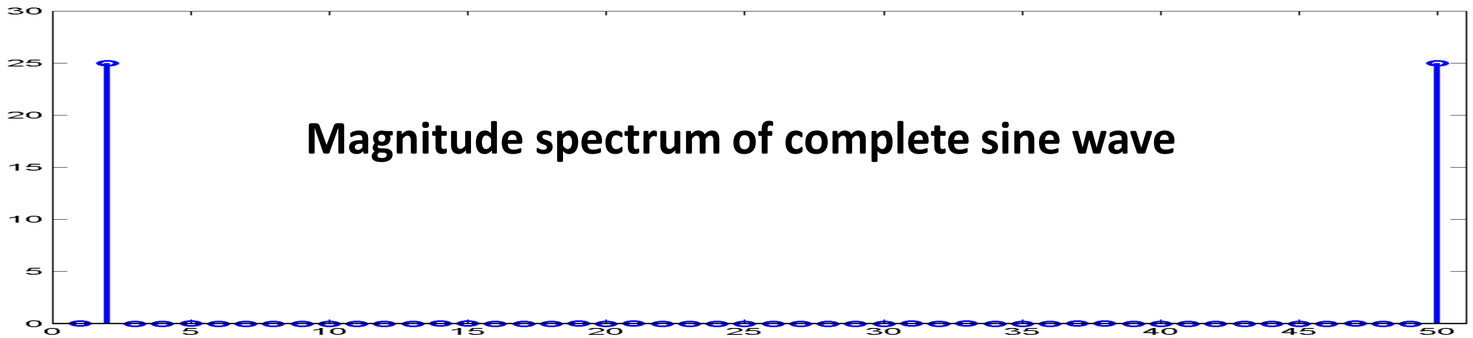
- The DFT of *any* sequence computes the Fourier series for an infinite repetition of that sequence
- The DFT of a partial segment of a sinusoid computes the Fourier series of an infinite repetition of that segment, and not of the entire sinusoid
- This will not give us the DFT of the sinusoid itself!

Windowing



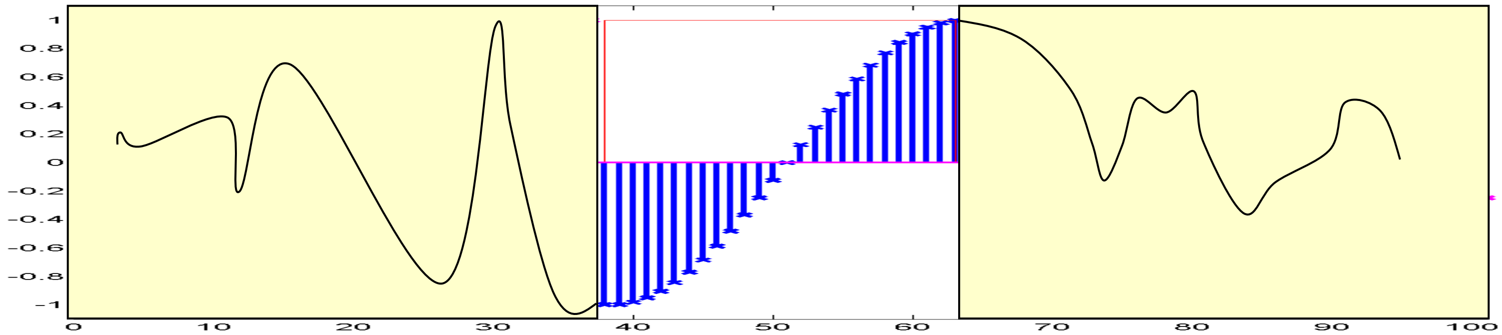


Magnitude spectrum of segment



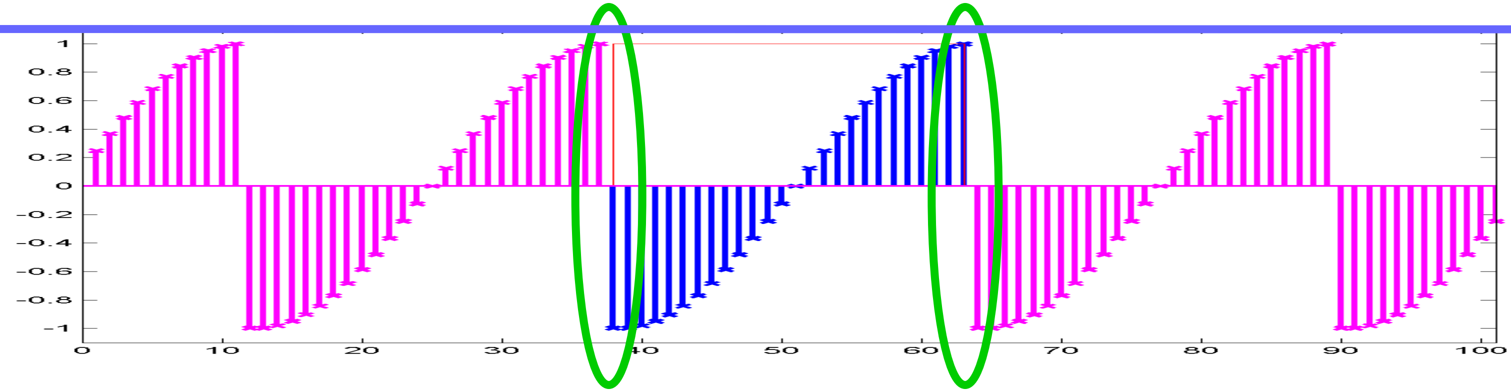
Magnitude spectrum of complete sine wave

Windowing



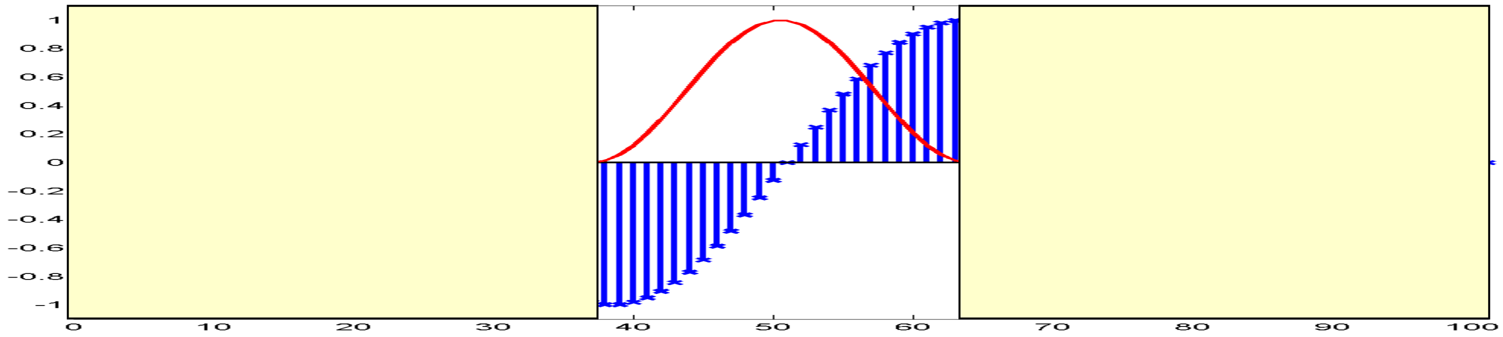
- The difference occurs due to two reasons:
- The transform cannot know what the signal actually looks like outside the observed window
  - We must infer what happens outside the observed window from what happens inside

Windowing



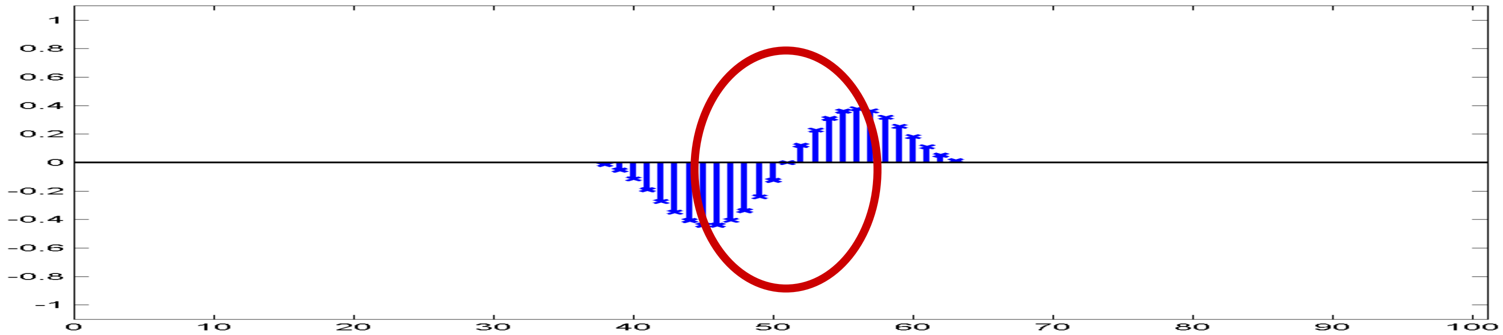
- The difference occurs due to two reasons:
- The transform cannot know what the signal actually looks like outside the observed window
  - We must infer what happens outside the observed window from what happens inside
- The implicit repetition of the observed signal introduces large discontinuities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



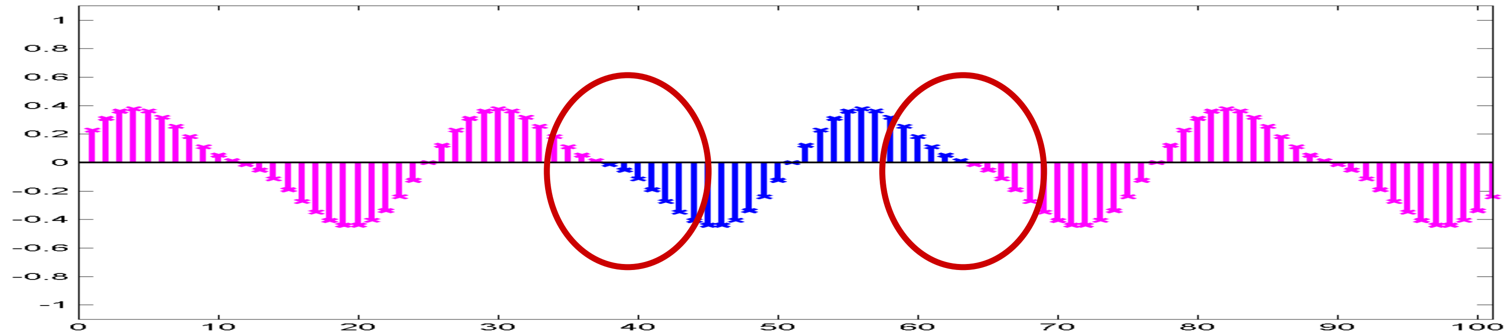
- While we can never know what the signal looks like outside the window, we can try to minimize the discontinuities at the boundaries
- We do this by multiplying the signal with a *window* function
  - We call this procedure windowing
  - We refer to the resulting signal as a "windowed" signal

Windowing



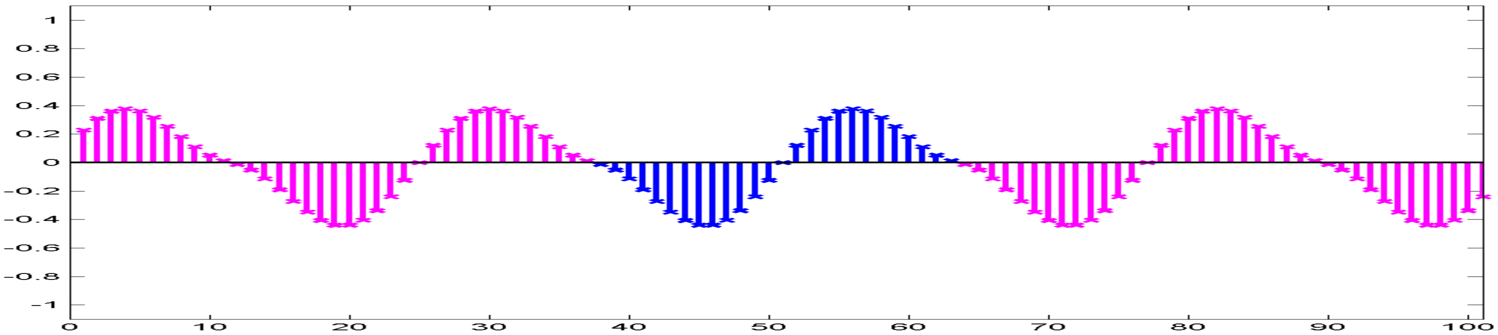
- While we can never know what the signal looks like outside the window, we can try to minimize the discontinuities at the boundaries
- We do this by multiplying the signal with a *window* function
  - We call this procedure windowing
  - We refer to the resulting signal as a "windowed" signal
- Windowing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

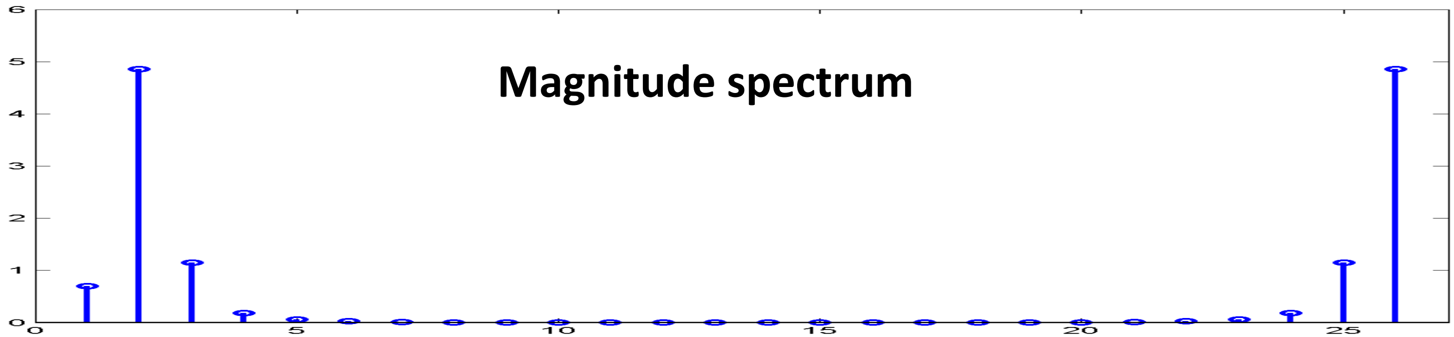
Windowing



- 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
  - Reduce or eliminate the discontinuities in the implicit periodic signal

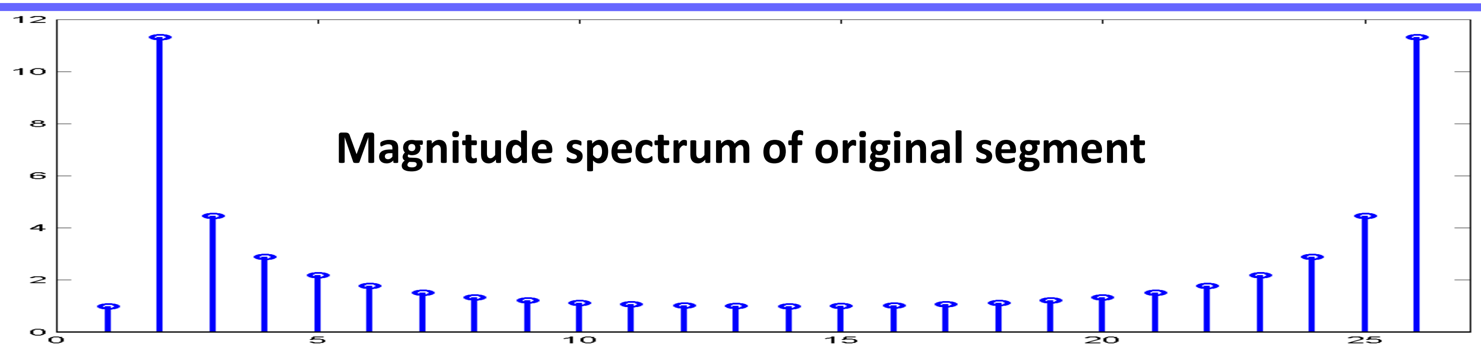
Windowing

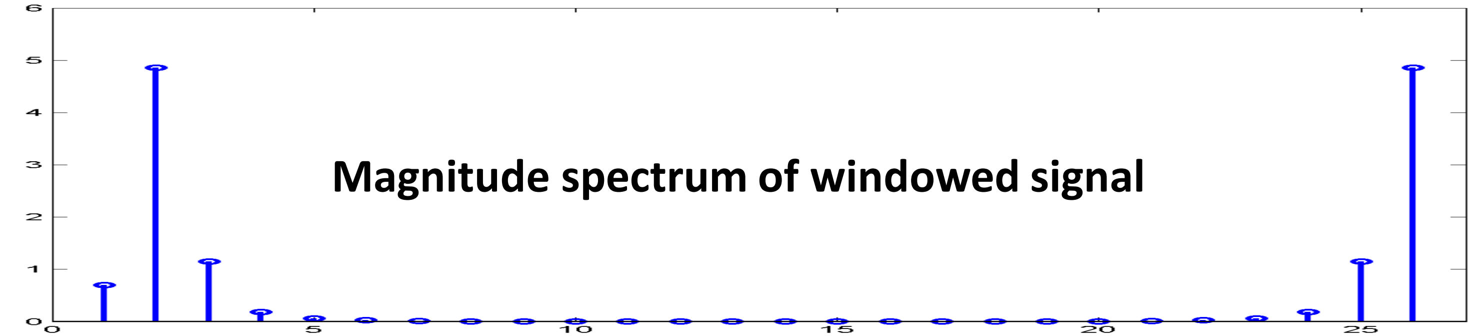


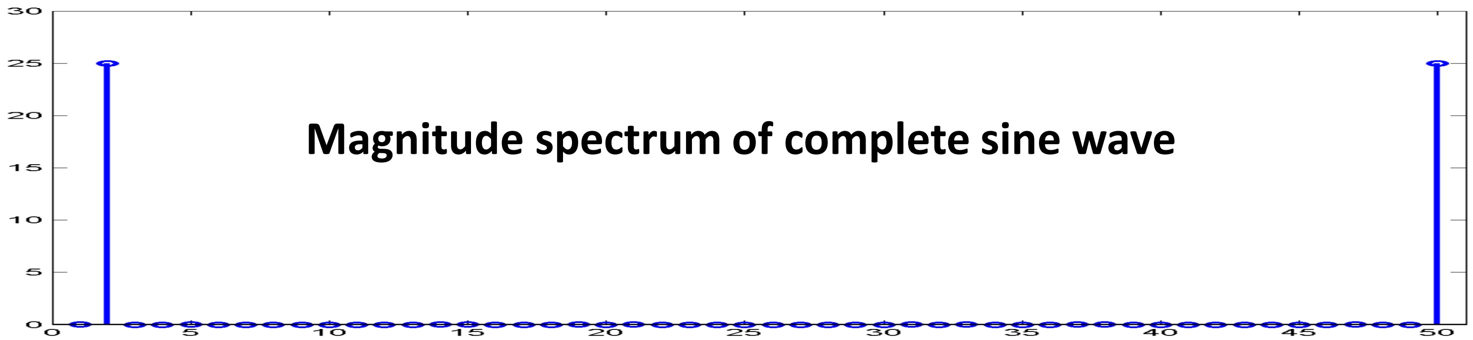


- The DFT of the windowed signal does not have any artifacts introduced by discontinuities in the signal
- Often it is also a more faithful reproduction of the DFT of the complete signal whose segment we have analyzed

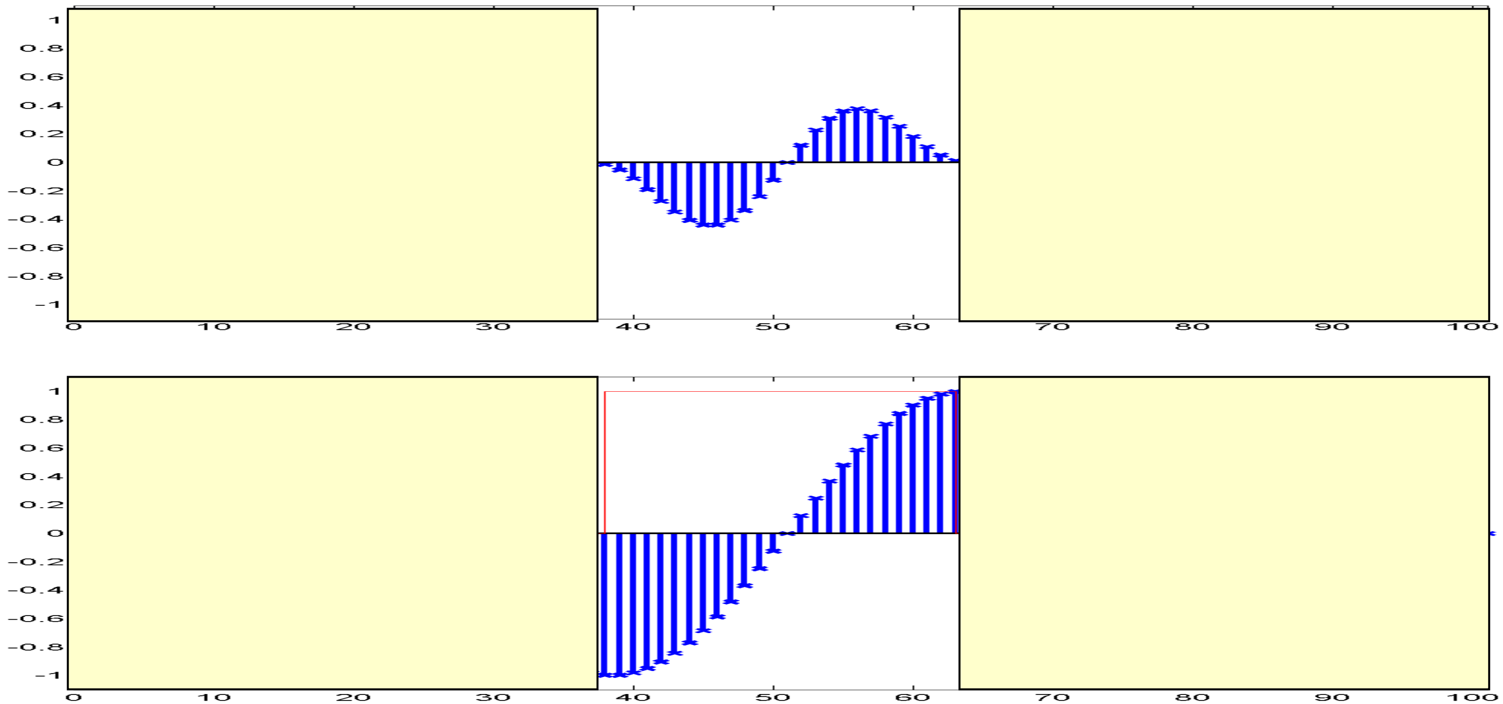
Windowing


Magnitude spectrum of original segment


Magnitude spectrum of windowed signal

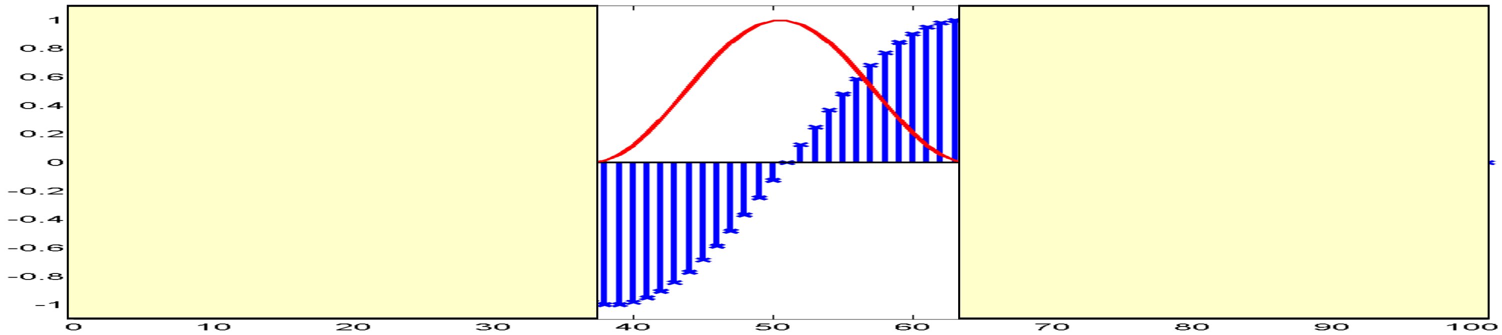

Magnitude spectrum of complete sine wave

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:

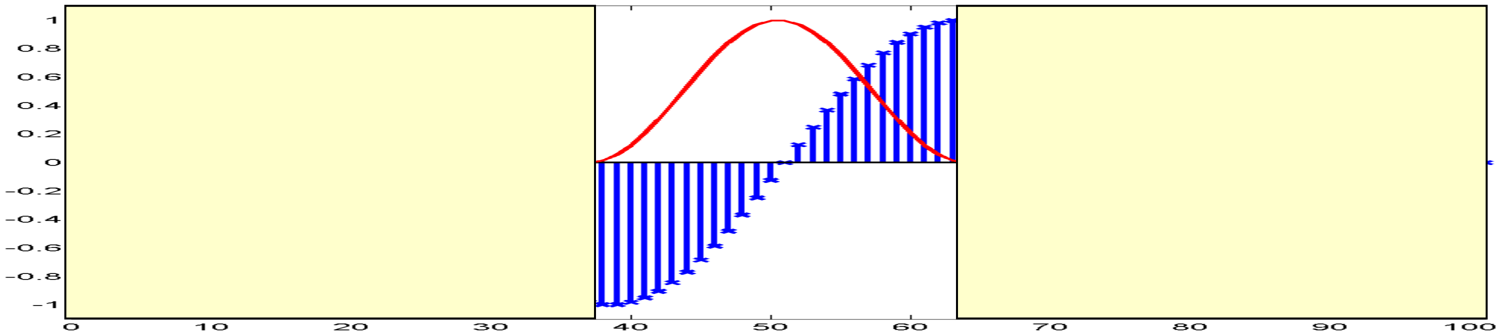
- Window length is M
- Index begins at 0

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

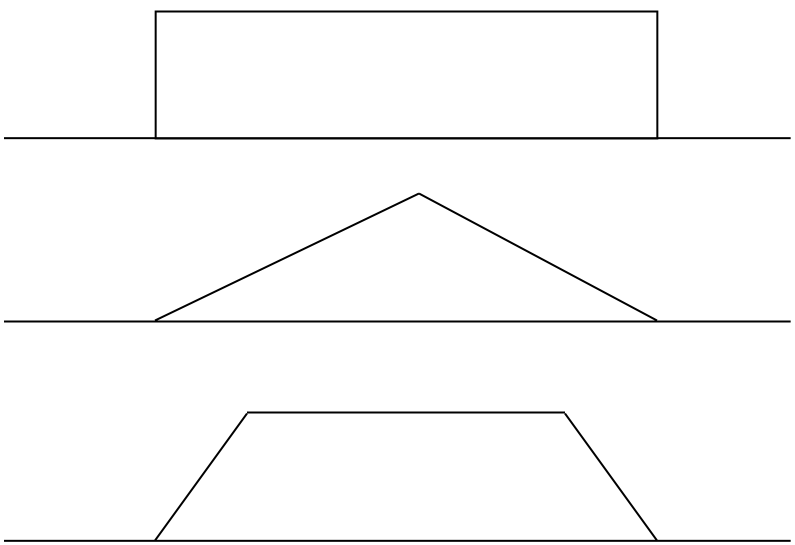


Geometric windows:

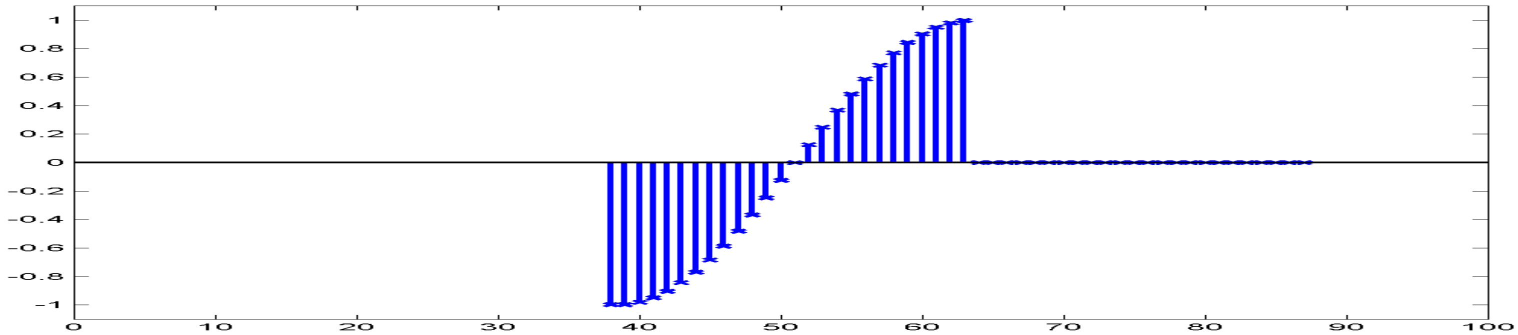
– Rectangular (boxcar):

– Triangular (Bartlett):

– Trapezoid:

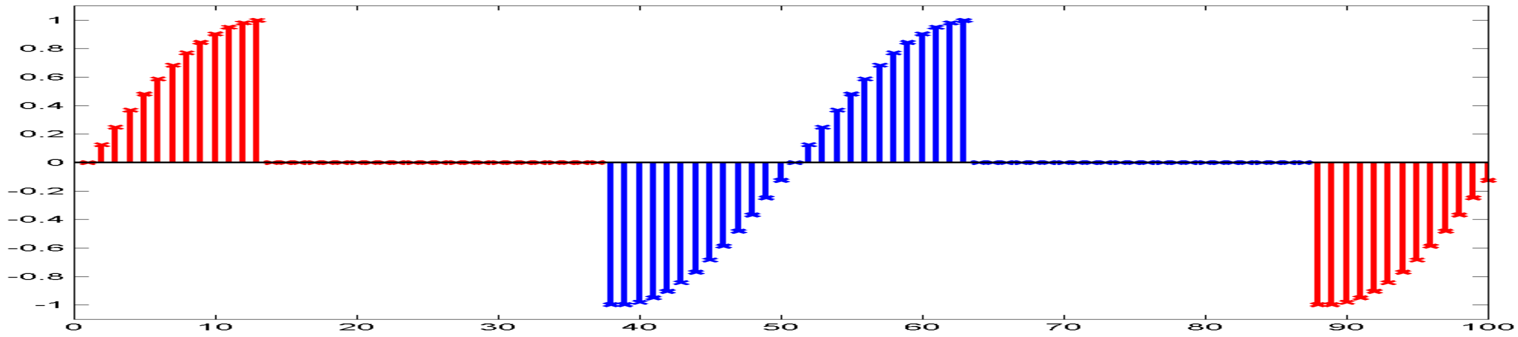


Zero Padding



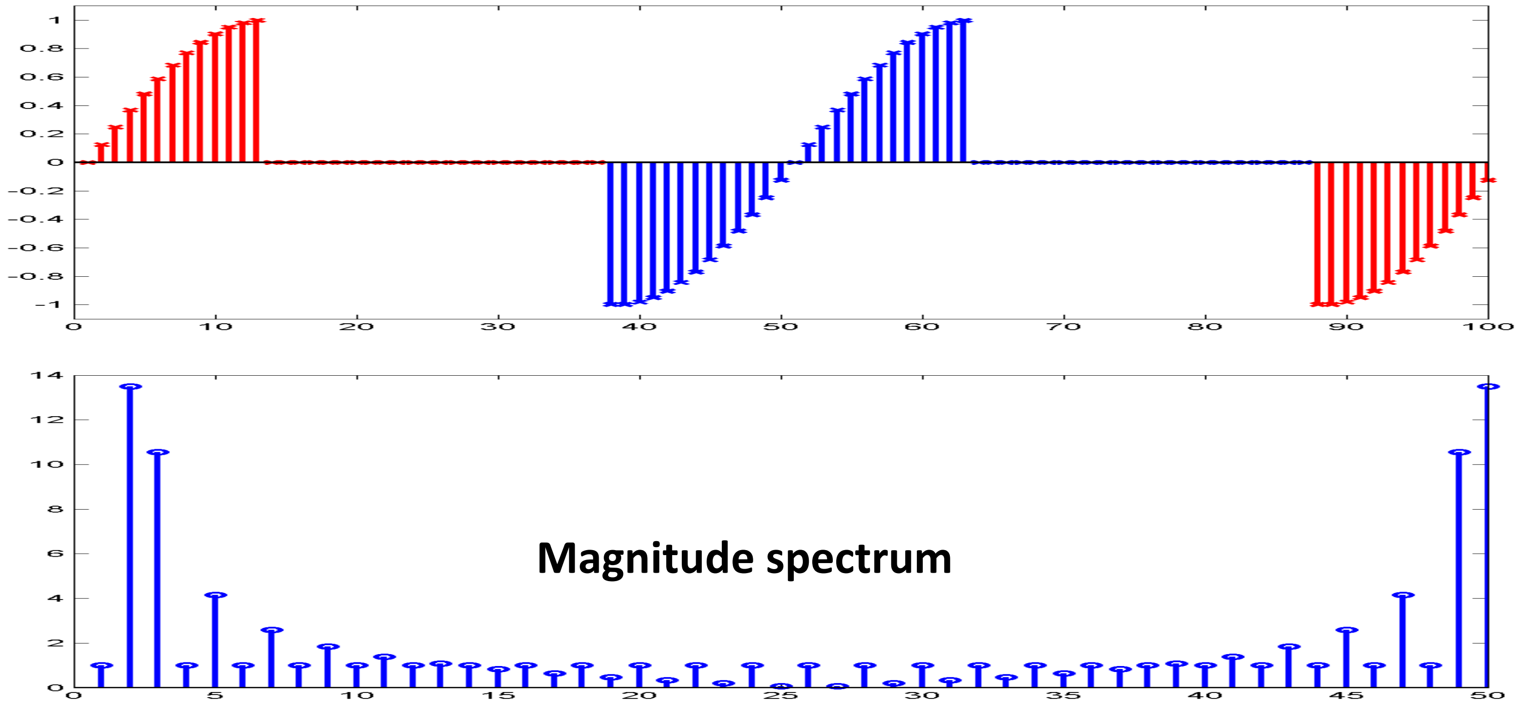
- We can pad zeros to the end of a signal to make it a desired length
  - Useful if the FFT (or any other algorithm we use) requires signals of a specified length
  - E.g. Radix 2 FFTs require signals of length 2^n *i.e.*, some power of 2. We must zero pad the signal to increase its length to the appropriate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

Zero Padding



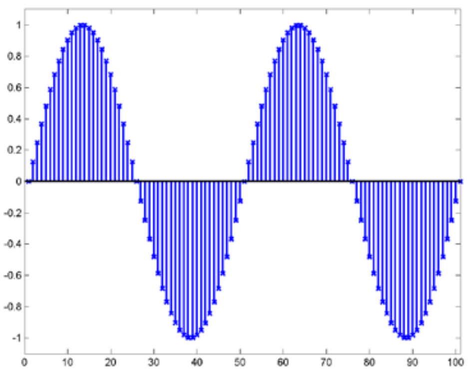
- We can pad zeros to the end of a signal to make it a desired length
  - Useful if the FFT (or any other algorithm we use) requires signals of a specified length
  - E.g. Radix 2 FFTs require signals of length 2^n i.e., some power of 2. We must zero pad the signal to increase its length to the appropriate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

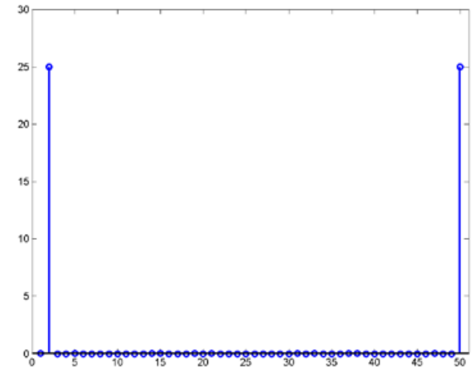
Zero Padding

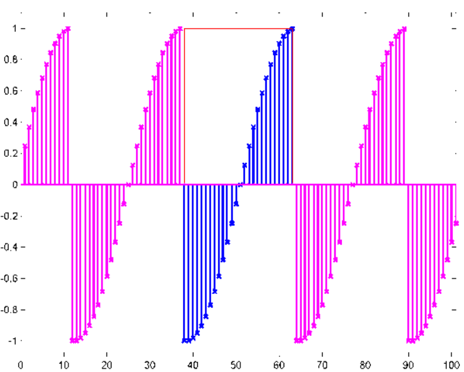


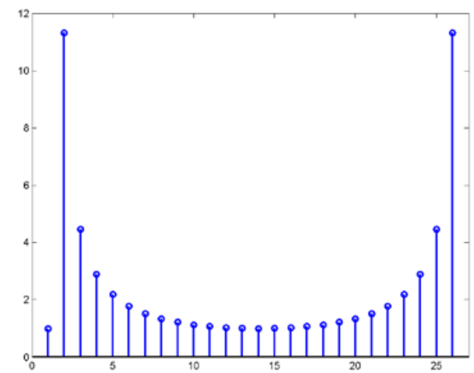
- The DFT of the zero padded signal is essentially the same as the DFT of the unpadded signal, with additional spectral samples inserted in between
  - It does not contain any additional information over the original DFT
  - It also does not contain less information

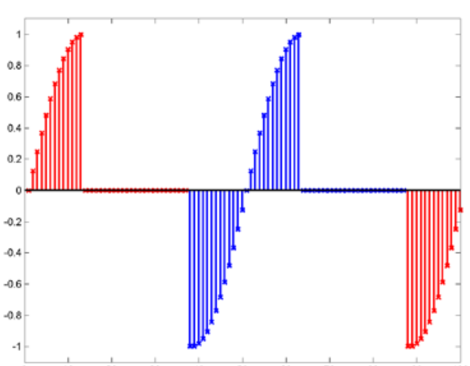
Magnitude spectra

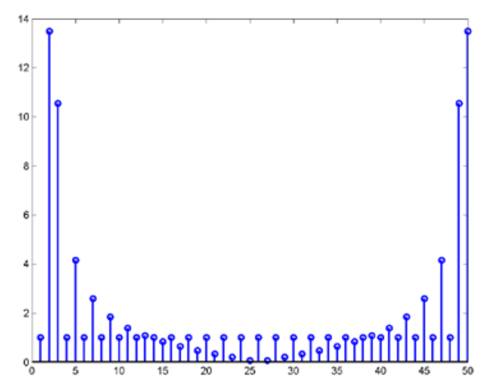




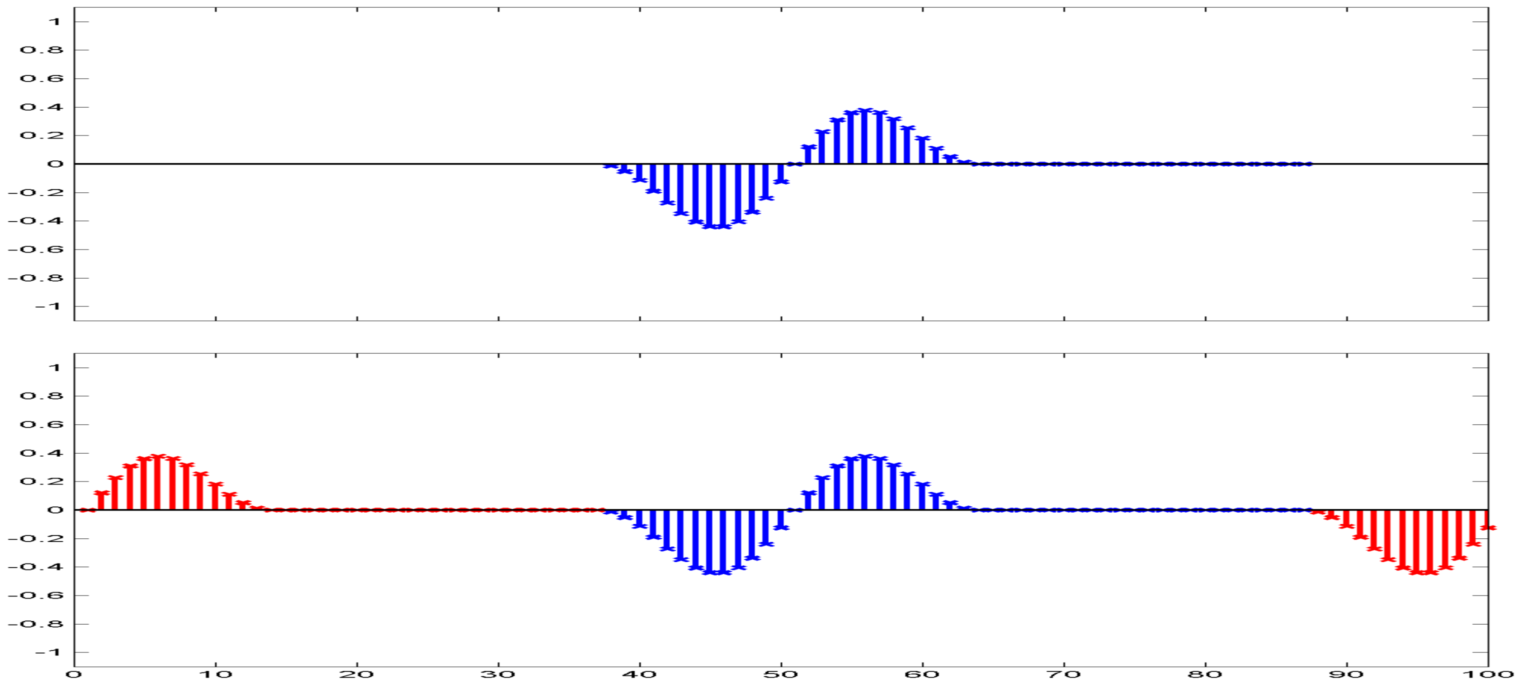






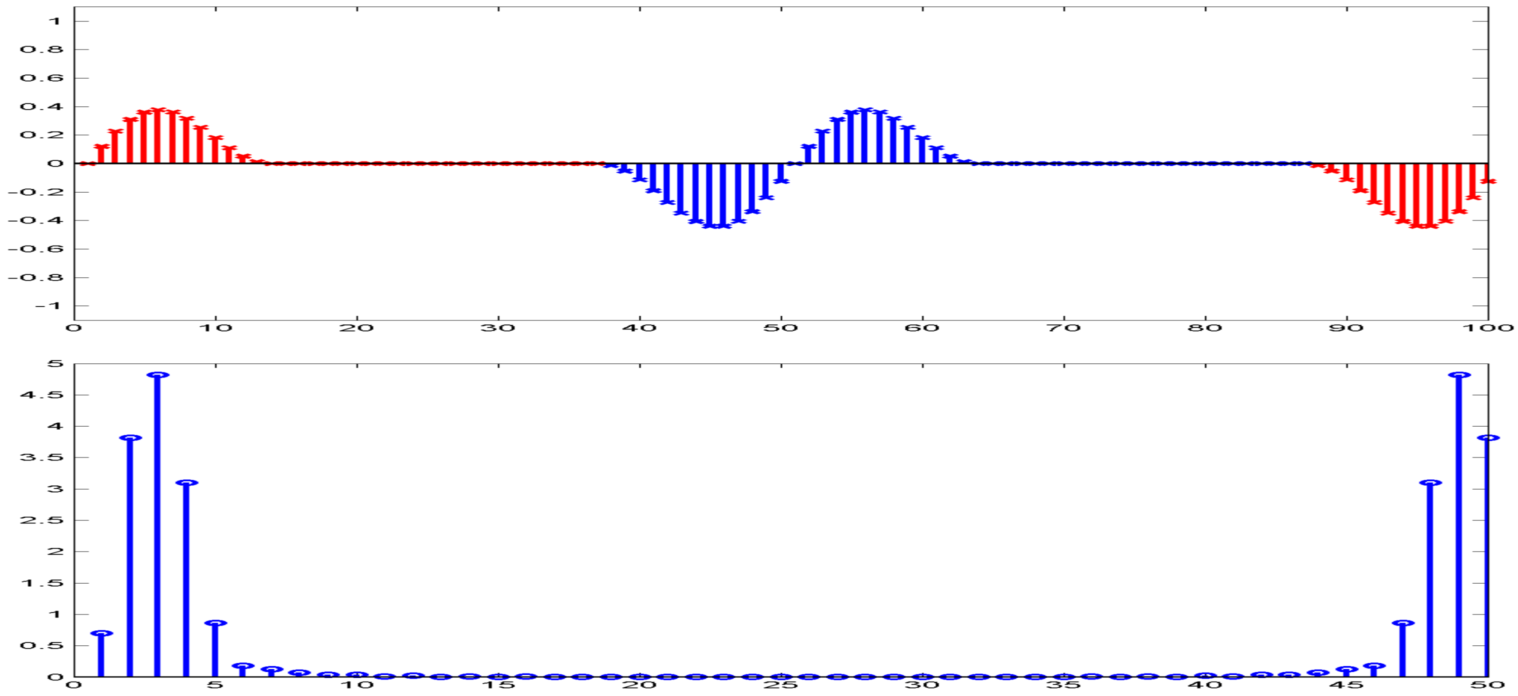


Zero Padding



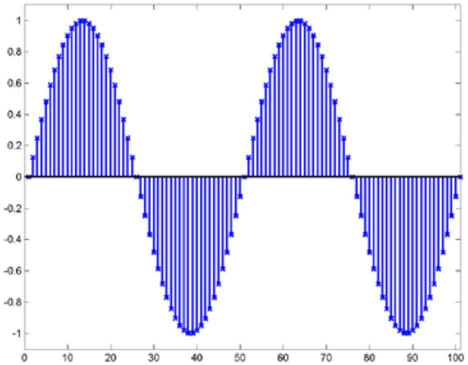
- Zero padding windowed signals results in signals that appear to be less discontinuous at the edges
  - This is only illusory
  - Again, we do not introduce any new information into the signal by merely padding it with zeros

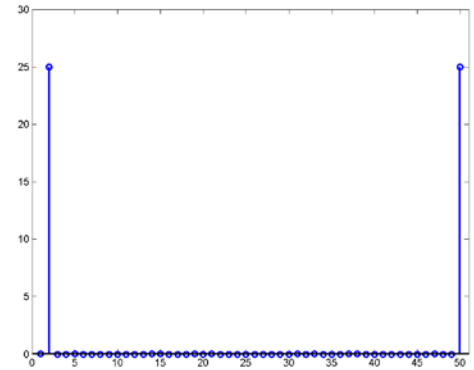
Zero Padding

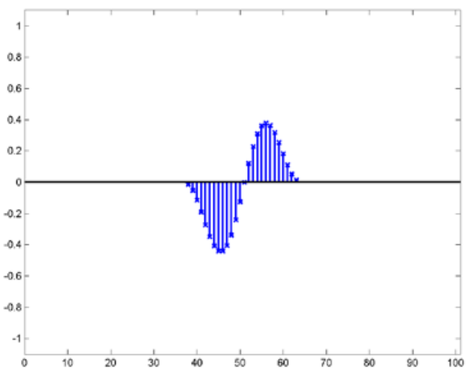


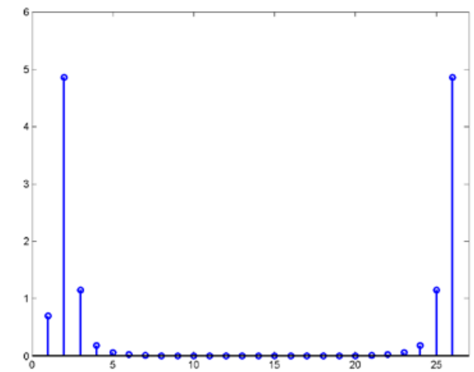
- The DFT of the zero padded signal is essentially the same as the DFT of the unpadded signal, with additional spectral samples inserted in between
  - It does not contain any additional information over the original DFT
  - It also does not contain less information

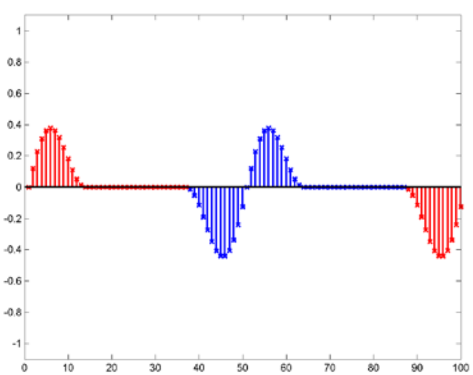
Magnitude spectra

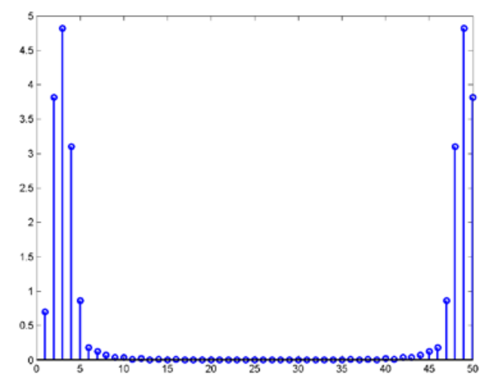






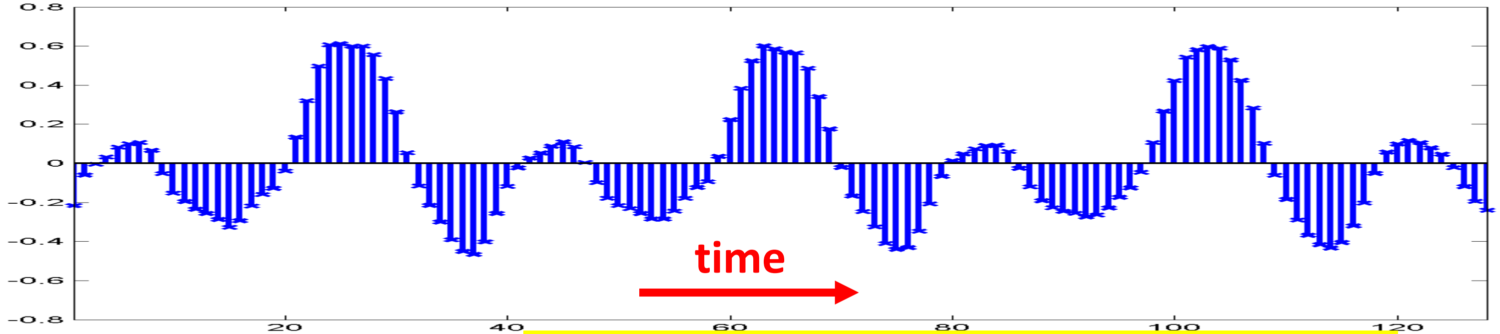




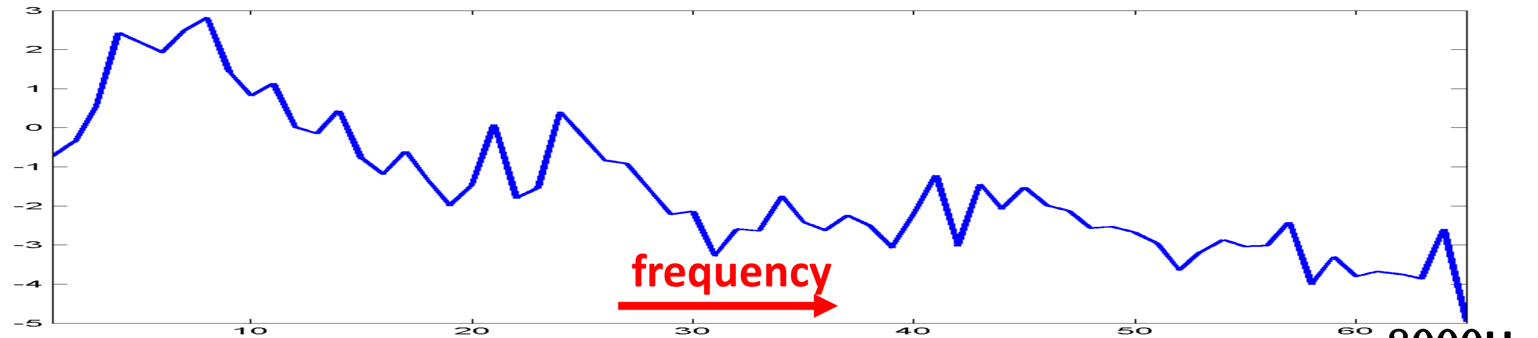


Zero padding a speech signal

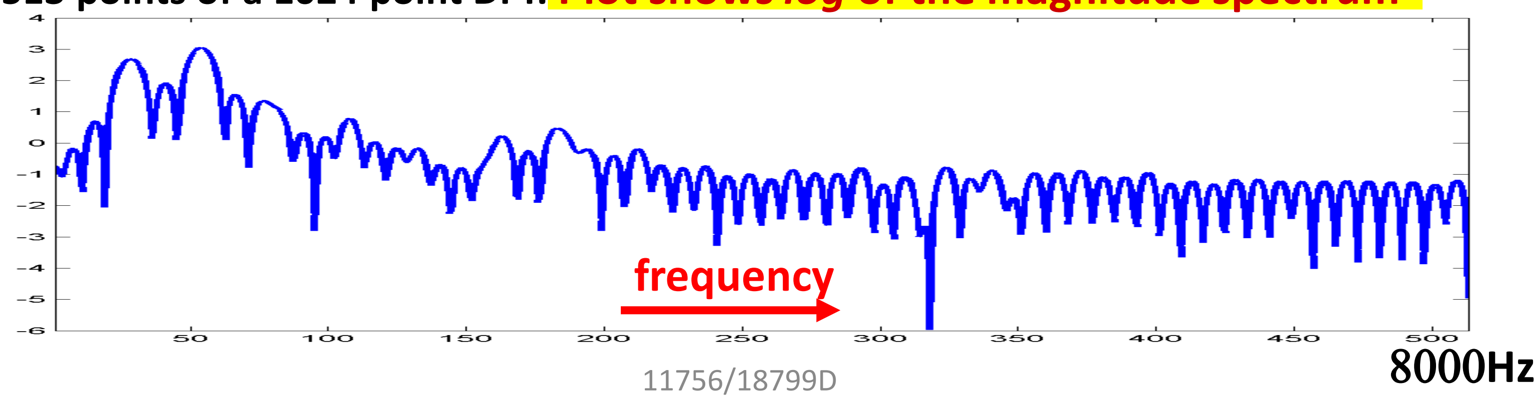
128 samples from a speech signal sampled at 16000 Hz



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



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

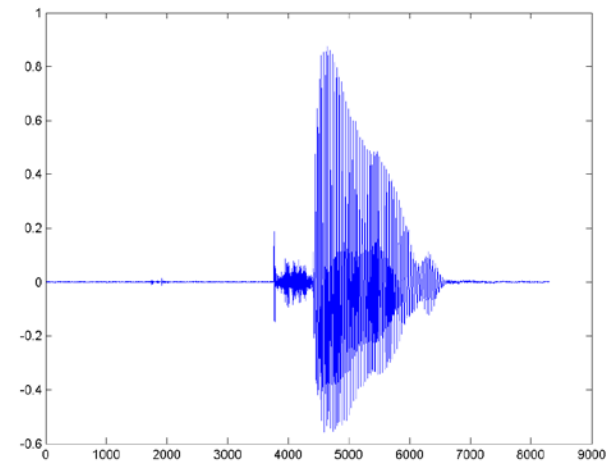


Preemphasizing a speech signal

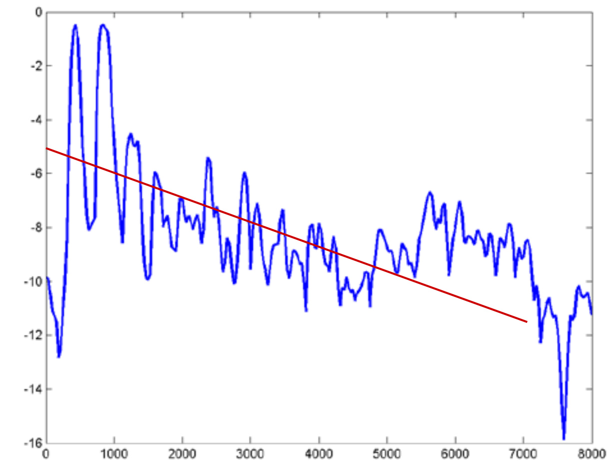
- The spectrum of the speech signal naturally has lower energy at higher frequencies

- 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

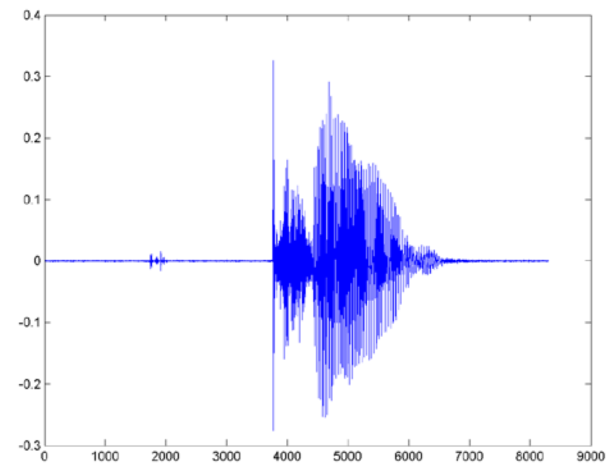


Log(average(magnitude spectrum))

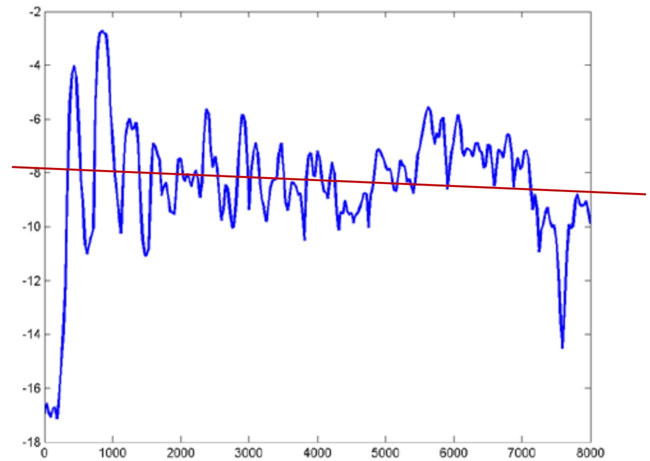


Preemphasizing a speech signal

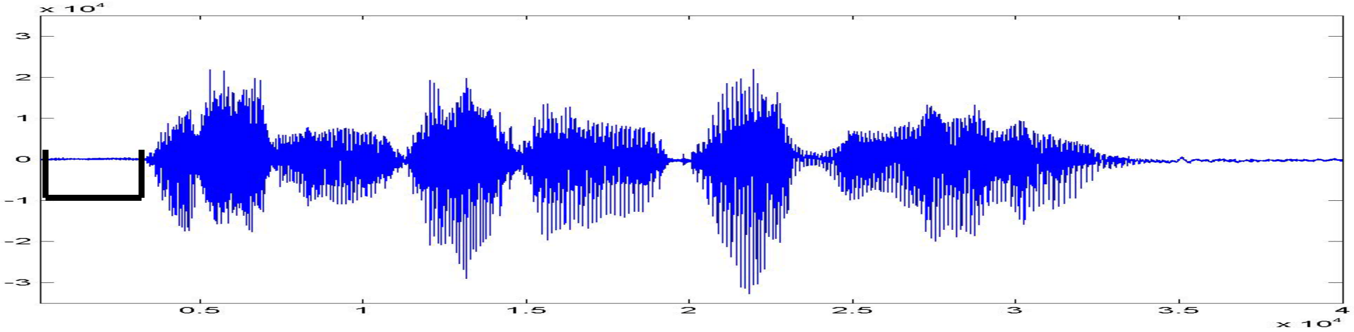
- This spectral tilt can be corrected by preemphasizing the signal
  - $s_{preemp}[n] = s[n] – \alpha * s[n-1]$
  - Typical value of $\alpha = 0.95$
- This is a form of differentiation that boosts high frequencies
- This spectrum of the preemphasized signal has more horizontal trend
  - Good for linear prediction and other similar methods



Log(average(magnitude spectrum))

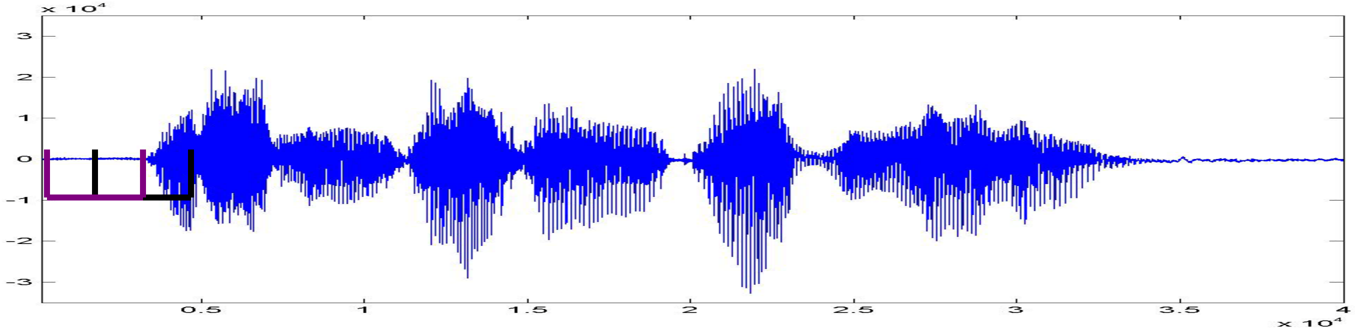


The process of parametrization



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

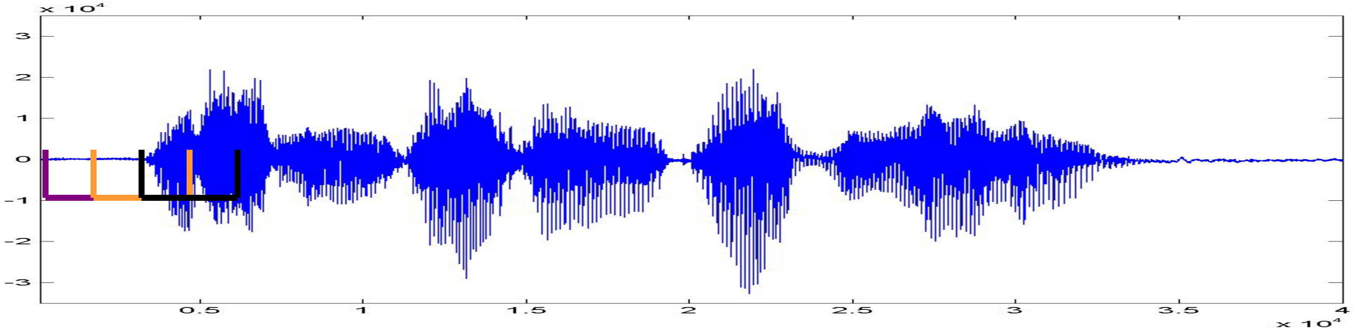
The process of parametrization



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

Adjacent segments typically overlap by 15 ms.

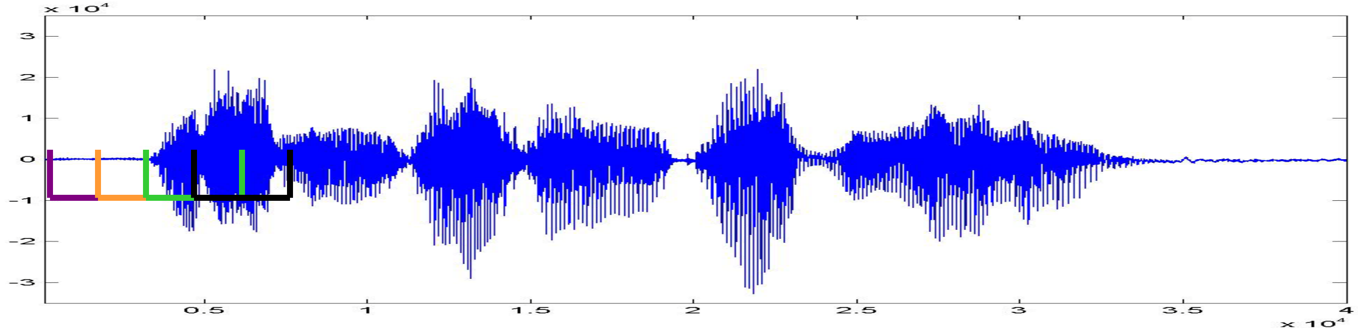
The process of parametrization



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

Adjacent segments typically overlap by 15 ms.

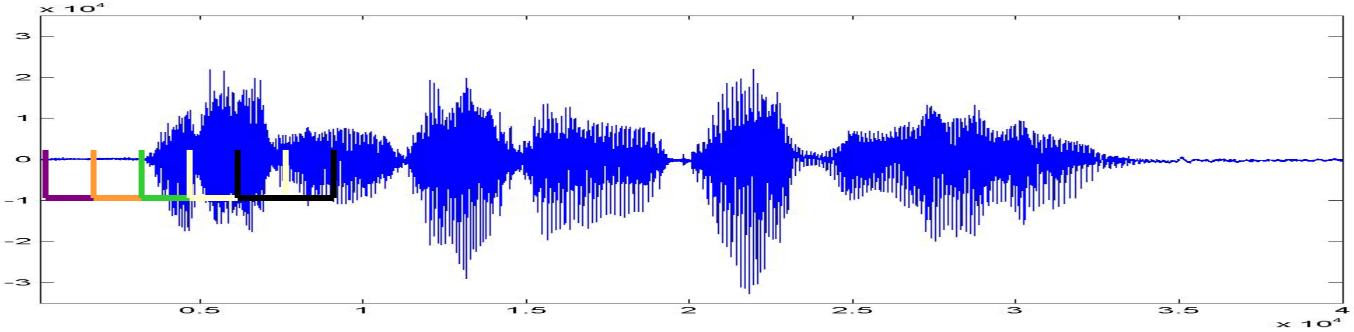
The process of parametrization



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

Adjacent segments typically overlap by 15 ms.

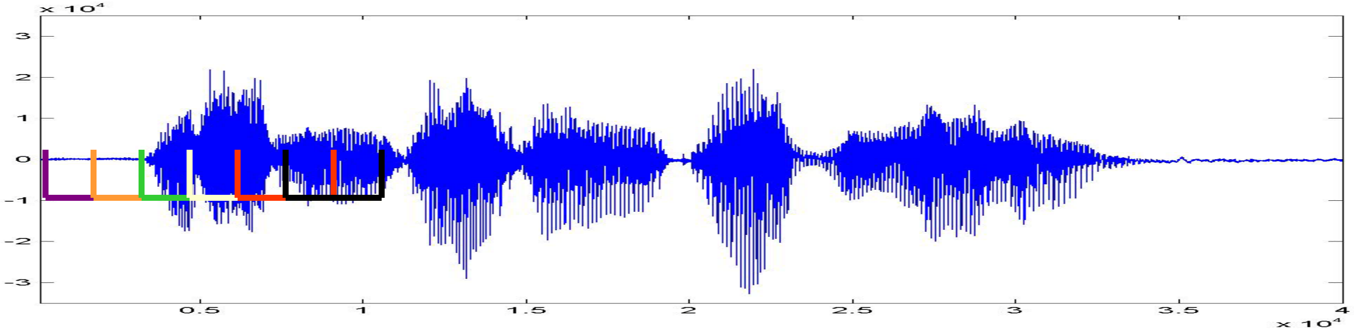
The process of parametrization



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

Adjacent segments typically overlap by 15 ms.

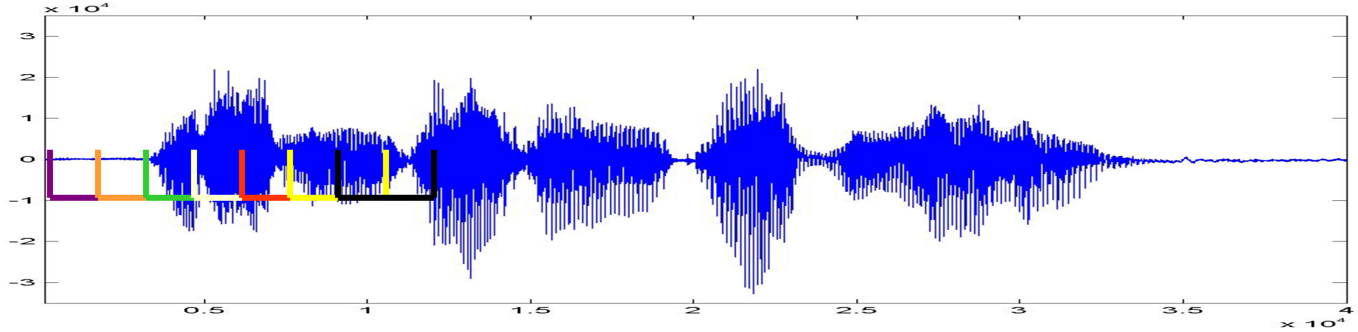
The process of parametrization



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

Adjacent segments typically overlap by 15 ms.

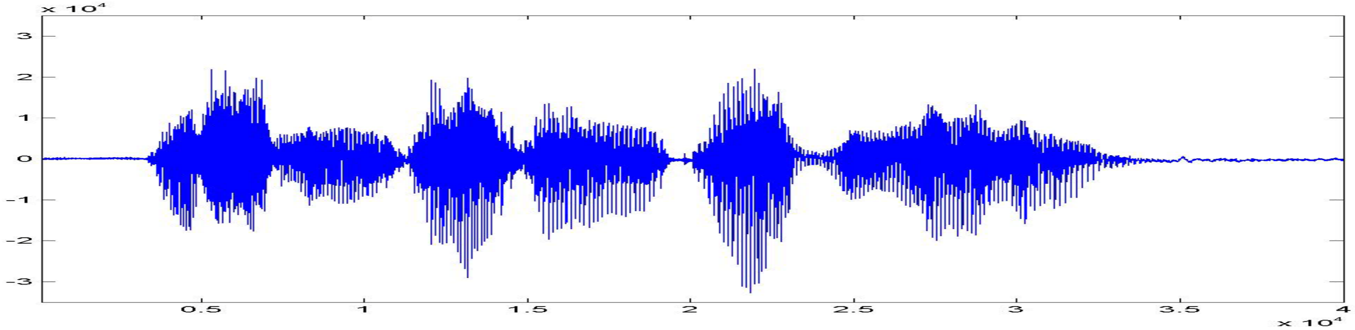
The process of parametrization

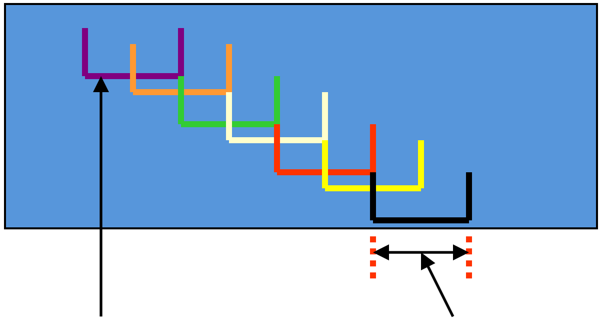


The signal is processed in segments. Segments are typically 25 ms wide.

Adjacent segments typically overlap by 15 ms.

The process of parametrization



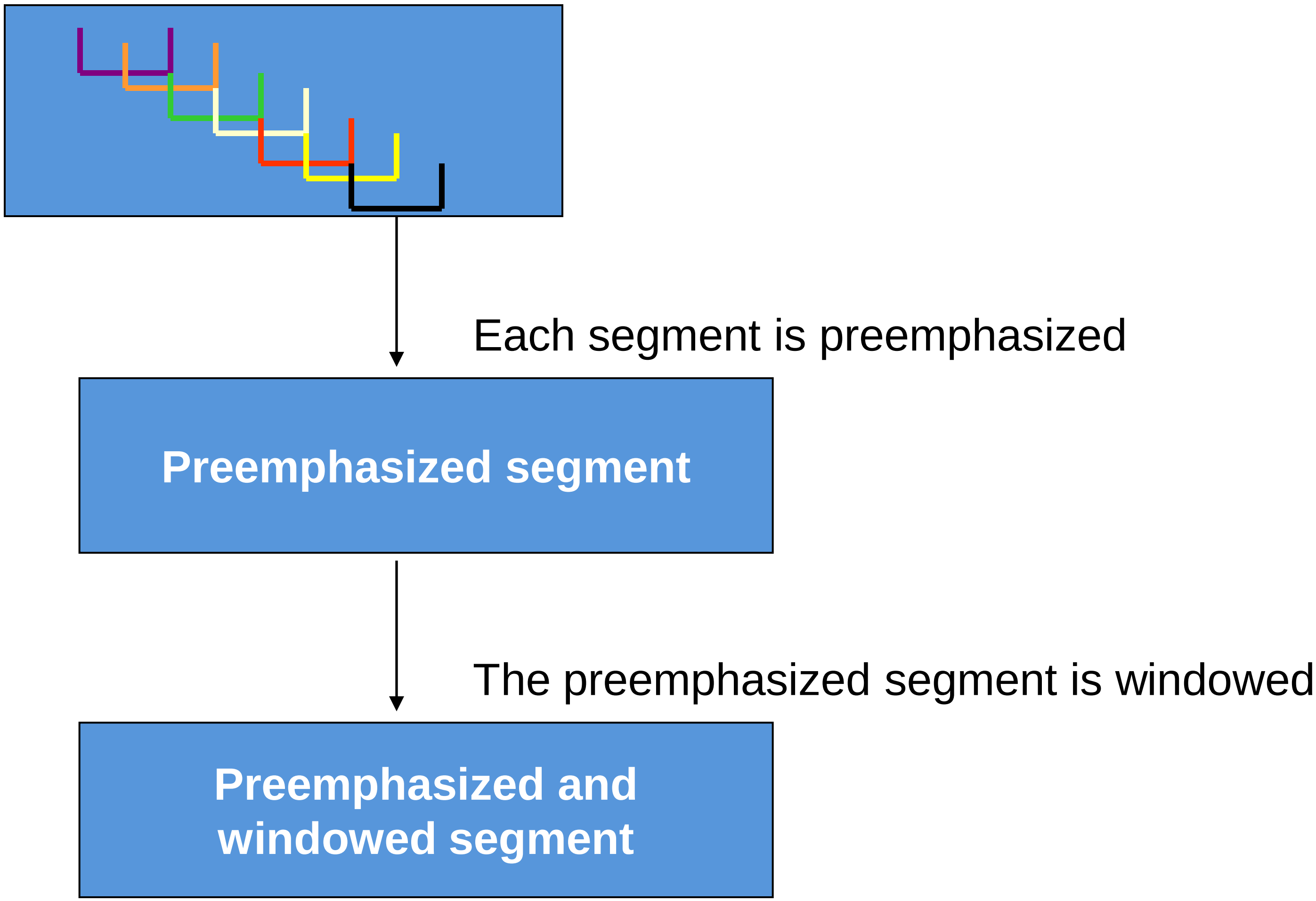


Segments shift every 10 milliseconds

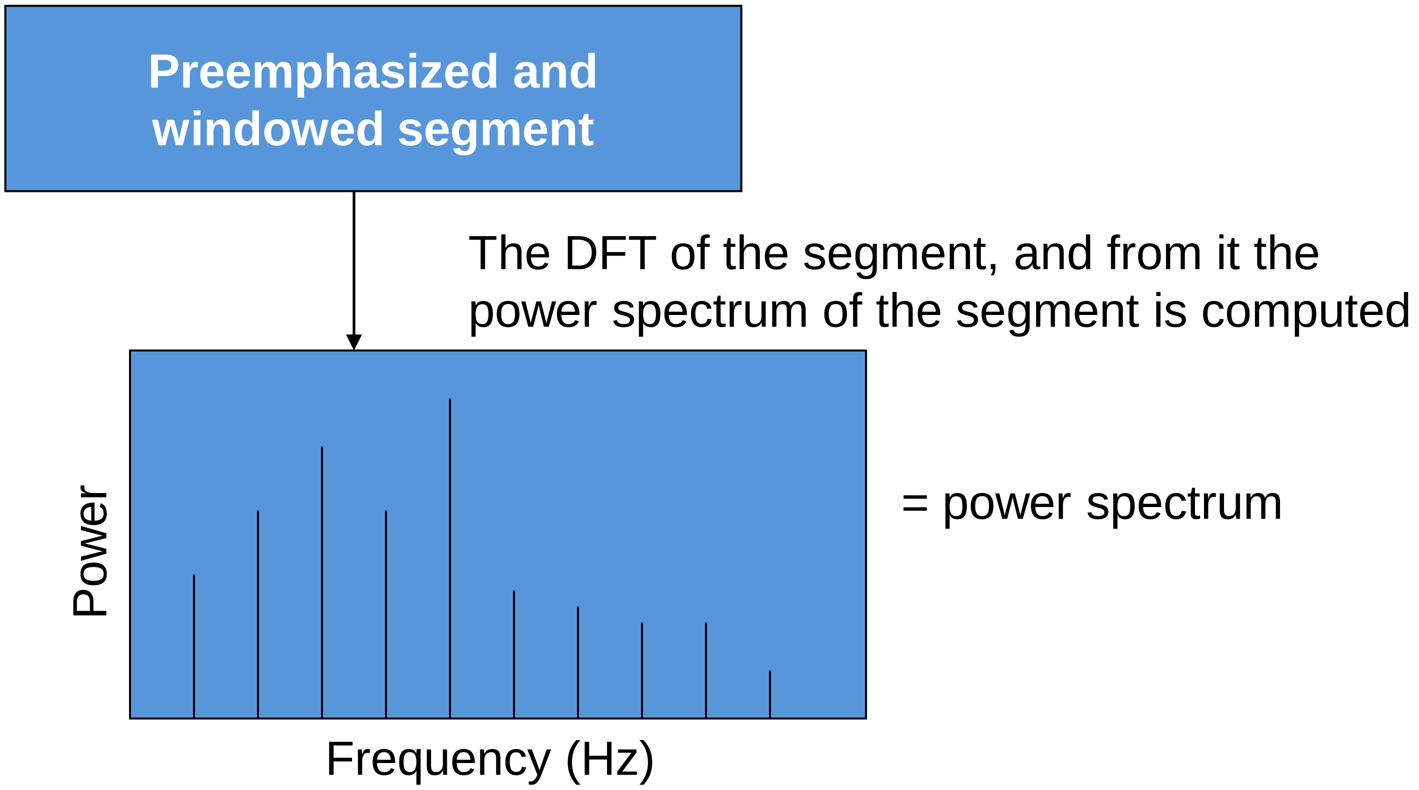
Each segment is typically 20 or 25 milliseconds wide

Speech signals do not change significantly within this short time interval

The process of parametrization



The process of parametrization



Auditory Perception

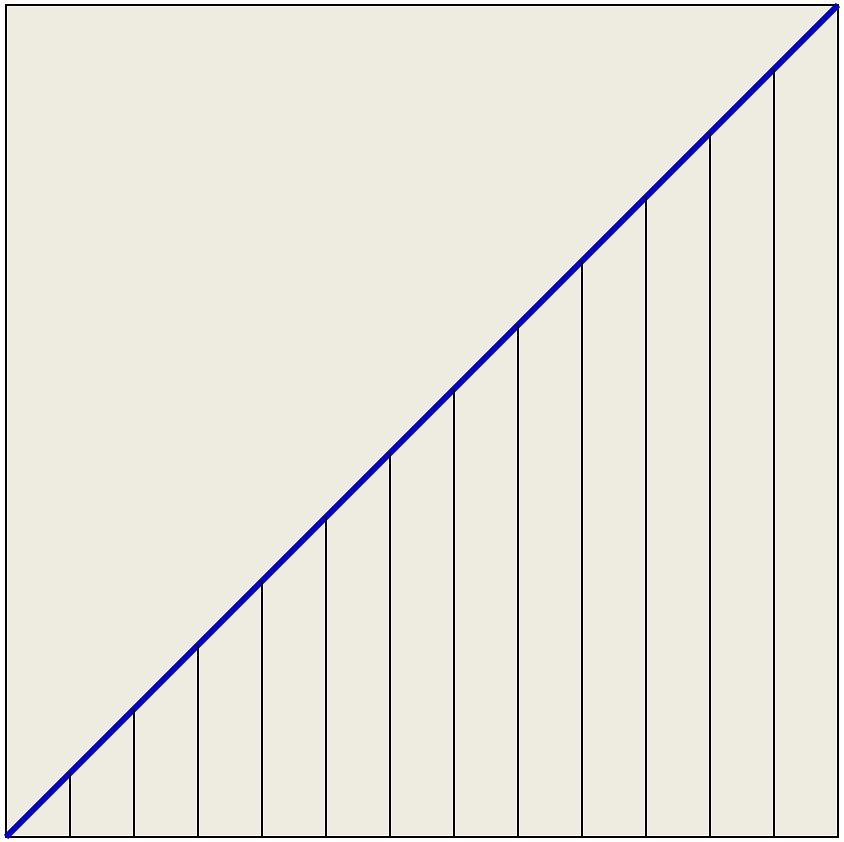
- Conventional Spectral analysis decomposes the signal into a number of linearly spaced frequencies
  - The resolution (differences between adjacent frequencies) is the same at all frequencies
- The human ear, on the other hand, has non-uniform resolution
  - At low frequencies we can detect small changes 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

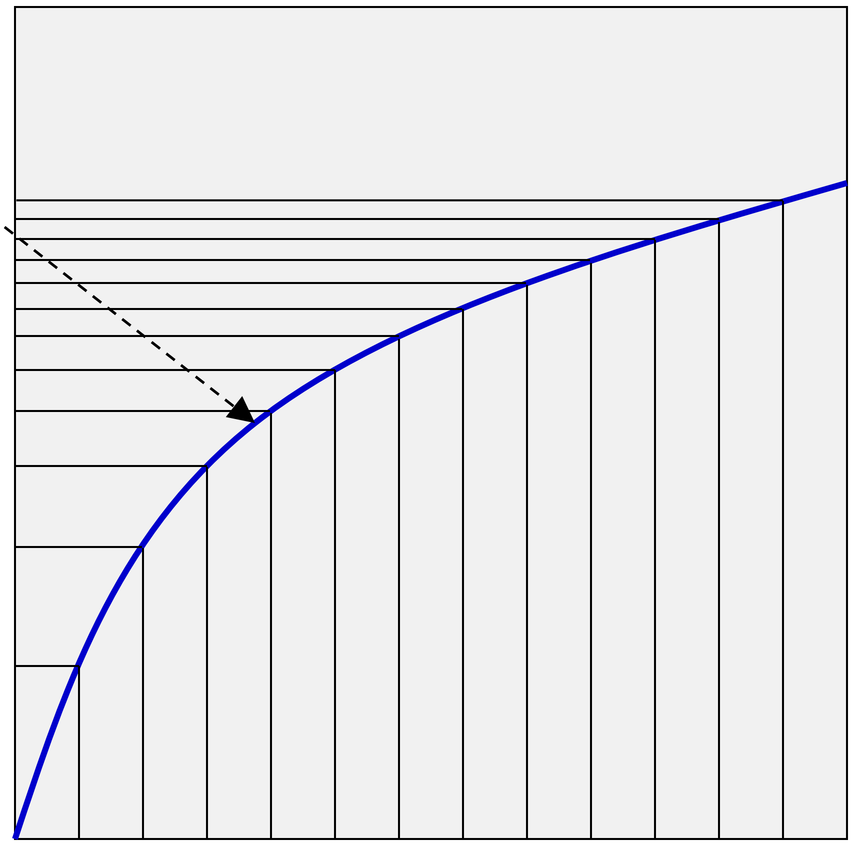
- *Warp* the frequency axis such that small differences between frequencies at lower frequencies are given the same importance as larger differences at higher frequencies

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)

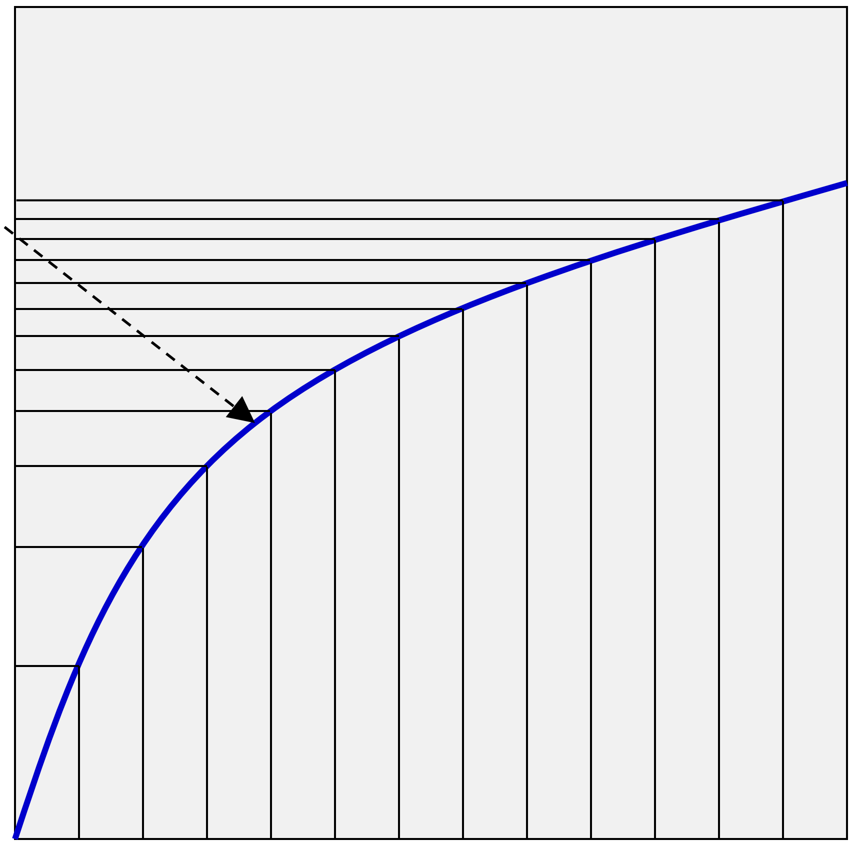
Perceptually warped frequency axis: unequal increments of frequency at equal intervals or conversely, equal increments of frequency at unequal intervals

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)

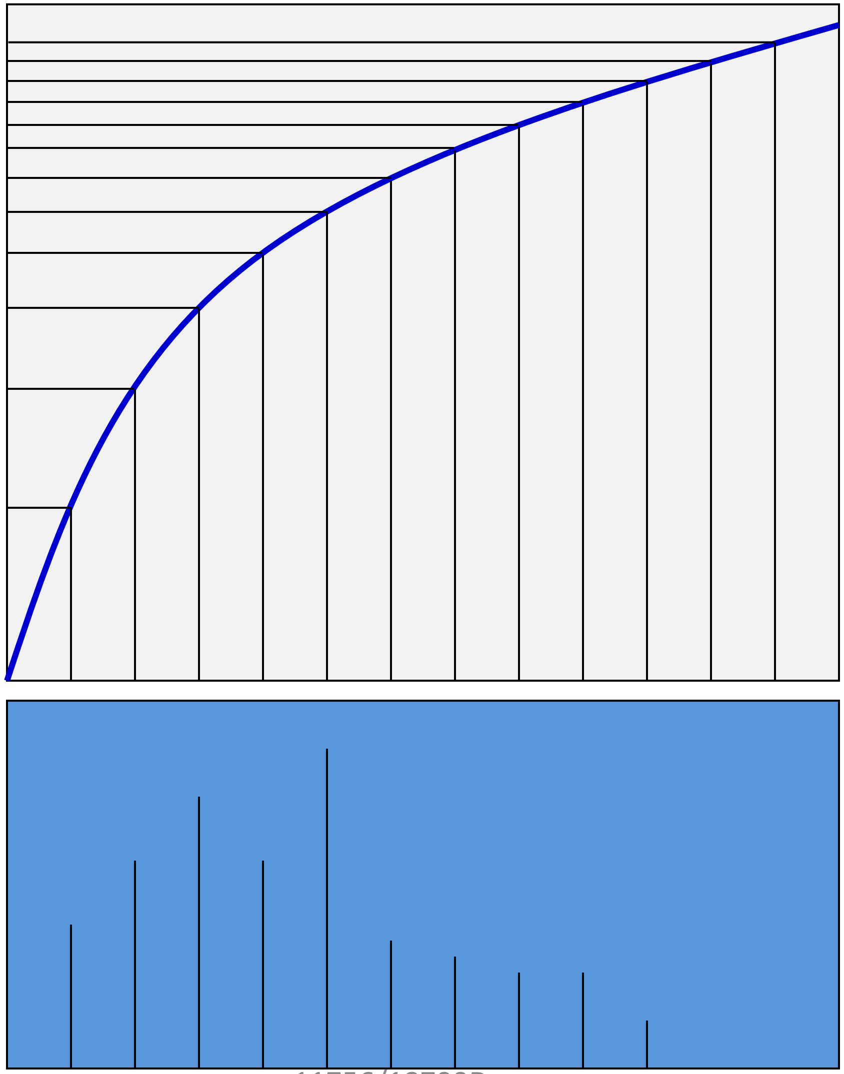


A standard warping function is the Mel warping function

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

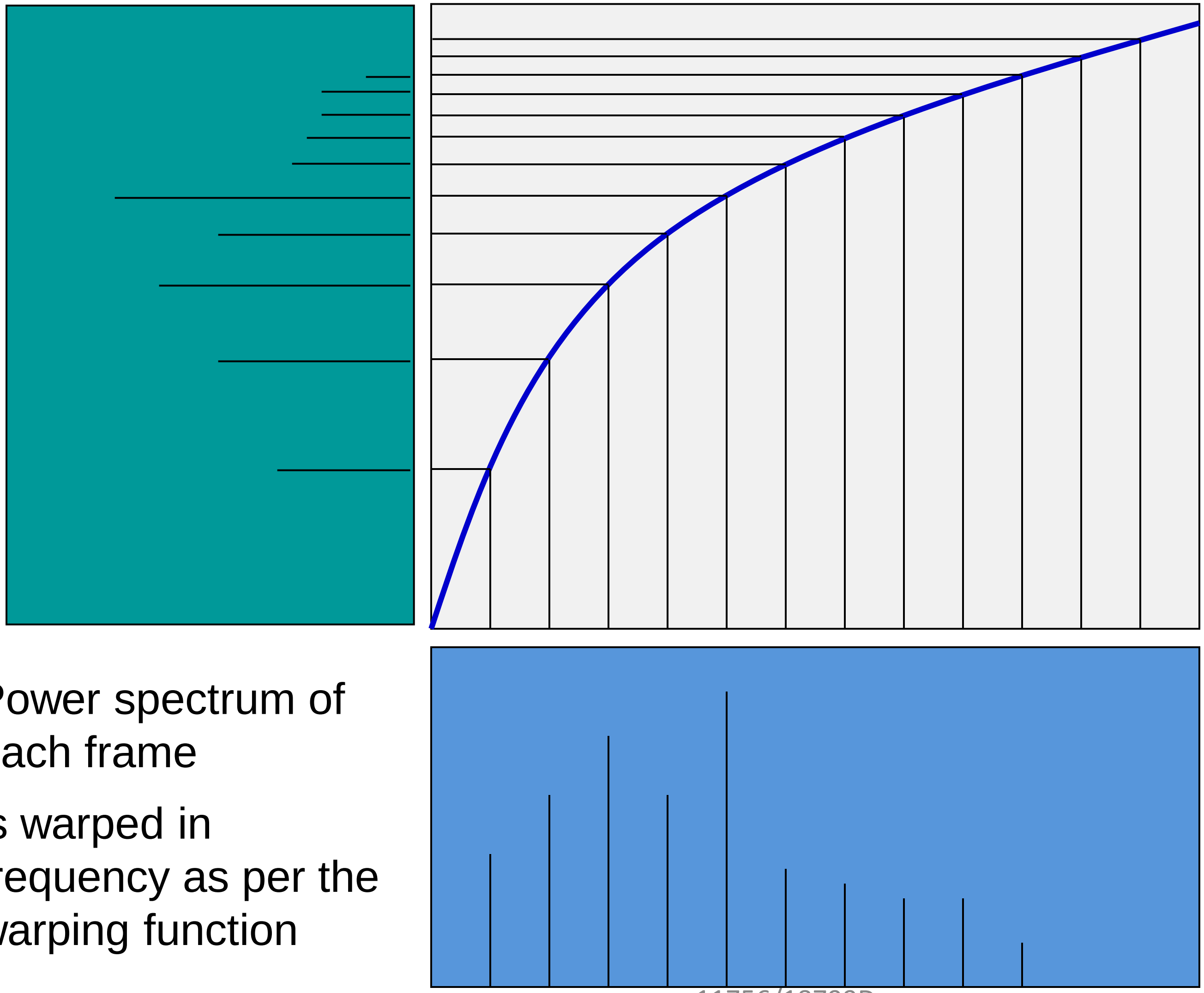
Linear frequency axis: Sampled at uniform intervals by an FFT

The process of parametrization



Power spectrum of each frame

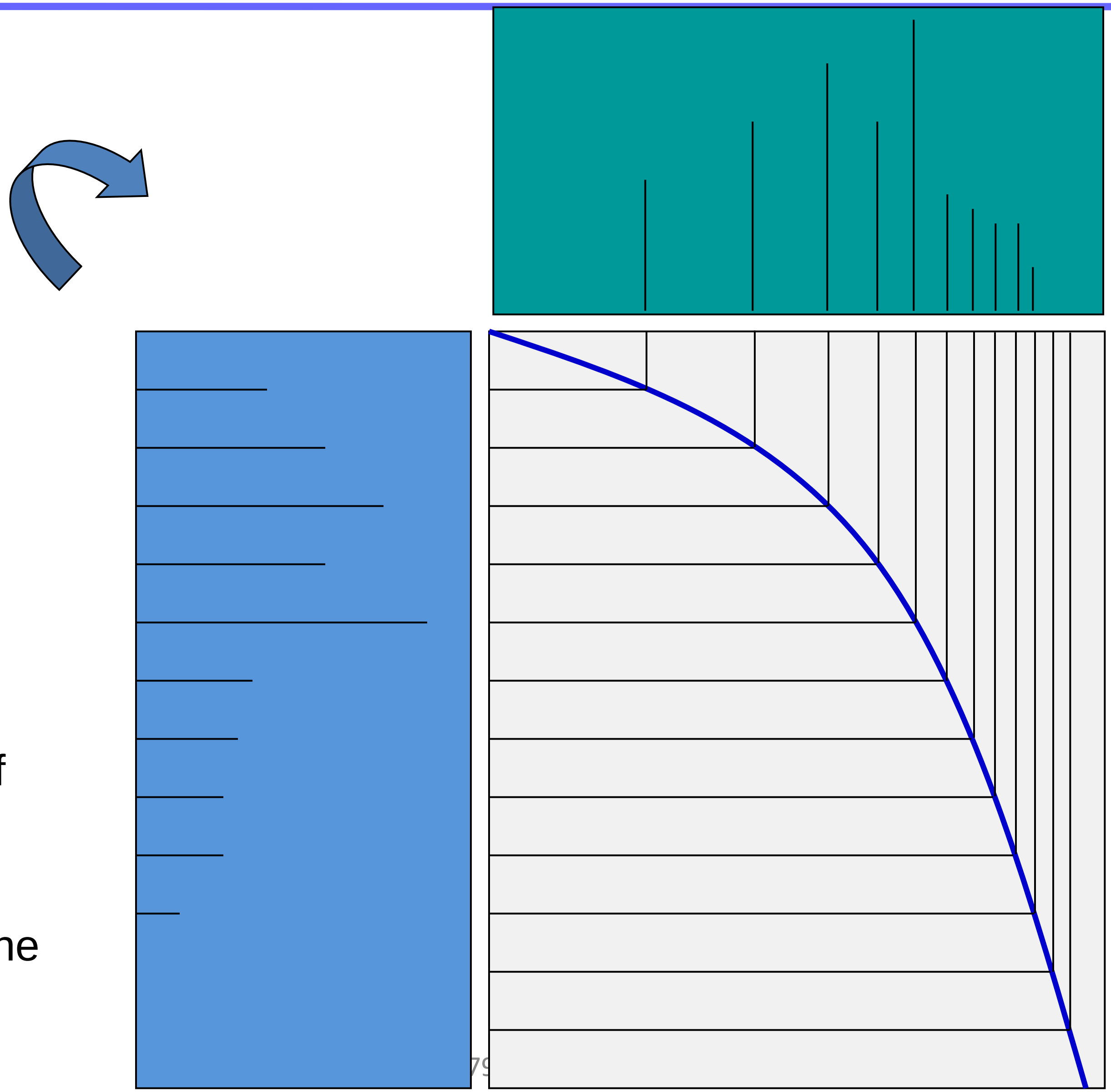
The process of parametrization



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

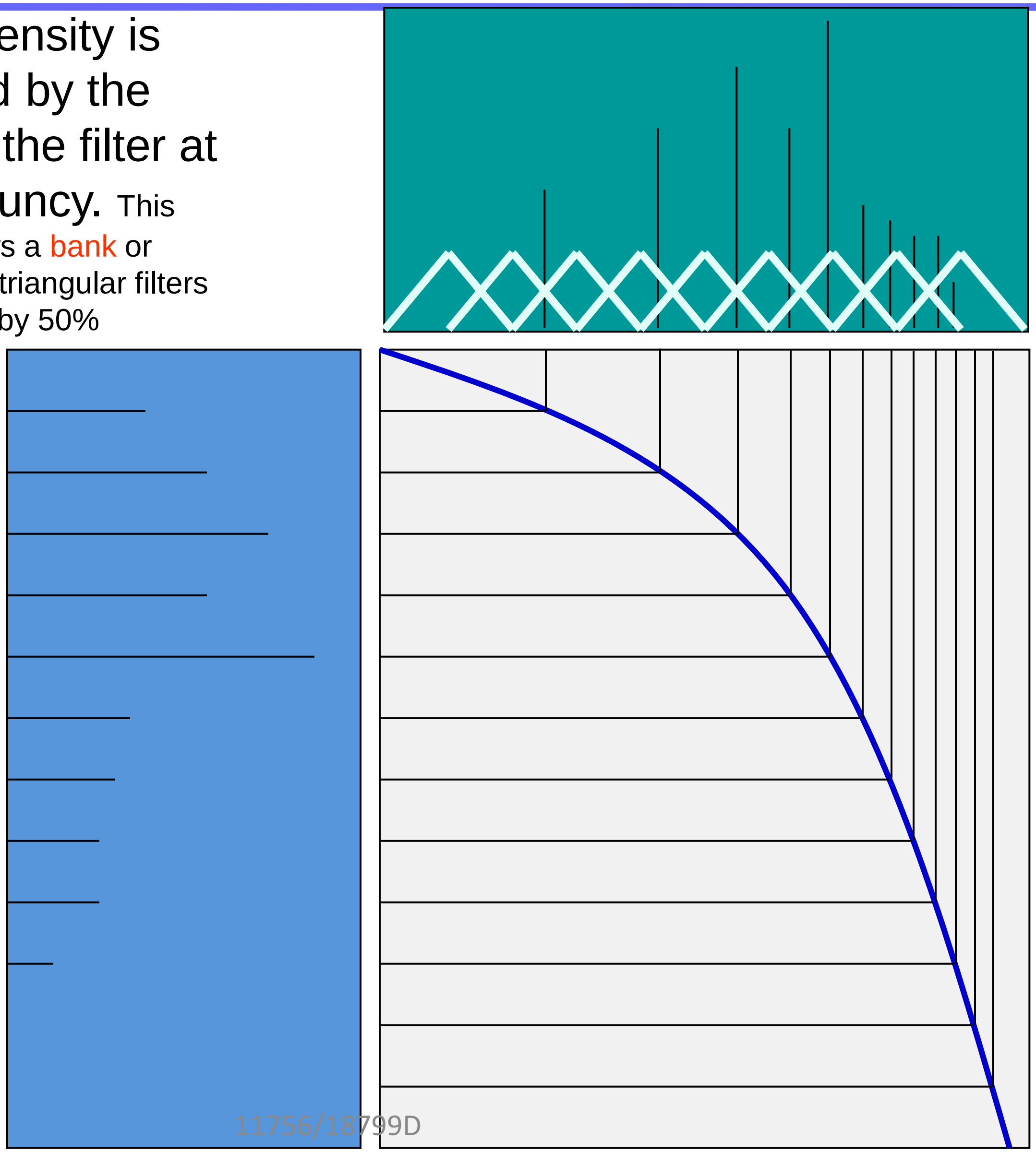
Filter Bank

- Each hair cells 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

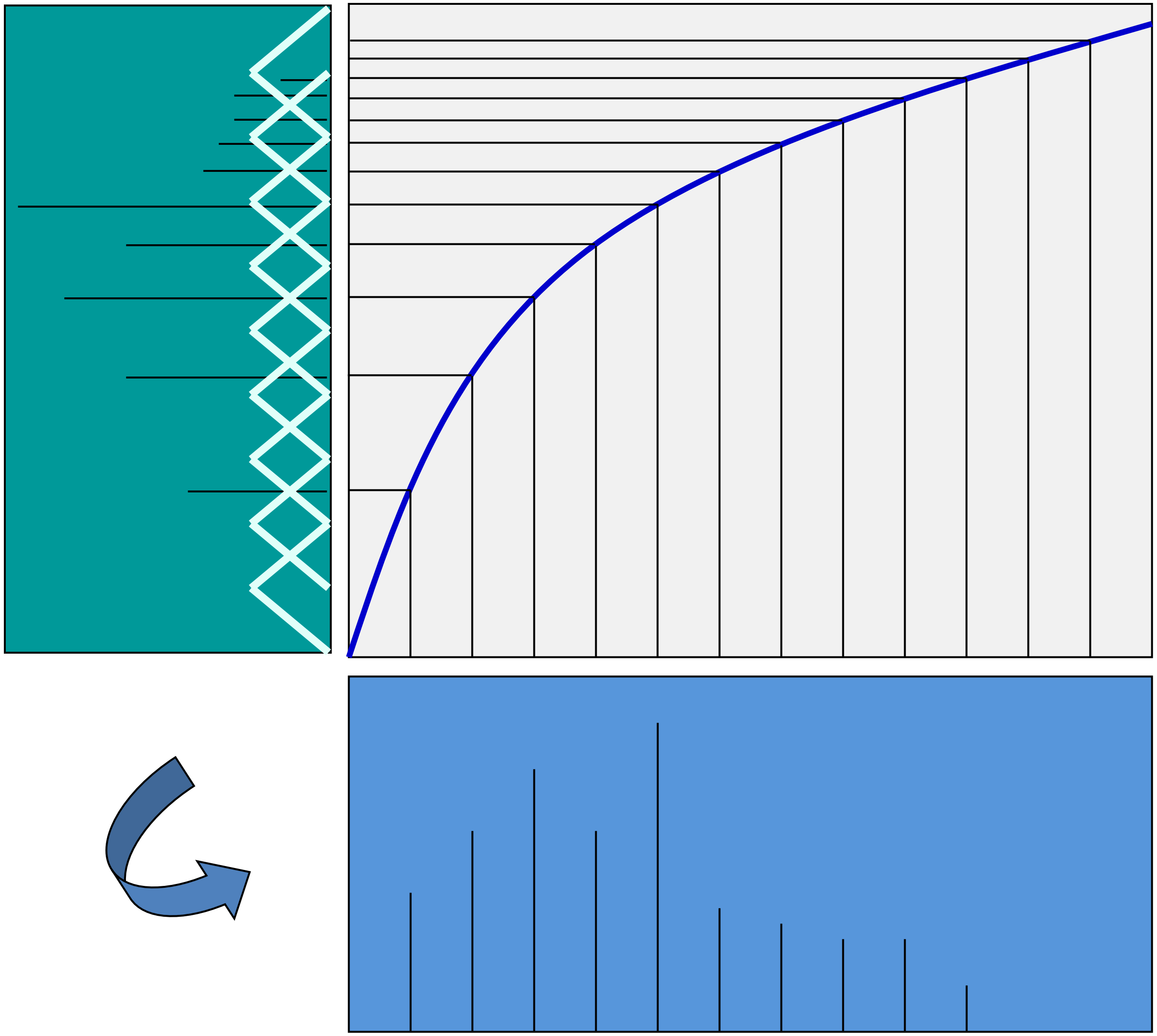
Each intensity is weighted by the value of the filter at that frequncy. This picture shows a bank or collection of triangular filters that overlap by 50%



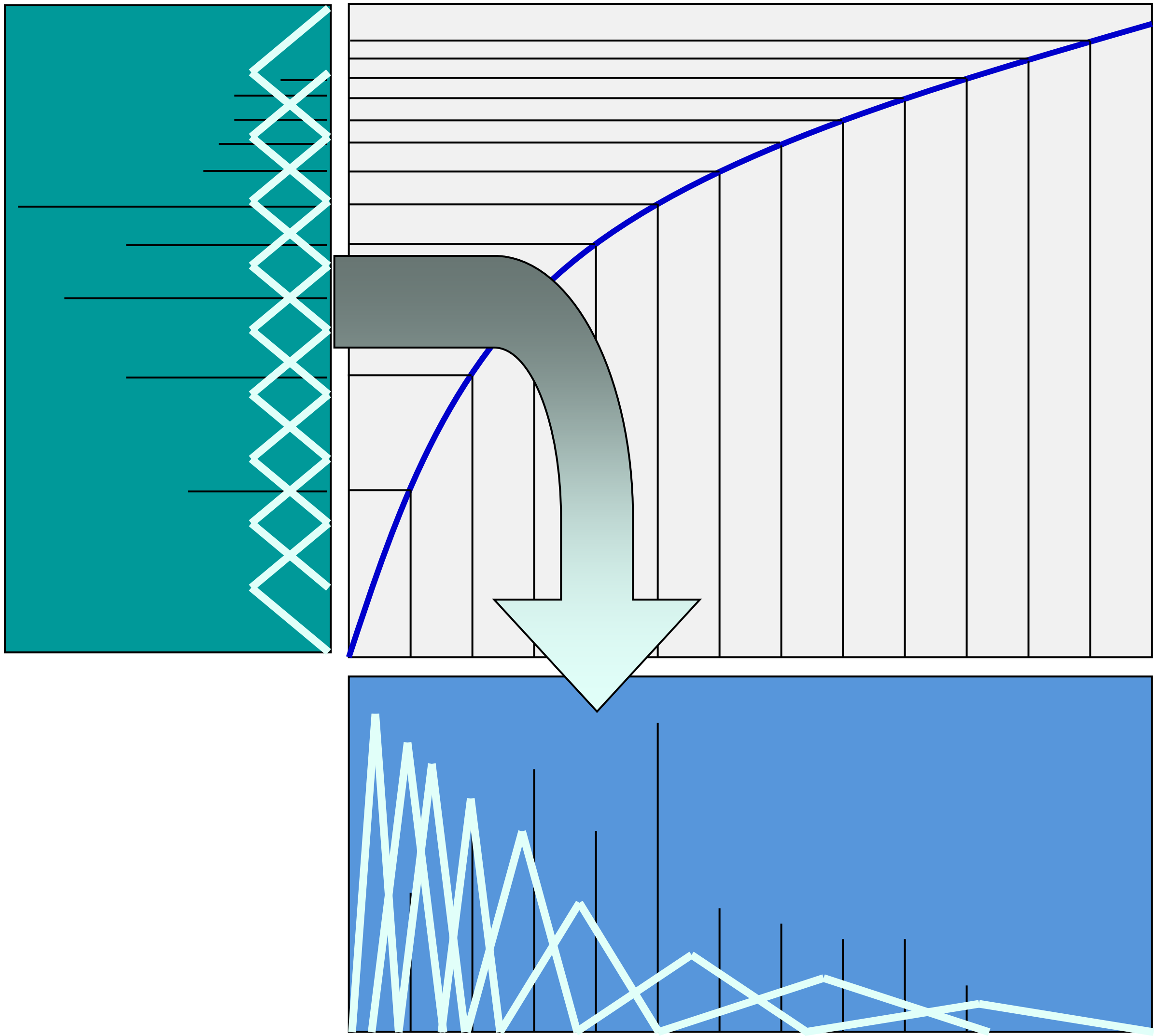
Power spectrum of each frame

is warped in frequency as per the warping function

The process of parametrization

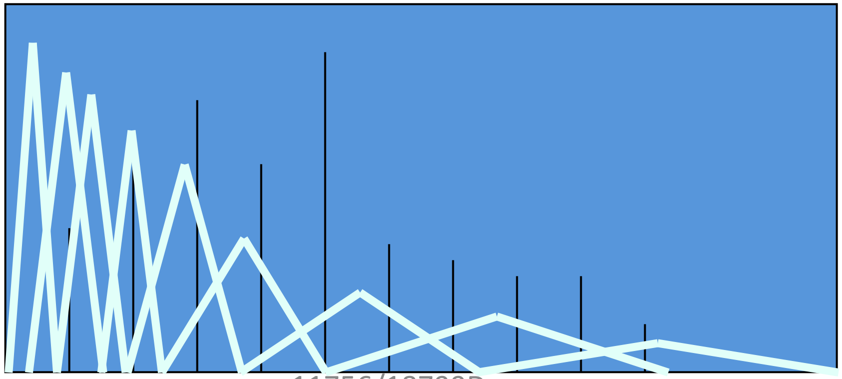


The process of parametrization



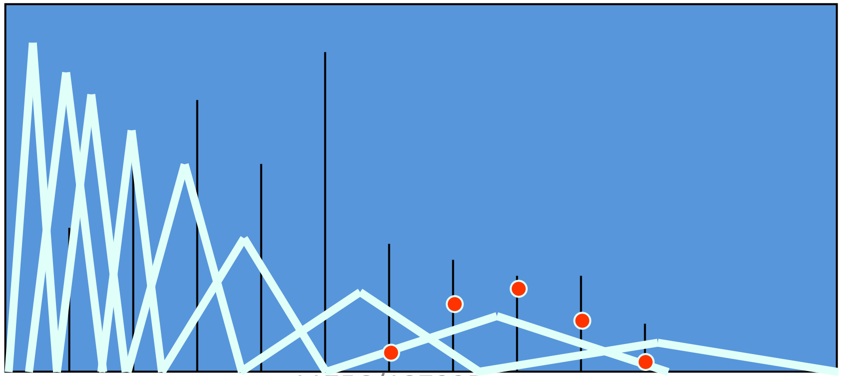
The process of parametrization

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

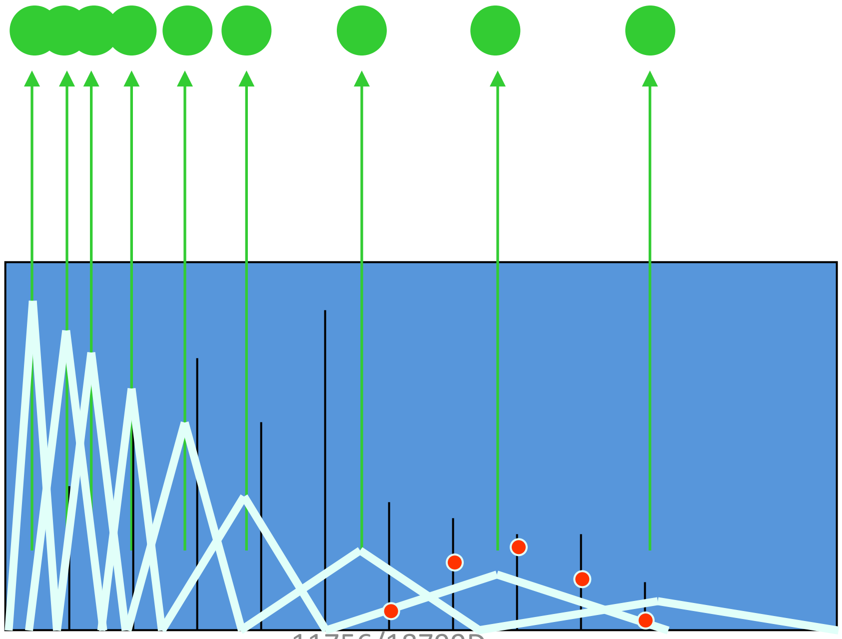


The process of parametrization

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



The process of parametrization

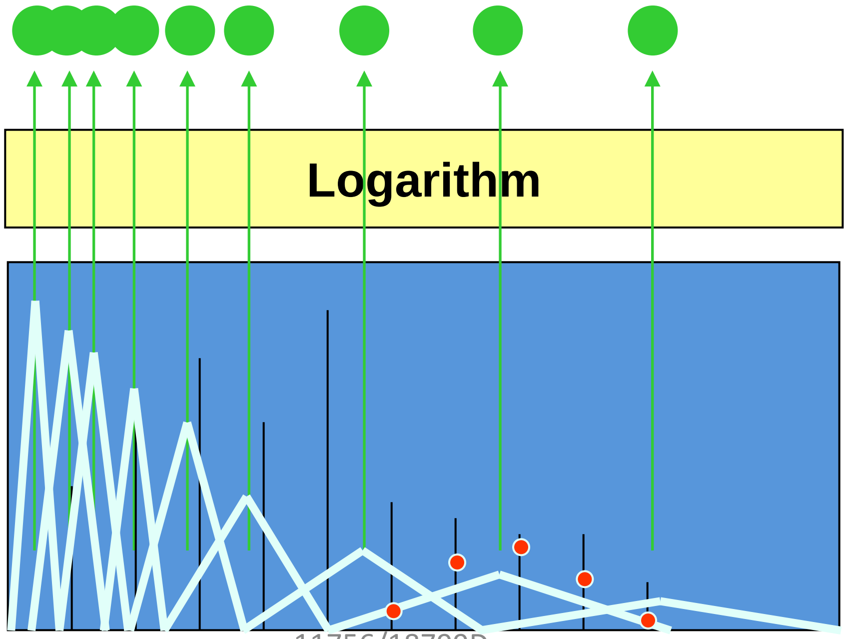


All weighted spectral values for each filter are integrated (added), giving one value per filter

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

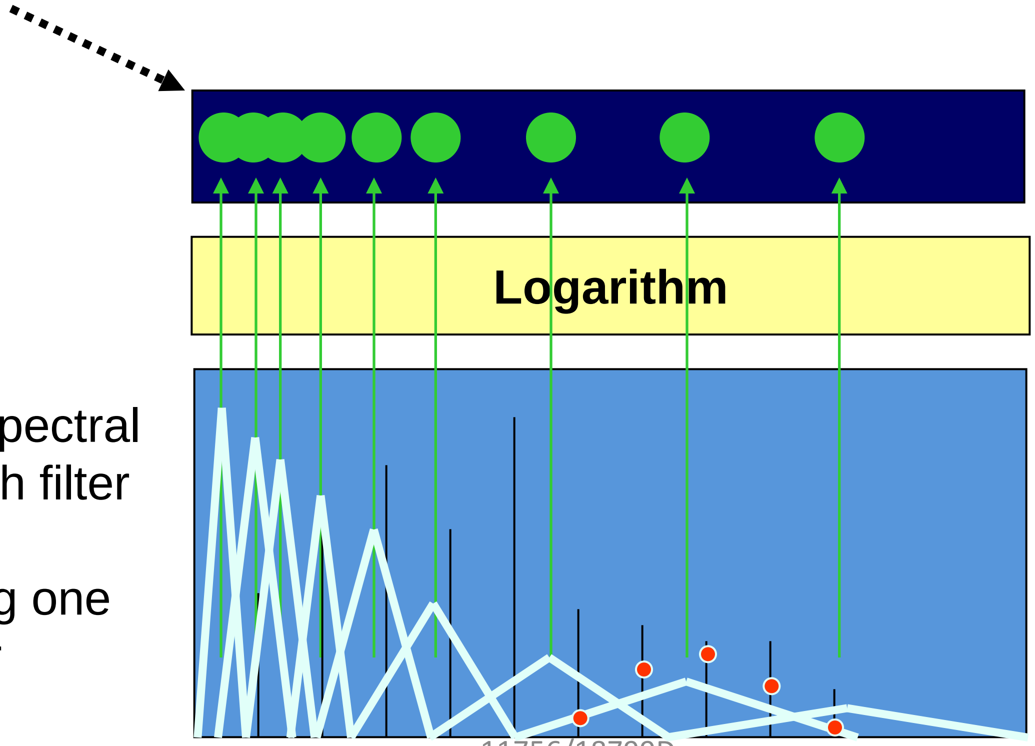


Compress Values

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

The process of parametrization

Log Mel spectrum

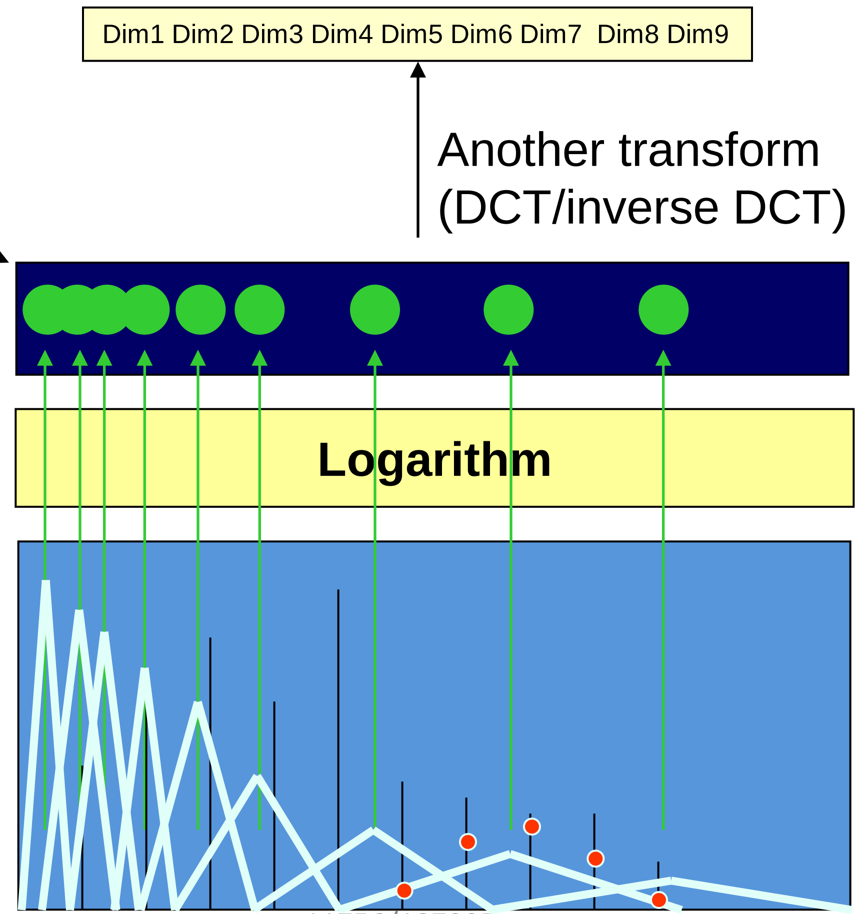


Compress Values

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

The process of parametrization

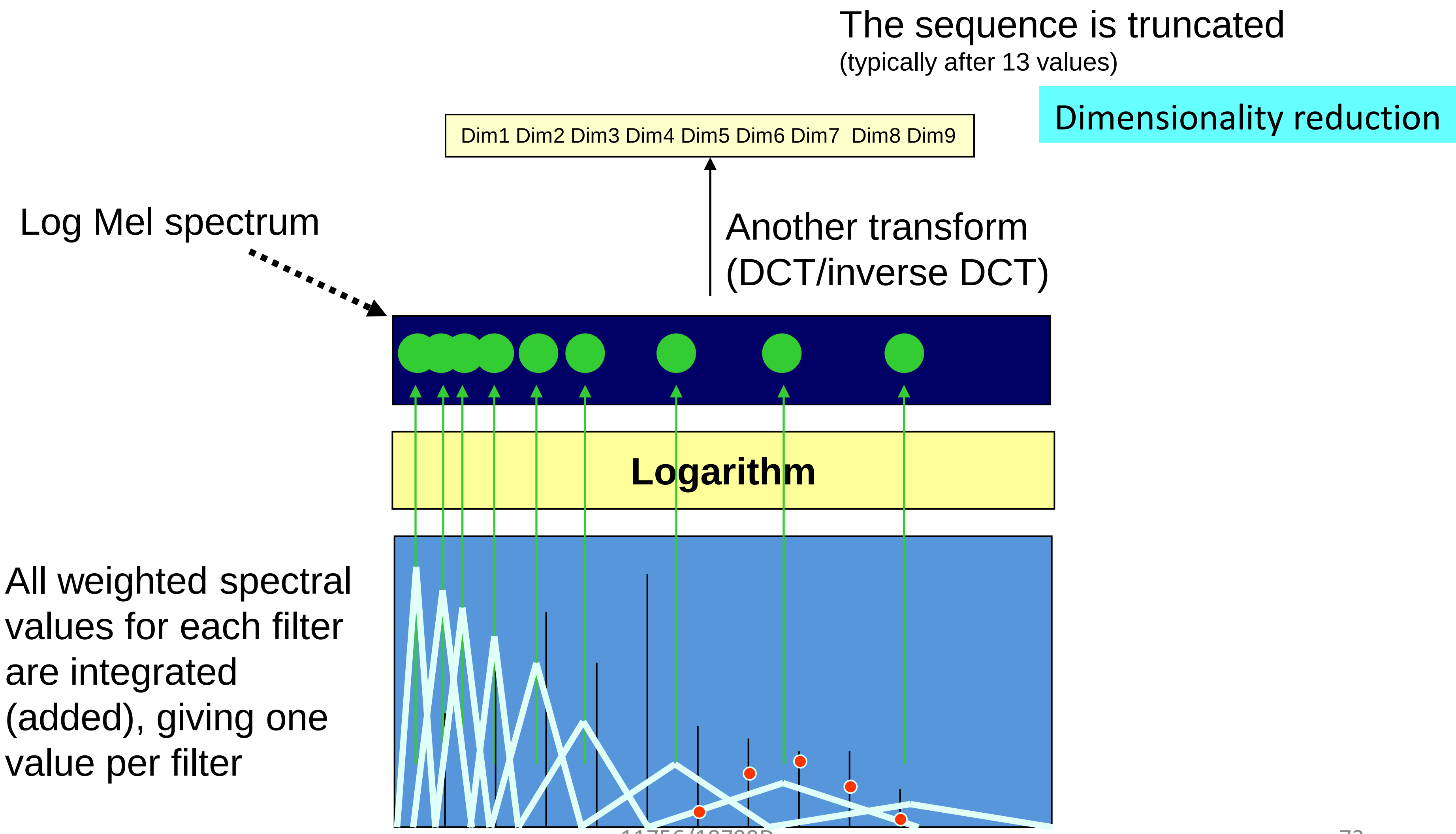
Log Mel spectrum



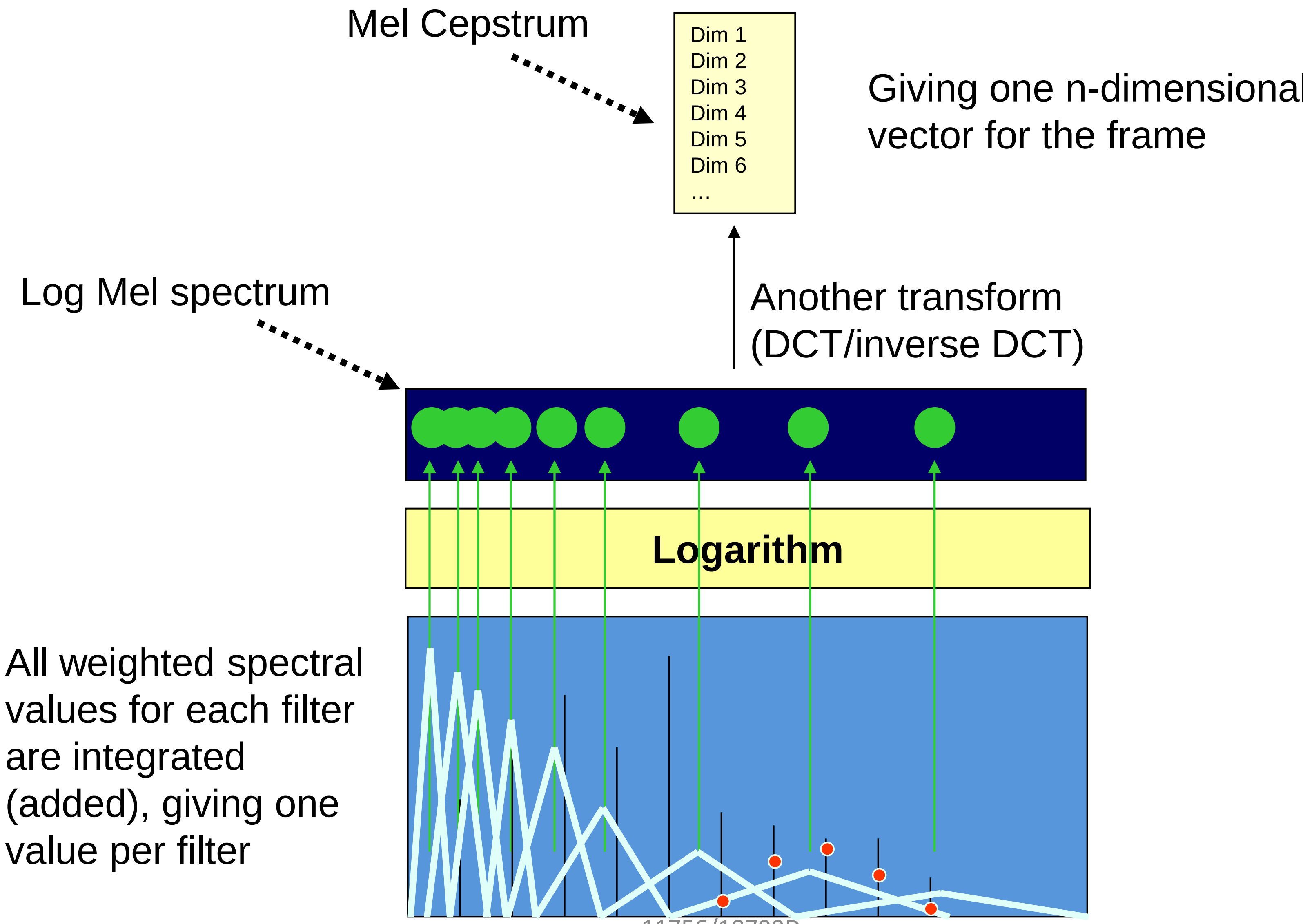
Compress Values

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

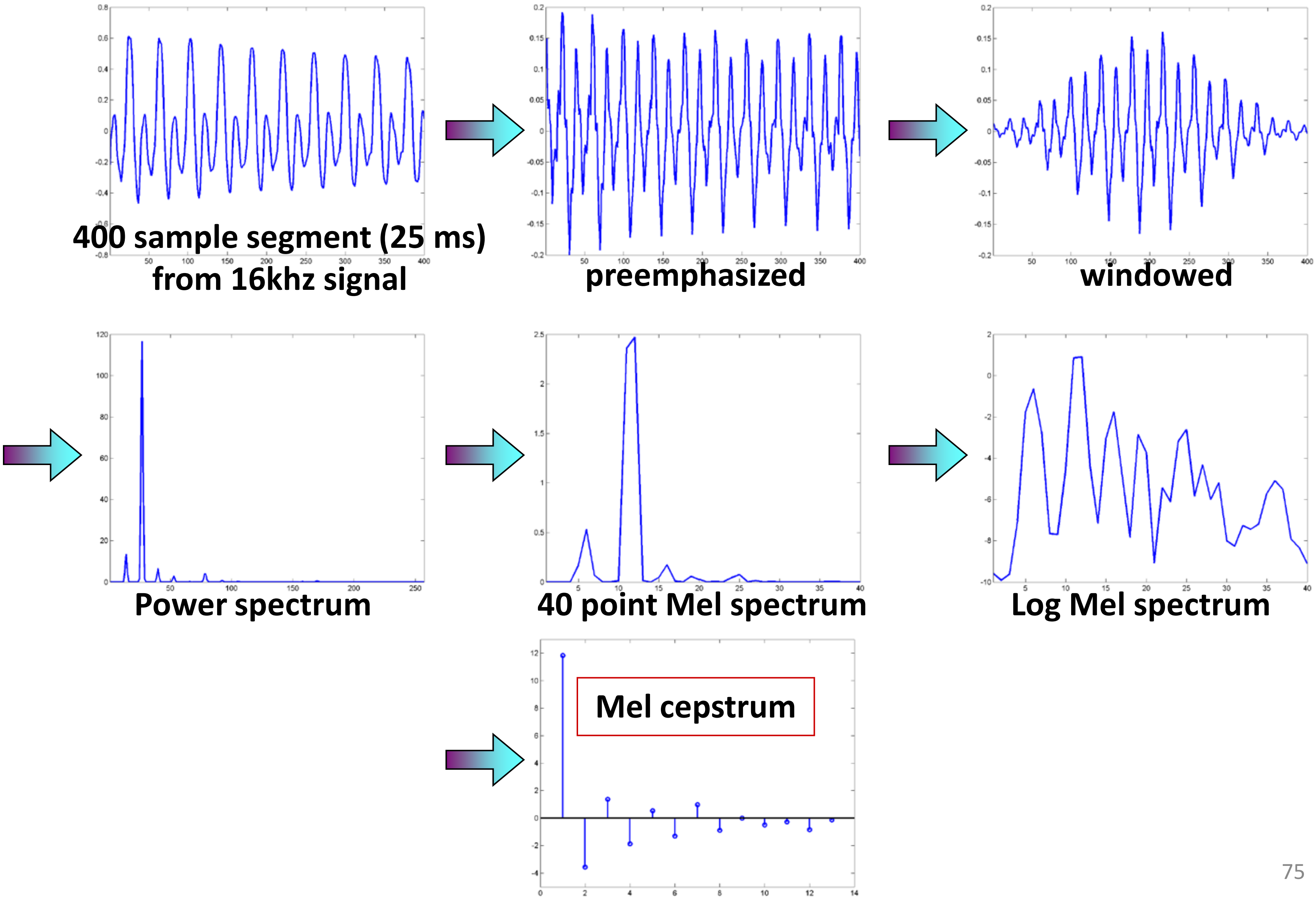
The process of parametrization



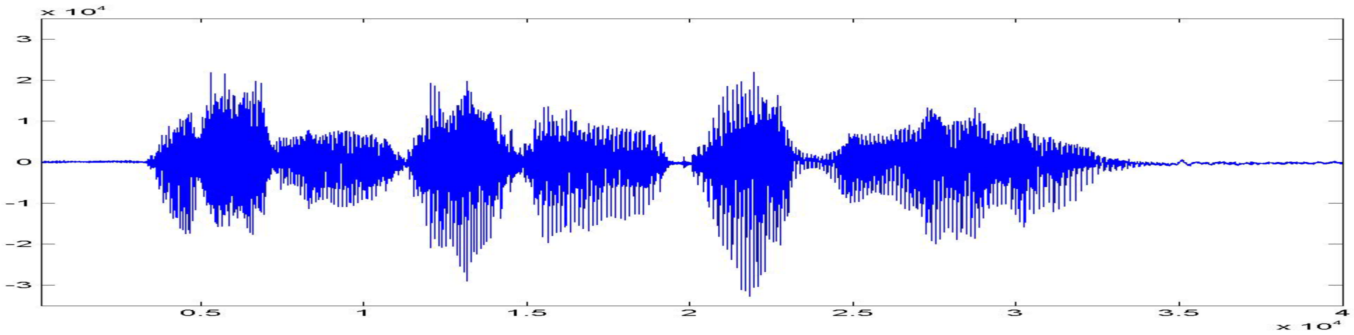
The process of parametrization

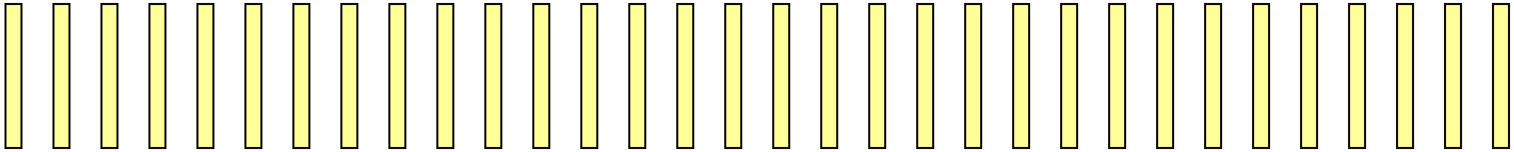


An example segment



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

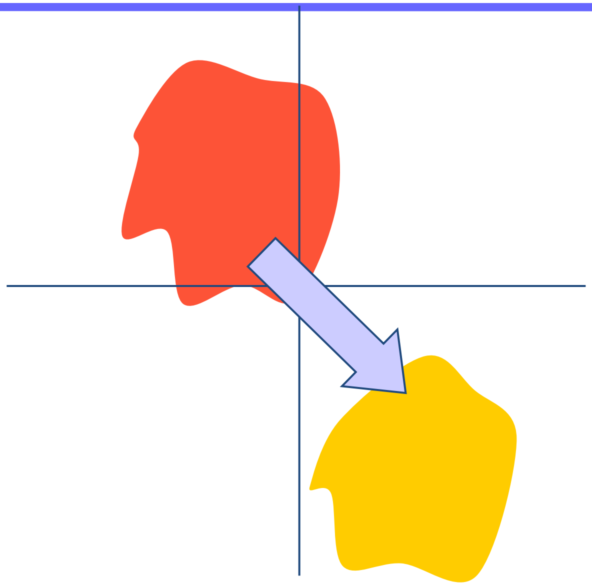
- Perceptual Linear Prediction (PLP) features:
  - ERB filters instead of MEL filters
  - Cube-root compression instead of Log
  - Linear-prediction spectrum instead of Fourier Spectrum

- Auditory features
  - Detailed and painful models of various components of the human ear

Cepstral Variations from Filtering and Noise

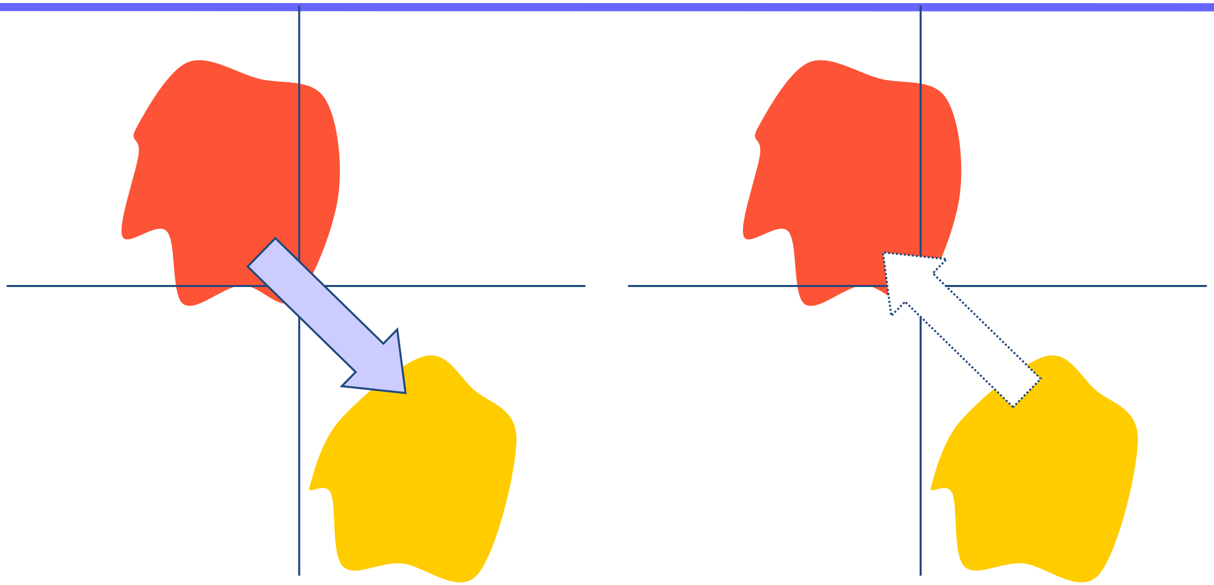
- Microphone characteristics modify the spectral characteristics of the captured signal
  - They change the value of the cepstra
- Noise too modifies spectral characteristics
- As do speaker variations
- All of these change the distribution of the cepstra

Effect of Speaker Variations, Microphone Variations, Noise etc.



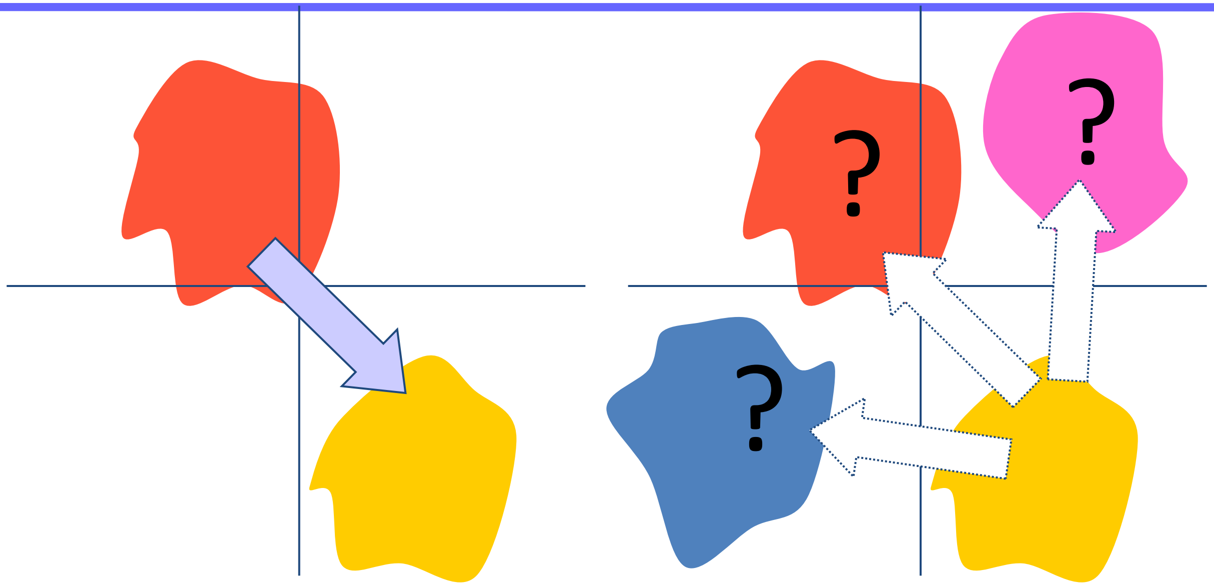
- Noise, channel and speaker variations change the *distribution* of cepstral values

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.

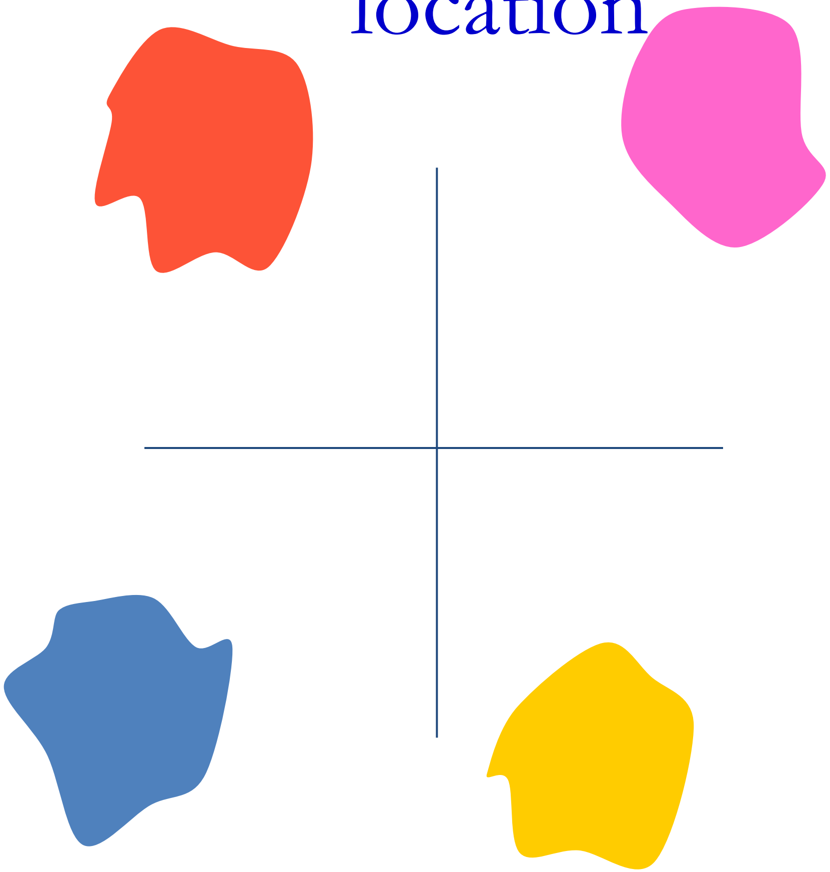


- Noise, channel and speaker variations change the *distribution* of cepstral values

- To compensate for these, we would like to undo these changes to the distribution

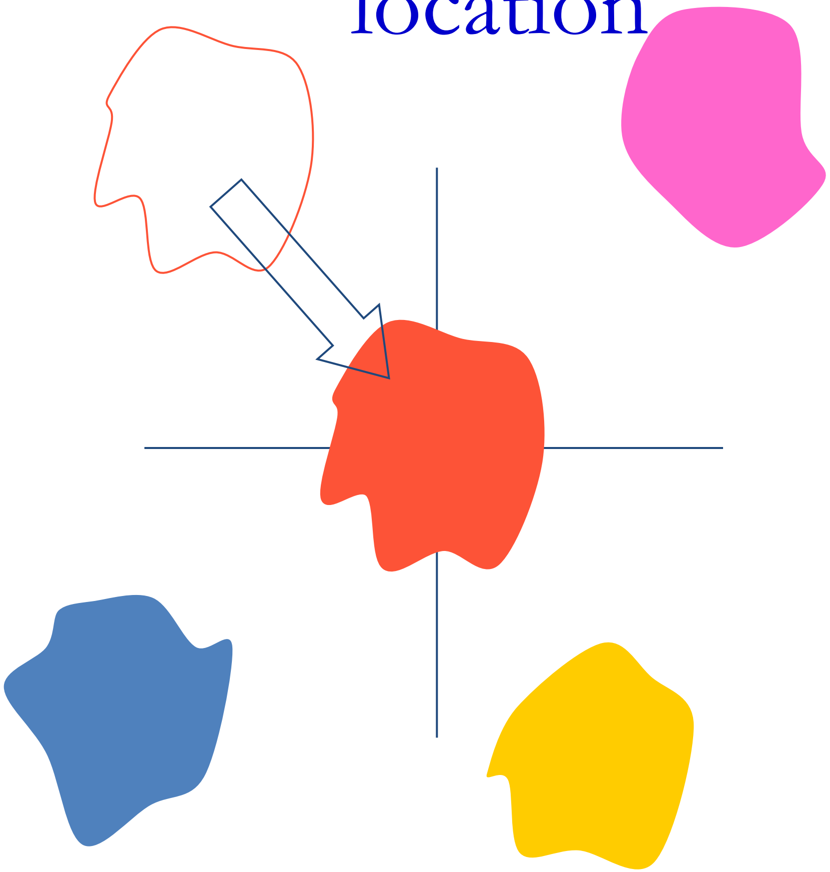
- Unfortunately, the precise position of the distributions of the "good" speech is hard to know

Solution: Move all distributions to a "standard" location



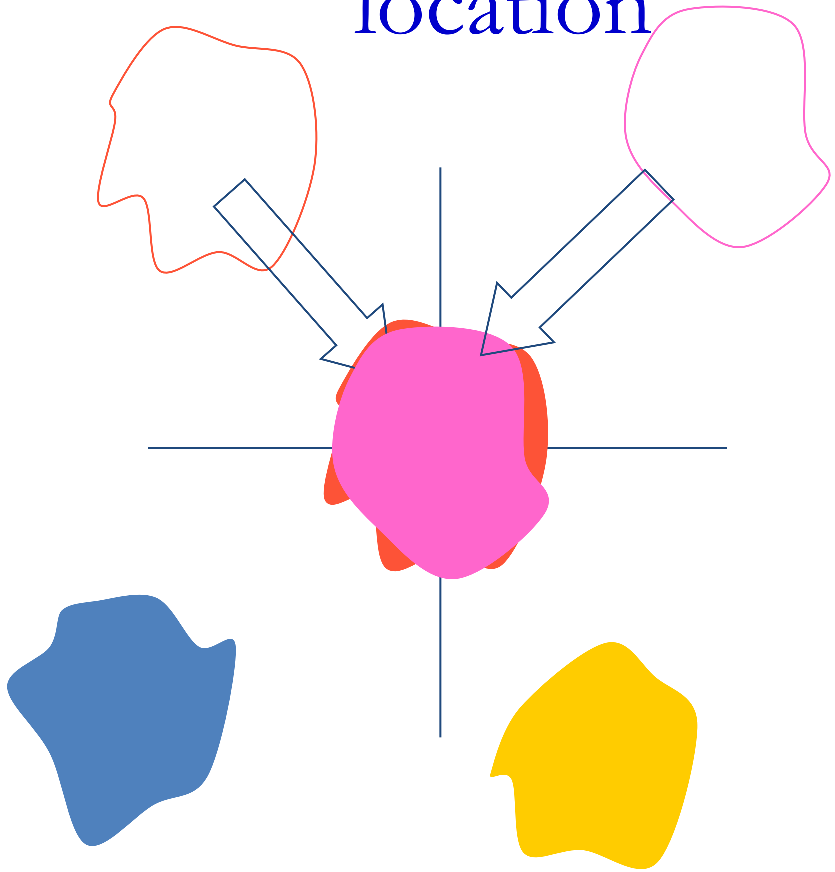
- "Move" all utterances to have a mean of 0
- This ensures that all the data is centered at 0
  - Thereby eliminating *some* of the mismatch

Solution: Move all distributions to a "standard" location



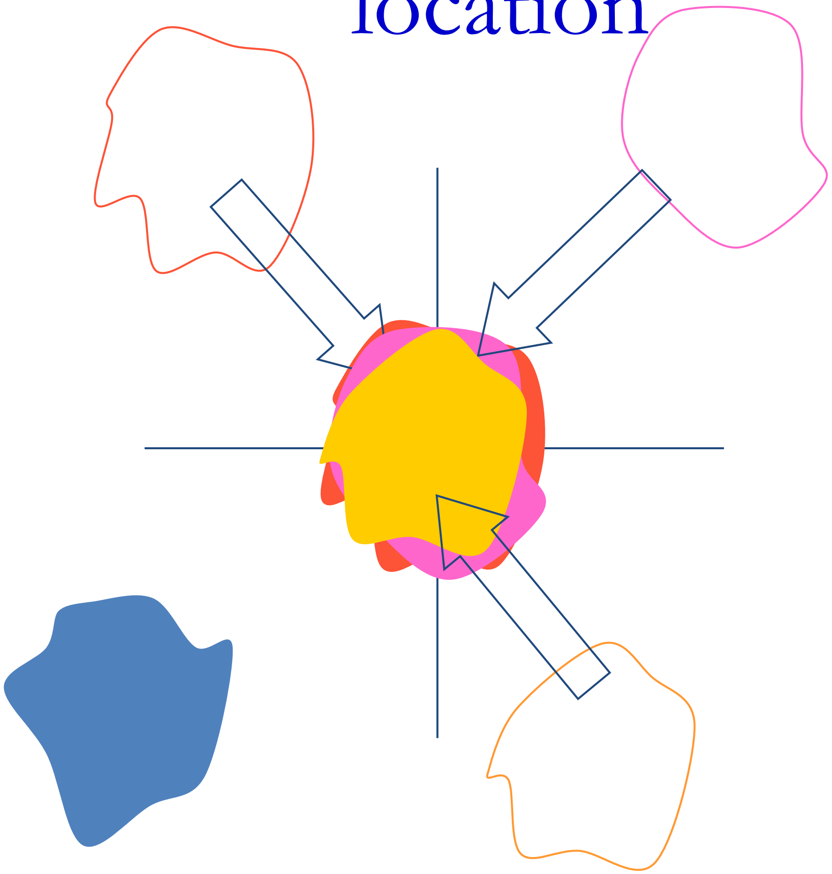
- "Move" all utterances to have a mean of 0
- This ensures that all the data is centered at 0
  - Thereby eliminating *some* of the mismatch

Solution: Move all distributions to a "standard" location



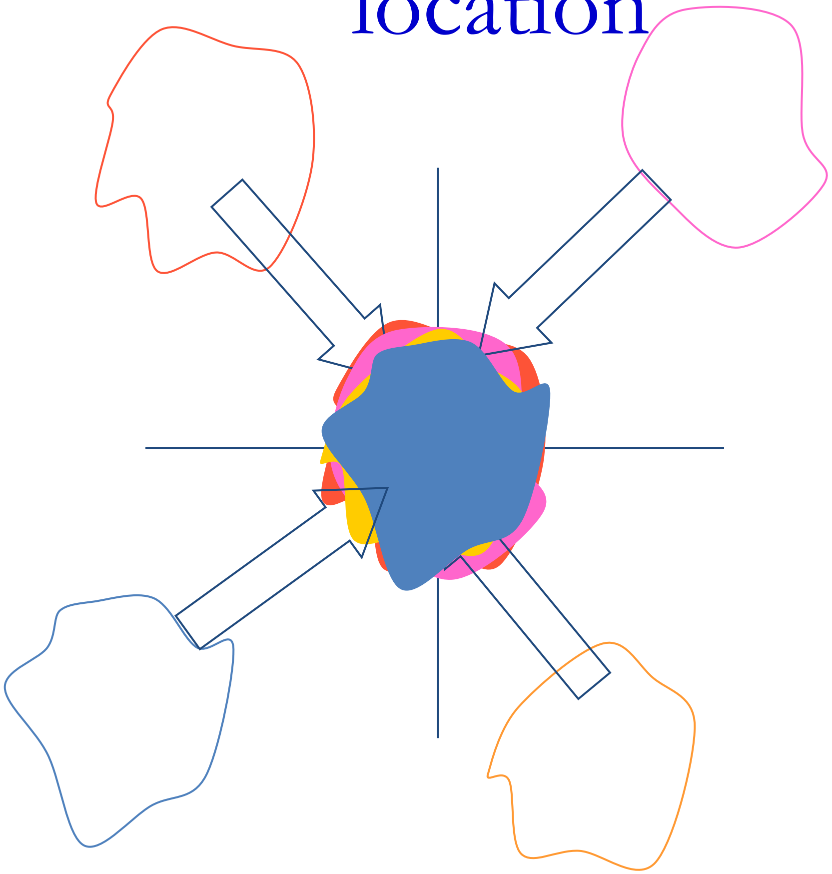
- "Move" all utterances to have a mean of 0
- This ensures that all the data is centered at 0
  - Thereby eliminating *some* of the mismatch

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

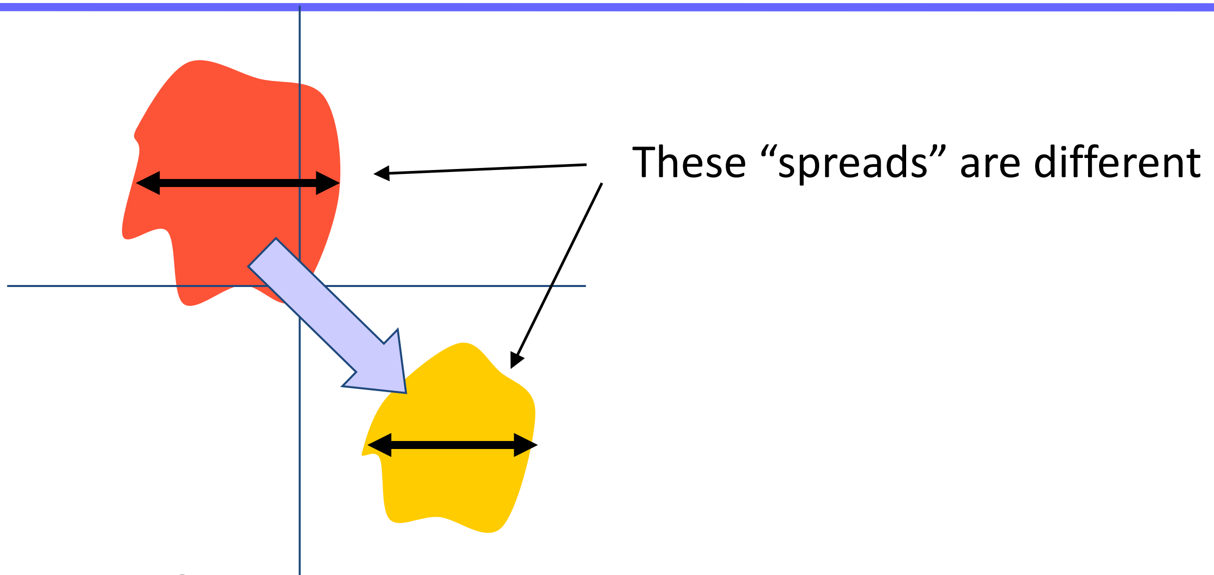
- For each utterance encountered (both in "training" and in "testing")
- Compute the mean of all cepstral vectors

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

- 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

$$sd_{recording} = \sqrt{\frac{1}{Nframes} \sum_t c^2_{normalized}(t)}$$

- Divide all mean-normalized cepstra by this standard deviation

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

- 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

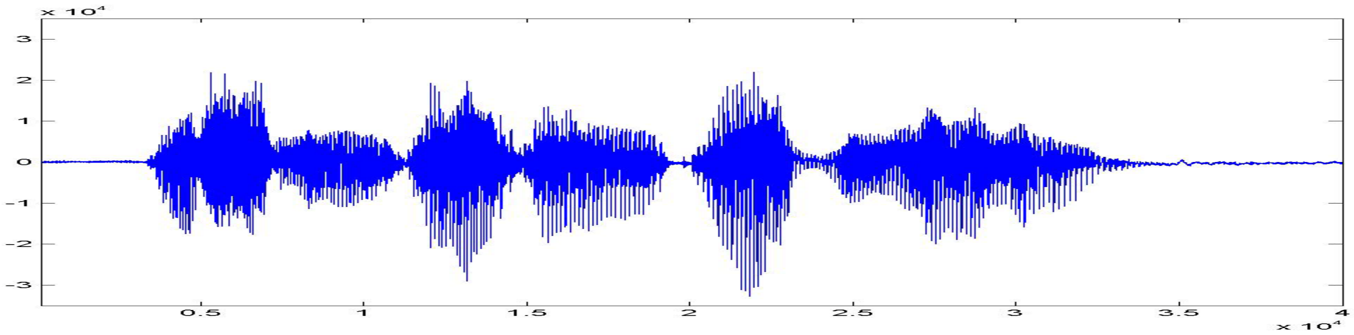
- The cepstral vectors capture instantaneous information only
  - Or, more precisely, current spectral structure within the analysis window
- Phoneme identity resides not just in the snapshot information, but also in the temporal structure
  - Manner in which these values change with time
  - Most characteristic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

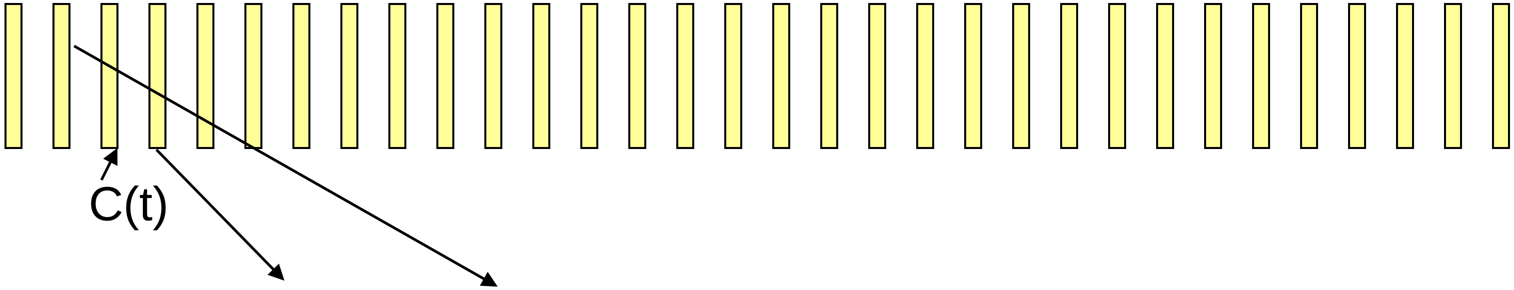
Velocity Features

- 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





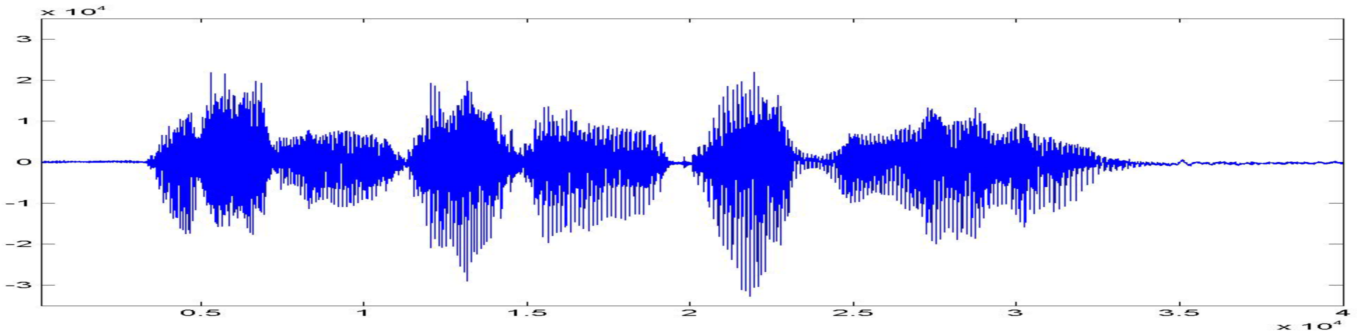
$$\Delta c(t)=c(t+\tau)-c(t-\tau)$$

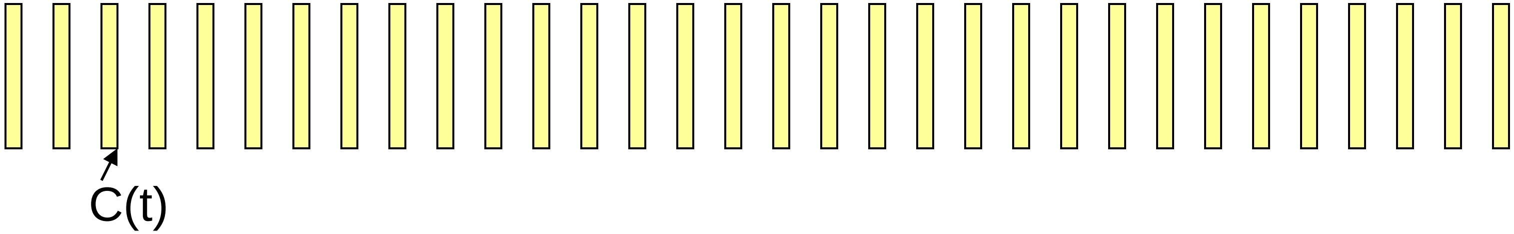
Representing Acceleration

- 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

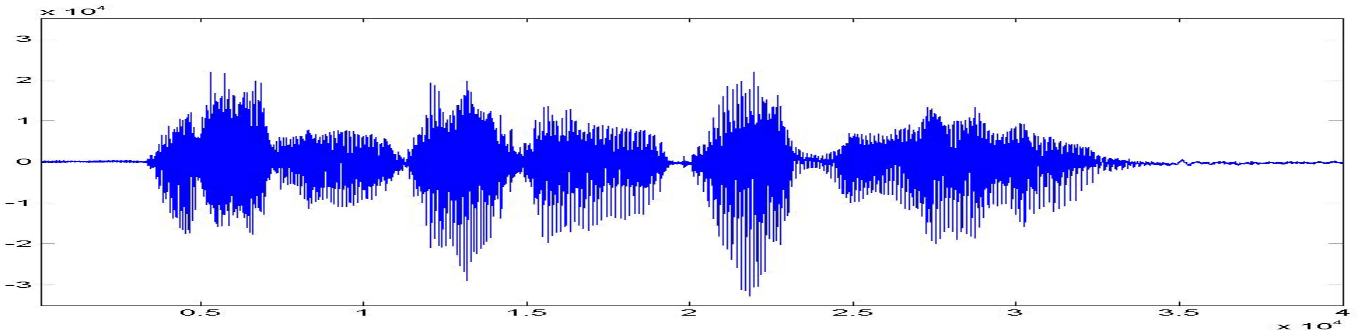


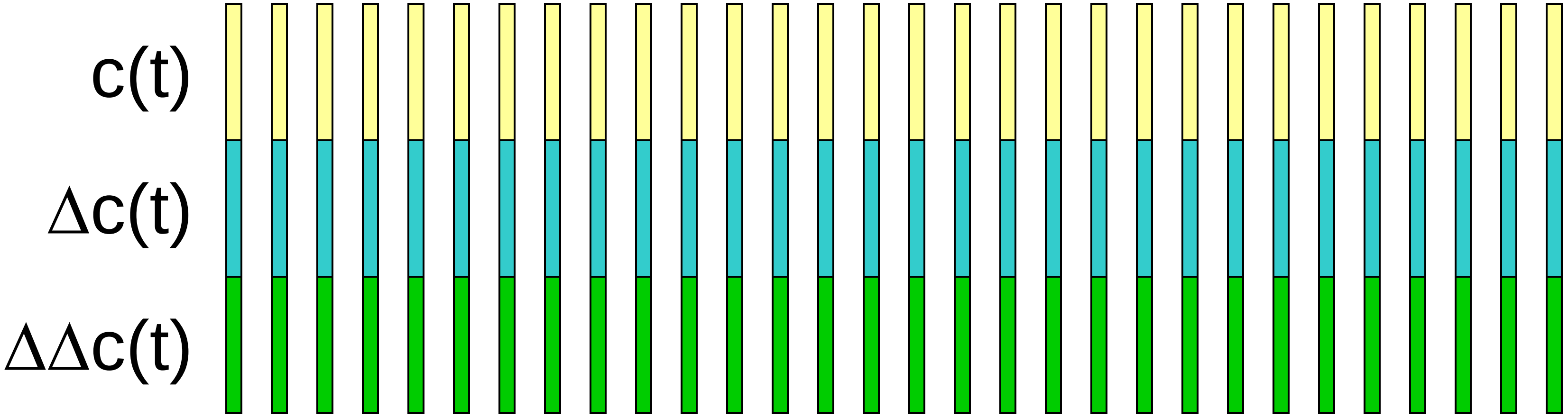


$$\Delta c(t)=c(t+\tau)-c(t-\tau)$$

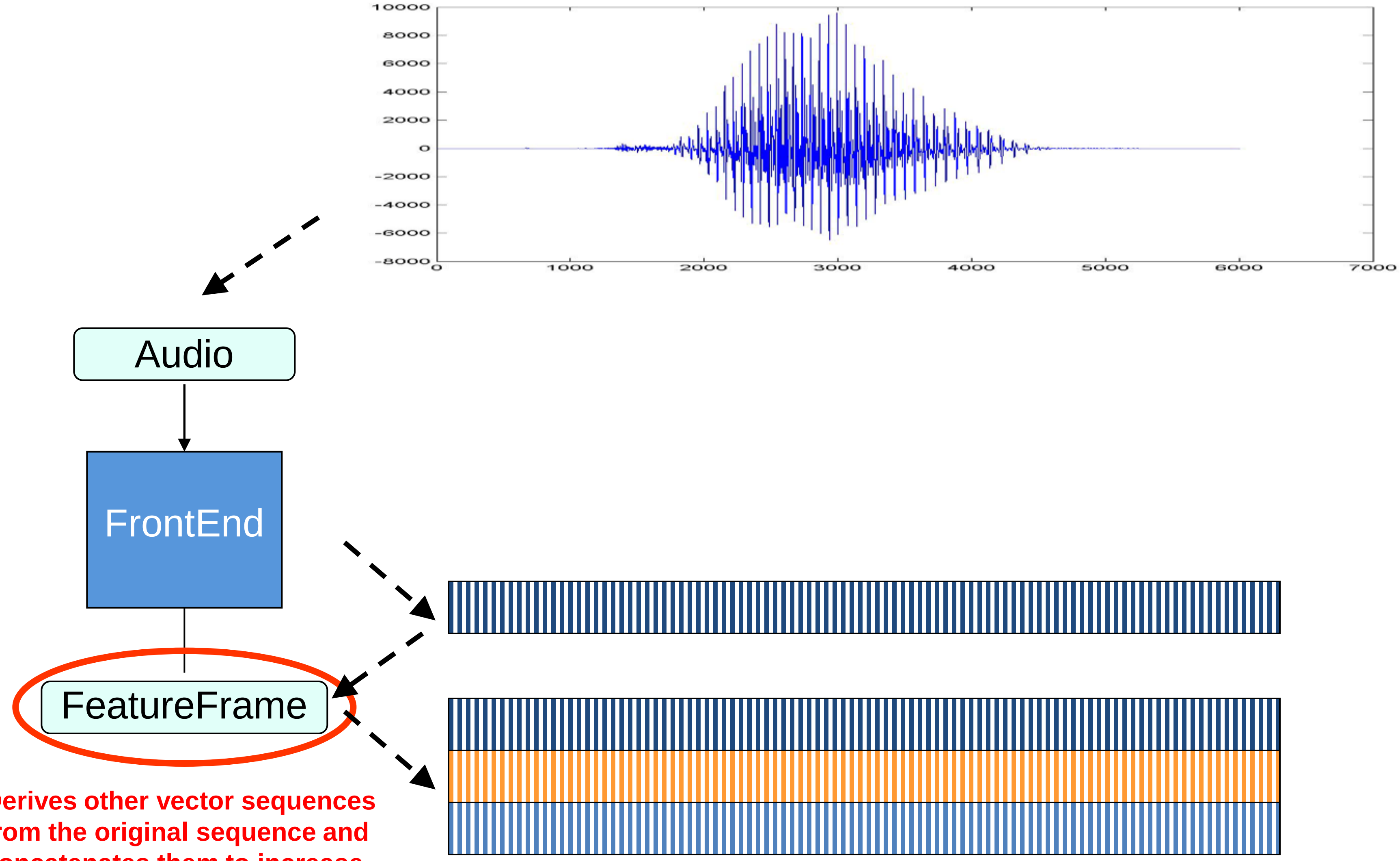
$$\Delta\Delta c(t)=\Delta c(t+\tau)-\Delta c(t-\tau)$$

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**

Other Operations

- Vocal Tract Length 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
  - VTLN attempts to reduce variations due to vocal tract length
- Denoising
  - Attempt to reduce the effects of noise on the featrues
- Discriminative feature projections
  - Additional projection operations to enhance separation between features obtained from signals representing different sounds

wav2feat : 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 : 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 : 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)

Project 1b

- Write a routine for computing MFCC from audio
- Record multiple instances of digits
  - Zero, One, Two etc.
  - 16Khz sampling, 16 bit PCM
  - Compute log spectra and cepstra
    - No. of features = 13 for cepstra
  - Visualize both spectrographically (easy using matlab)
    - 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
- Allowed to use wav2feat or code from web
  - Dan Ellis has some nice code on his page
  - Must be integrated with audio capture routine
    - Assuming kbhit for start. Stop of recording via automatic endpointing.