Feature Computation: Representing the Speech Signal

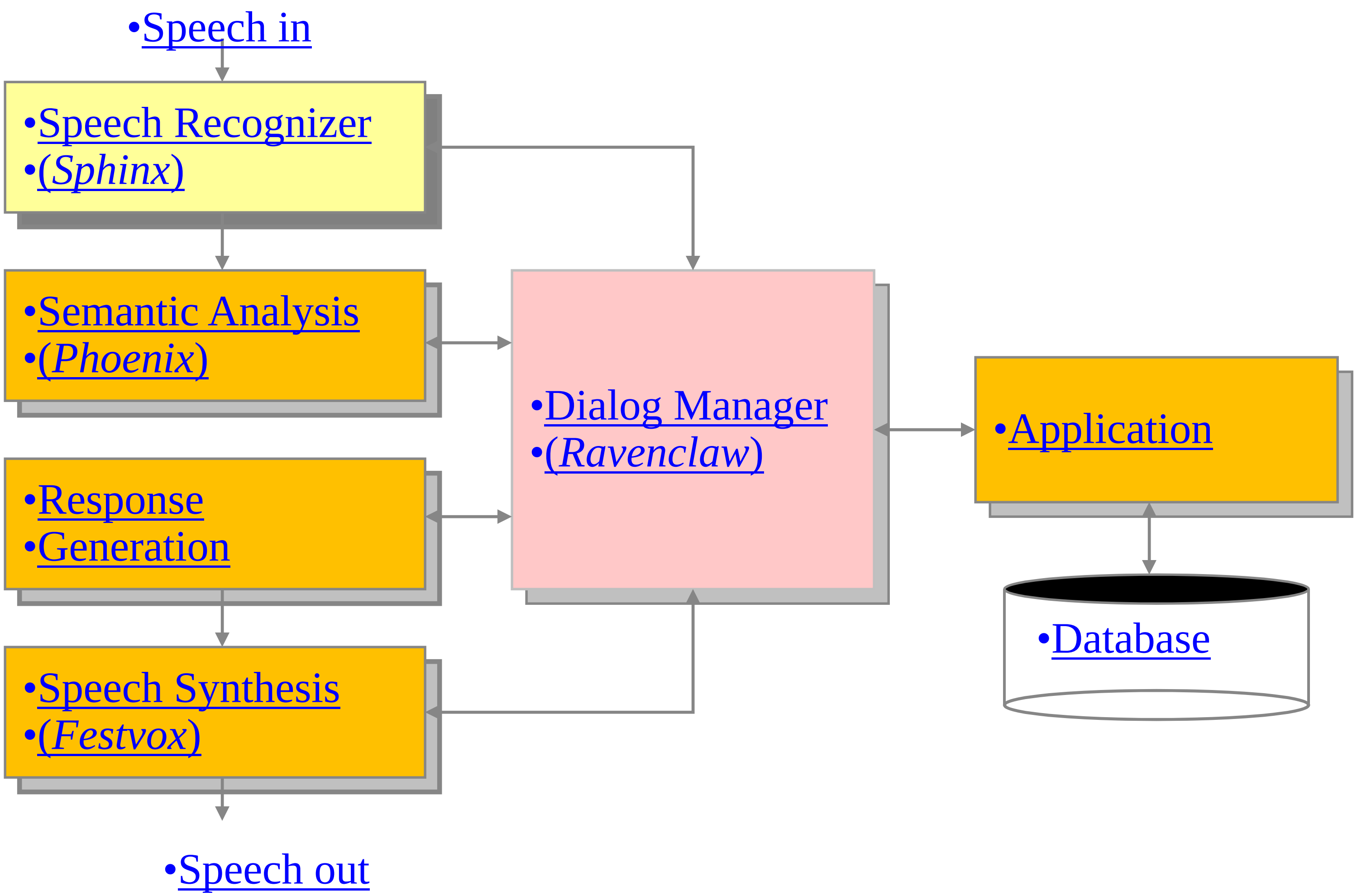
Bhiksha Raj

Administrivia

- Blackboard not functioning properly
  - Must manually add missing students
- Notes for class on course page:
  - http://asr.cs.cmu.edu/
- Groups not yet formed
  - Only 3 teams so far (two are singletons)
  - Will post randomly formed teams tonight
- Classroom: Wait for posting, may change

Speech Technology

- Covers many sub-areas, not just speech recognition
- Typical application based on speech technology:



Some Milestones in Speech Recognition

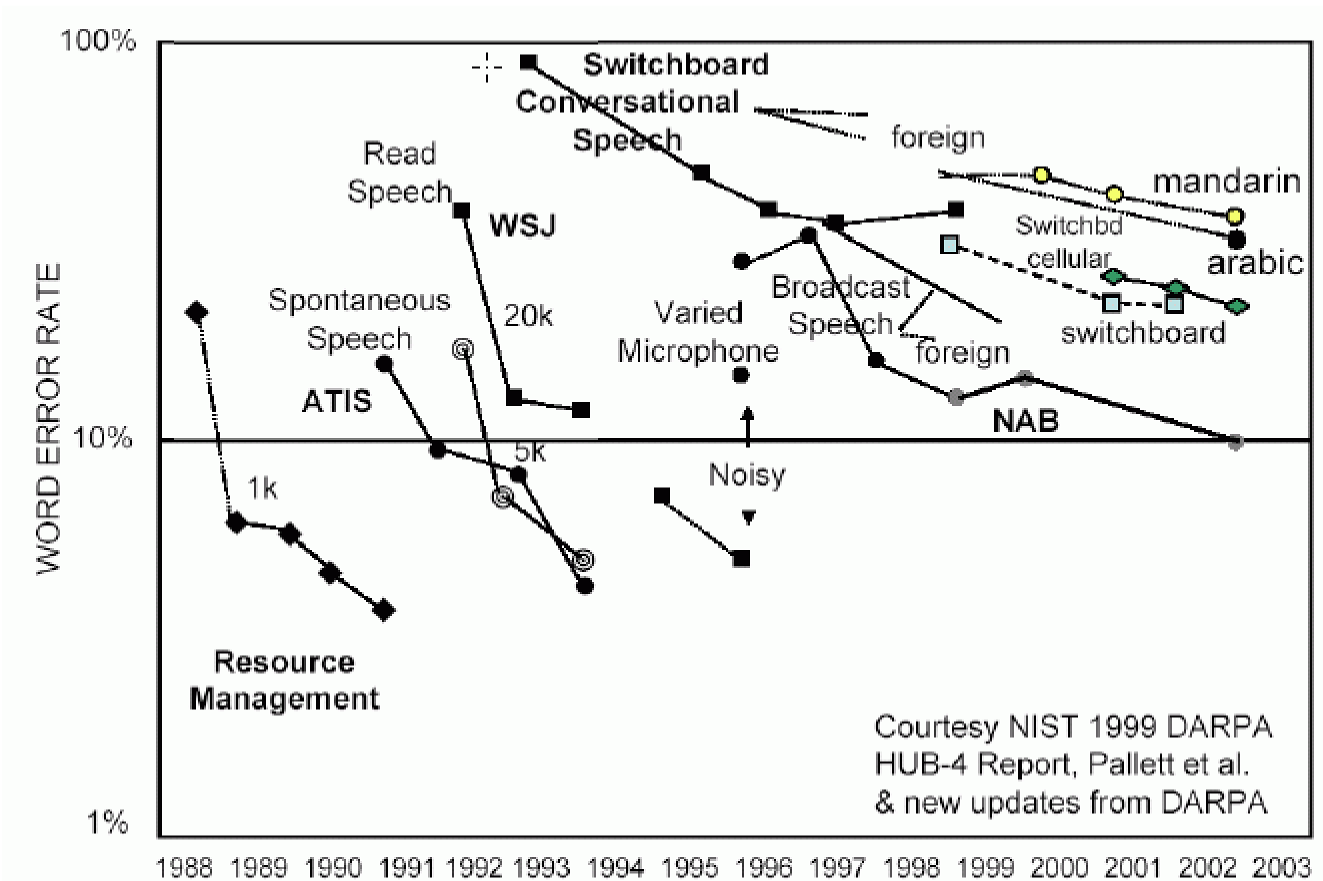
- 1968? – Vintsyuk proposes dynamic time warping algorithm
- 1971 – DARPA starts speech recognition program
- 1975 – Statistical models for speech recognition
  - James Baker at CMU
- 1988 – Speaker-independent continuous speech recognition
  - 1000 word vocabulary; *not* real time!
- 1992 – Large vocabulary dictation from Dragon Systems
  - Speaker dependent, isolated word recognition
- 1993 – Large vocabulary, real-time continuous speech recognition
  - 20k word vocabulary, speaker-independent
- 1995 – Large vocabulary continuous speech recognition
  - 60k word vocabulary at various universities and labs
- 1997? – Continuous speech, real-time dictation
  - 60k word vocabulary, Dragon Systems *Naturally Speaking*, IBM *ViaVoice*
- 1999 – Speech-to-speech translation, multi-lingual systems
- 2004 – Medium/large vocabulary dictation on small devices

Some Reasons for the Rapid Advances

- Improvements in acoustic modeling
  - Hidden Markov models, context-dependent models
  - Speaker adaptation
  - Discriminative models
- Improvements in Language modeling
  - Bigram, trigram, quadgram, structured and higher-order models
- Improvements in recognition algorithms

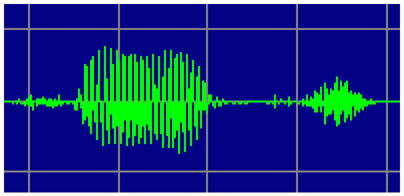
- Availability of more and more training data
  - Less than 10 hours to 10000 hours
  - Brute force
- Last but not least, unprecedented growth in computation and memory
  - MHz to GHz CPUs, MBs to GBs memory
  - Brute force, again

Speech Recognition Performance



•Every time ASR performance reached a respectable level, the focus shifted to a more difficult problem, broadening the research horizons

The Speech Recognition Problem



•Speech Recognizer
•or "Decoder"

•"speech"

- Speech recognition is a type of pattern recognition problem
  - Input is a stream of sampled and digitized speech data
  - Desired output is the sequence of words that were spoken
- If we know the signal patterns that represent every spoken word beforehand, we could try to identify the words whose patterns best match the input
- Problem: word patterns are never reproducible exactly
  - How do we represent these signal patterns?
  - Given this uncertainty, how do we compare the input to known patterns?
- Speech recognition is the study of these problems

Why is Speech Recognition Hard?

- Tremendous range of variability in speech, even though the message may be constant:
  - Human physiology: squeaky voice *vs* deep voice
  - Speaking style: clear, spontaneous, slurred or sloppy
  - Speaking rate: fast or slow speech
    - Speaking rate can change within a single sentence
  - Emotional state: happy, sad, etc.
  - Emphasis: stressed speech *vs.* unstressed speech
  - Accents, dialects, foreign words
  - Environmental or background noise
  - *Even the same person never speaks exactly the same way twice*
- In addition:
  - Large vocabulary and infinite language
  - Absence of word boundary markers in continuous speech
  - Inherent ambiguities: “I scream” or “Ice cream”?

What are the Technological Challenges?

- Representations of spoken words are inexact
  - We just saw the reasons for variations in speech
  - Even the same person never says a given sentence exactly the same way twice
    - Let alone two *different* people
  - No representation can capture the infinite range of variations
  - Yet, humans have apparently no difficulty
    - They *adapt* to new situations effortlessly
  - The problem is understanding and representing what is *invariant*

- Pattern matching is necessarily inexact
  - Given the above, there will always be mismatches in pattern matching, and hence *misrecognitions*
    - Even humans are not perfect
  - Finding *optimal* pattern matching algorithms, and hence minimizing misrecognitions, is another challenge

The Technological Challenges (contd.)

- As target vocabulary size increases, complexity increases
  - Computational resource requirements increase
    - Memory size to store patterns
    - Computational cost of matching
  - Most important, the degree of *confusability* between words increases
    - More and more words begin sounding alike
    - Requires finer and finer models (patterns)
    - Further aggravates the computational cost problem

The Quest in Speech Recognition

- Speech recognition is all about:
  - Turning a seemingly hard problem into a precise mathematical form
  - Finding solutions and algorithms that are:
    - *Elegant*; leads to efficiency and generality
    - *Optimal*, as opposed to *ad hoc* techniques without well defined properties of recognition accuracy
    - *Efficient*, that can be used in real-life applications

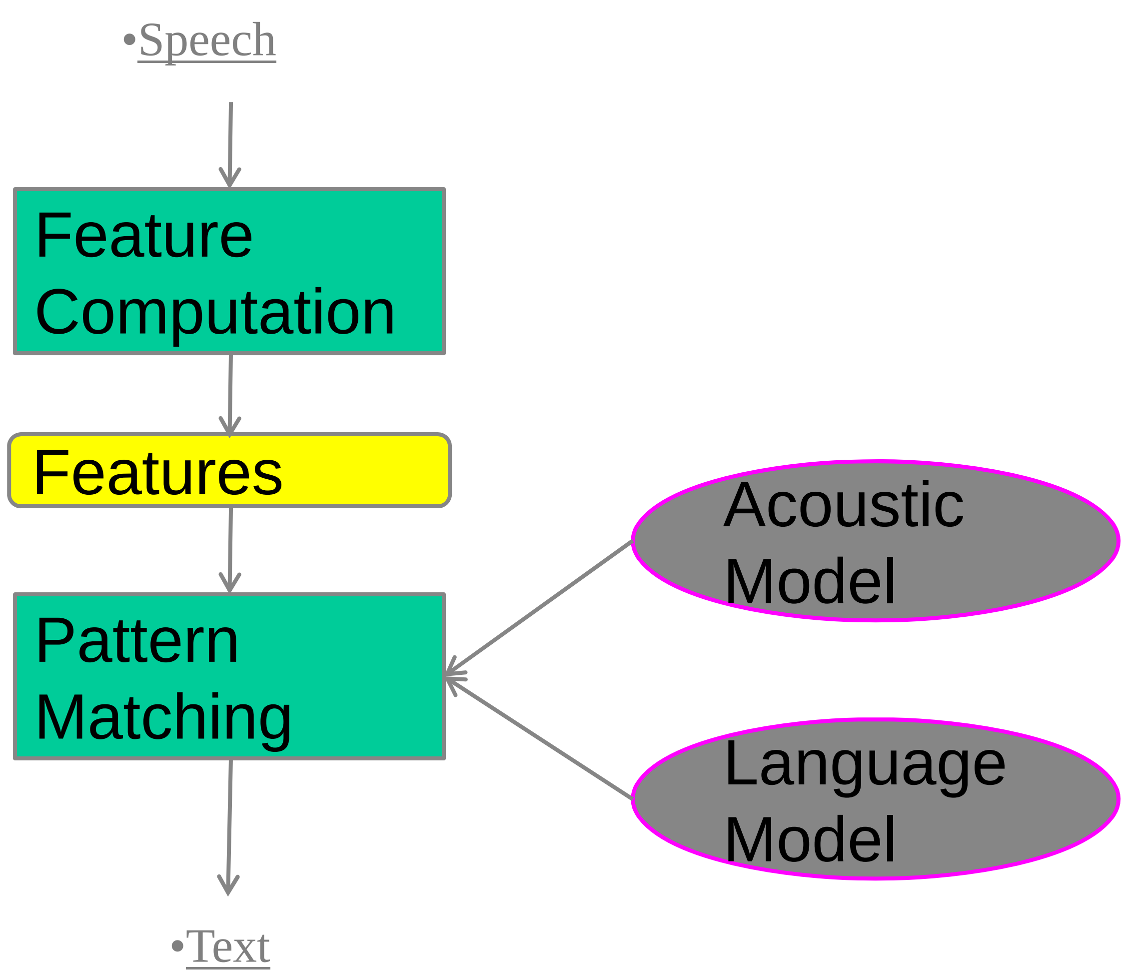
- However,
  - Not all problems are solved
    - *E.g.* Natural free-form language.
  - Moreover, some problems seem *inherently* hard
    - How do we represent "meaning"?
  - Speech recognition has its share of *ad hoc* approaches to many problems, which still need to be addressed

Disciplines in Speech Technology

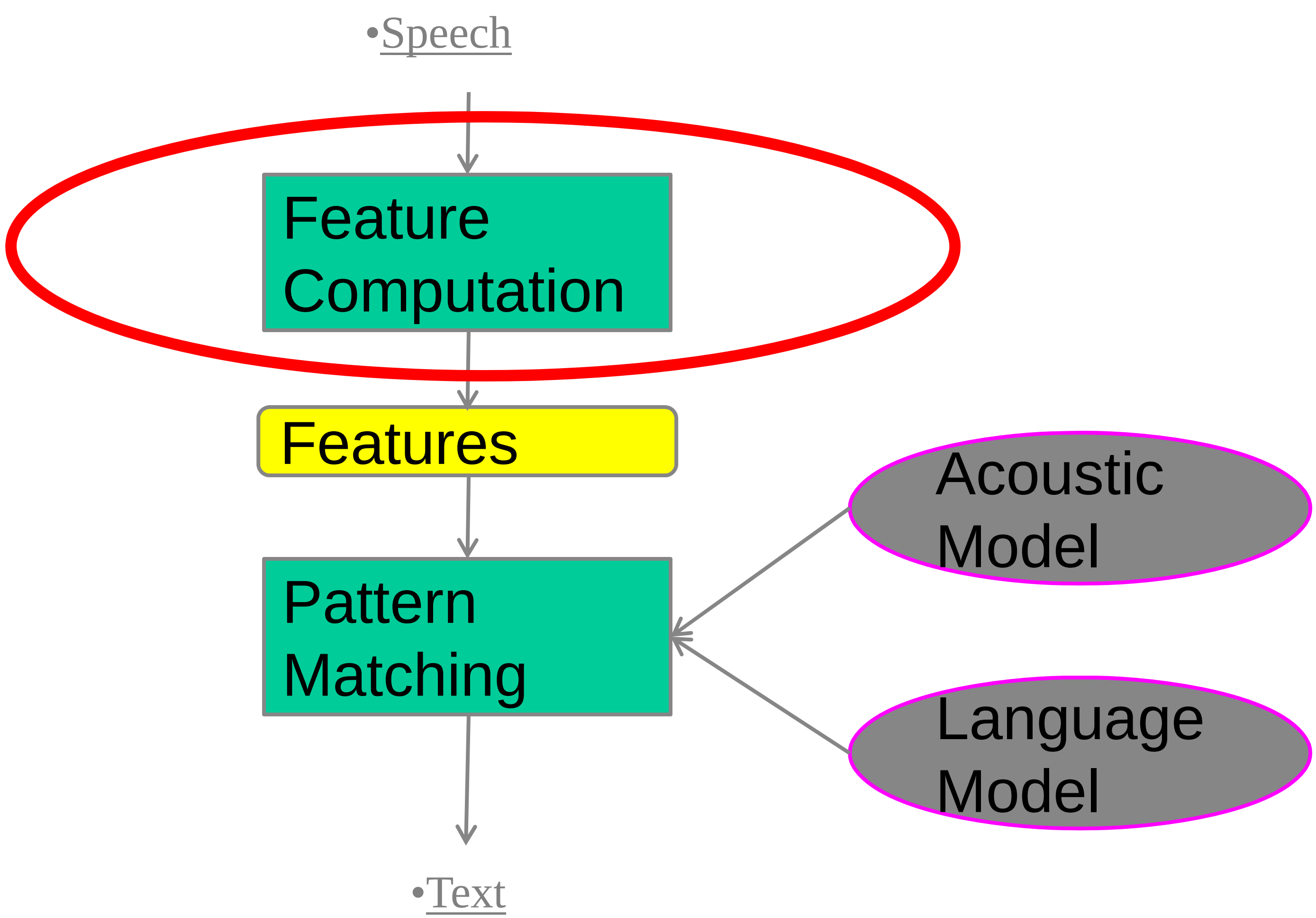
- Modern speech technology is combination of many disciplines
  - Physiology of speech production and hearing
  - Signal processing
  - Linear algebra
  - Probability theory
  - Statistical estimation and modeling
  - Information theory
  - Linguistics
  - Syntax and semantics
  - Computer science
    - Search algorithms
    - Machine learning
    - Computational complexity
    - Computer hardware

- Surprisingly complex task, for something humans do so easily

The Flow of a Speech Recognizer



ASR Modules



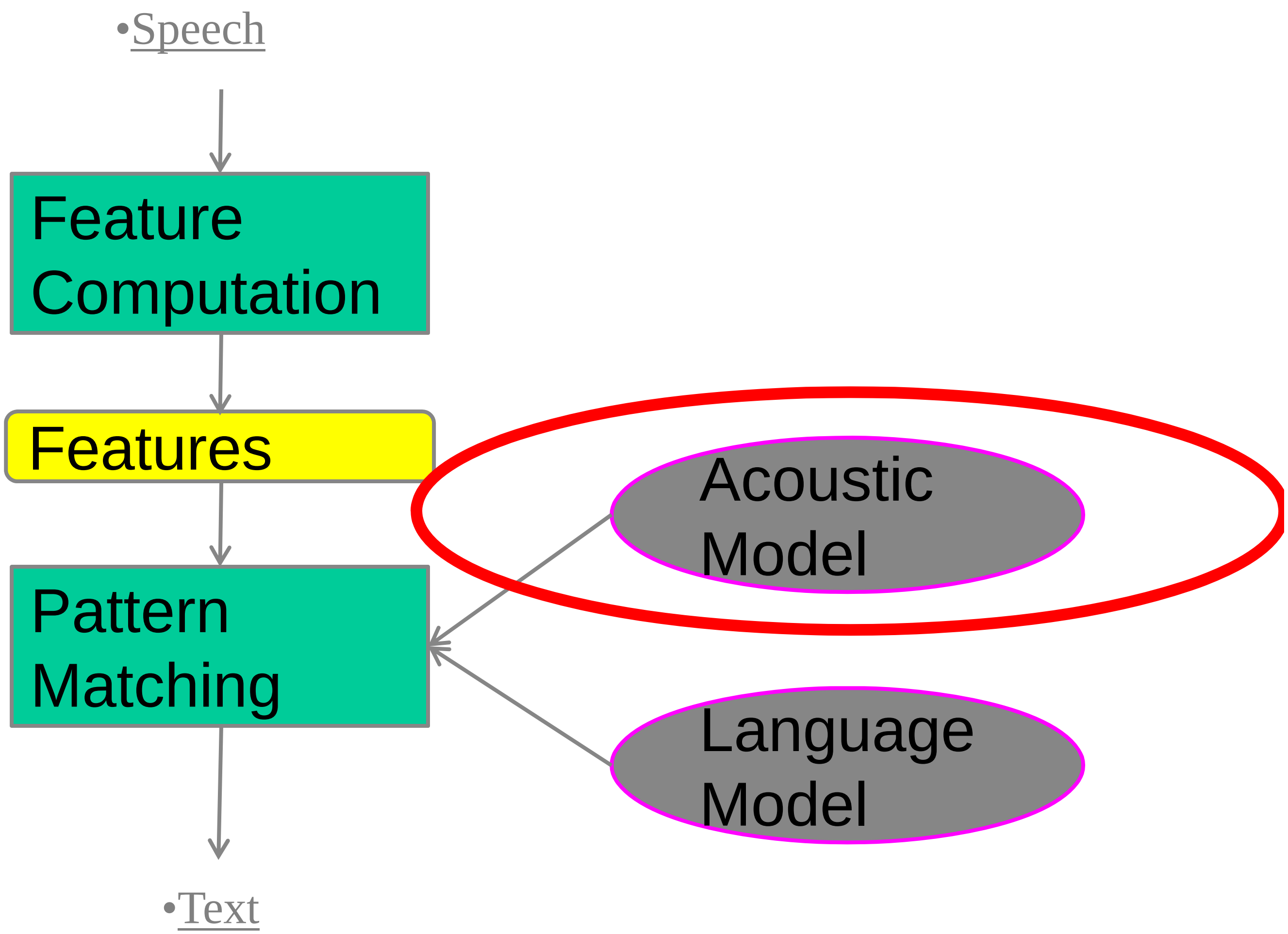
•Speech
Feature Computation
Features
Pattern Matching
Acoustic Model
Language Model
•Text

Front End

- The “Feature Computation” module is also often called the “Front End”.

- The raw speech signal is inappropriate for recognition
- *Features* must be computed from it
- The front end computes these features

ASR Components



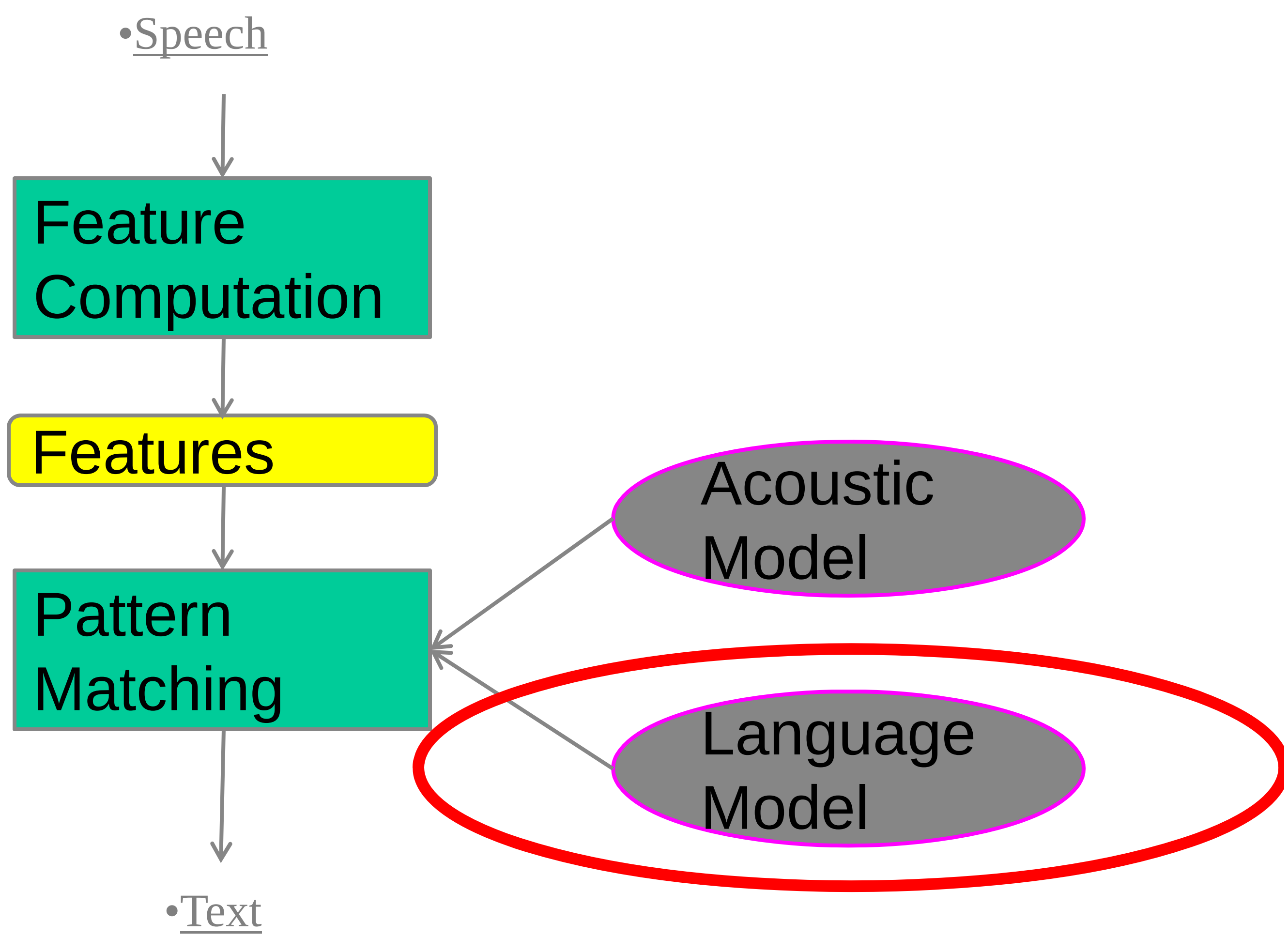
Carnegie Mellon

The Acoustic Model

- The Acoustic Model stores the statistical characteristics of different words/phonemes/sound units

- Typically as HMMs

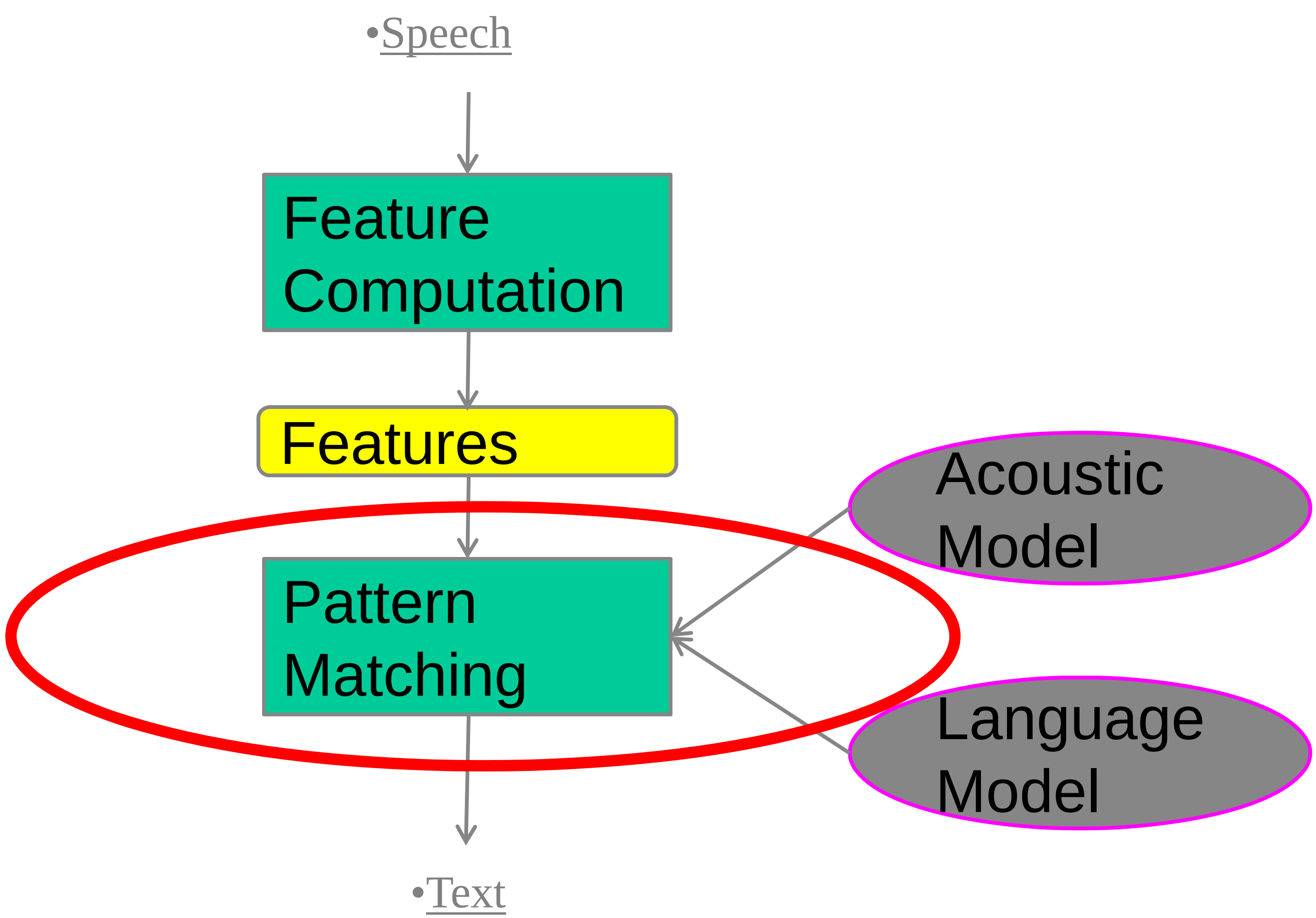
ASR Components



The Language Model

- What do we permit people to speak?
  - Isolated words
  - Restricted Grammars
  - Unrestricted language
- How do we model the language in each case
  - Finite-state / context-free grammars
  - N-gram language models
    - Combinations of the above
  - Class-based models
  - Application/Context-sensitive models
  - Whole sentence models

ASR Modules



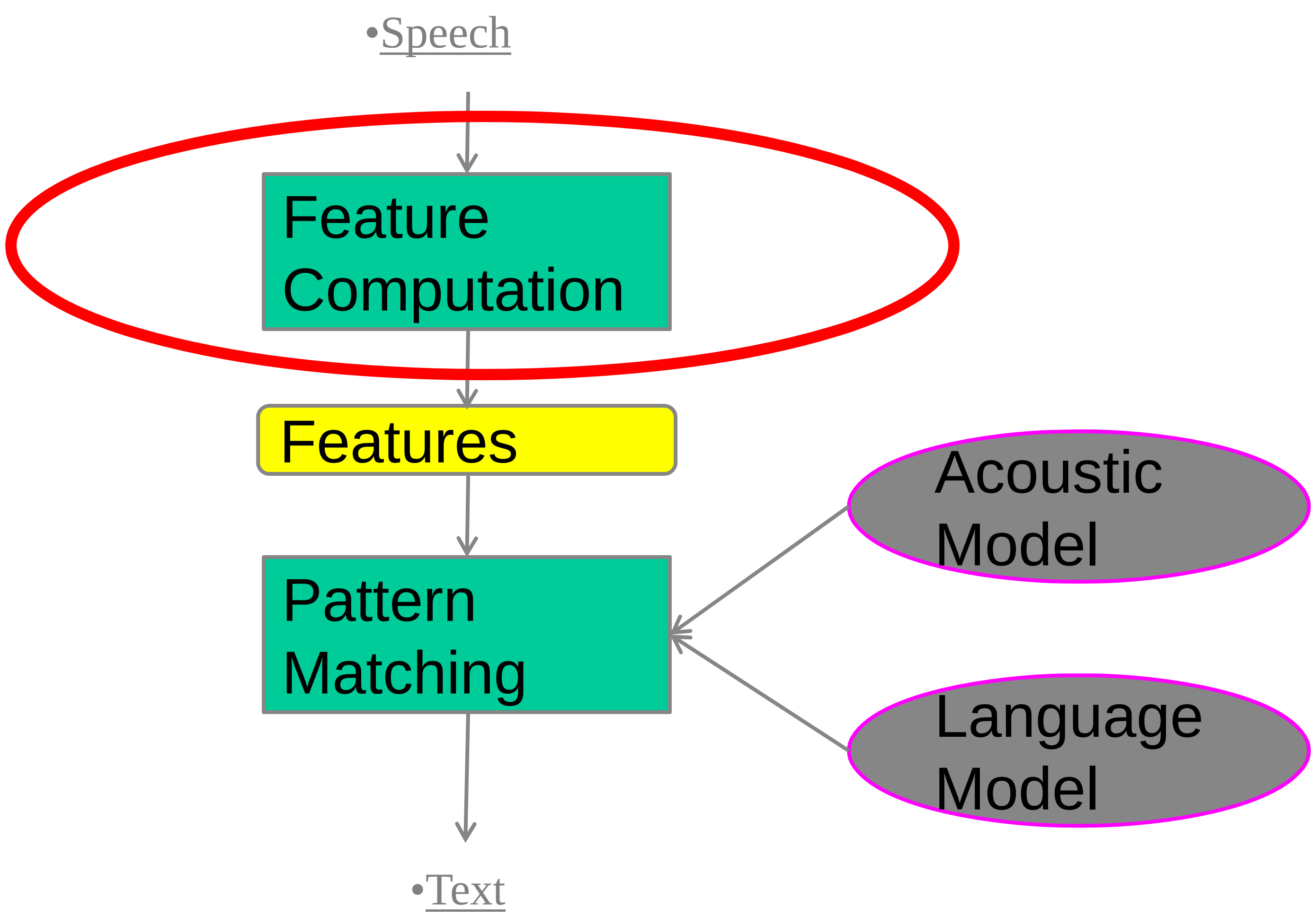
•Speech
Feature Computation
Features
Acoustic Model
Pattern Matching
Language Model
•Text

Pattern Matching

- Combines Acoustic and Language models to evaluate features from incoming speech

- Needs efficient representations of the language model
  - Lextrees
  - Flat structures
  - Approximations
  - Push-down automata / Finite-state networks
  - Weighted finite-state transducers
- Needs efficient search strategies
  - Viterbi search
  - Stack/A* searches
  - Other types

ASR Modules

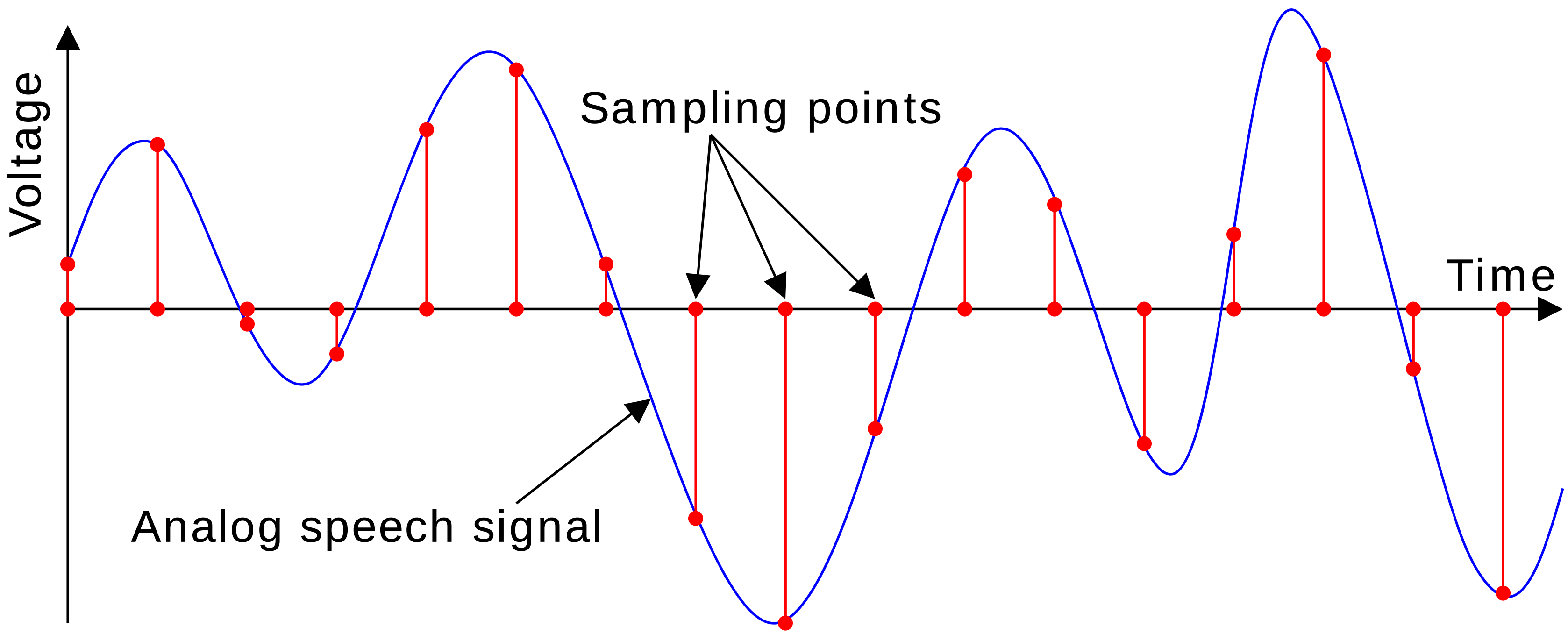


22

A crash course in signal processing

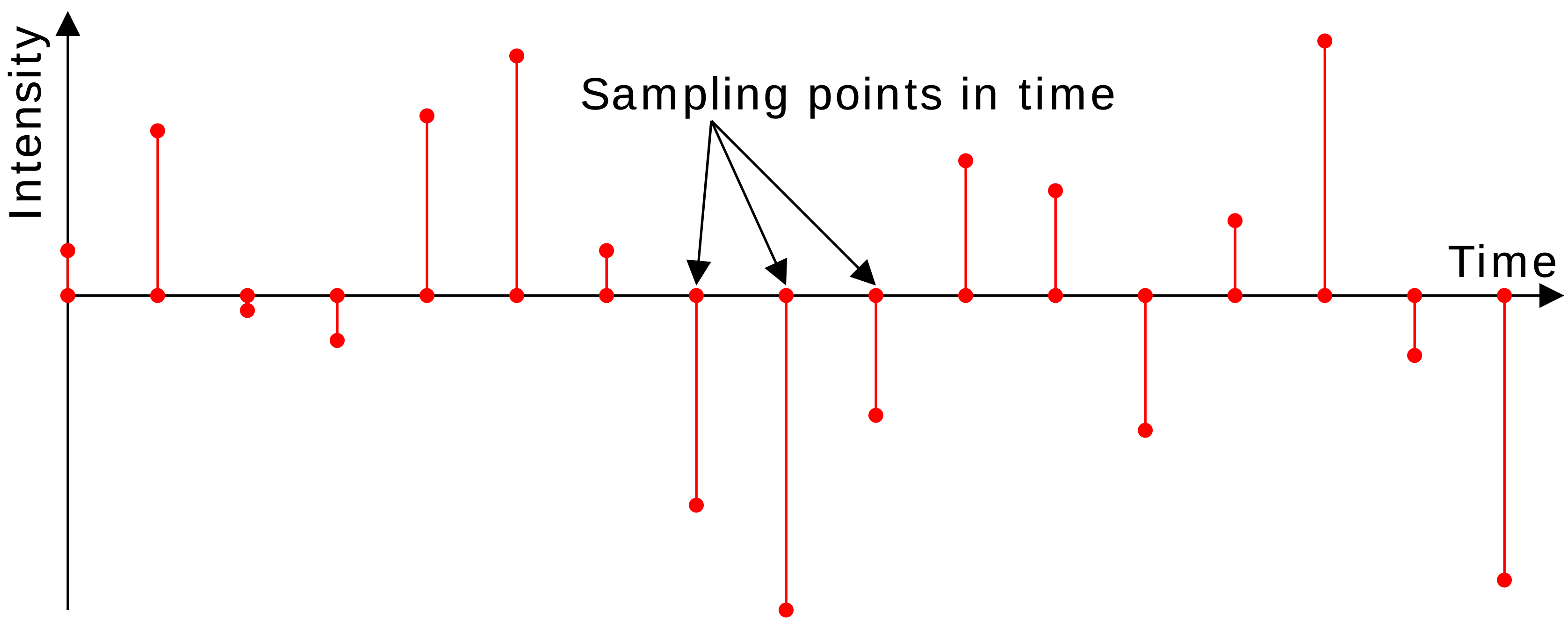
The Speech Signal: Sampling

- 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

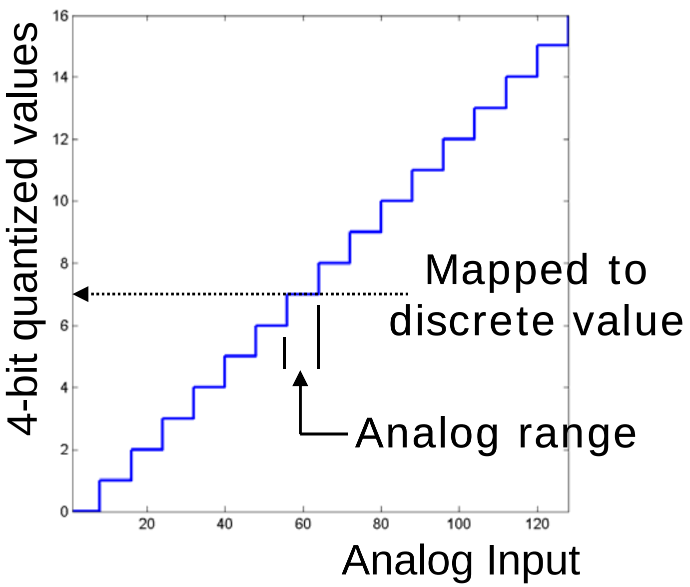
- Digitization can be *linear* (uniform) or *non-linear* (non-uniform)

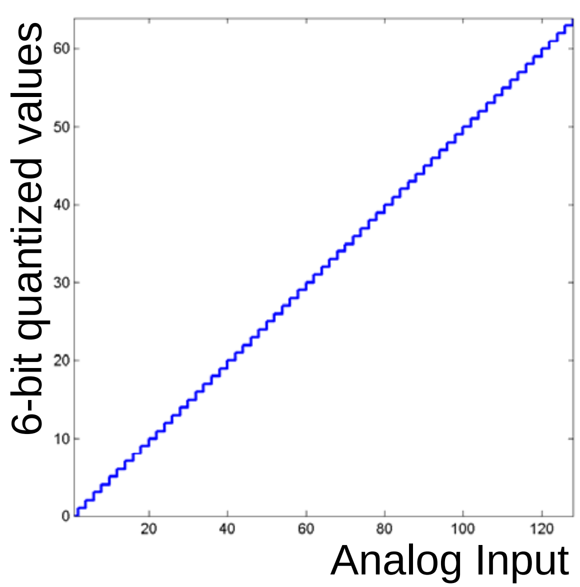
The Speech Signal: Linear Coding

- 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

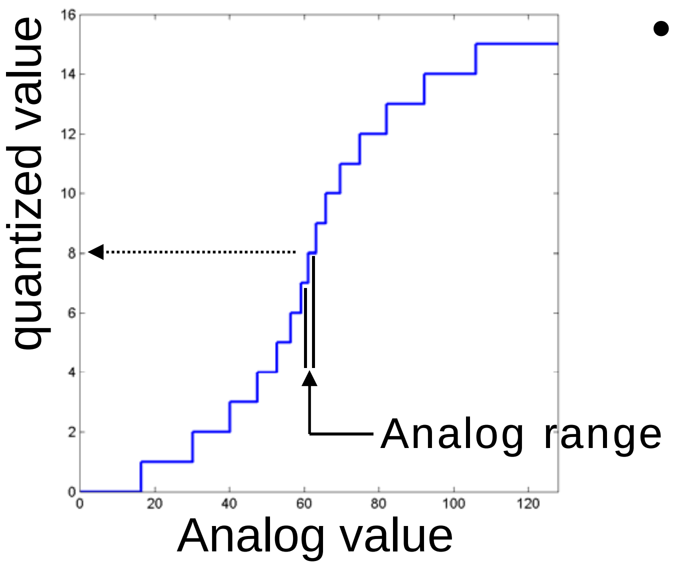
The Speech Signal: Linear Coding

- Example PCM quantizations into 16 and 64 levels:

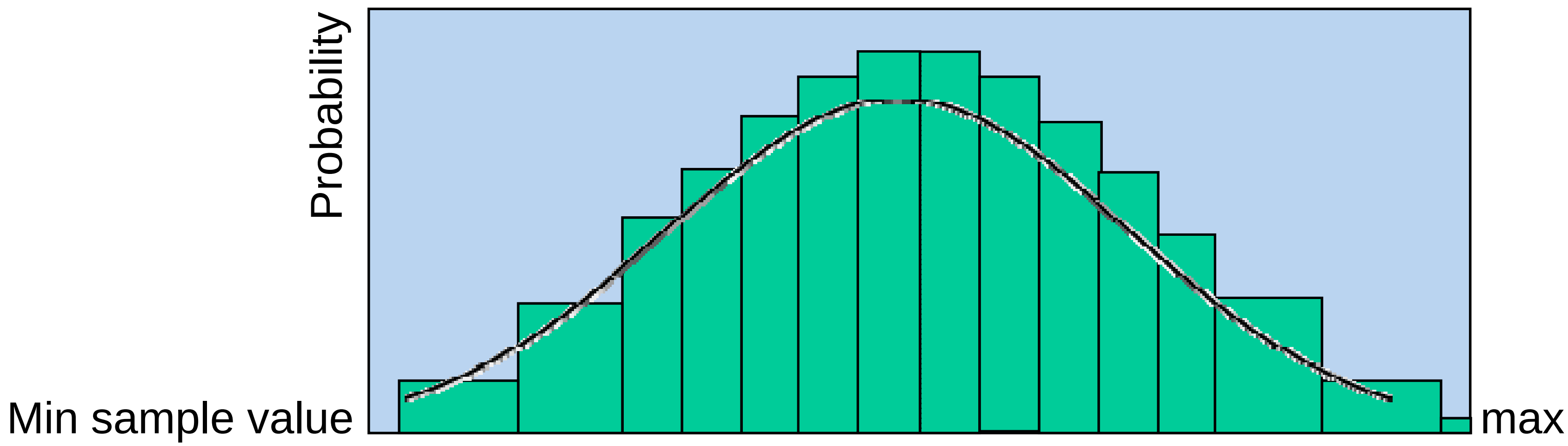




The Speech Signal: Non-Linear Coding



- 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

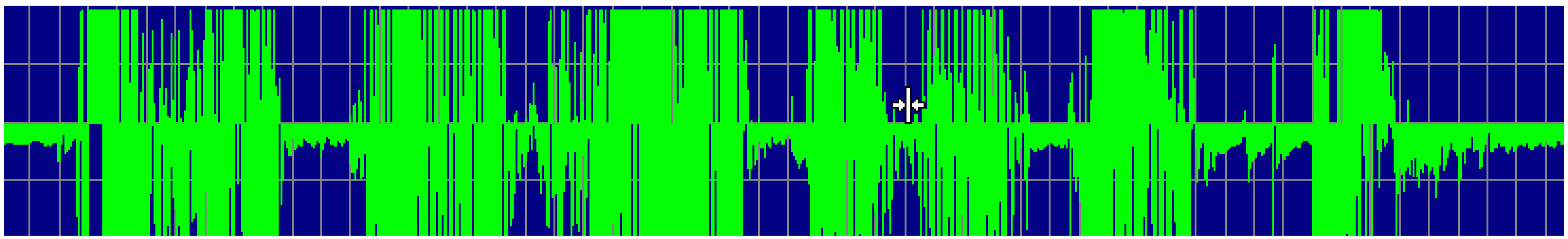
- Thus, fewer discrete levels can be used, without significantly worsening *average* quantization error
  - High resolution coding around the more frequent analog levels
  - Lower resolution coding around infrequent analog levels
- *A-law* and *μ-law* encoding schemes use only 256 levels (8-bit encodings)
  - 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

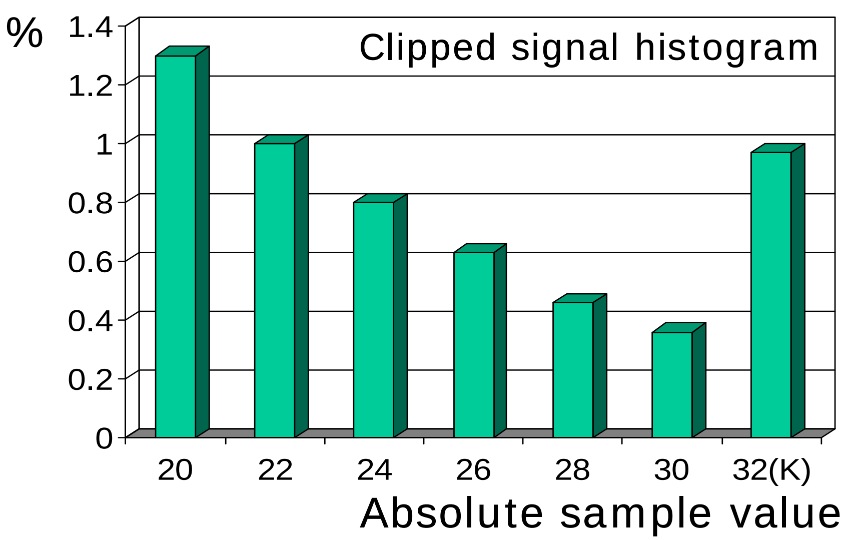
Effect of Signal Quality

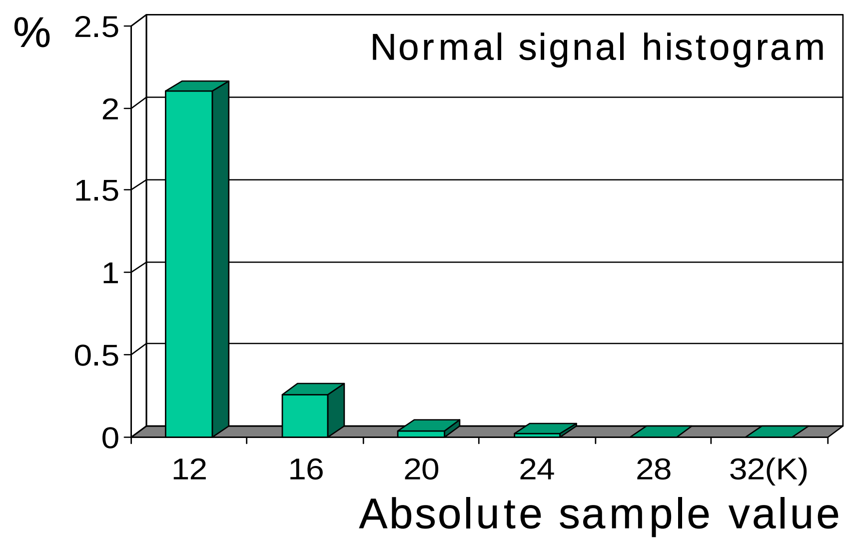
- The quality of the final digitized signal depends critically on all the other components:
  - The microphone quality
  - Ambient noise in recording environment
  - The electronics performing sampling and digitization
    - Poor quality electronics can severely degrade signal quality
      - *E.g.* 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)



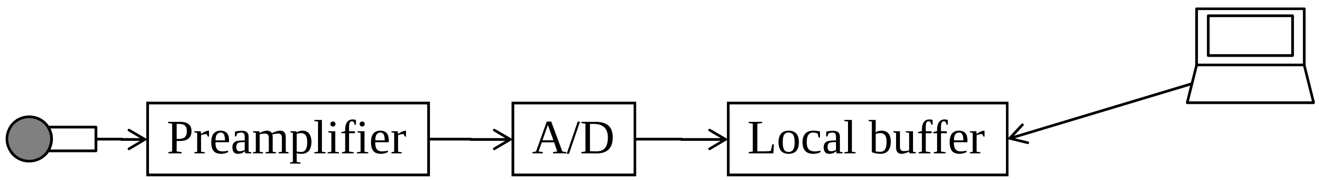




•Capturing speech signals

- Your computer must have a sound card, an A/D converter (which is sometimes external to the sound card), and audio input devices such as a microphone, line input etc.
- Offline capture: You can use tools available for your favorite OS
  - Windows provides a “Windows recorder”
  - Several audio capture tools are also available for windows
  - Linux and most Unix machines provide “arecord” and “aplay”
    - If these are not already on your machine, you can download them from the web
  - Other tools are also available for linux

Audio Capture



- Capture
  - Signal is captured by a microphone
  - Preamplified
  - Digitized
  - Store in a buffer on the sound card

- Processor
  - Reads from buffer
  - At some prespecified frequency
    - Too frequent: can use up all available CPU cycles
    - Too infrequent: High latency

Capturing Audio

- Capturing audio from your audio device
  - Open the audio device
    - Syntax is OS dependent
  - Set audio device parameters
  - Record blocks of audio
  - Close audio device

- Recorded audio can be stored in a file or used for live decoding
- Two modes of audio capture for live-mode decoding
  - Blocking: Application/decoder requests audio from the audio device when required
    - The program waits for the capture to be complete, after a request
  - Callback: An audio program monitors the audio device and captures data. When it has sufficient data it calls the application or decoder

Capturing speech signals

- Example linux pseudocode for capturing audio on an HP IPaq (for single-channel 16khz 16bit PCM sampling):

```
fd = open("/dev/dsp", O_RDONLY);
ioctl(fd, SOUND_PCM_WRITE_BITS, 16);
ioctl(fd, SOUND_PCM_WRITE_CHANNELS, 1);
ioctl(fd, SOUND_PCM_WRITE_RATE, 16000);
while (1) {
    read(fd, buffer, Nsamples*sizeof(short));
    process(buffer);
}
close(fd);
```

Storing Audio/Speech

- There are many storage formats in use.
- Important ones:
  - PCM raw data (*.raw)
  - NIST (*.sph)
  - Microsoft PCM (*.wav)
  - Microsoft ADPCM (*.wav)
  - SUN (*.au, *.snd) etc.

- The data are typically written in binary, but many of these formats have headers that can be read as ascii text.
  - Headers store critical information such as byte order, no. of samples, coding type, bits per sample, sampling rate etc.

- Speech files must be converted from store format to linear PCM format for further processing

First Step: Feature Extraction

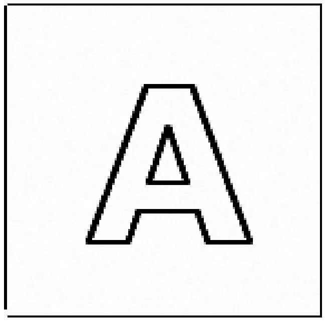
- Speech recognition is a type of pattern recognition problem
- *Q:* Should the pattern matching be performed on the audio sample streams directly? If not, what?
- *A:* Raw sample streams are not well suited for matching
- A visual analogy: recognizing a letter inside a box

  - 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

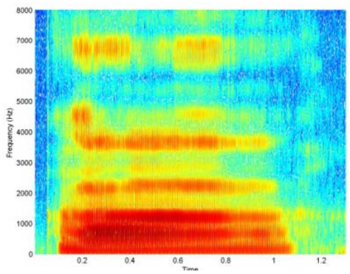


- Our brain does this kind of image analysis automatically and we can instantly identify the input letter as being the same as the template

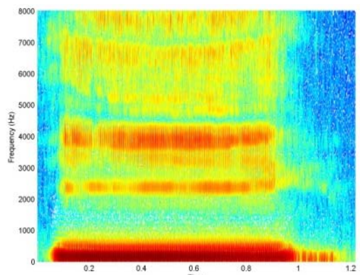
Sound Characteristics are in Frequency Patterns

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

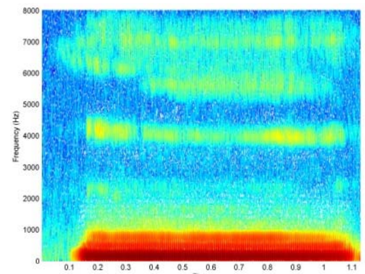
AA



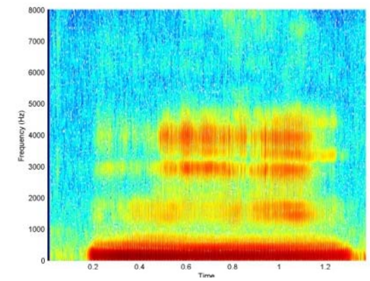
IY



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

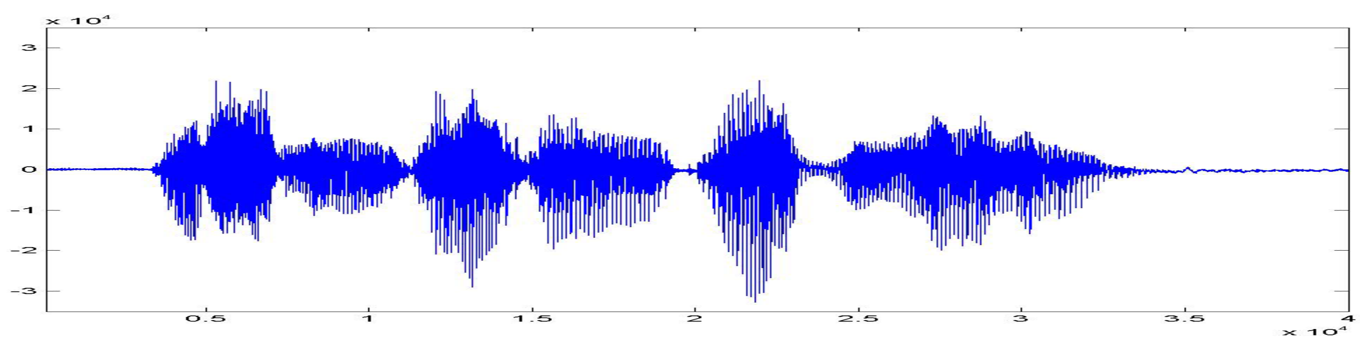
- What follows: A quick intro to signal processing
  - All necessary aspects

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

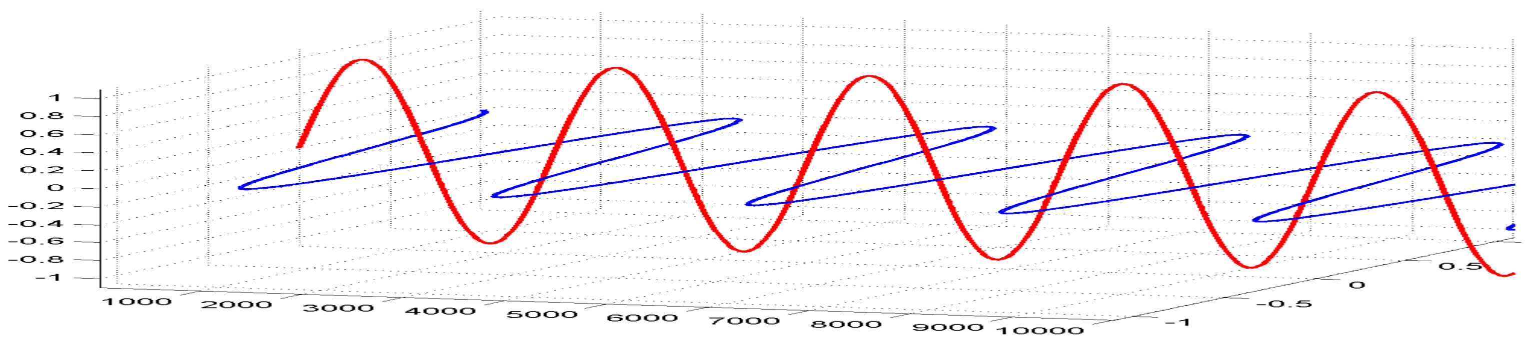
$$e^{j\theta} = \cos\theta + j \sin\theta$$

- 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

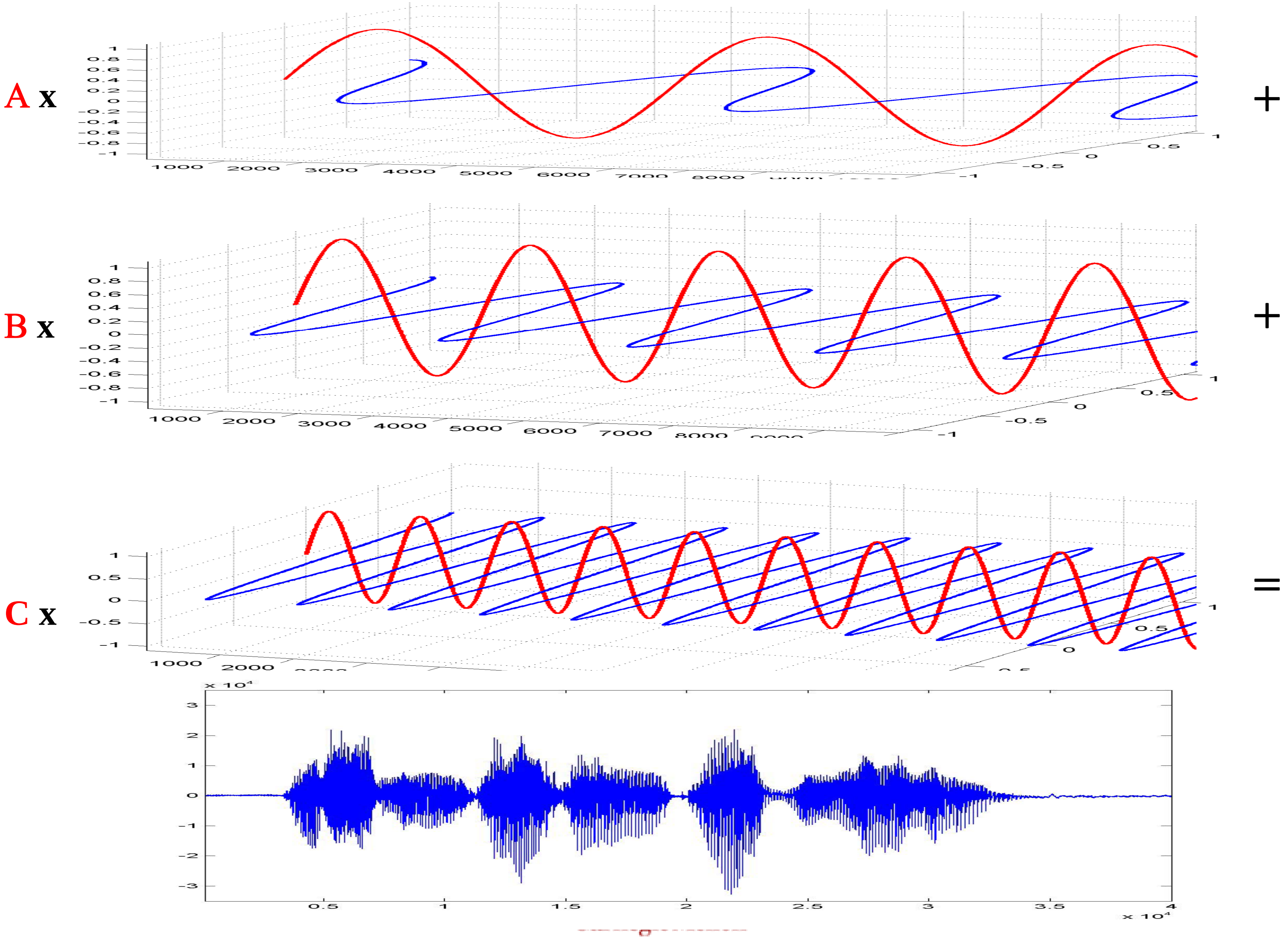
$$e^{j\omega t} = \cos(\omega t) + j \sin(\omega t)$$

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

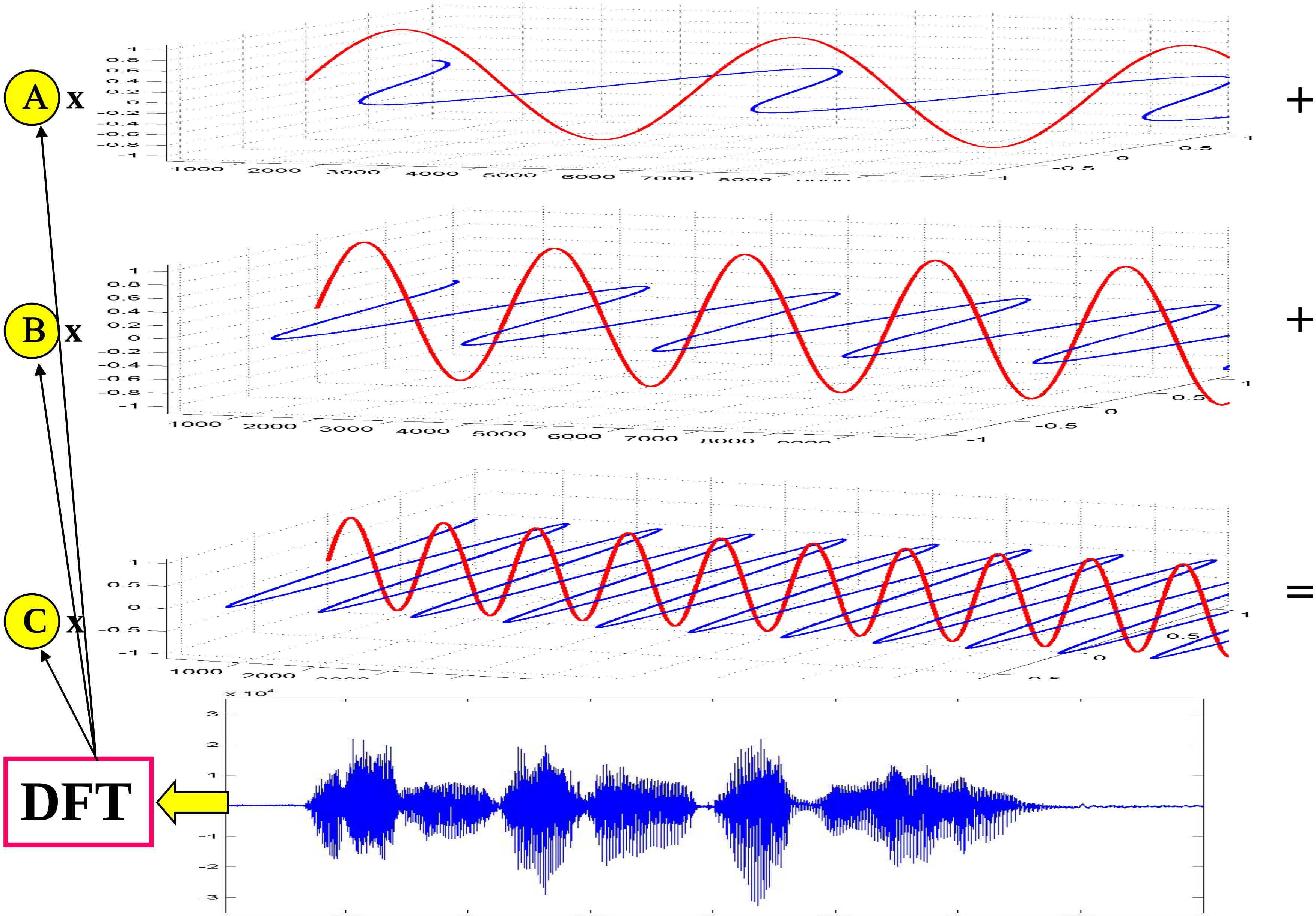
$$\int_{-\infty}^{\infty} e^{j\alpha t} e^{j\beta t} dt = 0 \qquad \text{if } \alpha \neq \beta$$



The discrete Fourier transform



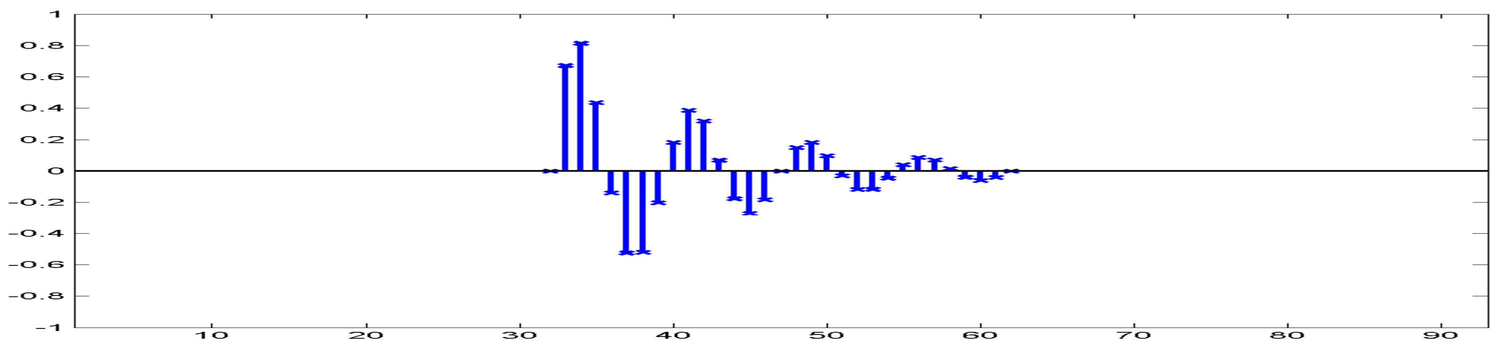
The discrete Fourier transform



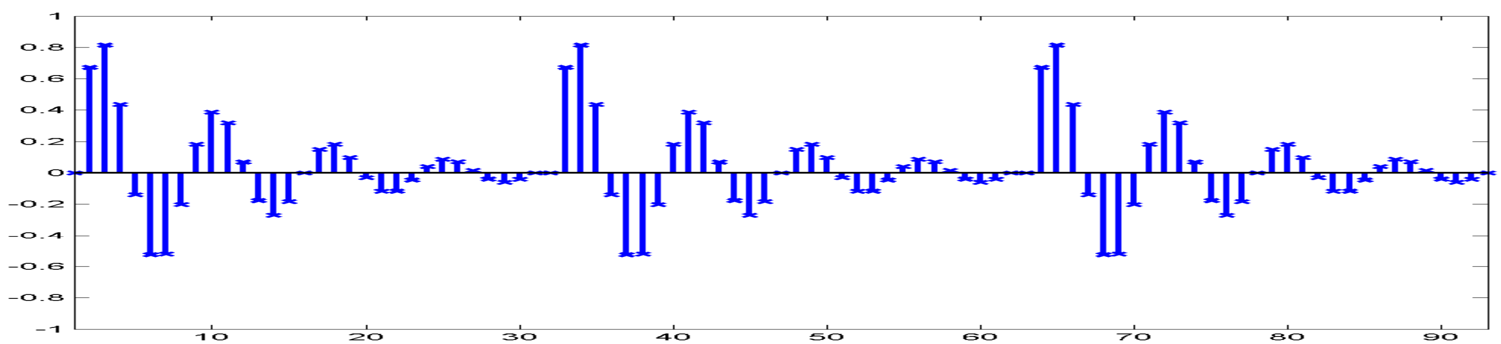
The discrete Fourier transform

- 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

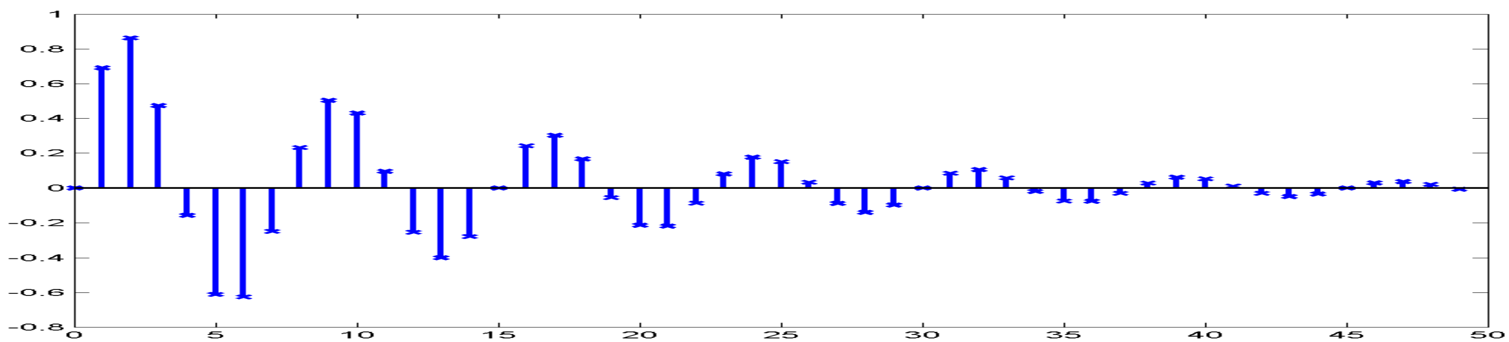
- The 0^{th} point in the DFT represents 0Hz, or the DC component of the signal

- The $(M-1)^{th}$ point in the DFT represents (M-1)/M times the sampling frequency

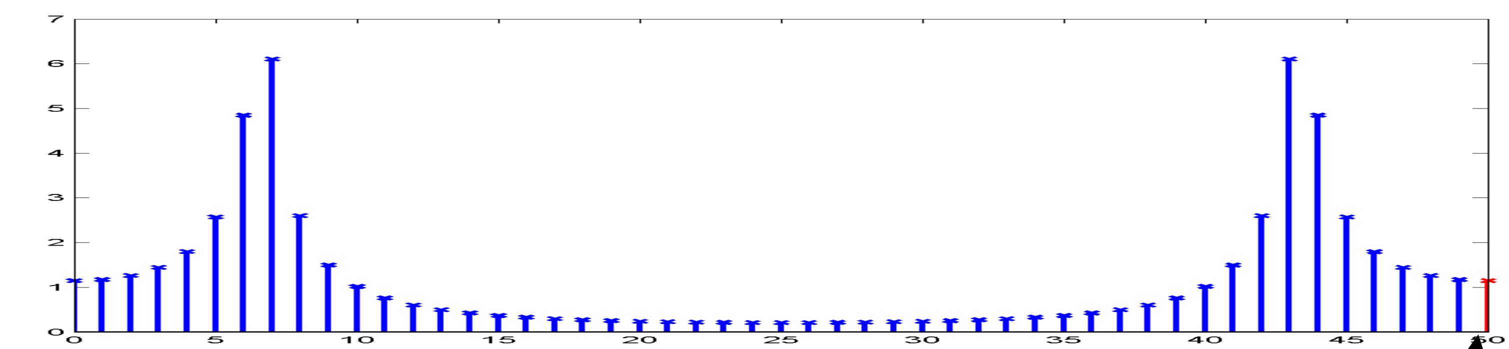
- All DFT points are uniformly spaced on the frequency axis between 0 and the sampling frequency

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.



Sample 0 = 0 Hz

Sample 50 is the 51st point

It is identical to Sample 0

Sample 50 = 8000Hz

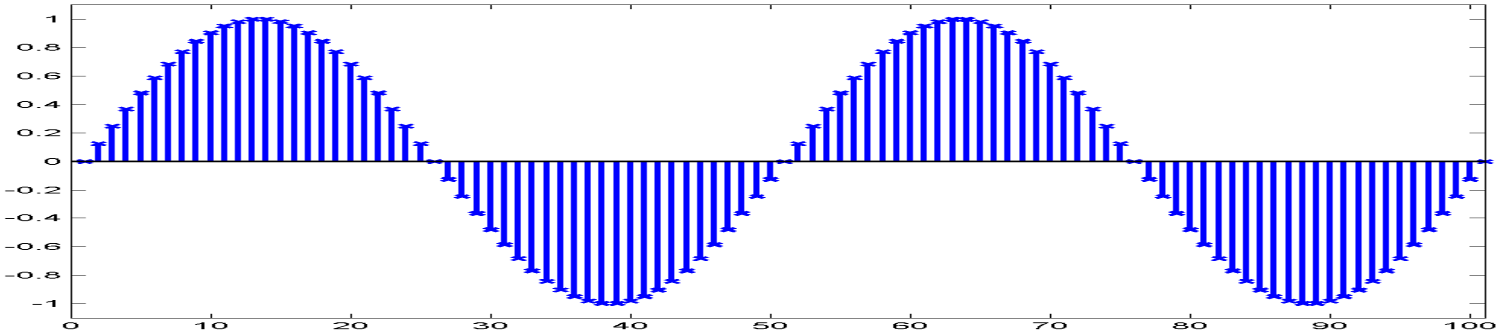
The discrete Fourier transform

- 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:

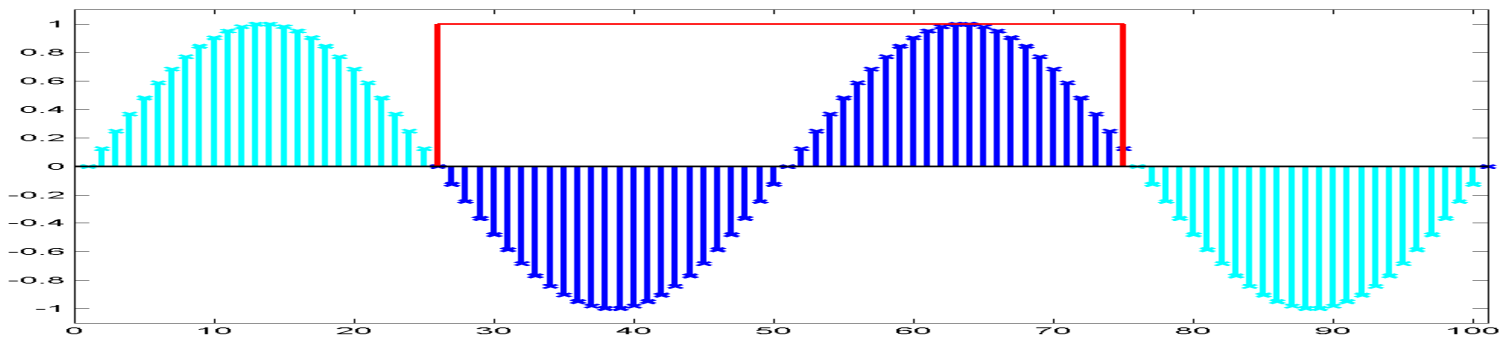
$$x[n] = \frac{1}{M} \sum_{k=0}^{M-1} X[k] e^{\frac{j2\pi kn}{M}}$$

Windowing



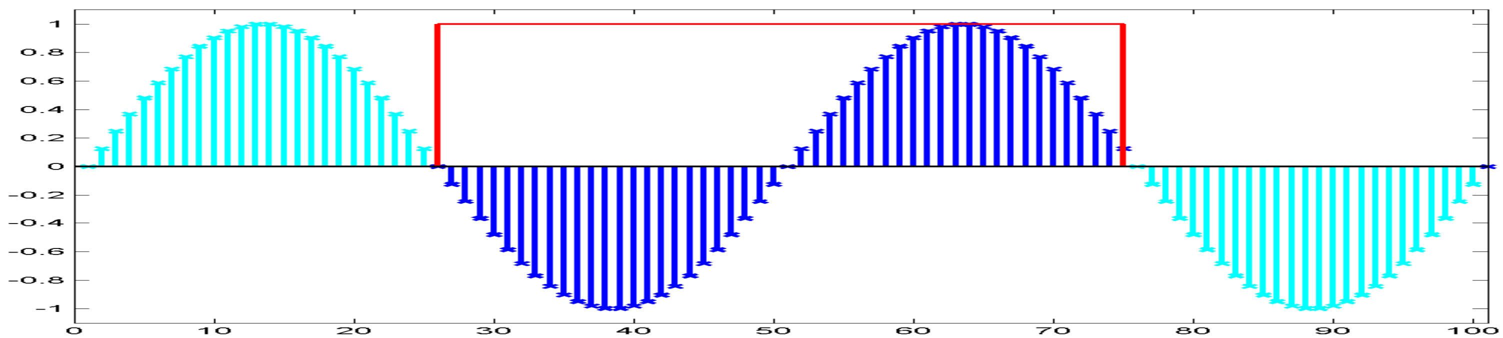
- The DFT of one period of the sinusoid shown in the figure computes the Fourier series of the entire sinusoid from –infinity to +infinity

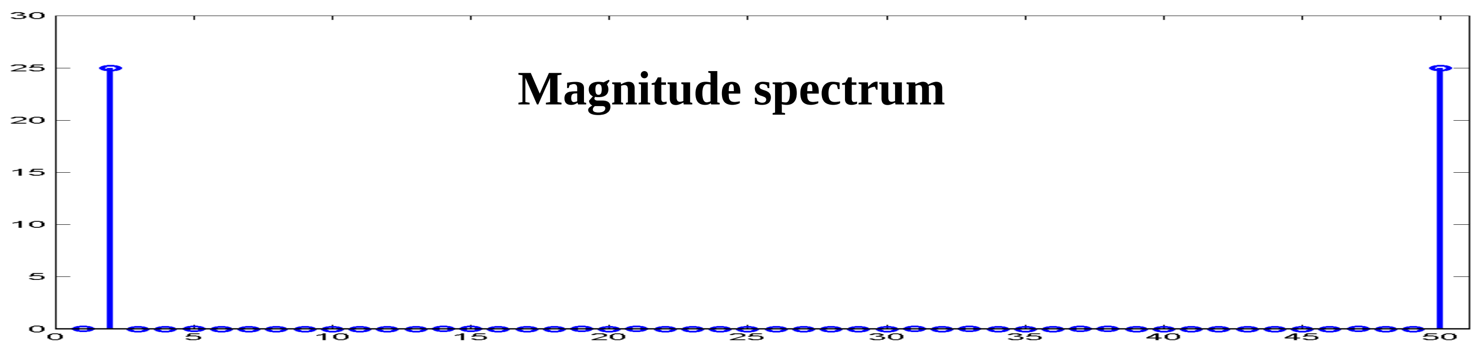
Windowing



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

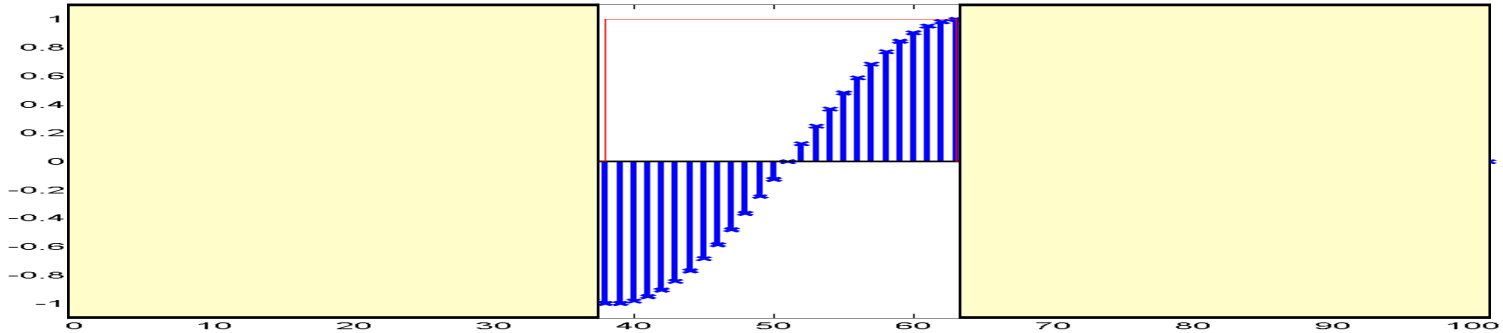
Windowing





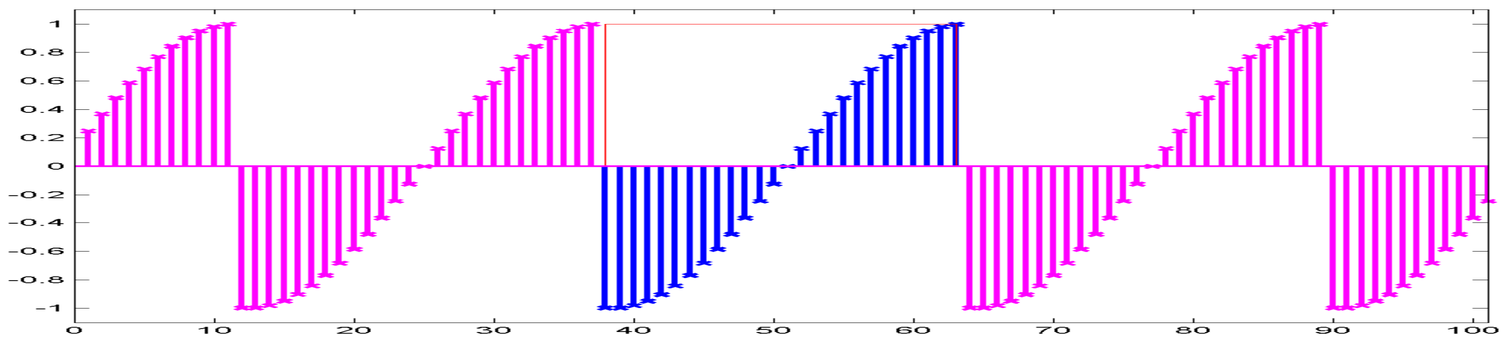
- The DFT of one period of the sinusoid shown in the figure computes the Fourier series of the entire sinusoid from –infinity to +infinity

Windowing



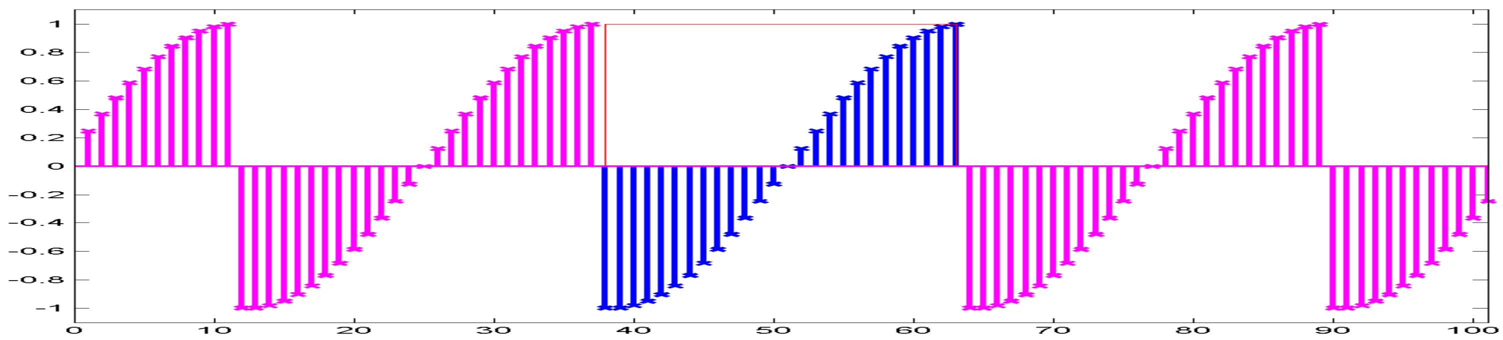
- The DFT of *any* sequence computes the Fourier series for an infinite repetition of that sequence

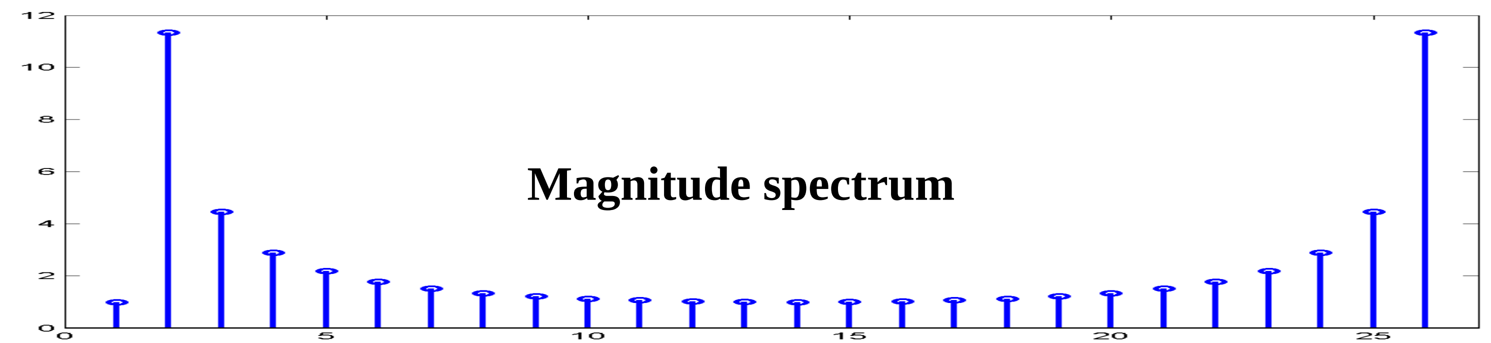
Windowing



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

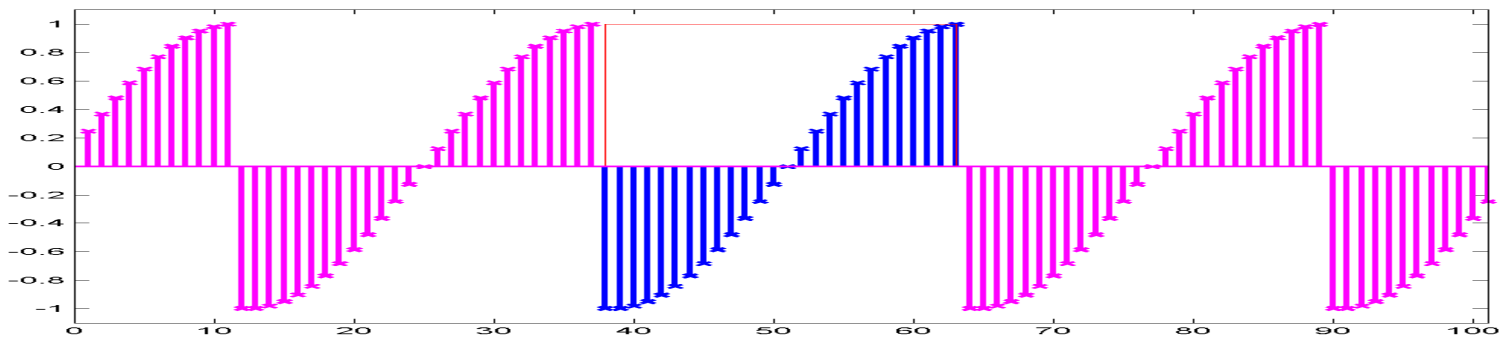
Windowing

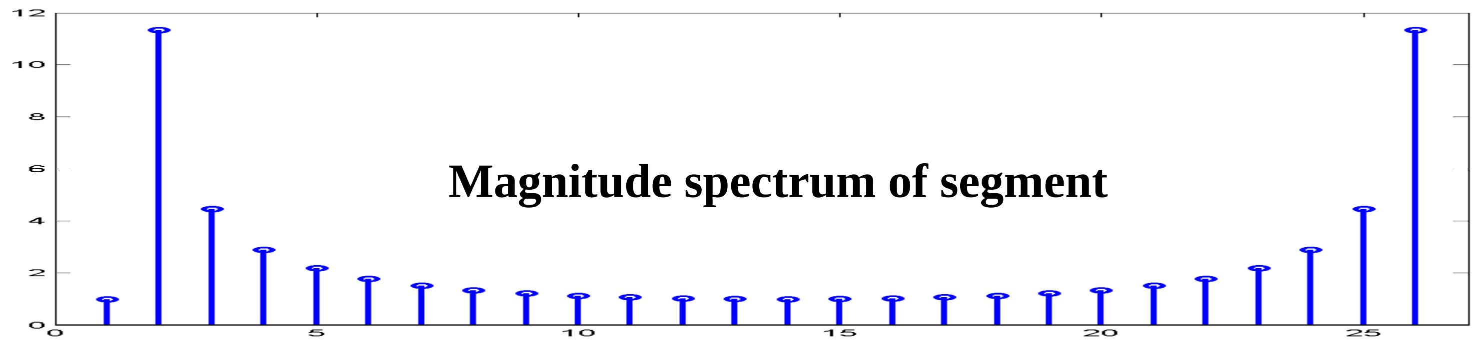


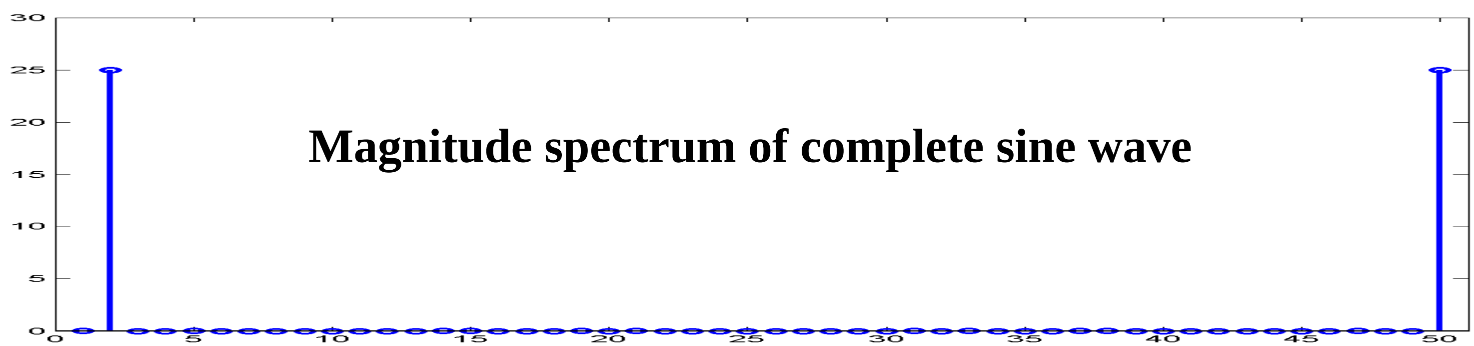


- The DFT of *any* sequence computes the Fourier series for an infinite repetition of that sequence
- The DFT of a partial segment of a sinusoid computes the Fourier series of an inifinite repetition of that segment, and not of the entire sinusoid
- This will not give us the DFT of the sinusoid itself!

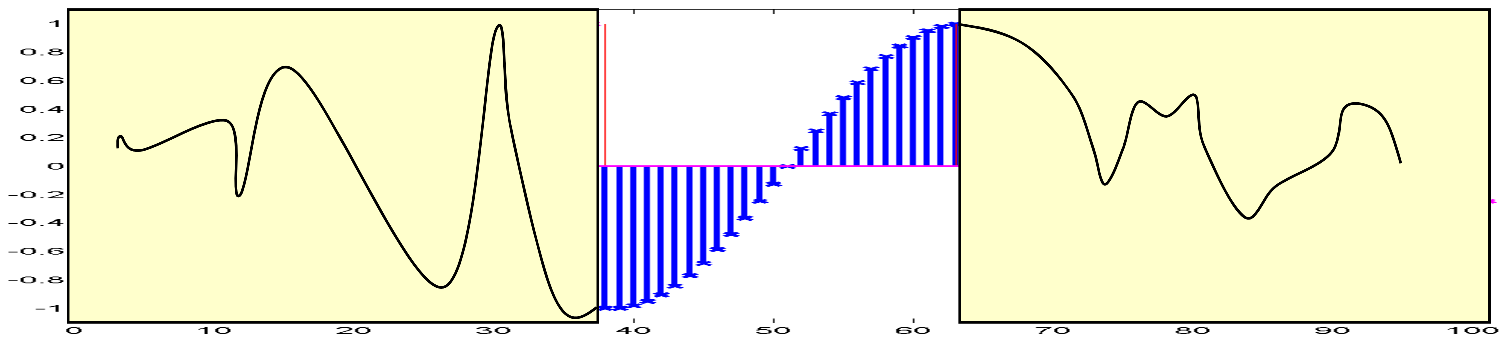
Windowing




Magnitude spectrum of segment

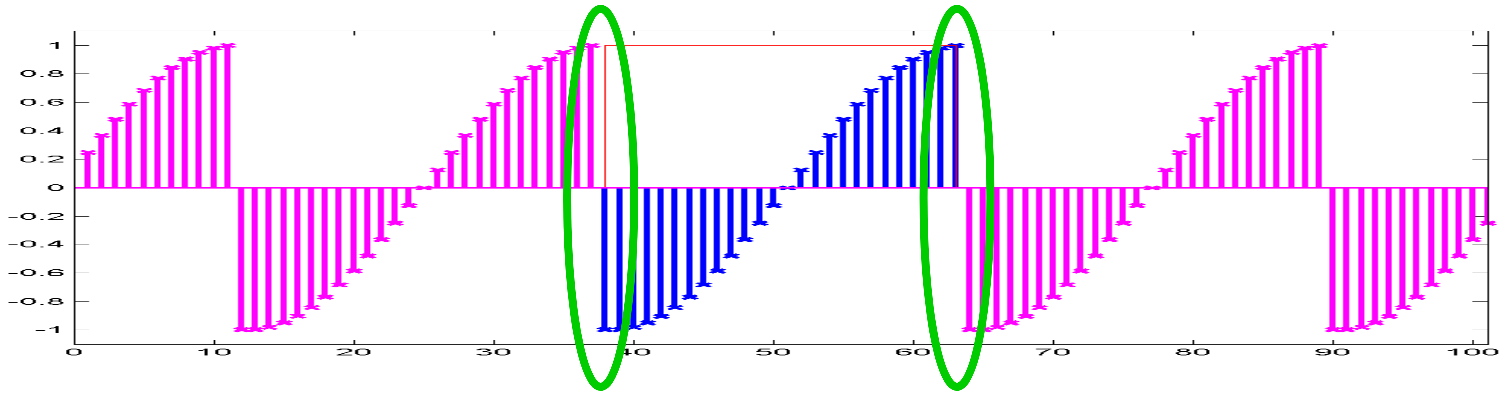

Magnitude spectrum of complete sine wave

Windowing



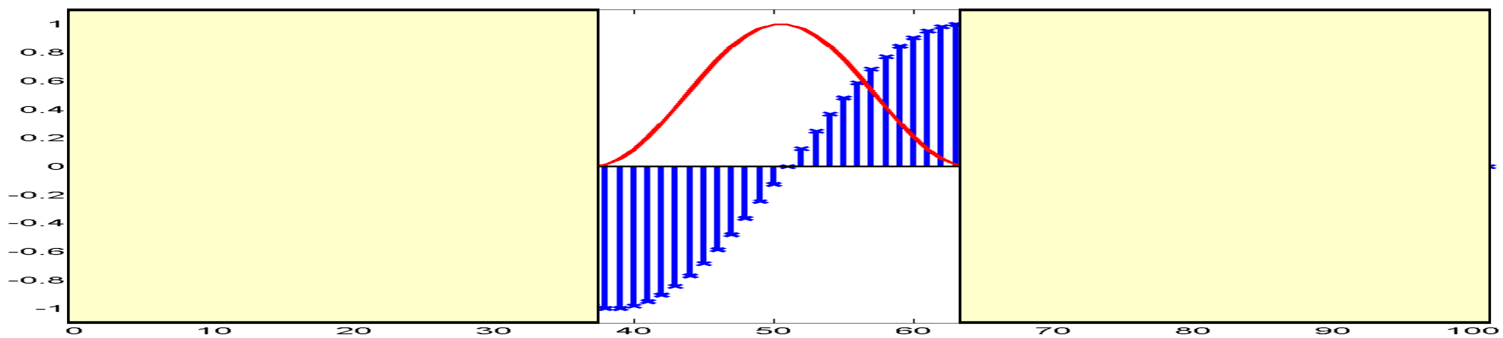
- The difference occurs due to two reasons:
- The transform cannot know what the signal actually looks like outside the observed window
  - We must infer what happens outside the observed window from what happens inside

Windowing



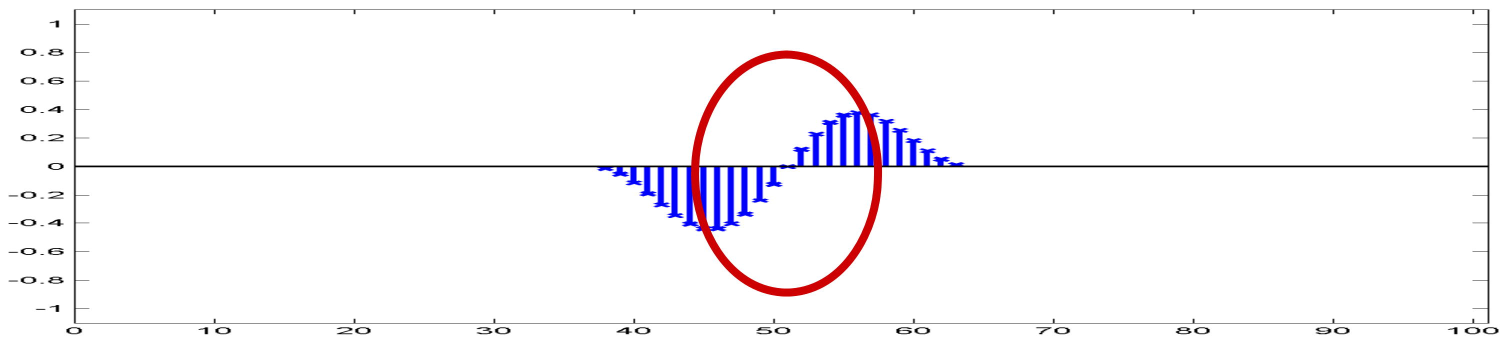
- The difference occurs due to two reasons:
- The transform cannot know what the signal actually looks like outside the observed window
  - We must infer what happens outside the observed window from what happens inside
- The implicit repetition of the observed signal introduces large discontinuities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



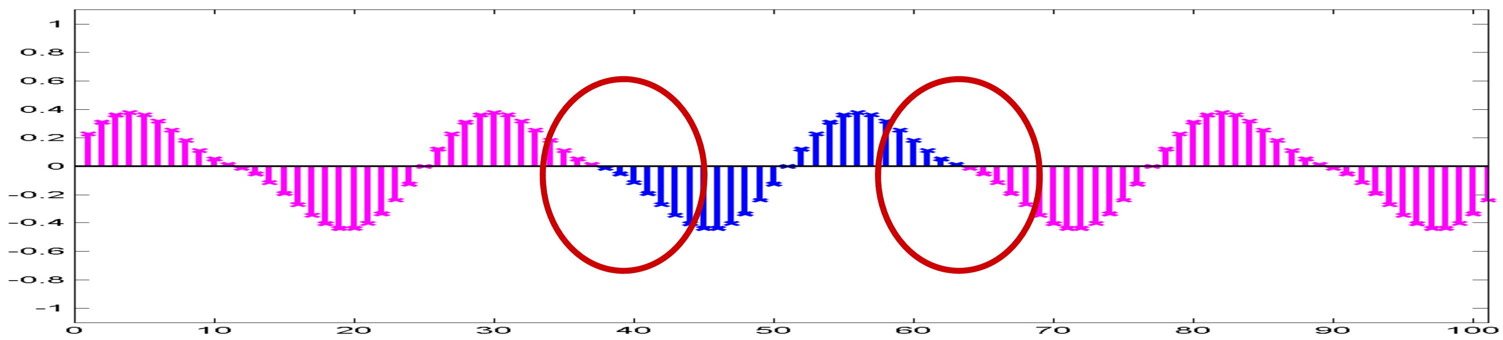
- While we can never know what the signal looks like outside the window, we can try to minimize the discontinuities at the boundaries
- We do this by multiplying the signal with a *window* function
  - We call this procedure windowing
  - We refer to the resulting signal as a "windowed" signal

Windowing



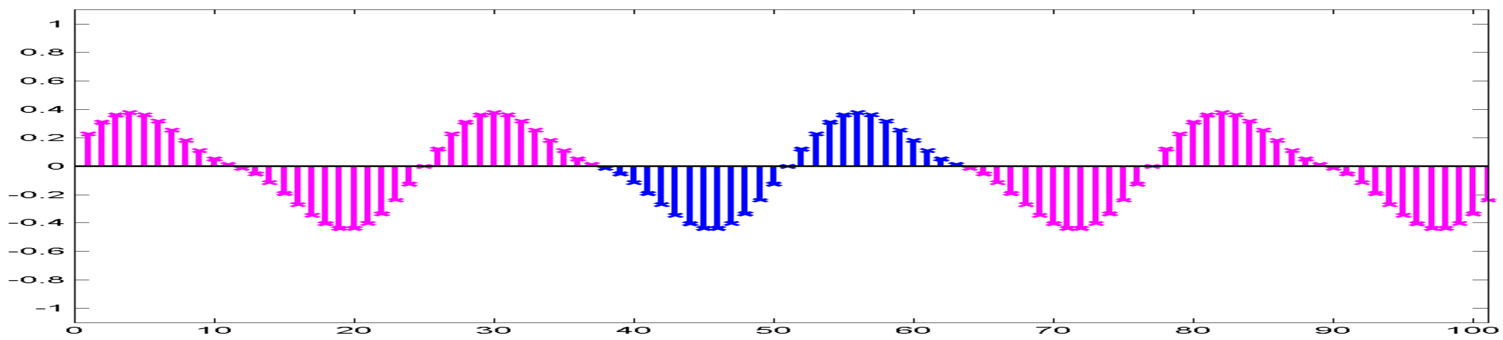
- While we can never know what the signal looks like outside the window, we can try to minimize the discontinuities at the boundaries
- We do this by multiplying the signal with a *window* function
  - We call this procedure windowing
  - We refer to the resulting signal as a “windowed” signal
- Windowing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

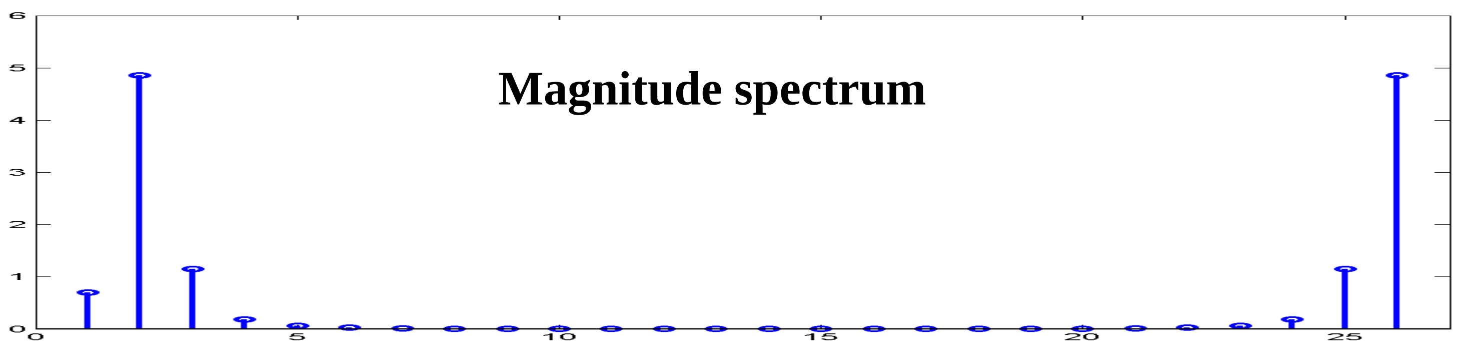
Windowing



- 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

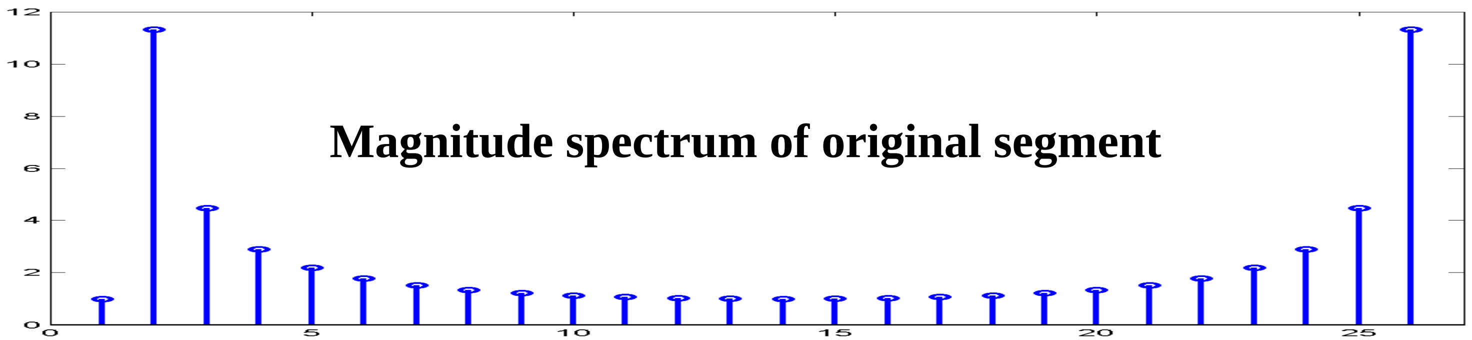
Windowing

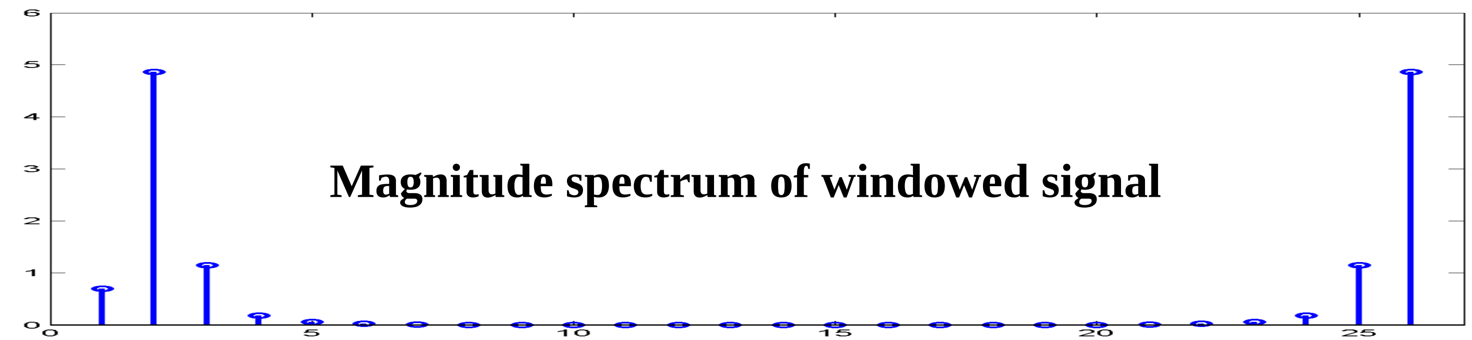


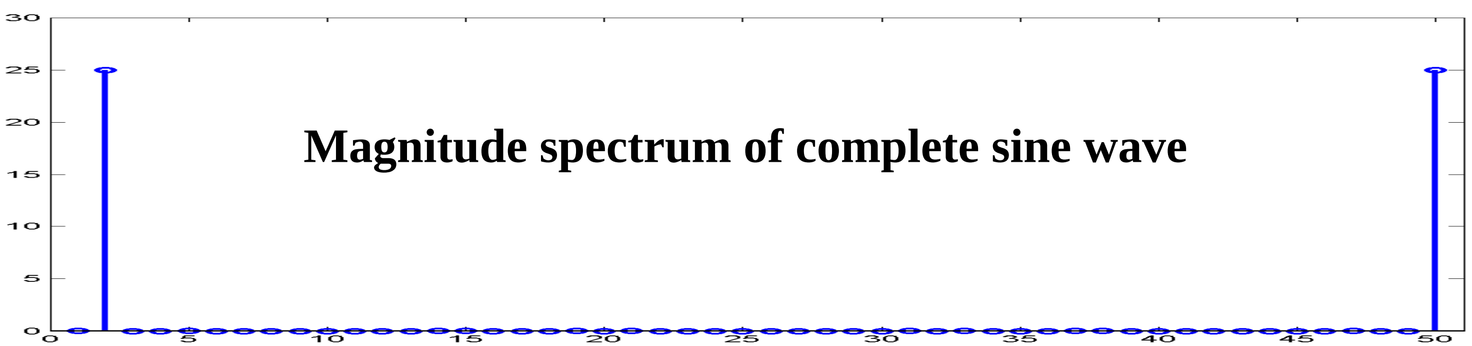


- The DFT of the windowed signal does not have any artifacts introduced by discontinuities in the signal
- Often it is also a more faithful reproduction of the DFT of the complete signal whose segment we have analyzed

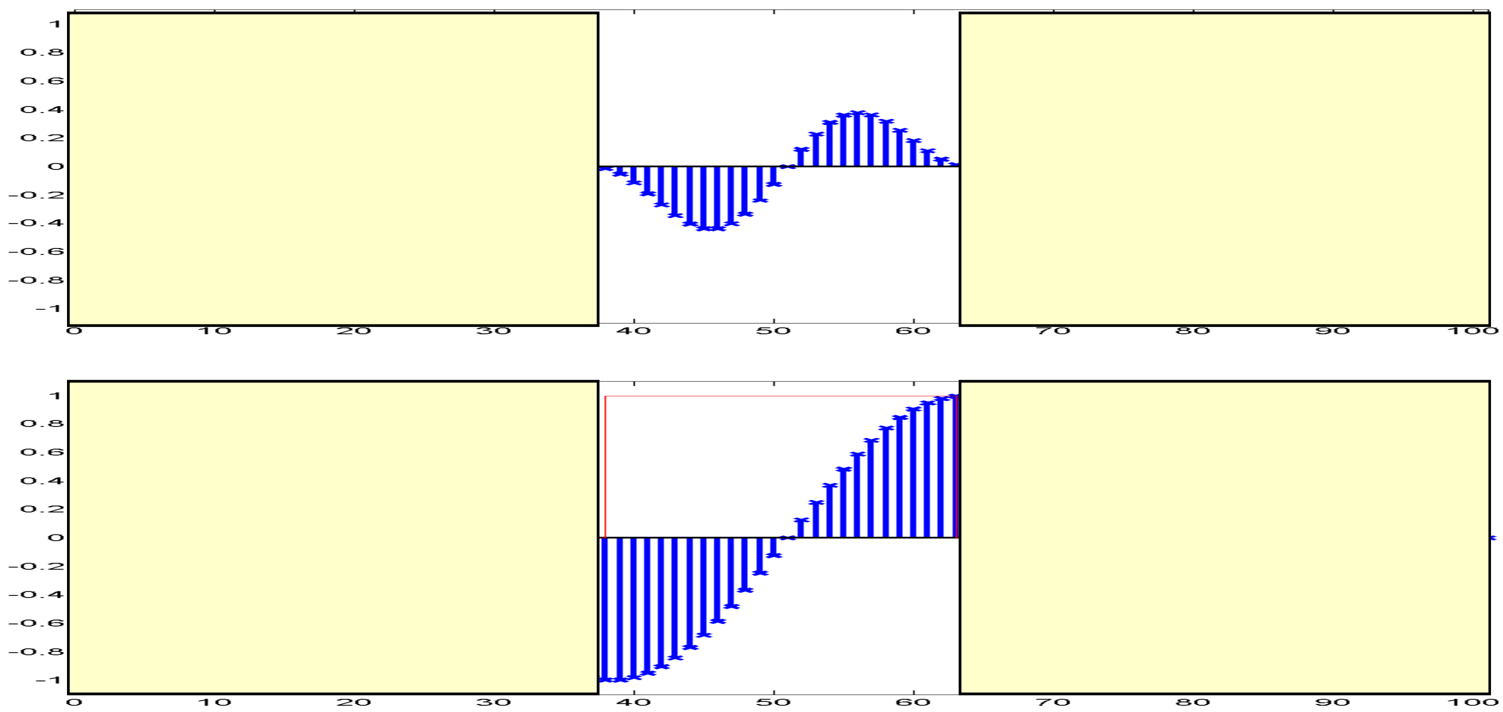
Windowing


Magnitude spectrum of original segment


Magnitude spectrum of windowed signal

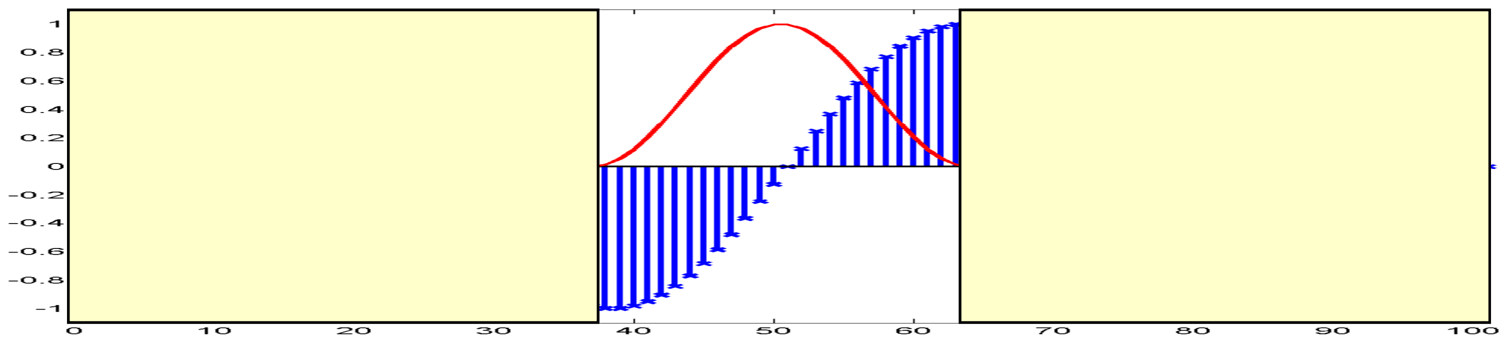

Magnitude spectrum of complete sine wave

Windowing



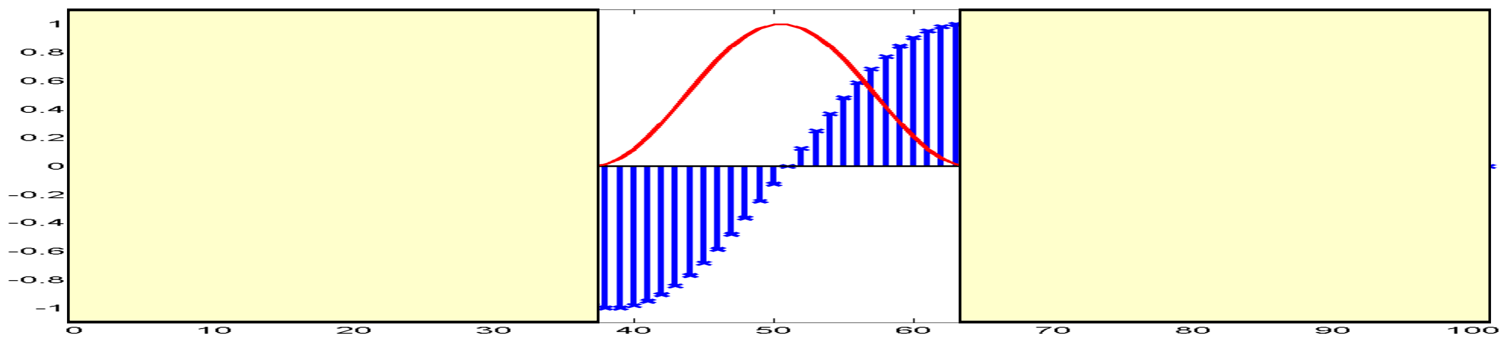
- Windowing is not a perfect solution
  - The original (unwindowed) segment is identical to the original (complete) signal within the segment
  - The windowed segment is often not identical to the complete signal anywhere
- Several windowing functions have been proposed that strike different tradeoffs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

Windowing

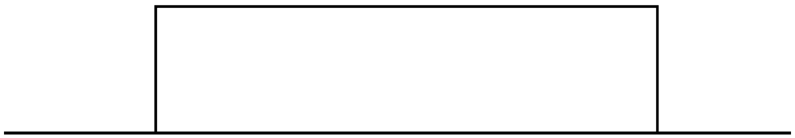


- Cosine windows:
  - Window length is M
  - Index begins at 0
- Hamming: $w[n] = 0.54 – 0.46 \cos(2\pi n/M)$
- Hanning: $w[n] = 0.5 – 0.5 \cos(2\pi n/M)$
- Blackman: $0.42 – 0.5 \cos(2\pi n/M) + 0.08 \cos(4\pi n/M)$

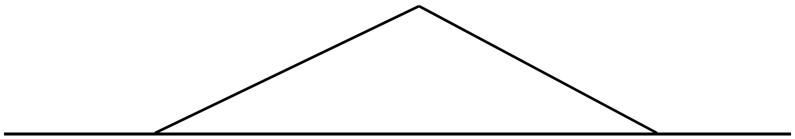
Windowing



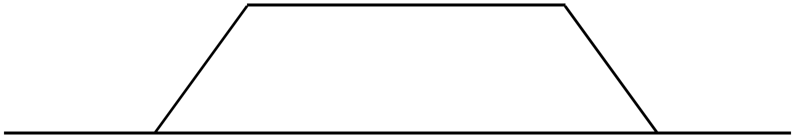
- Geometric windows:
  - Rectangular (boxcar):



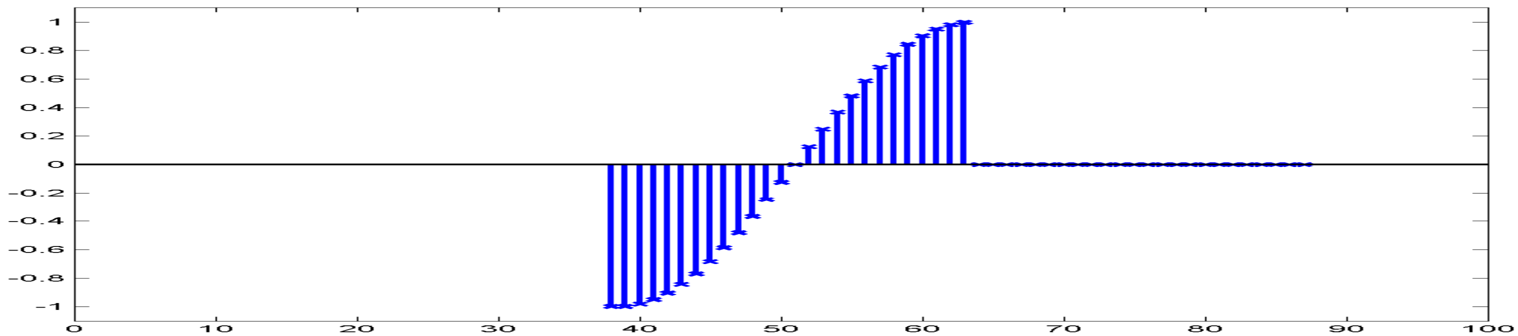
  - Triangular (Bartlett):



  - Trapezoid:

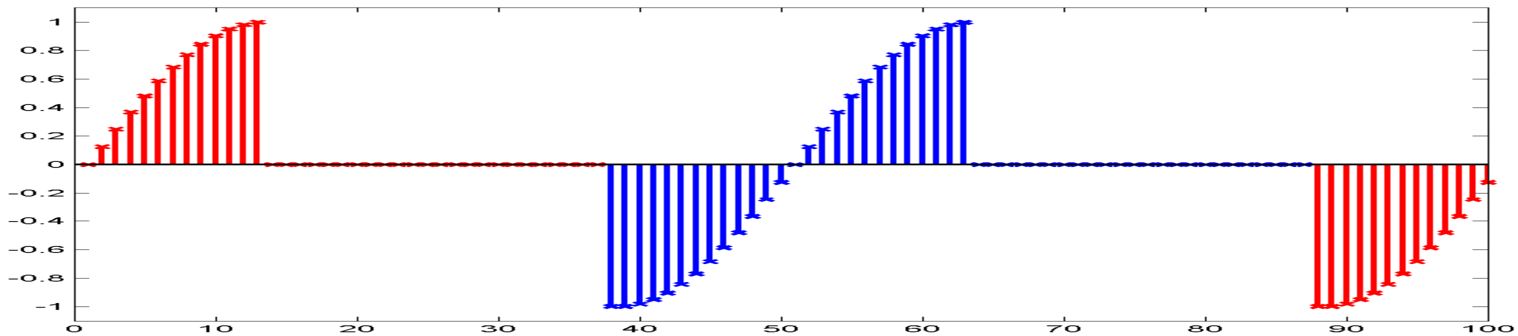


Zero Padding



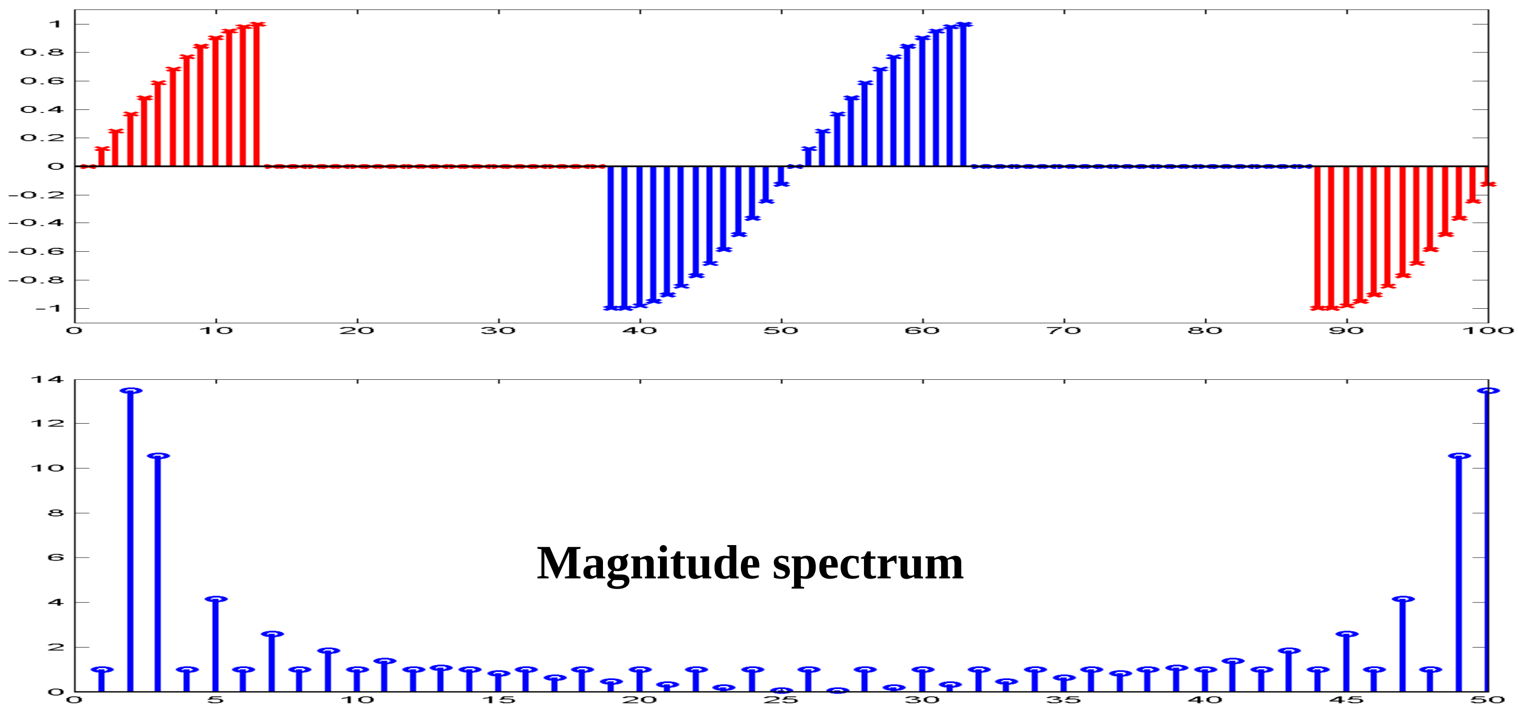
- We can pad zeros to the end of a signal to make it a desired length
  - Useful if the FFT (or any other algorithm we use) requires signals of a specified length
  - E.g. Radix 2 FFTs require signals of length 2^n i.e., some power of 2. We must zero pad the signal to increase its length to the appropriate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

Zero Padding



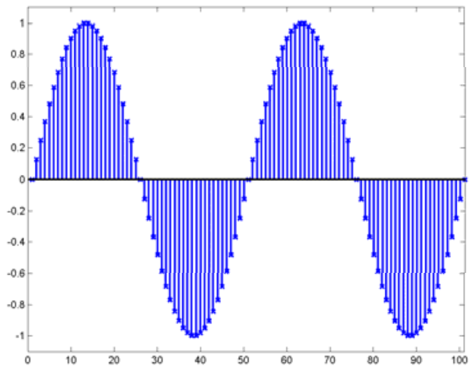
- We can pad zeros to the end of a signal to make it a desired length
  - Useful if the FFT (or any other algorithm we use) requires signals of a specified length
  - E.g. Radix 2 FFTs require signals of length 2^n i.e., some power of 2. We must zero pad the signal to increase its length to the appropriate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

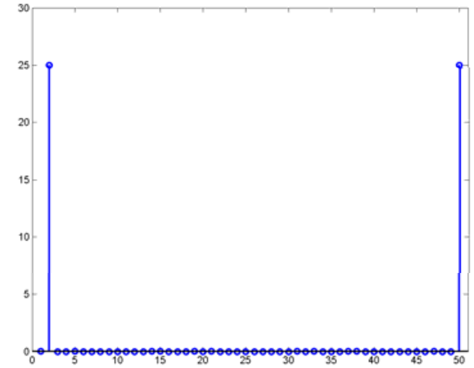
Zero Padding

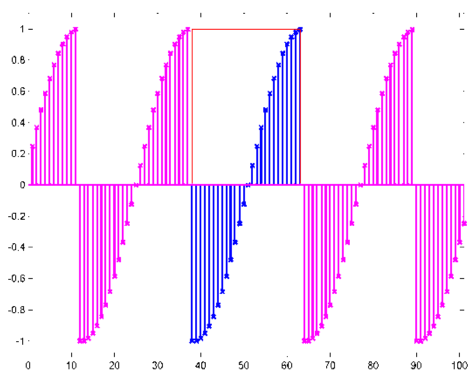


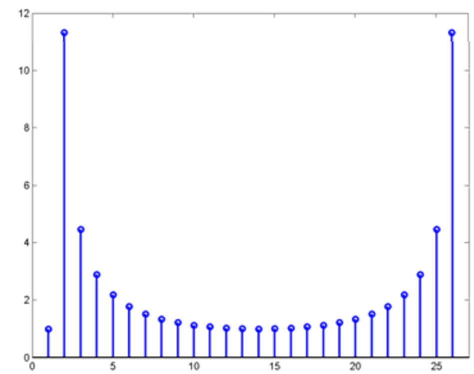
- The DFT of the zero padded signal is essentially the same as the DFT of the unpadded signal, with additional spectral samples inserted in between
  - It does not contain any additional information over the original DFT
  - It also does not contain less information

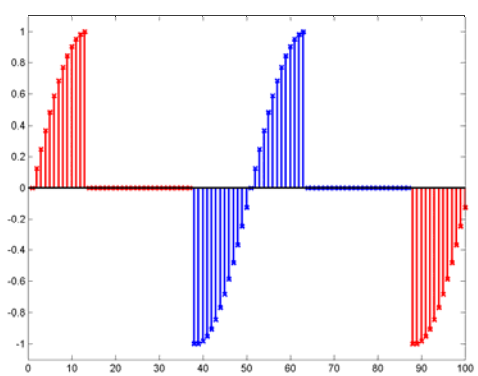
Magnitude spectra

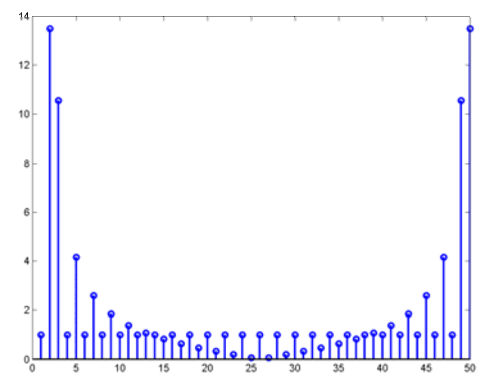




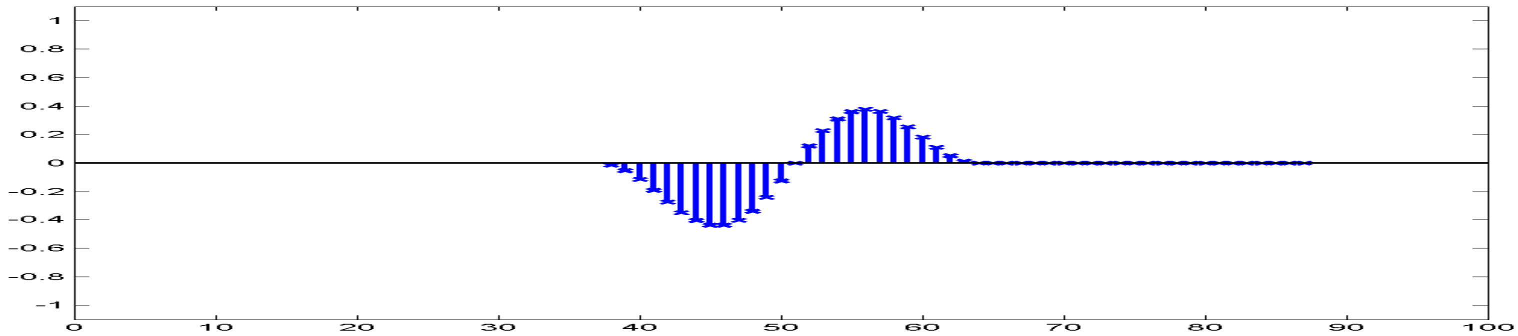


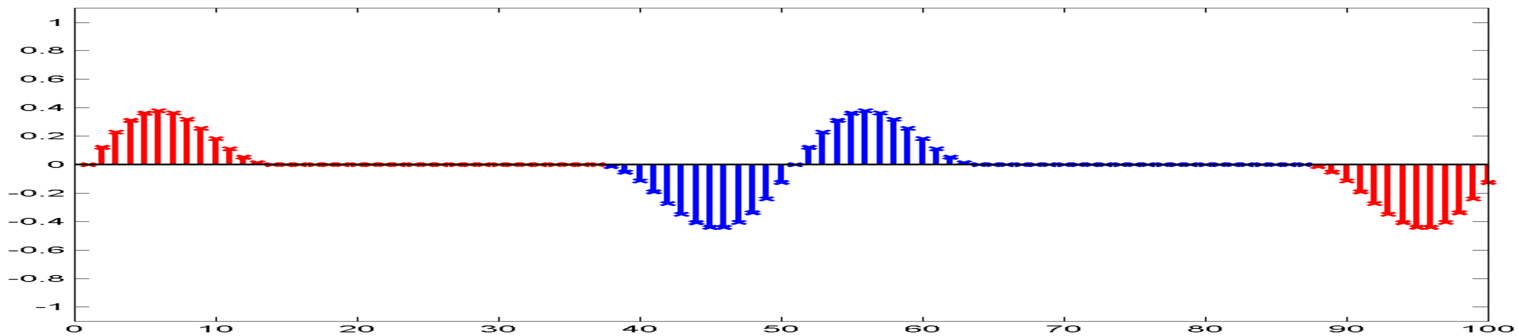






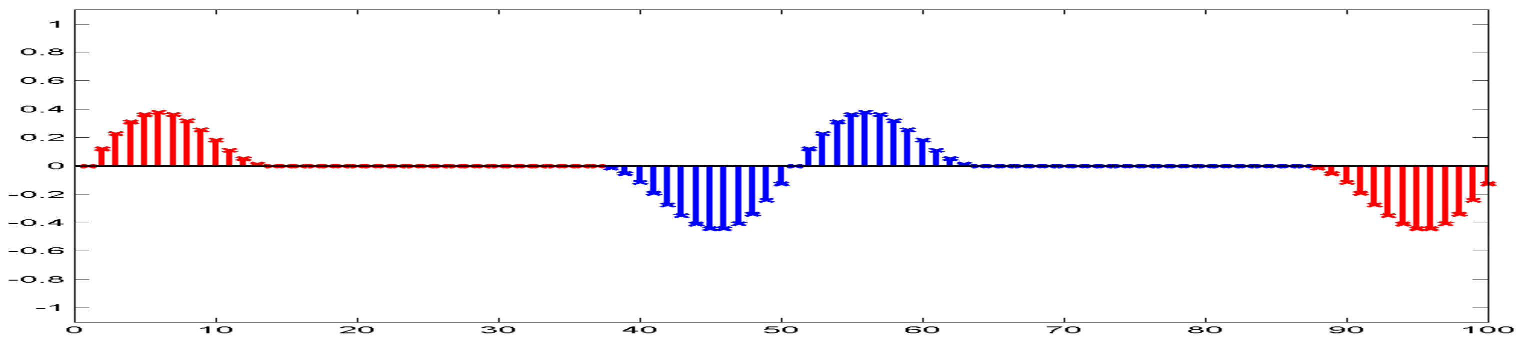
Zero Padding

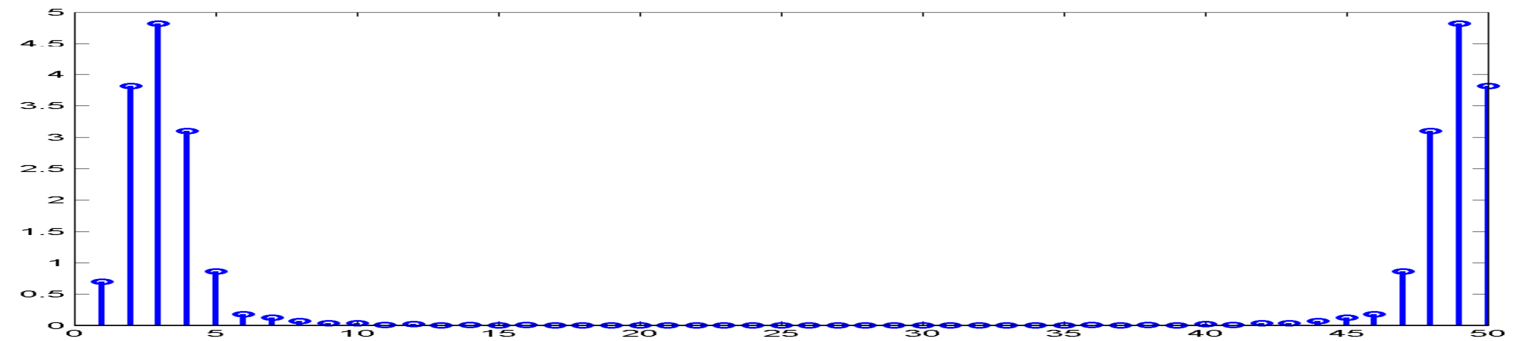




- Zero padding windowed signals results in signals that appear to be less discontinuous at the edges
  - This is only illusory
  - Again, we do not introduce any new information into the signal by merely padding it with zeros

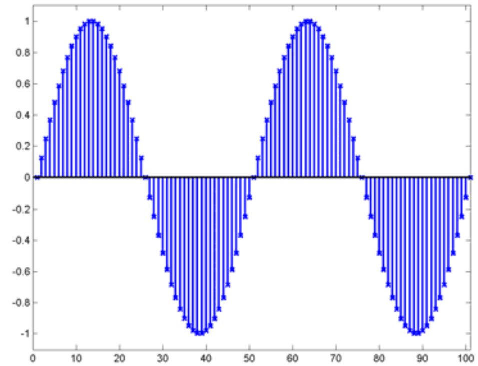
Zero Padding

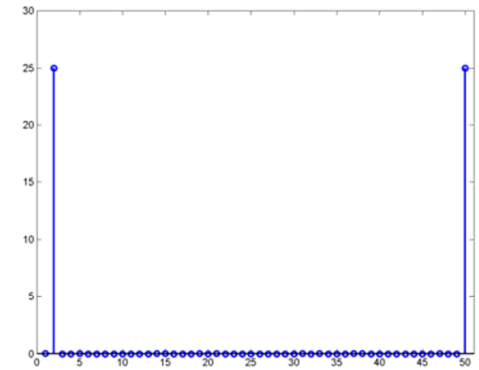


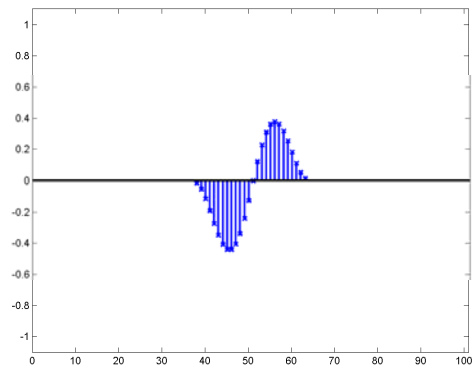


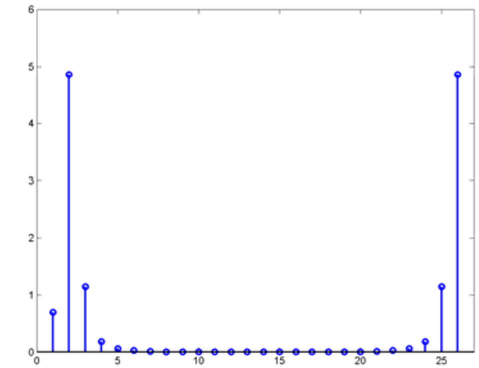
- The DFT of the zero padded signal is essentially the same as the DFT of the unpadded signal, with additional spectral samples inserted in between
  - It does not contain any additional information over the original DFT
  - It also does not contain less information

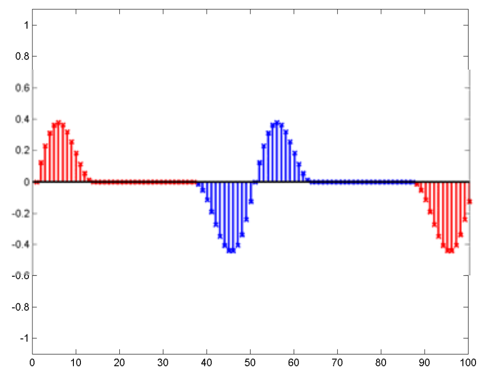
Magnitude spectra

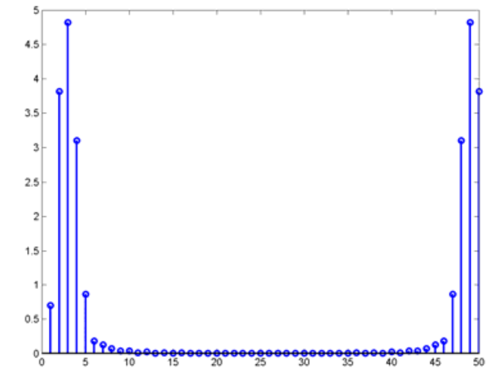






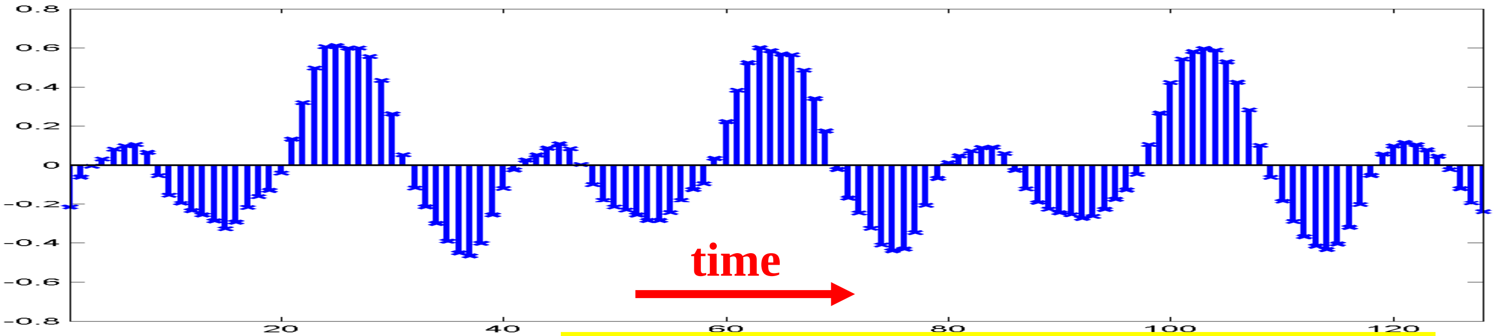




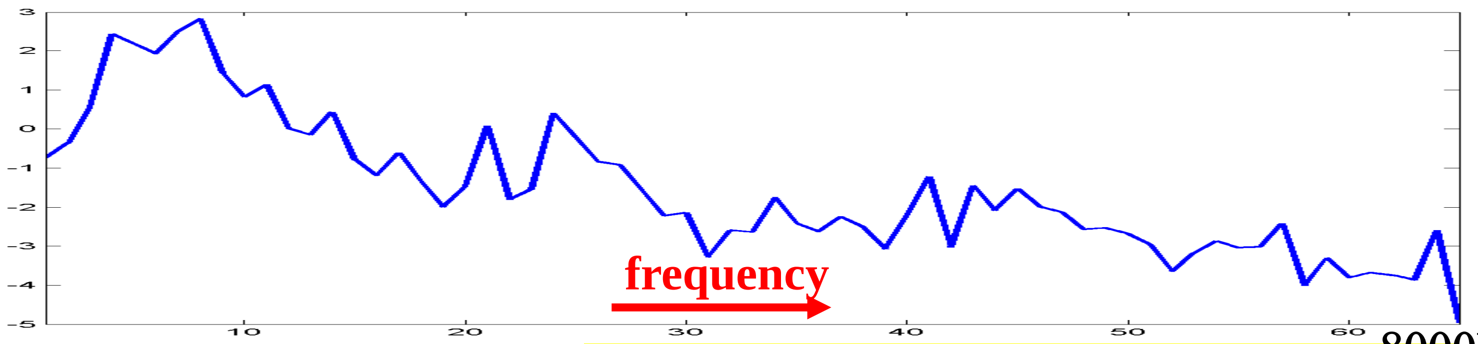


Zero padding a speech signal

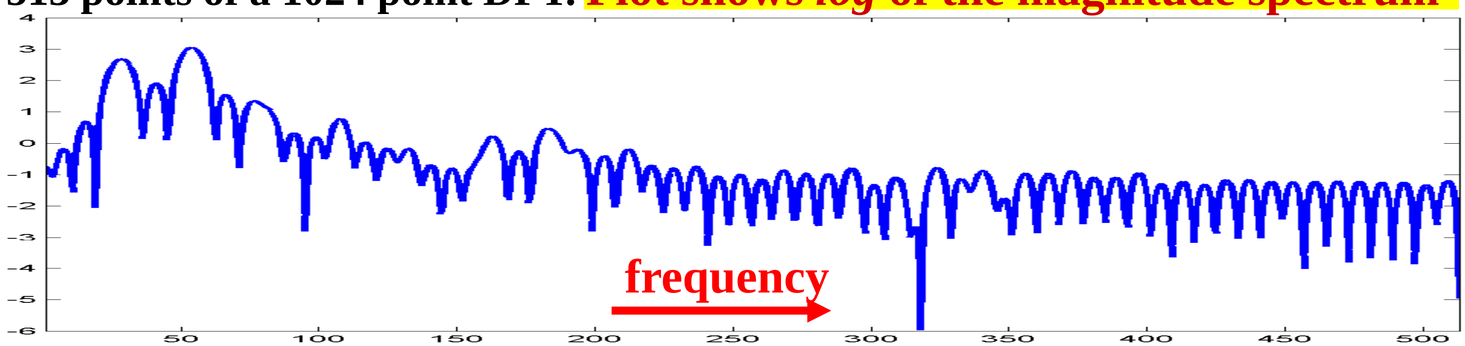
128 samples from a speech signal sampled at 16000 Hz



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

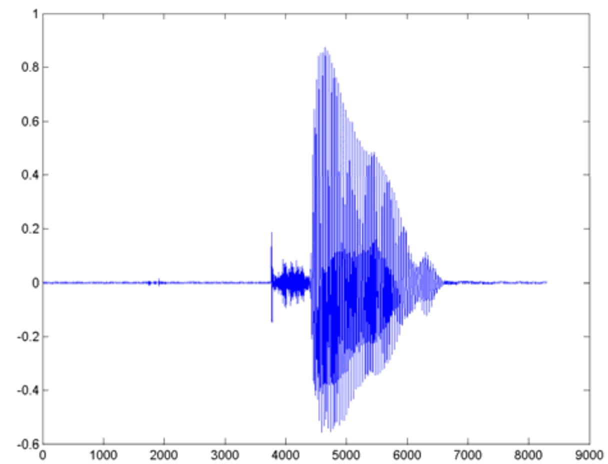


The first 513 points of a 1024 point DFT. **Plot shows *log* of the magnitude spectrum**

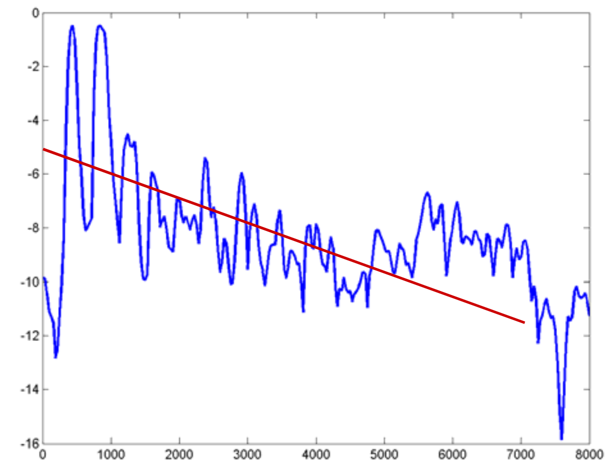


Preemphasizing a speech signal

- 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))

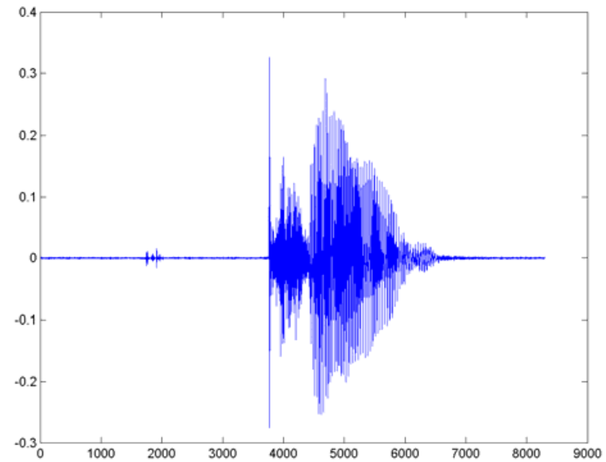


Preemphasizing a speech signal

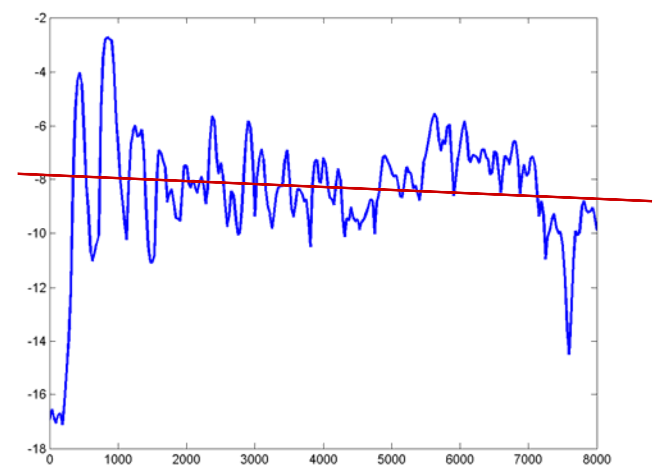
- This spectral tilt can be corrected by preemphasizing the signal
  - $s_{preemp}[n] = s[n] - \alpha * s[n-1]$
  - Typical value of $\alpha = 0.95$

- This is a form of differentiation that boosts high frequencies

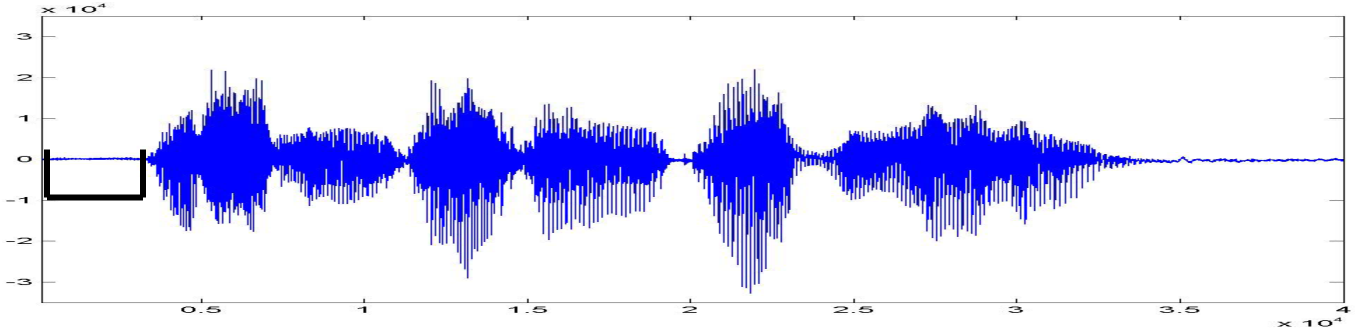
- This spectrum of the preemphasized signal has more horizontal trend
  - Good for linear prediction and other similar methods



Log(average(magnitude spectrum))

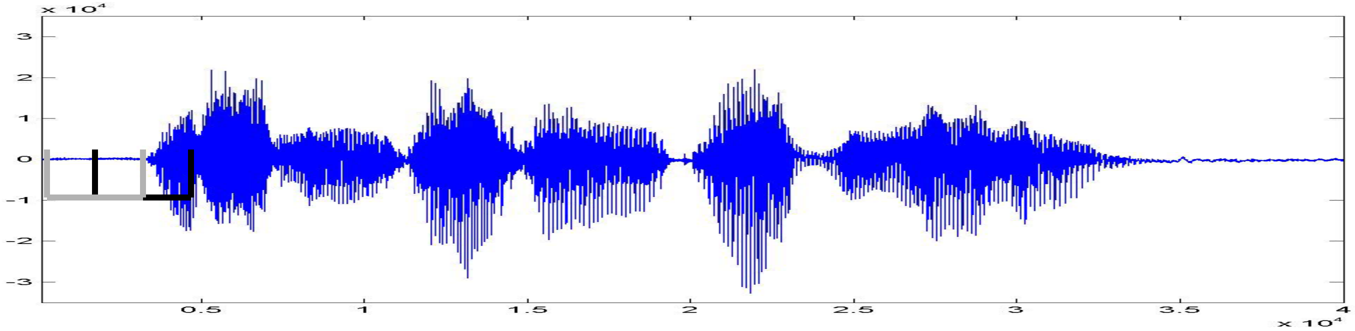


The process of parametrization



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

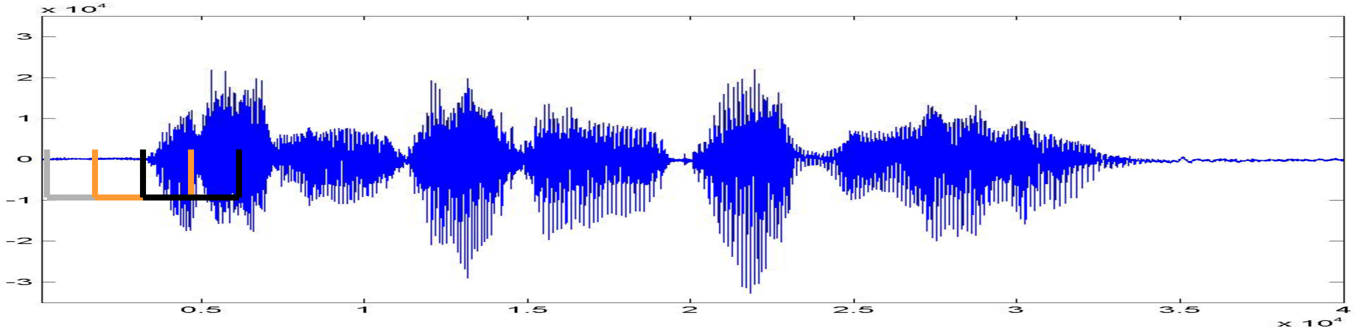
The process of parametrization



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

Adjacent segments typically overlap by 15 ms.

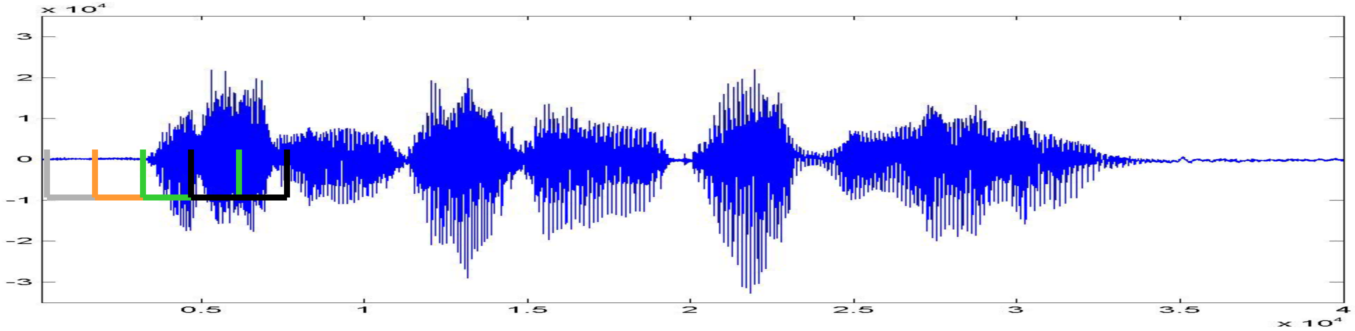
The process of parametrization



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

Adjacent segments typically overlap by 15 ms.

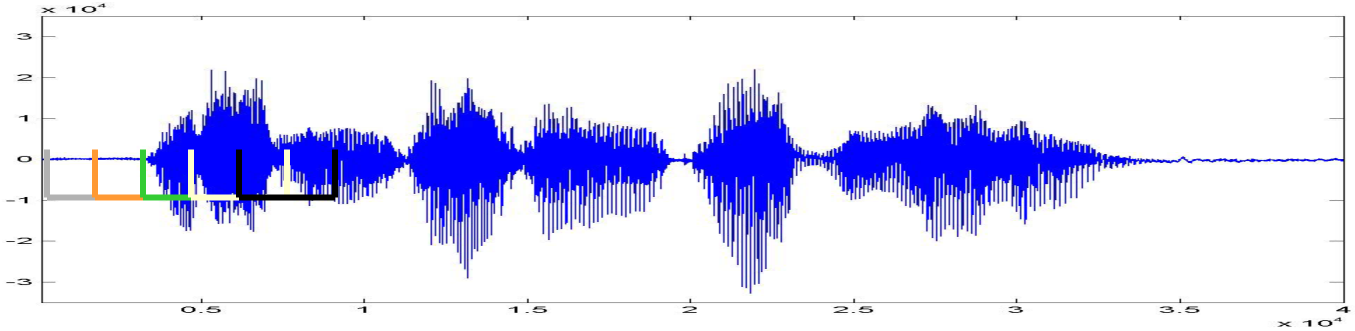
The process of parametrization



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

Adjacent segments typically overlap by 15 ms.

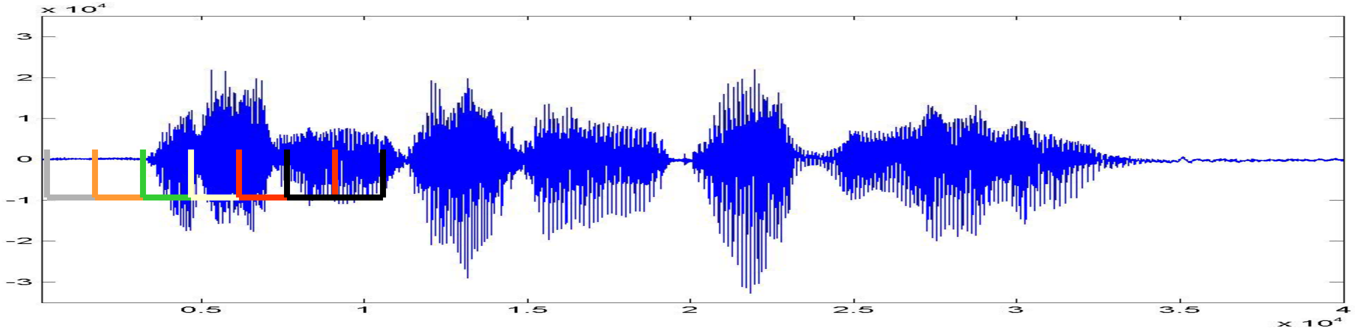
The process of parametrization



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

Adjacent segments typically overlap by 15 ms.

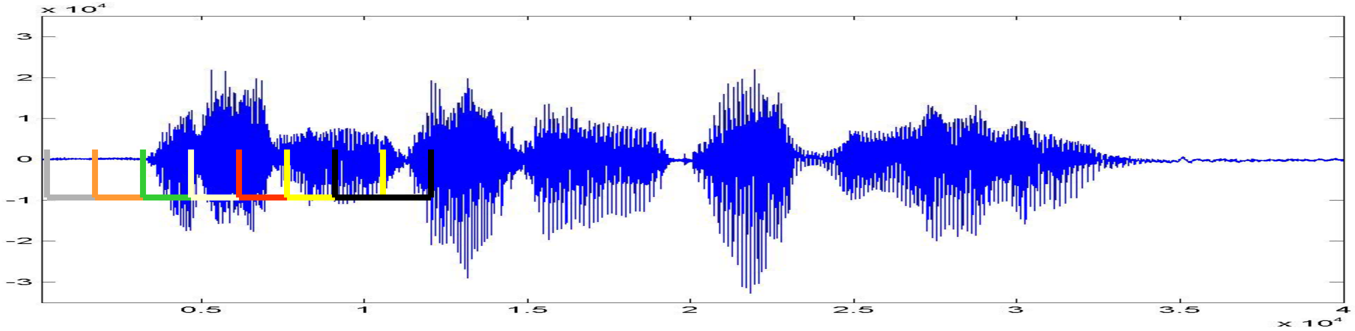
The process of parametrization



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

Adjacent segments typically overlap by 15 ms.

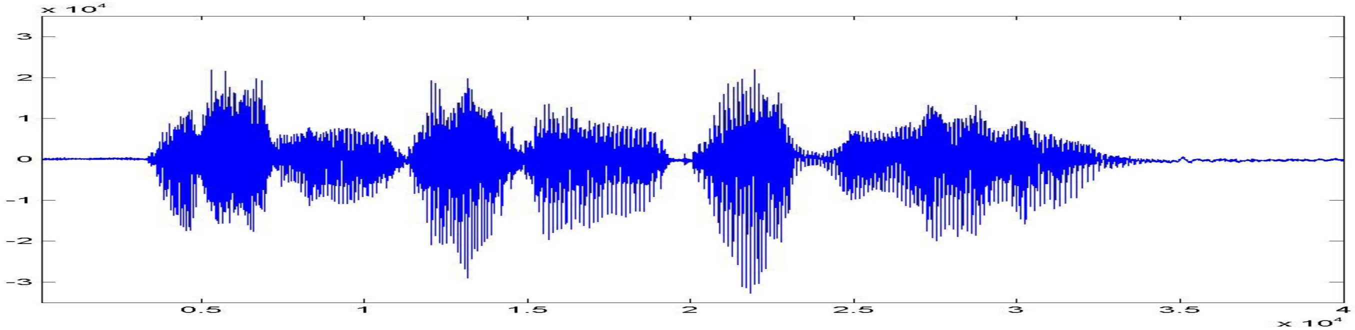
The process of parametrization

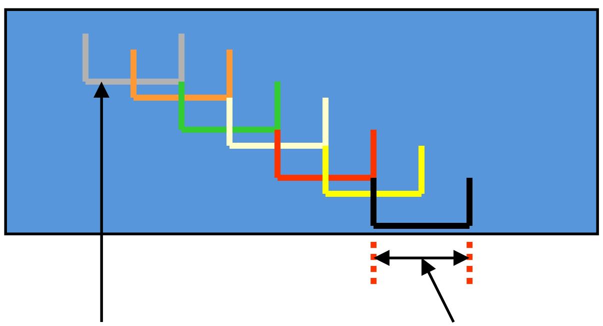


The signal is processed in segments. Segments are typically 25 ms wide.

Adjacent segments typically overlap by 15 ms.

The process of parametrization



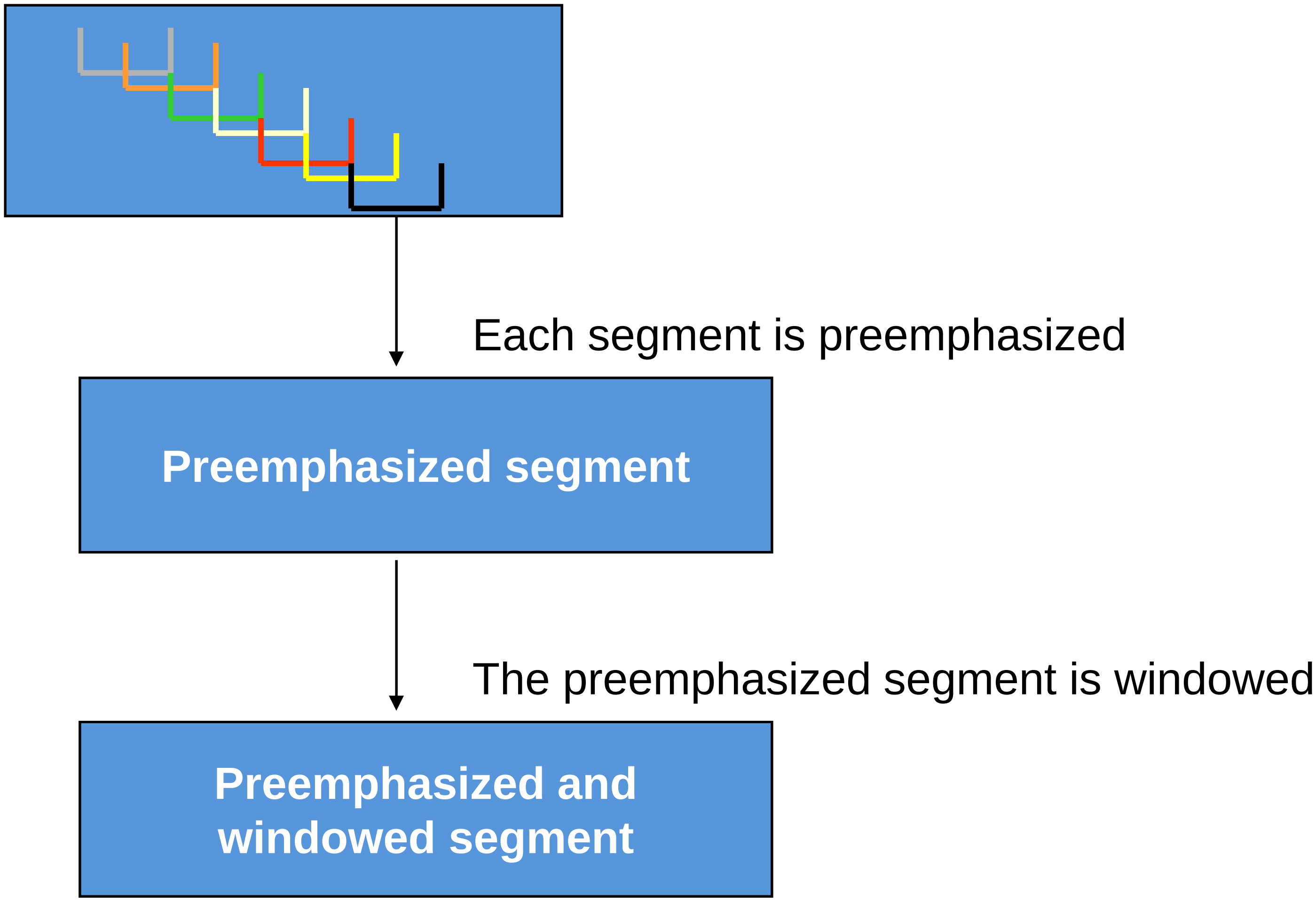


Segments shift every 10 milliseconds

Each segment is typically 20 or 25 milliseconds wide

Speech signals do not change significantly within this short time interval

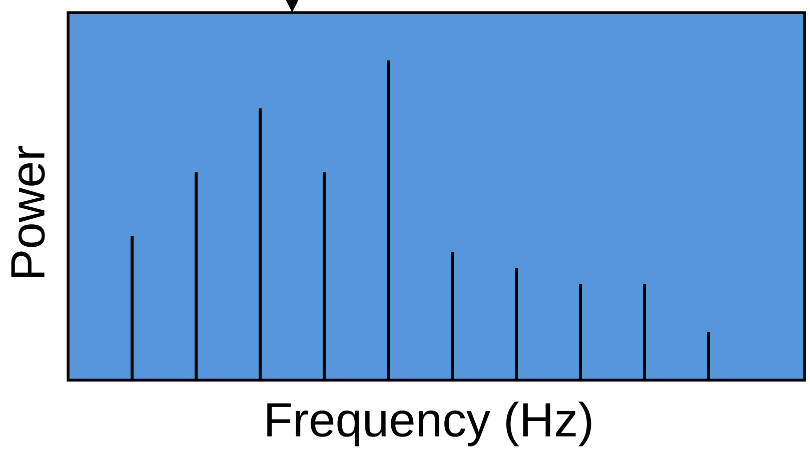
The process of parametrization



The process of parametrization

Preemphasized and windowed segment

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



= power spectrum

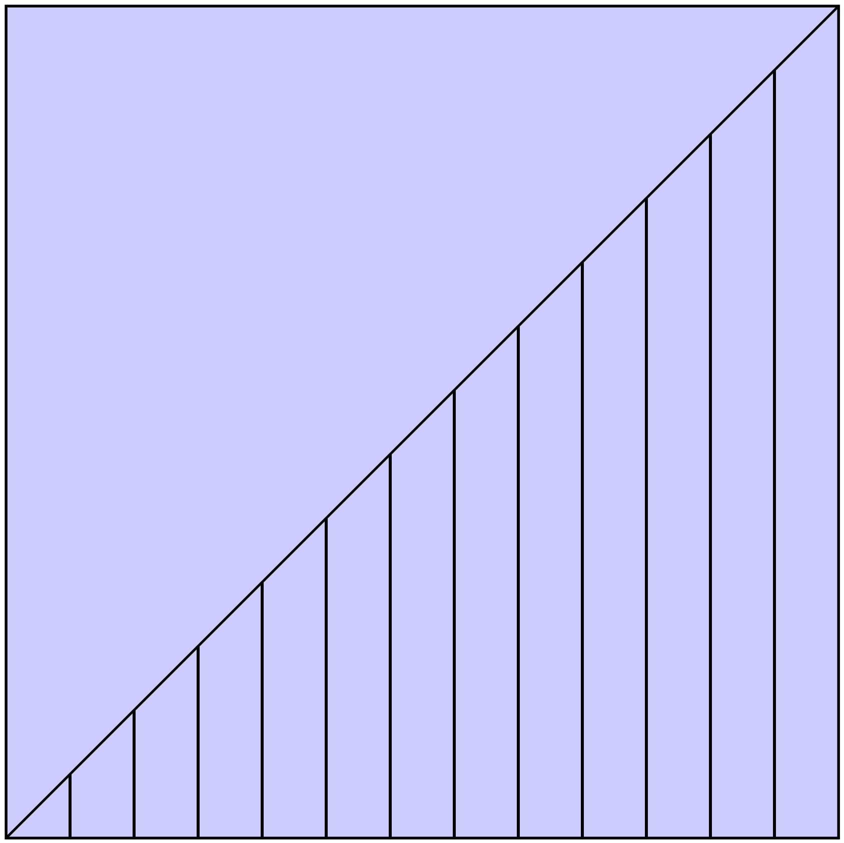
Auditory Perception

- 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

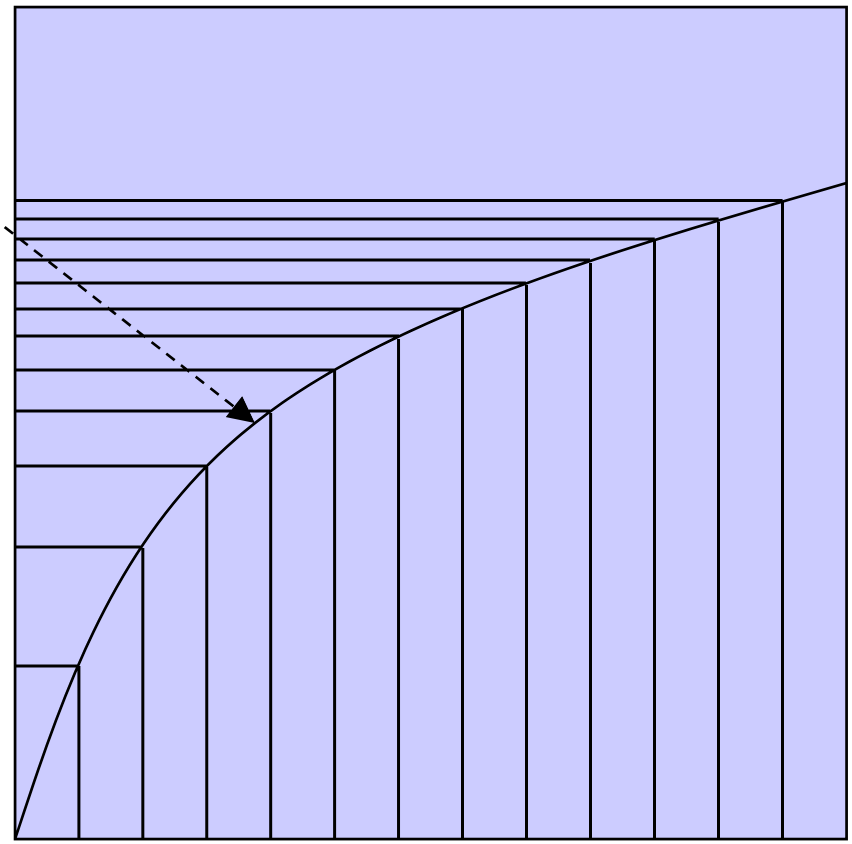


Linear frequency axis: equal increments of frequency at equal intervals

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



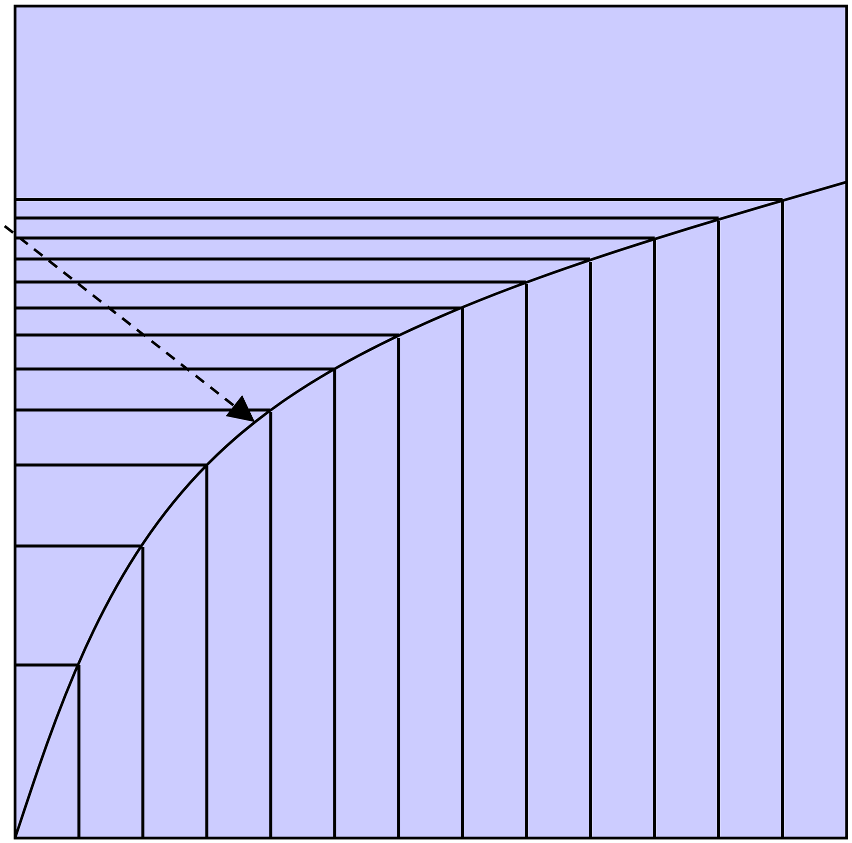
Linear frequency axis:
Sampled at uniform
intervals by an FFT

Warping the frequency axis

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

A standard warping function is the Mel warping function

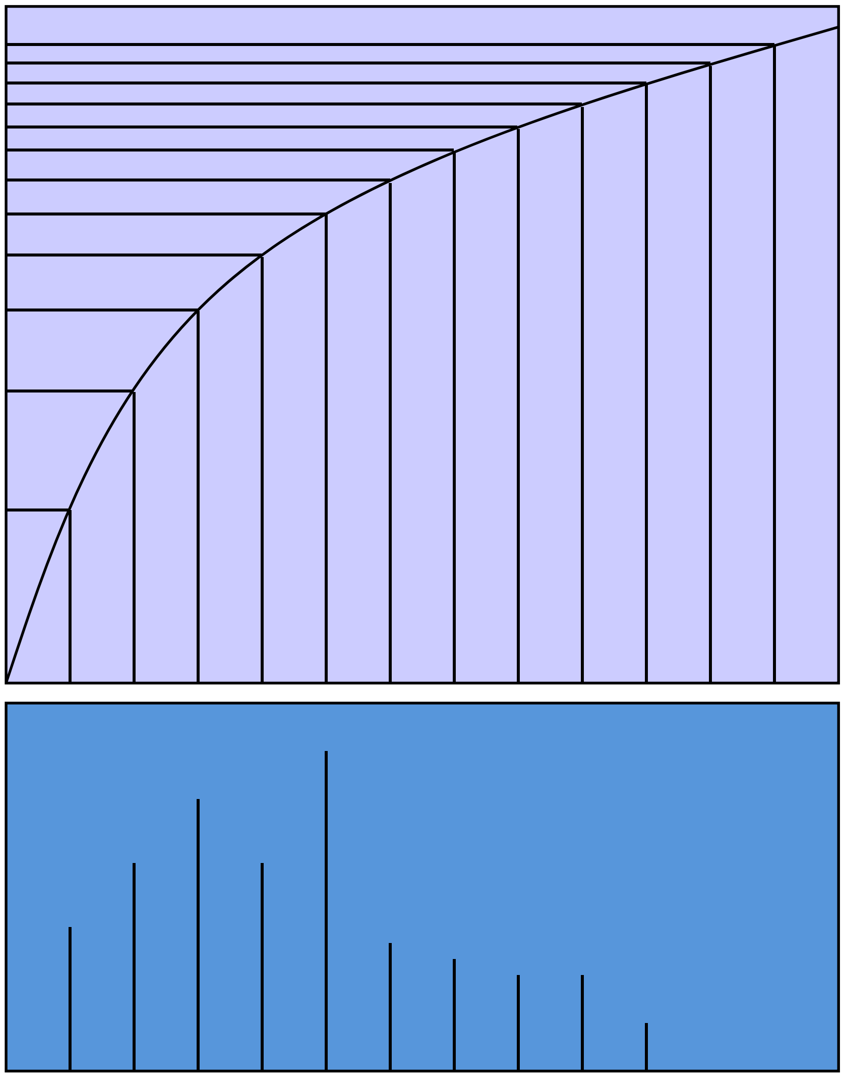
Warping function (based on studies of human hearing)



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

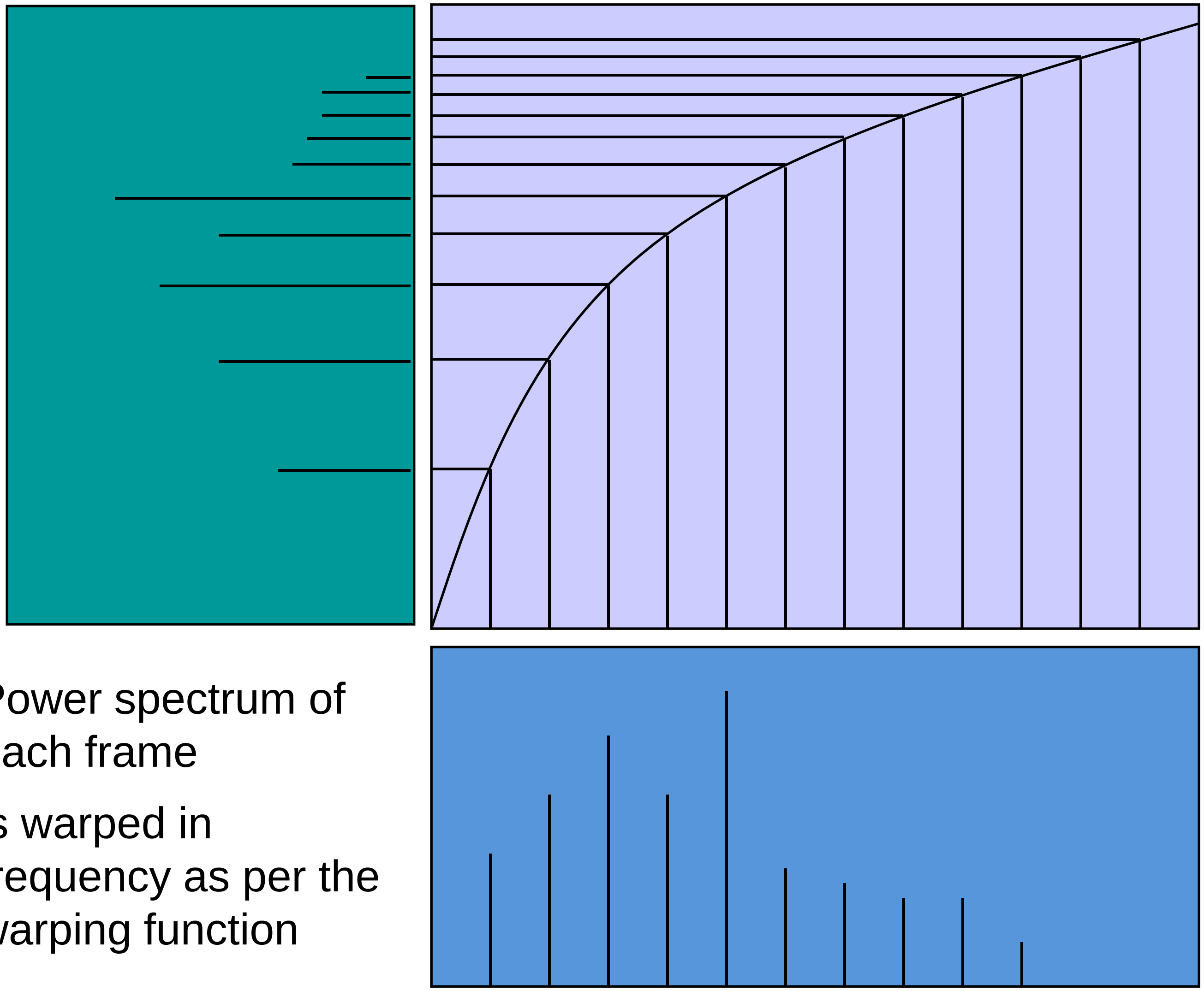
Linear frequency axis: Sampled at uniform intervals by an FFT

The process of parametrization



Power spectrum of each frame

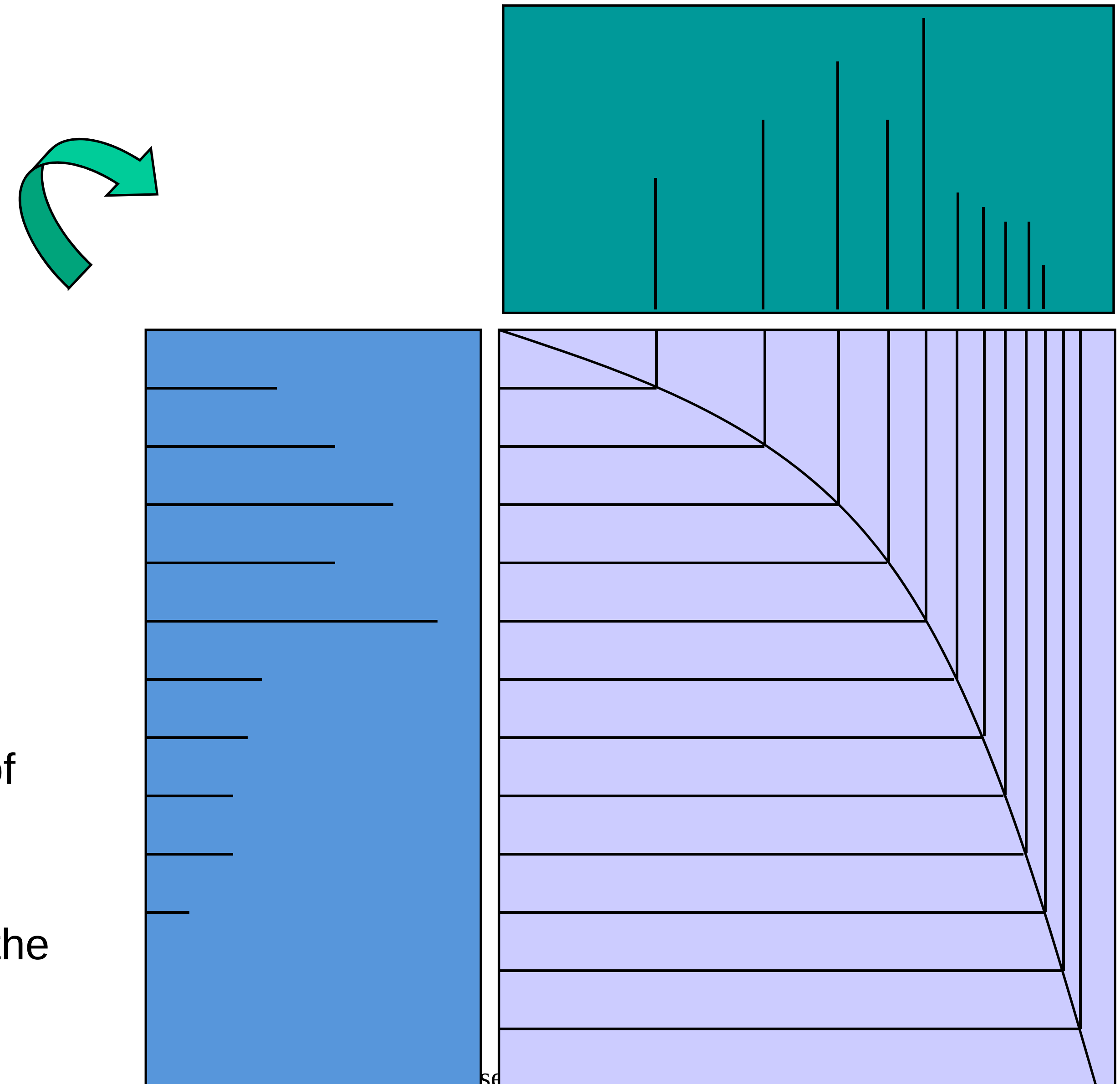
The process of parametrization



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

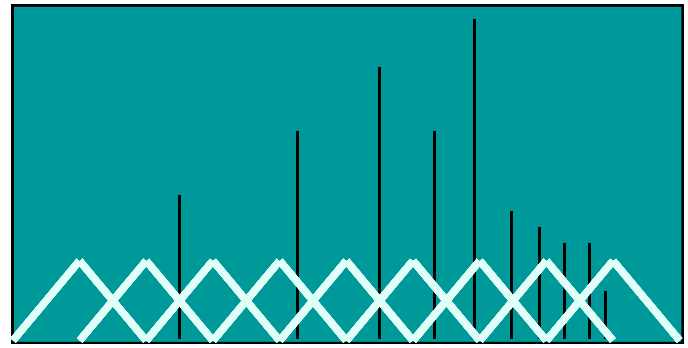
Filter Bank

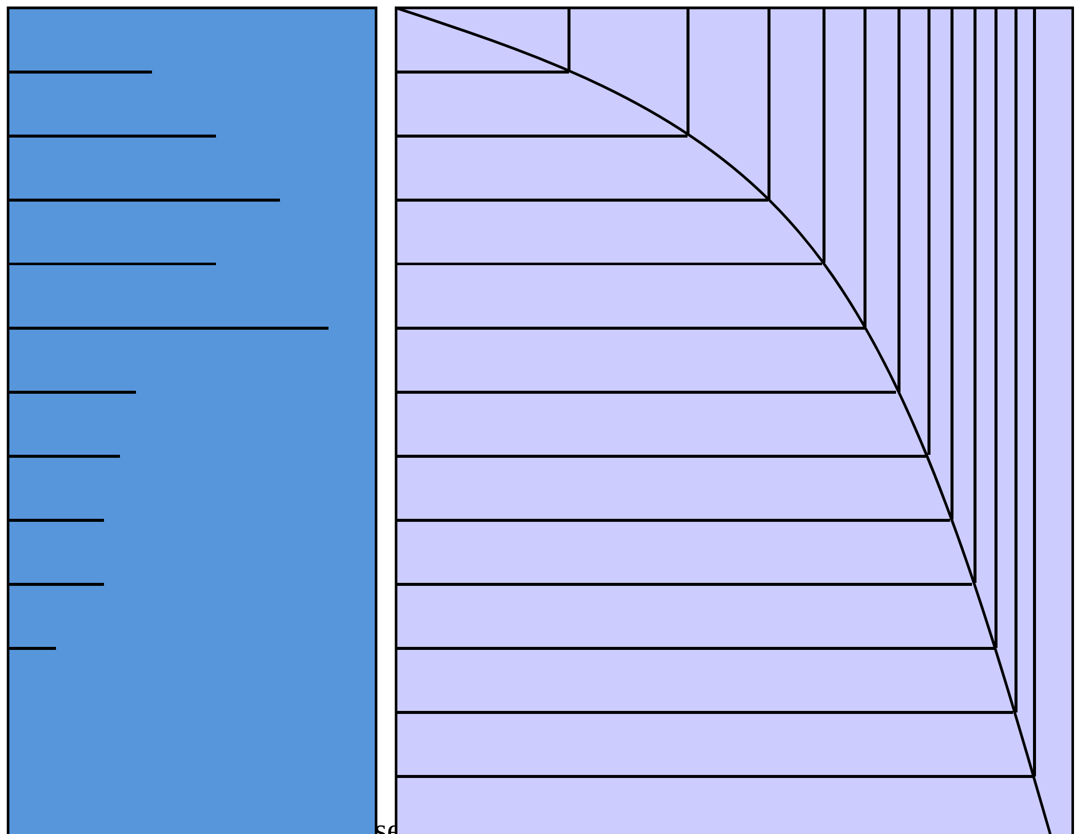
- Each hair cells in the human ear actually responds to a *band* of frequencies, with a peak response at a particular frequency

- 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%

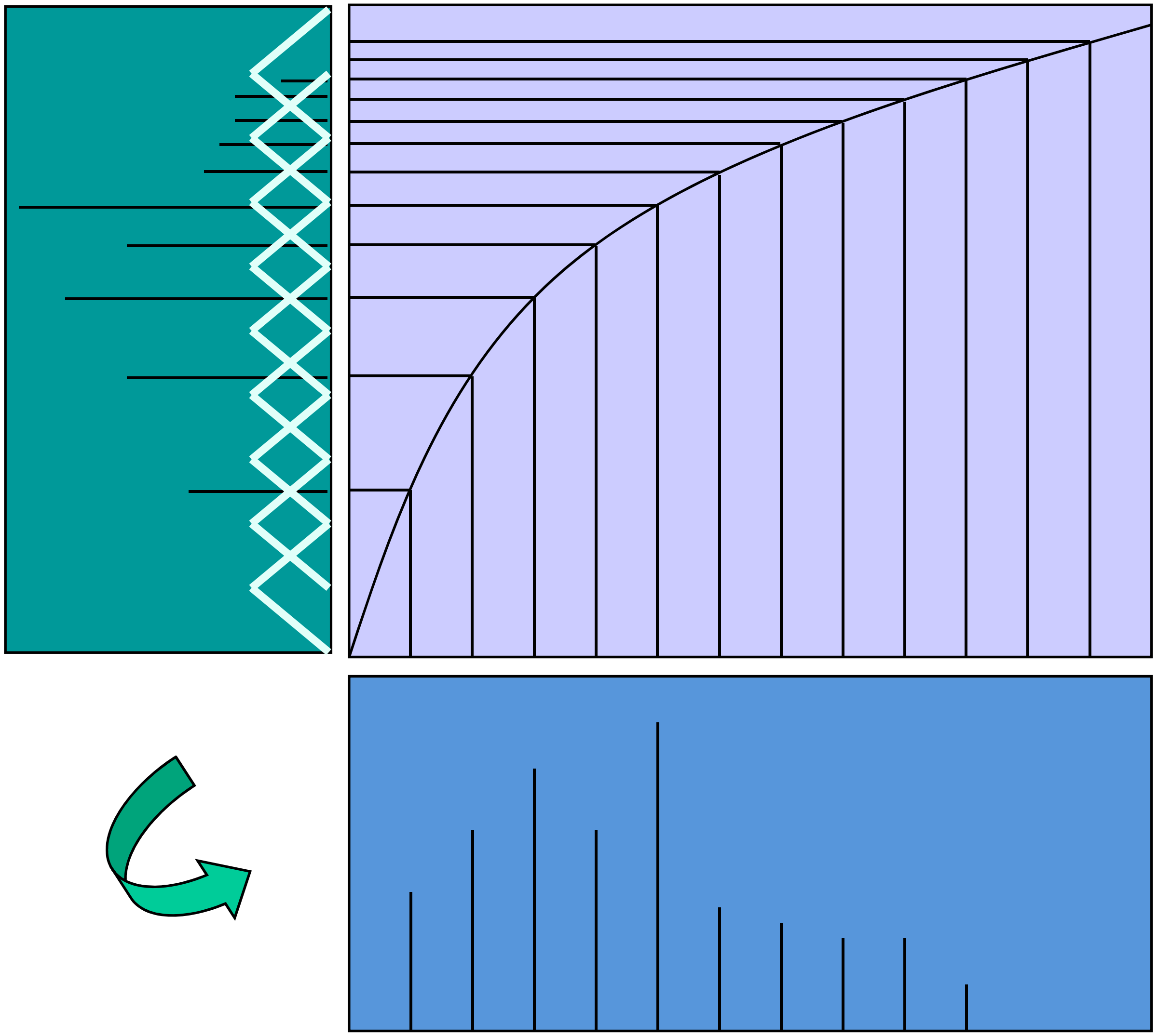




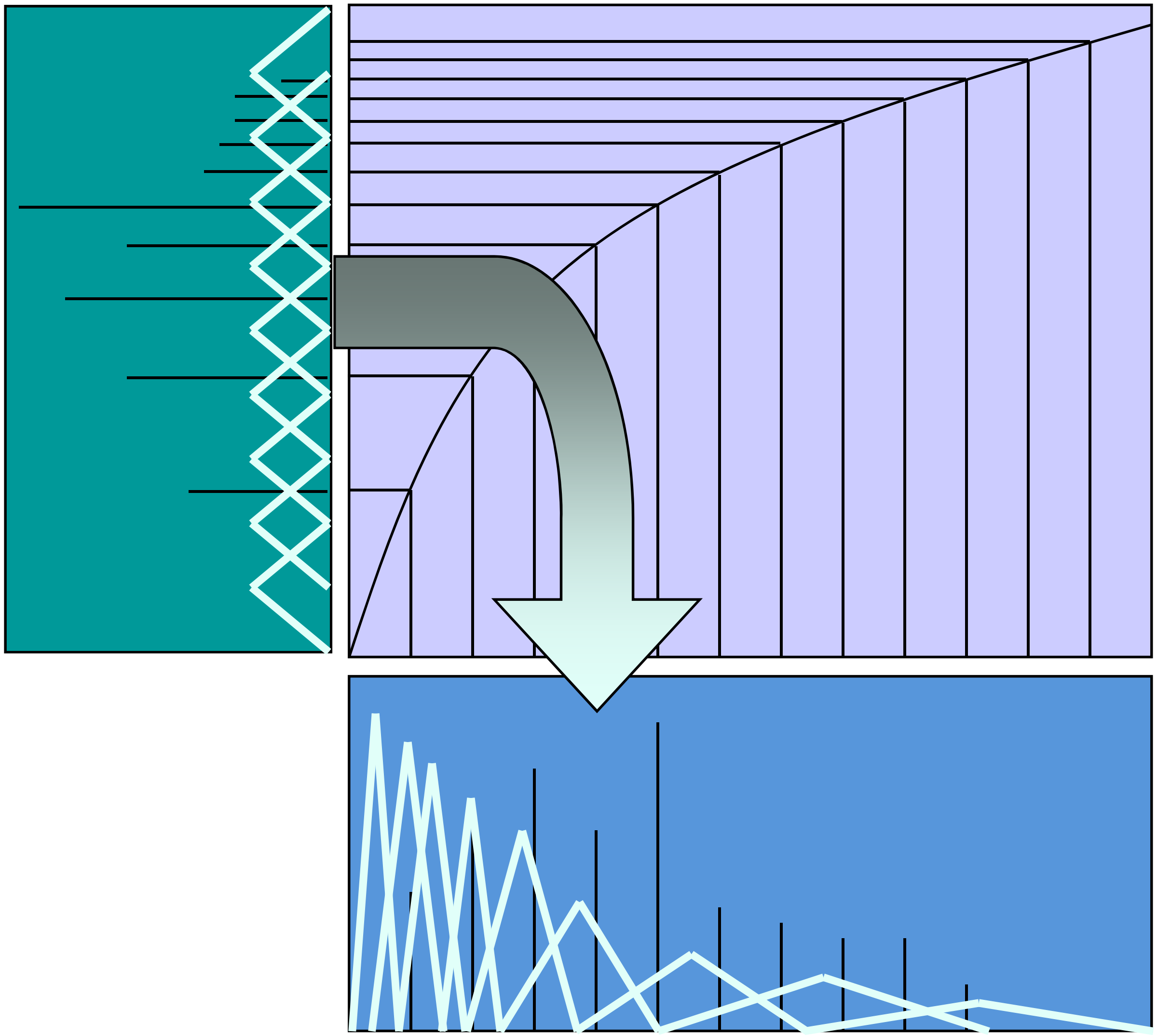
Power spectrum of each frame

is warped in frequency as per the warping function

The process of parametrization

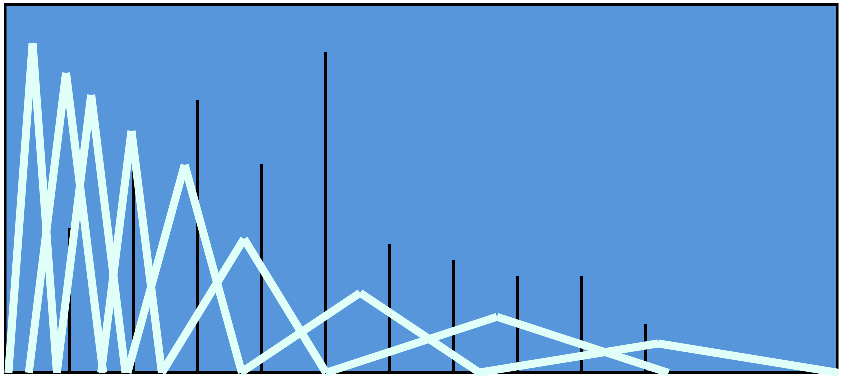


The process of parametrization



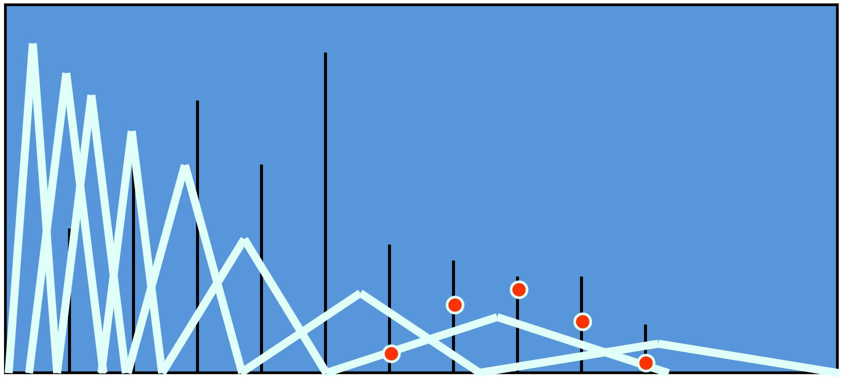
The process of parametrization

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

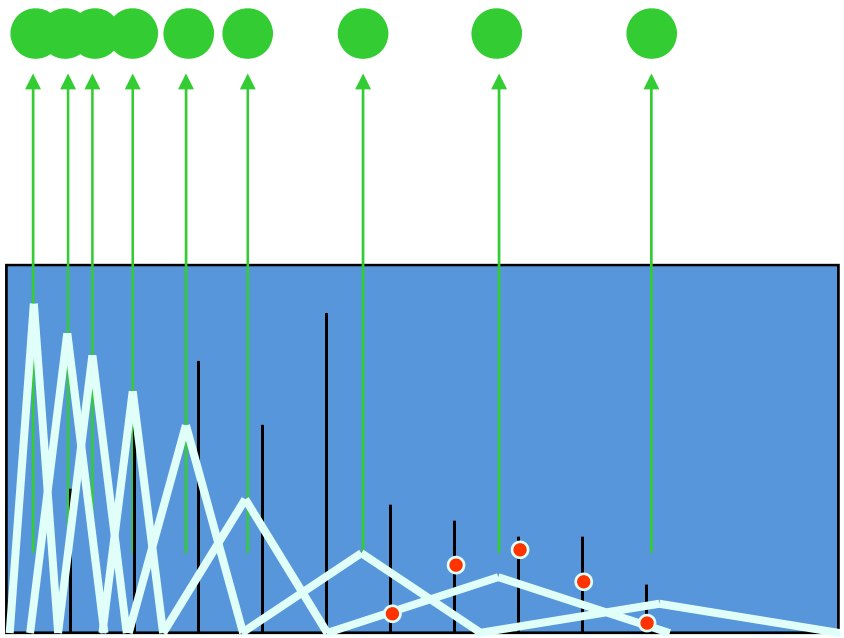


The process of parametrization

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



The process of parametrization

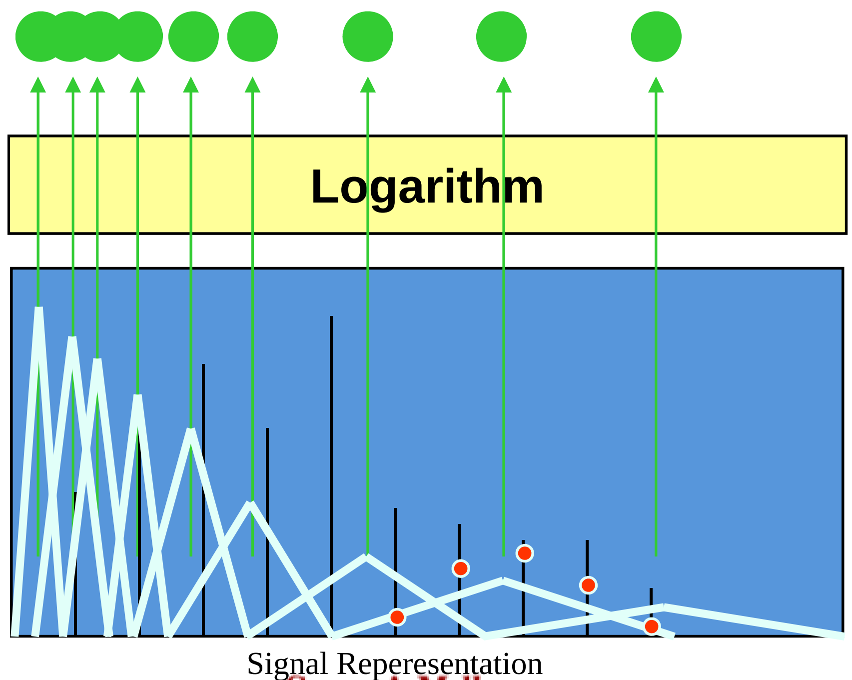


All weighted spectral values for each filter are integrated (added), giving one value per filter

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

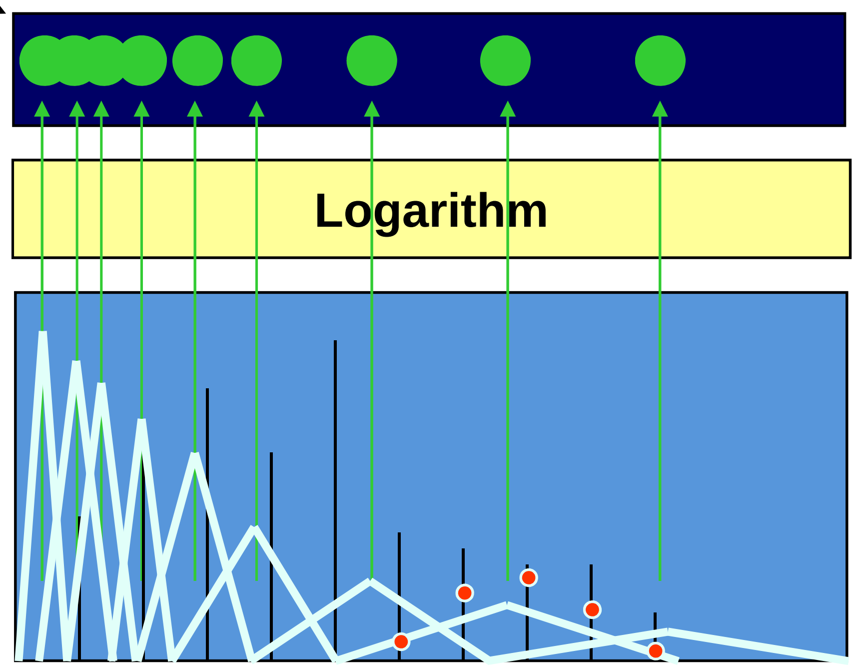


Compress Values

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

The process of parametrization

Log Mel spectrum



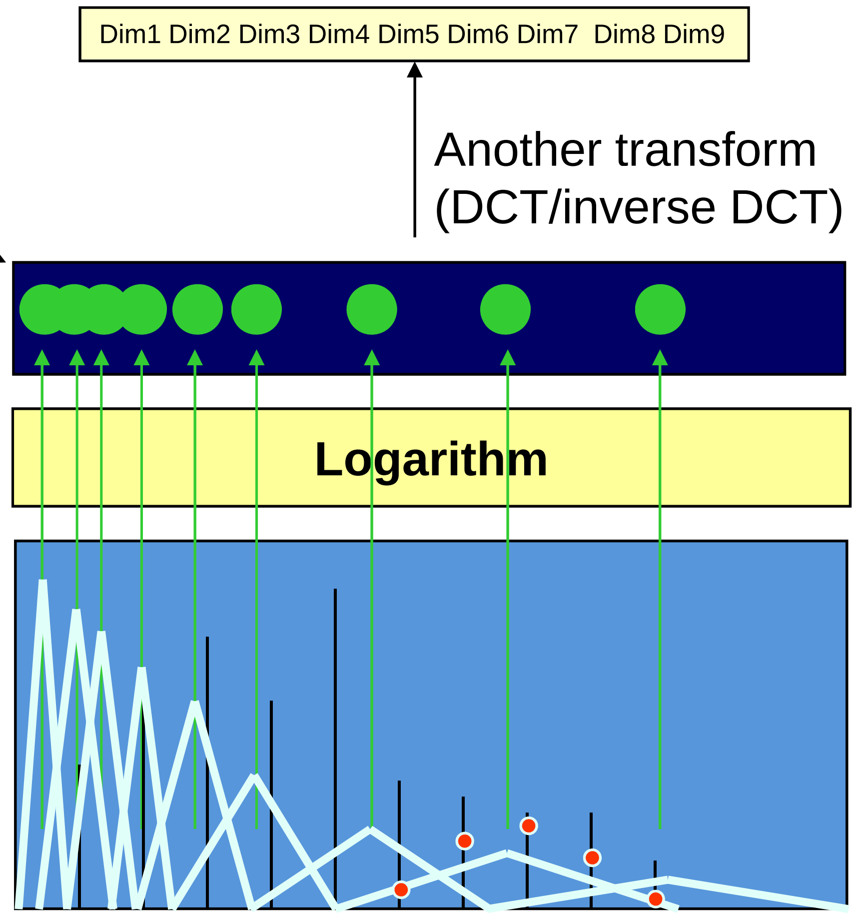
Compress Values

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

The process of parametrization

Log Mel spectrum

Another transform
(DCT/inverse DCT)



Compress Values

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

Signal Reperesentation

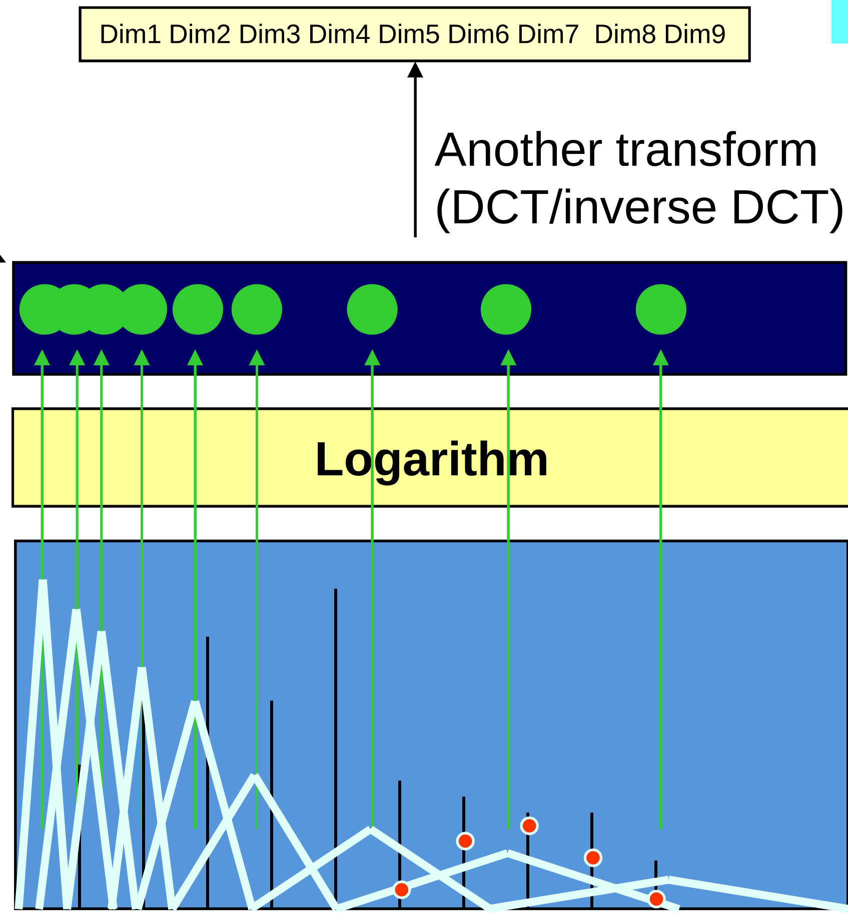
The process of parametrization

The sequence is truncated
(typically after 13 values)

Dimensionality reduction

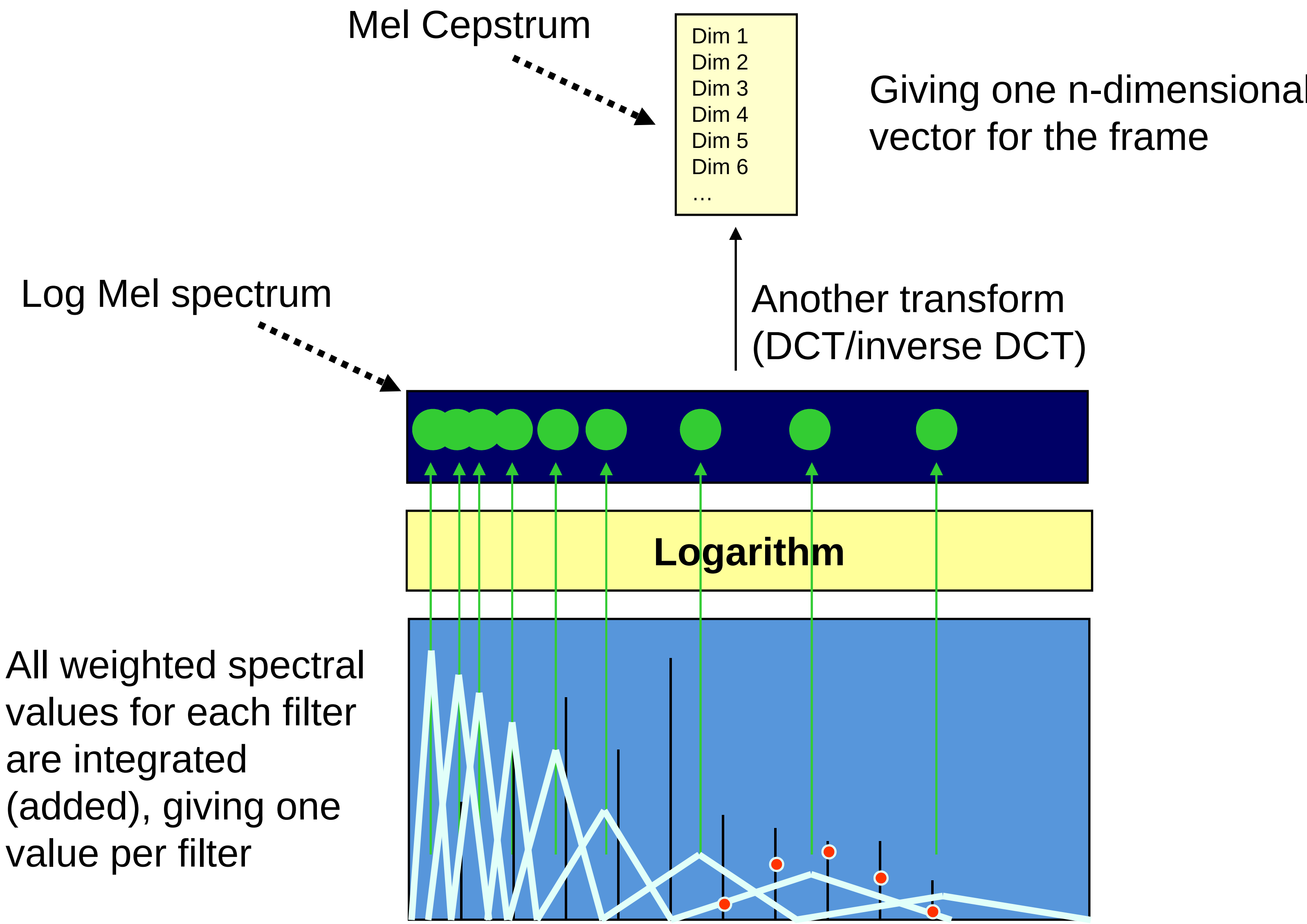
Log Mel spectrum

Another transform
(DCT/inverse DCT)

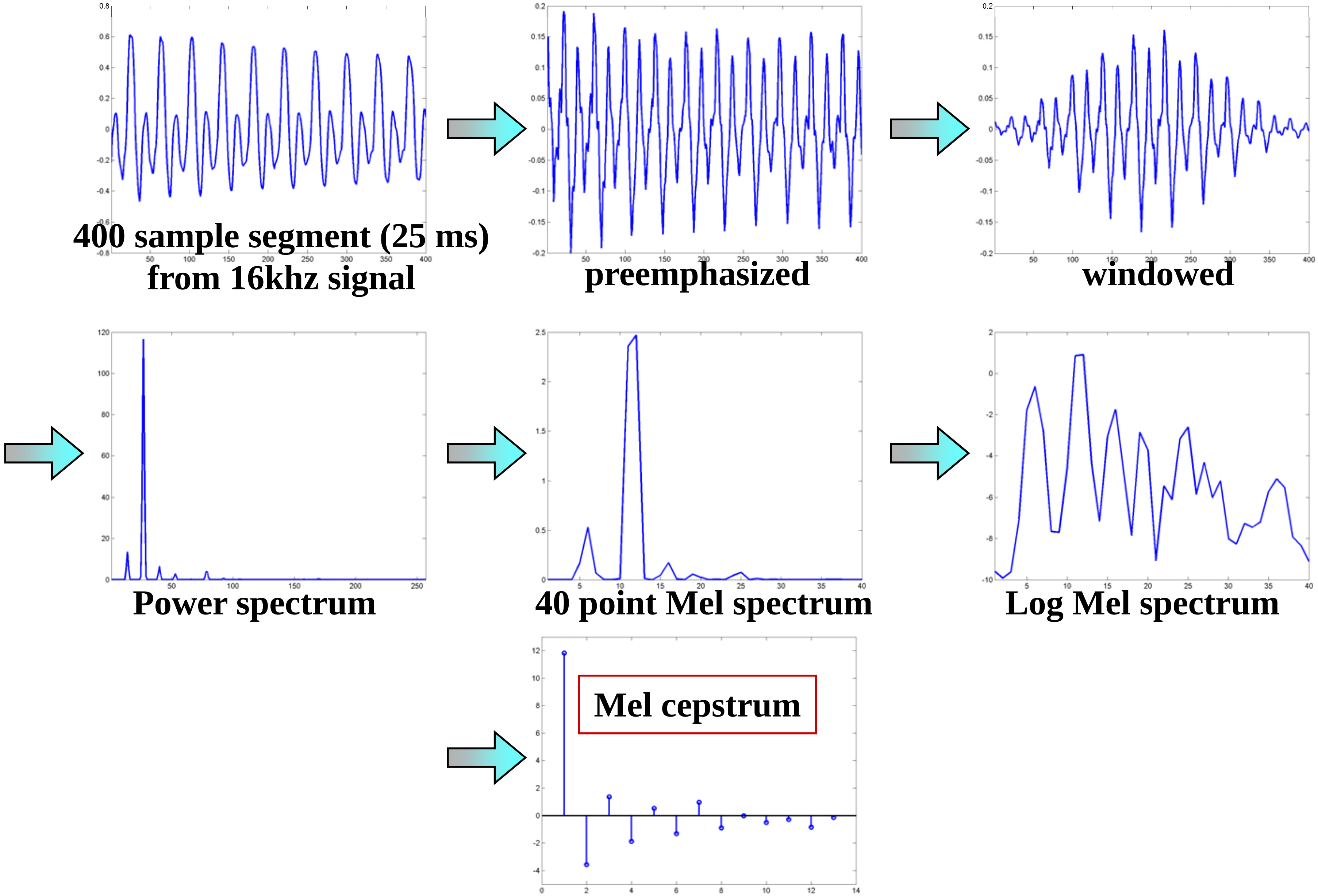


All weighted spectral values for each filter are integrated (added), giving one value per filter

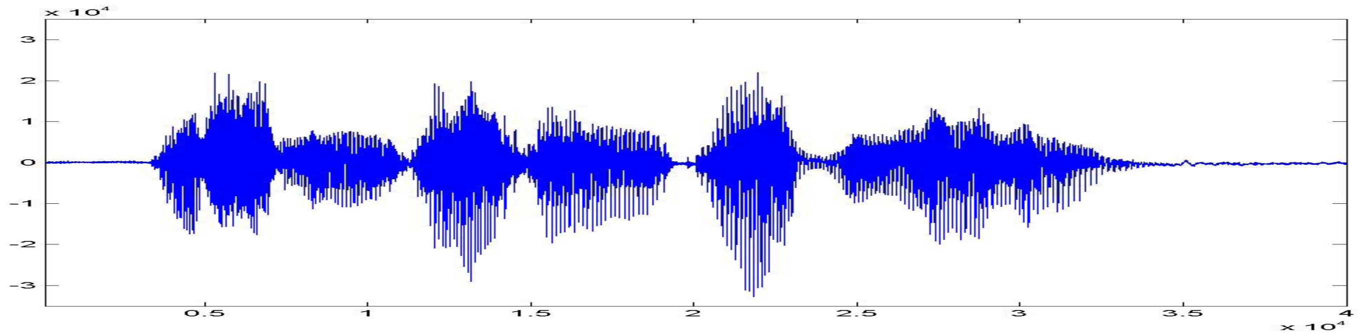
The process of parametrization

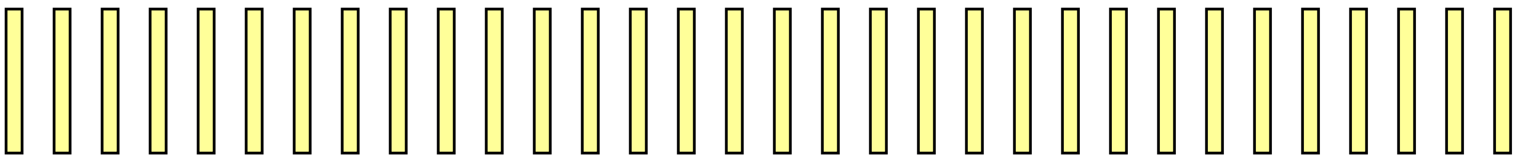


An example segment



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

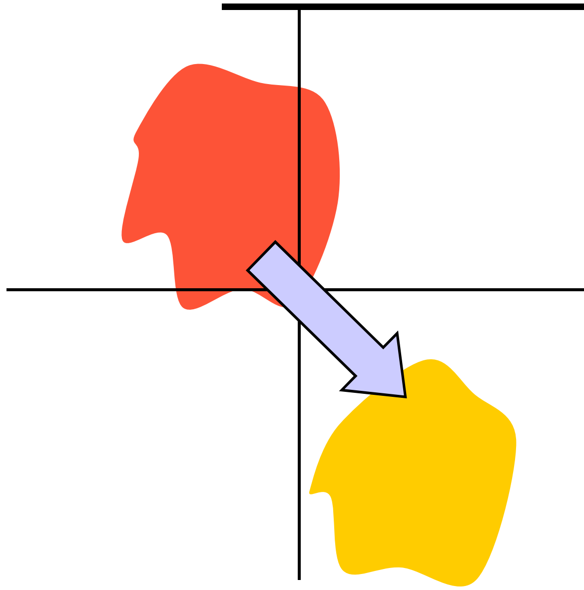
Variations to the basic theme

- 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

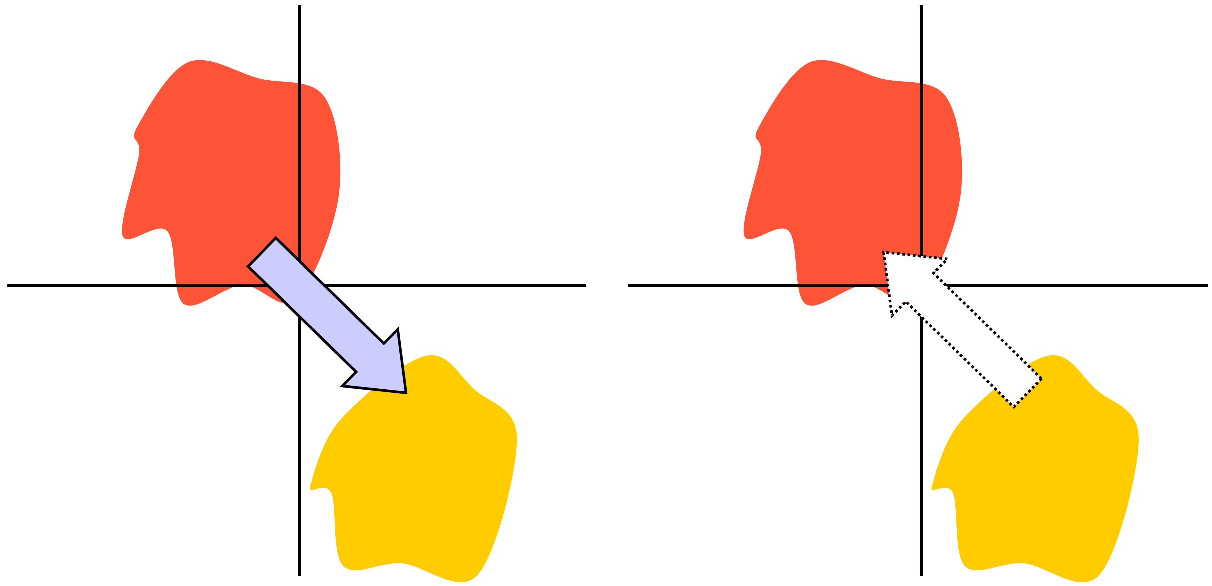
- Microphone characteristics modify the spectral characteristics of the captured signal
  - They change the value of the cepstra
- Noise too modifies spectral characteristics
- As do speaker variations
- All of these change the distribution of the cepstra

Effect of Speaker Variations, Microphone Variations, Noise etc.



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

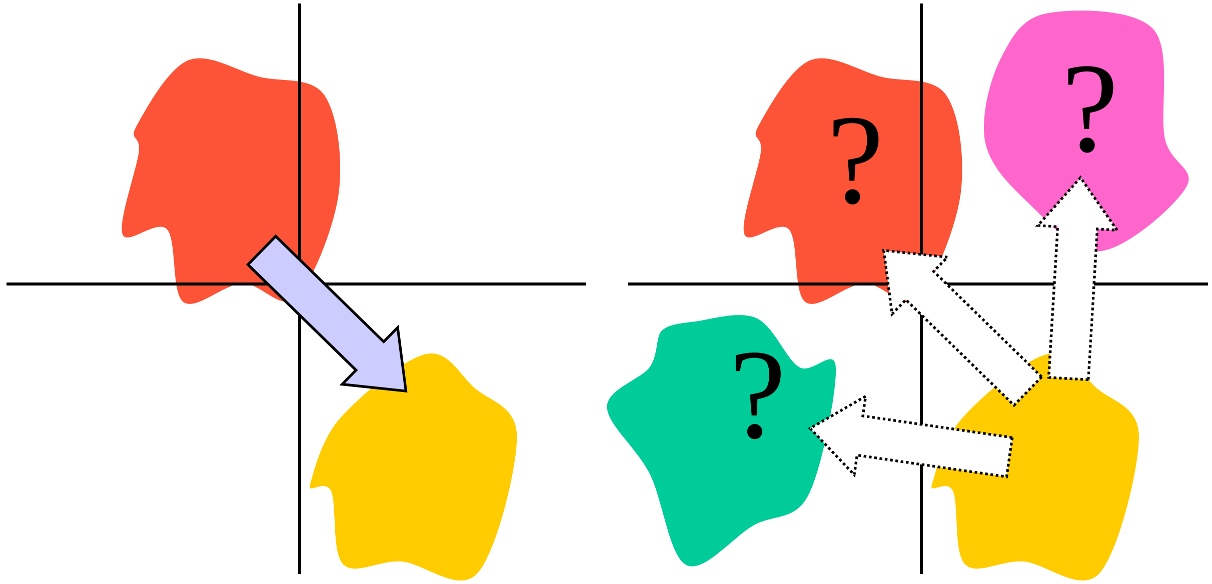
Ideal Correction for Variations



- 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.

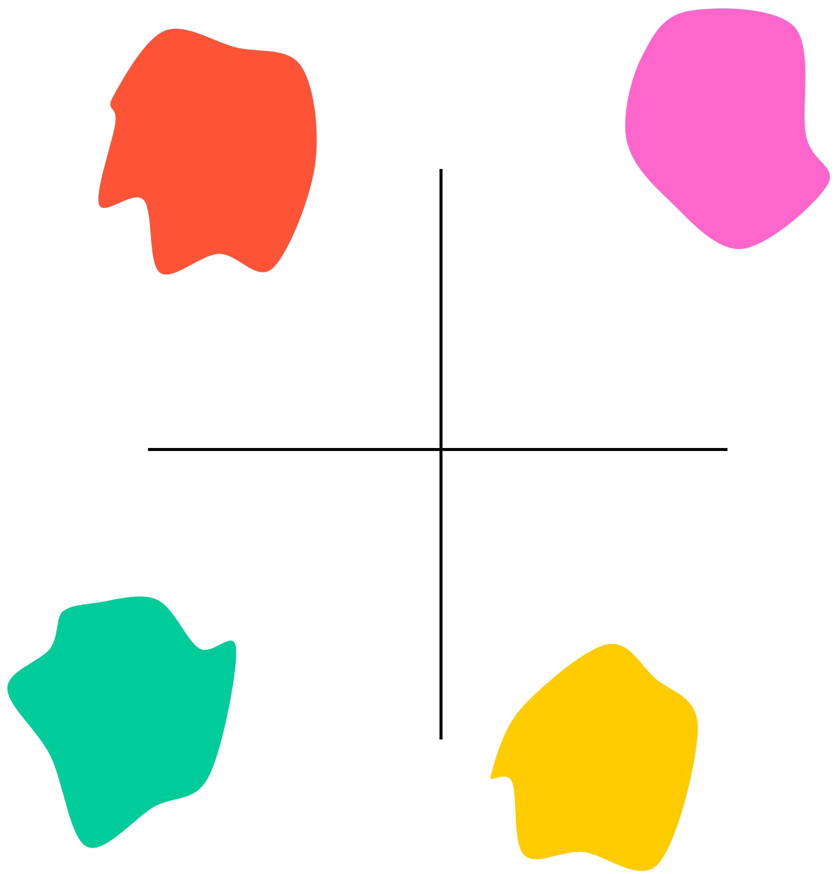


- Noise, channel and speaker variations change the *distribution* of cepstral values

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

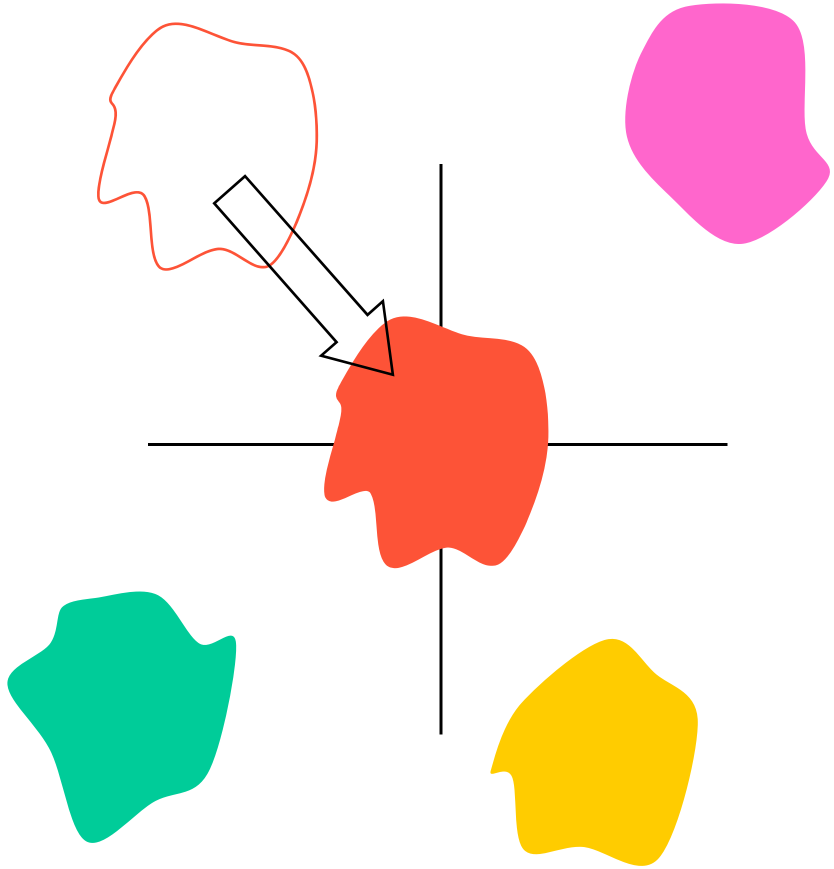
- Unfortunately, the precise position of the distributions of the “good” speech is hard to know

Solution: Move all distributions to a "standard" location



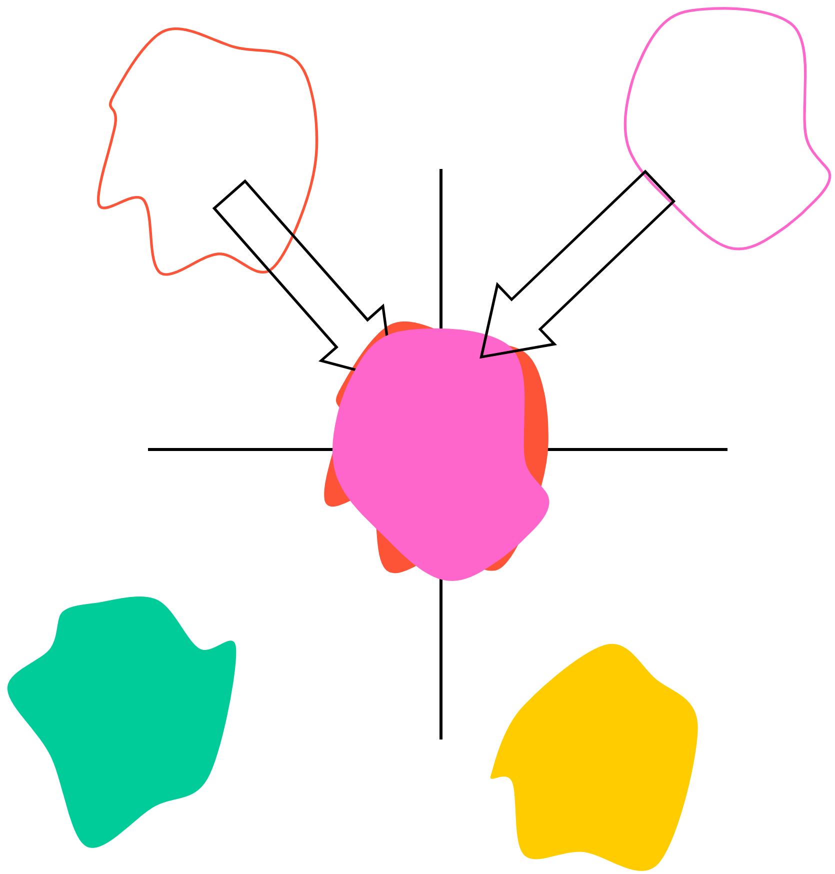
- "Move" all utterances to have a mean of 0
- This ensures that all the data is centered at 0
  - Thereby eliminating *some* of the mismatch

Solution: Move all distributions to a "standard" location



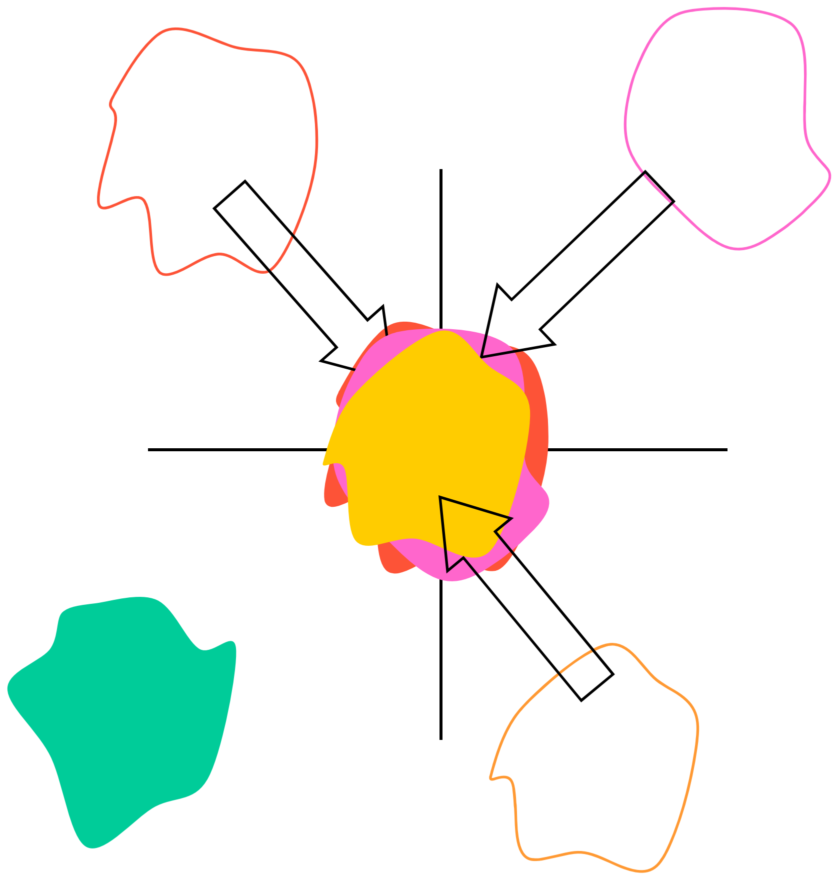
- "Move" all utterances to have a mean of 0
- This ensures that all the data is centered at 0
  - Thereby eliminating *some* of the mismatch

Solution: Move all distributions to a "standard" location



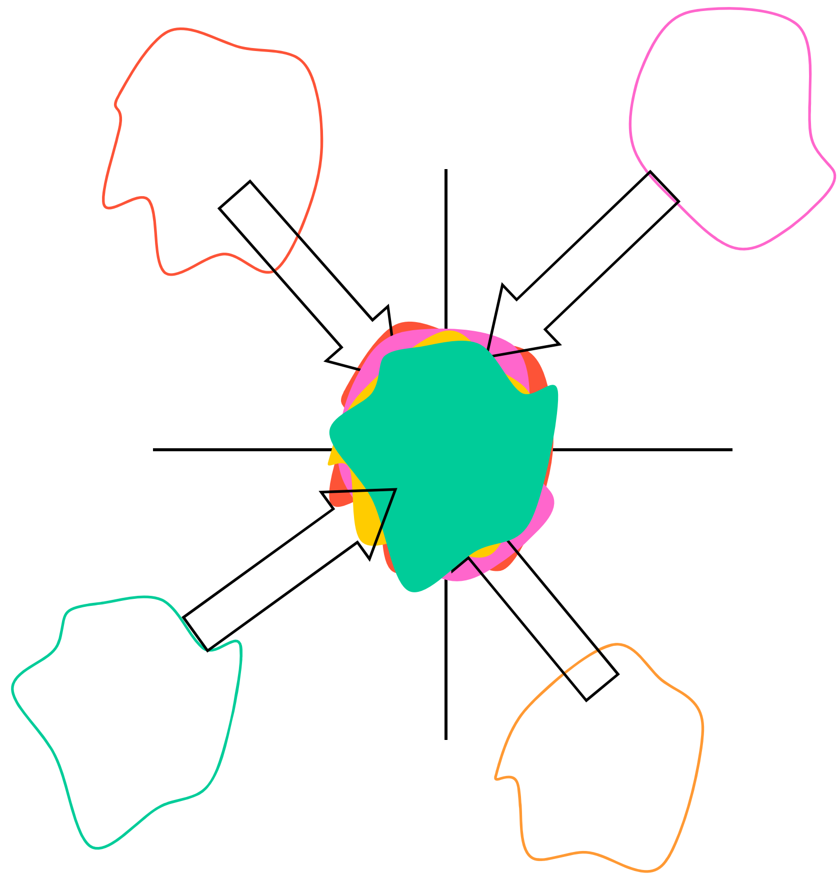
- "Move" all utterances to have a mean of 0
- This ensures that all the data is centered at 0
  - Thereby eliminating *some* of the mismatch

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

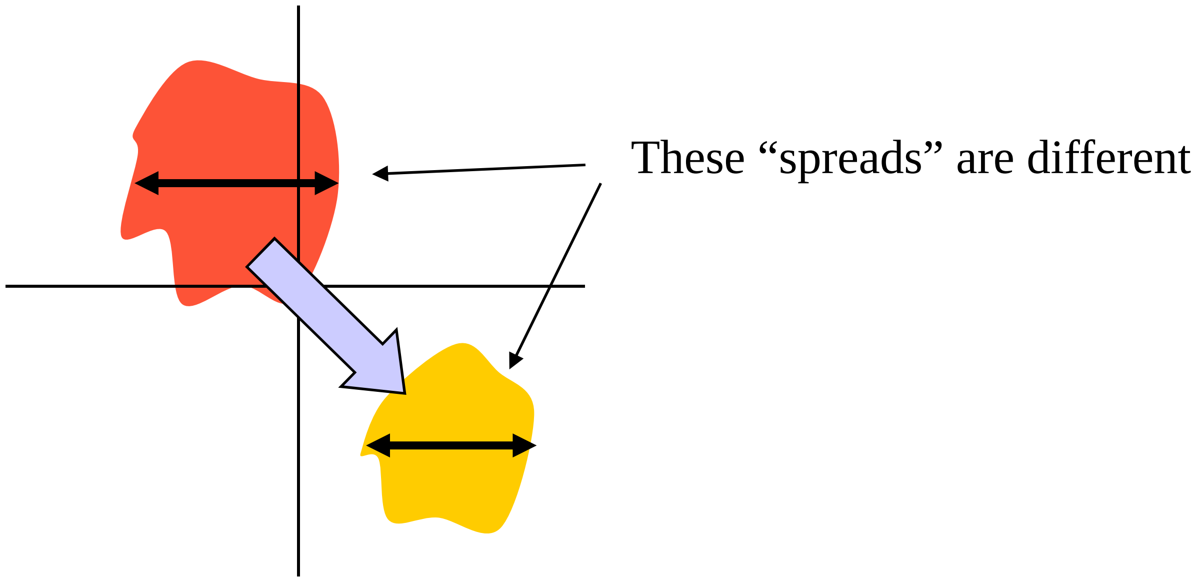
- For each utterance encountered (both in "training" and in "testing")
- Compute the mean of all cepstral vectors

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

- 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

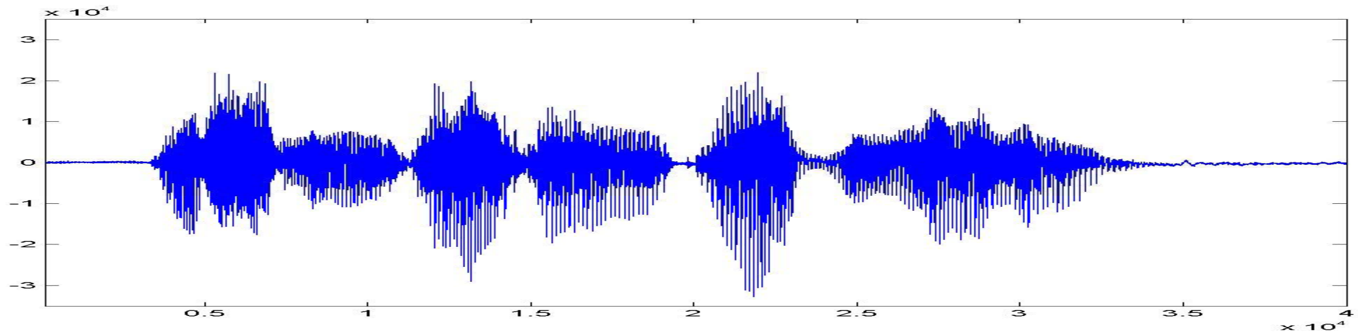
Temporal Variations

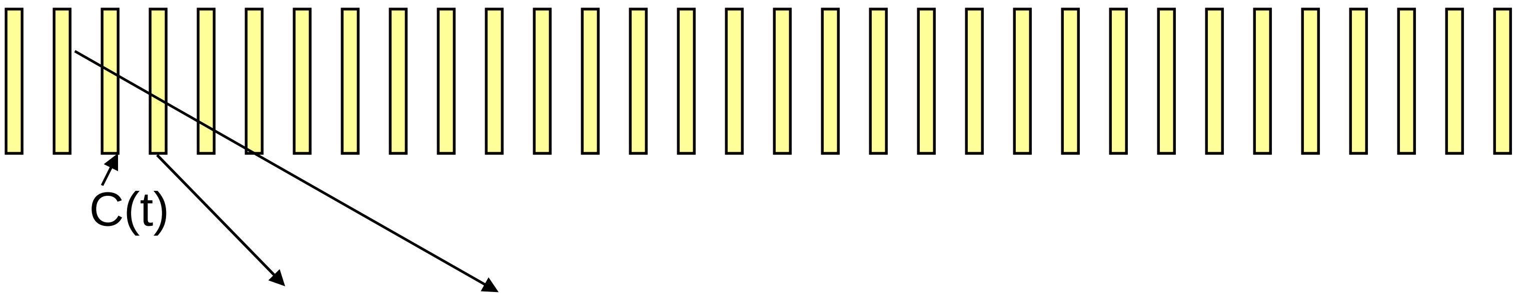
- The cepstral vectors capture instantaneous information only
  - Or, more precisely, current spectral structure within the analysis window
- Phoneme identity resides not just in the snapshot information, but also in the temporal structure
  - Manner in which these values change with time
  - Most characteristic 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



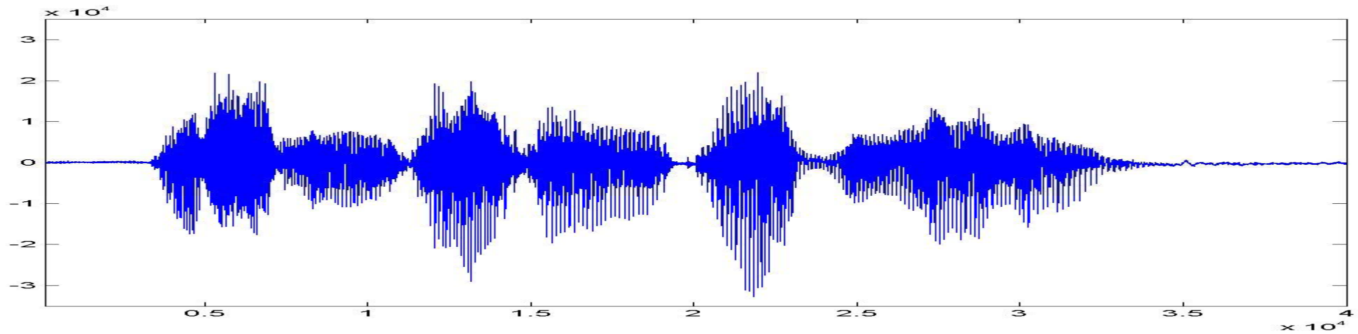


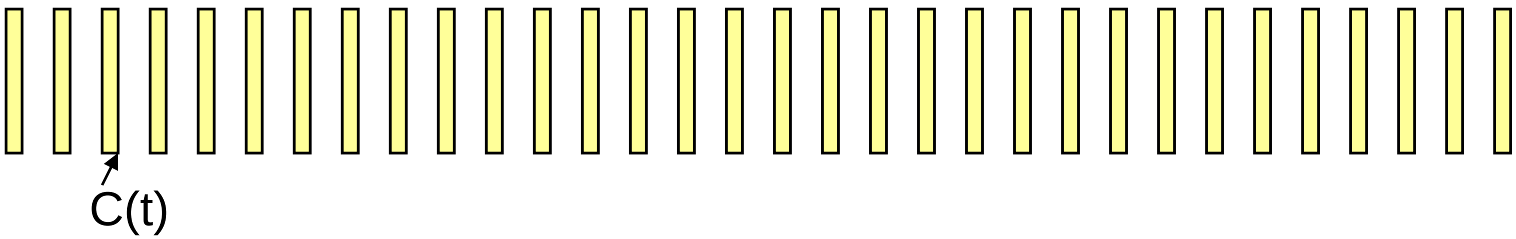
$$\Delta c(t)=c(t+\tau)-c(t-\tau)$$

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

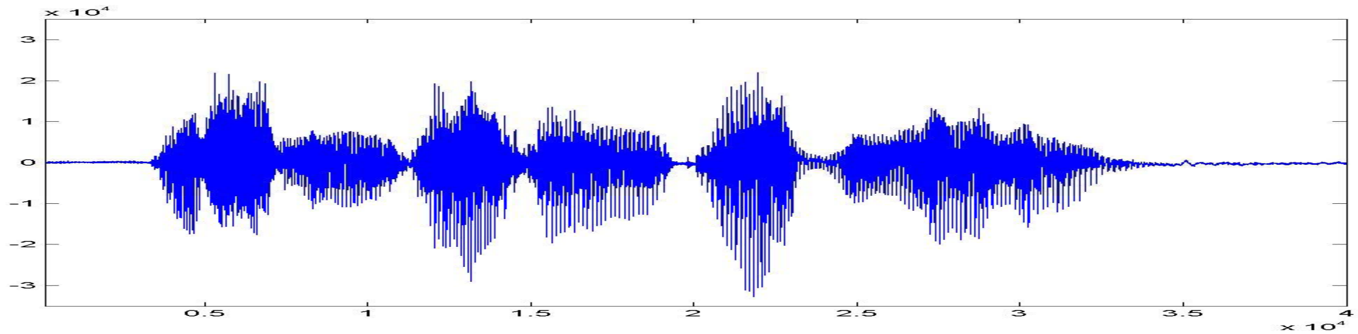


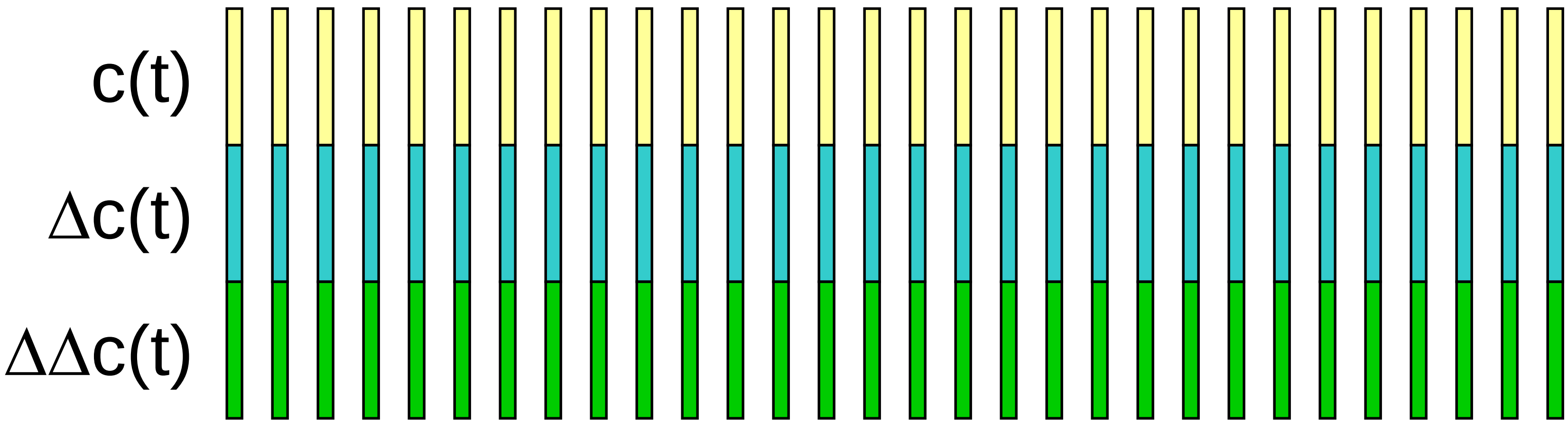


$$\Delta c(t)=c(t+\tau)-c(t-\tau)$$

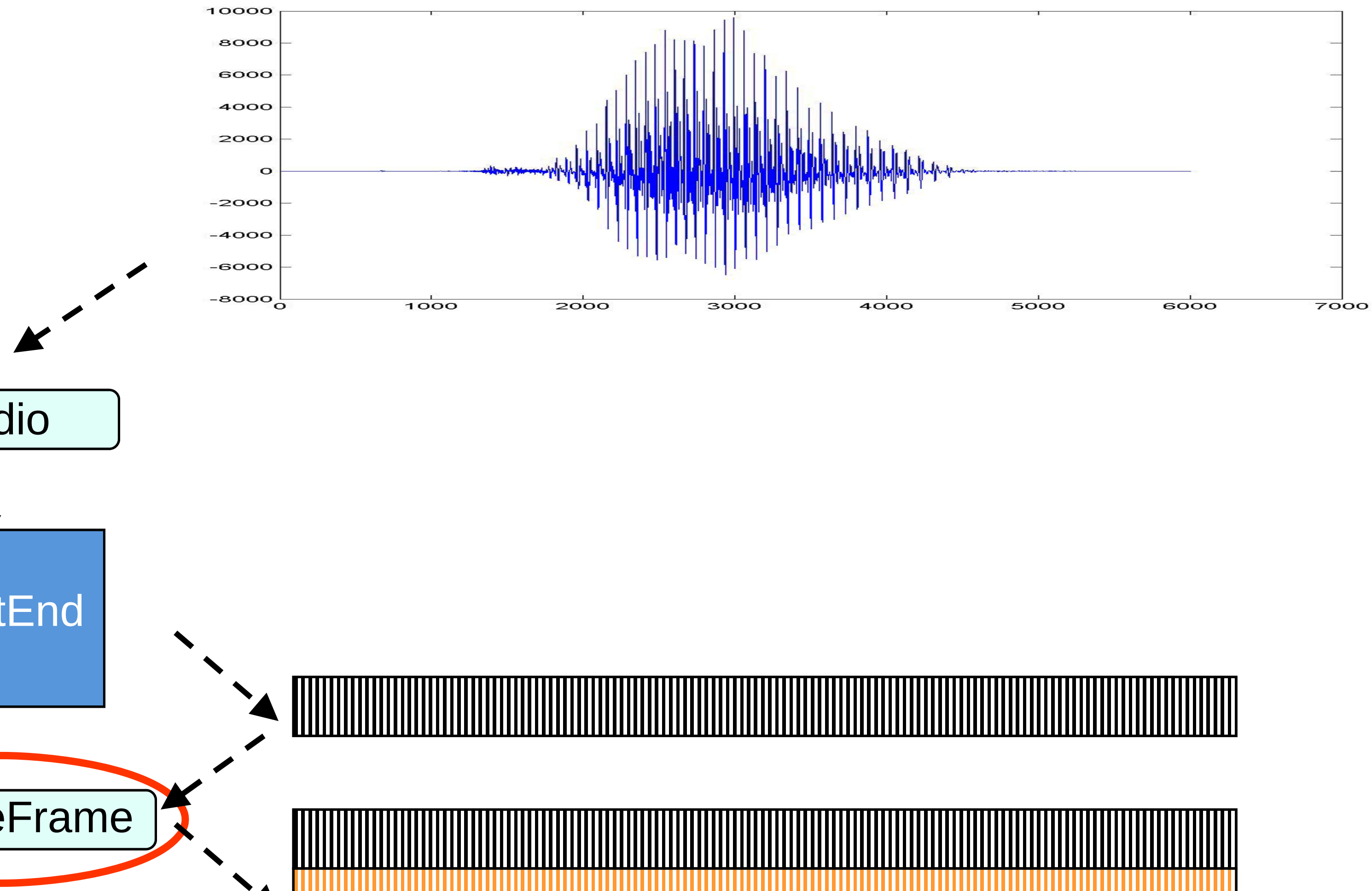
$$\Delta\Delta c(t)=\Delta c(t+\tau)-\Delta c(t-\tau)$$

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

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 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)

Project 1

- 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 and stop of audio recording