Modifying Audio Signals

Topics

- Denoising
- Rate/Pitch modification
  - Psola: Pitch-Synchronous Overlap and Add
  - Phase vocoder

De-noising

- Multifaceted problem
  - Removal of unwanted artifacts
  - Clicks, hiss, warps, interfering sounds, …
- For now
  - Constant noise removal
    - Wiener filters, spectral/power subtraction
  - Click detection and restoration
    - AR models for abnormality detection
    - AR models for making up missing data

The problem with audio recordings

- Recordings are inherently messy!!
- Recordings capture room resonances, air conditioners, street ambience, etc …
  - Resulting in low frequency rumbling sounds (the signature quality of a low-budget recording!)
- Media get damaged
  - Magnetic recording media get demagnetized
    - Results in high frequency hissing sounds (old tapes)
  - Mechanical recording media are littered with debris
    - Results in clicking and crackling sounds (ancient vinyl disks, optical film soundtracks)
  - Digital media feature sample drop-outs
    - Results in gaps in audio which when short are perceived as clicks, otherwise it is an audible gap (damaged CDs, poor internet streaming, bad bluetooth headsets)

Restoration of audio

- People don't like noisy recordings!!
  - There is a need for audio restoration work
- Early restoration work was an art form
  - Experienced engineers would design filters to best cover defects, cut and splice tapes to remove unwanted parts, etc.
  - Results were marginally acceptable
- Recent restoration work is a science
  - Extensive use of signal processing and machine learning
  - Results are quite impressive!

Audio Restoration I: Noise removal

- Noise is often inherent in a recording or slowly creeps in the recording media
- Hiss, rumbling, ambience, …
- Approach
  - Figure out noise characteristics
  - Spectral processing to make up for noise

Describing additive noise

- Assume additive noise

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

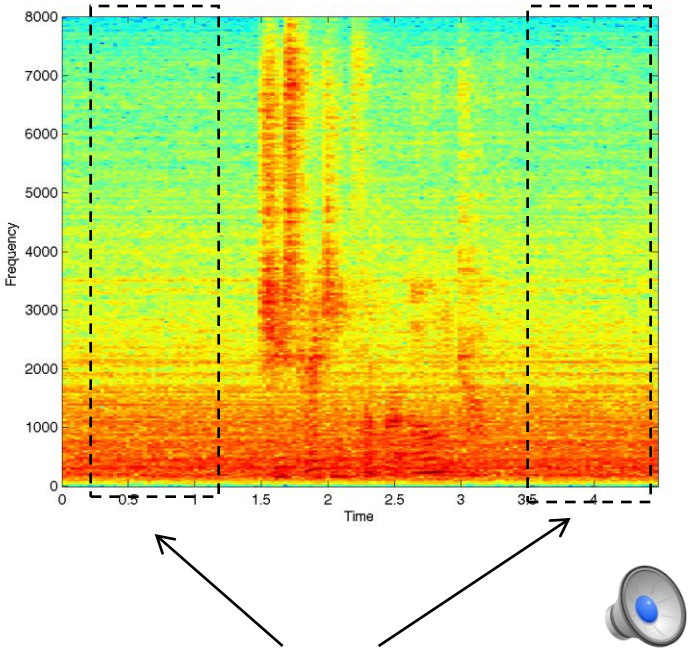
- In the frequency domain

$$X(t, f) = S(t, f) + N(t, f)$$

- Find the spots where we have only isolated noise
  - Average them and get noise spectrum

$$\mu(f) = \frac{1}{M} \sum_{\forall t, S(t,f) \approx 0} \|X(t, f)\|$$

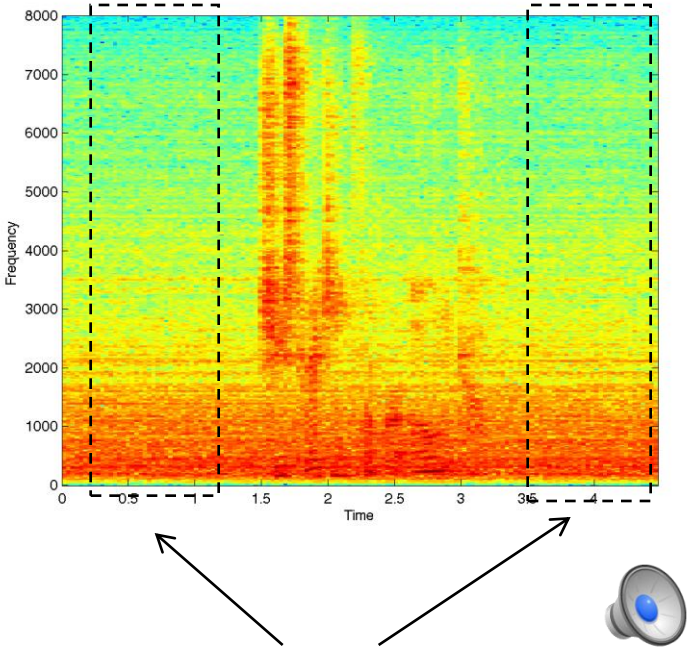
$$M = \text{number of noise frames}$$



Sections of isolated noise (or at least no useful signal)

Spectral subtraction methods

- We can now (perhaps) estimate the clean sound
  - We know the characteristics of the noise (as described from the spectrum $\mu(f)$)
- But, we will assume:
  - The noise source is constant
    - If the noise spectrum changes $\mu(f)$ is not a valid noise description anymore
  - The noise is additive



Sections of isolated noise (or at least no useful signal)

Spectral subtraction

- Magnitude subtraction
  - Subtract the noise magnitude spectrum from the recording's

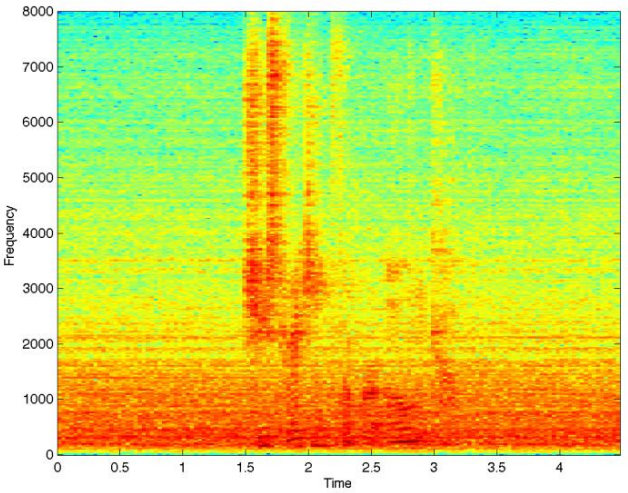
$$X(t,f) = S(t,f) + N(t,f) \Rightarrow$$

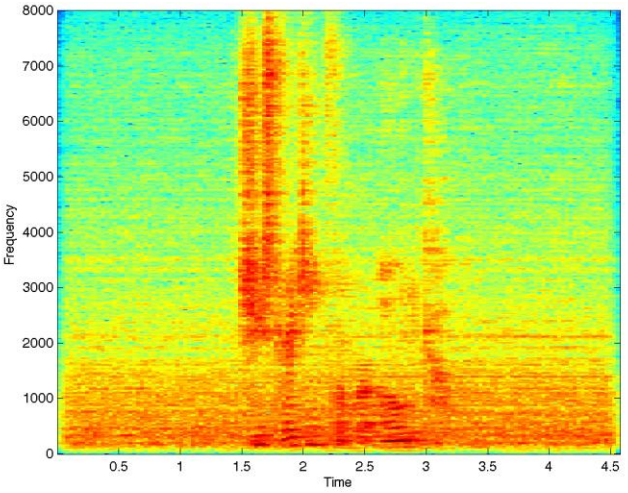
$$\left\|\hat{S}(t,f)\right\| = \left\|X(t,f)\right\| - \mu(f)$$

- We can then modulate the magnitude of the original input to reconstruct

$$\hat{S}(t,f) = \left[\left\|X(t,f)\right\| - \mu(f)\right] \angle X(t,f)$$

- Sounds pretty good …





Original input

After spectral subtraction

Estimating the noise spectrum

- Noise is usually not stationary
  - Although the rate of change with time may be slow
- A running estimate of noise is required
  - Update noise estimates at every frame of the audio
- The exact location of “noise-only” segments is never known
  - For speech signals we use an important characteristic of speech to discover speech segments (and, consequently noise-only segments) in the audio
  - The onset of speech is always indicated by a sudden increase in the energy level in the signal

A running estimate of noise

- The initial T frames in any recording are assumed to be free of the speech signal
  - Typically T = 10
- The noise estimate $N(\mathrm{T},\mathrm{f})$ is estimated as

$$N(\mathrm{T},\mathrm{f}) = (1/\mathrm{T})\ \Sigma_{\mathrm{t}}\ |X(\mathrm{t},\mathrm{f})|$$

- Subsequent estimates are obtained as follows
  - Assumption: The magnitude spectrum increases suddenly in value at the onset of speech

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

A running estimate of noise

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

- *p* is an exponent term that is typically set to either 2 or 1
  - *p* = 2 : power spectrum; p = 1 : magnitude spectrum

- λ is a noise update factor
  - Typically set in the range 0.1 – 0.5
  - Accounts for time-varying noise

- β is a thresholding term
  - A typical value of β is 5.0
  - If the signal energy jumps by a factor of β, speech onset has occurred

- Other more complex rules may be applied to detect speech offset

Cancelling the Noise

- Simple Magnitude Subtraction
  - $|S(t,f)| = |X(t,f)| - |N(t,f)|$
- Power subtraction
  - $|S(t,f)|^2 = |X(t,f)|^2 - |N(t,f)|^2$
- Filtering methods: $S(t,f) = H(t,f)X(t,f)$
  - Weiner Filtering: build an optimal filter to remove the estimated noise
  - Maximum-likelihood estimation..

The Filter Functions

- We have a source plus noise spectrum

$$X(t, f) = S(t, f) + N(t, f)$$

- The desired output is some function of the input and the noise spectrum

$$\hat{S}(t, f) = g\big(X(t, f), N(t, f)\big)$$

- Let's make it a "gain function"

$$H(t, f) = f\big(X(t, f), N(t, f)\big)$$

$$\hat{S}(t, f) = H(t, f)X(t, f)$$

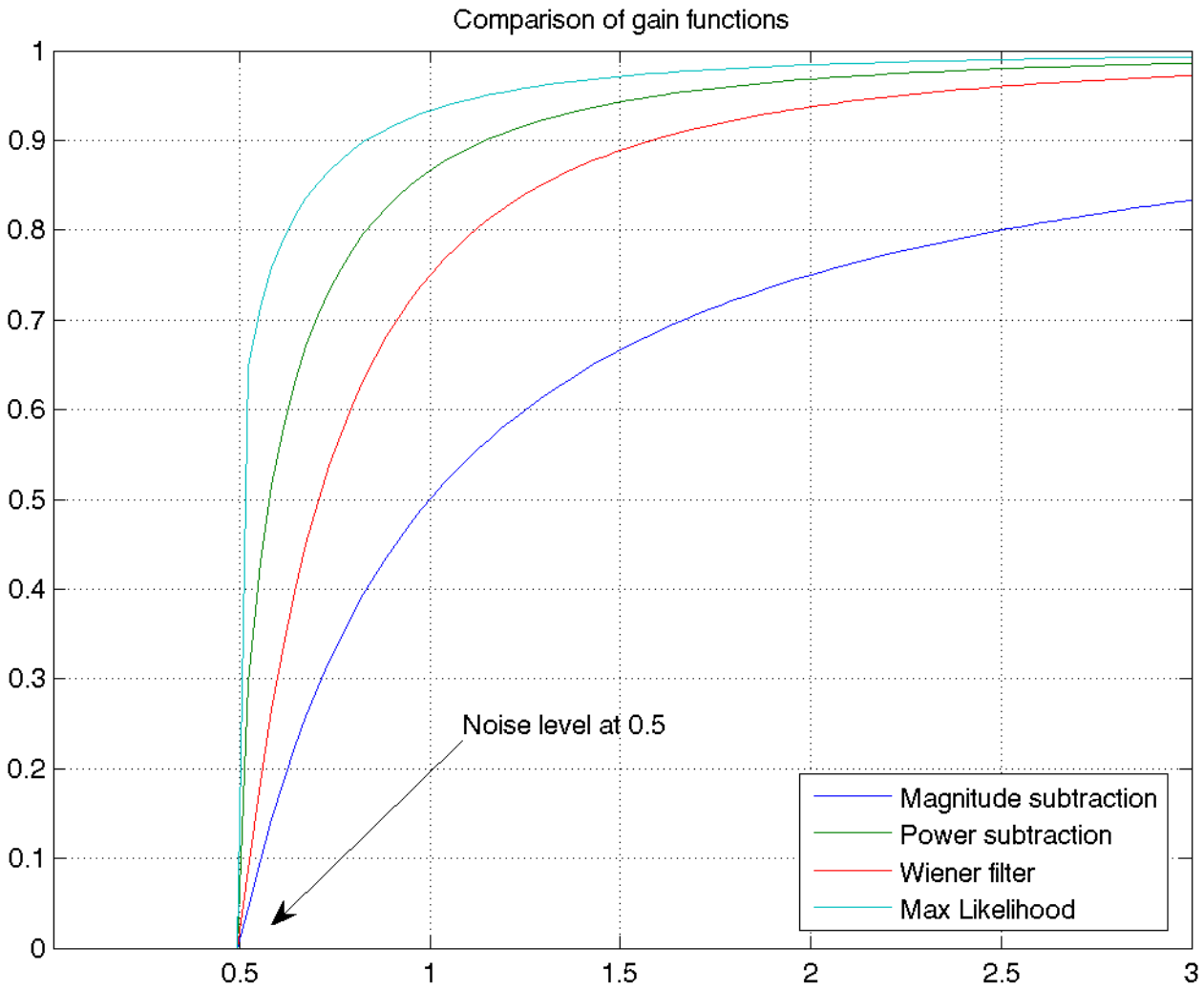
- For spectral subtraction the gain function is:

$$H(t, f) = 1 - \frac{\|N(t, f)\|}{\|X(t, f)\|}$$

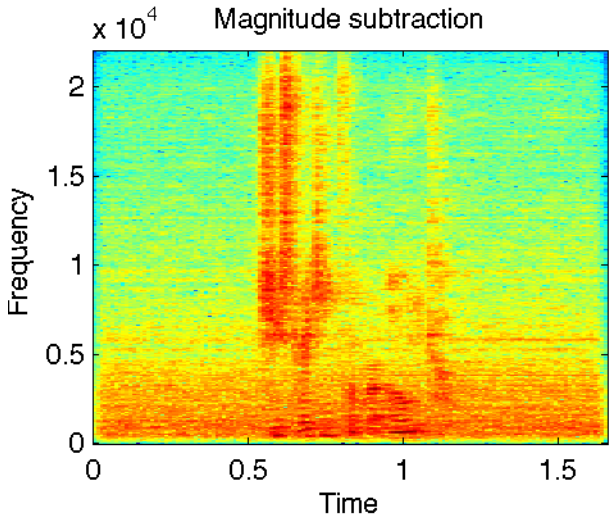
Filters for denoising

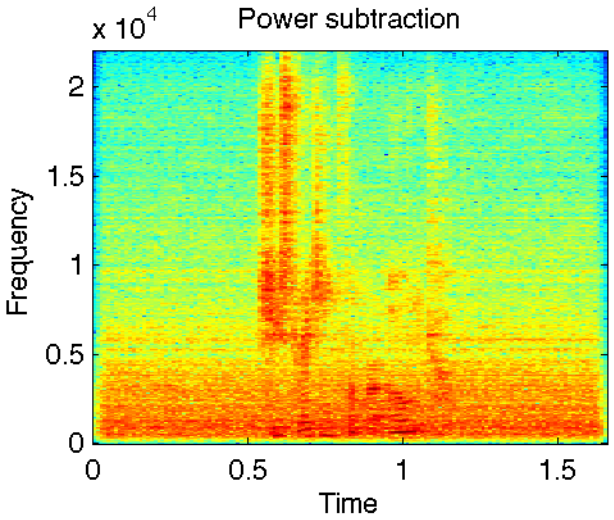
- Magnitude subtraction: $H(f) = 1 - \frac{N(f)}{\|X(f)\|}$
- Power subtraction: $H(f) = \sqrt{1 - \frac{N^2(f)}{\|X(f)\|^2}}$
- Wiener filter: $H(f) = 1 - \frac{N^2(f)}{\|X(f)\|^2}$
- Maximum likelihood: $H(f) = \frac{1}{2}\left[1 + \sqrt{1 - \frac{N^2(f)}{\|X(f)\|^2}}\right]$

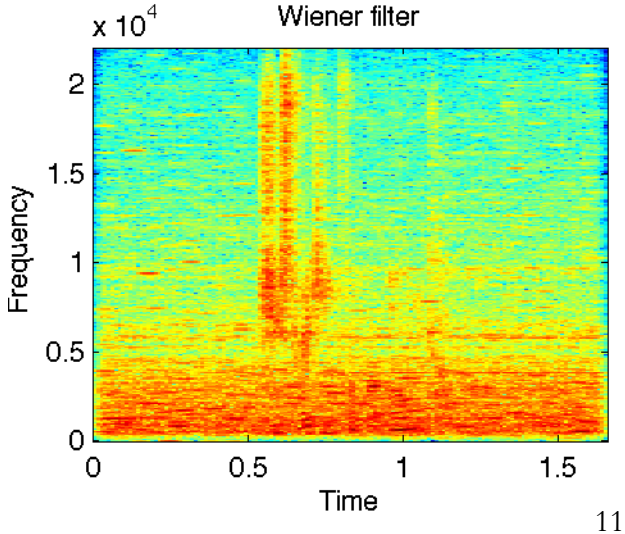
Filter function comparison

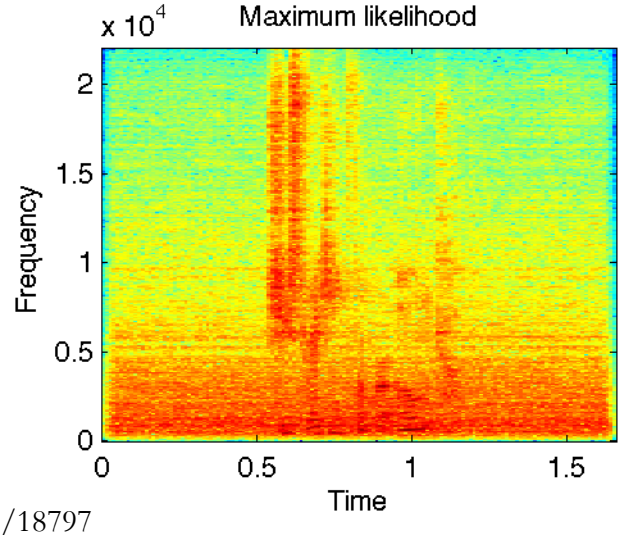


Examples of various filter functions









Original

Magnitude subtraction

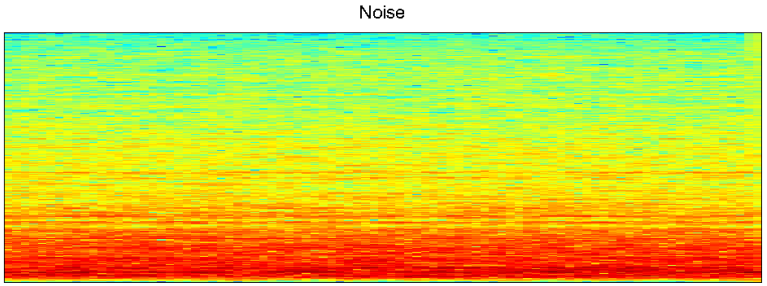
Power subtraction

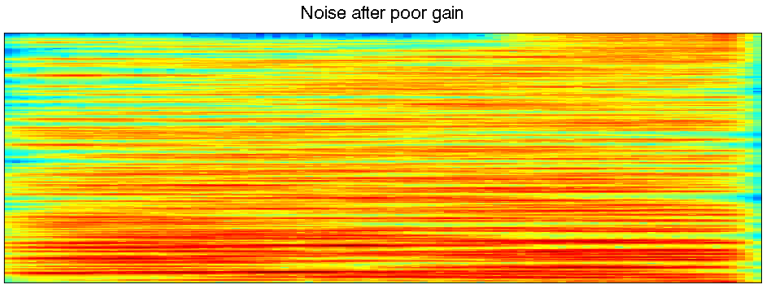
Wiener filter

Maximum likelihood

"Musical noise"

- What was that weirdness with the Wiener filter???
  - An artifact called *musical noise*
  - The other approaches had it too
- Takes place when the signal to noise ratio is small
  - Ends up on the steep part of the gain curve
    - Small fluctuations are then magnified
  - Results in complex or negative gain
    - An awkward situation!
- The result is sinusoids popping in and out
  - Hence the tonal overload





Noise reduced noise!
(lots of musical noise)

Reducing musical noise

- Thresholding

$$H'(f) = \begin{cases} H(f) & \text{if } \|X(f)\| > N(f) \\ 0 & \text{otherwise} \end{cases}$$

  - The gain curve is steeper on the negative side
  - This removes effects in that area

- Scale the noise spectrum

  $N(f) = \alpha\ N(f),\ \alpha > 1$

  - (Linearly) increases gain in the new location

- Smoothing

  e.g. $H(t,f) = .5H(t,f) + .5H(t-1,f)$

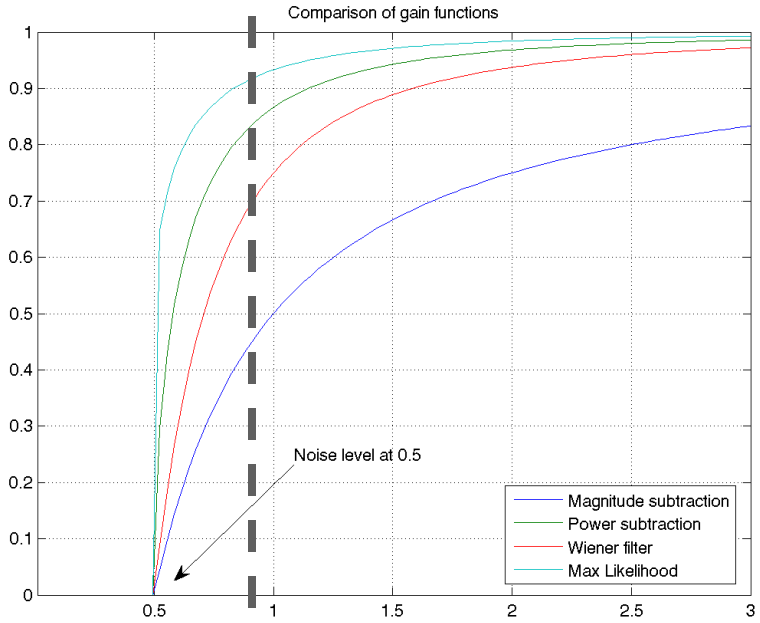
  - Or some other time averaging
  - Reduces sudden tone on/offs
  - But adds a slight echo

Wiener filter

With thresholding

With thresholding & smoothing

Reducing musical noise



Wiener filter

With thresholding and oversub

With thresholding, oversub, and smoothing

- Thresholding : Moves the operating point to a less sloped region of the curve
- Oversubtraction: Increases the slope in these regions for better differential gain
- Smoothing: $H(t,f) = 0.5H(t,f) + 0.5H(t-1,f)$
  - Adds an echo

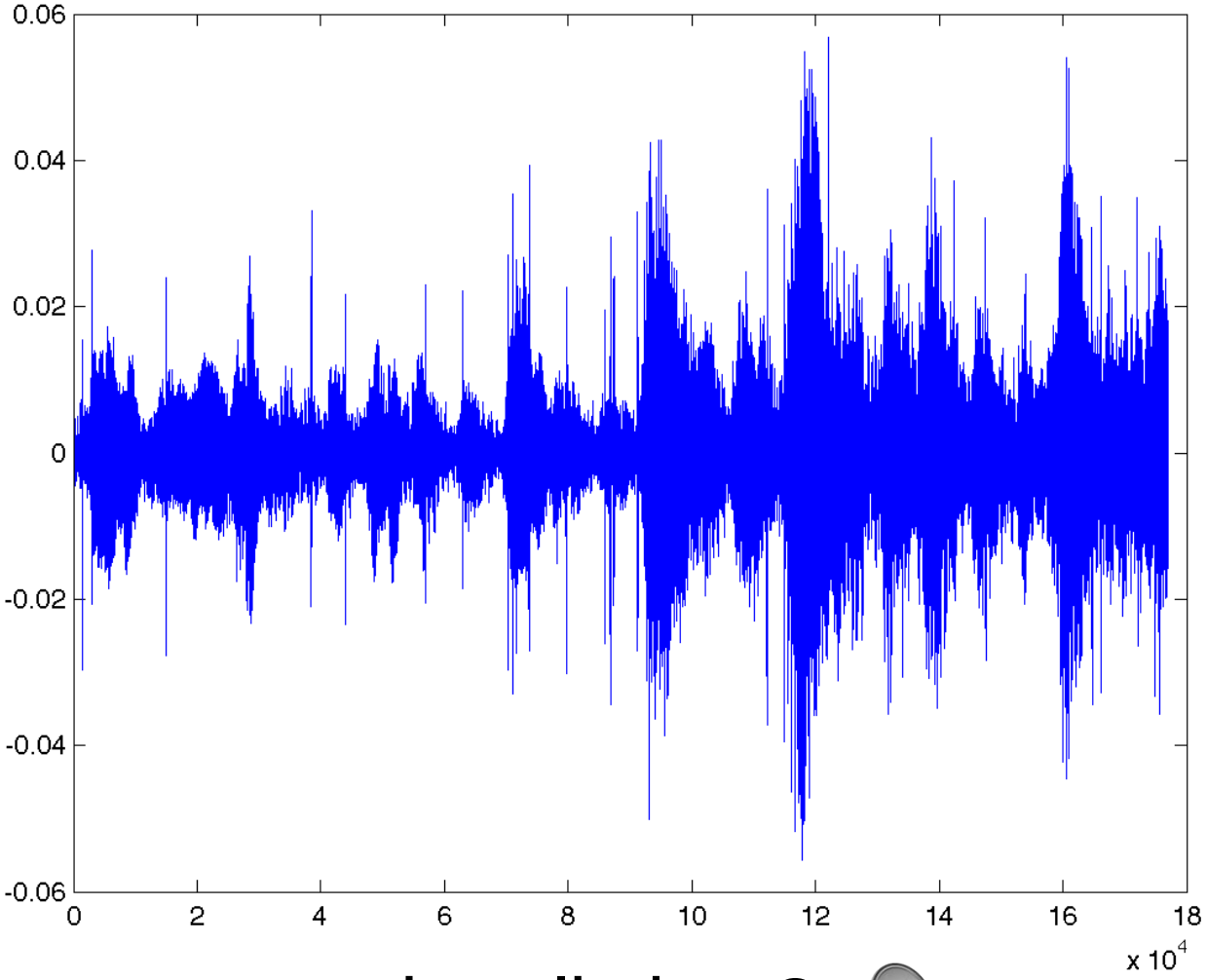
Non-stationary noises

- Same approach as stationary/slowly-varying noise, with tuning
- Source separation approaches (latent-variable decomposition, NMF)
- Switching Wiener filter
  - Have multiple Wiener filters (one per noise type)
    - Pre-trained for each noise type
    - On-line selection of Wiener filter / interpolation of Wiener filters

Audio restoration II: Click/glitch/gap removal

- Two step process
  - Detection of abnormality
  - Replacement of corrupted data
- Detection:
  - Autoregressive modeling for abnormality detection
- Data replacement:
  - Interpolation of missing data using autoregressive interpolation

Starting signal



- Can you spot the glitches?

Autoregressive (AR) models

- Predicting the next sample of a series using a weighted sum of the past samples

$$x(t) = \sum_{i=1}^{N} a(i)x(t-i) + e(t)$$

- The weights $\boldsymbol{a}$ can be estimated upon presentation of a training input $\boldsymbol{x}$
  - Least squares solution of above equation

Matrix formulation

- Scalar version

$$x(t) = \sum_{i=1}^{N} a(i)x(t-i) + e(t)$$

- Matrix version

$$\mathbf{x} = \begin{bmatrix} a_{N-1} & \cdots & a_0 & 0 & 0 \\ 0 & \ddots & \cdots & a_0 & 0 \\ 0 & 0 & \ddots & \cdots & a_0 \\ 0 & 0 & 0 & \ddots & \cdots \\ 0 & 0 & 0 & 0 & a_{N-1} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_M \end{bmatrix}$$

Measuring prediction error

- As Convolution

  $e = x - a * x$

- As matrix operation $e = Ax$

$$\mathbf{e} = \begin{bmatrix} -a_N & \cdots & -a_1 & 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & -a_N & \cdots & -a_1 & 1 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \vdots & \vdots & \\ \cdots & 0 & 0 & -a_N & \cdots & -a_1 & 1 & 0 & 0 \\ 0 & \cdots & 0 & 0 & -a_N & \cdots & -a_1 & 1 & 0 \\ 0 & 0 & \cdots & 0 & 0 & -a_N & \cdots & -a_1 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ \vdots \\ x_M \end{bmatrix}$$

- Overall error variance: $e^T e$

Measuring prediction error

- The predictor:

  $e = x - a * x; \quad e = Ax$

- Solution for a must minimize error variance: $E = e^T e$
  - While maintaining the Toeplitz structure of a!

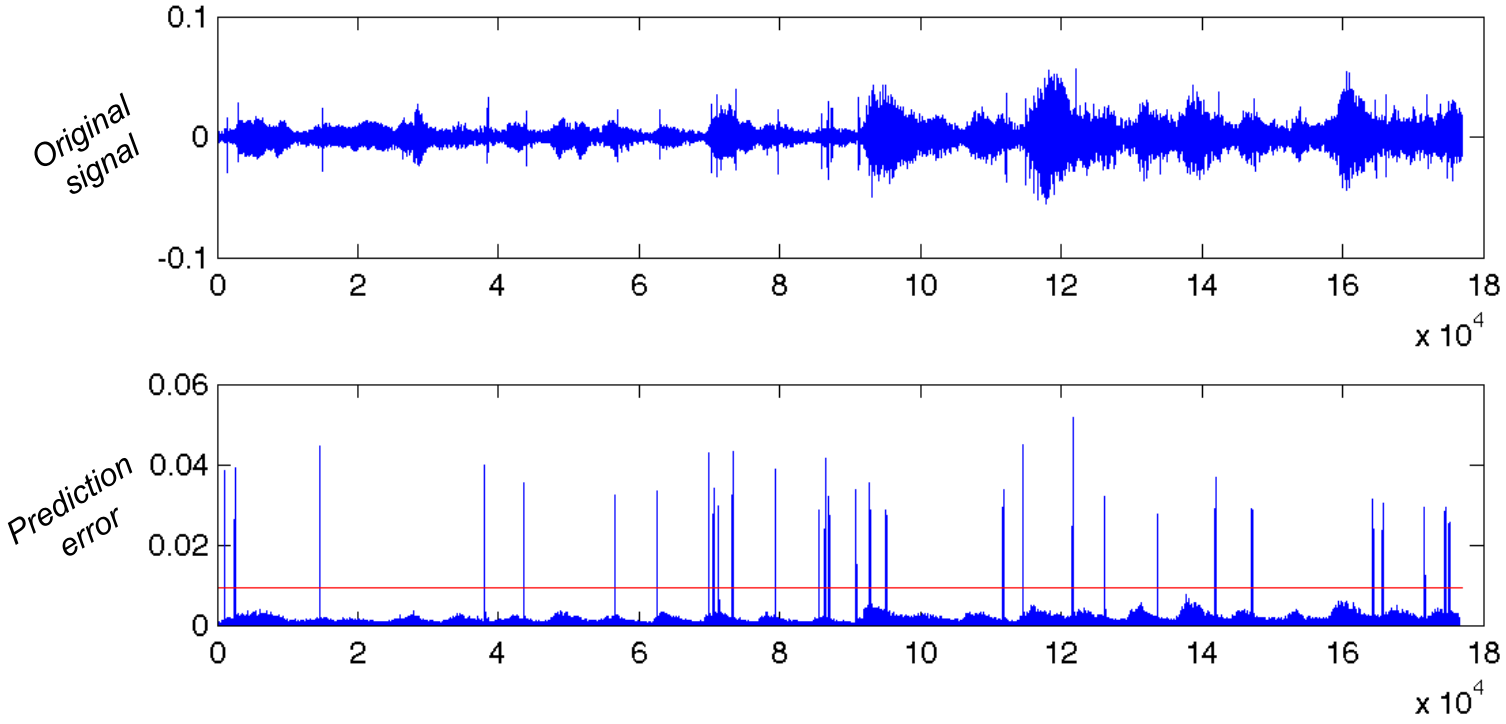
- A variety of solution techniques are available
  - Differentiate E w.r.t "a" and solve for "a" with Toeplitz constraint
  - Other algorithms
    - The most popular one is the "Levinson Durbin" algorithm

Discovering abnormalities

- The AR models smooth and predictable things, e.g. music, speech, etc
- Clicks, gaps, glitches, noise are not very predictable (at least in the sense of a meaningful signal)
- Methodology
  - Learn an AR model on your signal type
  - Measure prediction error on the noisy data
  - Abnormalities appear as spikes in error

Glitch detection example

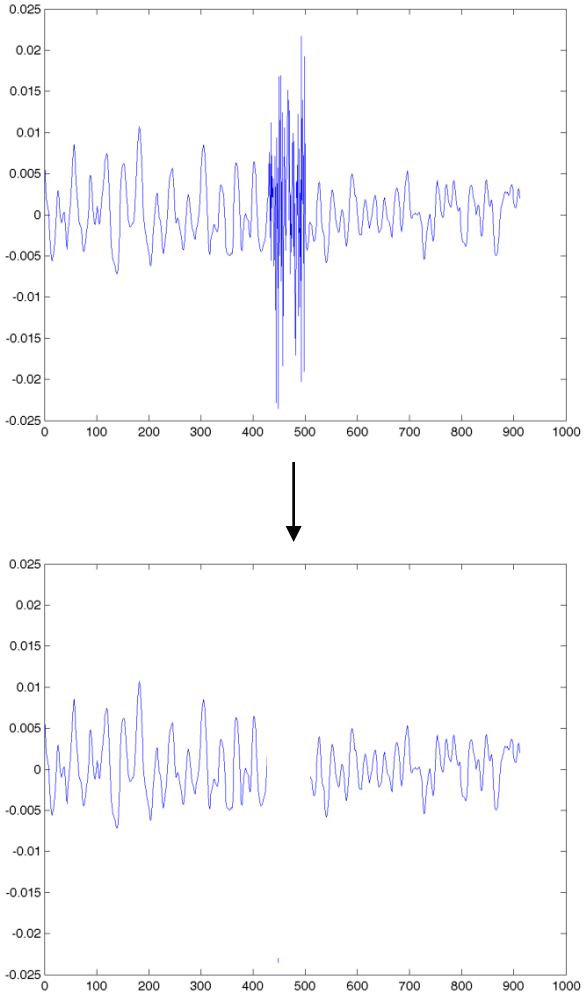
- Glitches are clearly detected as spikes in the prediction error



- Why? Glitches are unpredictable!

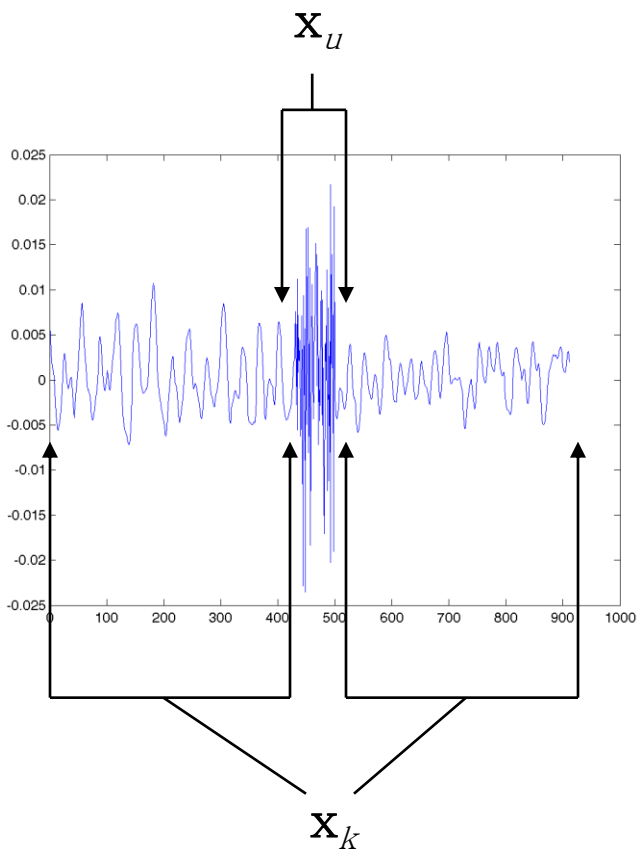
Now what?

- Detecting the glitches is only one step!
- How to we remove them?
- Information is lost!
  - We need to make up data!
- This is an interpolation problem
  - Filling in missing data
  - Hints provided from neighboring samples



Interpolation formulation

- Detection of spikes defines areas of missing samples
  - $\pm N$ samples from glitch point
- Group samples to known and unknown sets according to spike detection positions
  - $\mathbf{x}_k = \mathbf{K}\cdot\mathbf{x}$, $\mathbf{x}_u = \mathbf{U}\cdot\mathbf{x}$
  - $\mathbf{x} = (\mathbf{U}\cdot\mathbf{x} + \mathbf{K}\cdot\mathbf{x})$
  - Transforms $\mathbf{U}$ and $\mathbf{K}$ maintain only specific data (= unit matrices with appropriate missing rows)



Picking sets of samples

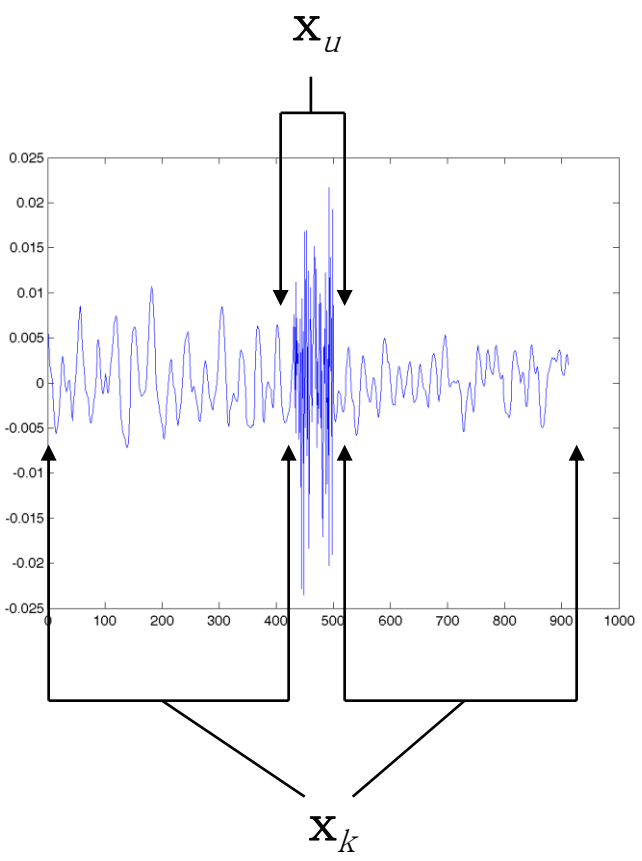
$$\mathbf{x} = \mathbf{U}\mathbf{x} + \mathbf{K}\mathbf{x} =$$

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} =$$

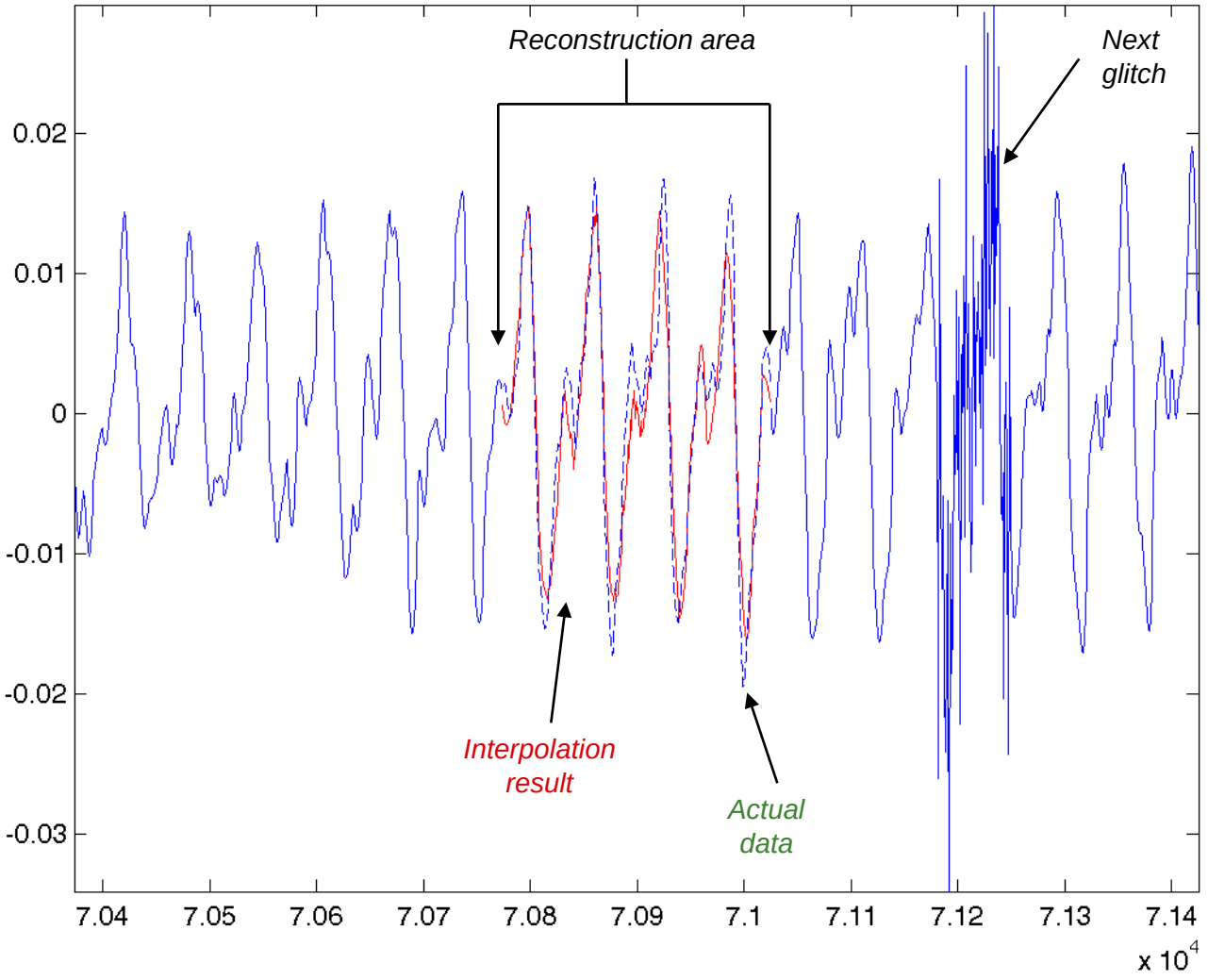
$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0 \\ x_2 \\ x_3 \\ 0 \end{bmatrix} + \begin{bmatrix} x_1 \\ 0 \\ 0 \\ x_4 \end{bmatrix}$$

Making up the data

- AR model error is
  - $e = A \cdot x = A \cdot (U \cdot x_u + K \cdot x_k)$
- We can solve for x_u
  - Ideally e is 0
- Hence zero error estimate for missing data is:
  - $A \cdot U \cdot x_u = -A \cdot K \cdot x_k$
  - $x_u = -(A \cdot U)^+ \cdot A \cdot K \cdot x_k$
    - $(A \cdot U)^+$ is pseudo-inverse



Reconstruction zoom in



Distorted signal

Recovered signal

Restoration recap

- Constant noise removal
  - Spectral subtraction/Wiener filters
  - Musical noise and tricks to avoid it
- Click/glitch/gap detection
  - Music/speech is very predictable
  - AR models to detect abnormalities
- Missing sample interpolation
  - AR model for creating missing data

Topics

- Voice rate modification
  - Psola: Pitch-Synchronous Overlap and Add
  - Phase vocoder
- Pitch Modification

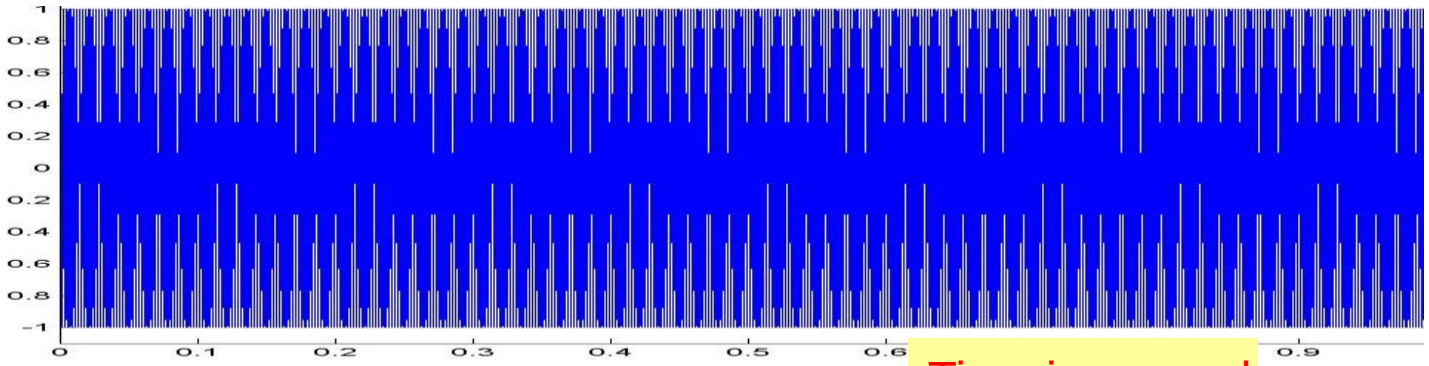
Changing the rate of audio

- Rate Modification:
  - Speed up: A given segment of audio must play back in half the time without sounding odd
  - Slow down: A given segment of audio must play back in twice the time without sounding odd
  - How?

- Two ways:
  - Time domain – somehow slice and dice the signal to get what you want
  - Do it all cleverly with filter banks or equivalent processing

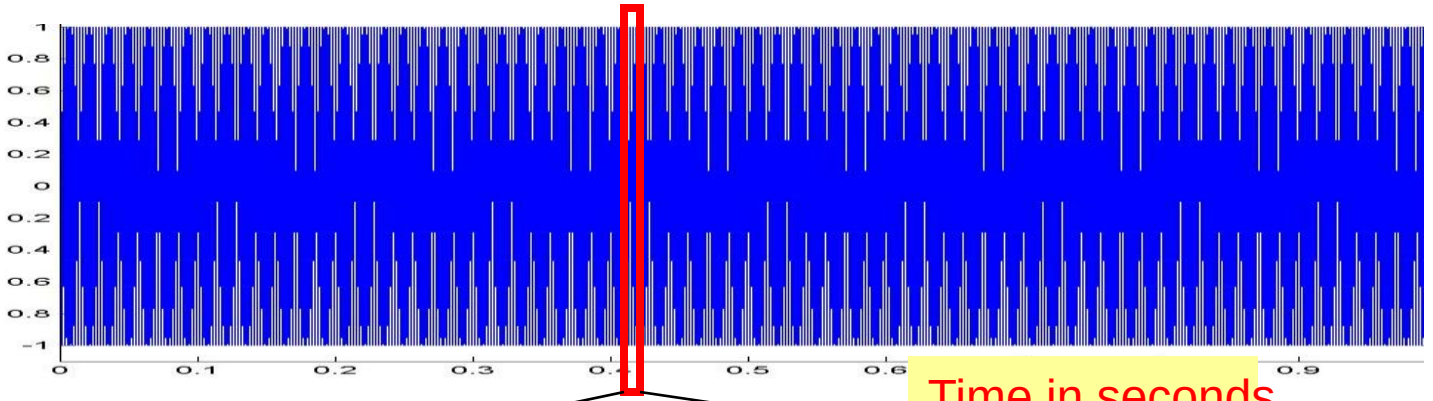
Speeding up a sinusoid

- A 500Hz sinusoid that is 1 second long
  - Sampled at 16000 samples per second
  - 1/16000 seconds between adjacent samples

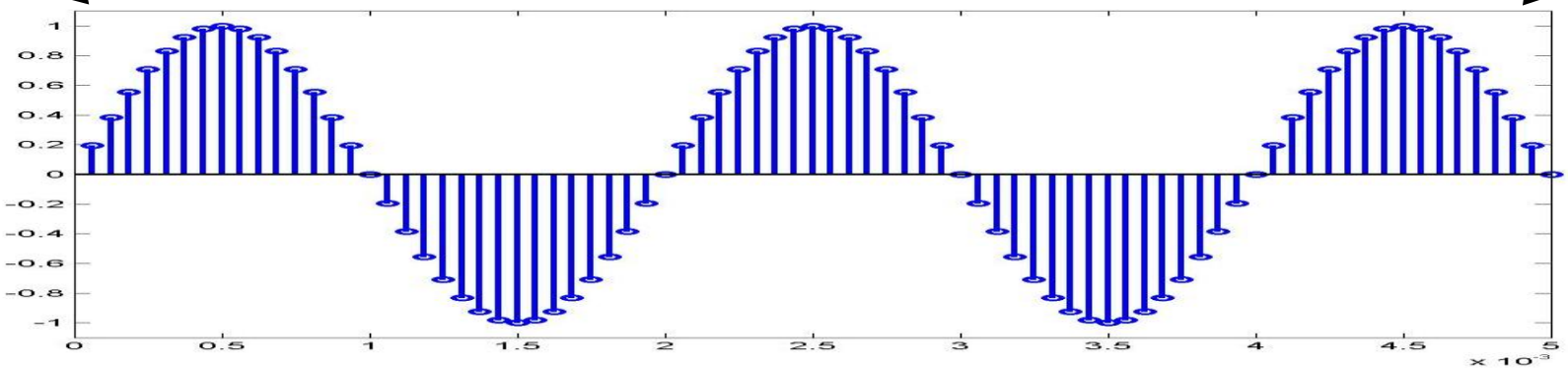


Speeding up a sinusoid

- A 500Hz sinusoid that is 1 second long
  - Sampled at 16000 samples per second
  - 1/16000 seconds between adjacent samples

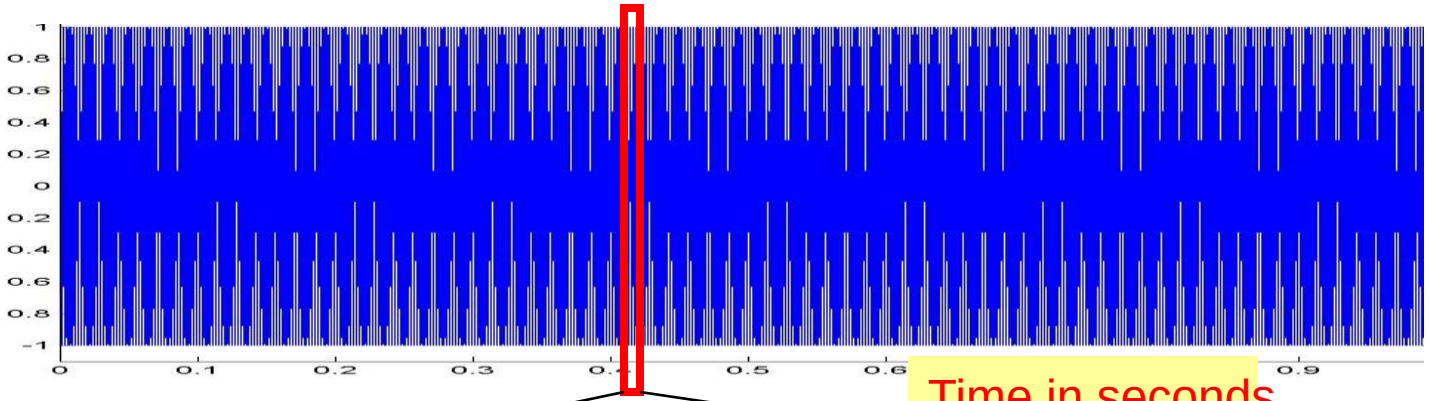


- ZOOMING IN

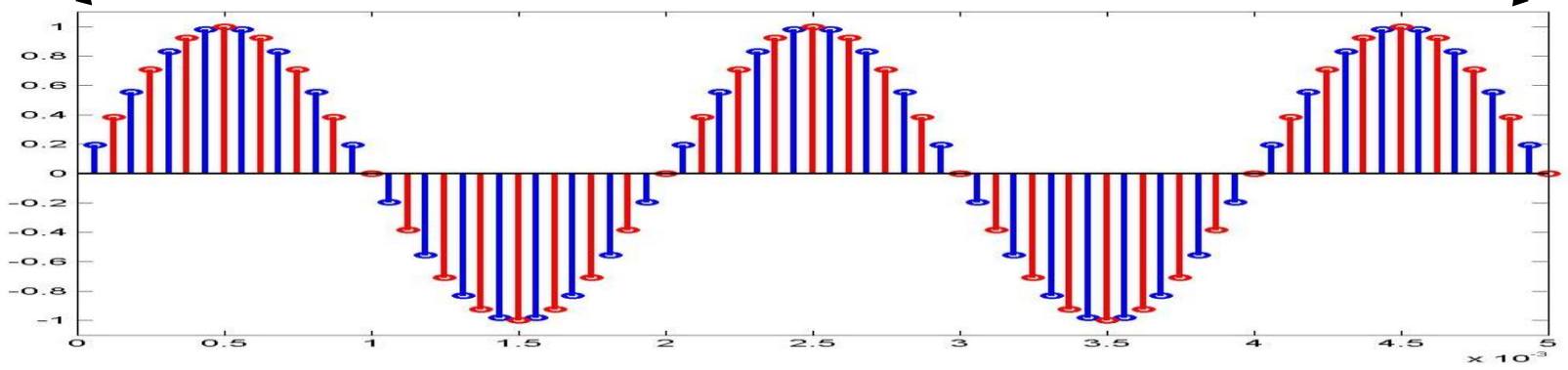


Speeding up a sinusoid

- Lets drop every second sample
  - Now left with 8000 samples, with 1/16000 seconds between adjacent samples

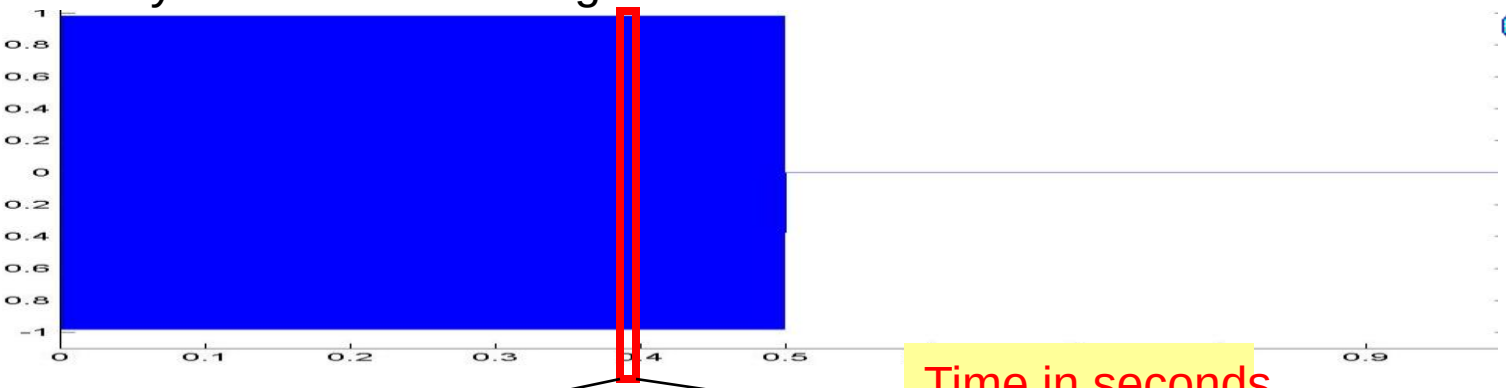


- ZOOMING IN

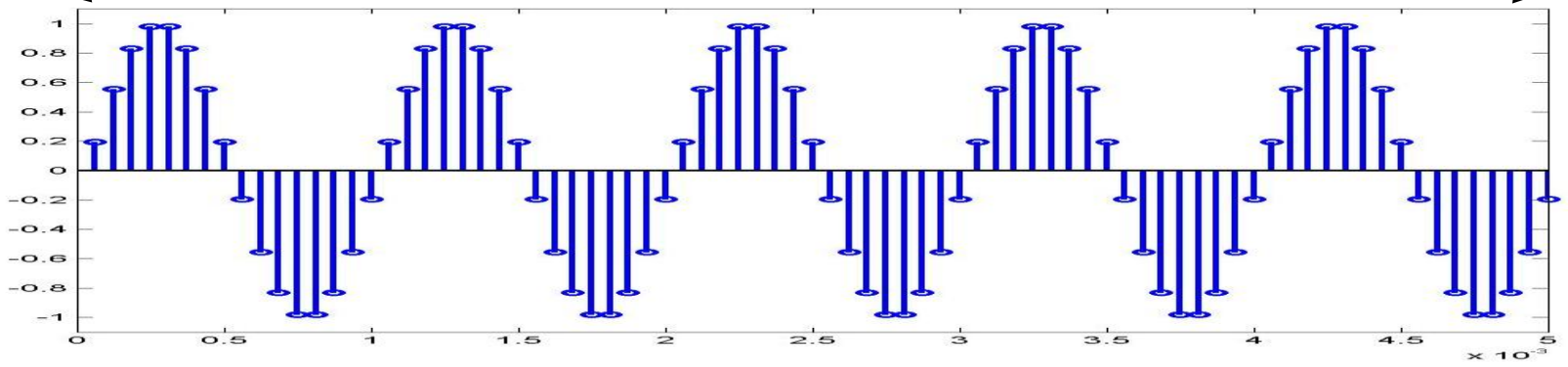


Speeding up a sinusoid

- Lets drop every second sample
  - Now left with 8000 samples, with 1/16000 samples between adjacent samples
  - Only half a second of signal



- ZOOMING IN



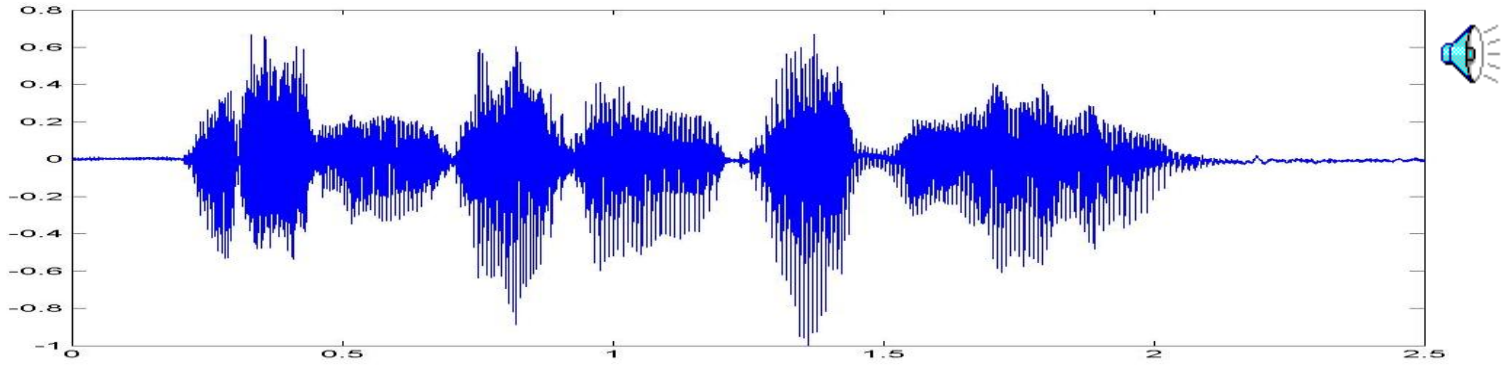
- Twice as many cycles as before in the same amount of time: i.e double the frequency, but only half as long in time

Downsampling is a bad thing

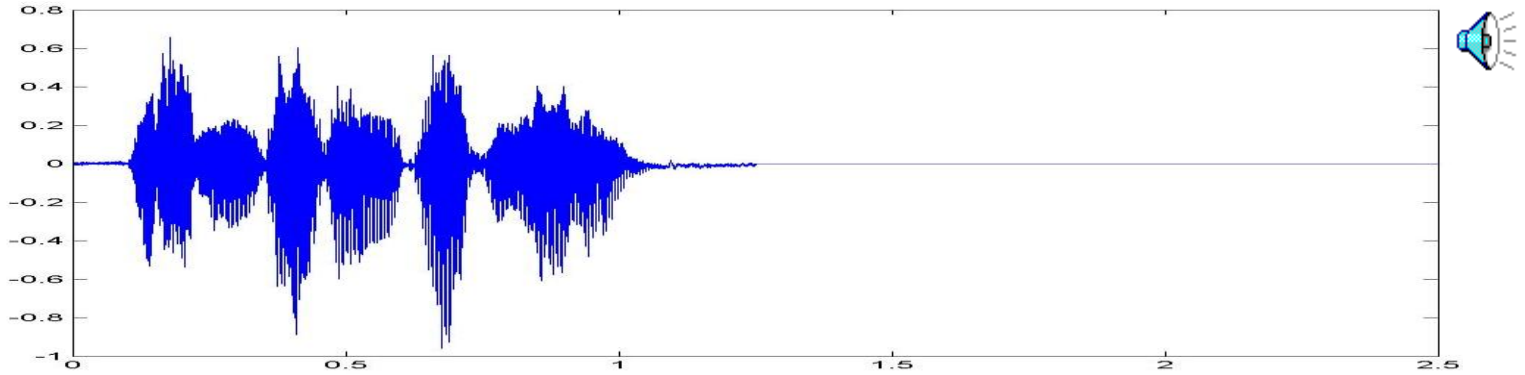
- We now have half the number of samples in the signal
- *We also have half the number of samples per cycle of the signal*
  - But the spacing (in time) between samples has not changed
- The total length (in time) of one cycle of the sinusoid has halved
  - *The frequency of the sinusoid had doubled*
- This is a natural outcome of downsampling

Downsampling a speech signal

- Tom Sullivan speaks his name (yet again)

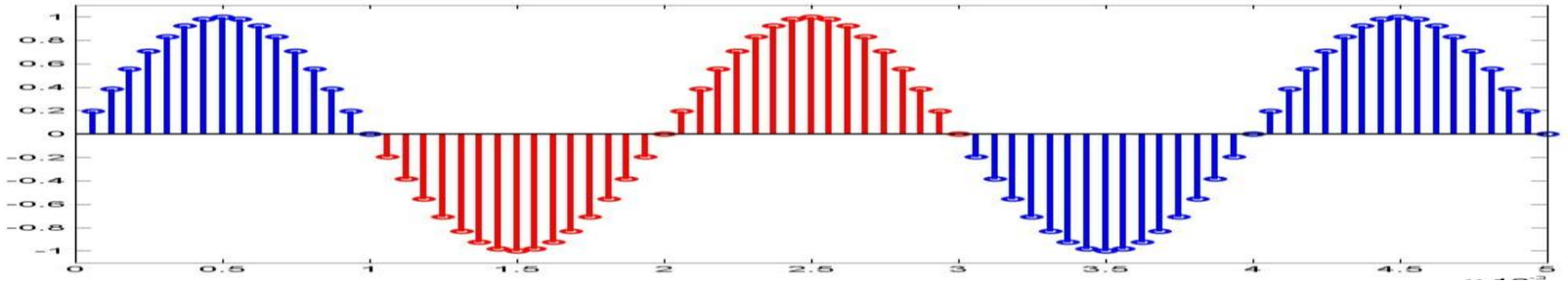


- Downsampling by a factor of 2



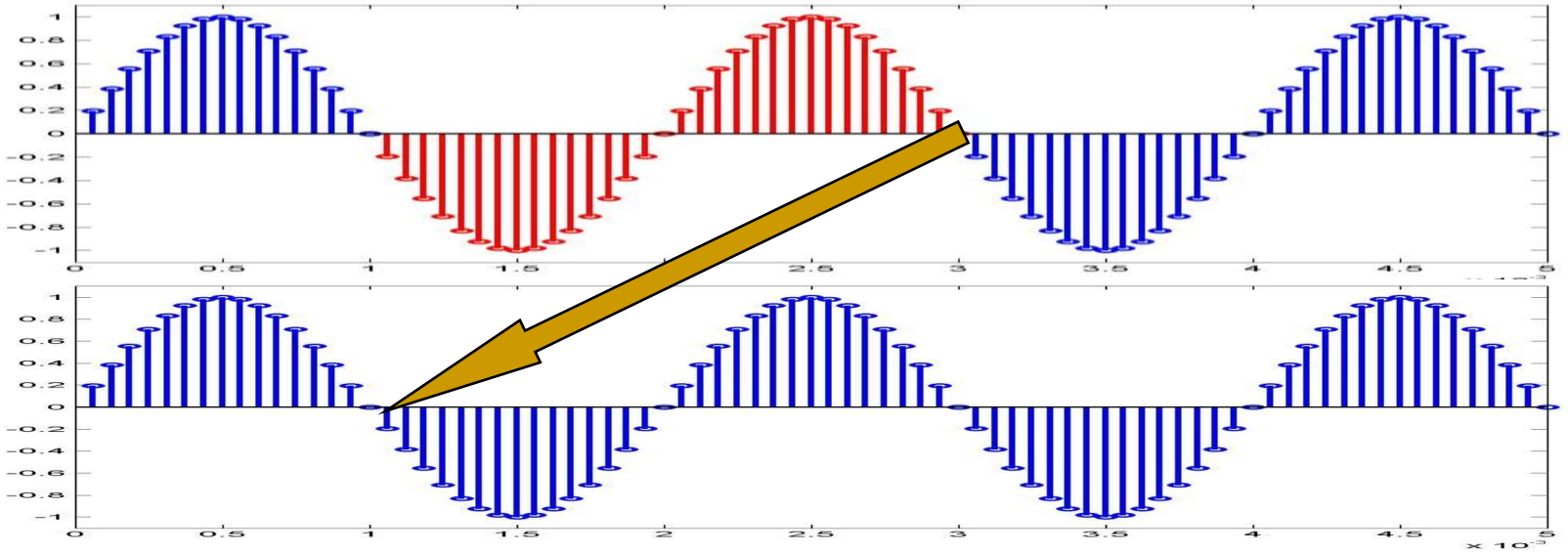
Doing it differently

- Instead of dropping alternate samples, lets drop alternate *cycles* of the sinusoid



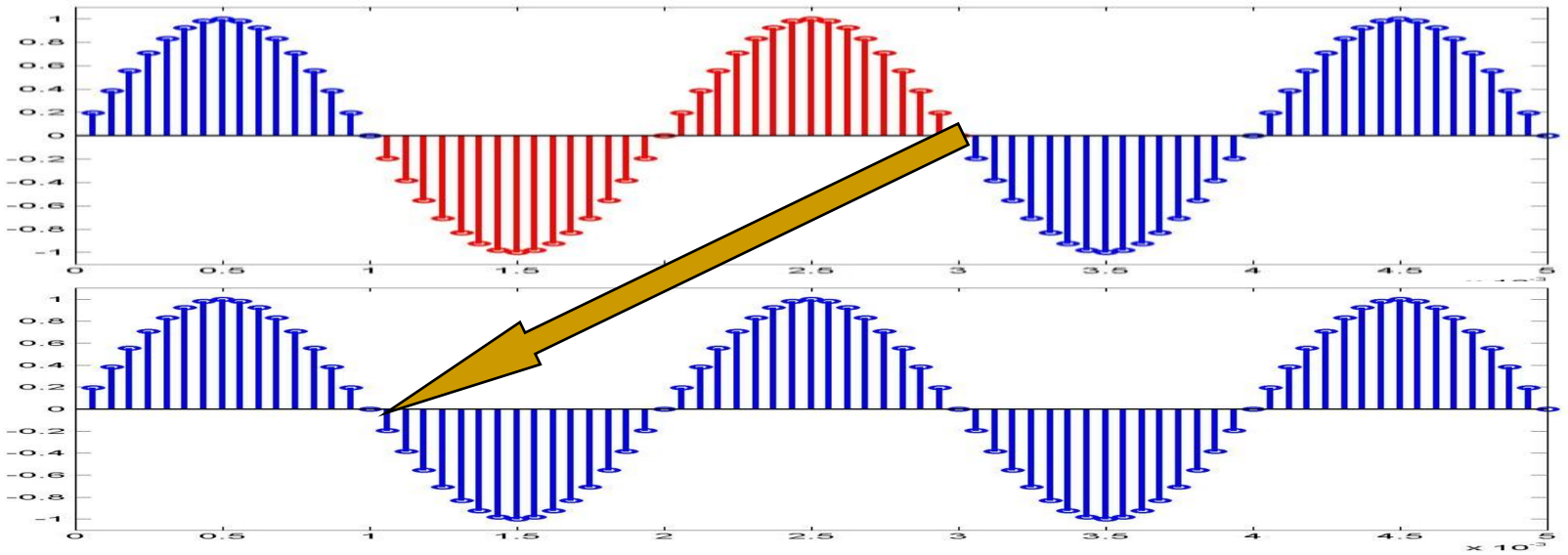
Doing it differently

- Instead of dropping alternate samples, lets drop alternate *cycles* of the sinusoid

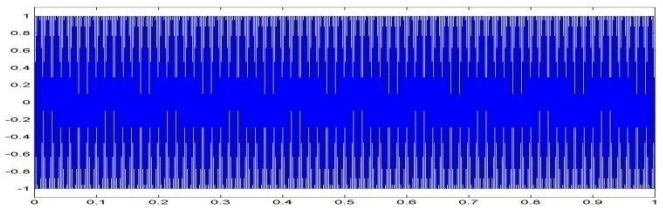


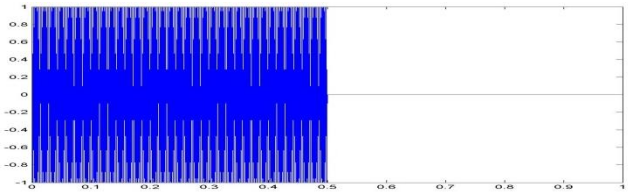
Doing it differently

- Instead of dropping alternate samples, lets drop alternate *cycles* of the sinusoid



- This gives us the correct result





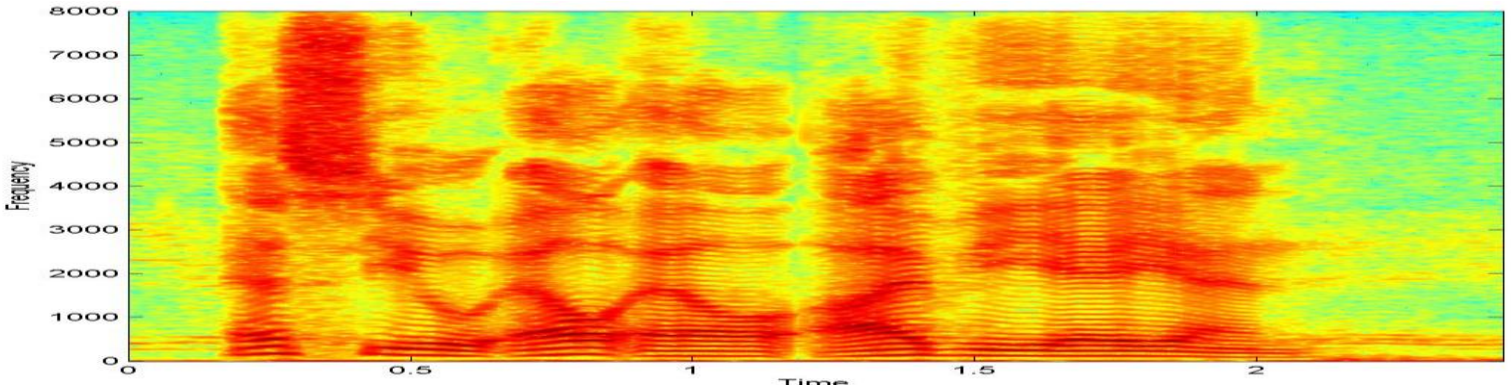
- Very important to match the phase properly though, otherwise discontinuities will happen that sound awful

How to do this for an audio signal

- PSOLA: The pitch-synchronous overlap addition method
- Identify repeating periods of the audio signal
  - Most speech signals occur in repeating patterns
    - Vowels, voiced sounds
    - They repeat at the “pitch” frequency
- Slice out periods of the signal to get the desired number of periods
  - To double the speed, we want half the periods
- Smooth transitions to eliminate discontinuities

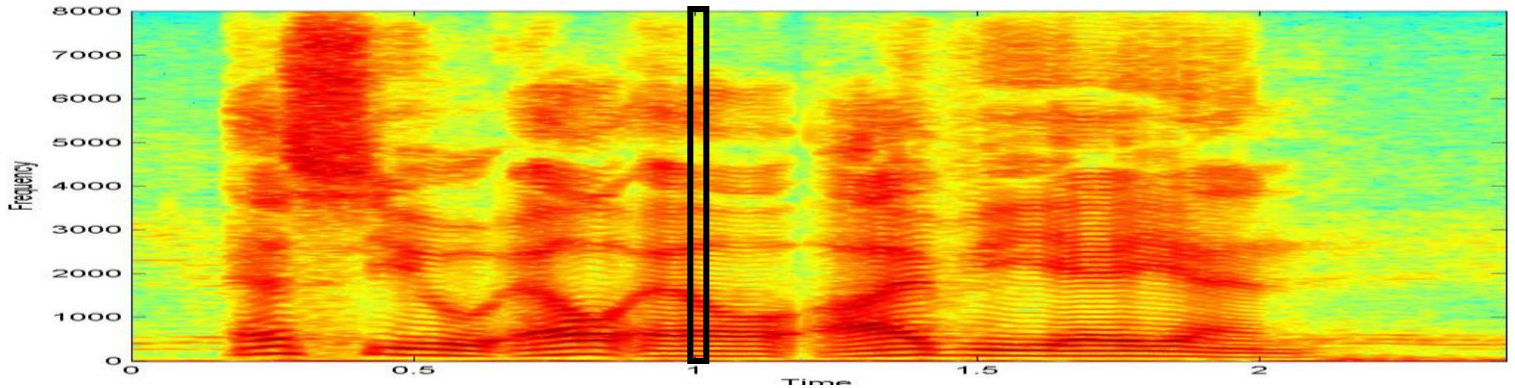
PSOLA

- Tom's Spectrogram

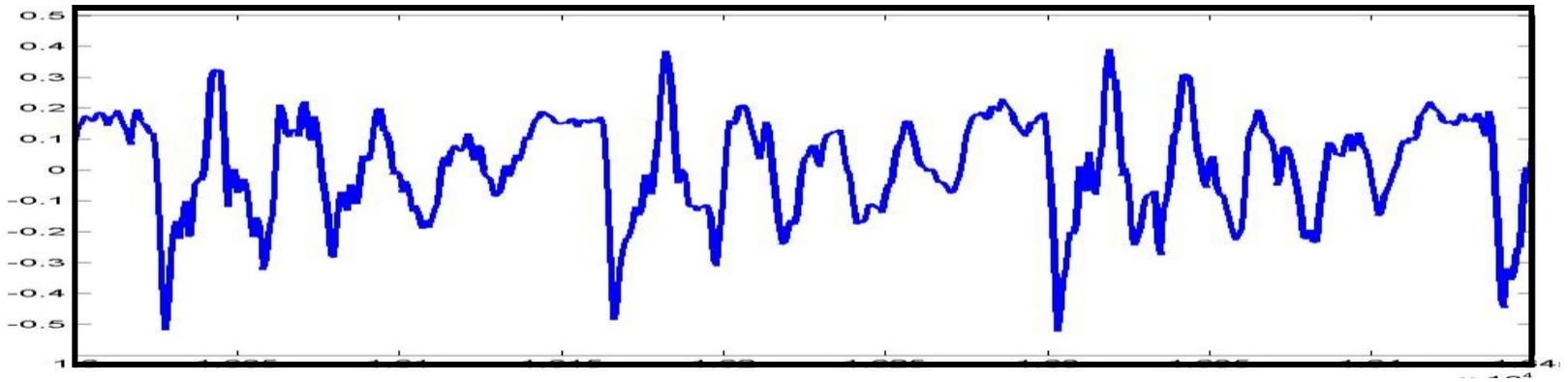


PSOLA

- Tom's Spectrogram

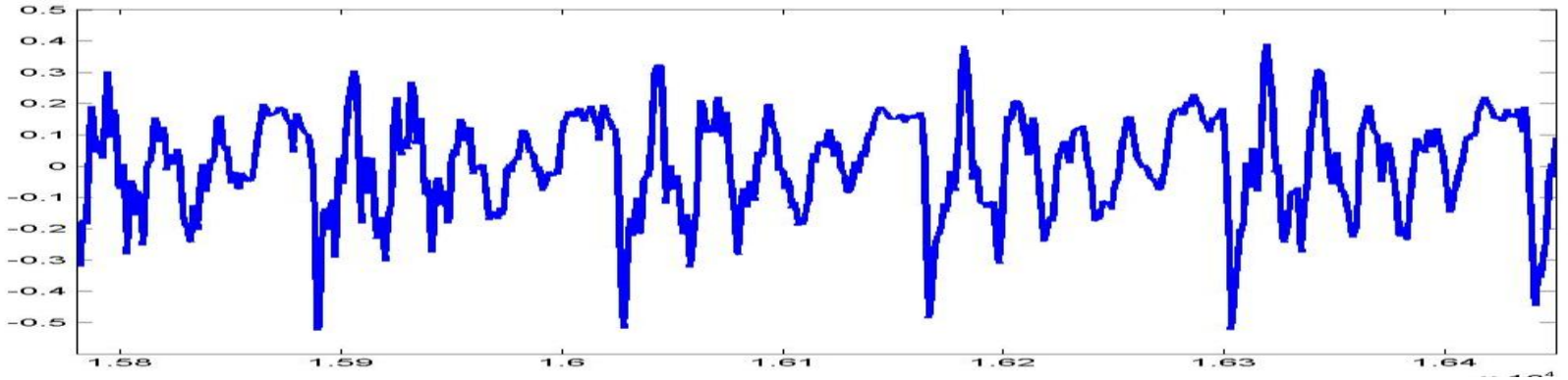


- A segment of voiced signal (observe periodicity)

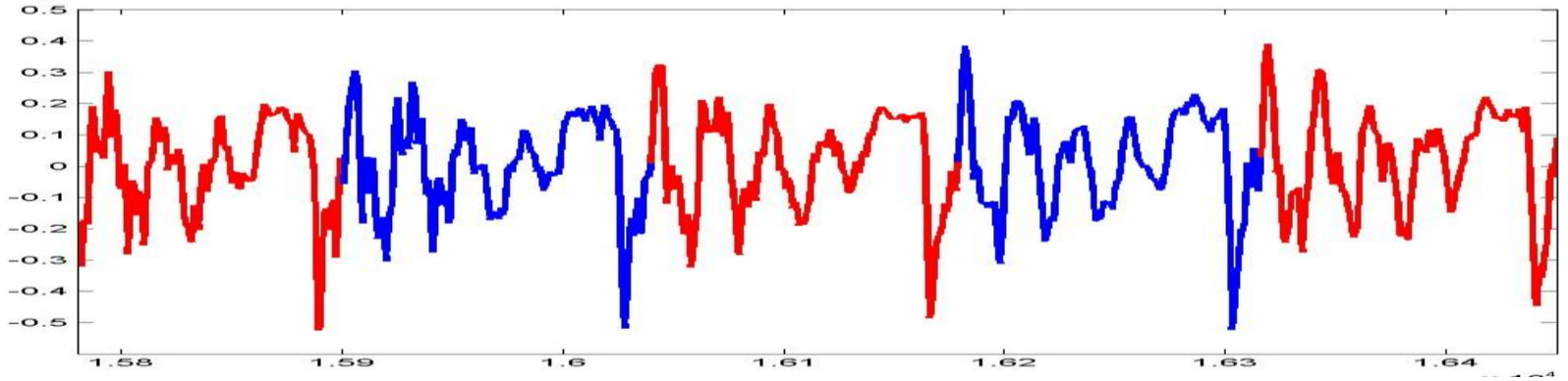


PSOLA for shortening a signal

- The original signal



- For halving the length Identify alternate pitch periods
  - Generally, for 1/X the length, identify every X-th period

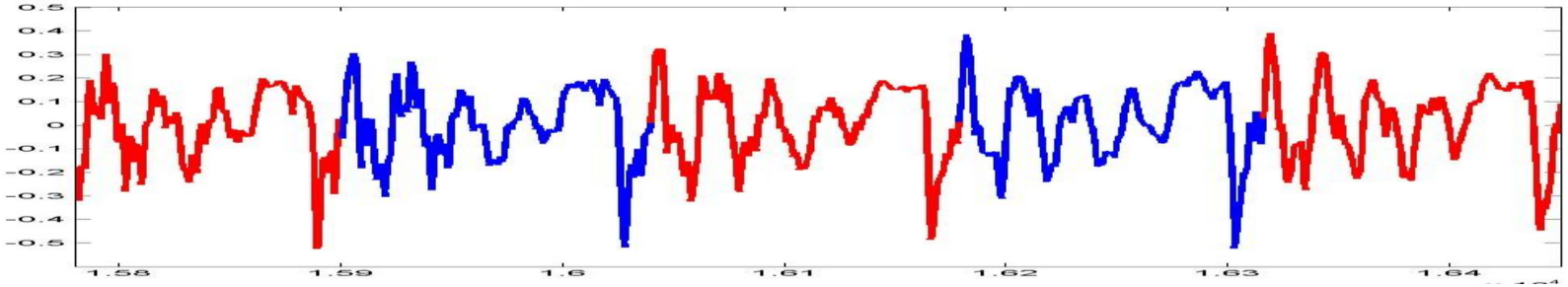


PSOLA for shortening a signal

- We could just patch them together directly, but that would generate some noise at the points where they're stuck together
- Instead we "window" each period we want to retain
  - Taper a longer segment that includes the desired period and the previous periods to 0 at the boundaries by applying a tapering window (e.g. a hamming window).
- The *Windowed* segments are brought closer to gether
  - Since we've extended the segments for tapering, there will be up to two samples at any time
- Overlapping samples from adjacent segments are added
  - Hence the name: "Pitch Synchronous Overlap Add" (PSOLA)

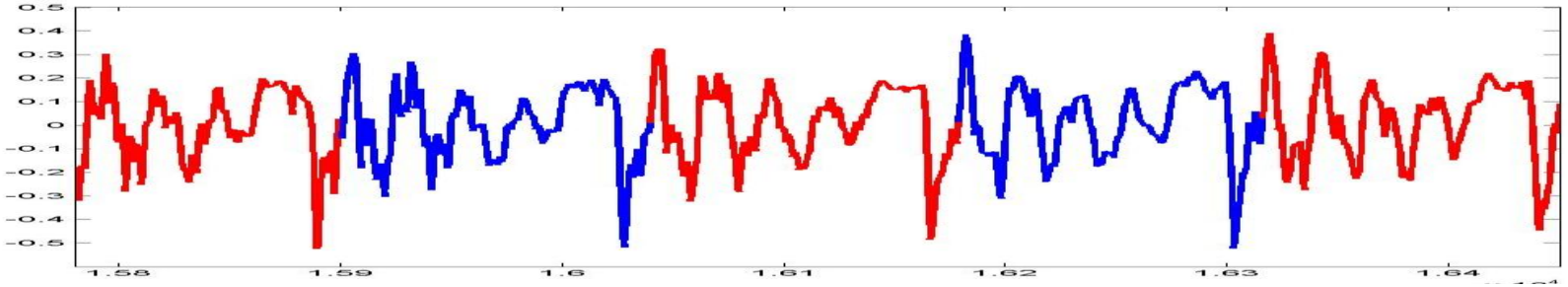
PSOLA in figures

- We wish to “delete” the red periods



PSOLA in figures

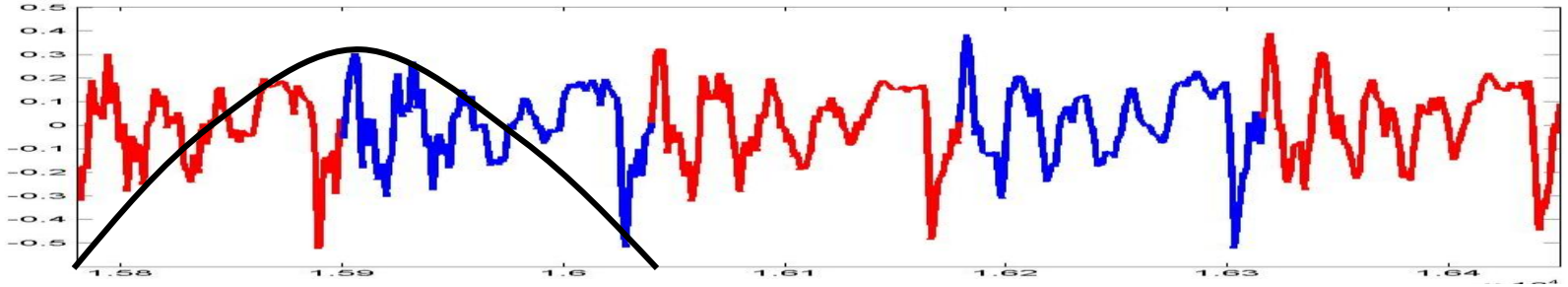
- We wish to “delete” the red periods



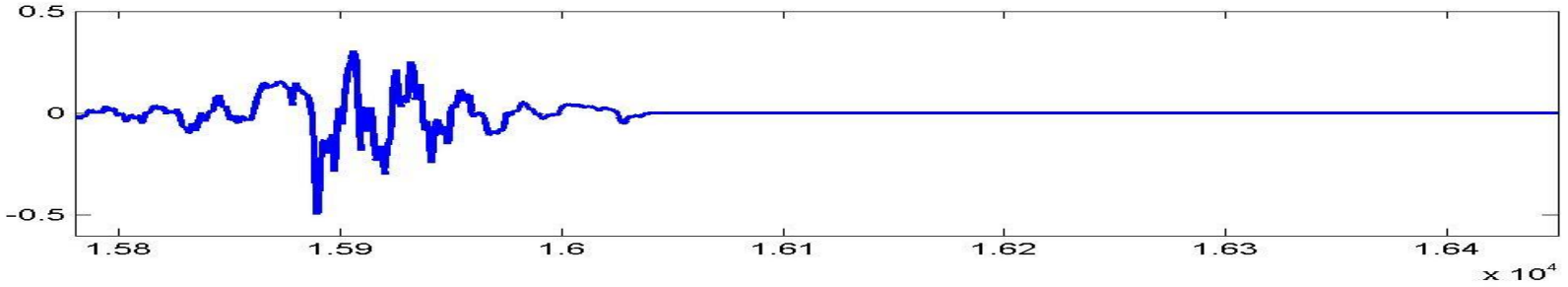
- For each blue period, append the previous period (regardless of whether it is red or blue) and taper the whole segment

PSOLA in figures

- We wish to “delete” the red periods

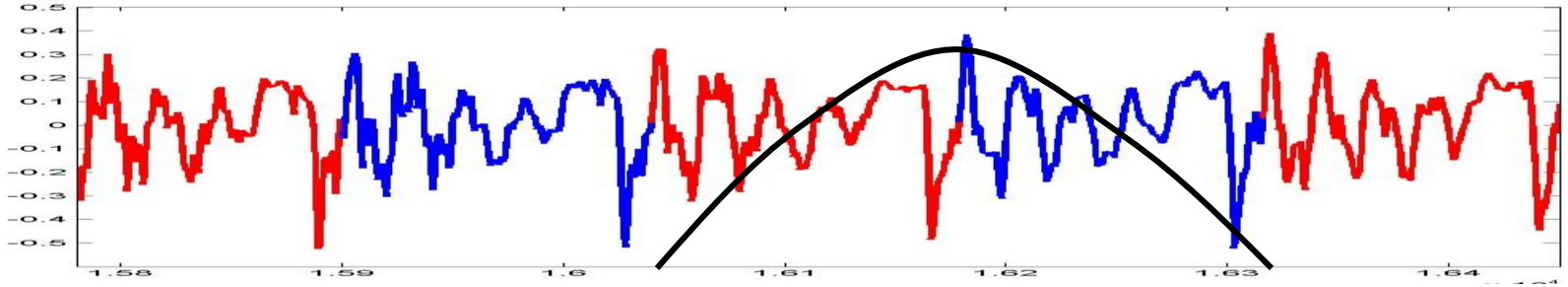


- For each blue period, append the previous period (regardless of whether it is red or blue) and taper the whole segment

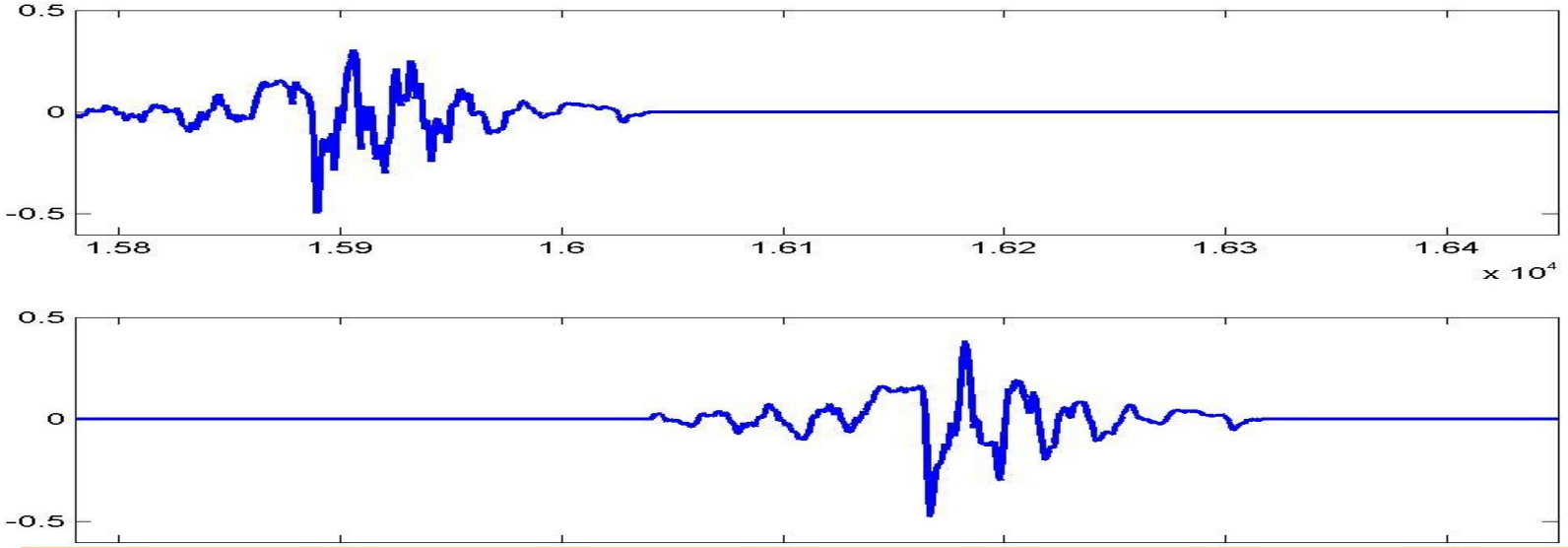


PSOLA in figures

- We wish to “delete” the red periods



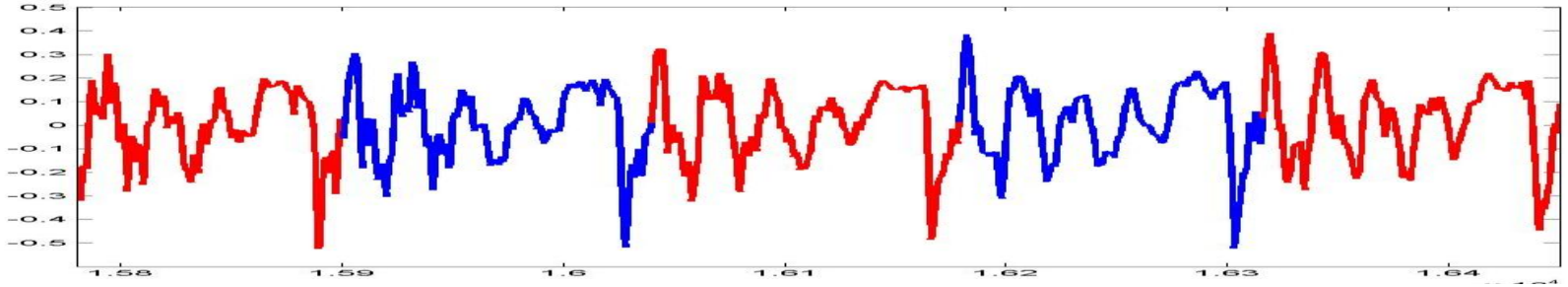
- For each blue period, append the previous period (regardless of whether it is red or blue) and taper the whole segment



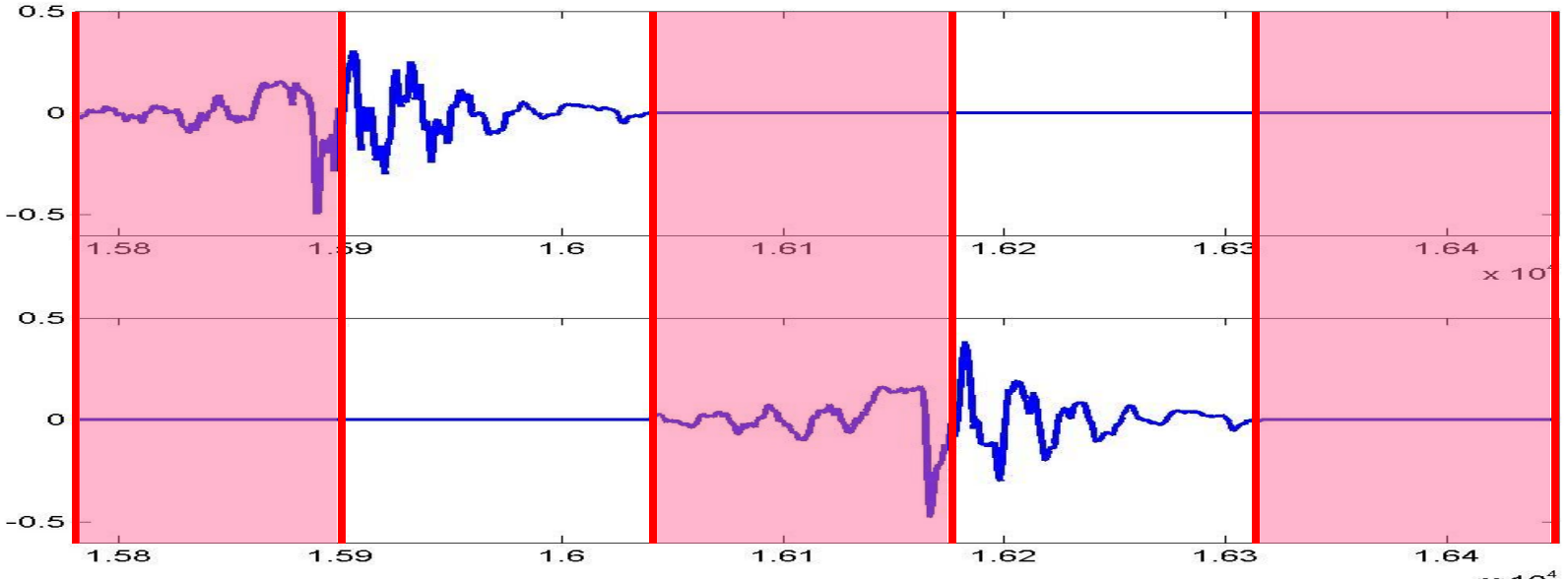
IMPORTANT: EACH TAPERING WINDOW MUST PEAK CLOSE TO THE INITIAL PEAK IN THE SELECTED PITCH PERIOD

PSOLA in figures

- We wish to “delete” the red periods

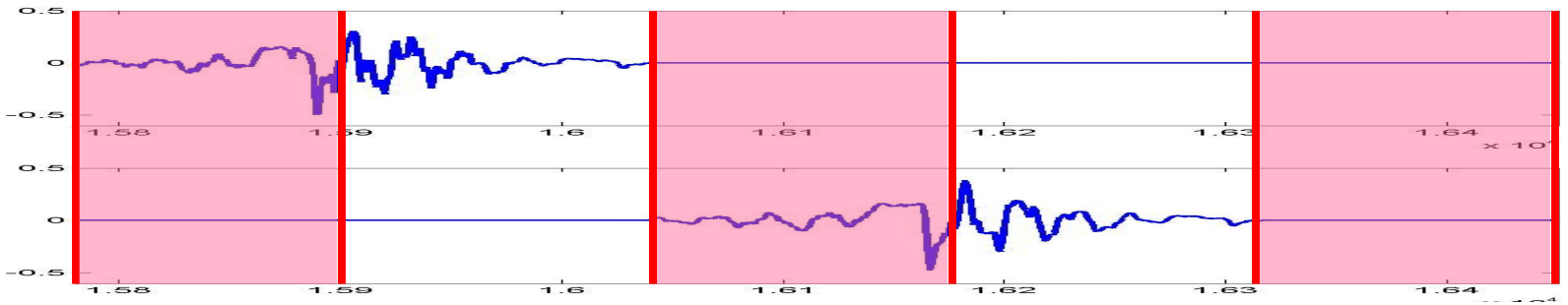


- The Regions in Red must be removed from the final signal



PSOLA in figures

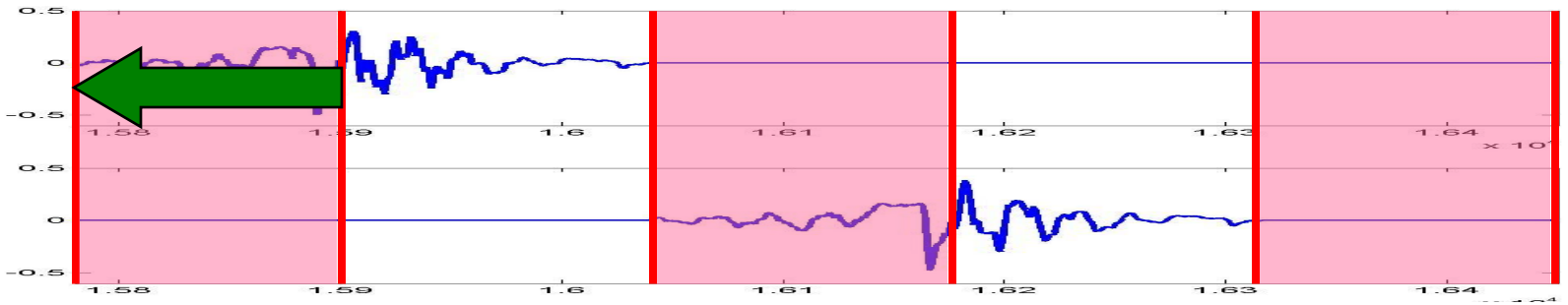
- The Regions in Red must be removed from the final signal



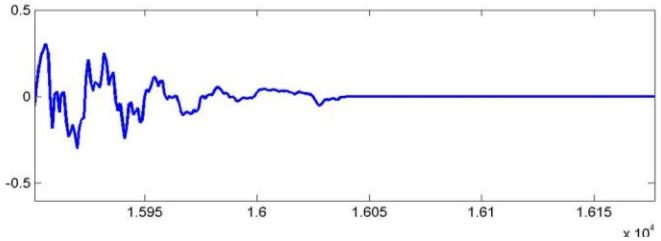
- Shift each segment such that the red line at the beginning of the segment lines us with the red line at the end of the previous segment
  - The k-th segment must be shifted before the (k+1)th segment

PSOLA in figures

- The Regions in Red must be removed from the final signal



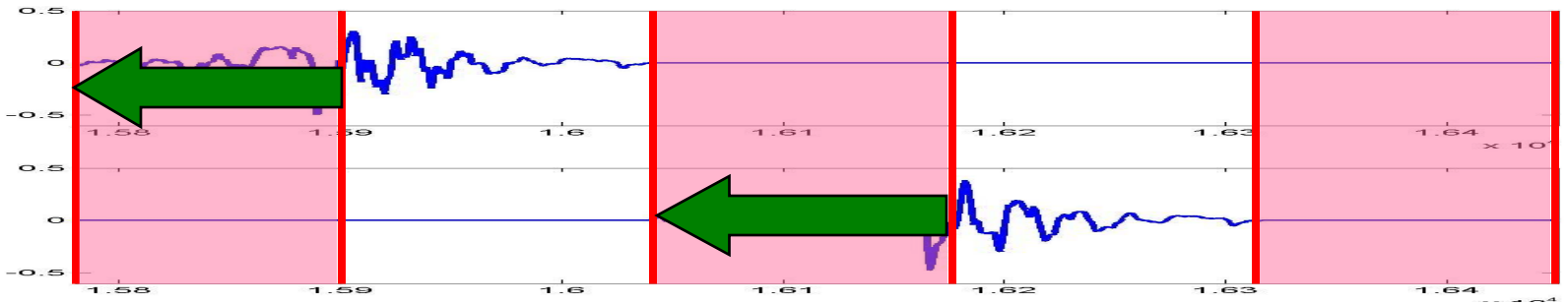
- Shift each segment such that the red line at the beginning of the segment lines us with the red line at the end of the previous segment
  - The k-th segment must be shifted before the (k+1)th segment



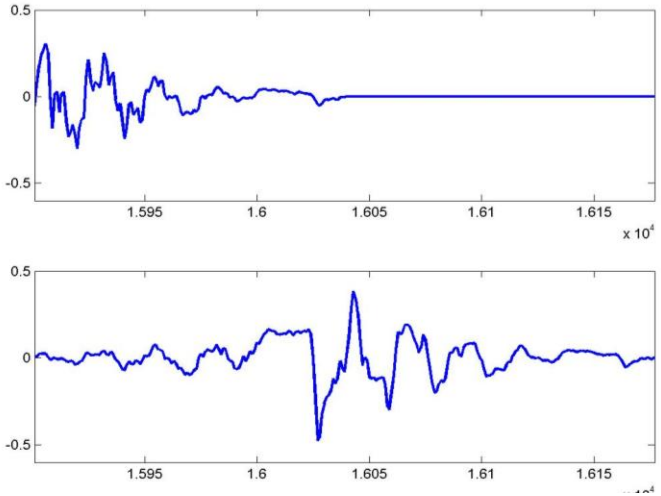
- The length is shorter and the boundary points also match up well

PSOLA in figures

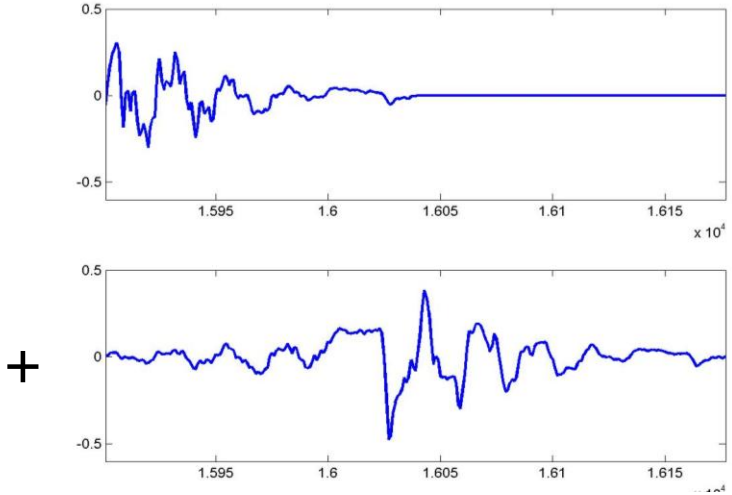
- The Regions in Red must be removed from the final signal



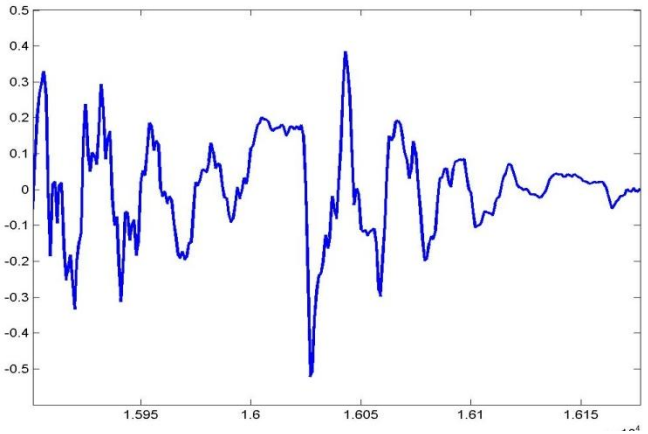
- Shift each segment such that the red line at the beginning of the segment lines us with the red line at the end of the previous segment
  - The k-th segment must be shifted before the (k+1)th segment



PSOLA in figures

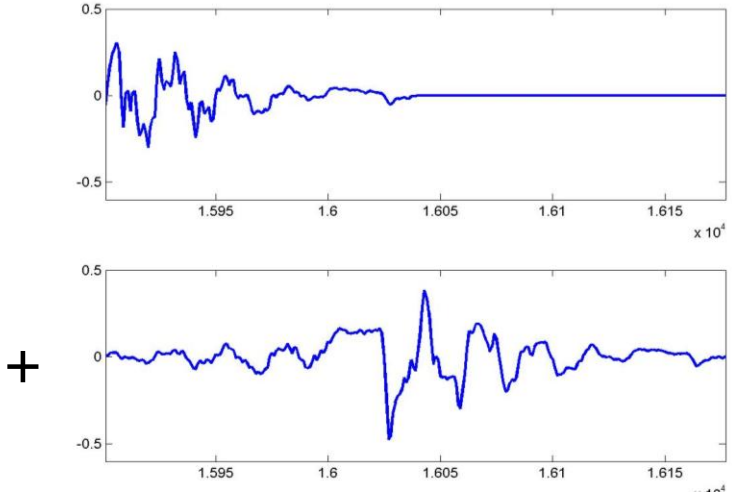


- Now Simply Add up the adjusted segments sample by sample



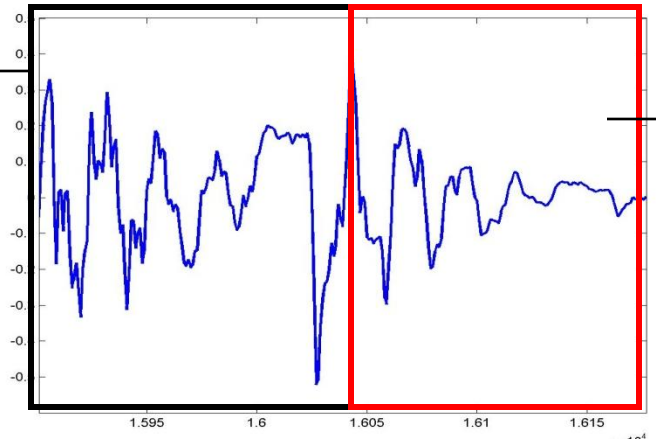
- We have a perfectly decent time-shortened but perceptually good signal

PSOLA in figures



- Now Simply Add up the adjusted segments sample by sample

This pitch period is a nearly perfect copy of the pitch periods in the orignal signal



This pitch period will become a nearly perfect copy when the NEXT pitch period is shifted in and added to it

- We have a perfectly decent time-shortened but perceptually good signal

PSOLA : Stretching

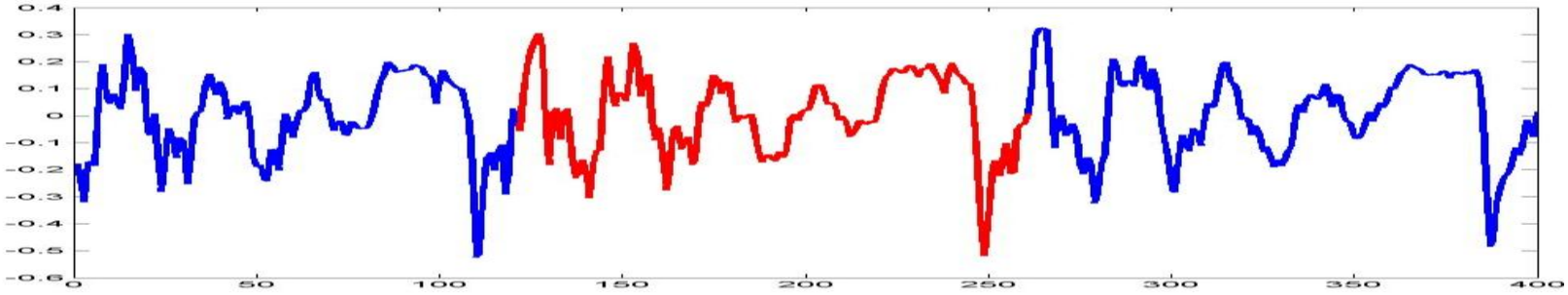
- We have considered SHORTENING a signal, but what about stretching?

- We use a very similar procedure for stretching:

  - Instead of deleting intermediate periods, we move adjacent periods (after windowing) away from each other to generate space for new pitch periods between them.
  - We then copy the closest pitch
  - One then performs overlap and add as before.

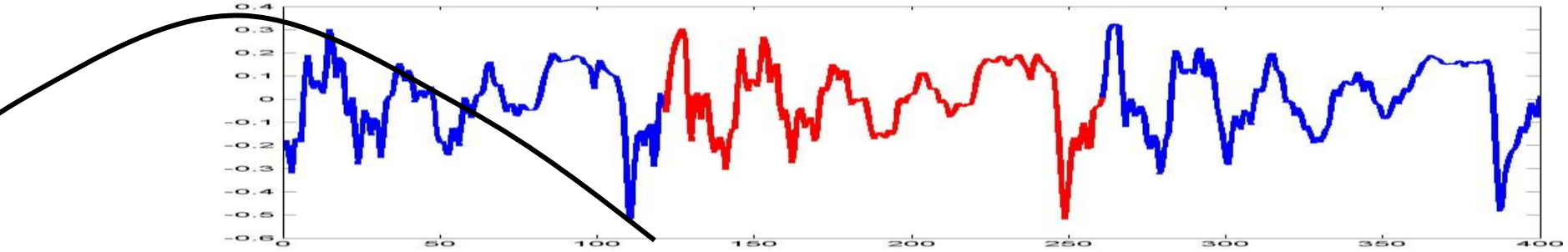
PSOLA Stretching

- Every Pitch Period must be replicated

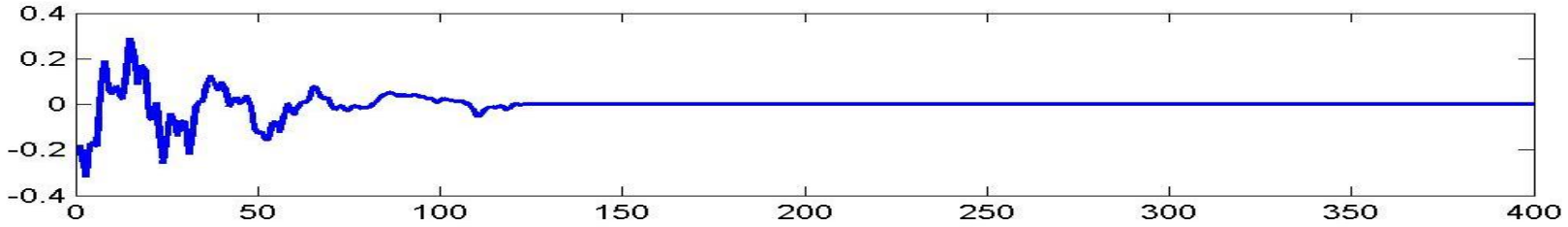


PSOLA Stretching

- Every Pitch Period must be replicated

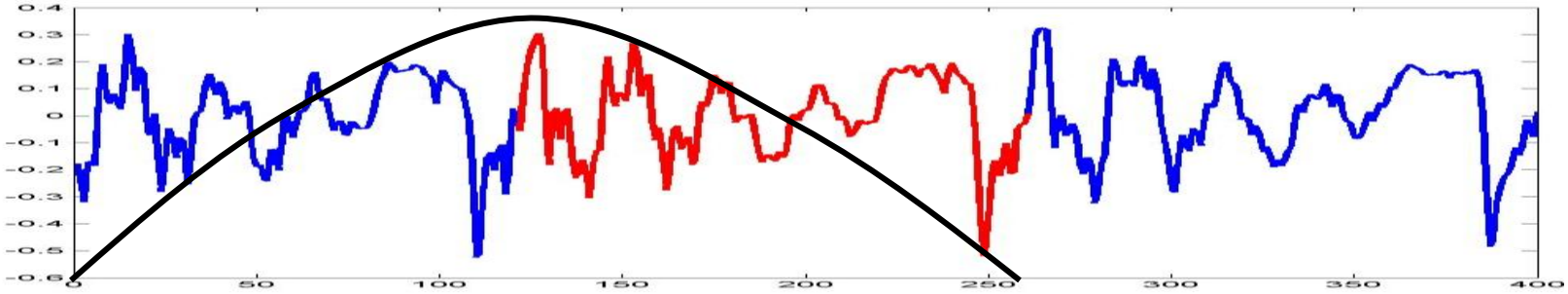


- Window each pitch period (along with preceding pitch period)

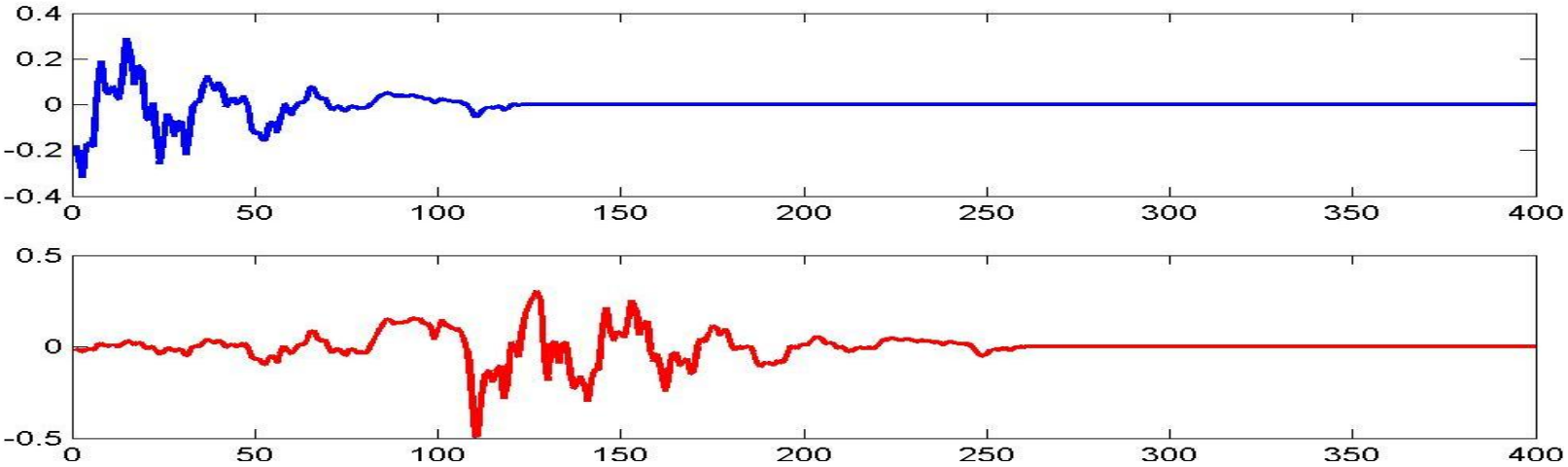


PSOLA Stretching

- Every Pitch Period must be replicated

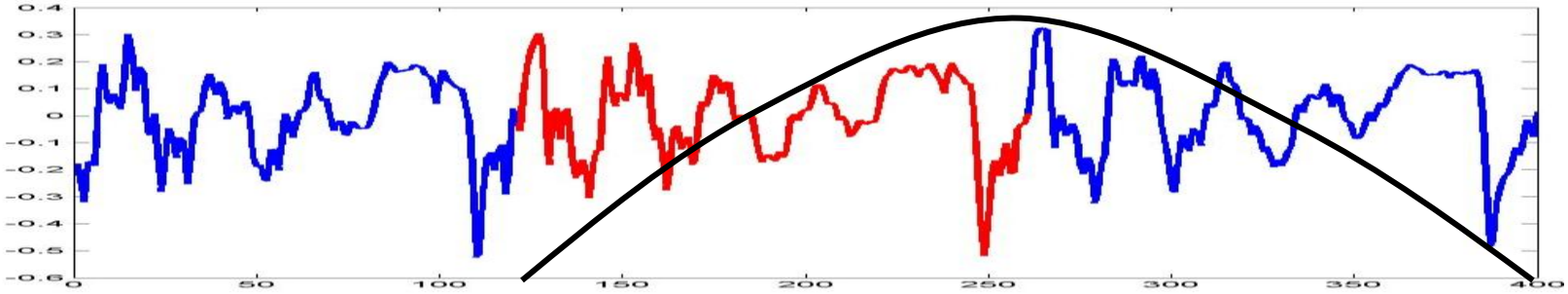


- Window each pitch period (along with preceding pitch period)

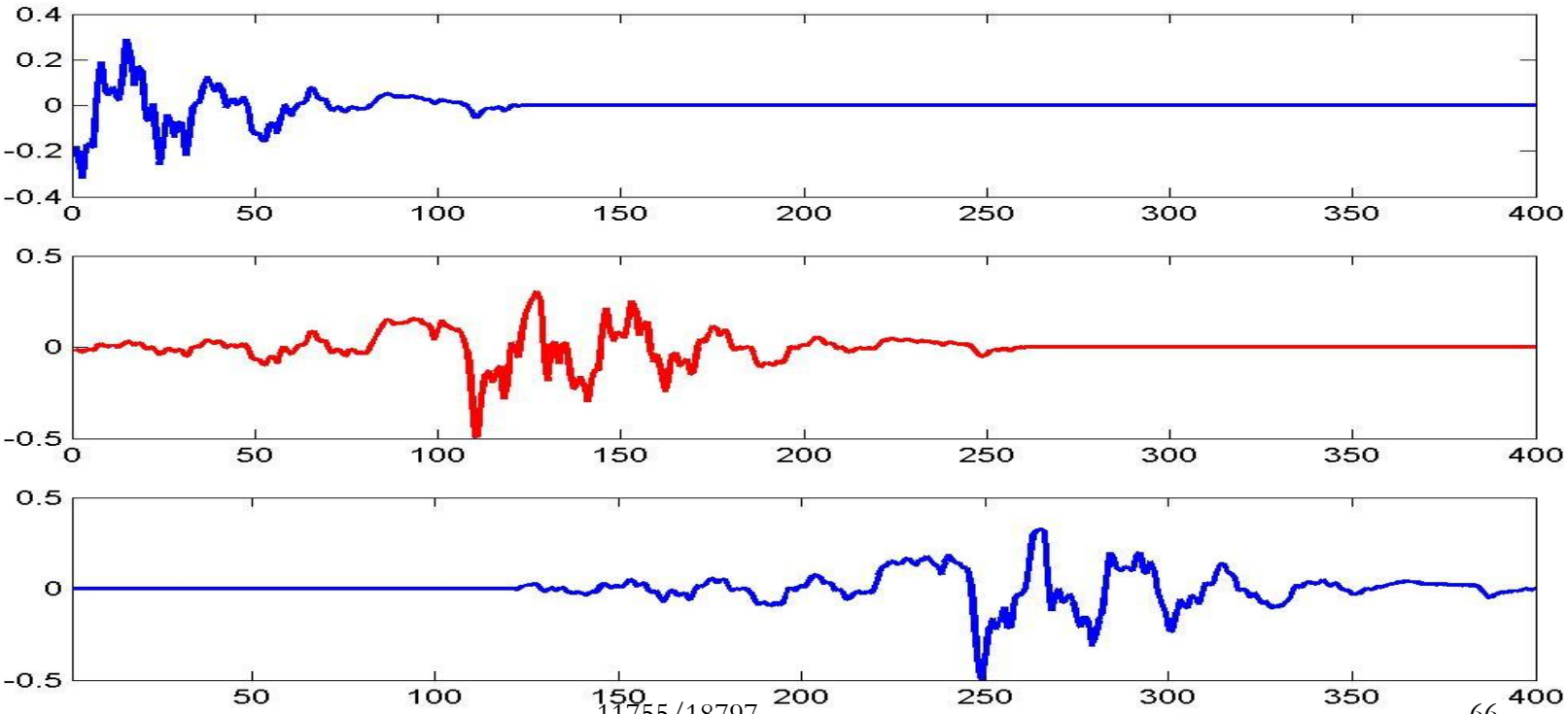


PSOLA Stretching

- Every Pitch Period must be replicated

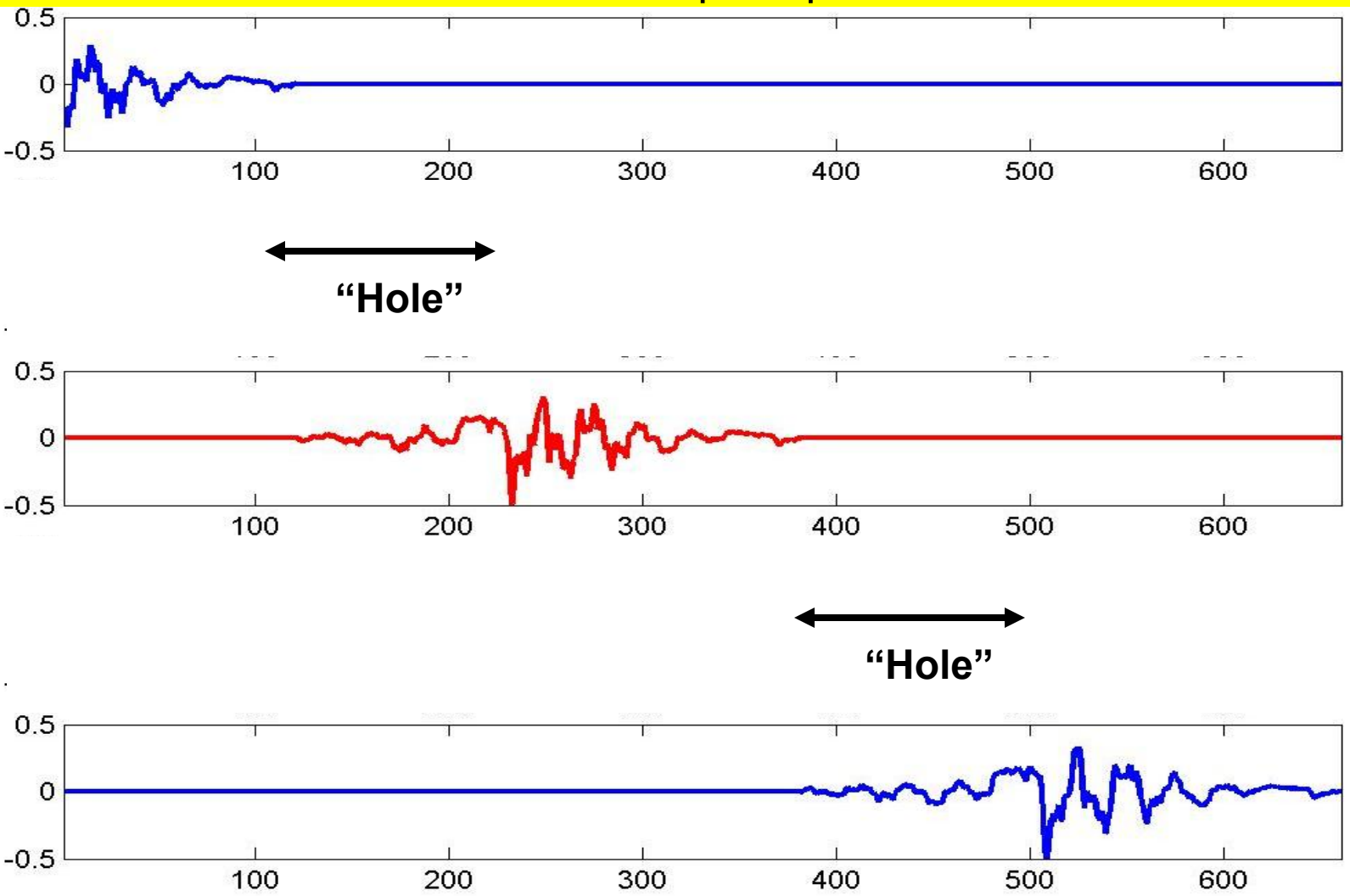


- Window each pitch period (along with preceding pitch period)



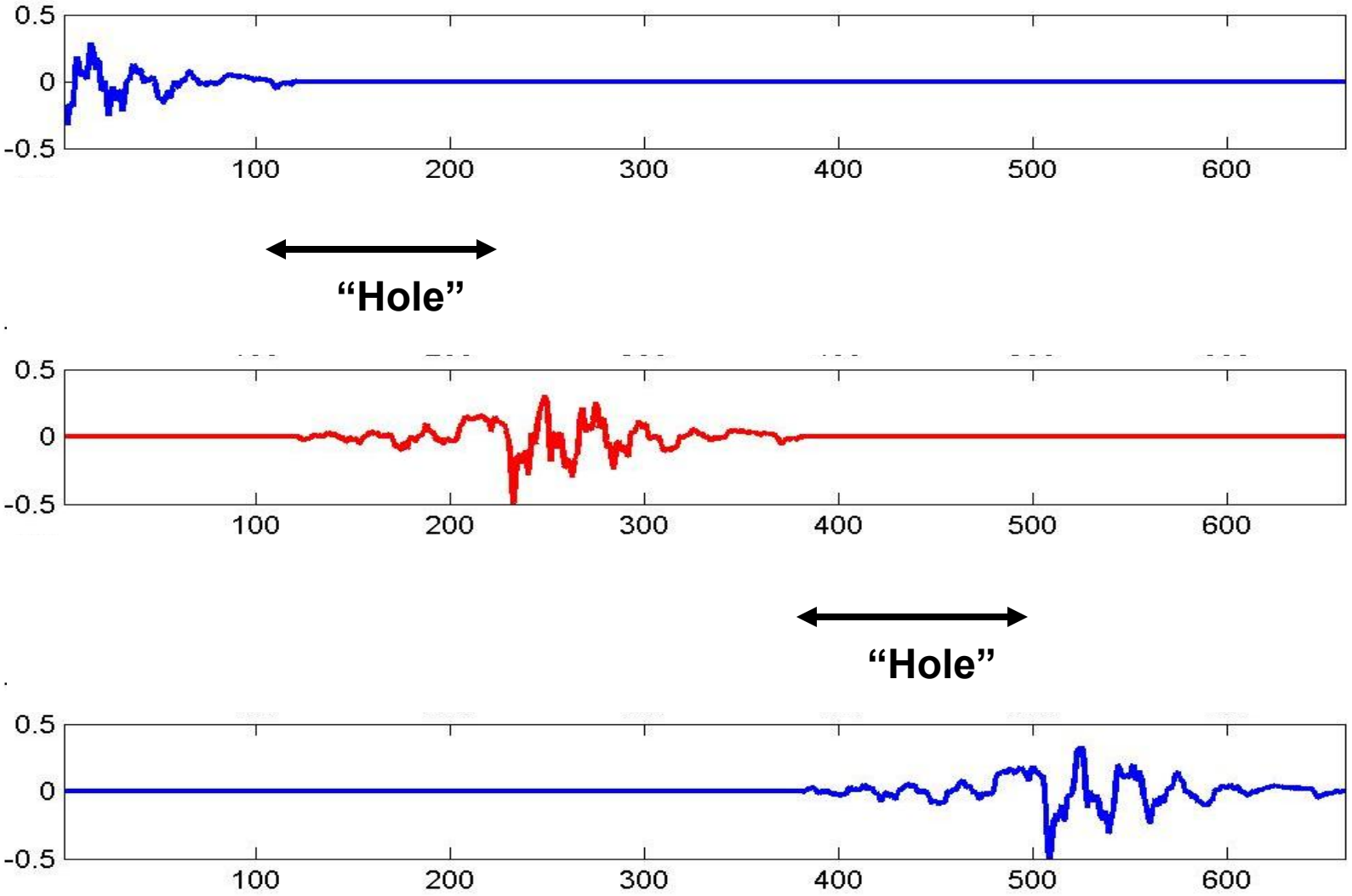
PSOLA Stretching

- Slide windowed pitch periods out (by integral pitch periods). This creates holes where there is no pitch period



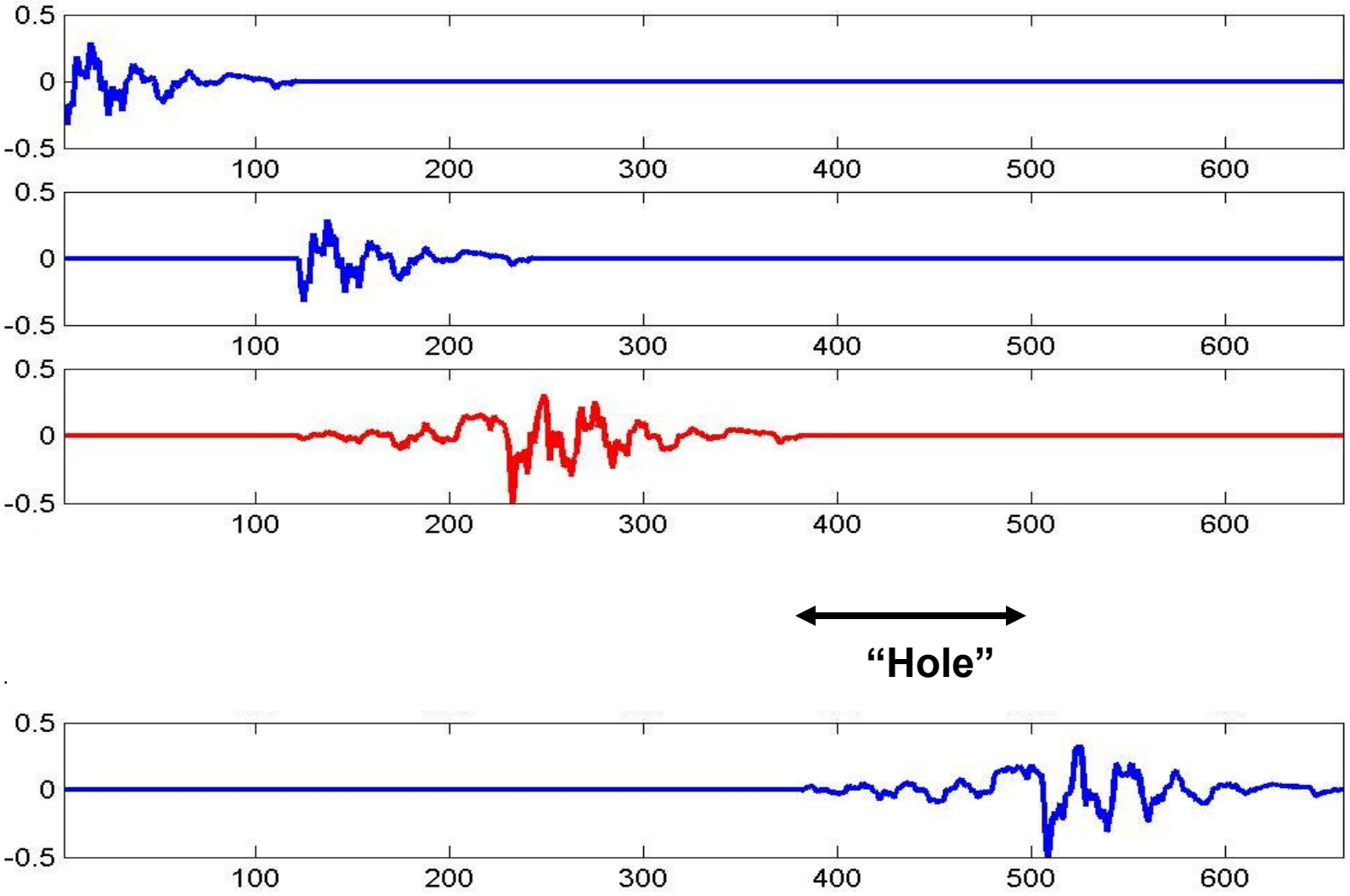
PSOLA Stretching

- Replicate each segment shifted by its own pitch period



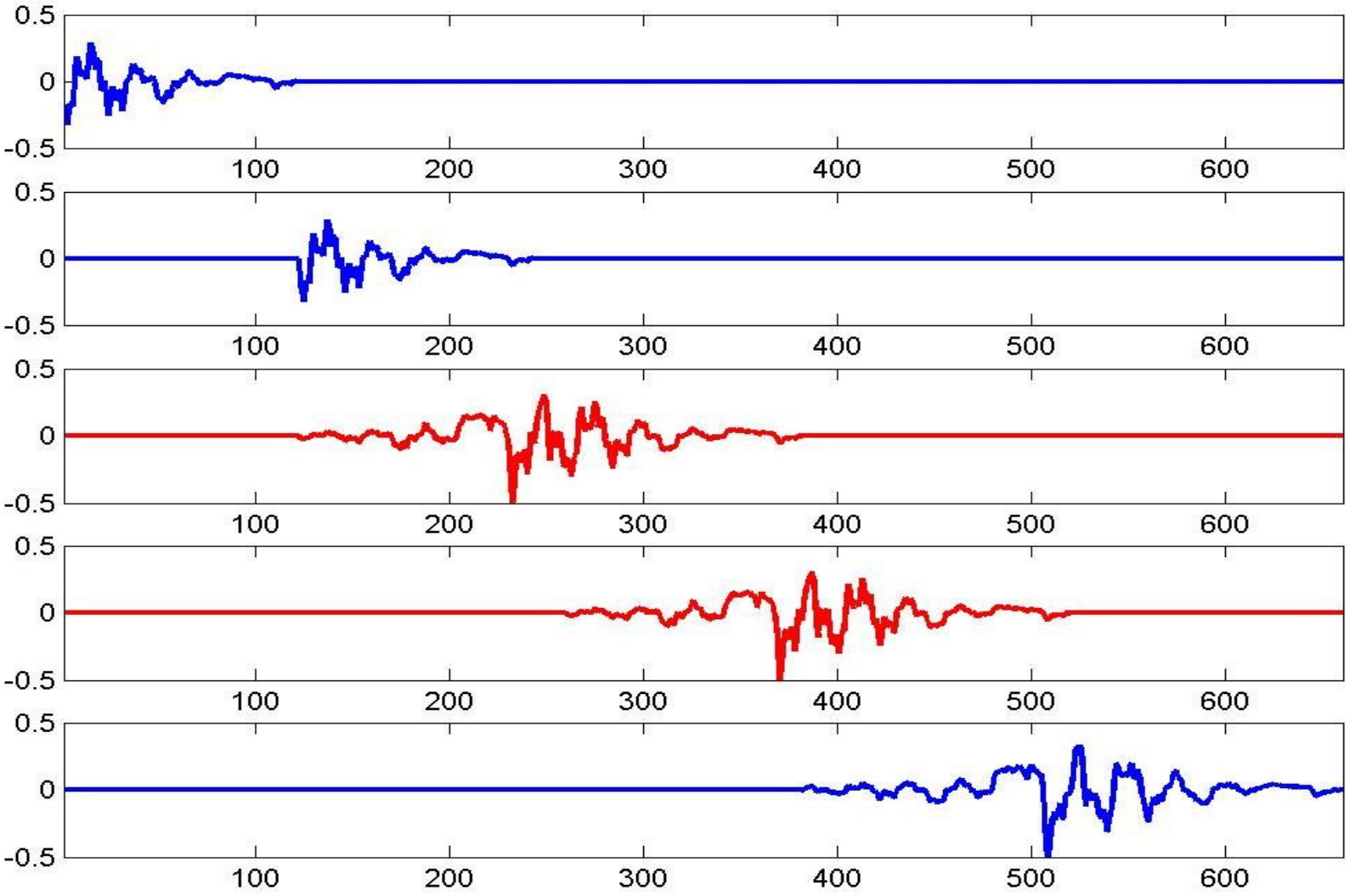
PSOLA Stretching

- Replicate each segment shifted by its own pitch period



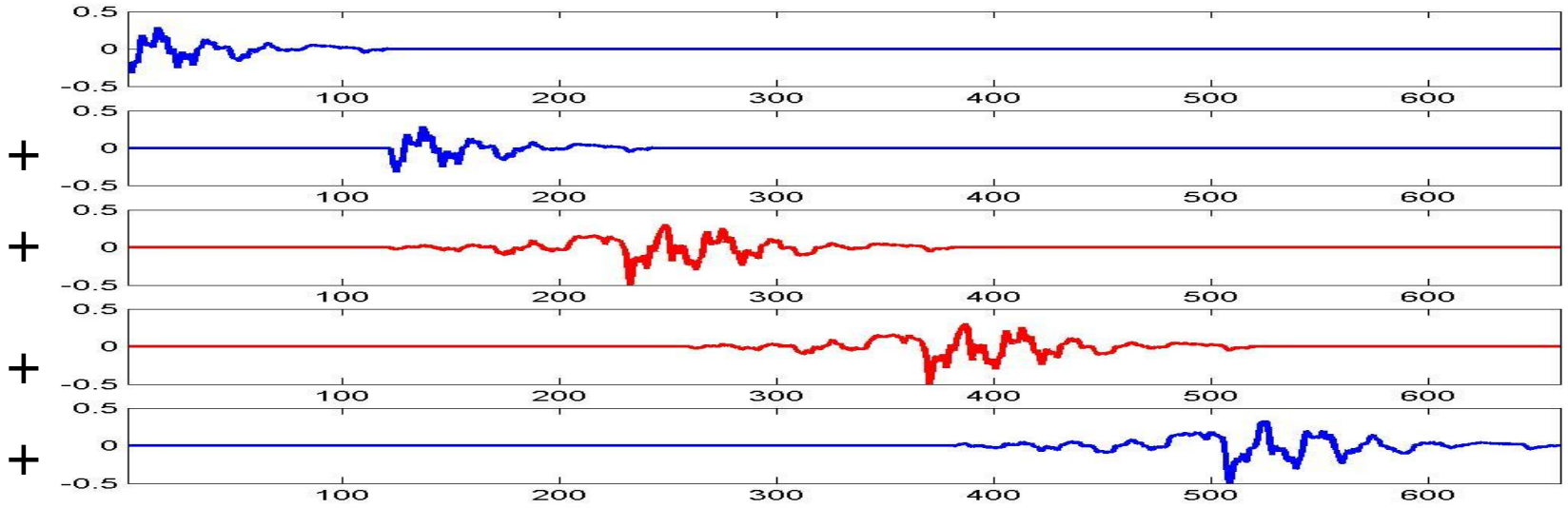
PSOLA Stretching

- Replicate each segment shifted by its own pitch period



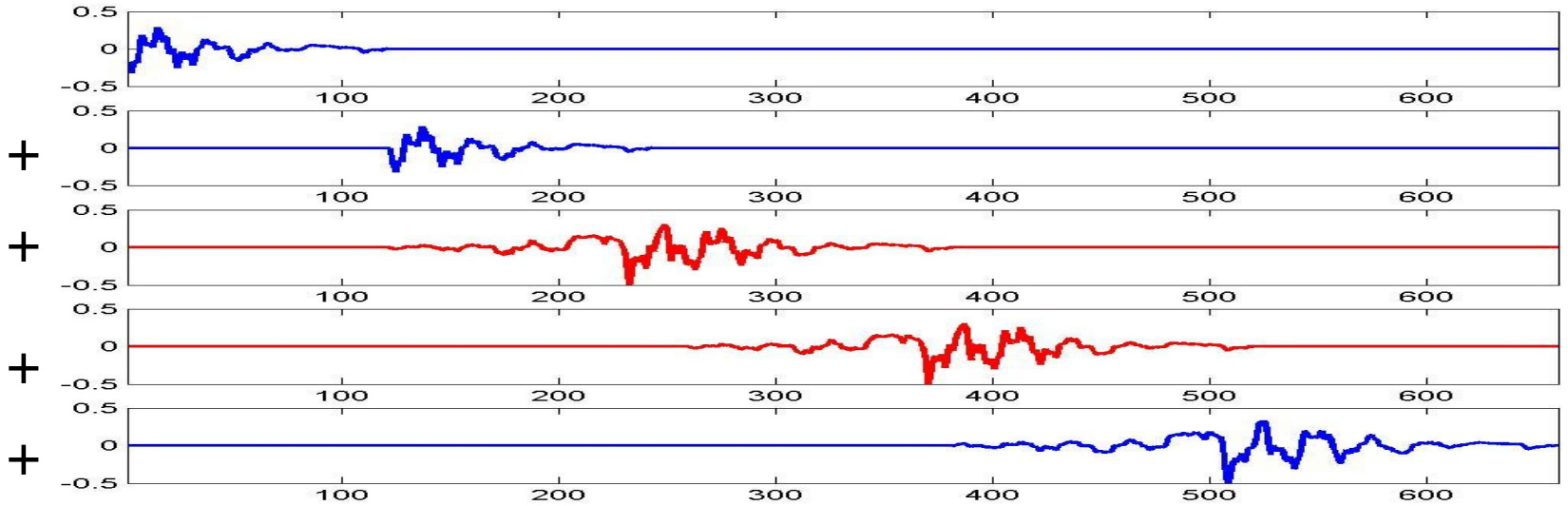
PSOLA Stretching

- Sum all segments up

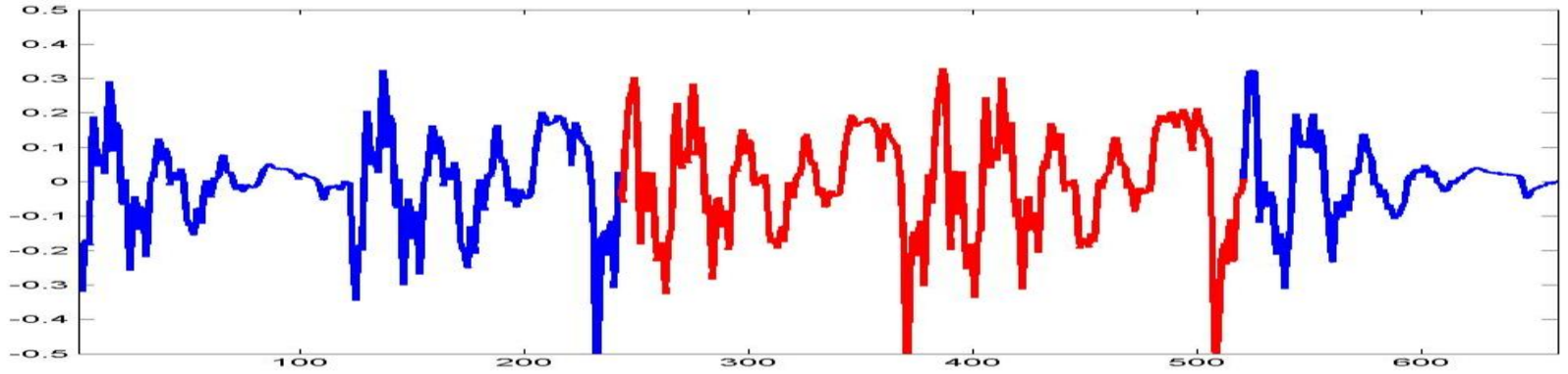


PSOLA Stretching

- Sum all segments up



- The result is a longer (double length here) perceptually reasonable signal



PSOLA

- The most important component of PSOLA is proper identification of the pitch periods
  - Ideally these would identify the position of the first peak in the pitch period
    - These form the basic reference points
  - The segments of the signal that repeat
- Unfortunately this is a very hard problem in most audio signals
  - Particularly polyphonic signals
- Several good algorithms exist for speech, solo voices and music with single instruments
  - PSOLA is good for these cases
- Advantage: PSOLA a very simple algorithm to implement
  - < 100 lines of matlab code, once the pitch periods are identified
  - If pitch estimation is good, the resulting signal is very good with few artifacts

Finding the Pitch

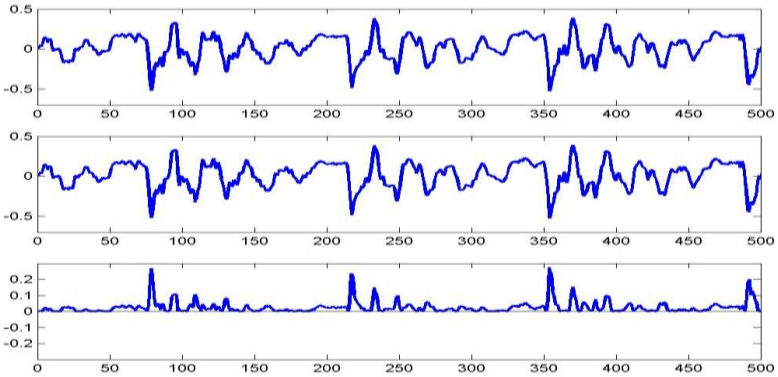
- A simple algorithm for finding the pitch is based on autocorrelations
- Based on a very simple principle: A signal adds up best with a shifted version of itself when the shift is 0
  - Or any integer of the period of the signal
- The autocorrelation of a signal is simply obtained by multiplying the signal by a shifted version of itself (sample-by-sample) and adding all the samples

$$R(T) = \sum_t x(t)x(t-T)$$

- This has a maximum value at T=0 and T=pitch period

Autocorrelations

- Consider the following segment:

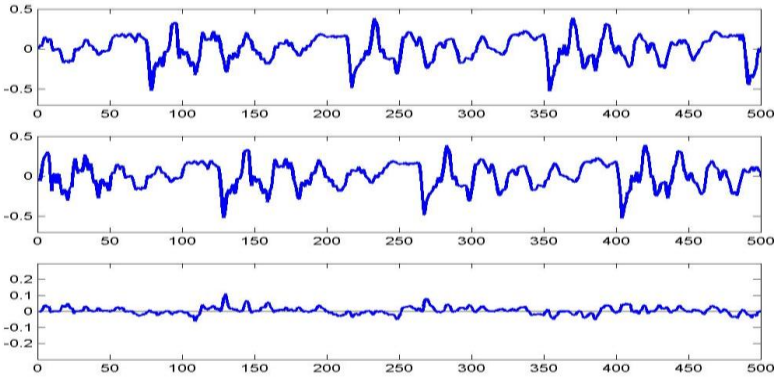


Original segment:

The same segment

The product of the two

Now lets shift the second segment a bit



Original segment:

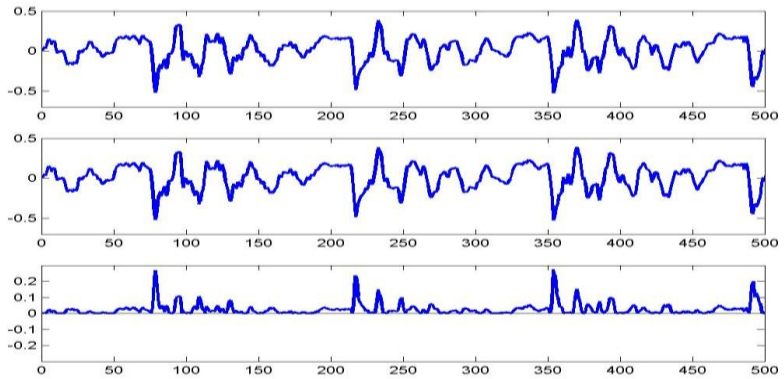
The segment, slightly shifted

The product of the two

The product is smaller in the second case, where the second signal is a slightly shifted version of the first one. The sum of all values in the product will be smaller also

Autocorrelations

- Consider the following segment:

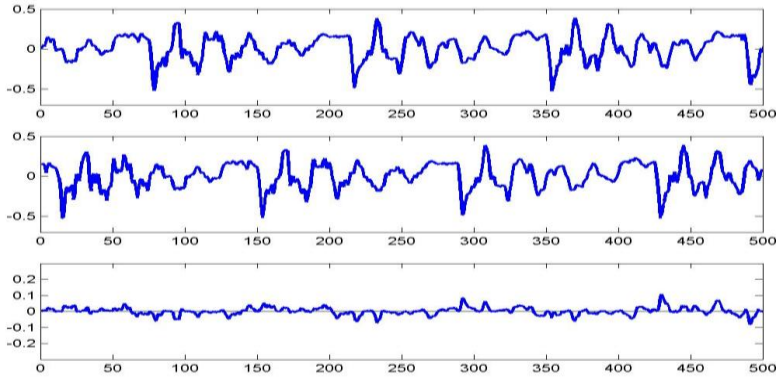


Original segment:

The same segment

The product of the two

Lets shift the second segment more



Original segment:

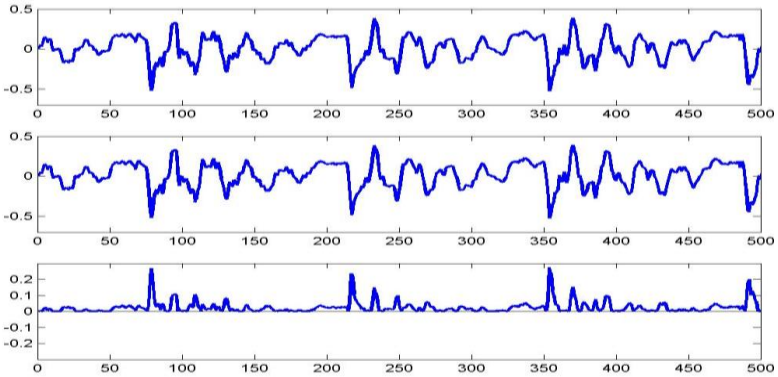
The segment, more shifted

The product of the two

The product is smaller in the shifted case, than in the unshifted case. So is the sum of all samples in the product

Autocorrelations

- Consider the following segment:

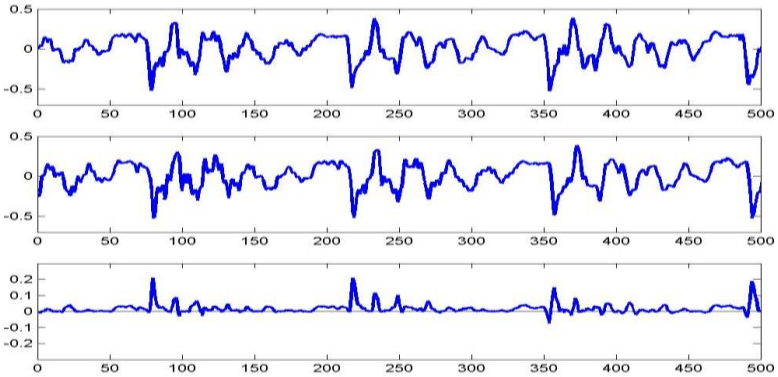


Original segment:

The same segment

The product of the two

Lets shift the second segment by an entire pitch period



Original segment:

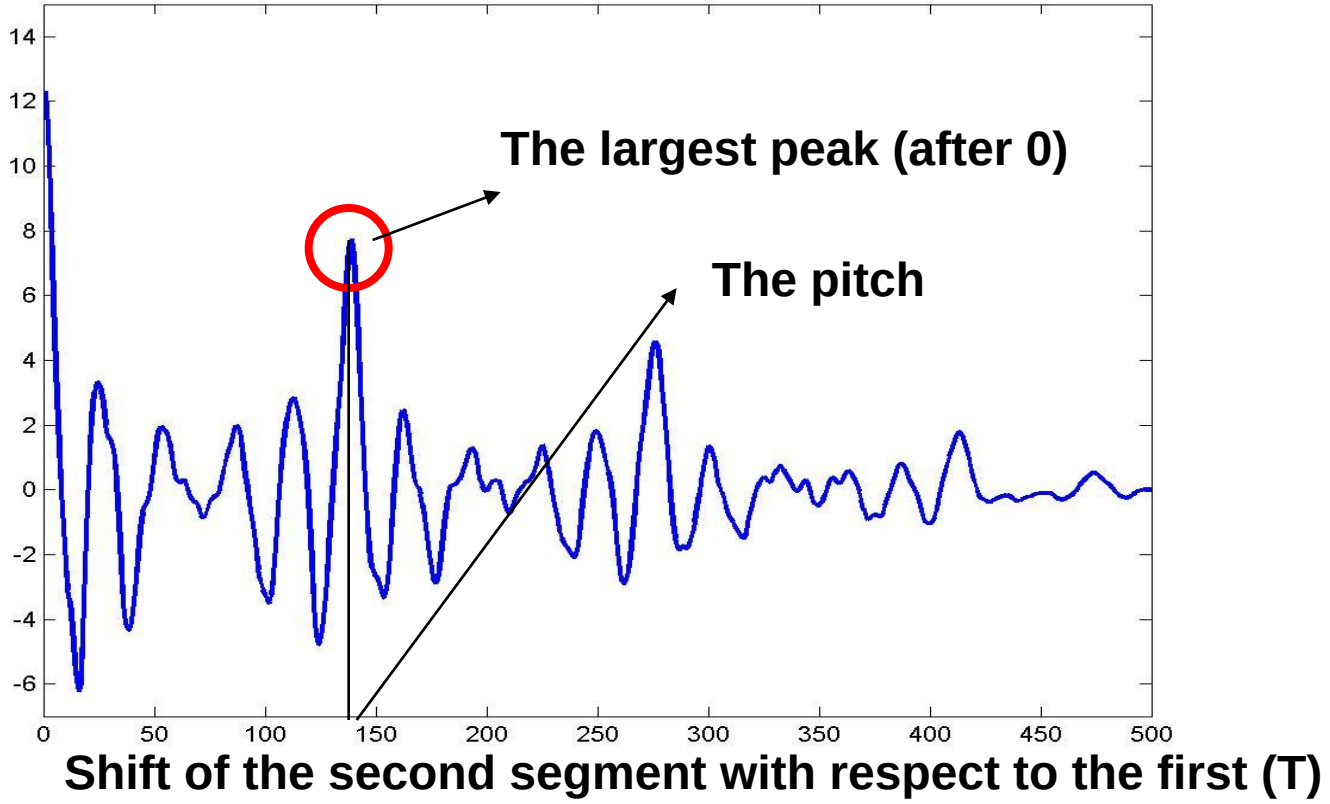
The segment, by a pitch period

The product of the two

When the shift is exactly one pitch period the product becomes large again! The sum of the samples in the product will also peak for this shift

Autocorrelations

- The autocorrelation (sum of samples in the product) as a function of shift



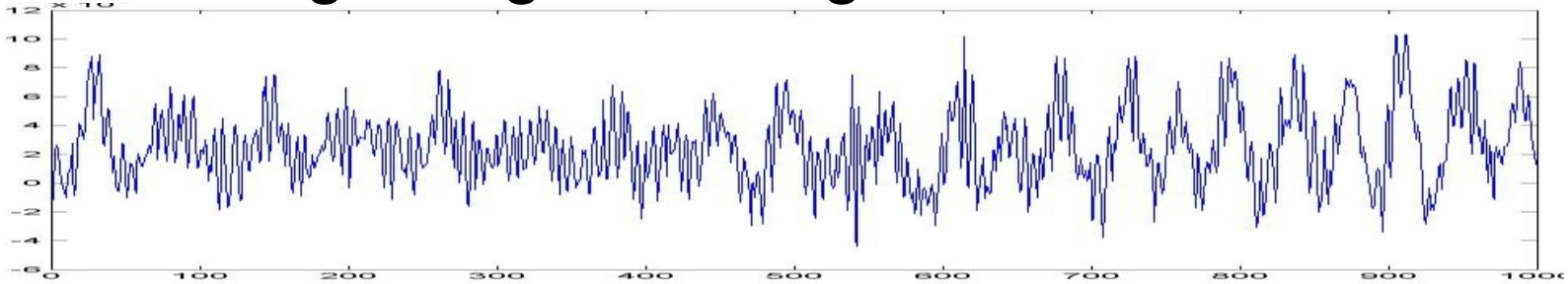
When the shift is exactly one pitch period the autocorrelation peaks
To estimate the pitch simply find the shift at which this peak occurs

Finding Pitch Periods

- Start at the beginning of the recording and find the first peak
  - We assume this is the beginning of our first pitch segment
- Consider a 30-50ms segment of signal from the beginning of the pitch segment.
  - Window it to taper the edges with a tapering window [IMPORTANT]
  - The segment has zero value outside the window
- Compute autocorrelation values at various shifts for the (windowed) segment
- Find the position of the peak in the autocorrelation
- This gives us the position of the beginning of the next pitch segment
- Repeat the above operation from this location

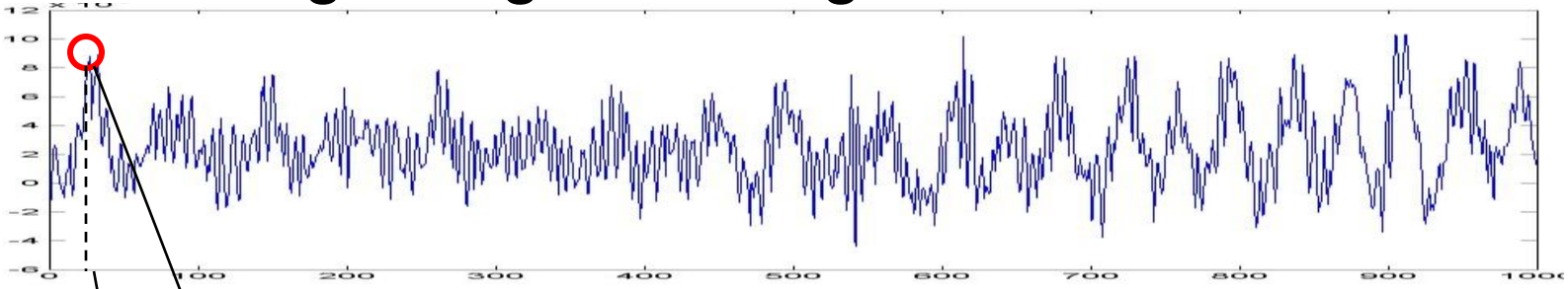
Finding pitch periods

- The beginning of the signal



Finding pitch periods

- The beginning of the signal

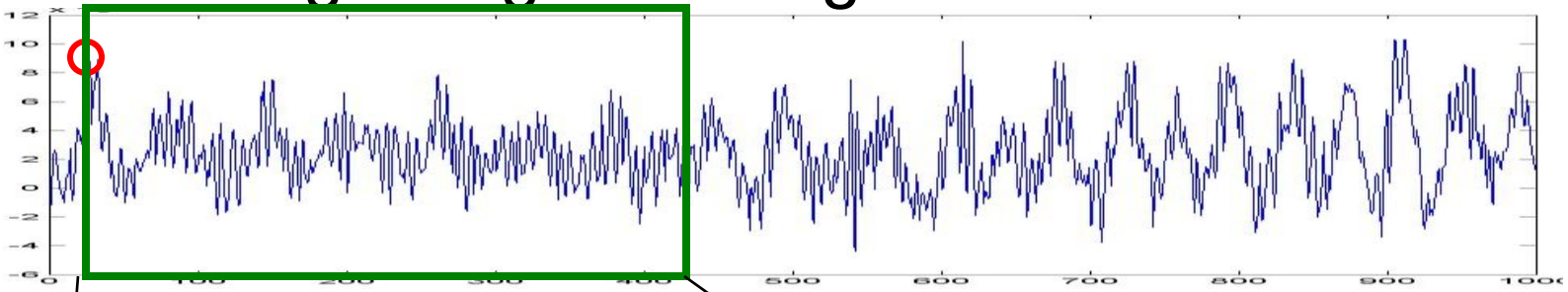


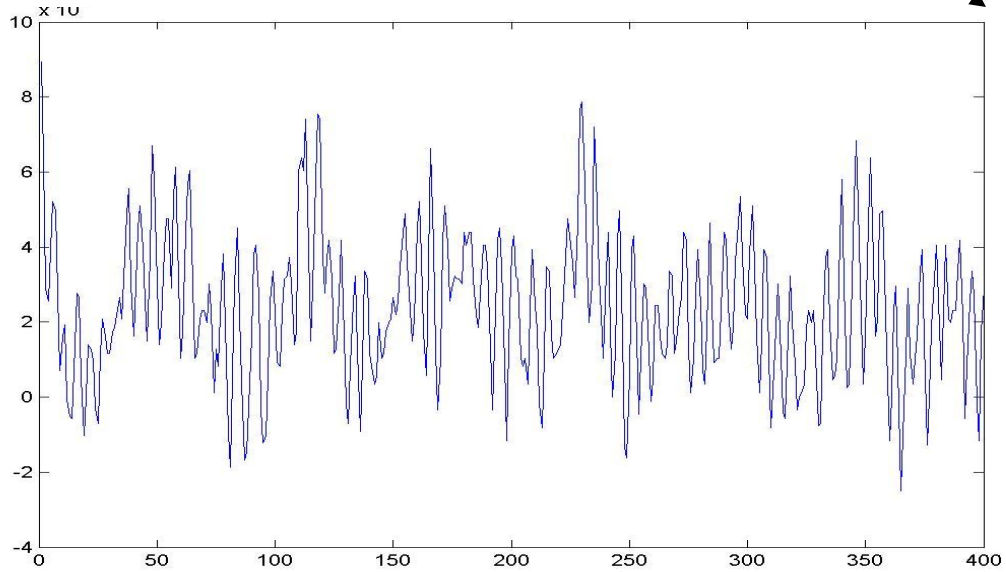
First peak:: First pitch tick

Beginning of first pitch period

Finding pitch periods

- The beginning of the signal

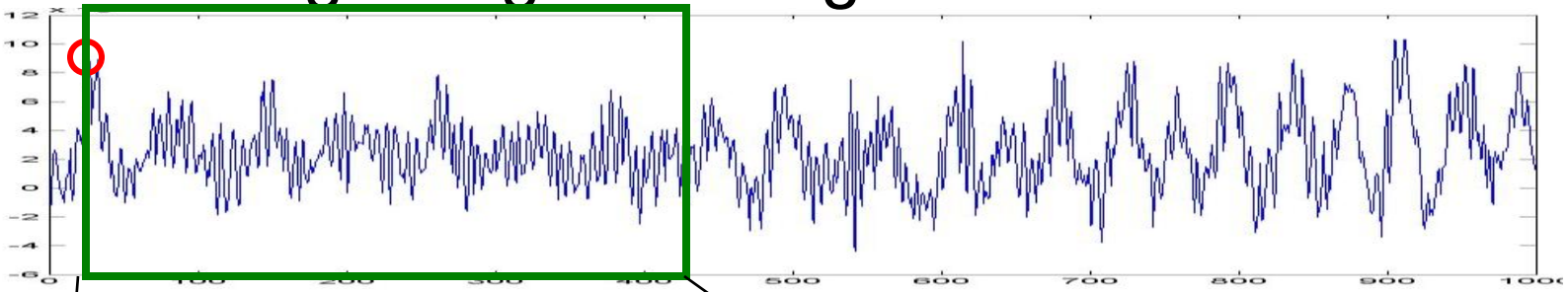


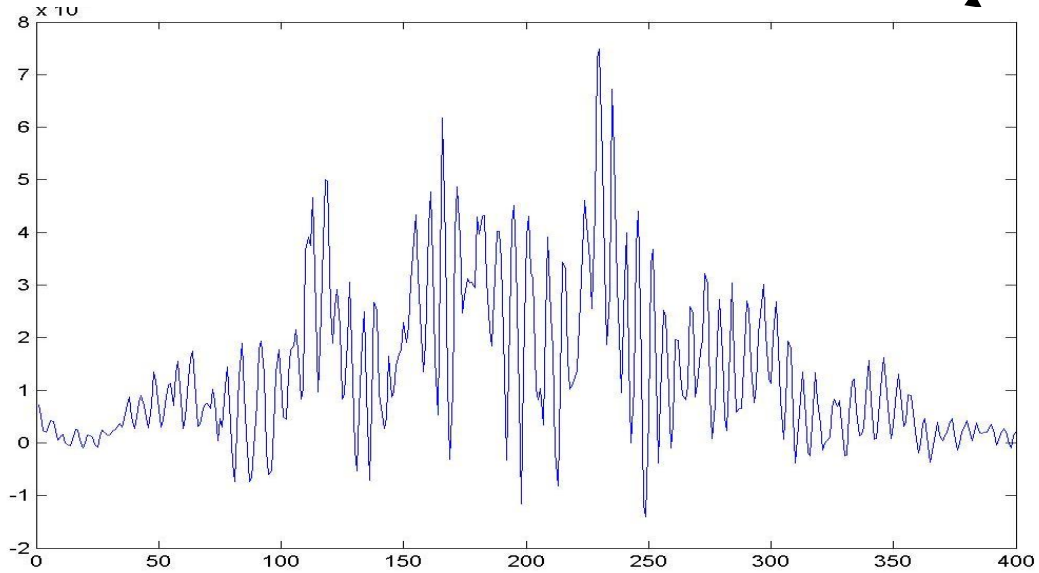


Grab the next 32ms of the signal

Finding pitch periods

- The beginning of the signal



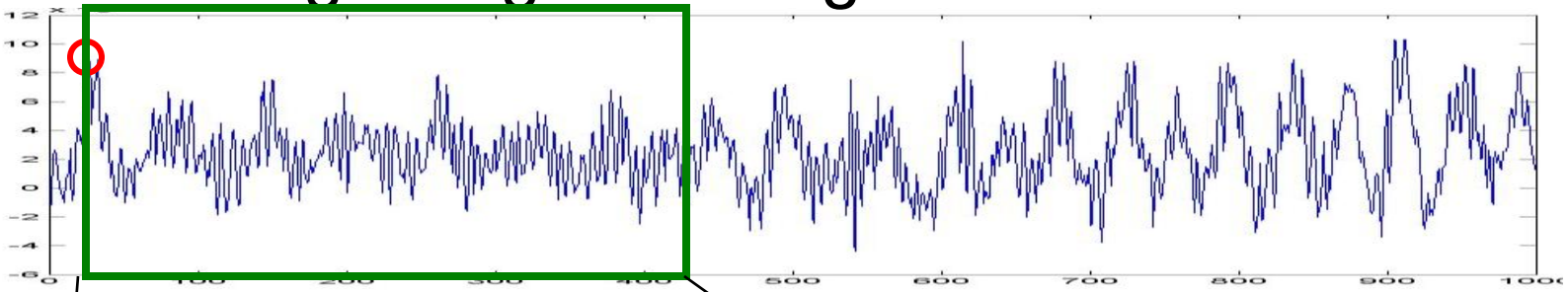


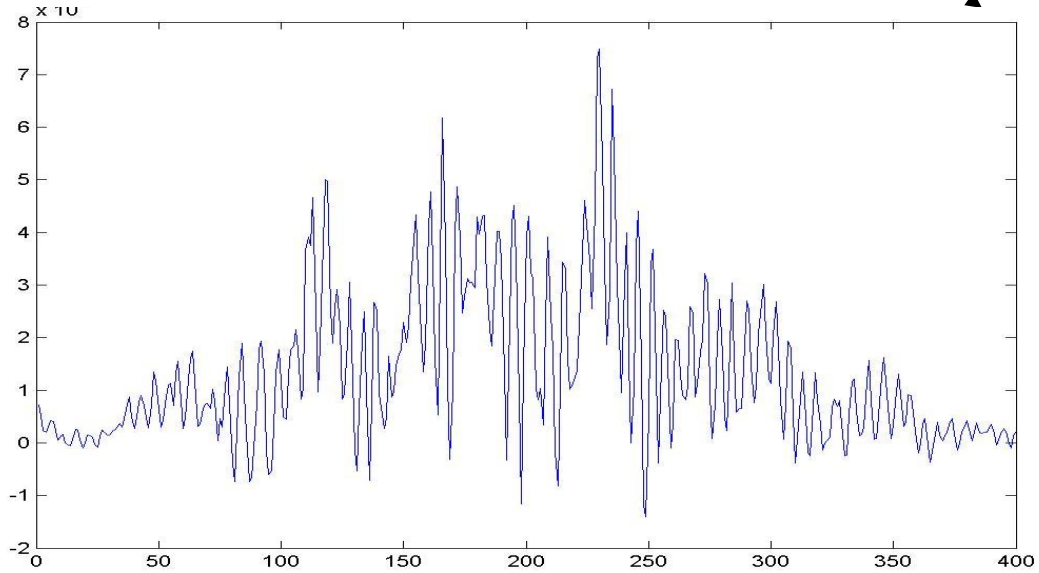
Grab the next 32ms of the signal

Taper the signal

Finding pitch periods

- The beginning of the signal





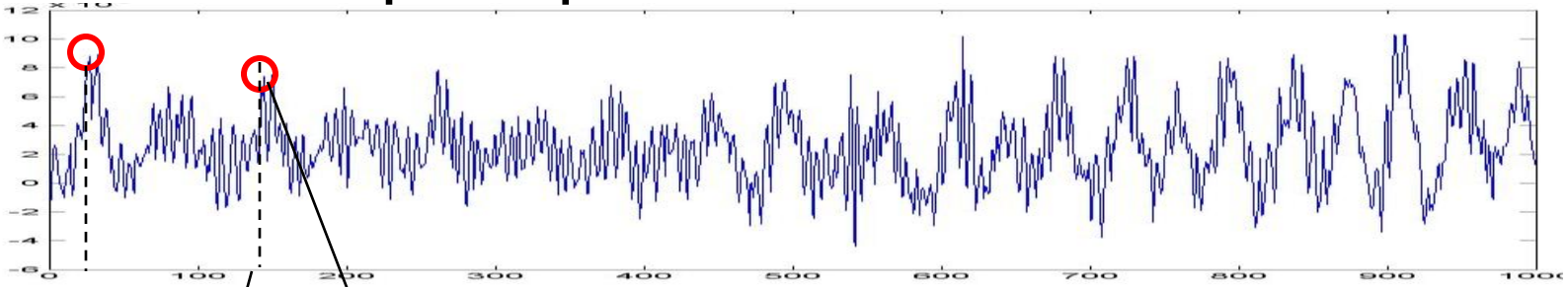
Grab the next 32ms of the signal

Taper the signal

Compute the autocorrelation and find the location of the peak

Finding pitch periods

- The next pitch period



Second pitch tick

Beginning of second pitch period

Grab the next 32ms of the signal

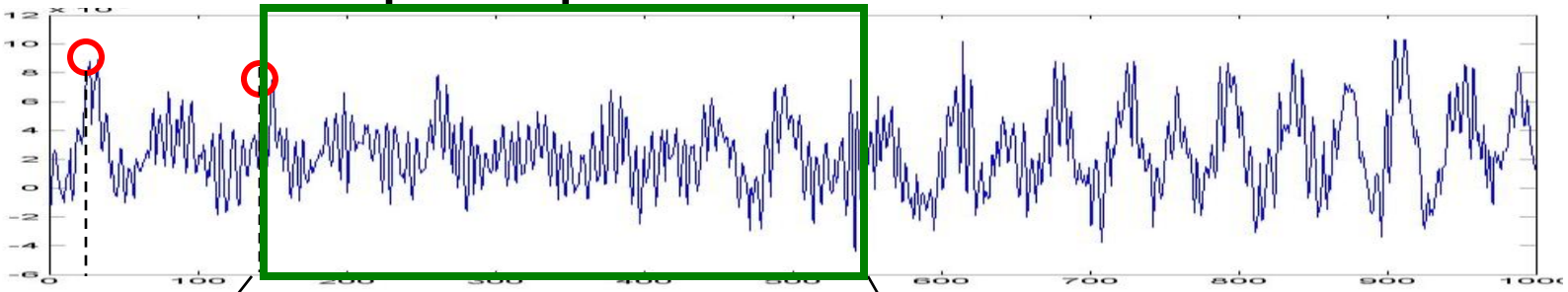
Taper the signal

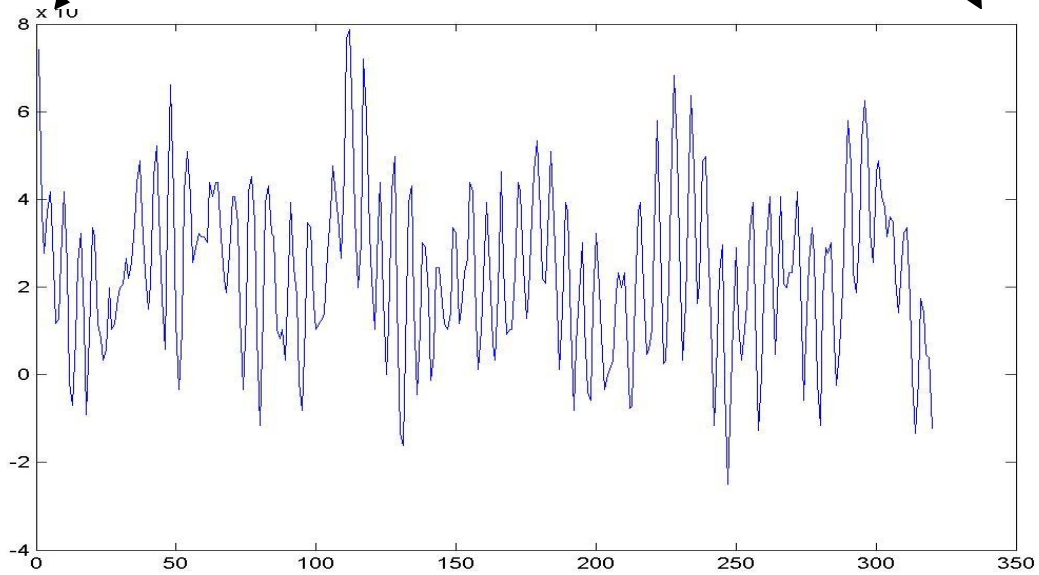
Compute the autocorrelation and find the time lag (shift) of the peak

Advance the marker by this lag to mark the beginning of the next pitch period

Finding pitch periods

- The next pitch period





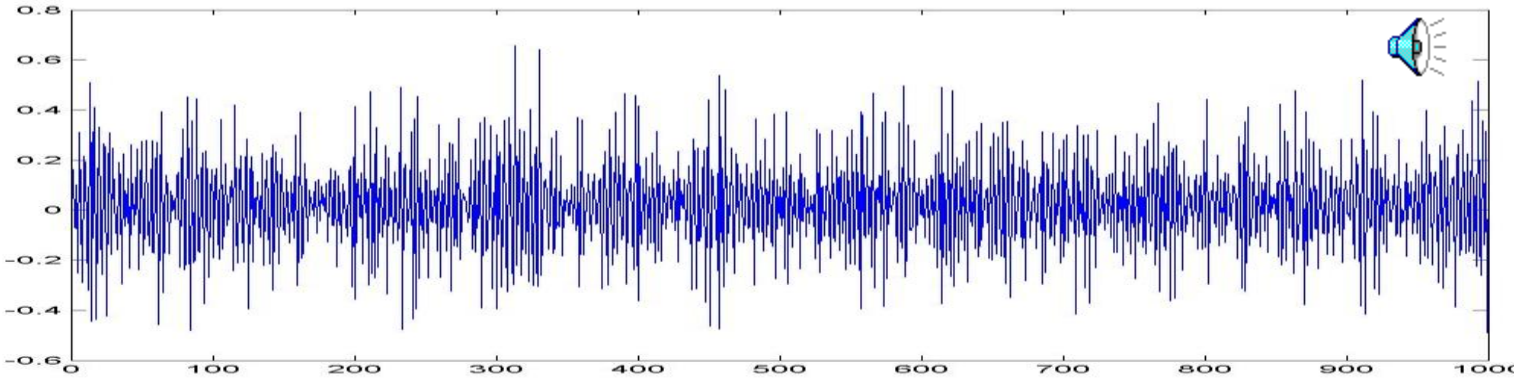
Advance to the next 32 ms segment beginning from the current pitch marker and repeat

Pitch Marker Caveats

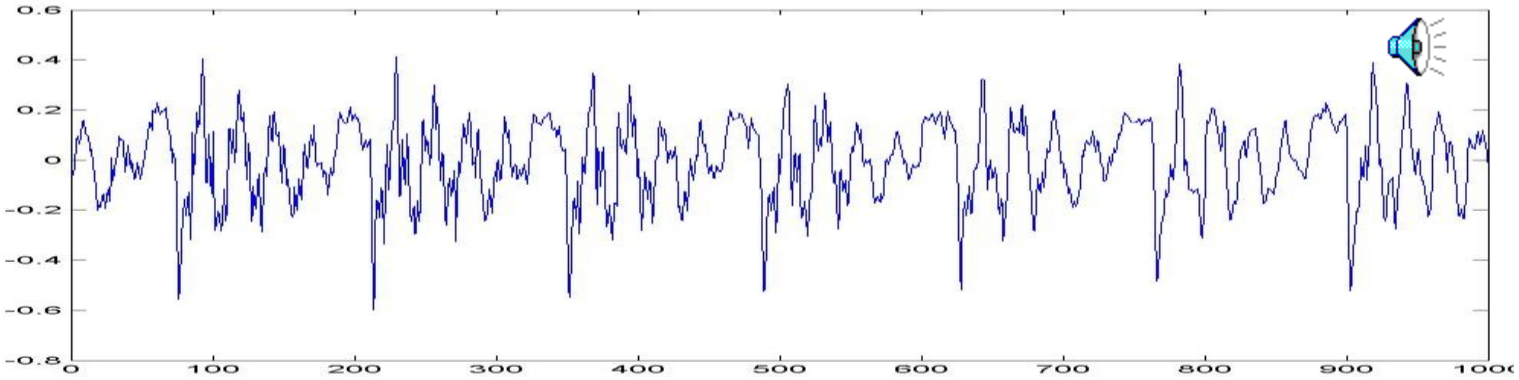
- Speech (in particular) comprises two kinds of sounds: voiced and unvoiced
- Voiced sounds exhibit repetitive patterns in the wave form, while unvoiced segments do not
- Unvoiced segments can be identified by the simple rule: autocor[1] / autocor[0] < threshold
  - Typically 0.5
- Pitch estimates will be meaningless in unvoiced segments
  - One usually simply identifies locations of zero crossings and uniformly spaced pitch segments in unvoiced regions
    - Unvoiced regions must be identified
- Pitch estimates will be poor in voiced, but consonantal regions
  - E.g. “v”, “b”, “z”
  - Better pitch tracking methods needed in these regions
    - Usually done by “sub-band” based estimation and voting
- Pitch estimation is difficult for noisy signals
  - Same reason as for voiced consonants

Voiced vs. Unvoiced Segments

- A typical unvoiced segment (r[1]/r[0] = 0.50)



- A typical unvoiced segment (r[1]/r[0] = 0.84)
  - Observe voicing

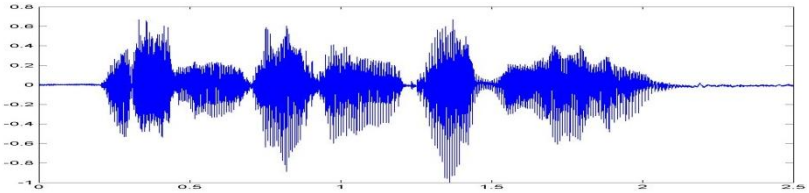


Psola Overall

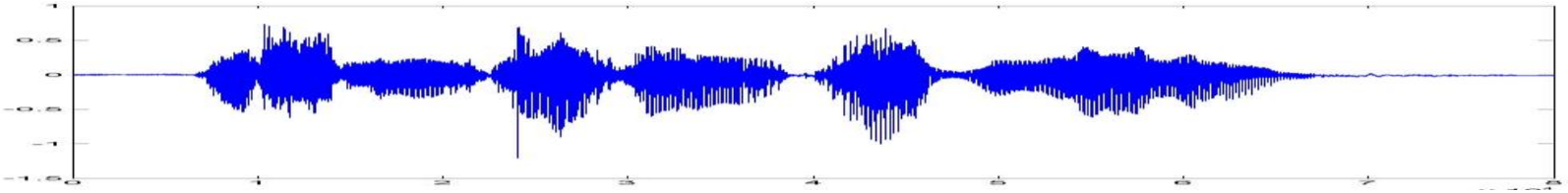
- First find the location of the beginning of all pitch period
- For stretching the signal, window every pair of pitch periods, space them out, insert new pitch periods (by replication) to get a signal of the desired length
- For shortening, identify the pitch period to retain, window them (along with the preceding pitch period) and glue them together by overlap-add

Psola Examples

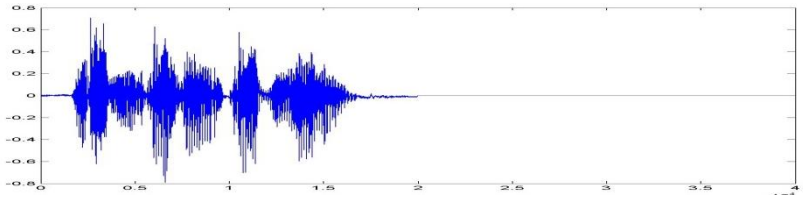
- Tom again



- Tom speaking slowly

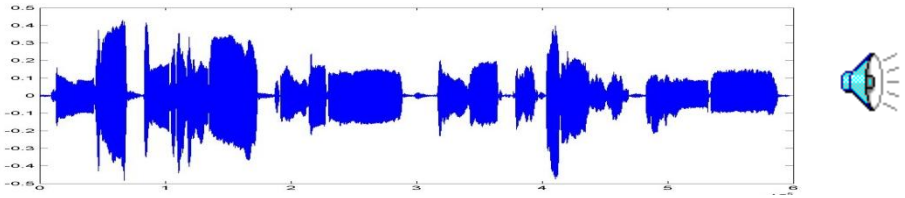


- Tom in a hurry

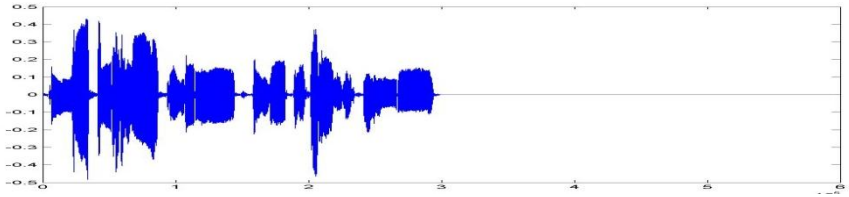


Psola Examples

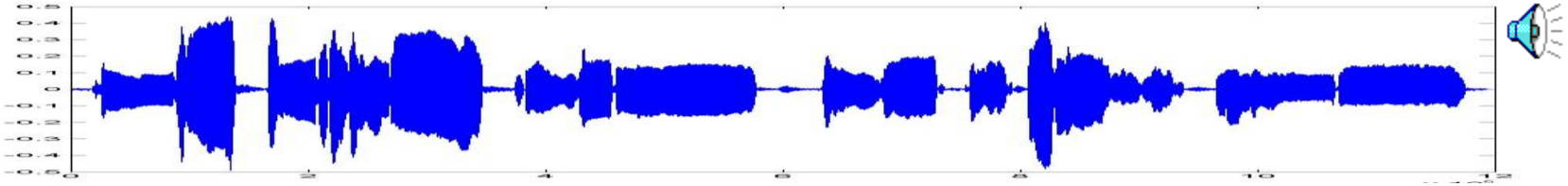
- Over the rainbow



- Faster

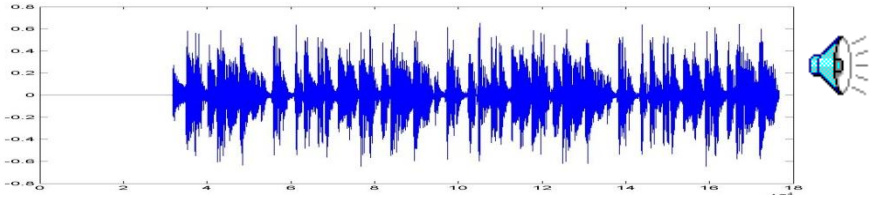


- Slower

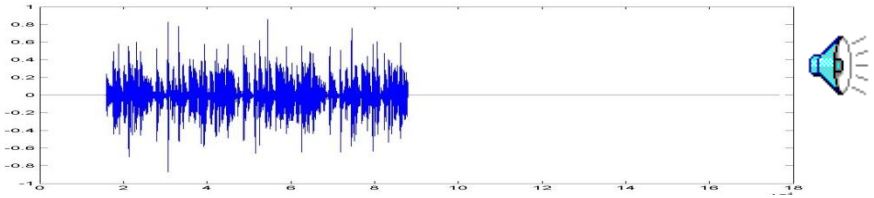


Psola Examples

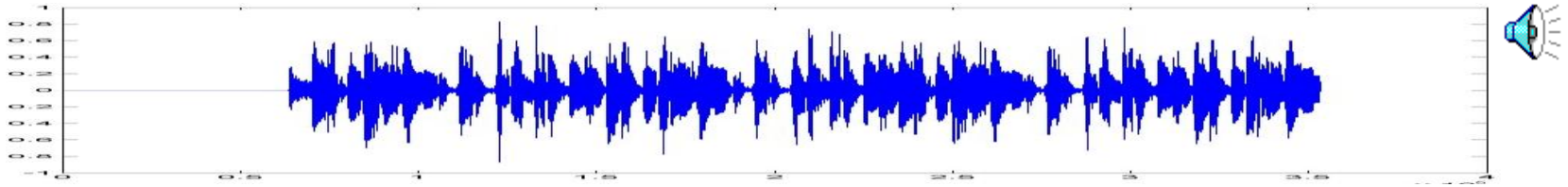
- A music segment



- Slower



- Faster



Pitch Shifting

- Time Scaling is the procedure by which we make a segment of audio longer or shorter
  - Without modifying the pitch (the sound doesn't sound squeaky or bass)
- Pitch shifting is the inverse process: Scaling the pitch of the signal without modifying the length
  - Remember that simply dropping samples (or interpolating new samples) increases (decreases) the pitch, but also modifies the length of the recording
  - We want the same length.

Pitch Shifting With Psola

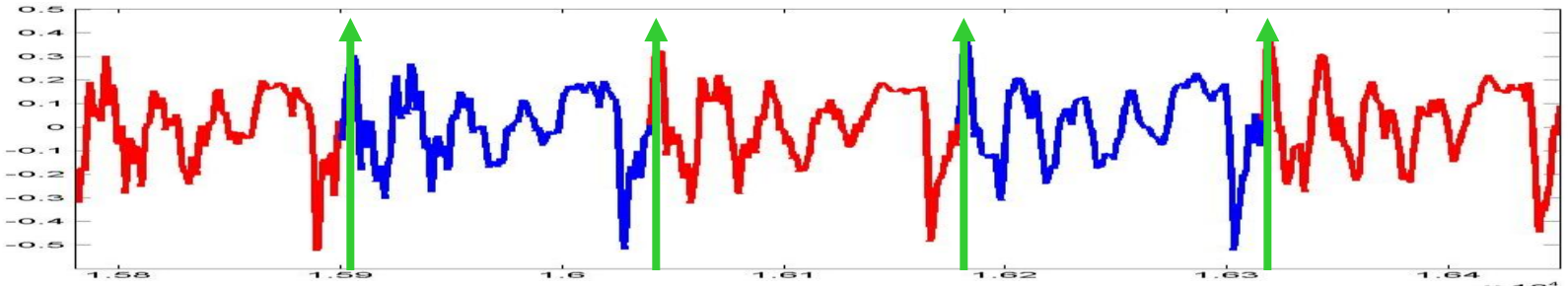
- We have seen how to speed up or slow down a signal using Psola
- Psola's most popular use, however, is for a different problem: Pitch Shifting
  - How to make Tom sound like Mary
  - How to have the same utterance, occurring in the same amount of time, with the same overall spectral characteristics, but with the pitch for the individual shifted up (more feminine) or down (more masculine)
- This is achieved very simply by sliding pitch periods with respect to each other to reduce or increase the distance between pitch markers
  - A pitch marker is the initial sample of a pitch period
  - Usually selected at a peak

PSOLA for pitch shifting

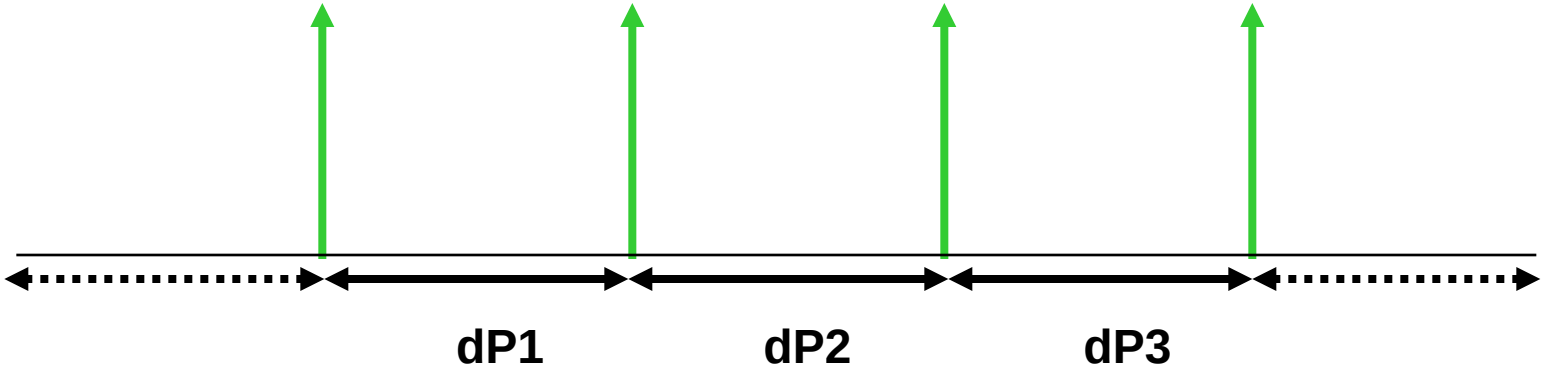
- The incoming speech has a number of pitch periods
- These pitch periods occur once every K samples, where K is the pitch period
  - Corresponding to a pitch of $f = (FS / K)$ where FS is the sampling frequency
- To modify the pitch to a new frequency f', we find the corresponding pitch period K' such that $f' = (FS / K')$
  - $K' = FS / f'$
- To get a signal with the modified pitch, we must get one pitch period every K' samples
- This is achieved by reducing the spacing between adjacent pitch periods to (K' / K) of their original value
  - So that the pitch periods in the new signal now occur at a spacing of K' samples
  - We use the notation $\beta = (K' / K)$

PSOLA for pitch shifting

- The original signal overlaid with pitch markers



- Showing only the pitch markers

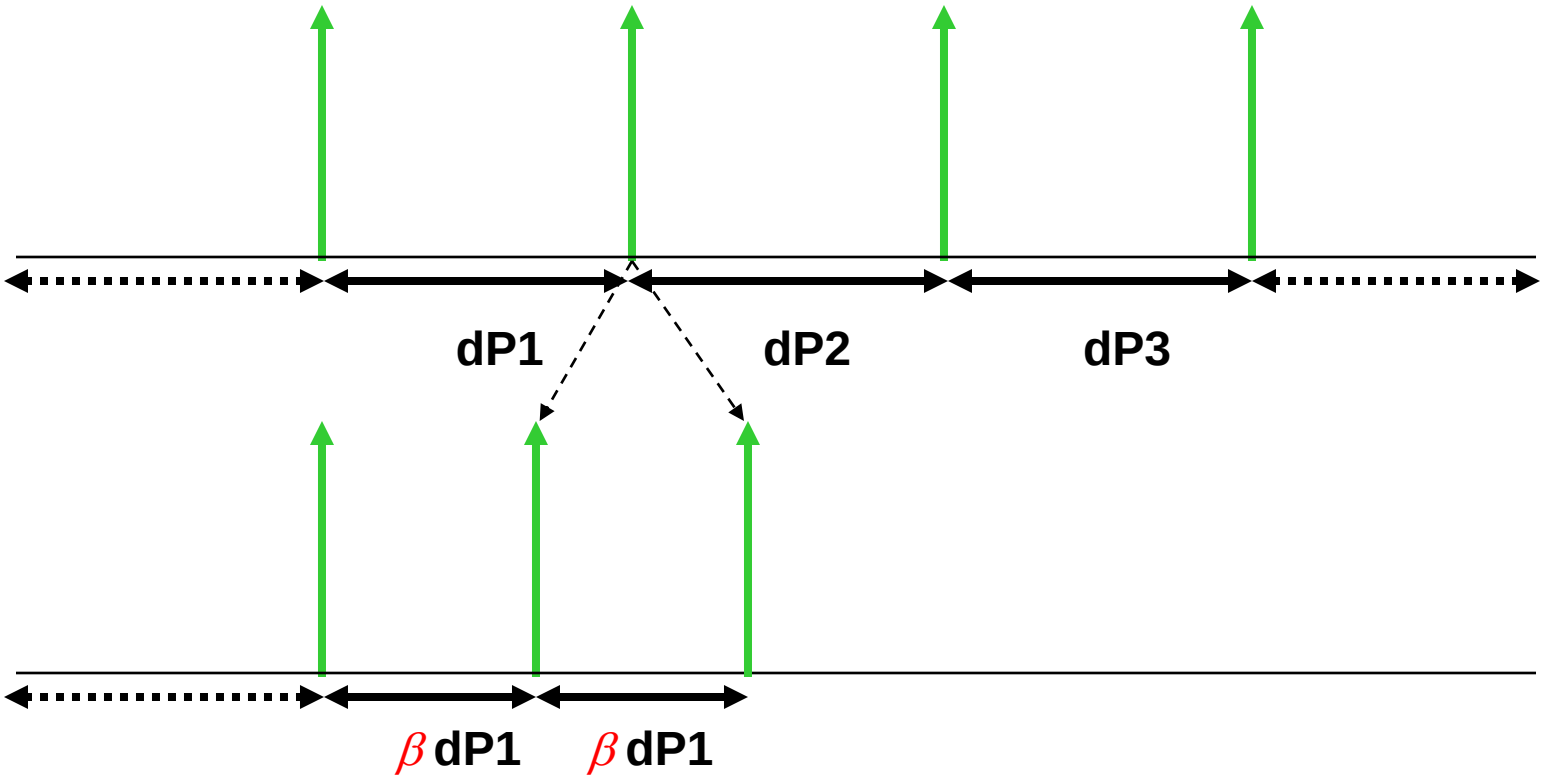


- The distance between adjacent pitch markers is noted

- To obtain a pitch scaling of β these inter-marker distances must be scaled by β

Finding new pitch markers

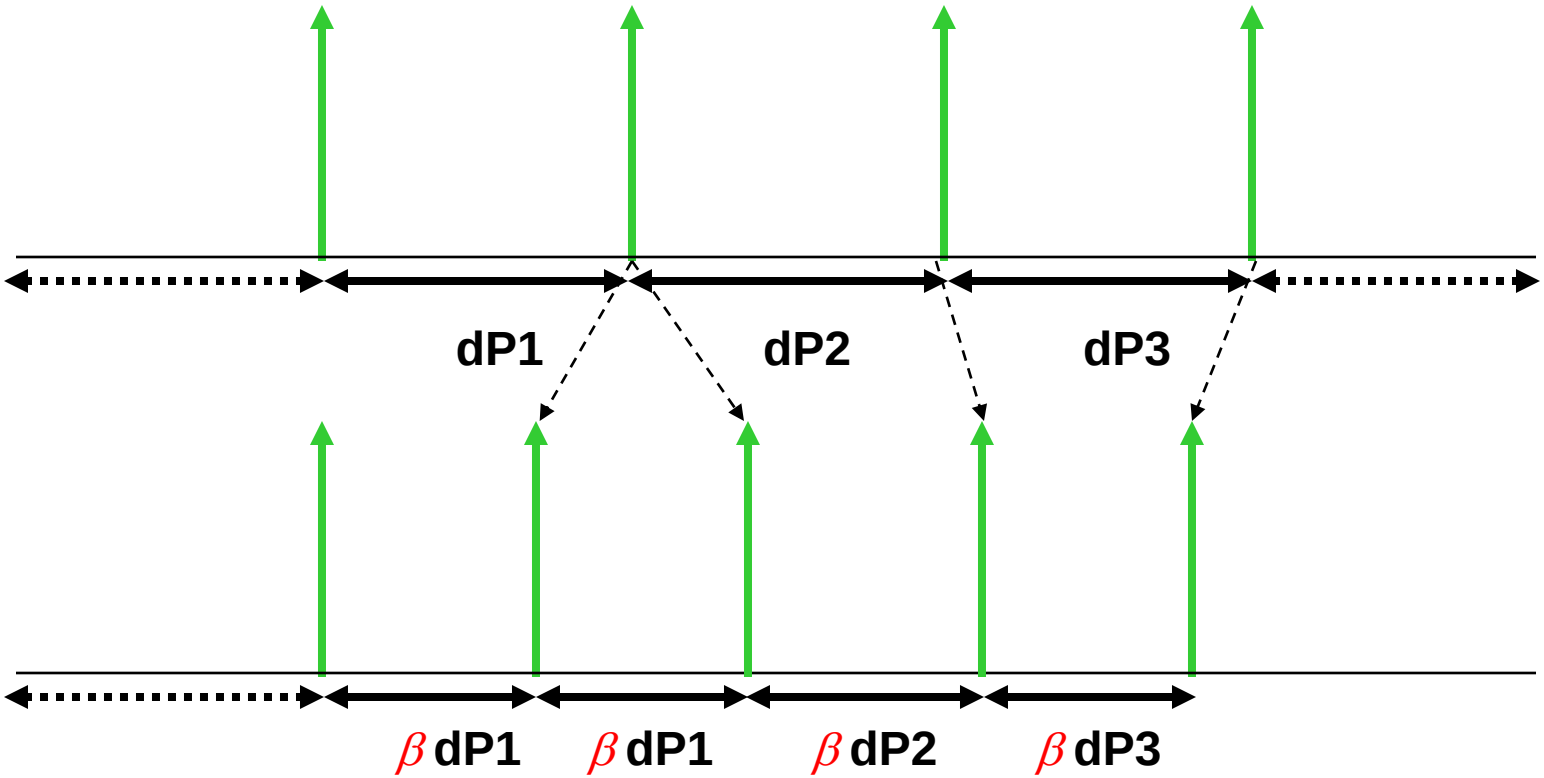
- The original pitch markers



- The next delta pitch is **dP1** . The scaled value is β **dP1**. The next pitch mark will occur after β **dP1** samples.
- The delta pitch corresponding to the next pitch mark (in the original signal) is also **dP1**. So the next scaled pitch mark also occurs after β **dP1** samples.

Finding new pitch markers

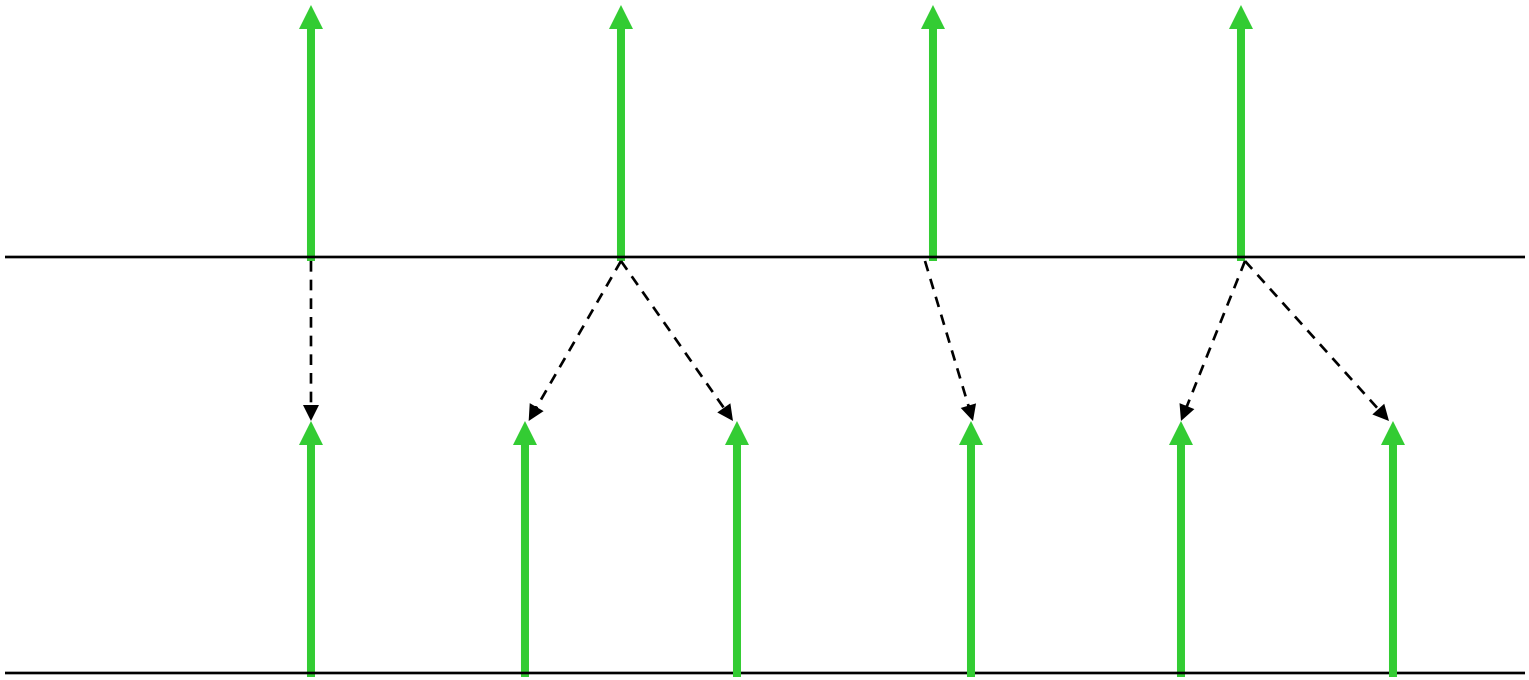
- The original pitch markers



- The next delta pitch is **dP2** . The scaled value is β **dP2**. The next pitch mark will occur after β **dP2** samples.
- The delta pitch corresponding to the next pitch mark (in the original signal) is **dP3**. So the next scaled pitch mark occurs after β **dP3** samples.

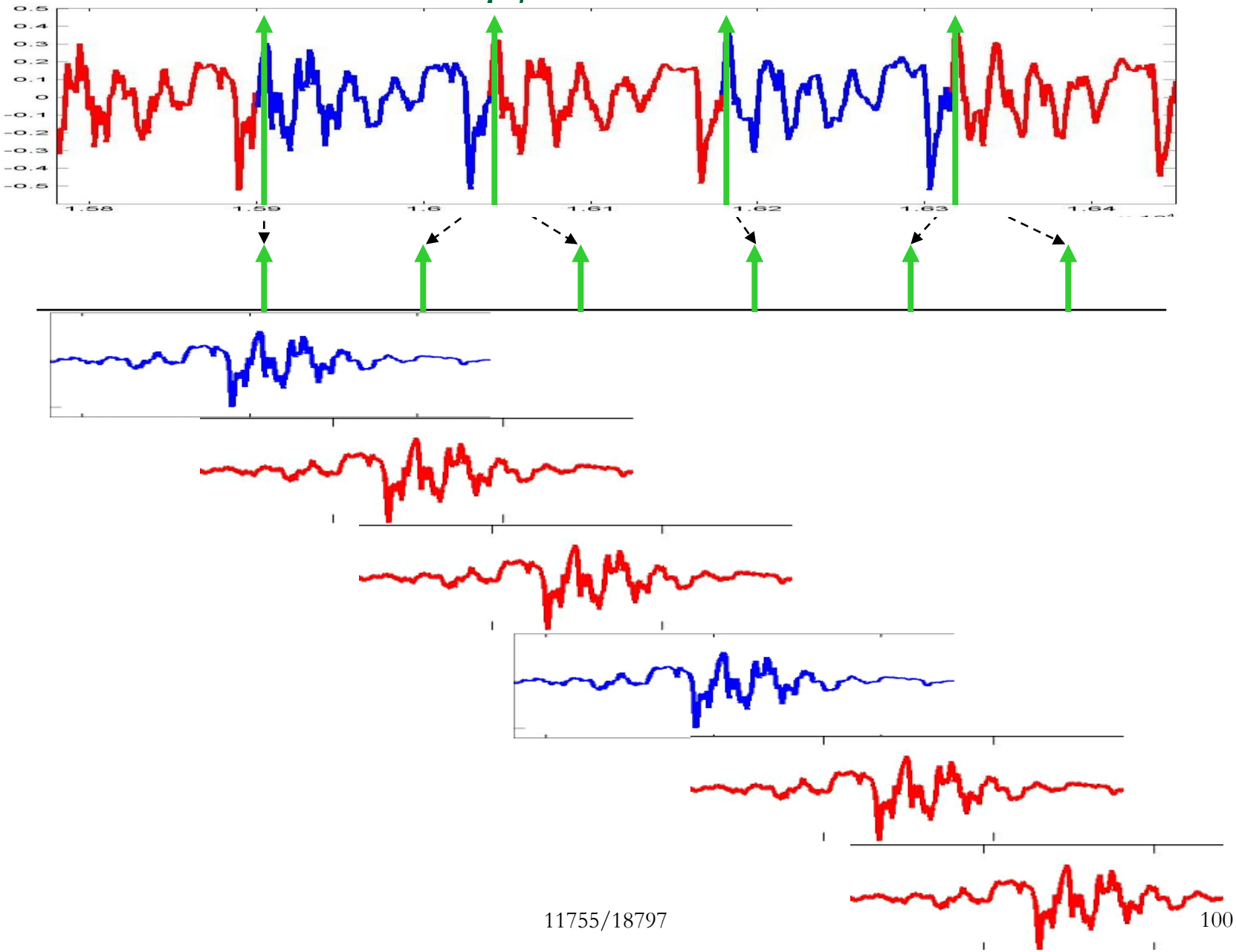
Finding new pitch markers

- The original pitch markers

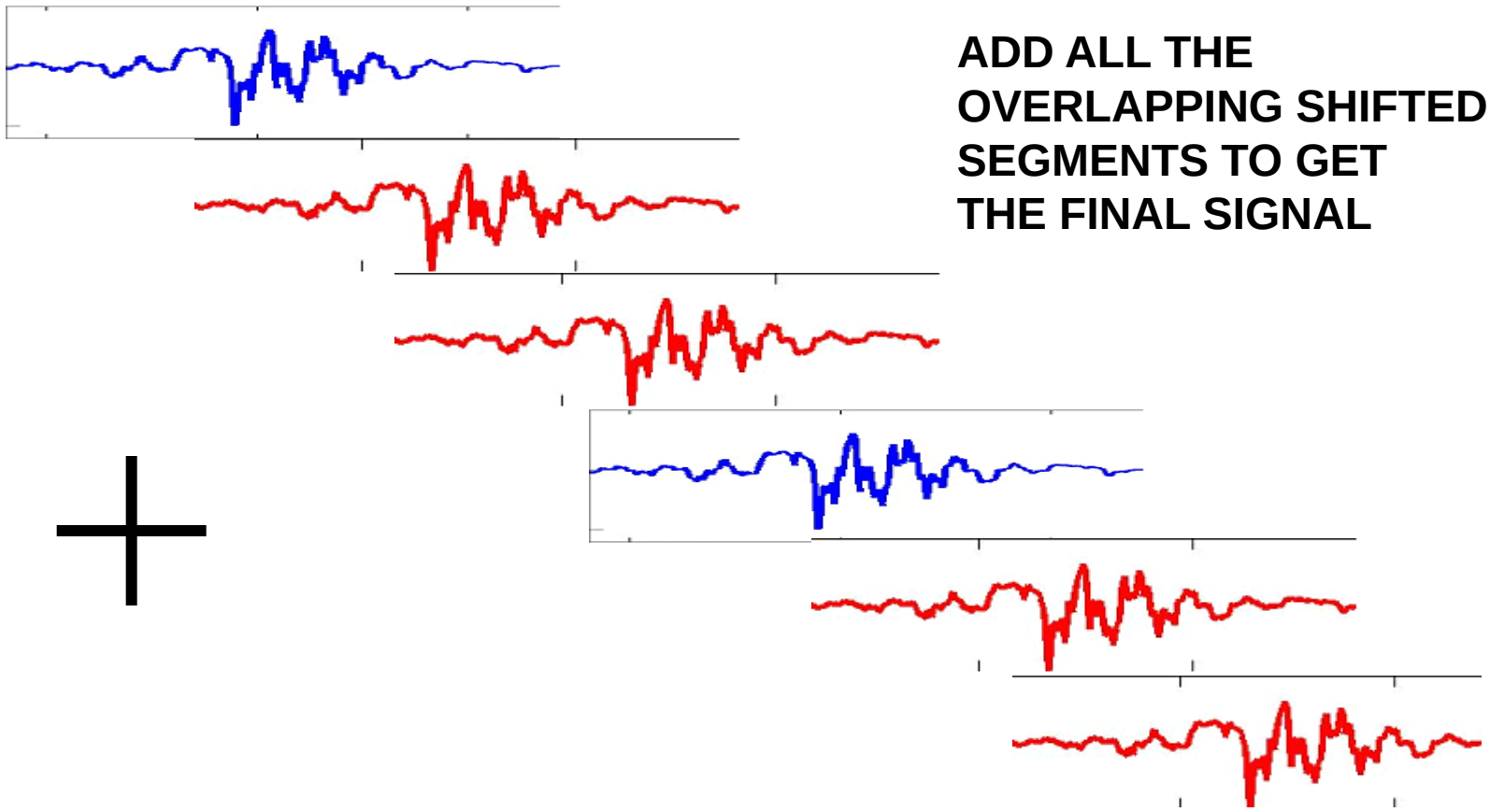


- The result is a series of pitch markers, each of which has a corresponding marker from the original signal
- A single marker in the original signal can correspond to multiple "scaled" markers
  - But a "scaled" marker can ONLY correspond to one of the markers for the original signal

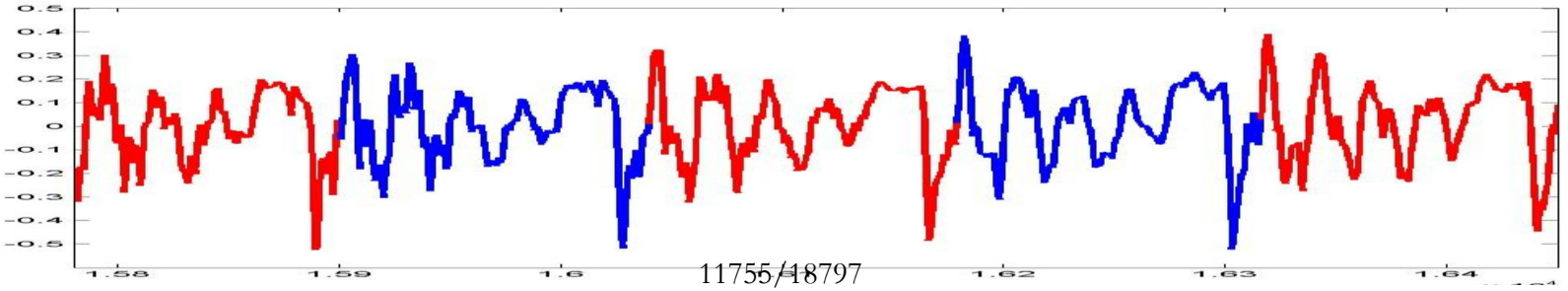
Pitch Shifting



Pitch Shifting



ADD ALL THE OVERLAPPING SHIFTED SEGMENTS TO GET THE FINAL SIGNAL

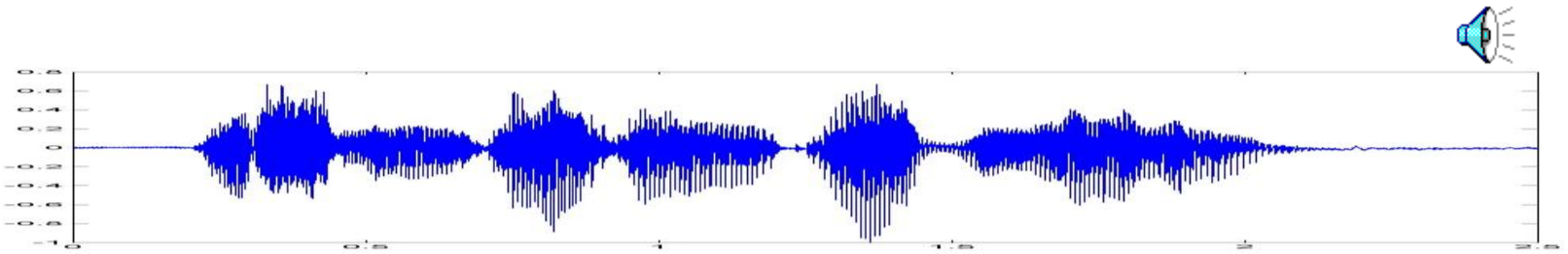


PSOLA pitch shifting Caveats

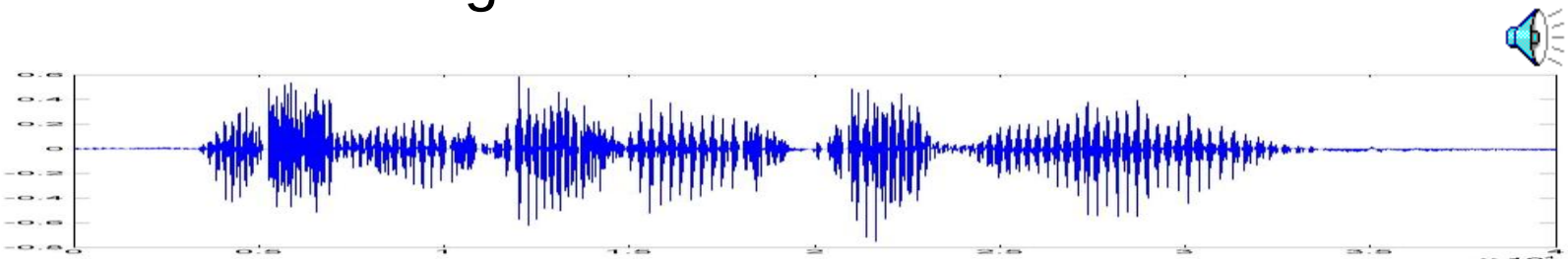
- Pitch shifting is preformed ONLY in voiced segments
  - Pitch shifting is irrelevant in unvoiced segments and may even cause distortion
  - No pitch shifting is performed in segments identified as unvoiced

PSOLA Examples

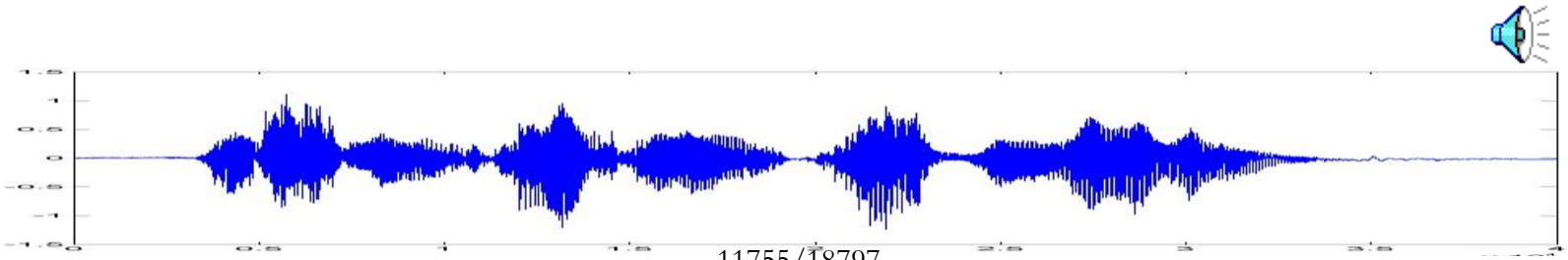
- Tom



- Tom Growling

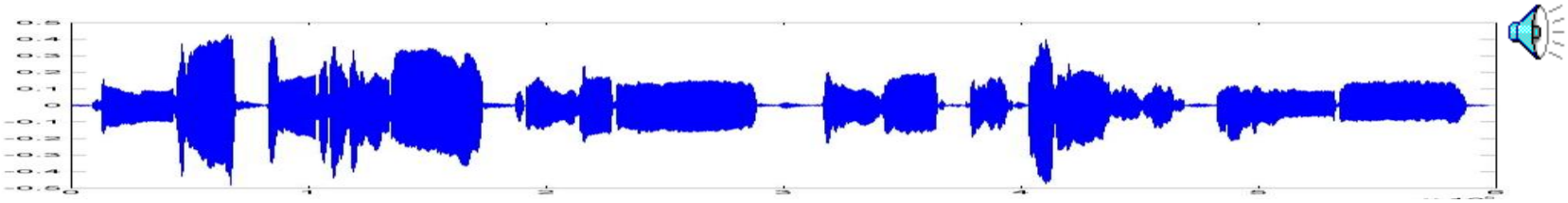


- Tom Squeaking

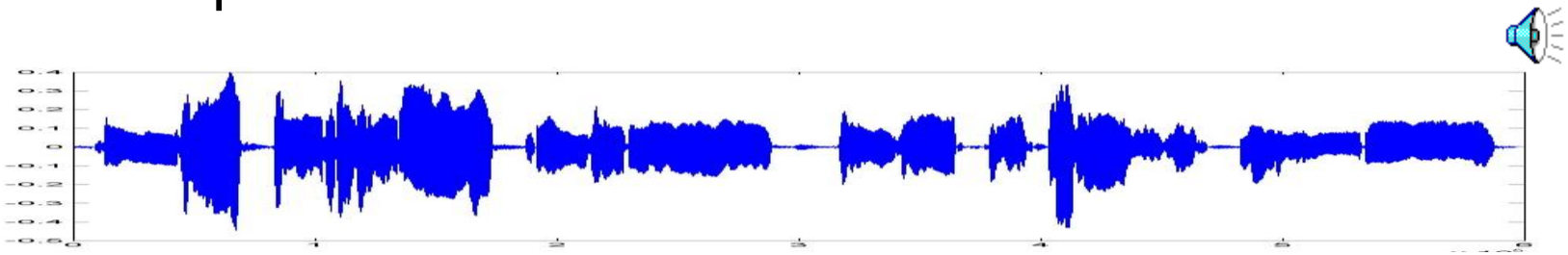


PSOLA Examples

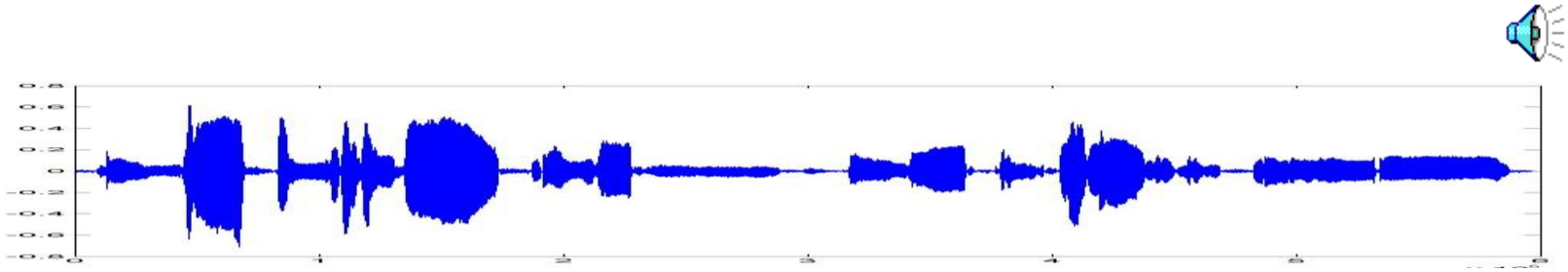
- Over the rainbow



- Low pitch



- Hi pitch



PSOLA for pitch shifting

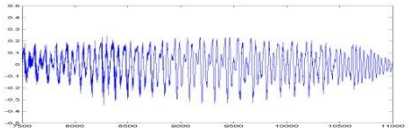
- The outcome of the overlap add is a pitch shifted signal
- The procedure can be used both for increasing and decreasing pitch
- It works best for signal where a unique pitch can be identified
  - Speech, singing voices
  - Not useful for polyphonic signals
- Not very effective for scaling factors outside the range 0.5-2.0.

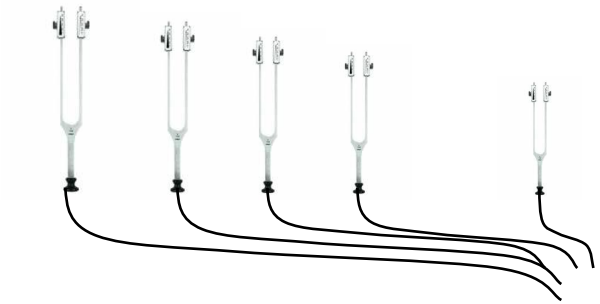
PHASE VOCODER

- A different method for time scaling is the phase vocoder
- This procedure does not depend on explicit detection of pitch periods
- Rather, it achieves time scaling by clever scaling of the envelopes of each of the sinusoids in a Fourier decomposition of the signal

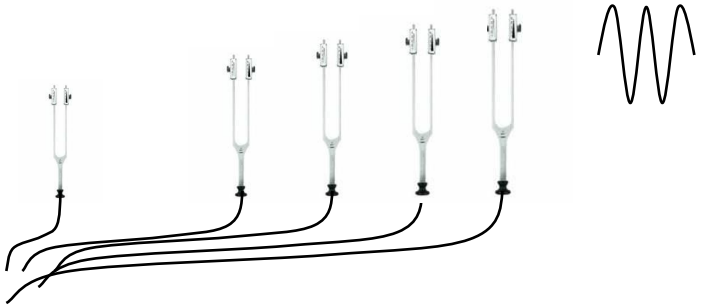
Time Scaling With Tuning Forks

- A set of tuned tuning forks will produce independent sinusoids when excited by sound

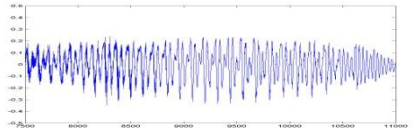




- If you summed the outputs of the tuning forks again, you'd get back the sound

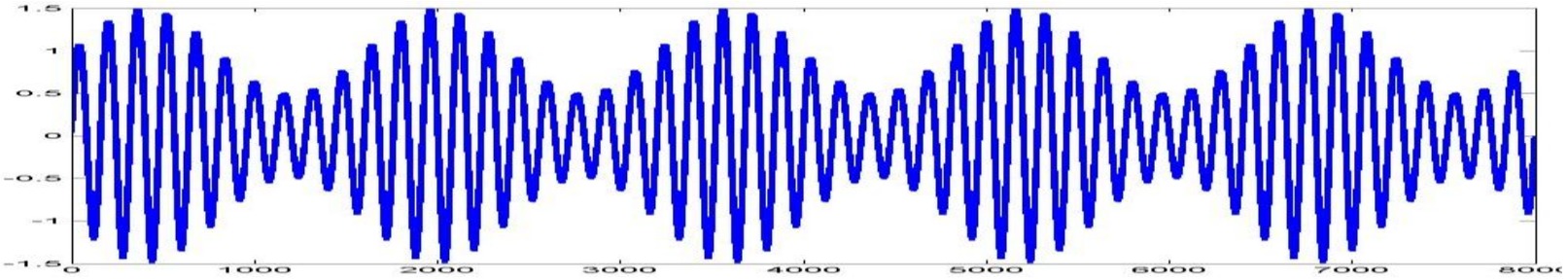


+

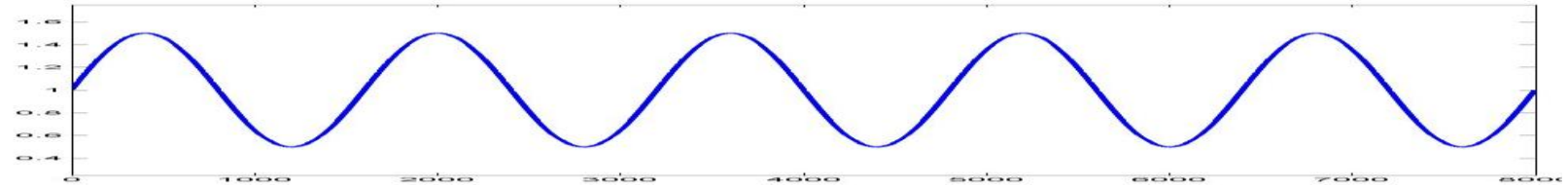


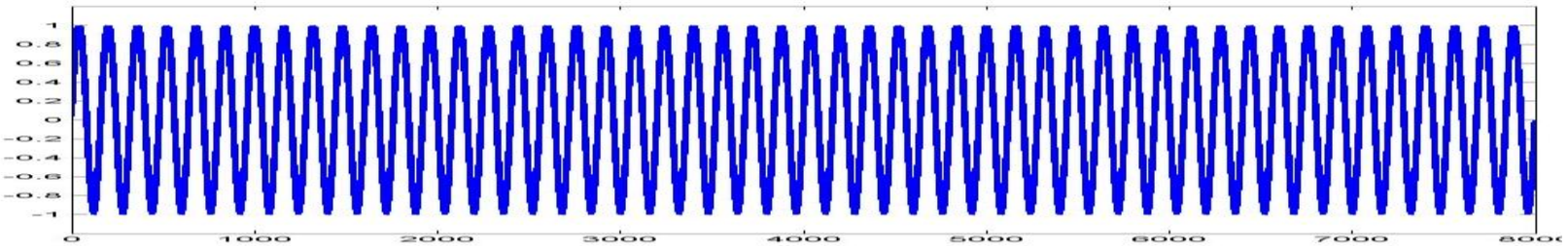
Time scaling with tuning forks

- If we scaled the output of each tuning fork independently and added the signals back, we'd get a time-scaled signal
- The output of each tuning fork is an amplitude modulated sinusoid



- This can be separated into an envelope and a uniform sinusoid

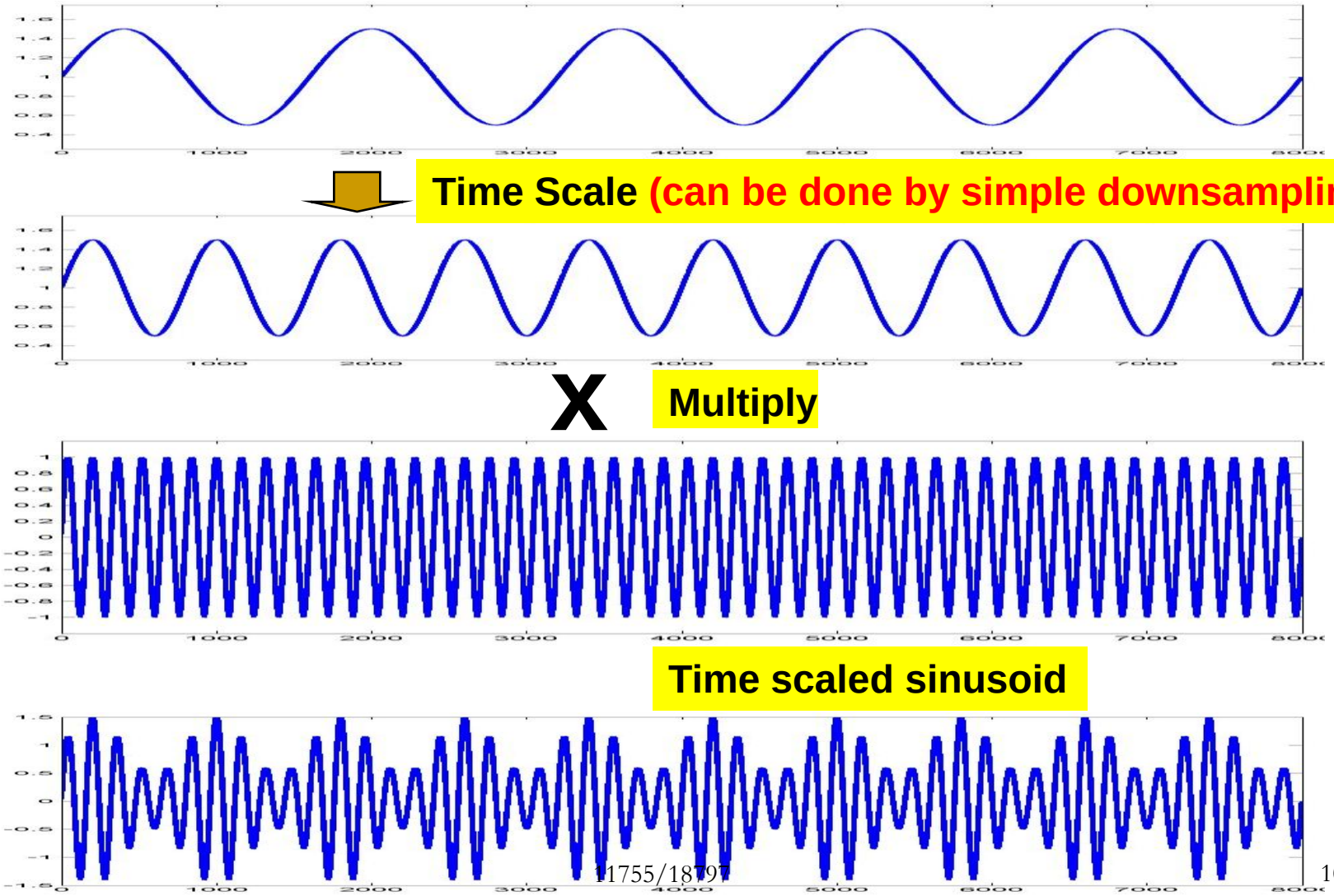




Multiplying these two gives us the original tuning fork output

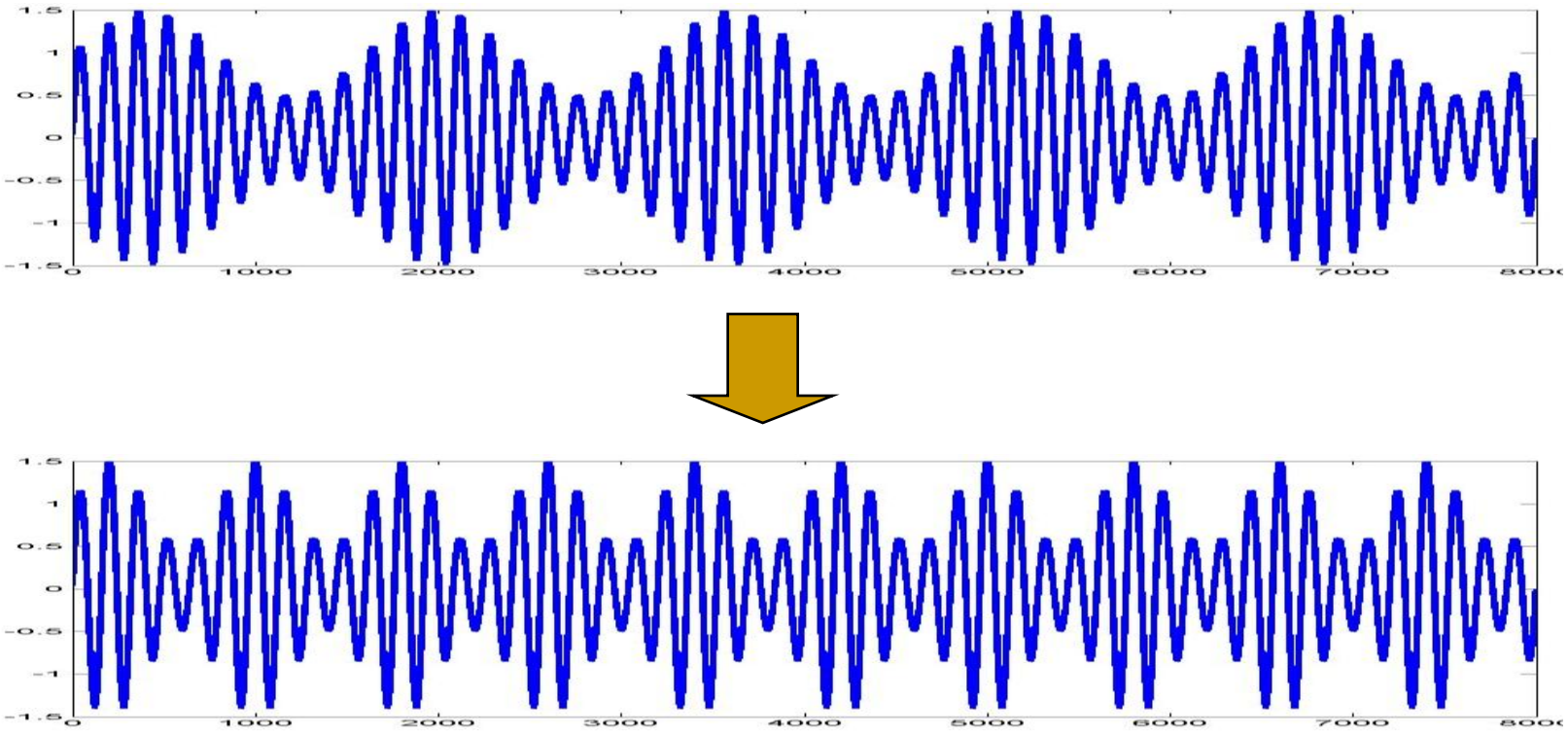
Time scaling with tuning forks

- Time scaling can be achieved by time scaling only the envelope and factoring it back into the sinusoid



Time scaling with tuning forks

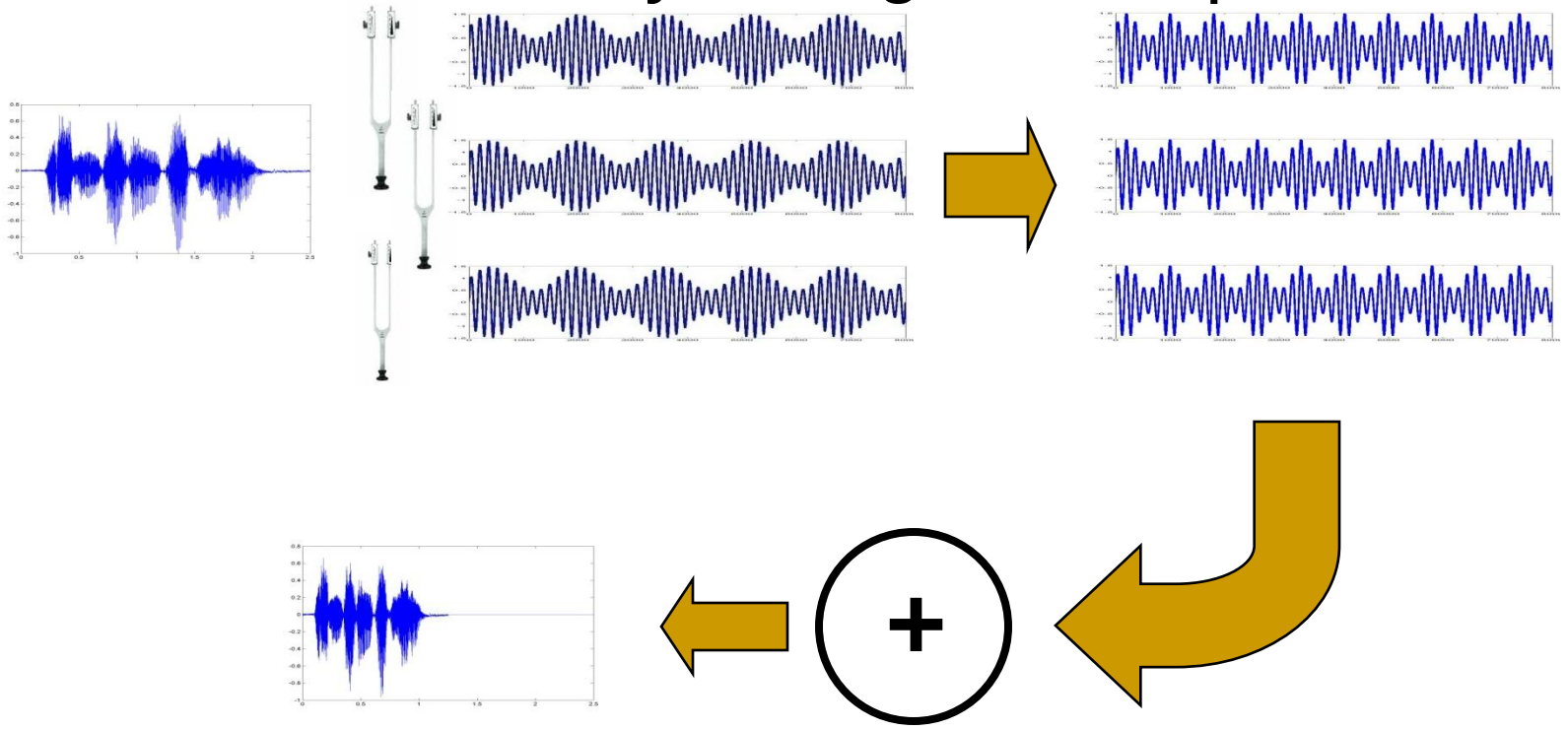
- Time scaling can be achieved by time scaling only the envelope and factoring it back into the sinusoid



- By shrinking the envelope and applying it to the sinusoid we get the same perceptual time pattern at twice the rate without changing the basic frequency

Time scaling with tuning forks

- Time scale every tuning fork output



- Add them back in: Voila, a properly time-scaled signal

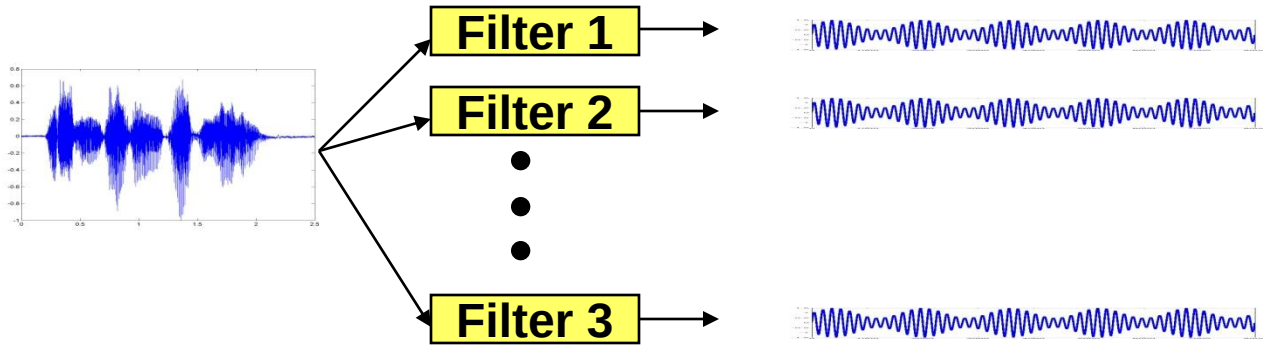
- **THIS IS THE PRINCIPLE WE WILL USE**

Phase Vocoder

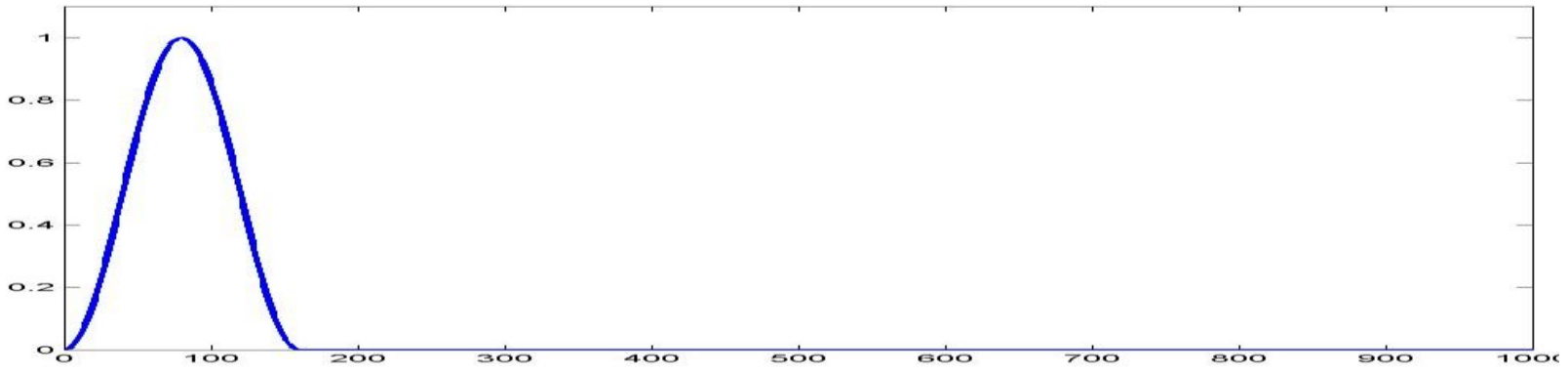
- Originally proposed by James Flanagan in 1966
  - For compressing speech signals!
- Mimics the tuning fork idea with filters
- Pass the input audio signal through a large bank of filters
  - Filters must be such that adding their outputs must return the original signal
- Time scale the output of each filter
  - Add back to reconstruct time scaled signal

Phase Vocoder

- Basic filter bank: the output of each filter is a narrow band of frequencies

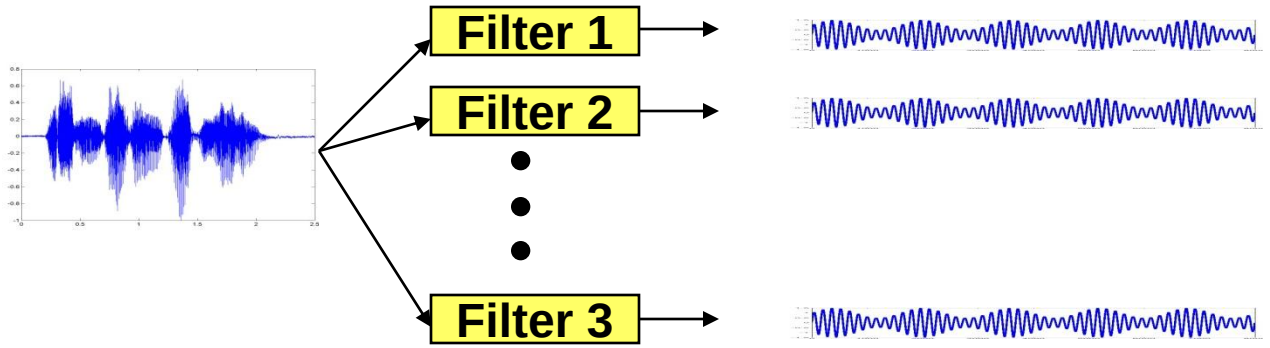


- The frequency responses of the filters must add to 1

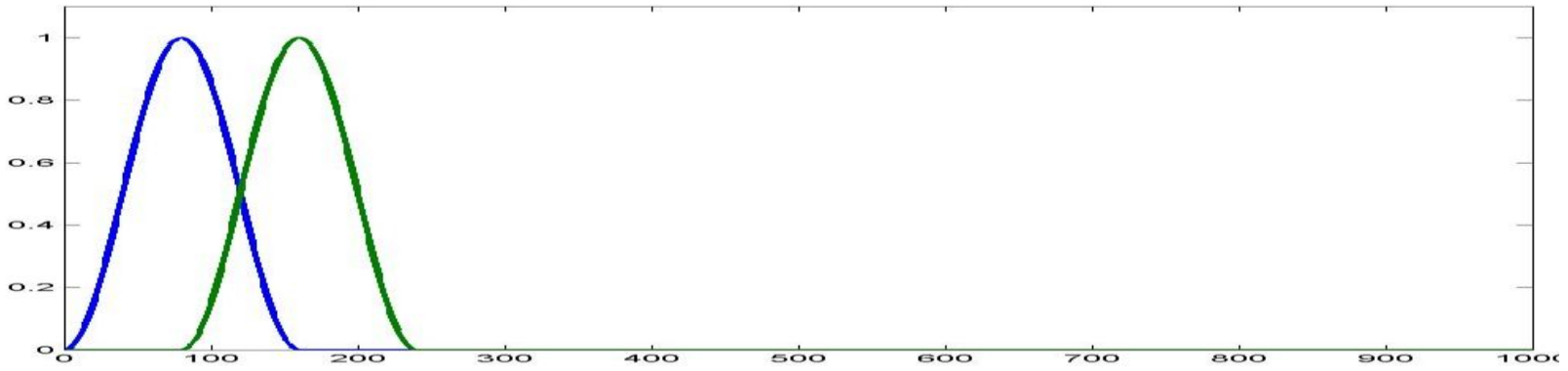


Phase Vocoder

- Basic filter bank: the output of each filter is a narrow band of frequencies

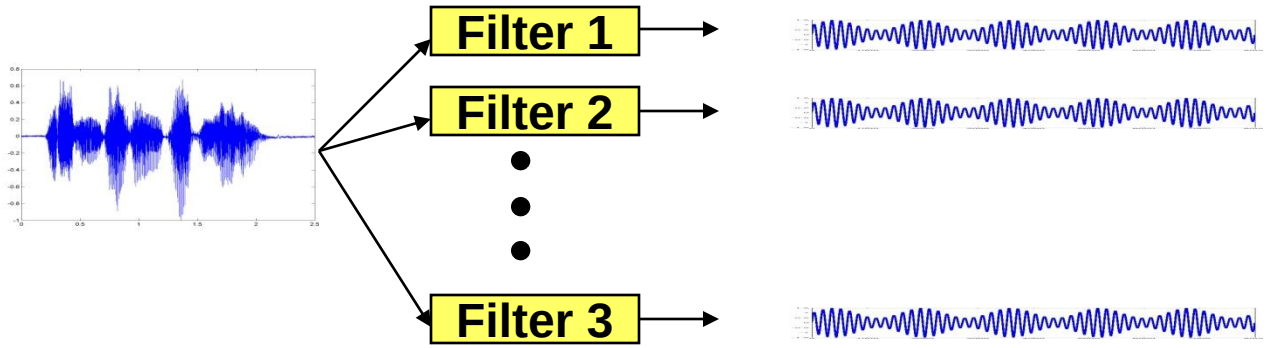


- The frequency responses of the filters must add to 1

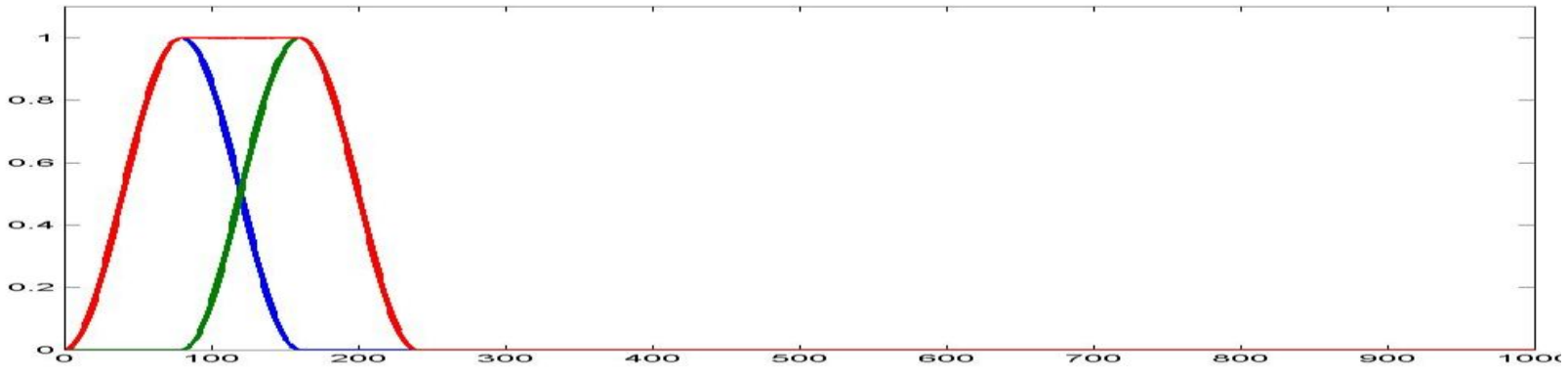


Phase Vocoder

- Basic filter bank: the output of each filter is a narrow band of frequencies

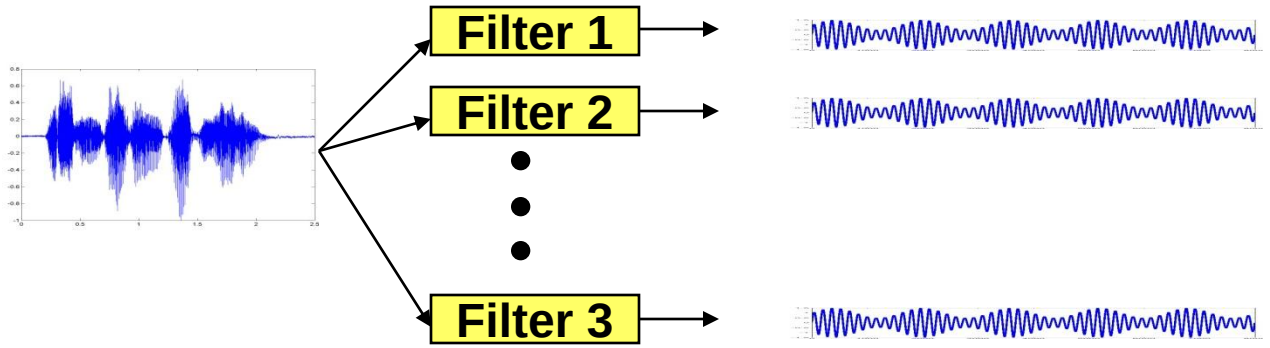


- The frequency responses of the filters must add to 1

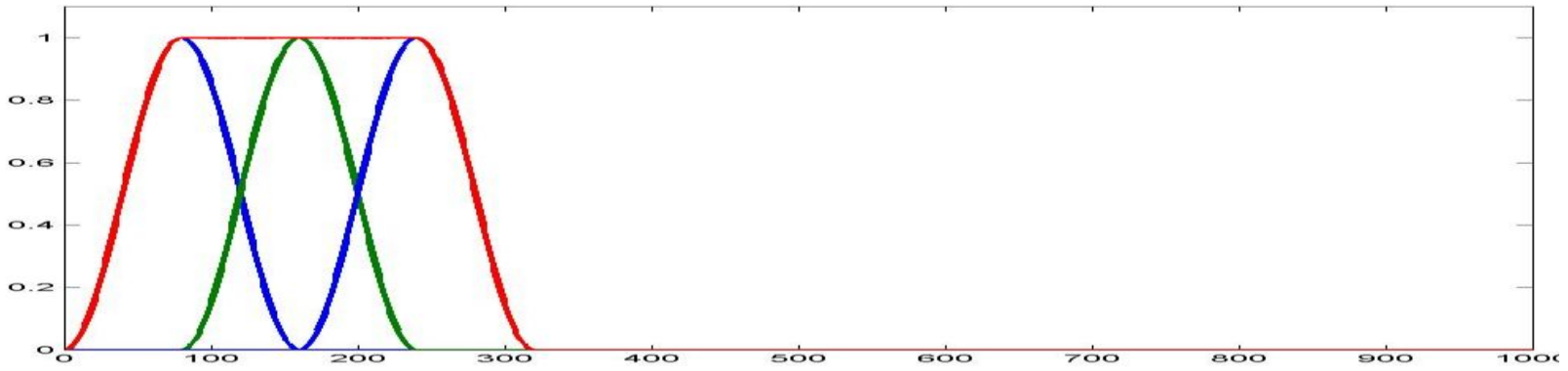


Phase Vocoder

- Basic filter bank: the output of each filter is a narrow band of frequencies

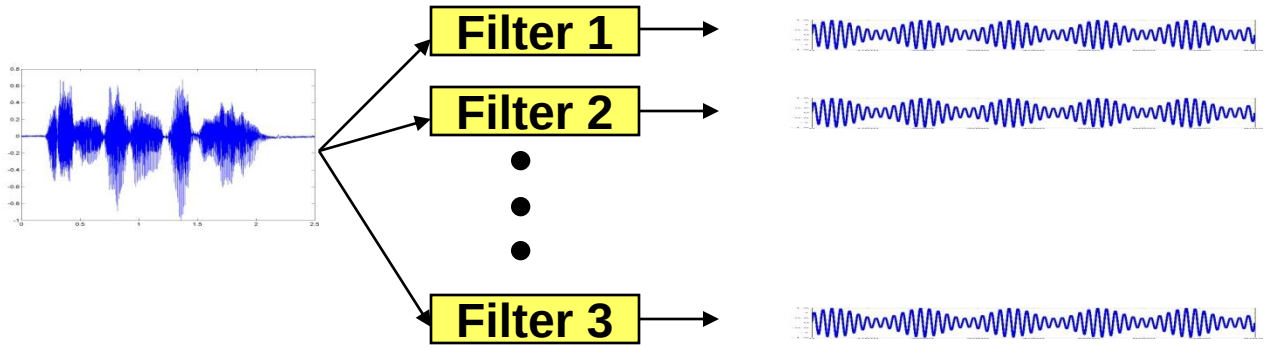


- The frequency responses of the filters must add to 1

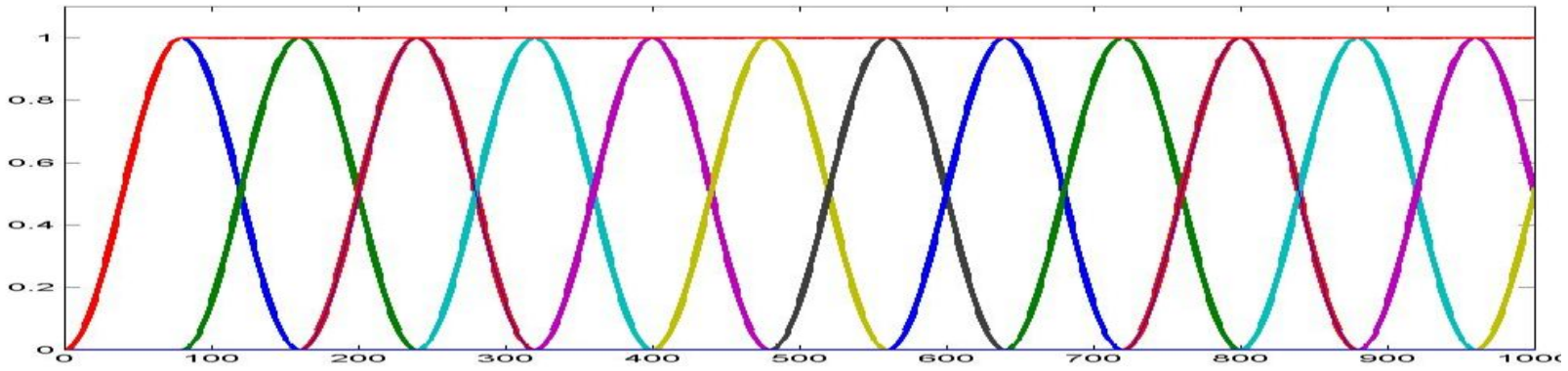


Phase Vocoder

- Basic filter bank: the output of each filter is a narrow band of frequencies

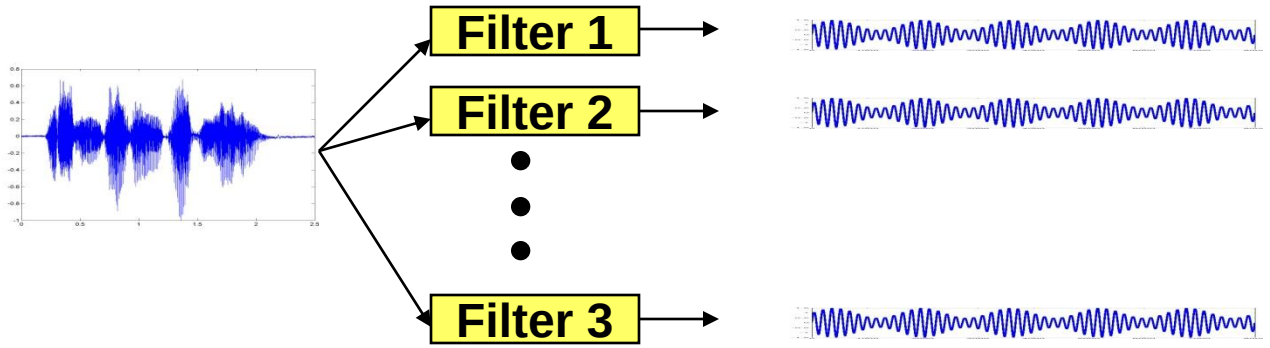


- The frequency responses of the filters must add to 1

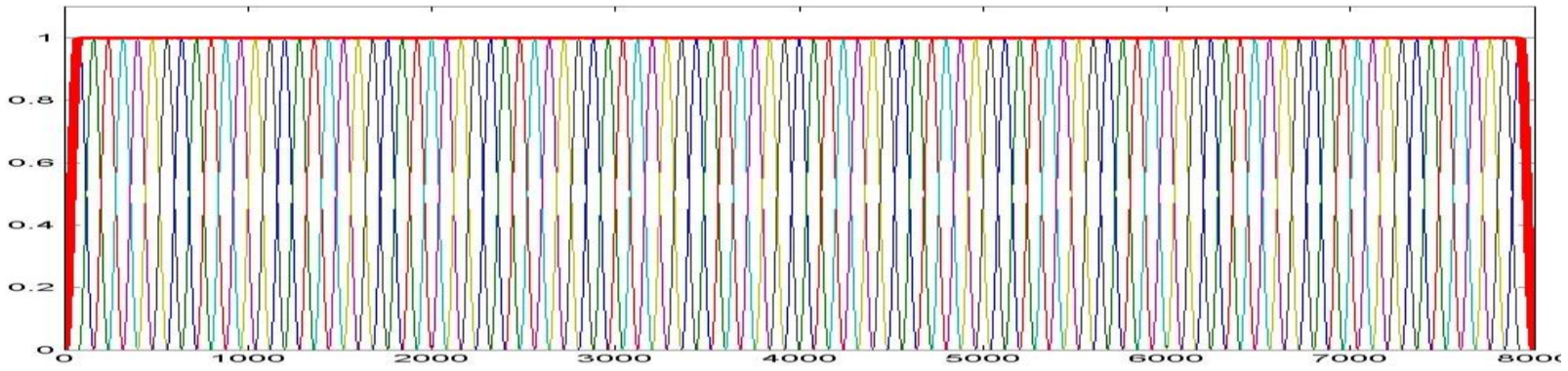


Phase Vocoder

- Basic filter bank: the output of each filter is a narrow band of frequencies



- The frequency responses of the filters must add to 1



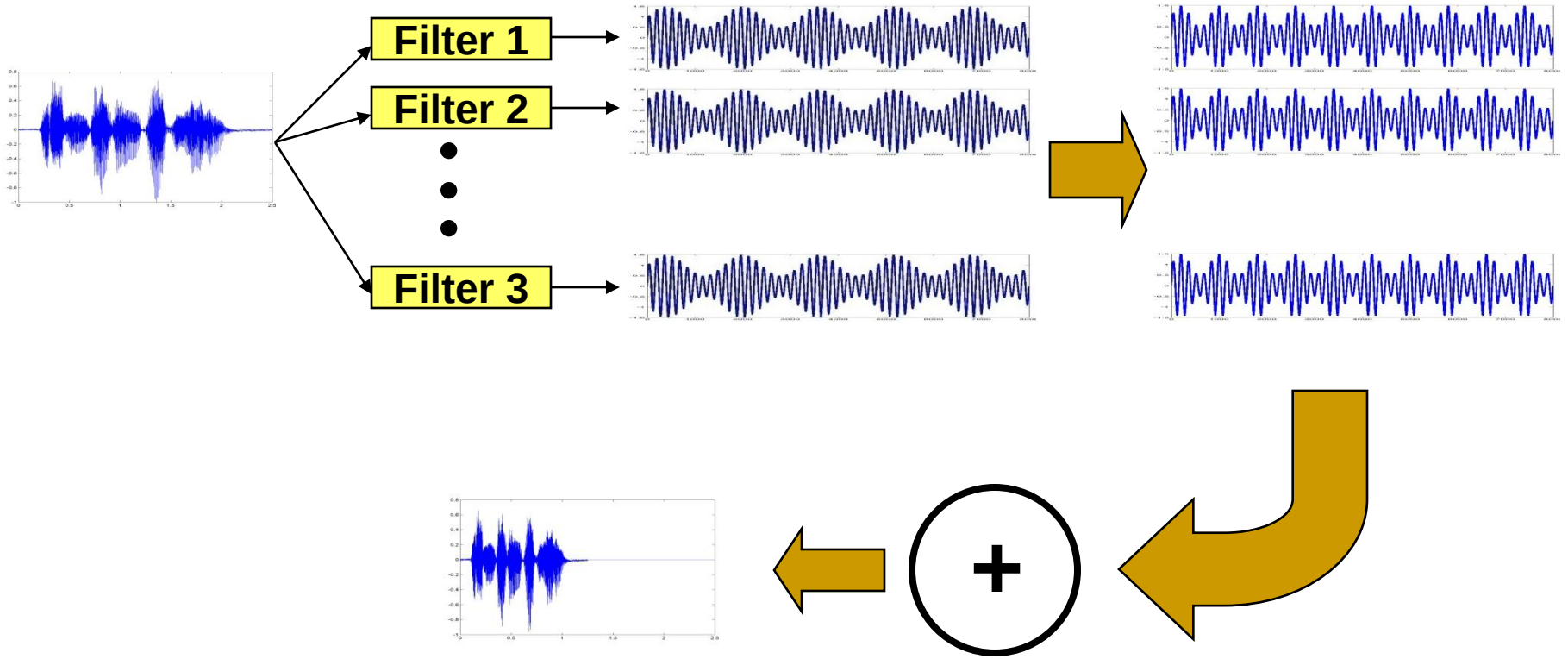
- The total number of filters must be large: No filter must cover more than one harmonic peak in the signal

Phase Vocoder

- Filter-bank constraints:
  - Filter center frequencies must be uniformly spaced.
  - The frequency response of the filters (in the frequency domain) must sum to 1.0
  - If the signal is sonorant (music, vowels), it contains one or more pitch frequencies and their harmonics
  - No filter must allow more than one of these harmonics through
    - i.e. there must be at least as many filters at there are harmonics in the spectrum
    - If your sampling frequency is FS and the lowest pitch frequency you expect is P, there must be more than FS/P filters

Phase Vocoder

- A Schematic: the output of each filter is time-scaled and added back



- The output is a time-scaled signal

Phase Vocoder: Each Filter

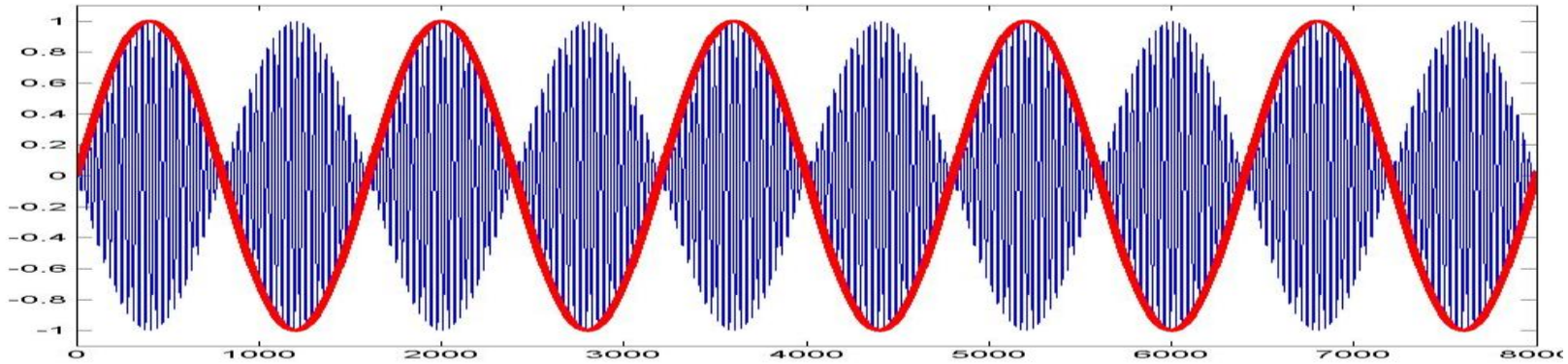
- The output of a single filter is essentially an amplitude modulated sinusoid
  - This is the reason we want our filters to be narrow: otherwise the output will not be even approximately sinusoidal

- The amplitude and the sinusoid can be separated by heterodyning
  - The same principle used in your AM receiver
  - Multiply the output of the filter by a sinewave of the center frequency of the filter
    - This generates a mixture of two signals, one that represents the envelope of the sinewave, and the other that represents the amplitude
  - Extract the amplitude by applying a low pass filter

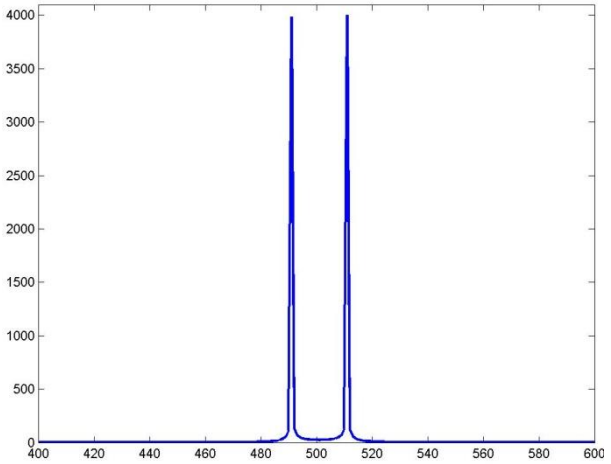
Phase Vocoder

- Basic equation:

$$\sin(a)\cos(b) = \sin(a-b)/2 + \sin(a+b)/2$$



- By the above equation, this is the same as the sum of two sinusoids distributed around the center frequency of the filter

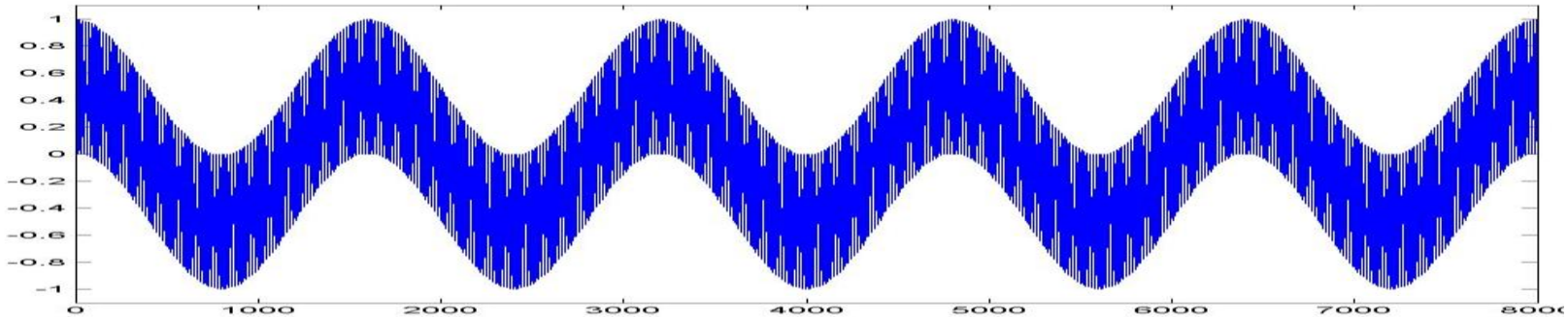


The spectrum of the above signal

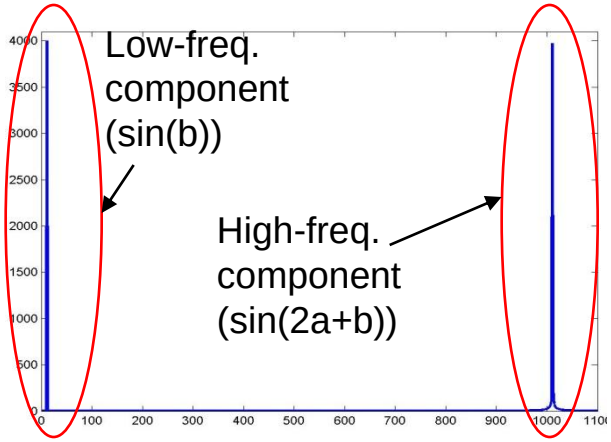
The Heterodyne Principle

- By multiplying the output of the filter by a sinusoid at the center frequency of the filter, the envelope can be separated into an additive low frequency component:

$$\sin(a+b)\cos(a) = \sin(2a+b)/2 + \sin(b)/2$$



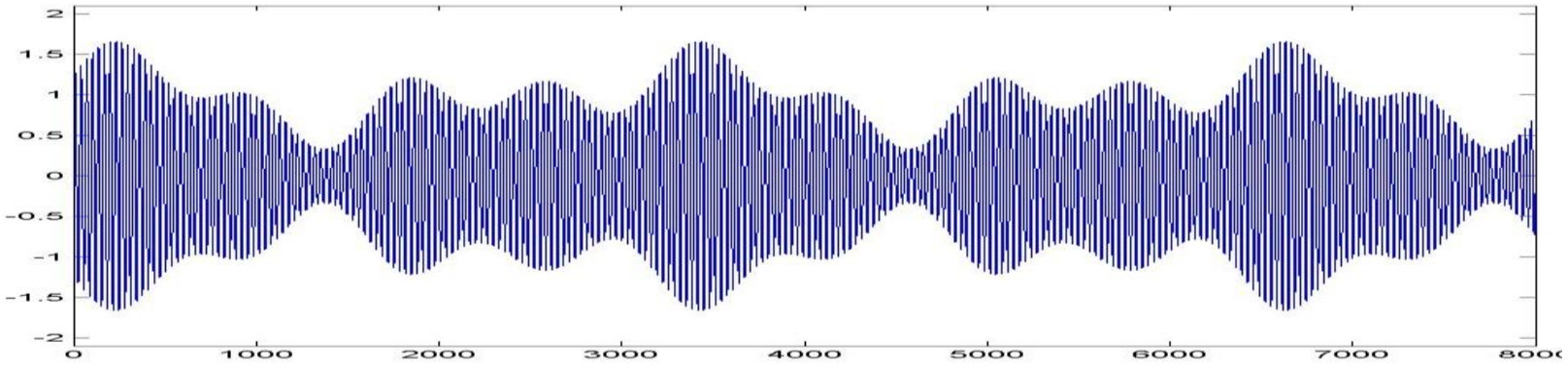
- The product of the amplitude modulated sine (sin(a+b)) with a sine wave at the base frequency (cos(a)) is a high-frequency sine wave whose bias is a low-frequency sinusoid



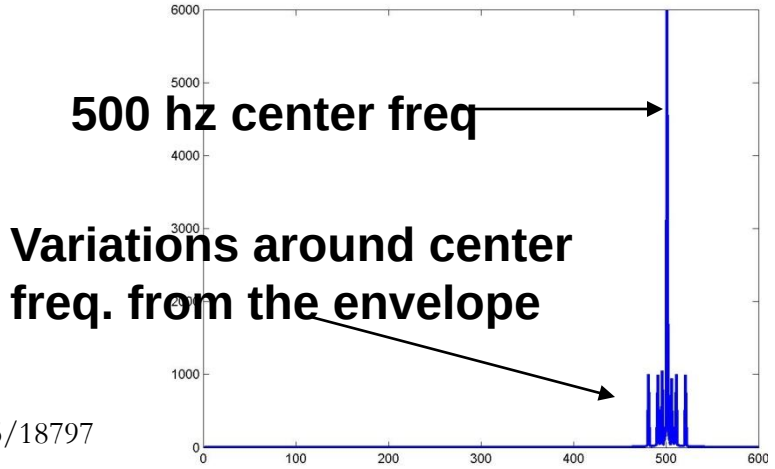
- The low-frequency sinusoid can be filtered out by a low pass filter

The Heterodyne Principle

- The envelope of the output of any **filter** is a low-frequency signal (< 50Hz). The low frequency envelope can be extracted by similar heterodyning (**multiply by center frequency sine, low pass filter**)



- The spectrum of the above signal, which is the output of a filter with a centre frequency of 500Hz (the high-freq component has a frequency of 500 Hz)



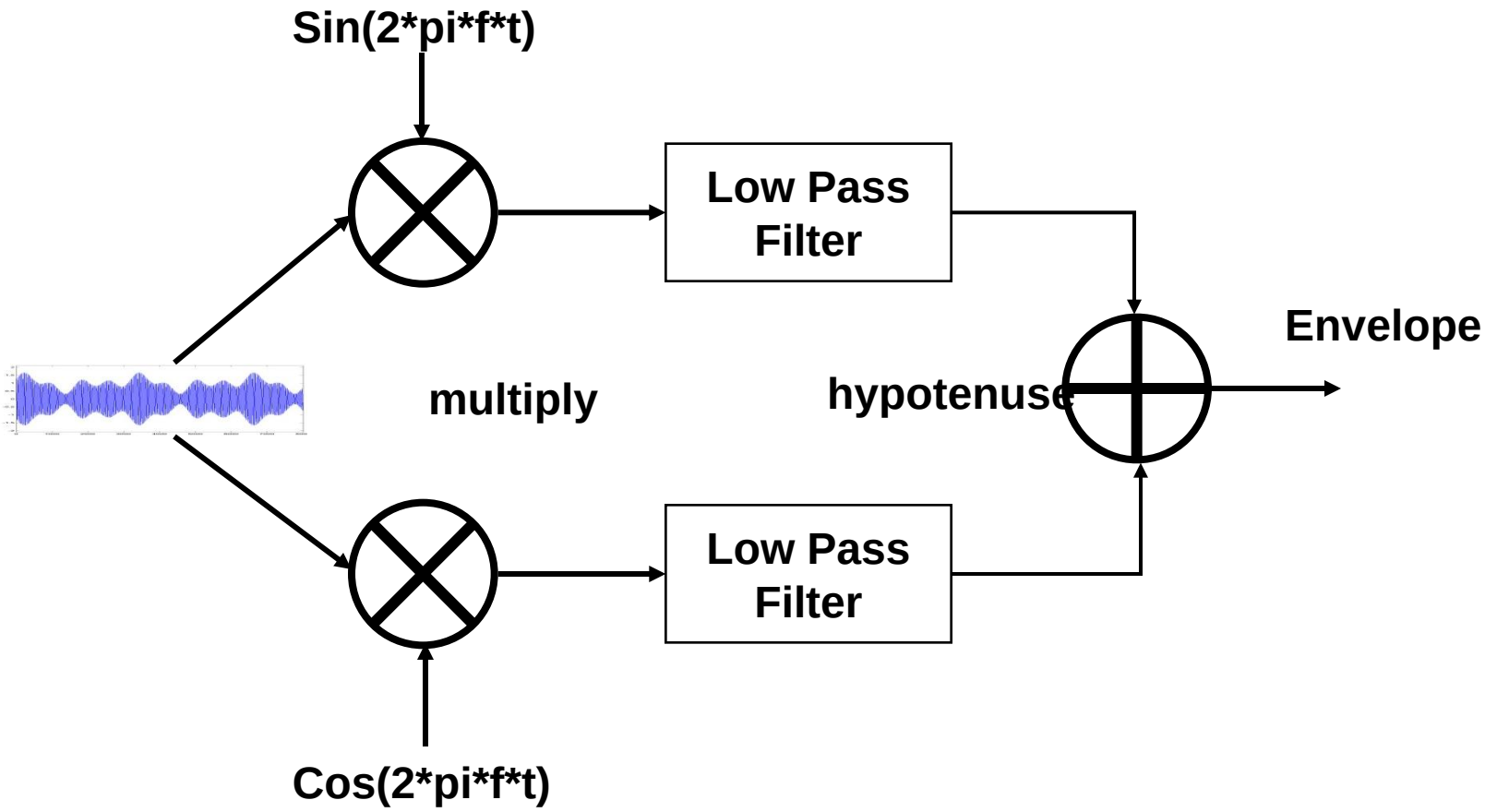
Heterodyning Caveat

- Mutiplying by a cos(a) wont always work
- Original:
  sin(a)cos(b) = 0.5sin(a+b) + 0.5sin(a-b)
- Heterodyning: multiply by cos(a):
  sin(a+b)*cos(a) = 0.5sin(2a-b) + 0.5sin(b)
  sin(a-b)*cos(a) = 0.5sin(2a+b) + 0.5sin(-b)
- Simply multiplying the original signal (sin(a)cos(b)) by cos(a) may result in no low frequency components
  sin(a)cos(b) * cos(a)
  = sin(a+b)*cos(a) + sin(a1b)*cos(a)
  = 0.5sin(2a-b) + 0.5sin(b) + 0.5sin(2a+b) + 0.5sin(-b)
  = 0.5sin(2a-b) + 0.5sin(2a+b) (both are high frequency terms)
  - Because sin(-b) = -sin(b)
- The result has no low frequency components at all

Heterodyning Caveat

- In the previous example, if we had multiplied the filter out by a sine (sin(a)) instead of a cosine (cos(a)), the output would have the correct low-frequency components
  - Work it out yourself to confirm this
- To account for this the heterodyne receiver multiplies the signal both by a sine and a cosine wave
- The outputs of both multiplications are filtered
- For the envelope combine the two low-pass filtered signals sample by sample as follows
  - Y = sqrt (X^2 + Y^2) [the hypotenuse]

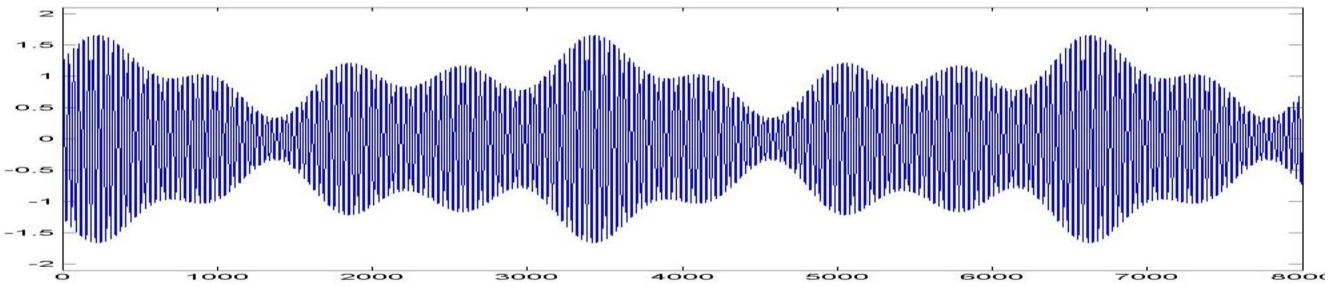
Heterodyne Receiver

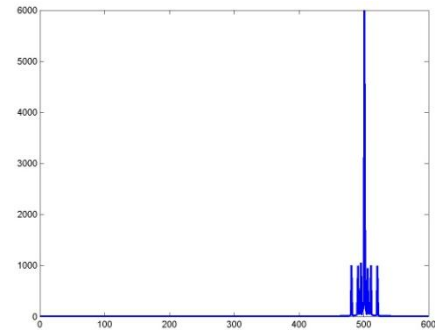


- The full Heterodyne receiver at the output of each filter in the phase vocoder

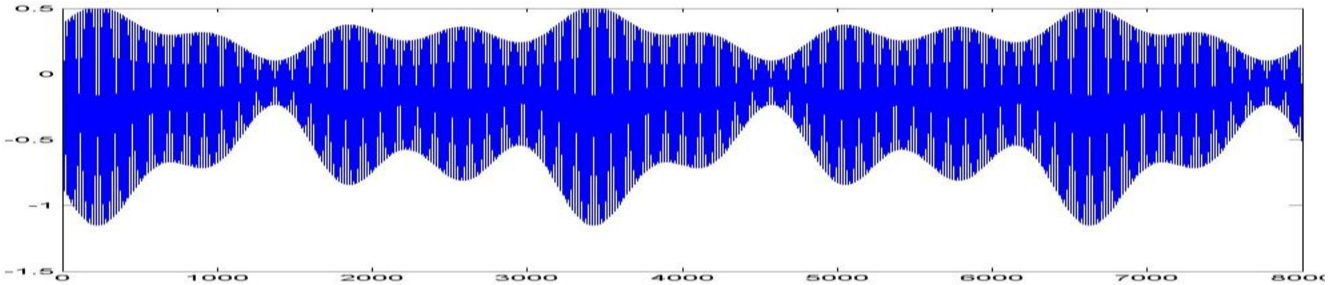
Heterodyning

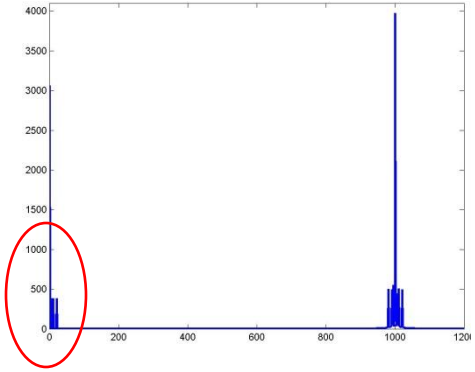
The original signal (output of one filter)



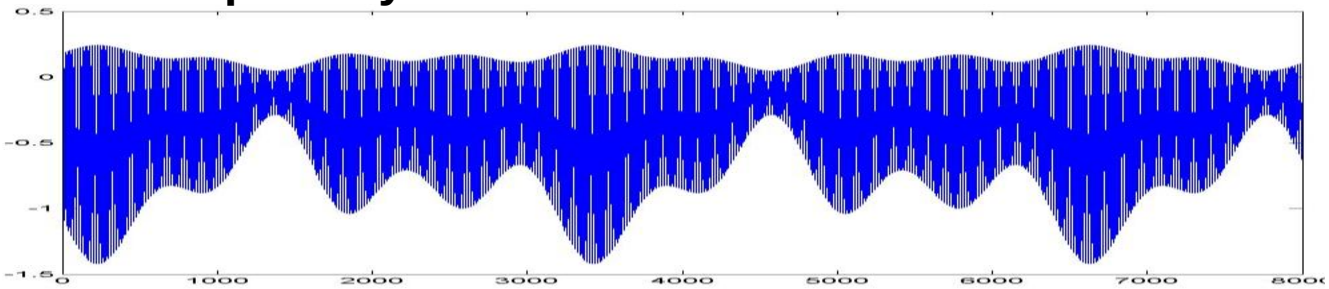


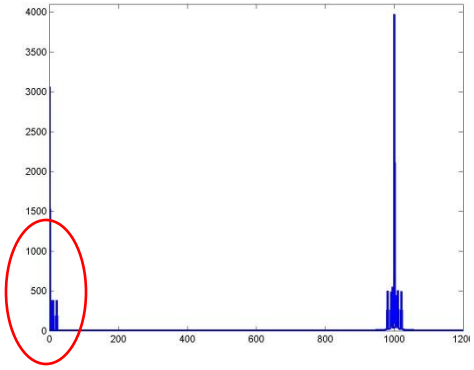
Multiplied by cosine





Multiplied by sine

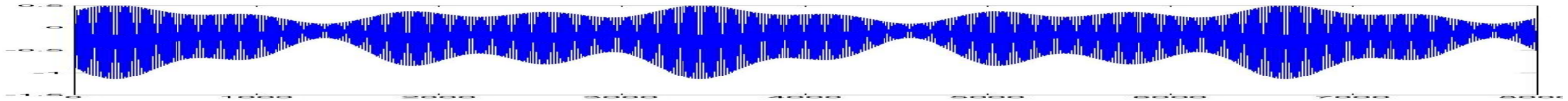




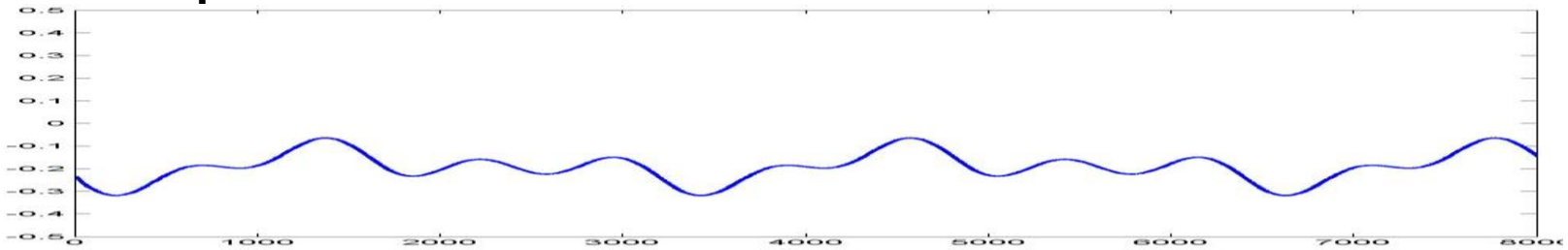
Both the output of the sine and the cosine are low pass filtered

Heterodyning

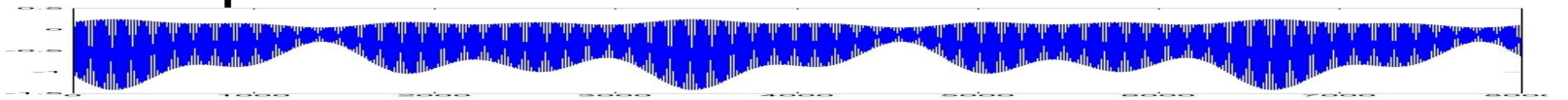
Cosine output



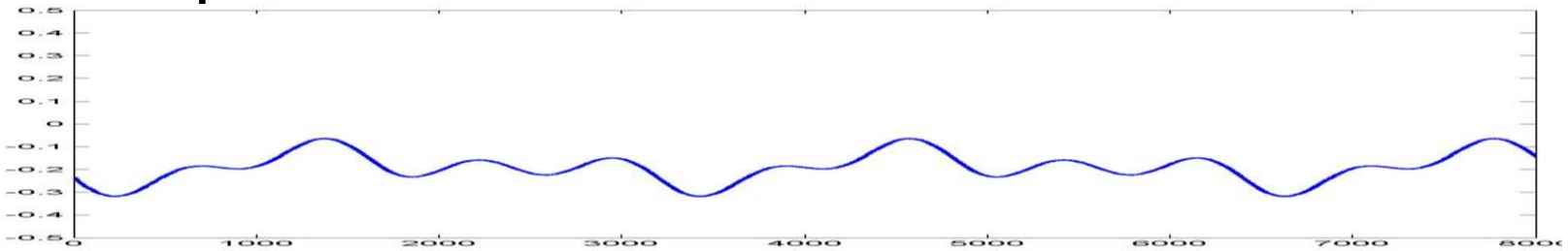
Low pass filtered



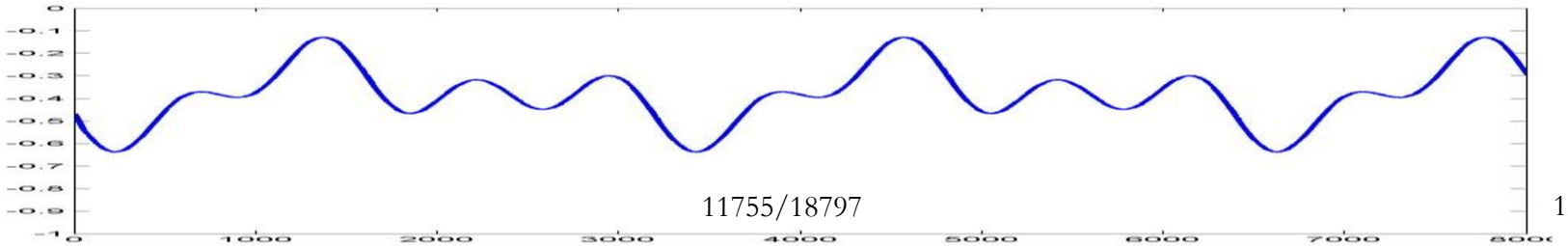
sine output



Low pass filtered



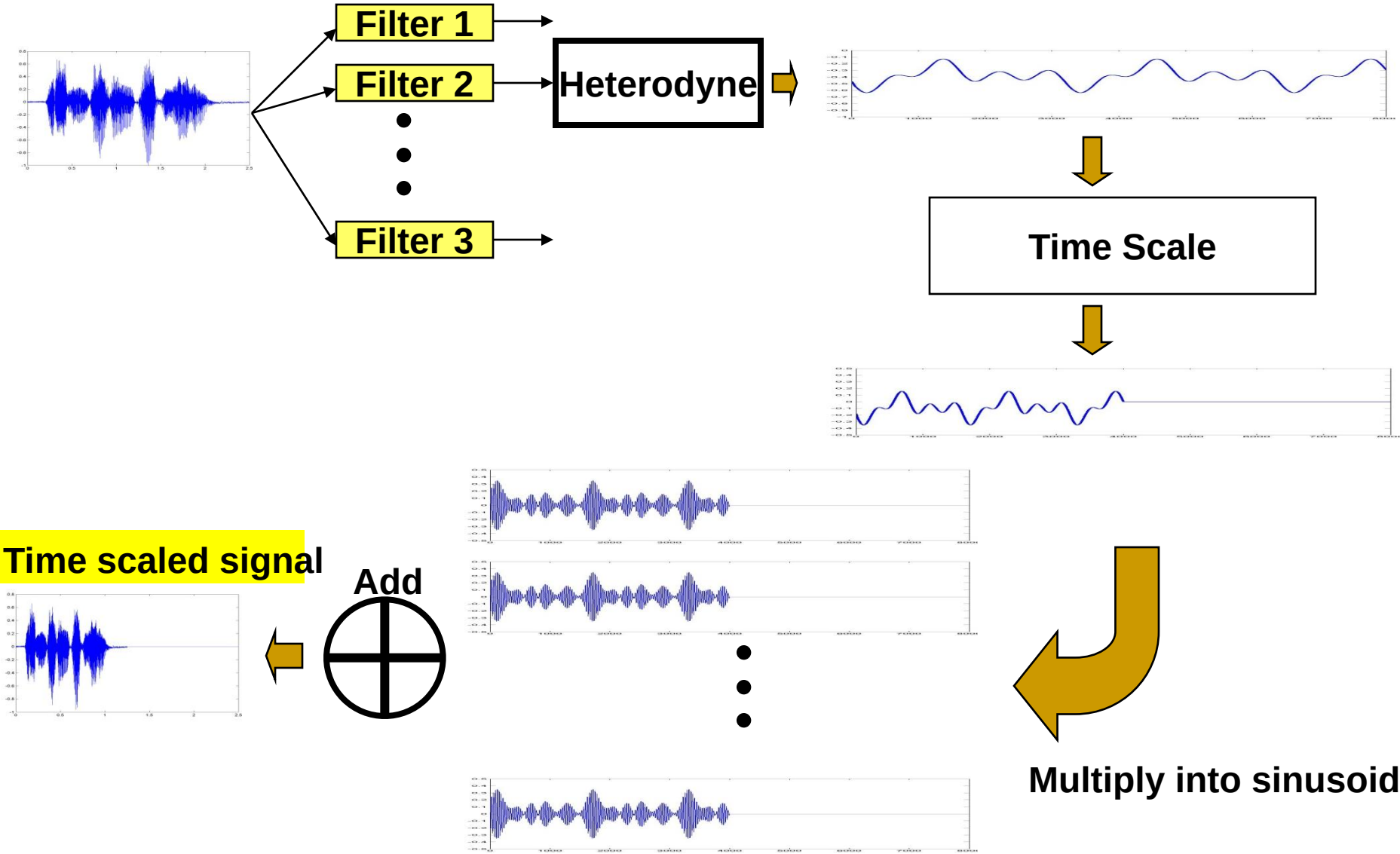
The hypotenuse the output of sine and cosine paths (the actual envelope)



Phase Vocoder: Time Scaling

- Time-scale the envelope of each sinusoid by decimation (dropping samples) or interpolation
- Multiply the time scaled envelope back into a sinusoid at the center frequency of the filter
- Do this for every filter output and add filter outputs back

Phase Vocoder: Time Scaling



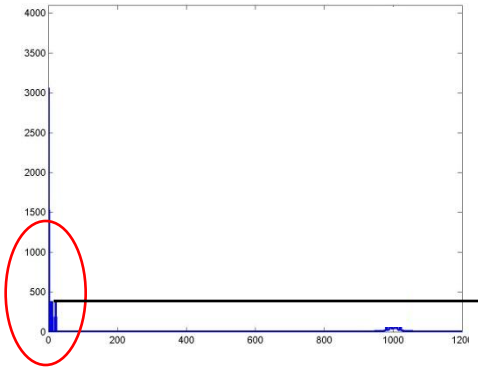
How does one implement it

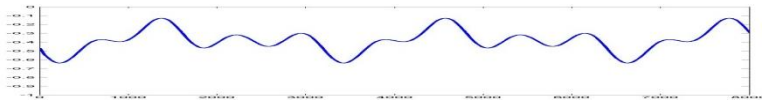
- Sometimes implementation lags behind intention

Phase Vocoder: Implementation

- But actual implementation with time-domain filters, while practicable, is not cost effective
- Can be performed much more efficiently in the frequency domain::

- The low pass filtered envelope of the output of any filter typically has a bandwidth less than 50 Hz
  - It is sufficient to sample it at 100 samples per second
    - 16000 samples per second are not necessary



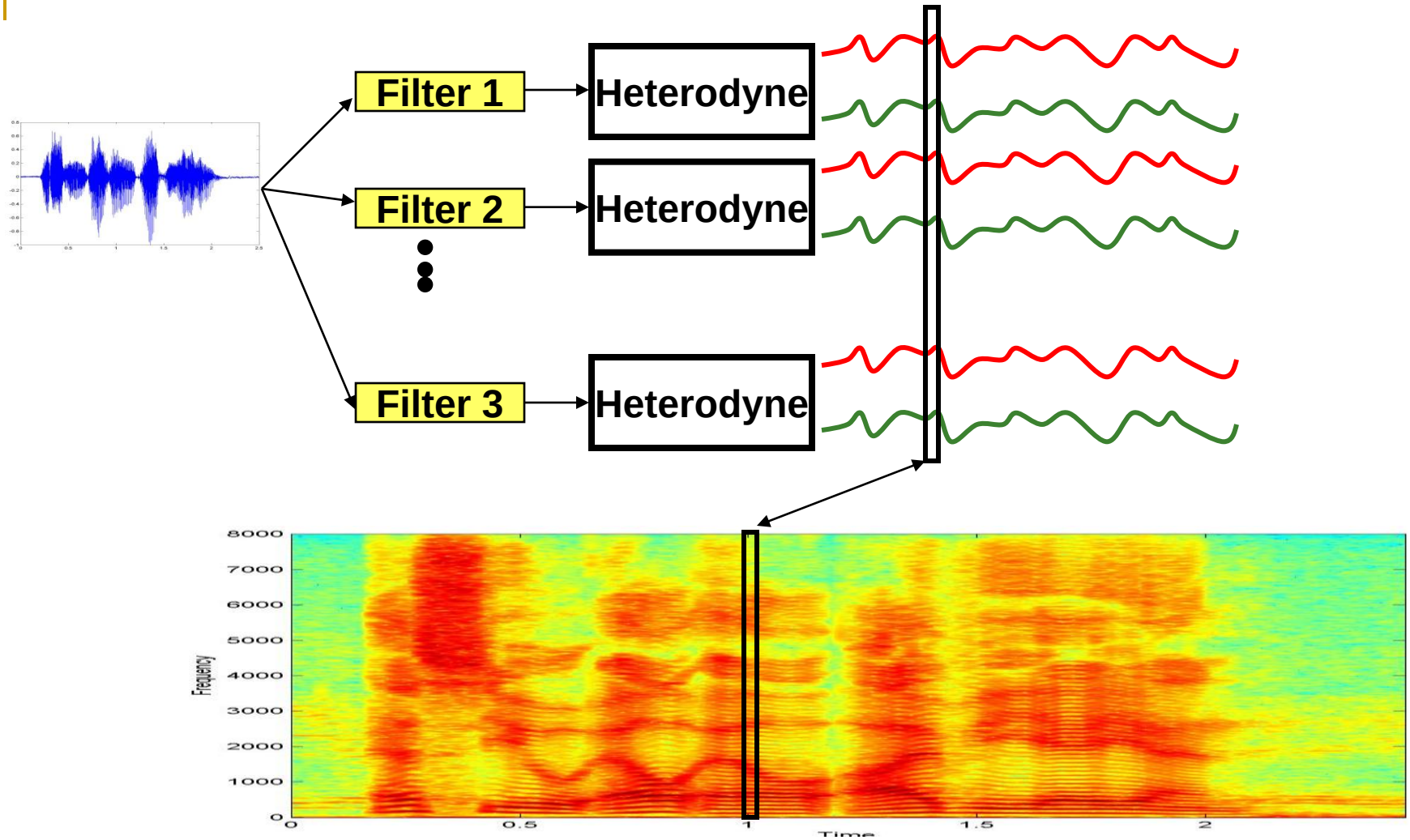


The highest frequency is less than 100 Hz

What is an STFT

- The real and imaginary parts of an STFT are snapshots of envelops of the output of the filters in a filterbank
  - The real and imaginary parts of the STFT represent the cosine and sine outputs of a heterodyne at the output of each filter
- An STFT that computes N spectral vectors per second of the signal takes these snapshots N times a second
  - If N > 2*(the largest frequency in the envelope), this is sufficient to reconstruct the entire envelope, and thereby the entire signal

What is an STFT

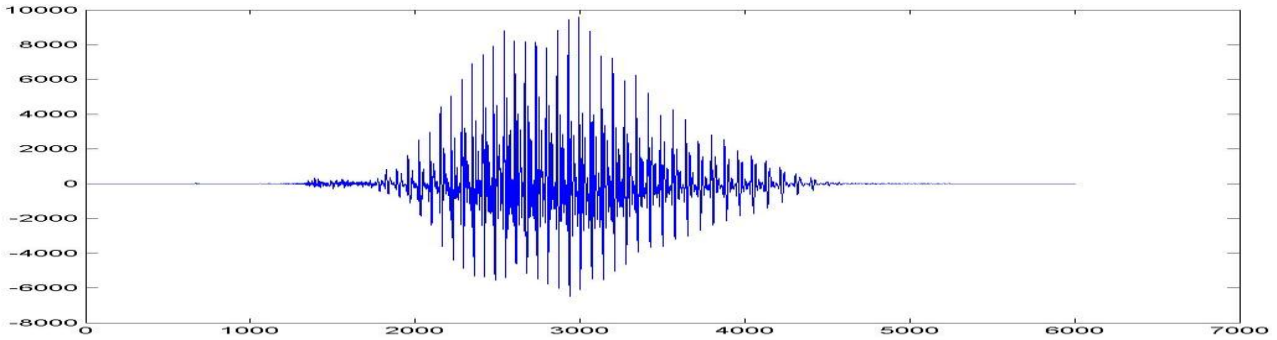


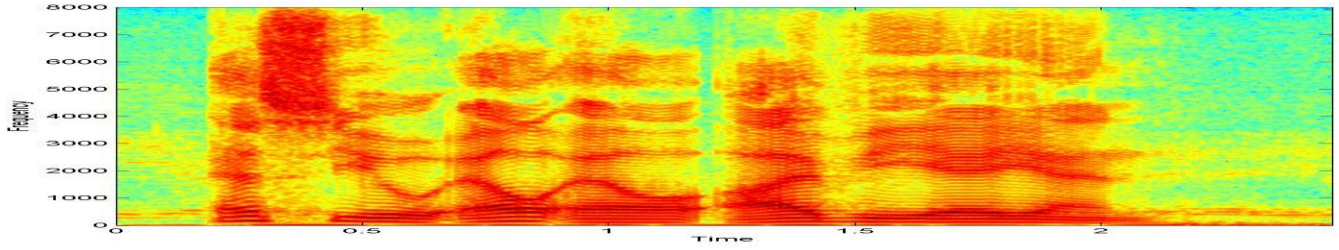
- The STFT captures snapshots of the outputs of the filterbank-heterodyne combination

Phase Vocoder: Implementation

- A short-time Fourier transform of a signal effectively samples the output of NFFT/2 filters, once per spectral vector computed
- An STFT that computes 100 spectral vectors per second effectively samples the envelope of the output of the filters at 100 Hz
  - Each spectral value is one sample of the output of one filter
  - Inversion of the STFT effectively multiplies these envelope values by sinusoids at the center frequencies of the filters and add them all back in
- Time-scale modification can be obtained by resampling the sequence of spectral vectors in the STFT of the signal

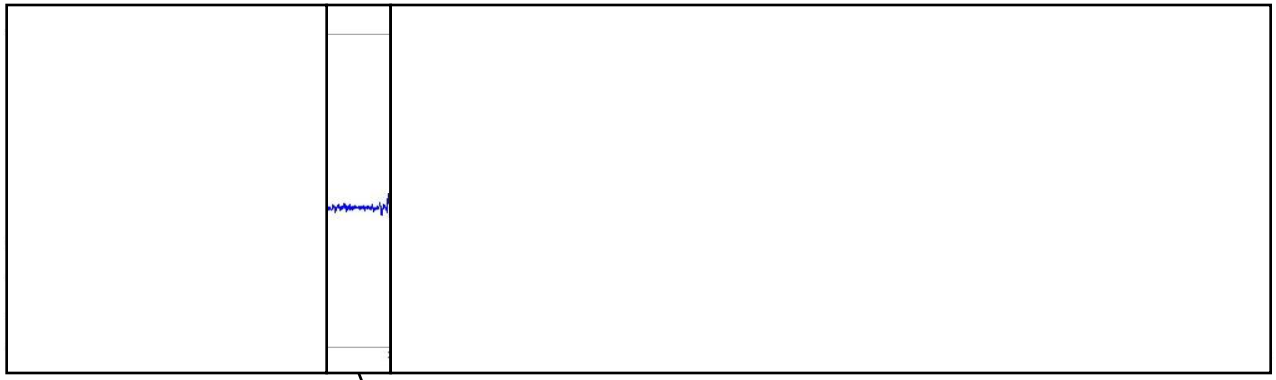
Recall: Computing a Spectrogram





Compute Fourier Spectra of segments of speech and stack them side-by-side

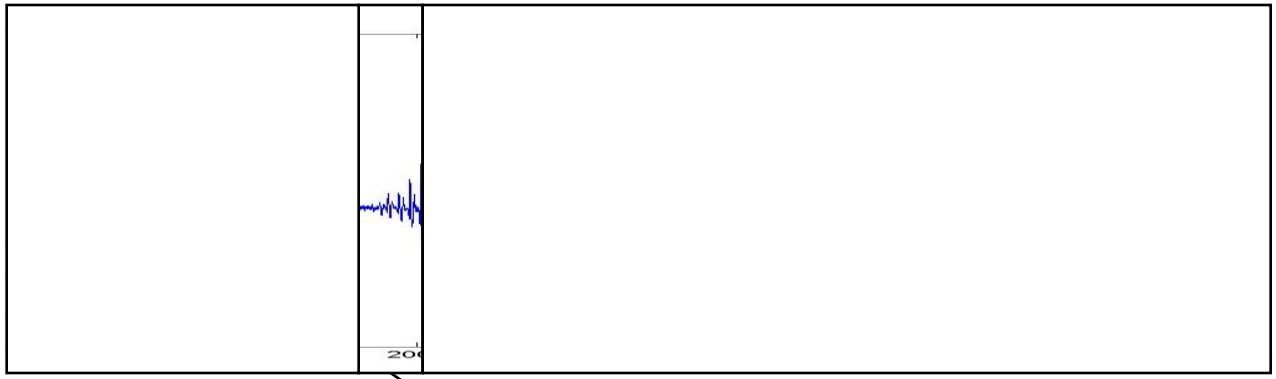
Recall: Computing a Spectrogram



frequency
frequency
frequency
frequency
frequency
frequency
frequency

Compute Fourier Spectra of segments of speech and stack them side-by-side

Recall: Computing a Spectrogram

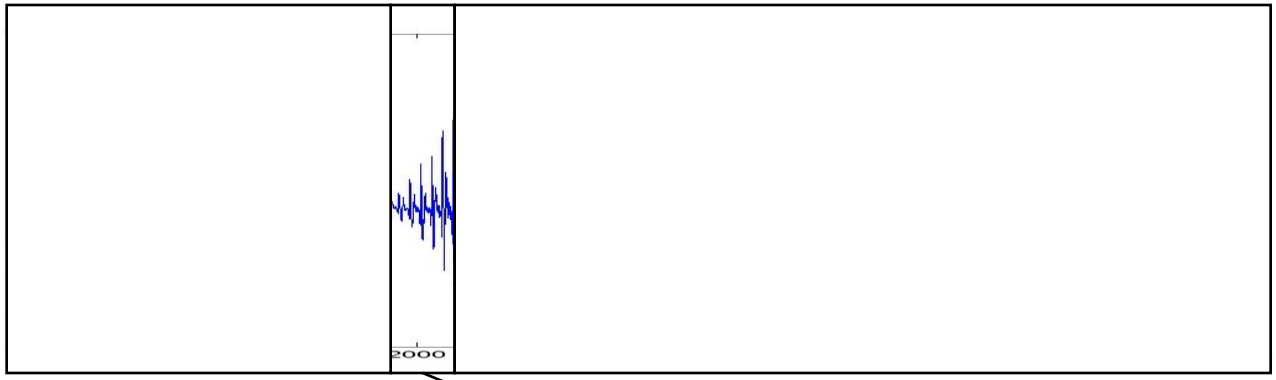


frequency frequency frequency frequency frequency frequency frequency

frequency frequency frequency frequency frequency frequency frequency

Compute Fourier Spectra of segments of speech and stack them side-by-side

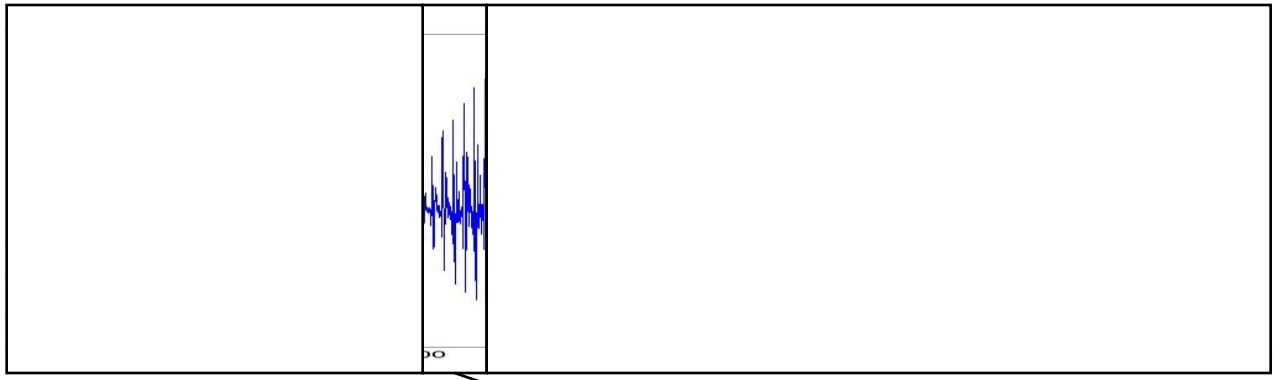
Recall: Computing a Spectrogram

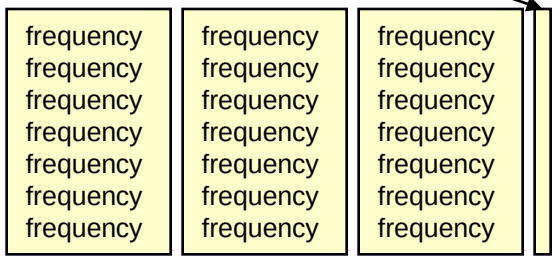


frequency	frequency	frequency
frequency	frequency	frequency
frequency	frequency	frequency
frequency	frequency	frequency
frequency	frequency	frequency
frequency	frequency	frequency
frequency	frequency	frequency

Compute Fourier Spectra of segments of speech and stack them side-by-side

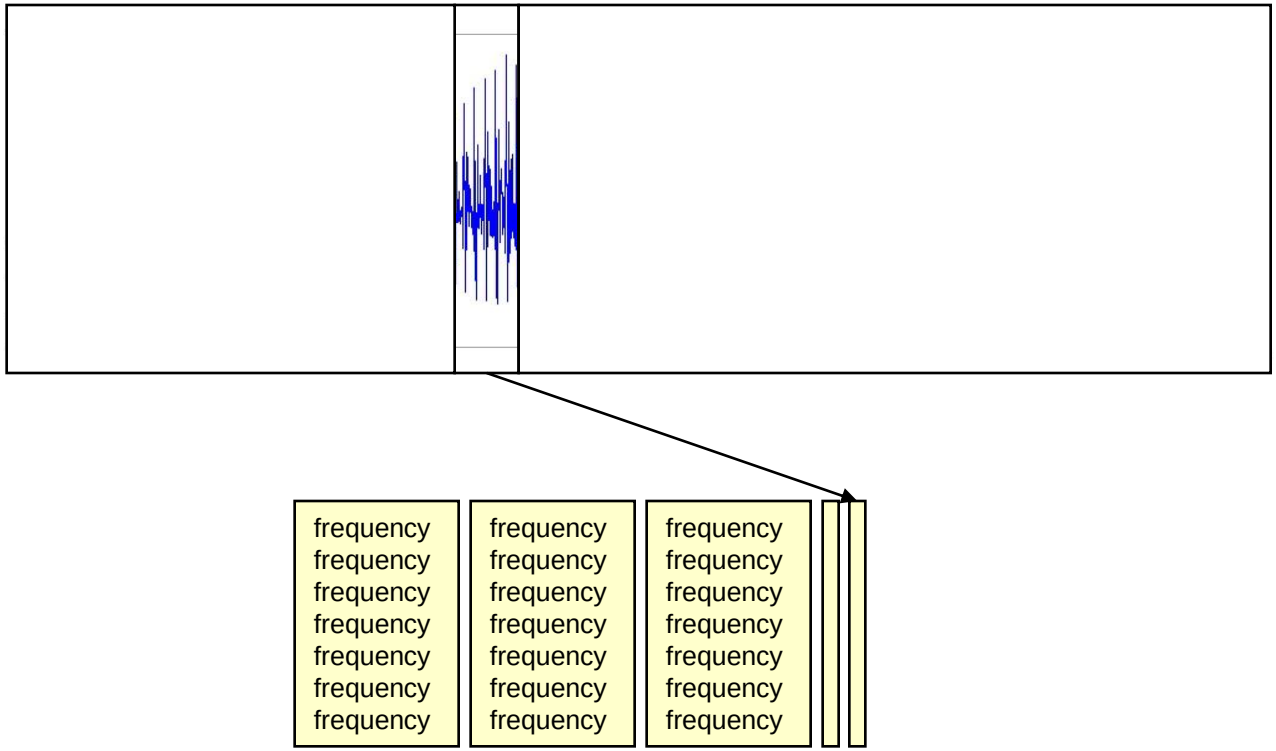
Recall: Computing a Spectrogram





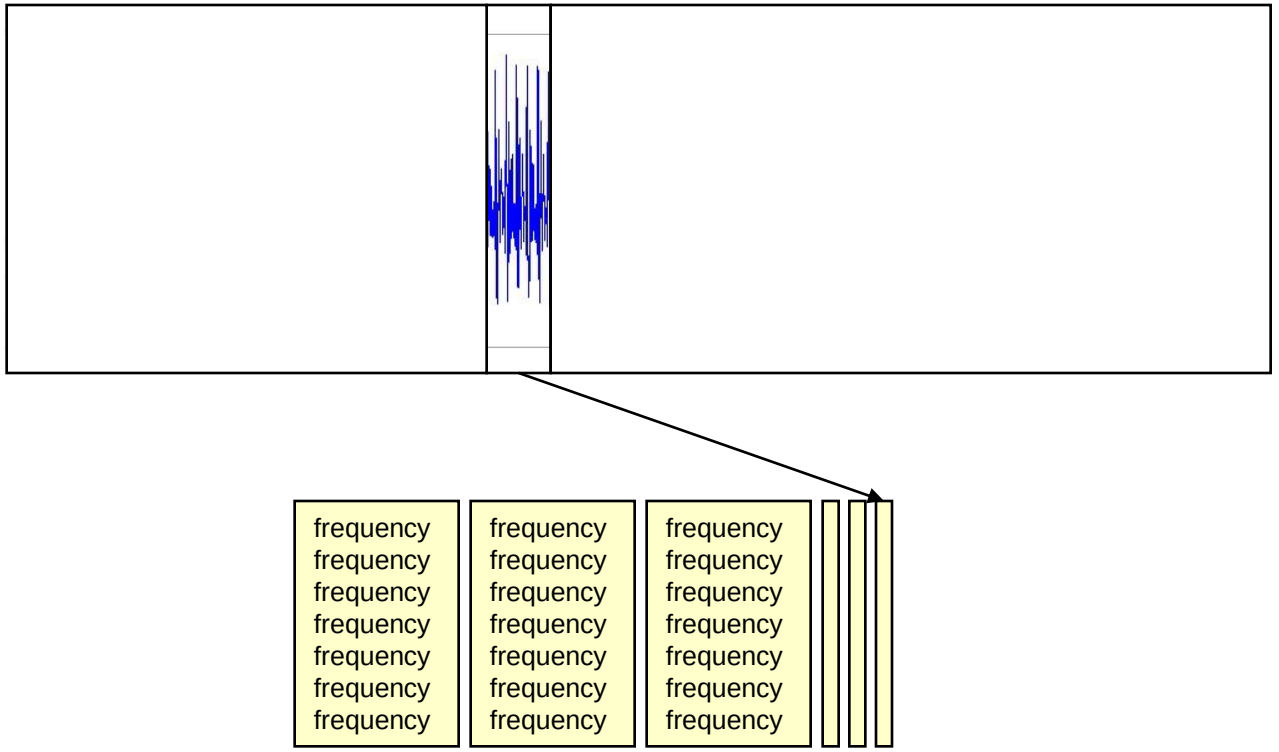
Compute Fourier Spectra of segments of speech and stack them side-by-side

Recall: Computing a Spectrogram



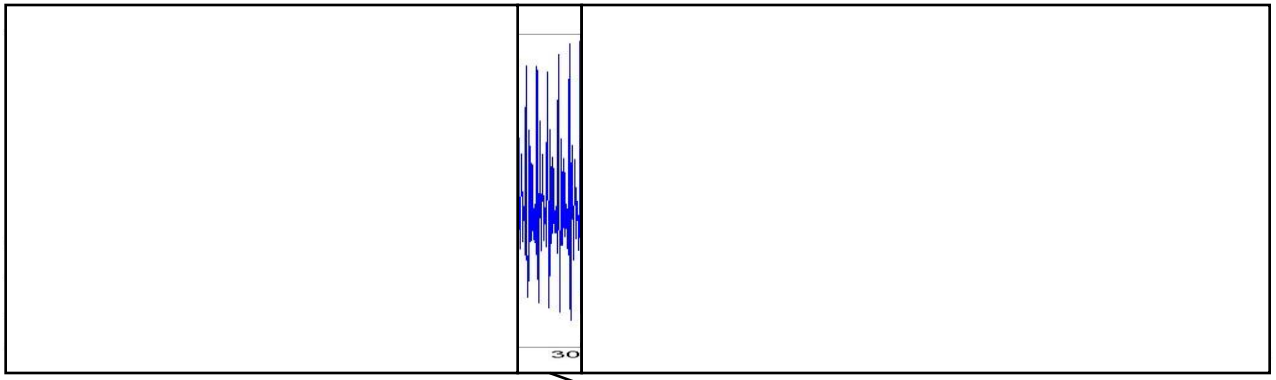
Compute Fourier Spectra of segments of speech and stack them side-by-side

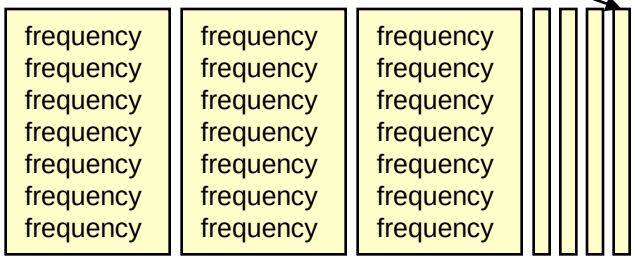
Recall: Computing a Spectrogram



Compute Fourier Spectra of segments of speech and stack them side-by-side

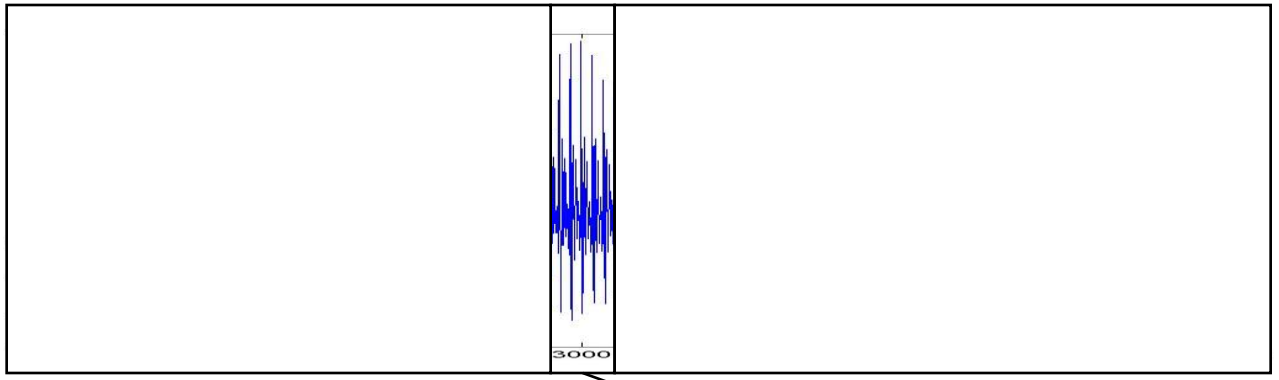
Recall: Computing a Spectrogram

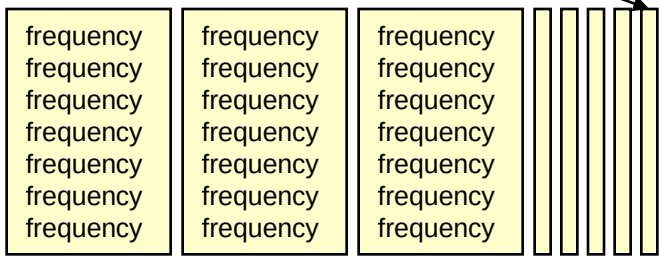




Compute Fourier Spectra of segments of speech and stack them side-by-side

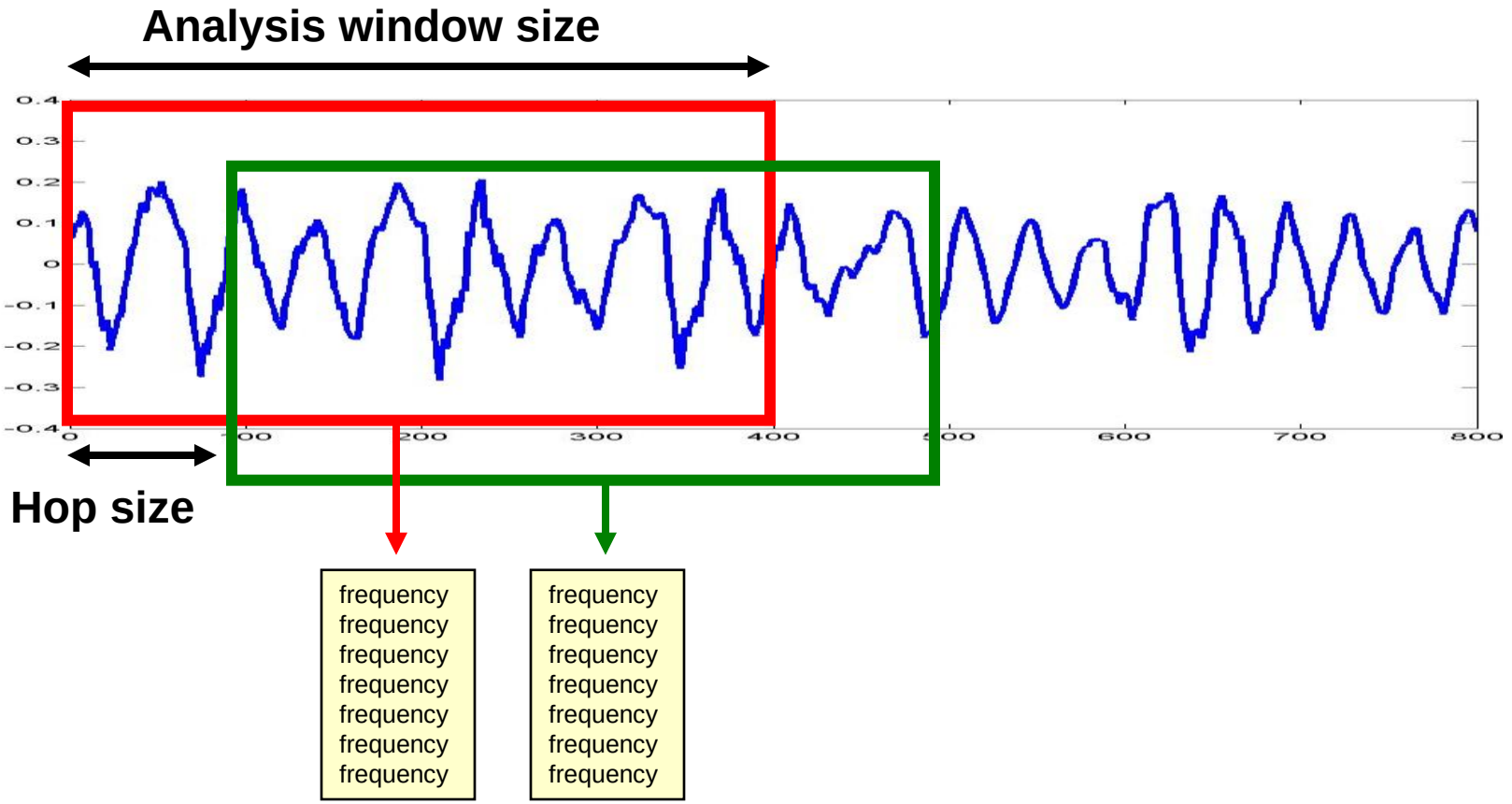
Recall: Computing a Spectrogram





Compute Fourier Spectra of segments of speech and stack them side-by-side

Zooming in

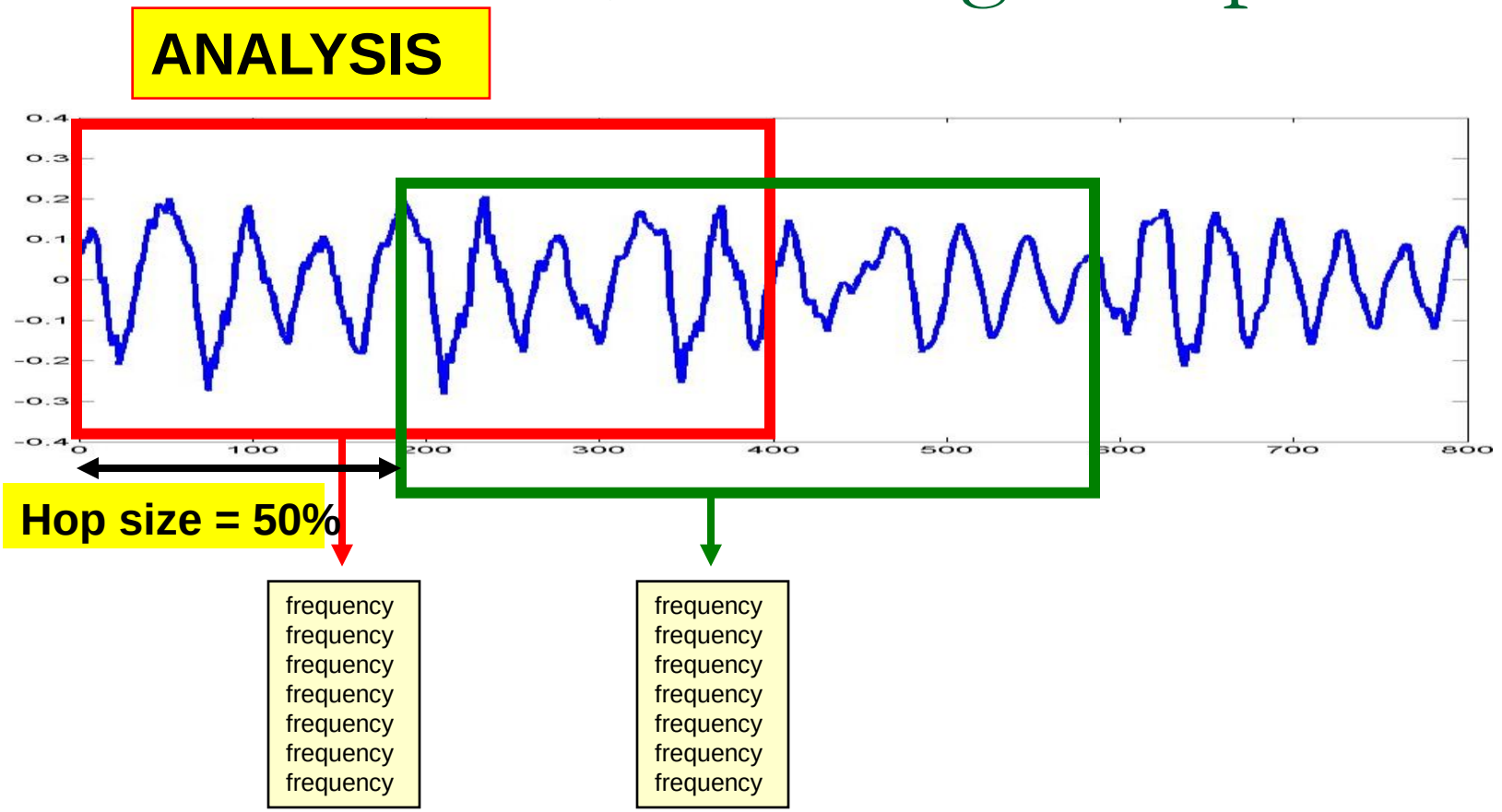


- The two parameters of an STFT are the length of the analysis window and the hop between adjacent analysis windows

Phase Vocoder: Implementation

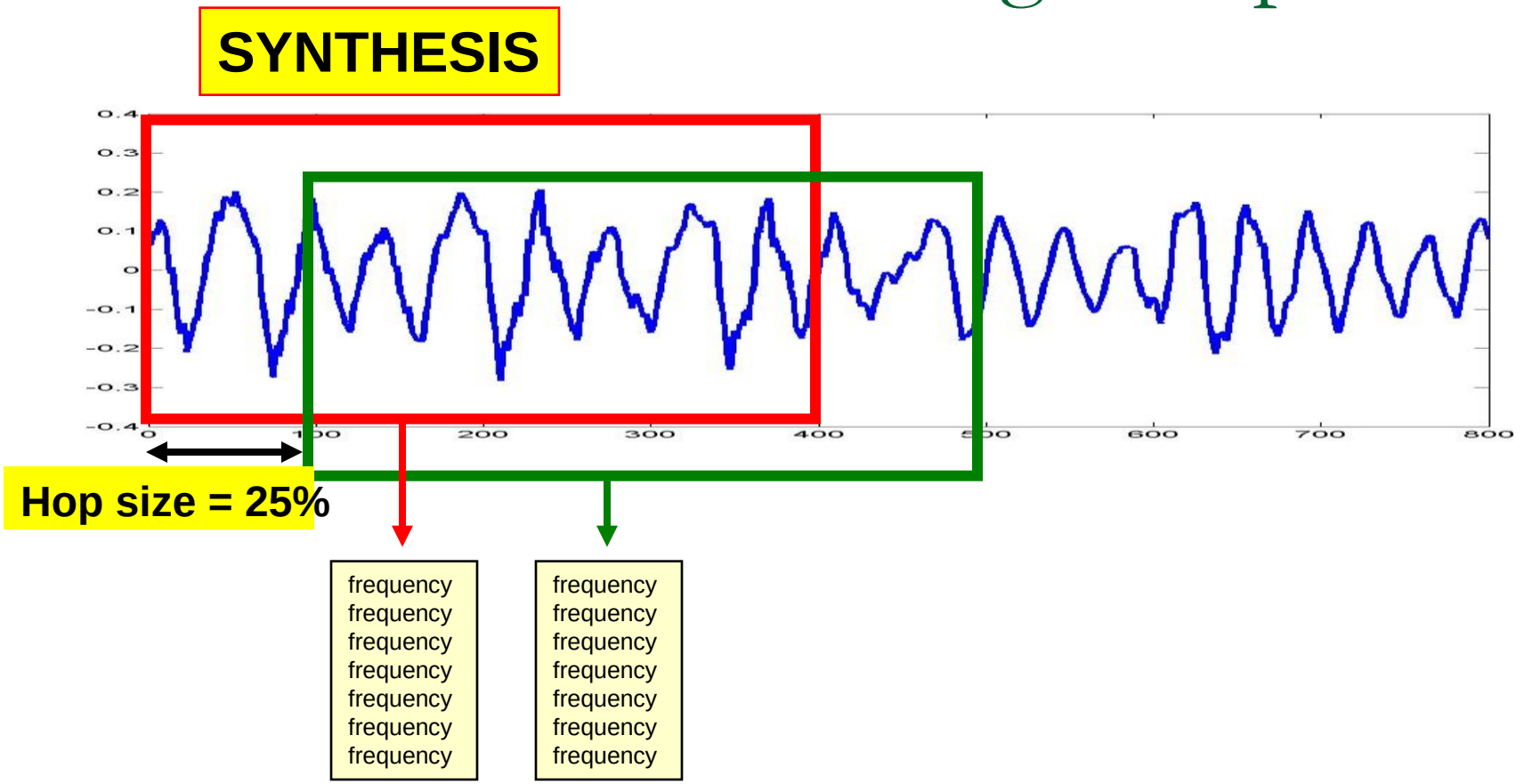
- Reconstruction of signal from time-scaled STFT:
  - Inversion of an STFT works best when the hop between adjacent frames (analysis windows) is no more than 25%
  - For time-scaling, the time-scaled STFT *must* have at least 100 frames per second (to ensure that the envelope is not under sampled)
  - Therefore, we select analysis window size and hopsize such that the hop size is 25% or less, and we get more than 100frames per second
    - A good selection is an analysis window of 32ms, with a hop size of 8ms
- Time-scaling has two stages:
  - Analysis: Compute an STFT from the signal with a hop size other than 25% (8ms)
  - Synthesis: Recompute the signal from the STFT assuming a hop size of 25%
    - The difference between the two rates will result in rate change

Phase vocoder, doubling the speed



- To double the speech rate, during analysis we compute STFTs with a hop size of 50%

Phase vocoder, doubling the speed



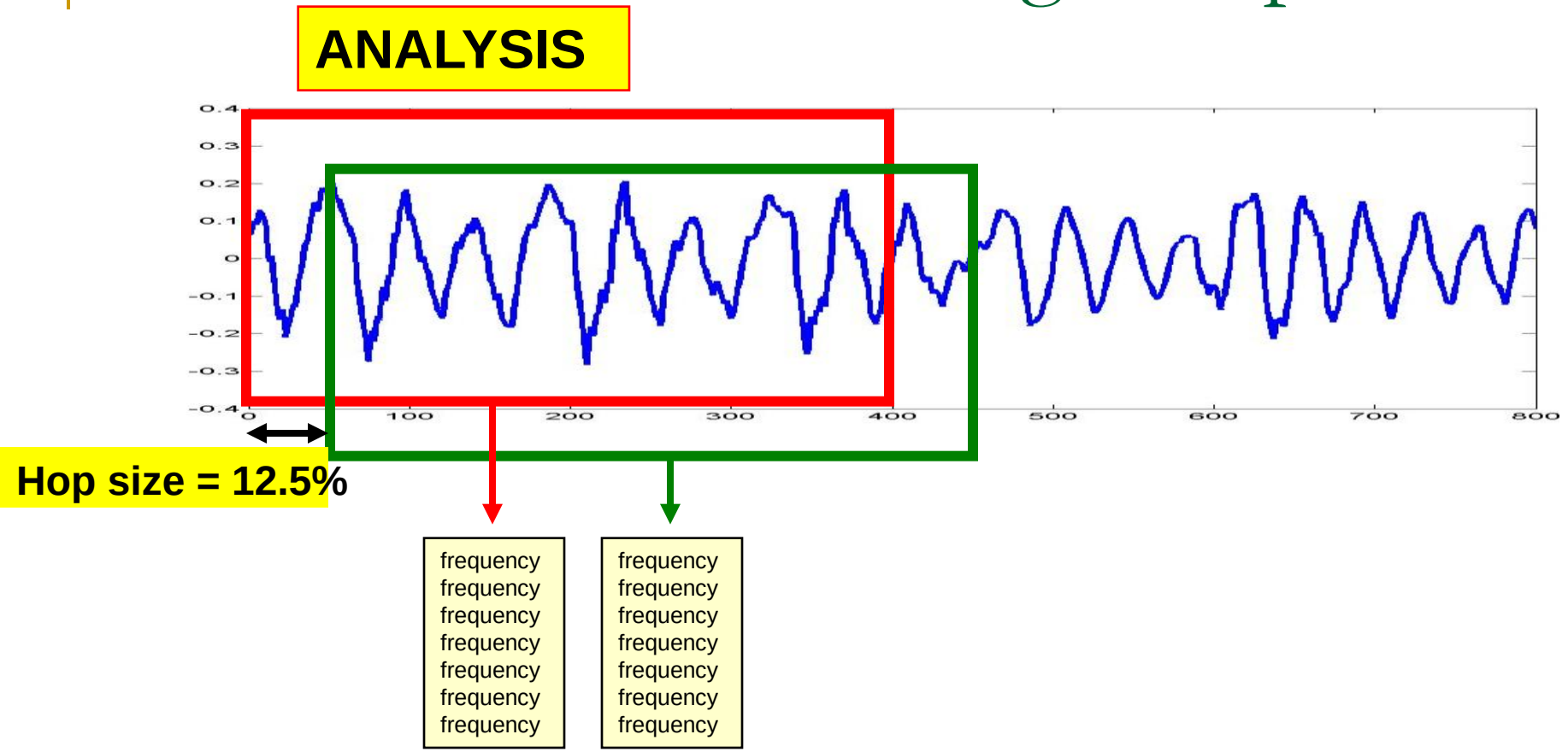
- To double the speech rate, during analysis we compute STFTs with a hop size of 50%
- During synthesis, use the SAME STFTs, but assume a hopsize of 25%

Doubling the speed

- STFT analysis with an analysis window of 40ms(say) with 50% hop computes 50 frames per second
  - 20 seconds of speech would generate 1000 frames

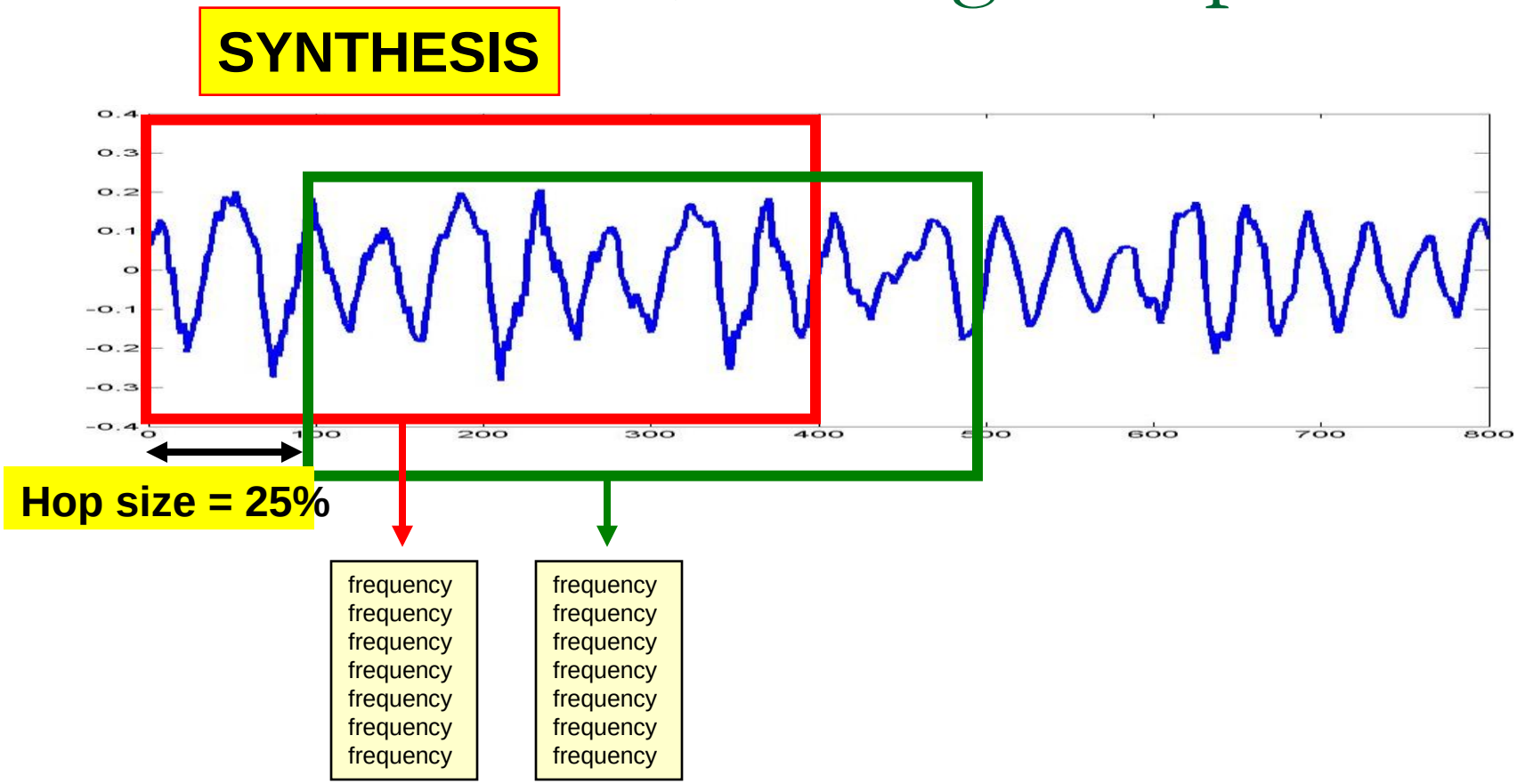
- STFT synthesis using all the spectral vectors, but with a 25% hop between frames inverts 100 frames per second
  - 1000 frames would be played back in 10 seconds
  - The playback speed will be double that of the original signal

Phase vocoder, halving the speed



- To double the speech rate, during analysis we compute STFTs with a hop size of 12.5%

Phase vocoder, halving the speed



- To double the speech rate, during analysis we compute STFTs with a hop size of 12.5%
- During synthesis, use the SAME STFTs, but assume a hopsize of 25%

Halving the speed

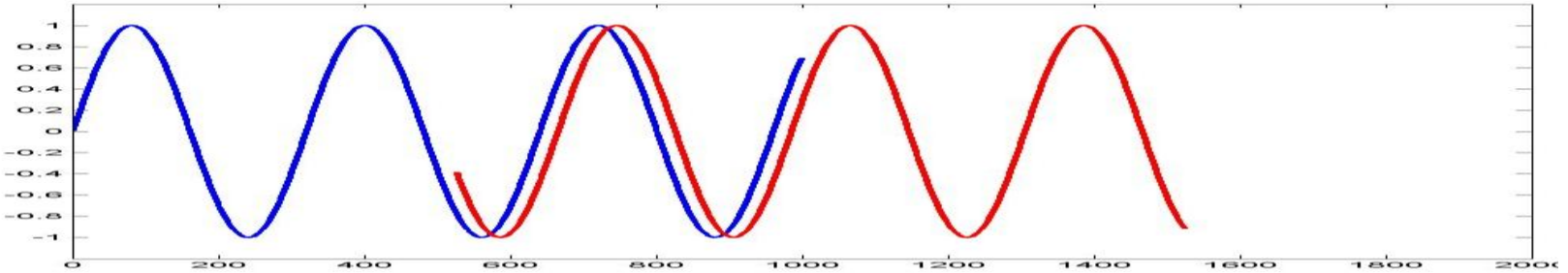
- STFT analysis with an analysis window of 40ms(say) with 12.5% hop computes 200 frames per second
  - 20 seconds of speech would generate 4000 frames
- STFT synthesis using all the spectral vectors, but with a 25% hop between frames inverts 100 frames per second
  - 4000 frames would be played back in 40 seconds
  - The playback speed will be half that of the original signal

What about the phase?

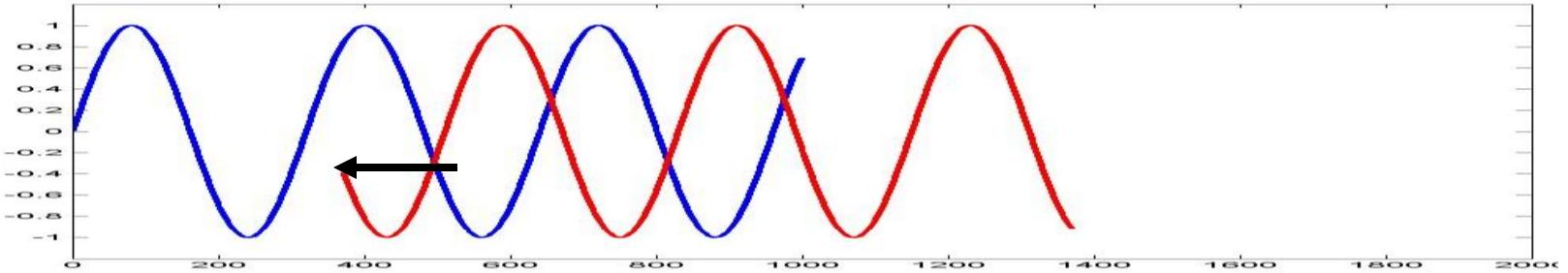
- Simply shifting the reconstructed signal for each analysis window to the appropriate hop size can cause some strange perceptual artifacts.
  - This is because the Fourier spectrum is complex
  - The phase of the Fourier spectra of frames that are X samples apart will differ by an amount that is different from the phase of Fourier spectra of frames that are Y samples apart
- We must account for this difference
  - If not, we might as well just window the signal, shift the frames to the correct overlap, and add without the FFT and inverse FFT inbetween

The phase problem in pictures

- The inverse fourier transform multiplies Fourier coefficients into sinusoids and adds them up
- When the synthesis hop size is identical to the analysis hop size, sinusoids in ajacent frames will line up perfectly for addition



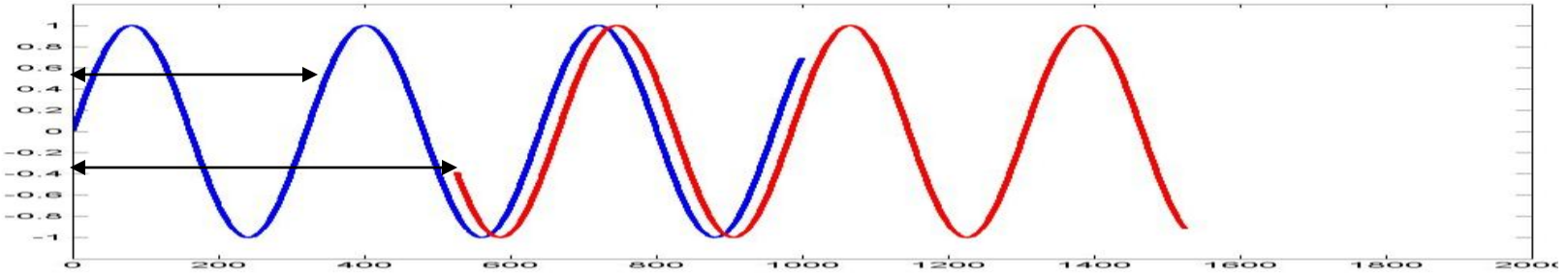
- At the analysis hop size, the sinusoid in the second frame lines up perfectly with the sinusoid in the earlier frame



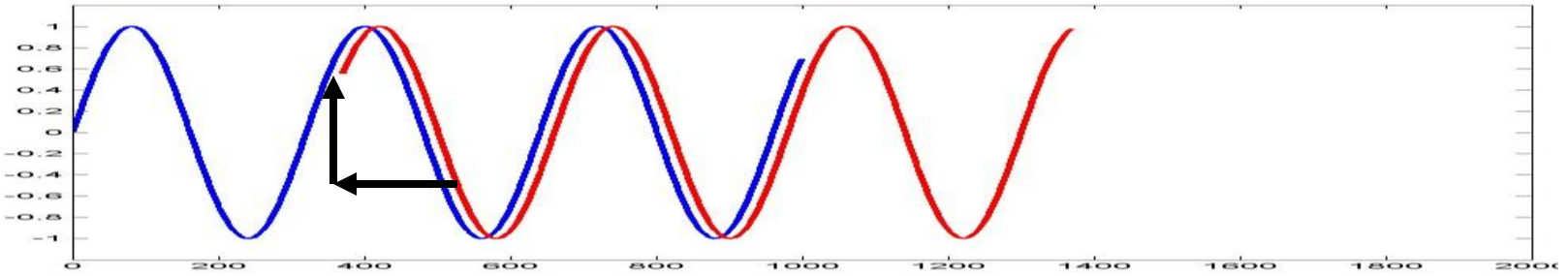
- If the second frame is shifted back (or forward) with respect to the first, for synthesis, the sinusoids no longer line up

The solution

- Modify the phase (the starting point) of the sinusoid of the second frame to line up properly with the first



- First compute how much the phase has advanced between the two frames (in radians)
- Compute how much is would have advanced at the desired new starting point of the frame
- Change the initial phase of the second frame to this new value



- The sinusoids will now line up

In Symbolic Terms..

- Do not reconstruct the signal directly from the FFT.
- Compute the magnitude and phase of every frame.
  - Let these be Mag(t) and Phase(t) for the t-th frame
- Compute the Phase difference (at any frequency) between the phase of the current frame and the phase of the previous frame
  - DeltaPhase(t) = Phase(t) – Phase(t-1)
- If the original hop size was X samples, and it was adjusted to Y samples, scale DeltaPhase by Y/X
  - DeltaPhaseNew(t) = Y * DeltaPhase(t) / X
  - This adjusts the phase difference between adjacent frames to what it must be for the modified hop size

In symbolic terms

- Reconstruct the phase for the t-th frame by adding the modified delta phase to the modified phase of the previous frame
  - PhaseNew(t) = PhaseNew(t-1) + DeltaPhaseNew(t)
- Reconstruct the entire Fourier spectrum from the magnitude and the new phase
  - FFTNew = Mag * exp(sqrt(-1) * PhaseNew)
- Invert and reconstruct the signal

Phase vocoder time scaling:

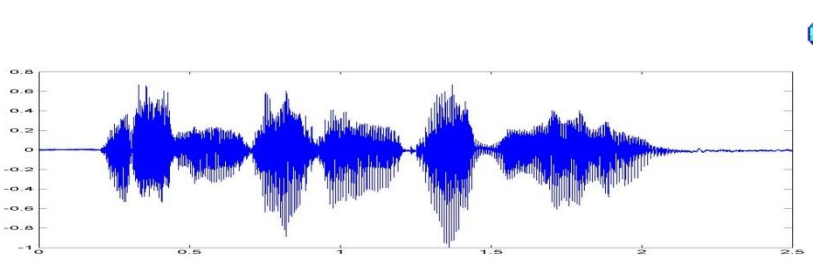
1. Determine a good analysis window size
    1. 32-64ms is good
2. If analysis windows are of size Z, the ideal hopsize for reconstruction is X = Z/4
3. To get a desired time scaling, select an integer Y such that (X/Y) is the desired scaling factor
4. During analysis, compute STFTs spectra for windows with a hopsize Y
5. Compute the magnitude and phase of the spectra for each frame
6. Compute the delta phase for each frame with respect to the previous frame
7. Adjust delta phase by scaling factor (X/Y)
8. Recompute phase by adding new delta phase to the (modified) phase of the previous window
9. Combine phase and magnitude to get new STFT coeffecients
10. Reconstruct a signal from the modified STFT with a hop size of X

Phase vocoder time scaling:

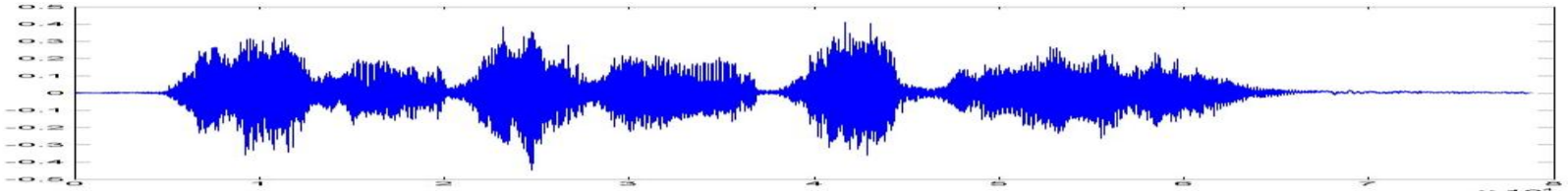
- Alternate mechanism (more correct):
  - Resample the magnitude spectrum:
    - Consider each frequency band as the output of a filter
    - Resample the sequence to get the desired number of frames
      - Matlab's resample routine
    - Predict phase at the appropriate positions using the mechanism described earlier
    - Reconstruct stft
    - Reconstruct signal

Time Scaling: Some examples

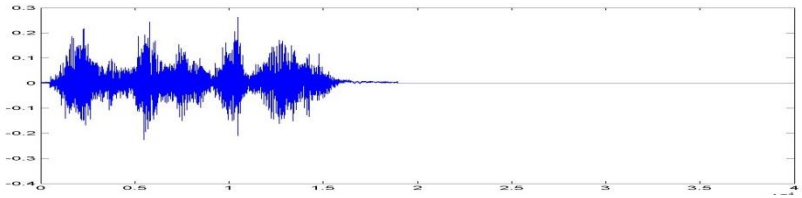
- A speech signal



- At half the rate

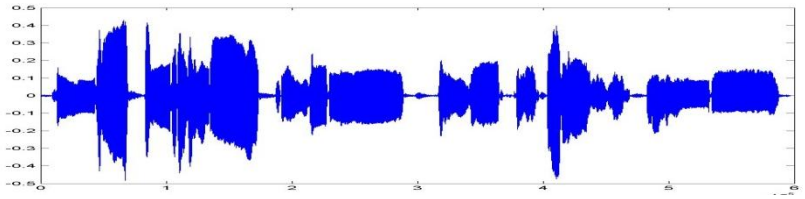


- At double rate

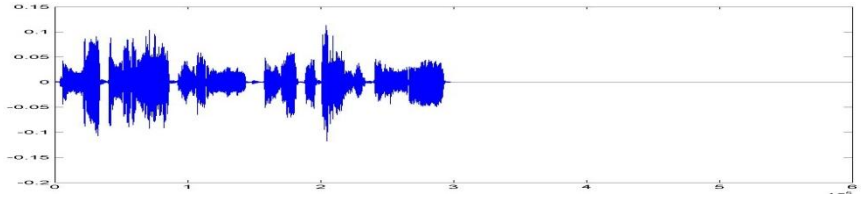


Psola Examples

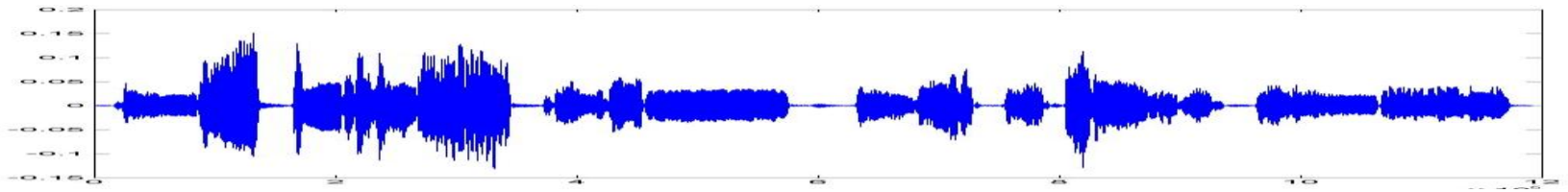
- Over the rainbow



- Faster

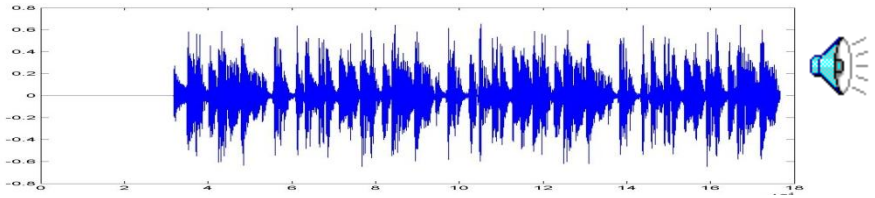


- Slower

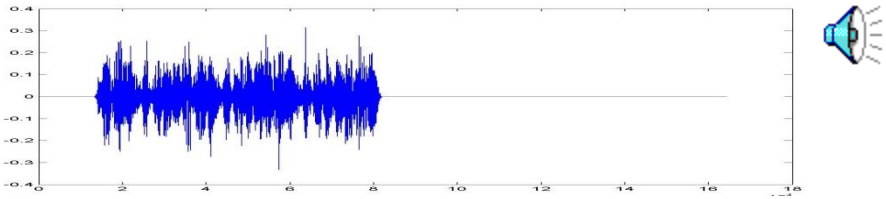


Psola Examples

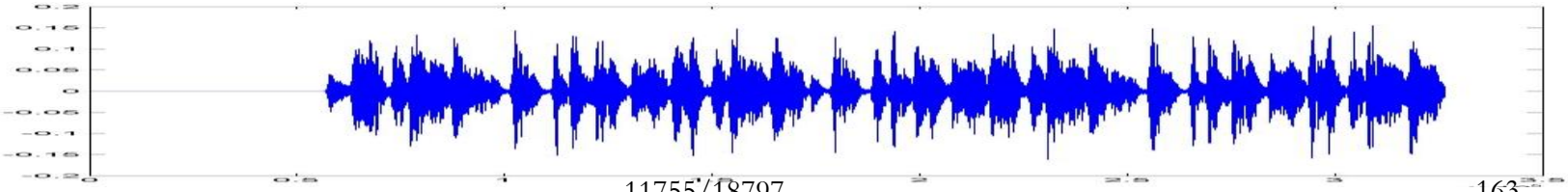
- A music segment



- Slower



- Faster



Phase Vocoder Time Scaling

- Basic procedure much simpler to implement than PSOLA when done
  - Sophisticated versions that attempt to simulate actual filterbanks and heterodynes are much more complex
- Effective for any kind of sound
  - Including speech, monophonic music, polyphonic music

Pitch Shifting

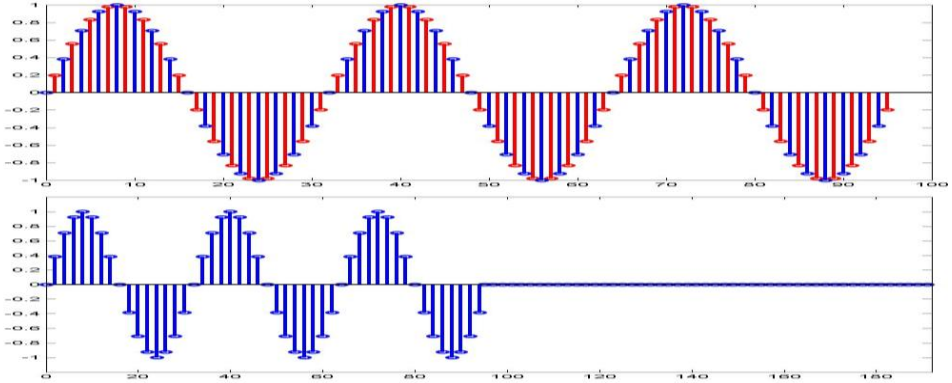
- Time Scaling is the procedure by which we make a segment of audio longer or shorter
  - Without modifying the pitch (the sound doesn't sound squeaky or bass)
- Pitch shifting is the inverse process: Scaling the pitch of the signal without modifying the length
  - Remember that simply dropping samples (or interpolating new samples) increases (decreases) the pitch, but also modifies the length of the recording
  - We want the same length.

Pitch Shifting

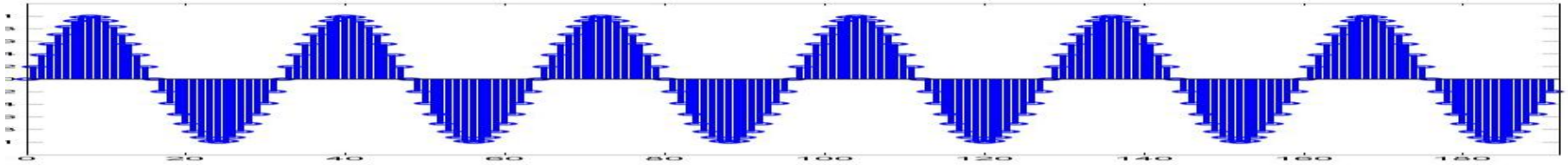
- Simple interpolation or decimation of the signal effects pitch shift, but changes the length of the signal
  - Resampling by factor X changes pitch by 1/X
  - Changes length by X
- Solution: To get a pitch scaling of factor X, first change the length of the signal by 1/X using a Phase vocoder
- Then resample
  - The length change due to resampling and the length change from the phase vocoder cancel out
  -

Pitch Shifting

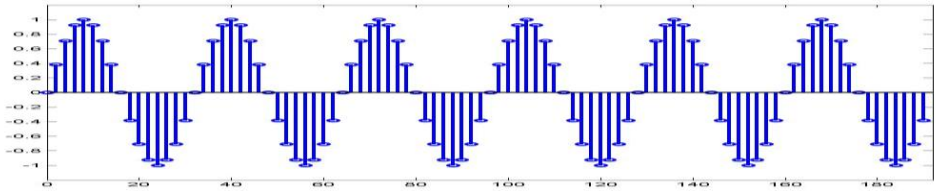
- Simple decimation (dropping of samples) increases the pitch, but reduces the length



- A phase vocoder initially increases the length by the right amount

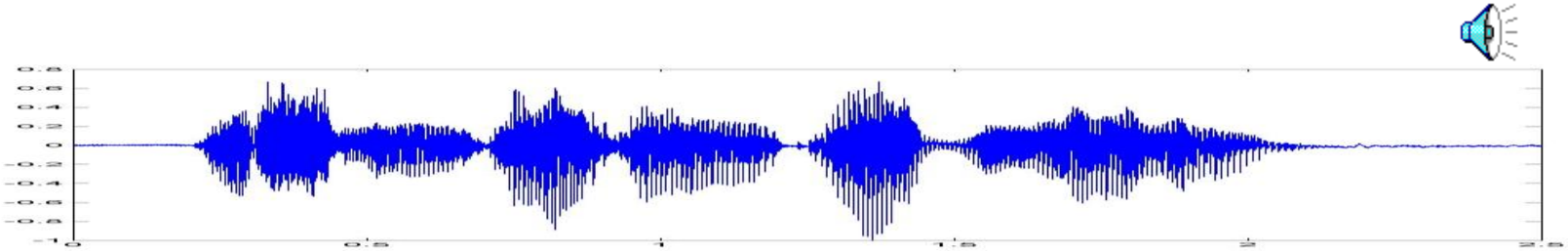


- Subsequent decimation gives a signal of the right pitch and length

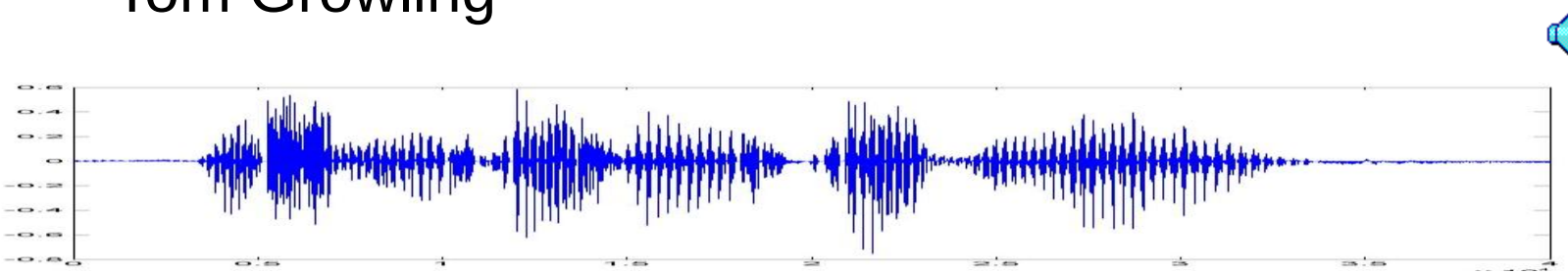


PSOLA Examples

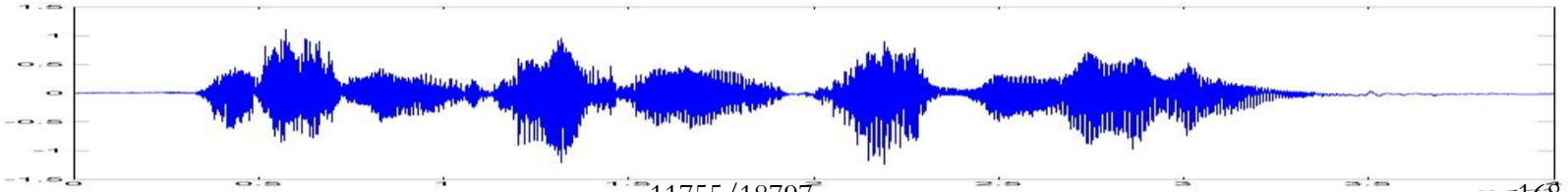
- Tom



- Tom Growling

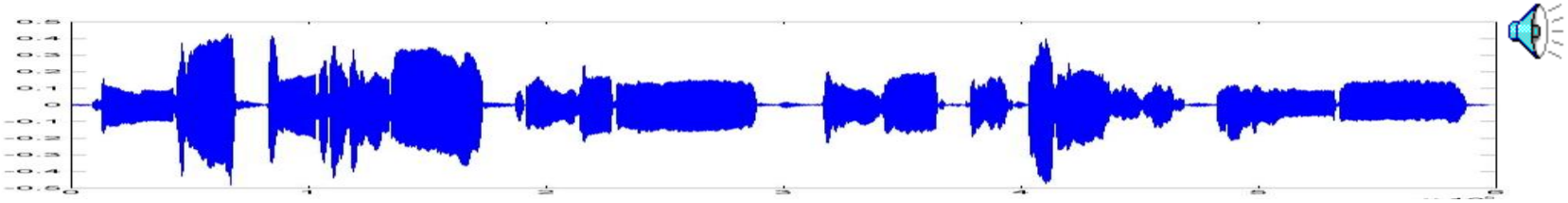


- Tom Squeaking

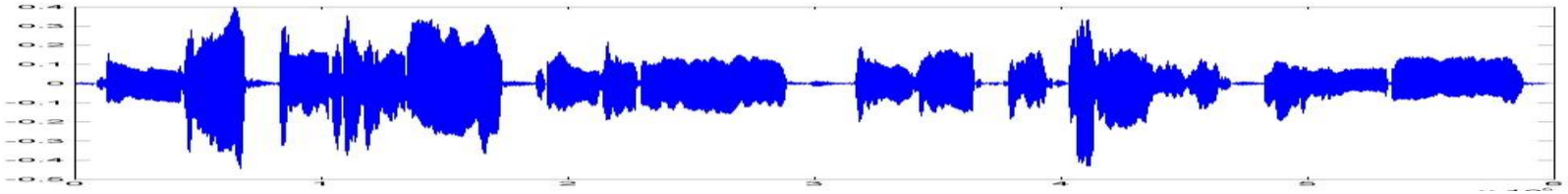


PSOLA Examples

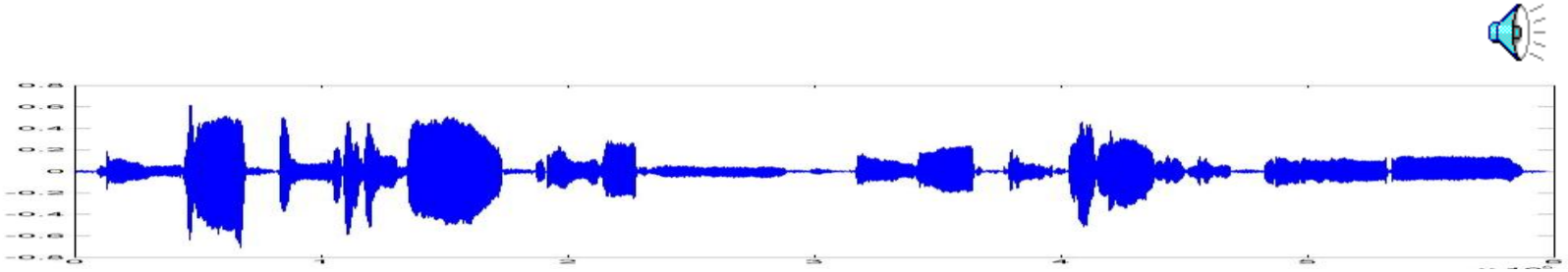
- Over the rainbow



- Low pitch



- Hi pitch



Phase Vocoder Pitch Shift

- Simple pitch shifting with phase vocoder not as effective as Psola
  - All details of the pitch get frequency shifted
  - We only want the basic pitch to change, not the details within the pitch
- More sophisticated techniques based on the phase vocoder modify the spectrum explicitly
  - Better than Psola
  - Can be applied to music

Some more examples




- What are we doing here?
  - Common sounds you hear everyday
    - Now you know how..