

HCS 7367  
Speech Perception Lab

Dr. Peter Assmann  
Fall 2009

## Filters and convolution

- Many applications in speech research require the use of digital filters. Filtering is defined any operation that changes the amplitude spectrum and/or phase spectrum of a signal.
- Anti-aliasing filters are low-pass filters that remove signal energy above  $f_s/2$  (one-half the sampling frequency).
- Matlab has two built-in filtering functions, `conv.m` and `filter.m`

## Filters and convolution

- The easiest way to filter speech in Matlab is to **convolve** the speech vector,  $x$ , with the filter vector,  $h$ , using the **conv** function:

»  $y = \text{conv}(x, h)$  ;

- Input: **speech vector**,  $x(n)$
- Input: filter **impulse response**,  $h(n)$
- Output: filtered speech vector,  $y(n)$

## Filters and convolution

- The function `conv.m` implements the mathematical operation of **convolution**, defined as:

$$y(n) = h(n) * x(n) = \sum_{m=-\infty}^{\infty} h(n-m)x(m)$$

## Filters and convolution

- A digital filter's output  $y(k)$  is related to its input  $x(k)$  by convolution with its **impulse response**,  $h(k)$ .

$$y(n) = h(n) * x(n) = \sum_{m=-\infty}^{\infty} h(n-m)x(m)$$

## Filters and convolution

- Use `conv.m` in digital filtering applications where the filter length,  $m$ , is finite, and the number of speech samples,  $n$ , is also finite

$$y(n) = h(n) * x(n) = \sum_{m=-\infty}^{\infty} h(n-m)x(m)$$

## Filters and convolution

- Convolution involves the following steps:
  - (1) reverse the impulse response in time;
  - (2) sum the cross-products of filter and signal to generate a sample of the output;
  - (3) shift the filter by one sample and repeat step 2. Continue until signal and filter no longer overlap.

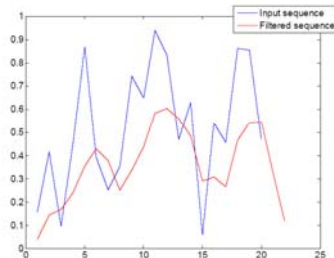
$$y(n) = h(n) * x(n) = \sum_{m=-\infty}^{\infty} h(n-m)x(m)$$

## Filters and convolution

- Signal  $x(k)$ 
  - Generate vector of 20 random numbers:
    - $x = \text{rand}(20, 1)$ ;
- Filter impulse response  $h(k)$ 
  - Create a 3-point moving average filter
    - $h = [1 \ 1 \ 1]/4$ ;
    - $y = \text{conv}(h, x)$ ;

## Filters and convolution

- `plot(x, 'b')`;
- `hold on`;
- `plot(y, 'r')`;



## Filtering and convolution

- Another way to describe the operation of filtering is in terms of the  $z$ -transform:

$$Y(z) = H(z)X(z) = \frac{b(1) + b(2)z^{-1} + \dots + b(nb+1)z^{-nb}}{a(1) + a(2)z^{-1} + \dots + a(na+1)z^{-na}} X(z)$$

- where  $X(z)$  is the  $z$ -transform of the speech signal,
- $H(z)$  is the **transfer function** of the filter, and
- $Y(z)$  is the  $z$ -transform of the filtered speech.

## Filtering and convolution

- The constants  $b(i)$  and  $a(i)$  are the filter coefficients; the **order** of the filter is the larger of  $nb$  and  $na$ .

$$Y(z) = H(z)X(z) = \frac{b(1) + b(2)z^{-1} + \dots + b(nb+1)z^{-nb}}{a(1) + a(2)z^{-1} + \dots + a(na+1)z^{-na}} X(z)$$

## Filtering and convolution

- When  $nb = 0$  ( $b$  is a scalar) the filter is an **Infinite Impulse Response (IIR)**, all-pole, recursive, or autoregressive (**AR**) filter.
- When  $na = 0$  ( $a$  is a scalar) the filter is a **Finite Impulse Response (FIR)**, all-zero, non-recursive, or moving average (**MA**) filter.
- When both  $na > 0$  and  $nb > 0$ , the filter is an IIR, pole-zero, recursive, or autoregressive moving average (**ARMA**) filter.

$$Y(z) = H(z)X(z) = \frac{b(1) + b(2)z^{-1} + \dots + b(nb+1)z^{-nb}}{a(1) + a(2)z^{-1} + \dots + a(na+1)z^{-na}} X(z)$$

## Filtering and convolution

- The z-transform expressed as a difference equation:

$$y(n) = b_1x(n) + b_2x(n-1) + \dots + b_{mb+1}x(n-nb) - a_2y(n-1) - \dots - a_{na+1}y(n-na)$$

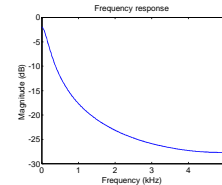
- The output samples of y are given by:

$$\begin{aligned} y(1) &= b_1x(1) \\ y(2) &= b_1x(2) + b_2x(1) - a_2y(1) \\ y(3) &= b_1x(3) + b_2x(2) + b_2x(1) - a_2y(2) - a_3y(1) \\ &\dots \end{aligned}$$

## Filtering and convolution

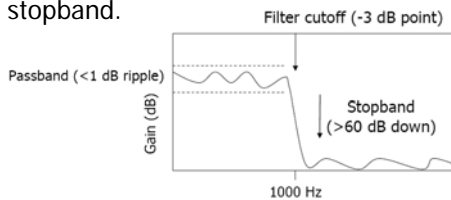
- Digital filters of this form can be implemented using the function `filter.m`
- Example:** a simple low-pass filter

- » `b = 1;`
- » `a = [1 -0.9]`
- » `x = [1; zeros(1023,1)];`
- » `y = filter(b,a,x);`



## Filter design

- Example:** design a 5-th order elliptical low-pass filter with a cutoff of 1000 Hz, no more than 1 dB of ripple in the passband, and at least 60 dB attenuation in the stopband.



## Elliptic filter

- » `help ellip`

### ELLIP Elliptic or Caueer digital and analog filter design.

[B,A] = ELLIP(N,Rp,Rs,Wn) designs an Nth order lowpass digital elliptic filter with Rp decibels of ripple in the passband and a stopband Rs decibels down. ELLIP returns the filter coefficients in length N+1 vectors B (numerator) and A (denominator).

The cut-off frequency Wn must be  $0.0 < Wn < 1.0$ , with 1.0 corresponding to half the sample rate. Use Rp = 0.5 and Rs = 20 as starting points, if you are unsure about choosing them.

If Wn is a two-element vector, Wn = [W1 W2], ELLIP returns an order 2N bandpass filter with passband  $W1 < W < W2$ .

[B,A] = ELLIP(N,Rp,Rs,Wn,'high') designs a highpass filter.

[B,A] = ELLIP(N,Rp,Rs,Wn,'stop') is a bandstop filter if Wn = [W1 W2].

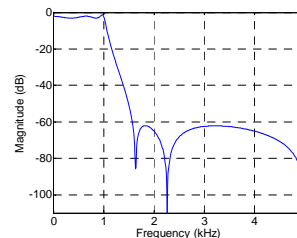
## Elliptic filter

- Example:** design a 5-th order elliptical low-pass filter with no more than 1 dB of ripple in the passband, and at least 60 dB attenuation in the stopband.

- » `N=5;`
- » `Rp=1;` % # of decibels of ripple in the passband
- » `Rs=60;` % # of decibels of attenuation in stopband
- » `Wn=1000 / (rate/2);` % normalized filter cutoff frequency
- » `[b ,a] = ellip(N,Rp,Rs,Wn);`
- » `x=[1; zeros(1023,1)];`
- » `y = filter (b,a,x);`

## Elliptic filter

- Example:** design a 5-th order elliptical low-pass filter with no more than 1 dB of ripple in the passband, and at least 60 dB attenuation in the stopband.



## Exercise: LP, HP, and BP filters

1. Design a 5-th order elliptical **low-pass** filter with 1000 Hz cutoff, no more than 1 dB of ripple in the passband, and at least 60 dB attenuation in the stopband.
2. Design a **high-pass** filter with these same parameters.
3. Design a **band-pass** filter centered at 1000 Hz with a 500-Hz bandwidth.
4. Filter the vowel /ae/ from your dataset. Plot the waveform and spectrum and listen to the filtered vowel.

## Long-term average speech spectrum

>> help ltass

LTASS: Computes long-term average speech spectrum via FFT.

Usage: mag=ltass(w,rate,twin,thop,nfft);

w: input waveform

rate: sample rate in Hz (default 10000 Hz)

twin: frame length in ms (default: 10 ms)

thop: frame update in ms (default: 5 ms)

nfft: FFT window length (default: 256 samples)

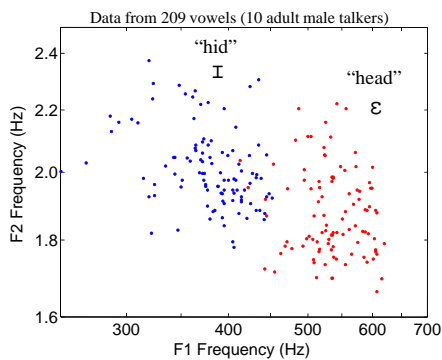
## Long-term average speech spectrum

- Compute and plot the LTASS of the vowel /ae/
  - twin=10;
  - thop=5;
  - nfft=256;
  - mag=ltass(y,rate,twin,thop,nfft);
  - freq=linspace(0,rate/2,length(mag));
  - plot(freq,mag);

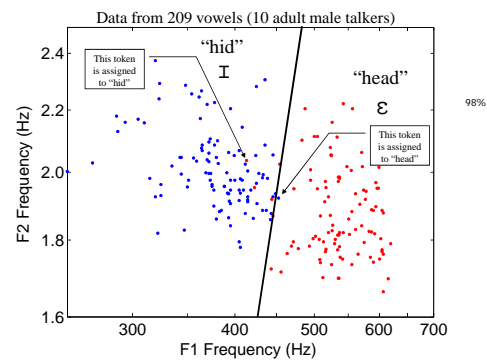
## Vowel space

- An incoming vowel token is projected in the  $N$ -dimensional **vowel space** defined by the measurements.
- Find a linear boundary in the vowel space that divides the space into separate regions, one for each vowel category.

## Vowel classification: 2 categories



## Vowel classification: 2 categories



## Distance measures

- Because the same vowel spoken by different talkers or in different contexts vary in their acoustic properties, we need to develop a measure of **distance** between points in the vowel space.

## Distance measures

### ■ Euclidean distance

- The distance  $d$  between points  $p$  and  $q$  in a 2-dimensional space is given by

$$d = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$$

## Exercise

- Compute Euclidean distances between your measured vowel samples and the formant means for N. Central Texas:

[http://www.utdallas.edu/~assmann/hcd7367/formant\\_data.html](http://www.utdallas.edu/~assmann/hcd7367/formant_data.html)

- Assign each token to the category for which distance is smallest.
- Analyze the pattern of errors.

## Formant frequencies

*Formant estimates for talker XYZ:*

	F1	F2	F3
heed	250	2250	3050
hid	341	1954	2984
hayed	386	1944	2877
head	422	1849	2744

Use Matlab to construct two matrices: one for your formant frequencies, the other for the N. Central Texas vowel database. Estimate F1, F2, F3 at **20%** of the vowel's duration.

## Euclidean distances

	heed	hid	hayed	head
heed	56	315	455	619
hid	231	32	250	366
hayed	314	201	29	232
head	430	324	178	11

Write a Matlab program to compute the Euclidean distance between the two sets of formant frequencies.

## Minimum distance classifier

	heed	hid	hayed	head
heed	1	0	0	0
hid	0	1	0	0
hayed	0	0	1	0
head	0	0	0	1

Based on the previous slide's distances, place a 1 in the column with the *smallest distance* and 0's in all the other columns. Repeat for each row (i.e. for each of the 12 vowels). Compute classification accuracy by averaging the on-diagonal elements.

## Category assignment

- Assign each token in the test set to the category in the training set it is most likely to represent.
  - Compute the parameter vector for the unknown token.
  - Compare it to the parameter vectors for each category (type) in the training set.
  - Compute the *distance* (similarity) between the unknown token and all members of the category in the training set.
  - Assign the token to the category that yields the smallest distance.

## Hillenbrand vowel set

- Linear discriminant analysis

[http://www.utdallas.edu/~assmann/hcs7367/hillen\\_lda\\_class.html](http://www.utdallas.edu/~assmann/hcs7367/hillen_lda_class.html)