MATLAB

LPC

Utilities – Read audio file

Syntax

```
[y,Fs] = audioread(filename)
[y,Fs] = audioread(filename,samples)
```

```
[y,Fs] = audioread( __ ,dataType)
```

Description

`[y,Fs] = audioread(filename)` reads data from the file named `filename`, and returns sampled data, `y`, and a sample rate for that data, `Fs`.

`[y,Fs] = audioread(filename,samples)` reads the selected range of audio samples in the file, where `samples` is a vector of the form `[start,finish]`.

`[y,Fs] = audioread( __ ,dataType)` returns sampled data in the data range corresponding to the `dataType` of `'native'` or `'double'`, and can include any of the input arguments in previous syntaxes.

Utilities – Levinson-Durbin recursion

Syntax

```
a = levinson(r)
a = levinson(r,n)
[a,e] = levinson(r,n)
[a,e,k] = levinson(r,n)
```

Description

The Levinson-Durbin recursion is an algorithm for finding an all-pole IIR filter with a prescribed deterministic autocorrelation sequence. It has applications in filter design, coding, and spectral estimation. The filter that `levinson` produces is minimum phase.

`a = levinson(r)` finds the coefficients of a `length(r)-1` order autoregressive linear process which has `r` as its autocorrelation sequence. `r` is a real or complex deterministic autocorrelation sequence. If `r` is a matrix, `levinson` finds the coefficients for each column of `r` and returns them in the rows of `a`. `n=length(r)-1` is the default order of the denominator polynomial $A(z)$; that is, `a = [1 a(2) ... a(n+1)]`. The filter coefficients are ordered in descending powers of z^{-1}.

$$H(z) = \frac{1}{A(z)} = \frac{1}{1 + a(2)z^{-1} + \cdots + a(n+1)z^{-n}}$$

`a = levinson(r,n)` returns the coefficients for an autoregressive model of order n.

`[a,e] = levinson(r,n)` returns the prediction error, e, of order n.

`[a,e,k] = levinson(r,n)` returns the reflection coefficients k as a column vector of length n.

Utilities – Cross-correlation

Syntax

```
r = xcorr(x,y)
r = xcorr(x)
```

```
r = xcorr( __ ,maxlag)
r = xcorr( __ ,scaleopt)
```

```
[r,lags] = xcorr( __ )
```

Description

`r = xcorr(x,y)` returns the cross-correlation of two discrete-time sequences, x and y. Cross-correlation measures the similarity between x and shifted (lagged) copies of y as a function of the lag. If x and y have different lengths, the function appends zeros at the end of the shorter vector so it has the same length, N, as the other.

`r = xcorr(x)` returns the autocorrelation sequence of x. If x is a matrix, then `r` is a matrix whose columns contain the autocorrelation and cross-correlation sequences for all combinations of the columns of x.

`r = xcorr( __ ,maxlag)` limits the lag range from –`maxlag` to `maxlag`. This syntax accepts one or two input sequences. `maxlag` defaults to $N – 1$.

`r = xcorr( __ ,scaleopt)` additionally specifies a normalization option for the cross-correlation or autocorrelation. Any option other than `'none'` (the default) requires `x` and `y` to have the same length.

`[r,lags] = xcorr( __ )` also returns a vector with the lags at which the correlations are computed.

Utilities – Convolution

Syntax

```
w = conv(u,v)
w = conv(u,v,shape)
```

Description

`w = conv(u,v)` returns the convolution of vectors u and v. If u and v are vectors of polynomial coefficients, convolving them is equivalent to multiplying the two polynomials.

`w = conv(u,v,shape)` returns a subsection of the convolution, as specified by `shape`. For example, `conv(u,v,'same')` returns only the central part of the convolution, the same size as u, and `conv(u,v,'valid')` returns only the part of the convolution computed without the zero-padded edges.

Utilities – Filter frequency response

Syntax

```
[h,w] = freqz(b,a,n)
[h,w] = freqz(sos,n)
[h,w] = freqz(d,n)
[h,w] = freqz( __ ,n,'whole')
```

```
[h,f] = freqz( __ ,n,fs)
[h,f] = freqz( __ ,n,'whole',fs)
```

```
h = freqz( __ ,w)
h = freqz( __ ,f,fs)
```

```
freqz( __ )
```

Description

`[h,w] = freqz(b,a,n)` returns the n-point frequency response vector, h, and the corresponding angular frequency vector, w, for the digital filter with numerator and denominator polynomial coefficients stored in b and a, respectively.

`[h,w] = freqz(sos,n)` returns the n-point complex frequency response corresponding to the second-order sections matrix, `sos`.

`[h,w] = freqz(d,n)` returns the n-point complex frequency response for the digital filter, d.

`[h,w] = freqz( __ ,n,'whole')` returns the frequency response at n sample points around the entire unit circle.

`[h,f] = freqz( __ ,n,fs)` returns the frequency response vector, h, and the corresponding physical frequency vector, `f`, for the digital filter with numerator and denominator polynomial coefficients stored in b and a, respectively, given the sample rate, `fs`.

`[h,f] = freqz( __ ,n,'whole',fs)` returns the frequency at n points ranging between 0 and `fs`.

`h = freqz( __ ,w)` returns the frequency response vector, h, at the normalized frequencies supplied in w.

`h = freqz( __ ,f,fs)` returns the frequency response vector, h, at the physical frequencies supplied in `f`.

Utilities – Find peaks

Syntax

```
pks = findpeaks(data)
[pks,locs] = findpeaks(data)
[pks,locs,w,p] = findpeaks(data)
```

```
[__] = findpeaks(data,x)
[__] = findpeaks(data,Fs)
```

```
[__] = findpeaks(__,Name,Value)
```

```
findpeaks(__)
```

Description

`pks = findpeaks(data)` returns a vector with the local maxima (peaks) of the input signal vector, `data`. A *local peak* is a data sample that is either larger than its two neighboring samples or is equal to `Inf`. Non-`Inf` signal endpoints are excluded. If a peak is flat, the function returns only the point with the lowest index.

`[pks,locs] = findpeaks(data)` additionally returns the indices at which the peaks occur.

`[pks,locs,w,p] = findpeaks(data)` additionally returns the widths of the peaks as the vector `w` and the prominences of the peaks as the vector `p`.

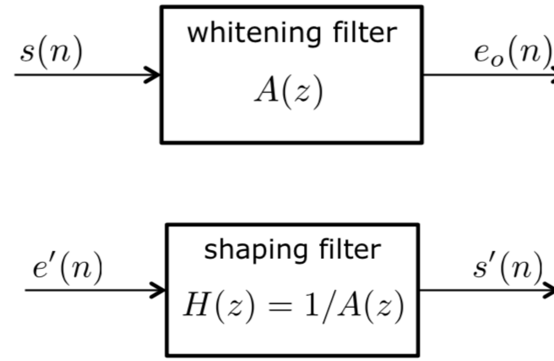
`[__] = findpeaks(data,x)` specifies `x` as the location vector and returns any of the output arguments from previous syntaxes. `locs` and `w` are expressed in terms of `x`.

`[__] = findpeaks(data,Fs)` specifies the sample rate, `Fs`, of the data. The first sample of `data` is assumed to have been taken at time zero. `locs` and `w` are converted to time units.

`[__] = findpeaks(__,Name,Value)` specifies options using name-value pair arguments in addition to any of the input arguments in previous syntaxes.

`findpeaks(__)` without output arguments plots the signal and overlays the peak values.

Exercise 1 – Pitch Detection



- Load the given audio signal
- Plot it in the time domain
- Define LPC and analysis parameters
  - Prediction order = 25
- Define windowing parameters
  - Window type = rectangular
  - Analysis window length = 25 ms
- For each one of the first N=10 windows of the signal do
  - Window the signal (*s(n)*)
  - Compute auto-correlation
  - Estimate whitening filter through Levinson-Durbin recursion
  - Compute the prediction error by whitening the signal (*e(n)*)
  - Find periodic peaks of the residual error
  - Compute the frequency response of the shaping filter (*H(f)*)
  - Plot the DFT magnitude of the windowed signal
  - Plot the frequency response of the shaping filter on the same axis
  - Plot the prediction error in the time domain
  - Plot the prediction error in the frequency domain

Exercise 2 – Cross Synthesis

- Load the guitar (excitation) and the voice (envelope) signals
- Define Hanning window of 400 samples
- Define LPC parameters
  - Order 20 for the envelope estimation
  - Order 9 for the excitation estimation
- Trim both audio signals to the same lenght
- Using a hop size of 200 samples, loop over each window of the two signals in an overlap-and-add fashion and to
  - Window both signals
  - Estimate LPC shaping filter from the voice
  - Estimate LPC residual from the guitar
  - Apply guitar residual to voice signal
- Listen to the synthesized signal