API Documentation

API Documentation

April 19, 2017

Contents

1 Module constructs

1.1 Class Wave

??-1

constructs.Wave

Represents a discrete-time signal.

1.1.1 Methods

__init__(*self*, *_typ*, *_name*, *_len*, *_var*=`None`, *_const*=`''`)

Initialize the signal. **Parameters**

- `_typ:` type of the signal samples - Double or Complex.
- `_name:` name used for the signal during code generation.
- `_len:` number of samples in the signal.
- `_var:` (Default value = None)
- `_const:` (Default value = ")

length(*self*)

Number of samples in the signal.

__len__(*self*)

type(*self*)

Type of the signal samples - Double or Complex.

variables(*self*)

isInput(*self*)

Whether this is an input signal, or has been computed.

__str__(*self*)

clone(*self*)

Clone the signal. **Return Value**
A deep copy of the signal.

__add__(*self*, *other*)

Add two signals elementwise. **Parameters**
`other:` signal to be added to this one.

Return Value
Sum of the 2 signals.

add(*wav1*, *wav2*, *out_len*)

convolve(*self*, *other*, *out_name*)

Convolve two signals. **Parameters**
`other:` signal to convolved with this one.

`out_name:` name of the output signal.

Return Value
Discrete linear convolution of the 2 signals.

correlate(*self*, *other*, *out_name*)

Cross-correlate two signals. **Parameters**
`other:` signal to correlated with this one.

`out_name:` name of the output signal.

Return Value
Discrete linear cross-correlation of the 2 signals.

fftconvolve(*self*, *other*, *out_name*)

Convolve two signals using FFT. **Parameters**
`other:` signal to convolved with this one.

`out_name:` name of the output signal.

Return Value
Discrete linear convolution of the 2 signals.

lfilter(*self*, *b*, *a*, *out_name*)

Filter signal with an IIR or FIR filter. **Parameters**

- `b`: the numerator coefficient vector.
- `a`: the denominator coefficient vector. Both a and b are normalized by a[0] to make a[0] equal to 1.
- `out_name`: name of the output signal.

Return Value

The output of the digital filter.

lfilter_fir(*self*, *b*, *out_name*)

Filter signal with an FIR filter. **Parameters**

- `b`: the filter coefficient vector.
- `out_name`: name of the output signal.

Return Value

The output of the digital filter.

lfilter_fir_and_delay(*self*, *b*, *out_name*, *_out_typ*=`None`)

Filter signal with an FIR filter and compensate for the delay introduced by filtering.
Parameters

- `b`: the filter coefficient vector.
- `out_name`: name of the output signal.
- `_out_typ`: type of the output signal samples. (Default value = None)

Return Value

The output of the digital filter compensated for delay.

lfilter_fir_and_delays(*wav*, *b*, *out_name*, *_out_typ*, *L*)

hilbert(*self*, *out_name*)

Compute the analytic signal, using the Hilbert transform. **Parameters**

- `out_name`: name of the output signal.

Return Value

Analytic signal of this signal.

upfirdn(*self*, *h*, *out_name*, *up*=1, *down*=1)

Upsample, FIR filter, and downsample. **Parameters**

`h:` FIR (finite-impulse response) filter coefficients.

`out_name:` name of the output signal.

`up:` upsampling rate. (Default value = 1)

`down:` downsampling rate. (Default value = 1)

Return Value

The output signal with size changed with respect to this signal based on the h, up, and down parameters.

downsample(*self*, *down*, *out_name*)

Downsample the signal. **Parameters**

`down:` the downsampling factor.

`out_name:` name of the output signal.

Return Value

The down-sampled signal.

downsamples(*wav*, *down*, *out_name*, *N*)

upsample(*self*, *up*, *out_name*, *_out_len*=None)

Upsample the signal. **Parameters**

`up:` the upsampling factor.

`out_name:` name of the output signal.

`_out_len:` number of samples in the output signal. (Default value = None)

Return Value

The up-sampled signal.

low_pass(*self*, *cutoff*, *out_name*, *factor*=0)

Attenuate frequencies above the cutoff. **Parameters**

`cutoff:` frequency in Hz.

`out_name:` name of the output signal.

`factor:` what to multiply the magnitude by. (Default value = 0)

Return Value

Signal with magnitudes of frequencies above cutoff scaled by factor.

high_pass(*self*, *cutoff*, *out_name*, *factor*=`0`)

Attenuate frequencies below the cutoff. **Parameters**

- `cutoff:` frequency in Hz.
- `out_name:` name of the output signal.
- `factor:` what to multiply the magnitude by. (Default value = 0)

Return Value

Signal with magnitudes of frequencies below cutoff scaled by factor.

band_stop(*self*, *low_cutoff*, *high_cutoff*, *out_name*, *factor*=`0`)

Attenuate frequencies between the cutoffs. **Parameters**

- `low_cutoff:` frequency in Hz.
- `high_cutoff:` frequency in Hz.
- `out_name:` name of the output signal.
- `factor:` what to multiply the magnitude by. (Default value = 0)

Return Value

Signal with magnitudes of frequencies between low_cutoff and high_cutoff scaled by factor.

freqz(*self*, *out_names*)

Compute the frequency response of a digital filter. **Parameters**

- `out_names:` tuple of 2 strings containing the names of output vectors.

Return Value

The normalized frequencies at which the frequency response was computed, in radians/sample; the frequency response, as complex numbers.

interp_fft(*self*, *r*, *out_name*, *_out_typ*=`None`)

Upsample the signal using Fourier method. **Parameters**

- `r:` the upsampling factor.
- `out_name:` name of the output signal.
- `_out_typ:` type of the output signal samples. (Default value = None)

Return Value

The up-sampled signal.

interps_fft(*wav*, *r*, *out_name*, *_out_typ*, *N*)

freq_shift(*self*, *shift*, *sample*, *out_name*)

Apply a frequency shift to the signal. **Parameters**

`shift:` frequency in Hz.

`sample:` sampling frequency of the signal.

`out_name:` name of the output signal.

Return Value

Signal with frequency shift applied to it.

subset(*self*, *start*, *stop*, *out_name*)

Take a subset of the signal. **Parameters**

`start:` first index of the subset in the signal.

`stop:` last index of the subset in the signal.

`out_name:` name of the output signal.

Return Value

Truncated signal of length stop - start + 1.

scalar_mul(*self*, *sval*, *out_name*)

Multiply the elements of the signal by a scalar value. **Parameters**

`sval:` scalar value to multiply.

`out_name:` name of the output signal.

Return Value

Signal with amplitude scaled by a factor of sval.

get_window(*cls*, *window*, *N*, *out_name*)

Return a window. **Parameters**

`window:` the type of window to create

`N:` the number of samples in the window

`out_name:` name of the output window

Return Value

A window of length N and type window.

firwin(*cls*, *N*, *cutoff*, *out_name*, *window*=`'hamming'`, *pass_zero*=`True`)

FIR filter design using the window method. **Parameters**

`N:` length of the filter (number of coefficients).

`cutoff:` cutoff frequency of filter OR a tuple of 2 cutoff frequencies (that is, band edges).

`out_name:` name of the output filter coefficient vector.

`window:` desired window to use. (Default value = 'hamming')

`pass_zero:` if True, the gain at the frequency 0 (i.e. the 'DC gain') is 1. Otherwise the DC gain is 0. (Default value = True)

Return Value

Coefficients of length N FIR filter.

kaiserord(*cls*, *ripple*, *width*)

Design a Kaiser window to limit ripple and width of transition region. **Parameters**

`ripple:` positive number specifying maximum ripple in passband (dB) and minimum ripple in stopband.

`width:` width of transition region (normalized so that 1 corresponds to pi radians / sample).

Return Value

The length of the kaiser window; the beta parameter for the kaiser window.

kaiser_beta(*cls*, *a*)

Compute the Kaiser parameter beta, given the attenuation a. **Parameters**

`a:` the desired attenuation in the stopband and maximum ripple in the passband, in dB. This should be positive.

Return Value

The beta parameter to be used in the formula for a Kaiser window.

fftfreq(*cls*, *n*, *out_name*, *real_input*=`True`)

Return the Discrete Fourier Transform sample frequencies. **Parameters**

`n:` window length

`out_name:` name of the output frequency vector

`real_input:` whether the input is real valued (Default value = True)

Return Value

Wave of length n containing the sample frequencies.

fft(*self*, *out_name*)

Compute the discrete Fourier Transform. **Parameters**

`out_name`: name of the output signal.

Return Value

The DFT of this signal.

ffts(*wav*, *out_name*, *N*)

ifft(*self*, *out_name*, *_out_len*=`None`, *real_input*=`True`)

Compute the inverse discrete Fourier Transform. The computed transform is unnormalized. The returned signal must be divided by its length if the normalized transform is required.

Parameters

`out_name`: name of the output signal.

`_out_len`: number of samples in the output signal. (Default value = None)

`real_input`: whether the input is real valued. (Default value = True)

Return Value

The IDFT of this signal.

Index