stft Documentation

Release 0.4.7

International AudioLabs Erlangen

March 31, 2016

Contents

This is a package for calculating short time fourier transforms with NumPy.

Contents:

CHAPTER 1

Installation

This package can be installed using PIP:

```
pip install stft
```

CHAPTER 2

Examples

2.1 Simple Example

Loading a file and calculating the spectrogram.

```
import stft
import scipy.io.wavfile as wav

fs, audio = wav.read('input.wav')
specgram = stft.spectrogram(audio)
```

See also:

module *stft.spectrogram()*

2.2 Back and Forth Example

Loading a file and calculating the spectrogram, its inverse and saving the result.

```
import stft
import scipy.io.wavfile as wav

fs, audio = wav.read('input.wav')
specgram = stft.spectrogram(audio)
output = stft.ispectrogram(specgram)
wav.write('output.wav', fs, output)
```

See also:

modules *stft.spectrogram()* *stft.ispectrogram()*

2.3 Passing multiple transfer functions

stft.spectrogram() and *stft.ispectrogram()* allow passing multiple transform functions as a list.

STFT will pick each transform for each frame it processes, the list of transforms will be extended indefinitely for as long as many frames need to be processed.

```
import stft
import scipy.io.wavfile as wav

fs, audio = wav.read('input.wav')
specgram = stft.spectrogram(audio, transform=[scipy.fftpack.fft, numpy.fft.fft])
output = stft.ispectrogram(specgram, transform=[scipy.fftpack.ifft, numpy.fft.ifft])
wav.write('output.wav', fs, output)
```

In this case, each frame will be processed using `scipy.fftpack.fft`, then `numpy.fft.fft`, then `scipy.fftpack.fft` again etc.

2.4 Saving Settings Example

You do not need to pass the same settings to *`stft.spectrogram()`* and *`stft.ispectrogram()`* twice as the settings are saved in the array itself.

```
import stft
import scipy.io.wavfile as wav

fs, audio = wav.read('input.wav')
specgram = stft.spectrogram(audio, framelength=512, overlap=4)
output = stft.ispectrogram(specgram)
wav.write('output.wav', fs, output)
```

See also:

modules *`stft.spectrogram()`* *`stft.ispectrogram()`* `stft.types.SpectogramArray`

CHAPTER 3

Modules

3.1 stft package

3.1.1 Module contents

stft.**spectrogram**(*data*, *framelength=1024*, *hopsize=None*, *overlap=None*, *centered=True*, *window=None*, *halved=True*, *transform=None*, *padding=0*, *save_settings=True*)

Calculate the spectrogram of a signal

Parameters

- **data** (*array_like*) – The signal to be transformed. May be a 1D vector for single channel or a 2D matrix for multi channel data. In case of a mono signal, the data is must be a 1D vector of length `samples`. In case of a multi channel signal, the data must be in the shape of `samples x channels`.
- **framelength** (*int*) – The signal frame length. Defaults to `1024`.
- **hopsize** (*int*) – The signal frame hopsize. Defaults to `None`. Setting this value will override `overlap`.
- **overlap** (*int*) – The signal frame overlap coefficient. Value `x` means `1/x` overlap. Defaults to `2`.
- **centered** (*boolean*) – Pad input signal so that the first and last window are centered around the beginning of the signal. Defaults to true.
- **window** (*callable, array_like*) – Window to be used for deringing. Can be `False` to disable windowing. Defaults to `scipy.signal.cosine`.
- **halved** (*boolean*) – Switch for turning on signal truncation. For real signals, the fourier transform of real signals returns a symmetrically mirrored spectrum. This additional data is not needed and can be removed. Defaults to `True`.
- **transform** (*callable*) – The transform to be used. Defaults to `scipy.fftpack.fft`.
- **padding** (*int*) – Zero-pad signal with x times the number of samples.
- **save_settings** (*boolean*) – Save settings used here in attribute `out.stft_settings` so that *ispectrogram()* can infer these settings without the developer having to pass them again.

Returns **data** – The spectrogram (or tensor of spectograms) In case of a mono signal, the data is formatted as `bins x frames`. In case of a multi channel signal, the data is formatted as `bins x frames x channels`.

Return type array_like

Notes

The data will be padded to be a multiple of the desired FFT length.

See also:

stft.stft.process() The function used to transform the data

stft.**ispectrogram**(*data*, *framelength=None*, *hopsize=None*, *overlap=None*, *centered=None*, *window=None*, *halved=None*, *transform=None*, *padding=None*, *outlength=None*)

Calculate the inverse spectrogram of a signal

Parameters

- **data** (*array_like*) – The spectrogram to be inverted. May be a 2D matrix for single channel or a 3D tensor for multi channel data. In case of a mono signal, the data must be in the shape of `bins x frames`. In case of a multi channel signal, the data must be in the shape of `bins x frames x channels`.
- **framelength** (*int*) – The signal frame length. Defaults to infer from data.
- **hopsize** (*int*) – The signal frame hopsize. Defaults to infer from data. Setting this value will override `overlap`.
- **overlap** (*int*) – The signal frame overlap coefficient. Value `x` means `1/x` overlap. Defaults to infer from data.
- **centered** (*boolean*) – Pad input signal so that the first and last window are centered around the beginning of the signal. Defaults to to infer from data.
- **window** (*callable, array_like*) – Window to be used for deringing. Can be `False` to disable windowing. Defaults to to infer from data.
- **halved** (*boolean*) – Switch to reconstruct the other halve of the spectrum if the forward transform has been truncated. Defaults to to infer from data.
- **transform** (*callable*) – The transform to be used. Defaults to infer from data.
- **padding** (*int*) – Zero-pad signal with x times the number of samples. Defaults to infer from data.
- **outlength** (*int*) – Crop output signal to length. Useful when input length of spectrogram did not fit into framelength and input data had to be padded. Not setting this value will disable cropping, the output data may be longer than expected.

Returns **data** – The signal (or matrix of signals). In case of a mono output signal, the data is formatted as a 1D vector of length `samples`. In case of a multi channel output signal, the data is formatted as `samples x channels`.

Return type array_like

Notes

By default *spectrogram()* saves its transformation parameters in the output array. This data is used to infer the transform parameters here. Any aspect of the settings can be overridden by passing the according parameter to this function.

During transform the data will be padded to be a multiple of the desired FFT length. Hence, the result of the inverse transform might be longer than the input signal. However it is safe to remove the additional data, e.g. by using

```
output.resize(input.shape)
```

where `input` is the input of `stft.spectrogram()` and `output` is the output of `stft.ispectrogram()`

See also:

stft.stft.iprocess() The function used to transform the data

Module to transform signals

`stft.stft.`**`cosine`**(*M*)
Gernerate a halfcosine window of given length

Uses `scipy.signal.cosine` by default. However since this window function has only recently been merged into mainline SciPy, a fallback calculation is in place.

Parameters **M** (*int*) – Length of the window.

Returns **data** – The window function

Return type array_like

`stft.stft.`**`iprocess`**(*data*, *window*, *halved*, *transform*, *padding*)
Calculate the inverse short time fourier transform of a spectrum

Parameters

- **data** (*array_like*) – The spectrum to be calculated. Must be a 1D array.
- **window** (*array_like*) – Tapering window
- **halved** (*boolean*) – Switch for turning on signal truncation. For real output signals, the inverse fourier transform consumes a symmetrically mirrored spectrum. This additional data is not needed and can be removed. Setting this value to `True` will automatically create a mirrored spectrum.
- **transform** (*callable*) – The transform to be used.
- **padding** (*int*) – Signal before FFT transform was padded with x zeros.

Returns **data** – The signal

Return type array_like

`stft.stft.`**`ispectrogram`**(*data*, *framelength=None*, *hopsize=None*, *overlap=None*, *centered=None*, *window=None*, *halved=None*, *transform=None*, *padding=None*, *outlength=None*)
Calculate the inverse spectrogram of a signal

Parameters

- **data** (*array_like*) – The spectrogram to be inverted. May be a 2D matrix for single channel or a 3D tensor for multi channel data. In case of a mono signal, the data must be in the shape of `bins x frames`. In case of a multi channel signal, the data must be in the shape of `bins x frames x channels`.
- **framelength** (*int*) – The signal frame length. Defaults to infer from data.
- **hopsize** (*int*) – The signal frame hopsize. Defaults to infer from data. Setting this value will override `overlap`.
- **overlap** (*int*) – The signal frame overlap coefficient. Value `x` means `1/x` overlap. Defaults to infer from data.
- **centered** (*boolean*) – Pad input signal so that the first and last window are centered around the beginning of the signal. Defaults to to infer from data.
- **window** (*callable, array_like*) – Window to be used for deringing. Can be `False` to disable windowing. Defaults to to infer from data.
- **halved** (*boolean*) – Switch to reconstruct the other halve of the spectrum if the forward transform has been truncated. Defaults to to infer from data.
- **transform** (*callable*) – The transform to be used. Defaults to infer from data.
- **padding** (*int*) – Zero-pad signal with x times the number of samples. Defaults to infer from data.
- **outlength** (*int*) – Crop output signal to length. Useful when input length of spectrogram did not fit into framelength and input data had to be padded. Not setting this value will disable cropping, the output data may be longer than expected.

Returns **data** – The signal (or matrix of signals). In case of a mono output signal, the data is formatted as a 1D vector of length `samples`. In case of a multi channel output signal, the data is formatted as `samples x channels`.

Return type array_like

Notes

By default *spectrogram()* saves its transformation parameters in the output array. This data is used to infer the transform parameters here. Any aspect of the settings can be overridden by passing the according parameter to this function.

During transform the data will be padded to be a multiple of the desired FFT length. Hence, the result of the inverse transform might be longer than the input signal. However it is safe to remove the additional data, e.g. by using

```
output.resize(input.shape)
```

where `input` is the input of `stft.spectrogram()` and `output` is the output of `stft.ispectrogram()`

See also:

stft.stft.iprocess() The function used to transform the data

stft.stft.**process**(*data*, *window*, *halved*, *transform*, *padding*)
Calculate a windowed transform of a signal

Parameters

- **data** (*array_like*) – The signal to be calculated. Must be a 1D array.
- **window** (*array_like*) – Tapering window
- **halved** (*boolean*) – Switch for turning on signal truncation. For real signals, the fourier transform of real signals returns a symmetrically mirrored spectrum. This additional data is not needed and can be removed.
- **transform** (*callable*) – The transform to be used.
- **padding** (*int*) – Zero-pad signal with x times the number of samples.

Returns **data** – The spectrum

Return type array_like

stft.stft.**spectrogram**(*data*, *framelength=1024*, *hopsize=None*, *overlap=None*, *centered=True*, *window=None*, *halved=True*, *transform=None*, *padding=0*, *save_settings=True*)

Calculate the spectrogram of a signal

Parameters

- **data** (*array_like*) – The signal to be transformed. May be a 1D vector for single channel or a 2D matrix for multi channel data. In case of a mono signal, the data is must be a 1D vector of length `samples`. In case of a multi channel signal, the data must be in the shape of `samples x channels`.
- **framelength** (*int*) – The signal frame length. Defaults to `1024`.
- **hopsize** (*int*) – The signal frame hopsize. Defaults to `None`. Setting this value will override `overlap`.
- **overlap** (*int*) – The signal frame overlap coefficient. Value `x` means `1/x` overlap. Defaults to `2`.
- **centered** (*boolean*) – Pad input signal so that the first and last window are centered around the beginning of the signal. Defaults to true.
- **window** (*callable, array_like*) – Window to be used for deringing. Can be `False` to disable windowing. Defaults to `scipy.signal.cosine`.
- **halved** (*boolean*) – Switch for turning on signal truncation. For real signals, the fourier transform of real signals returns a symmetrically mirrored spectrum. This additional data is not needed and can be removed. Defaults to `True`.
- **transform** (*callable*) – The transform to be used. Defaults to `scipy.fftpack.fft`.
- **padding** (*int*) – Zero-pad signal with x times the number of samples.
- **save_settings** (*boolean*) – Save settings used here in attribute `out.stft_settings` so that *ispectrogram()* can infer these settings without the developer having to pass them again.

Returns **data** – The spectrogram (or tensor of spectograms) In case of a mono signal, the data is formatted as `bins x frames`. In case of a multi channel signal, the data is formatted as `bins x frames x channels`.

Return type array_like

Notes

The data will be padded to be a multiple of the desired FFT length.

See also:

stft.stft.process() The function used to transform the data

3.2 stft types module

3.2.1 Module contents

class stft.types.**SpectrogramArray**
Bases: numpy.ndarray

NumpyArray with additional stft_settings attribute for saving stft-specific settings.

Attributes

T	Same as self.transpose(), except that self is returned if self.ndim < 2.
base	Base object if memory is from some other object.
ctypes	An object to simplify the interaction of the array with the ctypes module.
data	Python buffer object pointing to the start of the array's data.
dtype	Data-type of the array's elements.
flags	Information about the memory layout of the array.
flat	A 1-D iterator over the array.
imag	The imaginary part of the array.
itemsize	Length of one array element in bytes.
nbytes	Total bytes consumed by the elements of the array.
ndim	Number of array dimensions.
real	The real part of the array.
shape	Tuple of array dimensions.
size	Number of elements in the array.
strides	Tuple of bytes to step in each dimension when traversing an array.

Methods

all([axis, out])	Returns True if all elements evaluate to True.
any([axis, out])	Returns True if any of the elements of *a* evaluate to True.
argmax([axis, out])	Return indices of the maximum values along the given axis.
argmin([axis, out])	Return indices of the minimum values along the given axis of *a*.
argpartition(kth[, axis, kind, order])	Returns the indices that would partition this array.
argsort([axis, kind, order])	Returns the indices that would sort this array.
astype(dtype[, order, casting, subok, copy])	Copy of the array, cast to a specified type.
byteswap(inplace)	Swap the bytes of the array elements
choose(choices[, out, mode])	Use an index array to construct a new array from a set of choices.
clip(a_min, a_max[, out])	Return an array whose values are limited to [a_min, a_max].
compress(condition[, axis, out])	Return selected slices of this array along given axis.
conj()	Complex-conjugate all elements.
conjugate()	Return the complex conjugate, element-wise.
copy([order])	Return a copy of the array.
cumprod([axis, dtype, out])	Return the cumulative product of the elements along the given axis.

Table 3.2 – continued from previous page

cumsum([axis, dtype, out])	Return the cumulative sum of the elements along the given axis.
diagonal([offset, axis1, axis2])	Return specified diagonals.
dot(b[, out])	Dot product of two arrays.
dump(file)	Dump a pickle of the array to the specified file.
dumps()	Returns the pickle of the array as a string.
fill(value)	Fill the array with a scalar value.
flatten([order])	Return a copy of the array collapsed into one dimension.
getfield(dtype[, offset])	Returns a field of the given array as a certain type.
item(*args)	Copy an element of an array to a standard Python scalar and return it.
itemset(*args)	Insert scalar into an array (scalar is cast to array's dtype, if possible)
max([axis, out])	Return the maximum along a given axis.
mean([axis, dtype, out])	Returns the average of the array elements along given axis.
min([axis, out])	Return the minimum along a given axis.
newbyteorder([new_order])	Return the array with the same data viewed with a different byte order.
nonzero()	Return the indices of the elements that are non-zero.
partition(kth[, axis, kind, order])	Rearranges the elements in the array in such a way that value of the element in kth po
prod([axis, dtype, out])	Return the product of the array elements over the given axis
ptp([axis, out])	Peak to peak (maximum - minimum) value along a given axis.
put(indices, values[, mode])	Set `a.flat[n] = values[n]` for all *n* in indices.
ravel([order])	Return a flattened array.
repeat(repeats[, axis])	Repeat elements of an array.
reshape(shape[, order])	Returns an array containing the same data with a new shape.
resize(new_shape[, refcheck])	Change shape and size of array in-place.
round([decimals, out])	Return *a* with each element rounded to the given number of decimals.
searchsorted(v[, side, sorter])	Find indices where elements of v should be inserted in a to maintain order.
setfield(val, dtype[, offset])	Put a value into a specified place in a field defined by a data-type.
setflags([write, align, uic])	Set array flags WRITEABLE, ALIGNED, and UPDATEIFCOPY, respectively.
sort([axis, kind, order])	Sort an array, in-place.
squeeze([axis])	Remove single-dimensional entries from the shape of *a*.
std([axis, dtype, out, ddof])	Returns the standard deviation of the array elements along given axis.
sum([axis, dtype, out])	Return the sum of the array elements over the given axis.
swapaxes(axis1, axis2)	Return a view of the array with *axis1* and *axis2* interchanged.
take(indices[, axis, out, mode])	Return an array formed from the elements of *a* at the given indices.
tofile(fid[, sep, format])	Write array to a file as text or binary (default).
tolist()	Return the array as a (possibly nested) list.
tostring([order])	Construct a Python string containing the raw data bytes in the array.
trace([offset, axis1, axis2, dtype, out])	Return the sum along diagonals of the array.
transpose(*axes)	Returns a view of the array with axes transposed.
var([axis, dtype, out, ddof])	Returns the variance of the array elements, along given axis.
view([dtype, type])	New view of array with the same data.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

S

C

I

P

S