librosa.util.sync

`librosa.util.sync`(*data*, *idx*, *aggregate=None*, *pad=True*, *axis=-1*)[source]

Synchronous aggregation of a multi-dimensional array between boundaries

Note

In order to ensure total coverage, boundary points may be added to *idx*.

If synchronizing a feature matrix against beat tracker output, ensure that frame index numbers are properly aligned and use the same hop length.

Parameters:

data : np.ndarray

multi-dimensional array of features

idx : iterable of ints or slices

Either an ordered array of boundary indices, or an iterable collection of slice objects.

aggregate : function

aggregation function (default: *np.mean*)

pad : boolean

If *True*, *idx* is padded to span the full range *[0, data.shape[axis]]*

axis : int

The axis along which to aggregate data

Returns:

data_sync : ndarray

data_sync will have the same dimension as *data*, except that the *axis* coordinate will be reduced according to *idx*.

For example, a 2-dimensional *data* with *axis=-1* should satisfy

data_sync[:, i] = aggregate(data[:, idx[i-1]:idx[i]], axis=-1)

Raises:

ParameterError

If the index set is not of consistent type (all slices or all integers)

Notes

This function caches at level 40.

Examples

Beat-synchronous CQT spectra

```
>>> y, sr = librosa.load(librosa.util.example_audio_file())
>>> tempo, beats = librosa.beat.beat_track(y=y, sr=sr, trim=False)
>>> C = np.abs(librosa.cqt(y=y, sr=sr))
>>> beats = librosa.util.fix_frames(beats, x_max=C.shape[1])
```

By default, use mean aggregation

```
>>> C_avg = librosa.util.sync(C, beats)
```

Use median-aggregation instead of mean

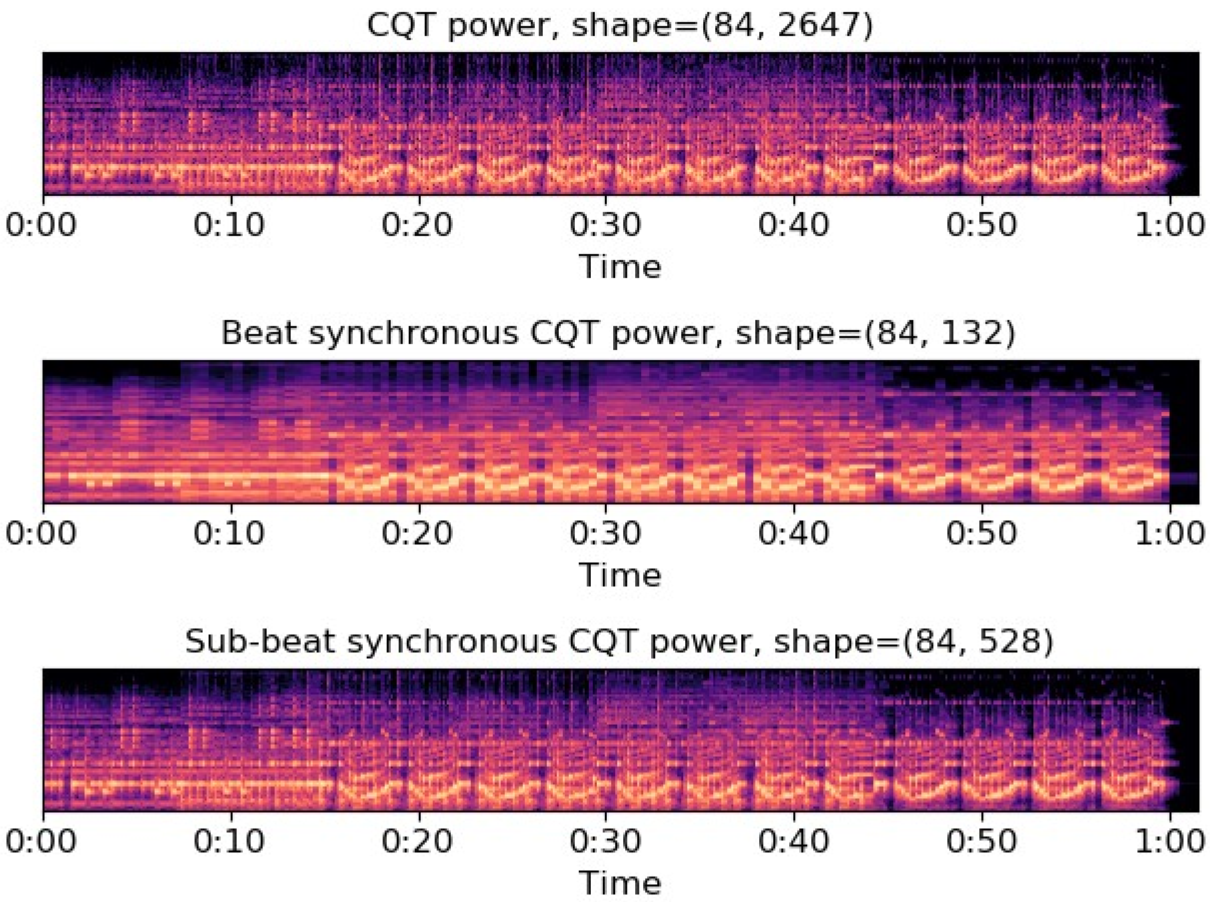
```
>>> C_med = librosa.util.sync(C, beats,
...                              aggregate=np.median)
```

Or sub-beat synchronization

```
>>> sub_beats = librosa.segment.subsegment(C, beats)
>>> sub_beats = librosa.util.fix_frames(sub_beats, x_max=C.shape[1])
>>> C_med_sub = librosa.util.sync(C, sub_beats, aggregate=np.median)
```

Plot the results

```
>>> import matplotlib.pyplot as plt
>>> beat_t = librosa.frames_to_time(beats, sr=sr)
>>> subbeat_t = librosa.frames_to_time(sub_beats, sr=sr)
>>> plt.figure()
>>> plt.subplot(3, 1, 1)
>>> librosa.display.specshow(librosa.amplitude_to_db(C,
...                                                   ref=np.max),
...                           x_axis='time')
>>> plt.title('CQT power, shape={}'.format(C.shape))
>>> plt.subplot(3, 1, 2)
>>> librosa.display.specshow(librosa.amplitude_to_db(C_med,
...                                                   ref=np.max),
...                           x_coords=beat_t, x_axis='time')
>>> plt.title('Beat synchronous CQT power, '
...           'shape={}'.format(C_med.shape))
>>> plt.subplot(3, 1, 3)
>>> librosa.display.specshow(librosa.amplitude_to_db(C_med_sub,
...                                                   ref=np.max),
...                           x_coords=subbeat_t, x_axis='time')
>>> plt.title('Sub-beat synchronous CQT power, '
...           'shape={}'.format(C_med_sub.shape))
>>> plt.tight_layout()
>>> plt.show()
```



librosa.util.axis_sort

`librosa.util.axis_sort`(*S*, *axis=-1*, *index=False*, *value=None*)[source]

Sort an array along its rows or columns.

Parameters: **S** : np.ndarray [shape=(d, n)]

Array to be sorted

axis : int [scalar]

The axis along which to compute the sorting values

- *axis=0* to sort rows by peak column index
- *axis=1* to sort columns by peak row index

index : boolean [scalar]

If true, returns the index array as well as the permuted data.

value : function

function to return the index corresponding to the sort order. Default: *np.argmax*.

Returns: **S_sort** : np.ndarray [shape=(d, n)]

S with the columns or rows permuted in sorting order

idx : np.ndarray (optional) [shape=(d,) or (n,)]

If *index == True*, the sorting index used to permute *S*. Length of *idx* corresponds to the selected *axis*.

Raises: ParameterError

If *S* does not have exactly 2 dimensions (*S.ndim != 2*)

Examples

Visualize NMF output for a spectrogram S

```
>>> # Sort the columns of W by peak frequency bin
>>> y, sr = librosa.load(librosa.util.example_audio_file())
>>> S = np.abs(librosa.stft(y))
>>> W, H = librosa.decompose.decompose(S, n_components=32)
>>> W_sort = librosa.util.axis_sort(W)
```

Or sort by the lowest frequency bin

```
>>> W_sort = librosa.util.axis_sort(W, value=np.argmin)
```

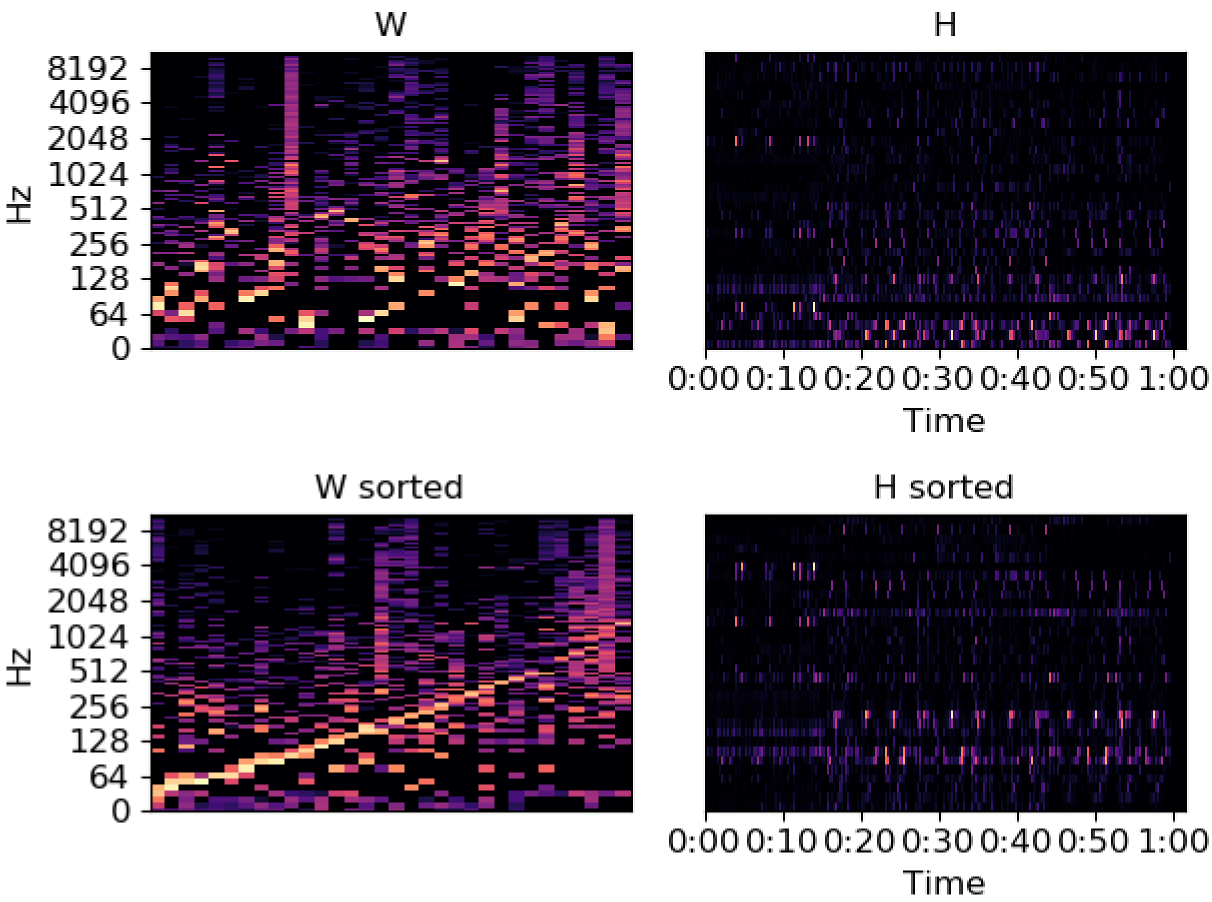
Or sort the rows instead of the columns

```
>>> W_sort_rows = librosa.util.axis_sort(W, axis=0)
```

Get the sorting index also, and use it to permute the rows of H

```
>>> W_sort, idx = librosa.util.axis_sort(W, index=True)
>>> H_sort = H[idx, :]
```

```
>>> import matplotlib.pyplot as plt
>>> plt.figure()
>>> plt.subplot(2, 2, 1)
>>> librosa.display.specshow(librosa.amplitude_to_db(W, ref=np.max),
...                          y_axis='log')
>>> plt.title('W')
>>> plt.subplot(2, 2, 2)
>>> librosa.display.specshow(H, x_axis='time')
>>> plt.title('H')
>>> plt.subplot(2, 2, 3)
>>> librosa.display.specshow(librosa.amplitude_to_db(W_sort,
...                                                  ref=np.max),
...                          y_axis='log')
>>> plt.title('W sorted')
>>> plt.subplot(2, 2, 4)
>>> librosa.display.specshow(H_sort, x_axis='time')
>>> plt.title('H sorted')
>>> plt.tight_layout()
>>> plt.show()
```



librosa.util.normalize

`librosa.util.normalize`(*S*, *norm=inf*, *axis=0*, *threshold=None*, *fill=None*)[source]

Normalize an array along a chosen axis.

Given a norm (described below) and a target axis, the input array is scaled so that

norm(S, axis=axis) == 1

For example, *axis=0* normalizes each column of a 2-d array by aggregating over the rows (0-axis). Similarly, *axis=1* normalizes each row of a 2-d array.

This function also supports thresholding small-norm slices: any slice (i.e., row or column) with norm below a specified *threshold* can be left un-normalized, set to all-zeros, or filled with uniform non-zero values that normalize to 1.

Note: the semantics of this function differ from `scipy.linalg.norm` in two ways: multi-dimensional arrays are supported, but matrix-norms are not.

Parameters: **S** : np.ndarray

The matrix to normalize

norm : {np.inf, -np.inf, 0, float > 0, None}

- *np.inf* : maximum absolute value
- *-np.inf* : mininum absolute value
- *0* : number of non-zeros (the support)
  - float : corresponding l_p norm

    See `scipy.linalg.norm` for details.
- None : no normalization is performed

axis : int [scalar]

Axis along which to compute the norm.

threshold : number > 0 [optional]

Only the columns (or rows) with norm at least *threshold* are normalized.

By default, the threshold is determined from the numerical precision of *S.dtype*.

fill : None or bool

If None, then columns (or rows) with norm below *threshold* are left as is.

If False, then columns (rows) with norm below *threshold* are set to 0.

If True, then columns (rows) with norm below *threshold* are filled uniformly such that the corresponding norm is 1.

Note

fill=True is incompatible with *norm=0* because no uniform vector exists with l0 “norm” equal to 1.

Returns: **S_norm** : np.ndarray [shape=S.shape]

Normalized array

Raises: ParameterError

If *norm* is not among the valid types defined above

If *S* is not finite

If *fill=True* and *norm=0*

See also

`scipy.linalg.norm`

Notes

This function caches at level 40.

Examples

```
>>> # Construct an example matrix
>>> S = np.vander(np.arange(-2.0, 2.0))
>>> S
array([[-8.,  4., -2.,  1.],
       [-1.,  1., -1.,  1.],
       [ 0.,  0.,  0.,  1.],
       [ 1.,  1.,  1.,  1.]])
>>> # Max (l-infinity)-normalize the columns
>>> librosa.util.normalize(S)
array([[-1.   ,  1.   , -1.   ,  1.   ],
       [-0.125,  0.25 , -0.5  ,  1.   ],
       [ 0.   ,  0.   ,  0.   ,  1.   ],
       [ 0.125,  0.25 ,  0.5  ,  1.   ]])
>>> # Max (l-infinity)-normalize the rows
>>> librosa.util.normalize(S, axis=1)
array([[-1.   ,  0.5  , -0.25 ,  0.125],
       [-1.   ,  1.   , -1.   ,  1.   ],
       [ 0.   ,  0.   ,  0.   ,  1.   ],
       [ 1.   ,  1.   ,  1.   ,  1.   ]])
>>> # l1-normalize the columns
>>> librosa.util.normalize(S, norm=1)
array([[-0.8  ,  0.667, -0.5  ,  0.25 ],
       [-0.1  ,  0.167, -0.25 ,  0.25 ],
       [ 0.   ,  0.   ,  0.   ,  0.25 ],
       [ 0.1  ,  0.167,  0.25 ,  0.25 ]])
>>> # l2-normalize the columns
>>> librosa.util.normalize(S, norm=2)
array([[-0.985,  0.943, -0.816,  0.5  ],
       [-0.123,  0.236, -0.408,  0.5  ],
       [ 0.   ,  0.   ,  0.   ,  0.5  ],
       [ 0.123,  0.236,  0.408,  0.5  ]])

>>> # Thresholding and filling
>>> S[:, -1] = 1e-308
>>> S
array([[ -8.000e+000,   4.000e+000,  -2.000e+000,
          1.000e-308],
       [ -1.000e+000,   1.000e+000,  -1.000e+000,
          1.000e-308],
       [  0.000e+000,   0.000e+000,   0.000e+000,
          1.000e-308],
       [  1.000e+000,   1.000e+000,   1.000e+000,
          1.000e-308]])

>>> # By default, small-norm columns are left untouched
>>> librosa.util.normalize(S)
array([[ -1.000e+000,   1.000e+000,  -1.000e+000,
          1.000e-308],
       [ -1.250e-001,   2.500e-001,  -5.000e-001,
          1.000e-308],
       [  0.000e+000,   0.000e+000,   0.000e+000,
          1.000e-308],
       [  1.250e-001,   2.500e-001,   5.000e-001,
          1.000e-308]])
>>> # Small-norm columns can be zeroed out
>>> librosa.util.normalize(S, fill=False)
array([[-1.   ,  1.   , -1.   ,  0.   ],
       [-0.125,  0.25 , -0.5  ,  0.   ],
       [ 0.   ,  0.   ,  0.   ,  0.   ],
       [ 0.125,  0.25 ,  0.5  ,  0.   ]])
>>> # Or set to constant with unit-norm
>>> librosa.util.normalize(S, fill=True)
array([[-1.   ,  1.   , -1.   ,  1.   ],
```

```
       [-0.125,  0.25 , -0.5  ,  1.   ],
       [ 0.   ,  0.   ,  0.   ,  1.   ],
       [ 0.125,  0.25 ,  0.5  ,  1.   ]])
>>> # With an l1 norm instead of max-norm
>>> librosa.util.normalize(S, norm=1, fill=True)
array([[-0.8  ,  0.667, -0.5  ,  0.25 ],
       [-0.1  ,  0.167, -0.25 ,  0.25 ],
       [ 0.   ,  0.   ,  0.   ,  0.25 ],
       [ 0.1  ,  0.167,  0.25 ,  0.25 ]])
```

librosa.util.shear

`librosa.util.shear`(*X*, *factor=1*, *axis=-1*)[source]

Shear a matrix by a given factor.

The *n`th column `X[:, n]* will be displaced (rolled) by *factor * n*.

This is primarily useful for converting between lag and recurrence representations: shearing with *factor=-1* converts the main diagonal to a horizontal. Shearing with *factor=1* converts a horizontal to a diagonal.

Parameters: **X** : np.ndarray [ndim=2] or scipy.sparse matrix

The array to be sheared

factor : integer

The shear factor: *X[:, n] -> np.roll(X[:, n], factor * n)*

axis : integer

The axis along which to shear

Returns: **X_shear** : same type as *X*

The sheared matrix

Examples

```
>>> E = np.eye(3)
>>> librosa.util.shear(E, factor=-1, axis=-1)
array([[1., 1., 1.],
       [0., 0., 0.],
       [0., 0., 0.]])
>>> librosa.util.shear(E, factor=-1, axis=0)
array([[1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.]])
>>> librosa.util.shear(E, factor=1, axis=-1)
array([[1., 0., 0.],
       [0., 0., 1.],
       [0., 1., 0.]])
```

librosa.util.sparsify_rows

`librosa.util.sparsify_rows`(*x*, *quantile=0.01*)[source]

Return a row-sparse matrix approximating the input *x*.

Parameters: **x** : np.ndarray [ndim <= 2]

The input matrix to sparsify.

quantile : float in [0, 1.0)

Percentage of magnitude to discard in each row of *x*

Returns: **x_sparse** : `scipy.sparse.csr_matrix` [shape=x.shape]

Row-sparsified approximation of *x*

If *x.ndim == 1*, then *x* is interpreted as a row vector, and *x_sparse.shape == (1, len(x))*.

Raises: ParameterError

If *x.ndim > 2*

If *quantile* lies outside *[0, 1.0)*

Notes

This function caches at level 40.

Examples

```
>>> # Construct a Hann window to sparsify
>>> x = scipy.signal.hann(32)
>>> x
array([ 0.   ,  0.01 ,  0.041,  0.09 ,  0.156,  0.236,  0.326,
        0.424,  0.525,  0.625,  0.72 ,  0.806,  0.879,  0.937,
        0.977,  0.997,  0.997,  0.977,  0.937,  0.879,  0.806,
        0.72 ,  0.625,  0.525,  0.424,  0.326,  0.236,  0.156,
        0.09 ,  0.041,  0.01 ,  0.   ])
>>> # Discard the bottom percentile
>>> x_sparse = librosa.util.sparsify_rows(x, quantile=0.01)
>>> x_sparse
<1x32 sparse matrix of type '<type 'numpy.float64'>'
    with 26 stored elements in Compressed Sparse Row format>
>>> x_sparse.todense()
matrix([[ 0.   ,  0.   ,  0.   ,  0.09 ,  0.156,  0.236,  0.326,
          0.424,  0.525,  0.625,  0.72 ,  0.806,  0.879,  0.937,
          0.977,  0.997,  0.997,  0.977,  0.937,  0.879,  0.806,
          0.72 ,  0.625,  0.525,  0.424,  0.326,  0.236,  0.156,
          0.09 ,  0.   ,  0.   ,  0.   ]])
>>> # Discard up to the bottom 10th percentile
```

```
>>> x_sparse = librosa.util.sparsify_rows(x, quantile=0.1)
>>> x_sparse
<1x32 sparse matrix of type '<type 'numpy.float64'>'
    with 20 stored elements in Compressed Sparse Row format>
>>> x_sparse.todense()
matrix([[ 0.   ,  0.   ,  0.   ,  0.   ,  0.   ,  0.   ,  0.326,
          0.424,  0.525,  0.625,  0.72 ,  0.806,  0.879,  0.937,
          0.977,  0.997,  0.997,  0.977,  0.937,  0.879,  0.806,
          0.72 ,  0.625,  0.525,  0.424,  0.326,  0.   ,  0.   ,
          0.   ,  0.   ,  0.   ,  0.   ]])
```

librosa.util.buf_to_float

`librosa.util.buf_to_float`(*x*, *n_bytes=2*, *dtype=<class 'numpy.float32'>*)[source]

Convert an integer buffer to floating point values. This is primarily useful when loading integer-valued wav data into numpy arrays.

Parameters: **x** : np.ndarray [dtype=int]

The integer-valued data buffer

n_bytes : int [1, 2, 4]

The number of bytes per sample in *x*

dtype : numeric type

The target output type (default: 32-bit float)

Returns: **x_float** : np.ndarray [dtype=float]

The input data buffer cast to floating point

See also

`buf_to_float`

librosa.util.tiny

`librosa.util.tiny`(*x*)[source]

Compute the tiny-value corresponding to an input’s data type.

This is the smallest “usable” number representable in *x*’s data type (e.g., float32).

This is primarily useful for determining a threshold for numerical underflow in division or multiplication operations.

Parameters: **x** : number or np.ndarray

The array to compute the tiny-value for. All that matters here is *x.dtype*.

Returns: **tiny_value** : float

The smallest positive usable number for the type of *x*. If *x* is integer-typed, then the tiny value for *np.float32* is returned instead.

See also

[numpy.finfo]

Examples

For a standard double-precision floating point number:

```
>>> librosa.util.tiny(1.0)
2.2250738585072014e-308
```

Or explicitly as double-precision

```
>>> librosa.util.tiny(np.asarray(1e-5, dtype=np.float64))
2.2250738585072014e-308
```

Or complex numbers

```
>>> librosa.util.tiny(1j)
2.2250738585072014e-308
```

Single-precision floating point:

```
>>> librosa.util.tiny(np.asarray(1e-5, dtype=np.float32))
1.1754944e-38
```

Integer

```
>>> librosa.util.tiny(5)
1.1754944e-38
```

librosa.util.match_intervals

`librosa.util.match_intervals`(*intervals_from*, *intervals_to*, *strict=True*)[source]

Match one set of time intervals to another.

This can be useful for tasks such as mapping beat timings to segments.

Each element *[a, b]* of *intervals_from* is matched to the element *[c, d]* of *intervals_to* which maximizes the Jaccard similarity between the intervals:

$$max(0, |min(b, d) - max(a, c)|) / |max(d, b) - min(a, c)|$$

In *strict=True* mode, if there is no interval with positive intersection with *[a,b]*, an exception is thrown.

In *strict=False* mode, any interval *[a, b]* that has no intersection with any element of *intervals_to* is instead matched to the interval *[c, d]* which minimizes

> *min(|b - c|, |a - d|)*

that is, the disjoint interval *[c, d]* with a boundary closest to *[a, b]*.

Note

An element of *intervals_to* may be matched to multiple entries of *intervals_from*.

Parameters: **intervals_from** : np.ndarray [shape=(n, 2)]

> The time range for source intervals. The *i* th interval spans time *intervals_from[i, 0]* to *intervals_from[i, 1]*. *intervals_from[0, 0]* should be 0, *intervals_from[-1, 1]* should be the track duration.

intervals_to : np.ndarray [shape=(m, 2)]

> Analogous to *intervals_from*.

strict : bool

> If *True*, intervals can only match if they intersect. If *False*, disjoint intervals can match.

Returns: **interval_mapping** : np.ndarray [shape=(n,)]

> For each interval in *intervals_from*, the corresponding interval in *intervals_to*.

Raises: ParameterError

> If either array of input intervals is not the correct shape
>
> If *strict=True* and some element of *intervals_from* is disjoint from every element of *intervals_to*.

See also

[match_events]

Examples

```
>>> ints_from = np.array([[3, 5], [1, 4], [4, 5]])
>>> ints_to = np.array([[0, 2], [1, 3], [4, 5], [6, 7]])
>>> librosa.util.match_intervals(ints_from, ints_to)
array([2, 1, 2], dtype=uint32)
>>> # [3, 5] => [4, 5]  (ints_to[2])
>>> # [1, 4] => [1, 3]  (ints_to[1])
>>> # [4, 5] => [4, 5]  (ints_to[2])
```

The reverse matching of the above is not possible in *strict* mode because *[6, 7]* is disjoint from all intervals in *ints_from*. With *strict=False*, we get the following:

```
>>> librosa.util.match_intervals(ints_to, ints_from, strict=False)
array([1, 1, 2, 2], dtype=uint32)
>>> # [0, 2] => [1, 4]  (ints_from[1])
>>> # [1, 3] => [1, 4]  (ints_from[1])
>>> # [4, 5] => [4, 5]  (ints_from[2])
>>> # [6, 7] => [4, 5]  (ints_from[2])
```

---

librosa.util.match_events

`librosa.util.match_events`(*events_from*, *events_to*, *left=True*, *right=True*)[source]

Match one set of events to another.

This is useful for tasks such as matching beats to the nearest detected onsets, or frame-aligned events to the nearest zero-crossing.

Note

A target event may be matched to multiple source events.

Parameters: **events_from** : ndarray [shape=(n,)]

Array of events (eg, times, sample or frame indices) to match from.

events_to : ndarray [shape=(m,)]

Array of events (eg, times, sample or frame indices) to match against.

left : bool
right : bool

If *False*, then matched events cannot be to the left (or right) of source events.

Returns: **event_mapping** : np.ndarray [shape=(n,)]

For each event in *events_from*, the corresponding event index in *events_to*.

event_mapping[i] == arg min |events_from[i] - events_to[:]|

Raises: ParameterError

If either array of input events is not the correct shape

See also

match_intervals

Examples

```
>>> # Sources are multiples of 7
>>> s_from = np.arange(0, 100, 7)
>>> s_from
array([ 0,  7, 14, 21, 28, 35, 42, 49, 56, 63, 70, 77, 84, 91,
       98])
>>> # Targets are multiples of 10
>>> s_to = np.arange(0, 100, 10)
>>> s_to
array([ 0, 10, 20, 30, 40, 50, 60, 70, 80, 90])
>>> # Find the matching
>>> idx = librosa.util.match_events(s_from, s_to)
>>> idx
array([0, 1, 1, 2, 3, 3, 4, 5, 6, 6, 7, 8, 8, 9, 9])
>>> # Print each source value to its matching target
>>> zip(s_from, s_to[idx])
[(0, 0), (7, 10), (14, 10), (21, 20), (28, 30), (35, 30),
 (42, 40), (49, 50), (56, 60), (63, 60), (70, 70), (77, 80),
 (84, 80), (91, 90), (98, 90)]
```

librosa.util.localmax

`librosa.util.localmax`(*x*, *axis=0*)[source]

Find local maxima in an array *x*.

An element *x[i]* is considered a local maximum if the following conditions are met:

- *x[i] > x[i-1]*
- *x[i] >= x[i+1]*

Note that the first condition is strict, and that the first element *x[0]* will never be considered as a local maximum.

Parameters: **x** : np.ndarray [shape=(d1,d2,…)]

input vector or array

axis : int

axis along which to compute local maximality

Returns: **m** : np.ndarray [shape=x.shape, dtype=bool]

indicator array of local maximality along *axis*

Examples

```
>>> x = np.array([1, 0, 1, 2, -1, 0, -2, 1])
>>> librosa.util.localmax(x)
array([False, False, False,  True, False,  True, False,  True], dtype=bool)

>>> # Two-dimensional example
>>> x = np.array([[1,0,1], [2, -1, 0], [2, 1, 3]])
>>> librosa.util.localmax(x, axis=0)
array([[False, False, False],
       [ True, False, False],
       [False,  True,  True]], dtype=bool)
>>> librosa.util.localmax(x, axis=1)
array([[False, False,  True],
       [False, False,  True],
       [False, False,  True]], dtype=bool)
```

librosa.util.peak_pick

`librosa.util.peak_pick`(*x*, *pre_max*, *post_max*, *pre_avg*, *post_avg*, *delta*, *wait*)[source]

Uses a flexible heuristic to pick peaks in a signal.

A sample n is selected as an peak if the corresponding x[n] fulfills the following three conditions:

1. *x[n] == max(x[n - pre_max:n + post_max])*
2. *x[n] >= mean(x[n - pre_avg:n + post_avg]) + delta*
3. *n - previous_n > wait*

where *previous_n* is the last sample picked as a peak (greedily).

This implementation is based on [1] and [2].

[1] Boeck, Sebastian, Florian Krebs, and Markus Schedl. "Evaluating the Online Capabilities of Onset Detection Methods." ISMIR. 2012.

[2] https://github.com/CPJKU/onset_detection/blob/master/onset_program.py

Parameters: **x** : np.ndarray [shape=(n,)]

input signal to peak picks from

pre_max : int >= 0 [scalar]

number of samples before *n* over which max is computed

post_max : int >= 1 [scalar]

number of samples after *n* over which max is computed

pre_avg : int >= 0 [scalar]

number of samples before *n* over which mean is computed

post_avg : int >= 1 [scalar]

number of samples after *n* over which mean is computed

delta : float >= 0 [scalar]

threshold offset for mean

wait : int >= 0 [scalar]

number of samples to wait after picking a peak

Returns: **peaks** : np.ndarray [shape=(n_peaks,), dtype=int]

indices of peaks in *x*

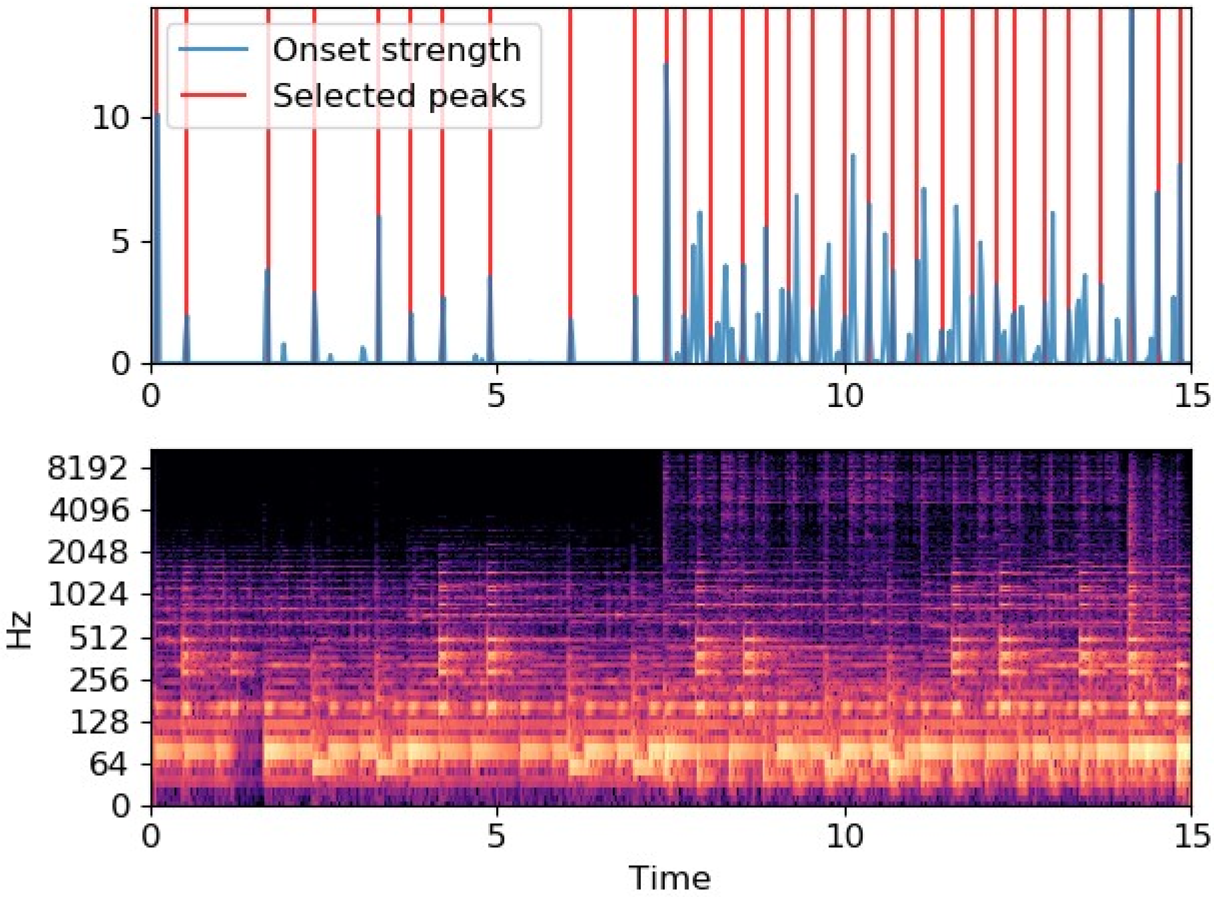
Raises: ParameterError

If any input lies outside its defined range

Examples

```
>>> y, sr = librosa.load(librosa.util.example_audio_file(), duration=15)
>>> onset_env = librosa.onset.onset_strength(y=y, sr=sr,
...                                          hop_length=512,
...                                          aggregate=np.median)
>>> peaks = librosa.util.peak_pick(onset_env, 3, 3, 3, 5, 0.5, 10)
>>> peaks
array([  4,  23,  73, 102, 142, 162, 182, 211, 261, 301, 320,
       331, 348, 368, 382, 396, 411, 431, 446, 461, 476, 491,
       510, 525, 536, 555, 570, 590, 609, 625, 639])

>>> import matplotlib.pyplot as plt
>>> times = librosa.times_like(onset_env, sr=sr, hop_length=512)
>>> plt.figure()
>>> ax = plt.subplot(2, 1, 2)
>>> D = librosa.stft(y)
>>> librosa.display.specshow(librosa.amplitude_to_db(D, ref=np.max),
...                          y_axis='log', x_axis='time')
>>> plt.subplot(2, 1, 1, sharex=ax)
>>> plt.plot(times, onset_env, alpha=0.8, label='Onset strength')
>>> plt.vlines(times[peaks], 0,
...            onset_env.max(), color='r', alpha=0.8,
...            label='Selected peaks')
>>> plt.legend(frameon=True, framealpha=0.8)
>>> plt.axis('tight')
>>> plt.tight_layout()
>>> plt.show()
```



librosa.util.nnls

`librosa.util.nnls`(*A*, *B*, ***kwargs*)[source]

Non-negative least squares.

Given two matrices A and B, find a non-negative matrix X that minimizes the sum squared error:

err(X) = sum_i,j ((AX)[i,j] - B[i, j])^2

Parameters: **A** : np.ndarray [shape=(m, n)]

The basis matrix

B : np.ndarray [shape=(m, N)]

The target matrix.

kwargs

Additional keyword arguments to `scipy.optimize.fmin_l_bfgs_b`

Returns: **X** : np.ndarray [shape=(n, N), non-negative]

A minimizing solution to $|AX - B|^2$

See also

[scipy.optimize.nnls]
[scipy.optimize.fmin_l_bfgs_b]

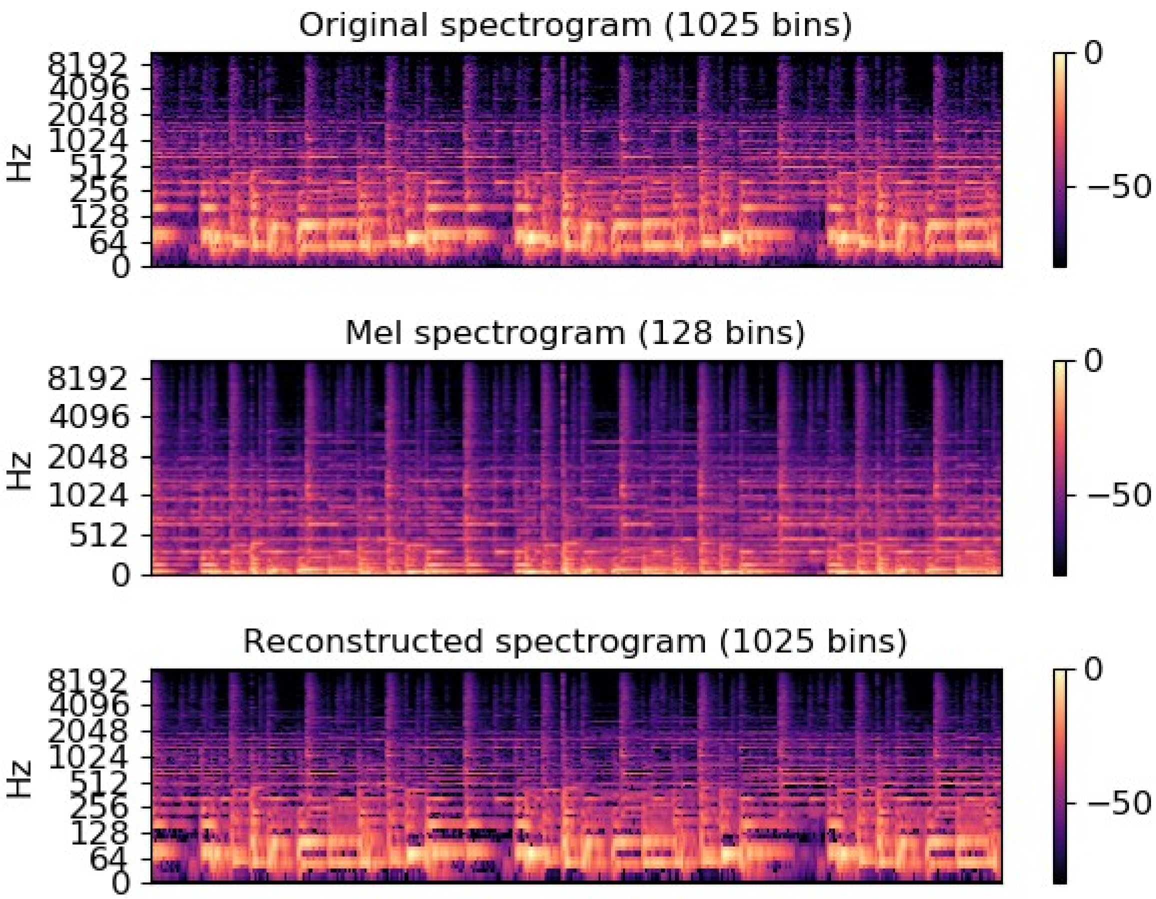
Examples

Approximate a magnitude spectrum from its mel spectrogram

```
>>> y, sr = librosa.load(librosa.util.example_audio_file(), offset=30,
duration=10)
>>> S = np.abs(librosa.stft(y, n_fft=2048))
>>> M = librosa.feature.melspectrogram(S=S, sr=sr, power=1)
>>> mel_basis = librosa.filters.mel(sr, n_fft=2048, n_mels=M.shape[0])
>>> S_recover = librosa.util.nnls(mel_basis, M)
```

Plot the results

```
>>> import matplotlib.pyplot as plt
>>> plt.figure()
>>> plt.subplot(3,1,1)
>>> librosa.display.specshow(librosa.amplitude_to_db(S, ref=np.max),
y_axis='log')
>>> plt.colorbar()
>>> plt.title('Original spectrogram (1025 bins)')
>>> plt.subplot(3,1,2)
>>> librosa.display.specshow(librosa.amplitude_to_db(M, ref=np.max),
...                          y_axis='mel')
>>> plt.title('Mel spectrogram (128 bins)')
>>> plt.colorbar()
>>> plt.subplot(3,1,3)
>>> librosa.display.specshow(librosa.amplitude_to_db(S_recover,
ref=np.max),
...                          y_axis='log')
>>> plt.colorbar()
>>> plt.title('Reconstructed spectrogram (1025 bins)')
>>> plt.tight_layout()
>>> plt.show()
```



librosa.util.cyclic_gradient

`librosa.util.cyclic_gradient`(*data*, *edge_order=1*, *axis=-1*)[source]

Estimate the gradient of a function over a uniformly sampled, periodic domain.

This is essentially the same as *np.gradient*, except that edge effects are handled by wrapping the observations (i.e. assuming periodicity) rather than extrapolation.

Parameters: **data** : np.ndarray

The function values observed at uniformly spaced positions on a periodic domain

edge_order: {1, 2}

The order of the difference approximation used for estimating the gradient

axis : int

The axis along which gradients are calculated.

Returns: **grad** : np.ndarray like *data*

The gradient of *data* taken along the specified axis.

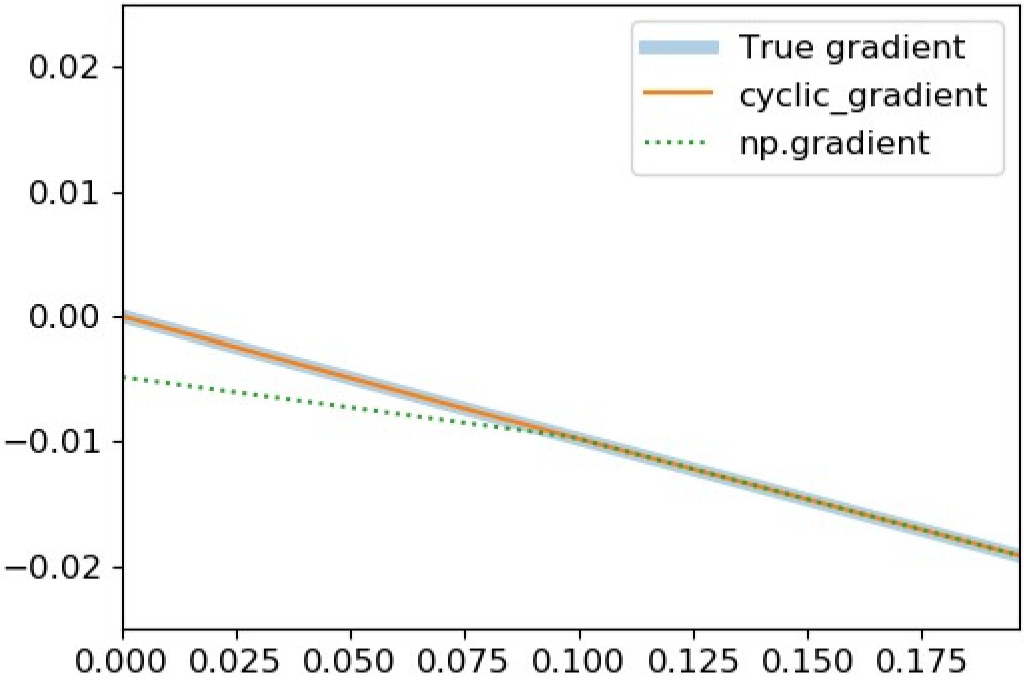
See also

`np.gradient`

Examples

This example estimates the gradient of cosine (-sine) from 64 samples using direct (aperiodic) and periodic gradient calculation.

```
>>> import matplotlib.pyplot as plt
>>> x = 2 * np.pi * np.linspace(0, 1, num=64, endpoint=False)
>>> y = np.cos(x)
>>> grad = np.gradient(y)
>>> cyclic_grad = librosa.util.cyclic_gradient(y)
>>> true_grad = -np.sin(x) * 2 * np.pi / len(x)
>>> plt.plot(x, true_grad, label='True gradient', linewidth=5,
...          alpha=0.35)
>>> plt.plot(x, cyclic_grad, label='cyclic_gradient')
>>> plt.plot(x, grad, label='np.gradient', linestyle=':')
>>> plt.legend()
>>> # Zoom into the first part of the sequence
>>> plt.xlim([0, np.pi/16])
>>> plt.ylim([-0.025, 0.025])
>>> plt.show()
```



librosa.util.valid_audio

`librosa.util.valid_audio`(*y*, *mono=True*)[source]

Validate whether a variable contains valid, mono audio data.

Parameters: **y** : np.ndarray

The input data to validate

mono : bool

Whether or not to force monophonic audio

Returns: **valid** : bool

True if all tests pass

Raises: ParameterError

In any of these cases:

- *type(y)* is not *np.ndarray*
- *y.dtype* is not floating-point

- *mono == True* and *y.ndim* is not 1
- *mono == False* and *y.ndim* is not 1 or 2
- *np.isfinite(y).all()* is False
- *y.flags["F_CONTIGUOUS"]* is False

See also

`stack`
`numpy.asfortranarray`
`numpy.float32`

Notes

This function caches at level 20.

Examples

```
>>> # By default, valid_audio allows only mono signals
>>> filepath = librosa.util.example_audio_file()
>>> y_mono, sr = librosa.load(filepath, mono=True)
>>> y_stereo, _ = librosa.load(filepath, mono=False)
>>> librosa.util.valid_audio(y_mono), librosa.util.valid_audio(y_stereo)
True, False
```

```
>>> # To allow stereo signals, set mono=False
>>> librosa.util.valid_audio(y_stereo, mono=False)
True
```

librosa.util.valid_int

`librosa.util.valid_int`(*x*, *cast=None*)[source]

Ensure that an input value is integer-typed. This is primarily useful for ensuring integrable-valued array indices.

Parameters: **x** : number

A scalar value to be cast to int

cast : function [optional]

A function to modify *x* before casting. Default: *np.floor*

Returns: **x_int** : int

x_int = int(cast(x))

Raises: ParameterError

If *cast* is provided and is not callable.

librosa.util.valid_intervals

`librosa.util.valid_intervals`(*intervals*)[source]

Ensure that an array is a valid representation of time intervals:

- intervals.ndim == 2
- intervals.shape[1] == 2
- intervals[i, 0] <= intervals[i, 1] for all i

Parameters: **intervals** : np.ndarray [shape=(n, 2)]

set of time intervals

Returns: **valid** : bool

True if *intervals* passes validation.

librosa.util.example_audio_file

`librosa.util.example_audio_file`()[source]

Get the path to an included audio example file.

Vibe Ace (Kevin MacLeod) / CC BY 3.0

Returns: **filename** : str

Path to the audio example file included with librosa

Examples

```
>>> # Load the waveform from the example track
>>> y, sr = librosa.load(librosa.util.example_audio_file())
```

librosa.util.find_files

`librosa.util.find_files`(*directory*, *ext=None*, *recurse=True*, *case_sensitive=False*, *limit=None*, *offset=0*)[source]

Get a sorted list of (audio) files in a directory or directory sub-tree.

Parameters: **directory** : str

Path to look for files

ext : str or list of str

A file extension or list of file extensions to include in the search.

Default: *['aac', 'au', 'flac', 'm4a', 'mp3', 'ogg', 'wav']*

recurse : boolean

If *True*, then all subfolders of *directory* will be searched.

Otherwise, only *directory* will be searched.

case_sensitive : boolean

If *False*, files matching upper-case version of extensions will be included.

limit : int > 0 or None

Return at most *limit* files. If *None*, all files are returned.

offset : int

Return files starting at *offset* within the list.

Use negative values to offset from the end of the list.

Returns: **files** : list of str

The list of audio files.

Examples

```
>>> # Get all audio files in a directory sub-tree
>>> files = librosa.util.find_files('~/Music')

>>> # Look only within a specific directory, not the sub-tree
>>> files = librosa.util.find_files('~/Music', recurse=False)

>>> # Only look for mp3 files
>>> files = librosa.util.find_files('~/Music', ext='mp3')

>>> # Or just mp3 and ogg
>>> files = librosa.util.find_files('~/Music', ext=['mp3', 'ogg'])
```

```
>>> # Only get the first 10 files
>>> files = librosa.util.find_files('~/Music', limit=10)

>>> # Or last 10 files
>>> files = librosa.util.find_files('~/Music', offset=-10)
```

librosa.util.roll_sparse

`librosa.util.roll_sparse`(*x*, *shift*, *axis=0*)[source]

Sparse matrix roll

This operation is equivalent to `numpy.roll`, but operates on sparse matrices.

Warning

This function is deprecated in version 0.7.1. It will be removed in version 0.8.0.

Parameters: **x** : scipy.sparse.spmatrix or np.ndarray

The sparse matrix input

shift : int

The number of positions to roll the specified axis

axis : (0, 1, -1)

The axis along which to roll.

Returns: **x_rolled** : same type as *x*

The rolled matrix, with the same format as *x*

See also

`numpy.roll`

Examples

```
>>> # Generate a random sparse binary matrix
>>> X = scipy.sparse.lil_matrix(np.random.randint(0, 2, size=(5,5)))
>>> X_roll = roll_sparse(X, 2, axis=0)  # Roll by 2 on the first axis
>>> X_dense_r = roll_sparse(X.toarray(), 2, axis=0)  # Equivalent dense
roll
>>> np.allclose(X_roll, X_dense_r.toarray())
True
```

Filters

Filter bank construction

mel(sr, n_fft[, n_mels, fmin, fmax, htk, …])	Create a Filterbank matrix to combine FFT bins into Mel-frequency bins
chroma(sr, n_fft[, n_chroma, tuning, A440, …])	Create a Filterbank matrix to convert STFT to chroma
constant_q(sr[, fmin, n_bins, …])	Construct a constant-Q basis.
_multirate_fb([center_freqs, sample_rates, …])	Helper function to construct a multirate filterbank.
semitone_filterbank([center_freqs, tuning, …])	Constructs a multirate filterbank of infinite-impulse response (IIR) band-pass filters at user-defined center frequencies and sample rates.

Window functions

window_bandwidth(window[, n])	Get the equivalent noise bandwidth of a window function.
get_window(window, Nx[, fftbins])	Compute a window function.

Miscellaneous

constant_q_lengths(sr, fmin[, n_bins, …])	Return length of each filter in a constant-Q basis.
cq_to_chroma(n_input[, bins_per_octave, …])	Convert a Constant-Q basis to Chroma.
mr_frequencies(tuning)	Helper function for generating center frequency and sample rate pairs.
window_sumsquare(window, n_frames[, …])	Compute the sum-square envelope of a window function at a given hop length.
diagonal_filter(window, n[, slope, angle, …])	Build a two-dimensional diagonal filter.

librosa.filters.mel

`librosa.filters.mel`(*sr, n_fft, n_mels=128, fmin=0.0, fmax=None, htk=False, norm=1, dtype=<class 'numpy.float32'>*)[source]

Create a Filterbank matrix to combine FFT bins into Mel-frequency bins

Parameters: **sr** : number > 0 [scalar]

> sampling rate of the incoming signal

n_fft : int > 0 [scalar]

> number of FFT components

n_mels : int > 0 [scalar]

> number of Mel bands to generate

fmin : float >= 0 [scalar]

> lowest frequency (in Hz)

fmax : float >= 0 [scalar]

> highest frequency (in Hz). If *None*, use *fmax = sr / 2.0*

htk : bool [scalar]

> use HTK formula instead of Slaney

norm : {None, 1, np.inf} [scalar]

> if 1, divide the triangular mel weights by the width of the mel band (area normalization). Otherwise, leave all the triangles aiming for a peak value of 1.0

dtype : np.dtype

> The data type of the output basis. By default, uses 32-bit (single-precision) floating point.

Returns: **M** : np.ndarray [shape=(n_mels, 1 + n_fft/2)]

> Mel transform matrix

Notes

This function caches at level 10.

Examples

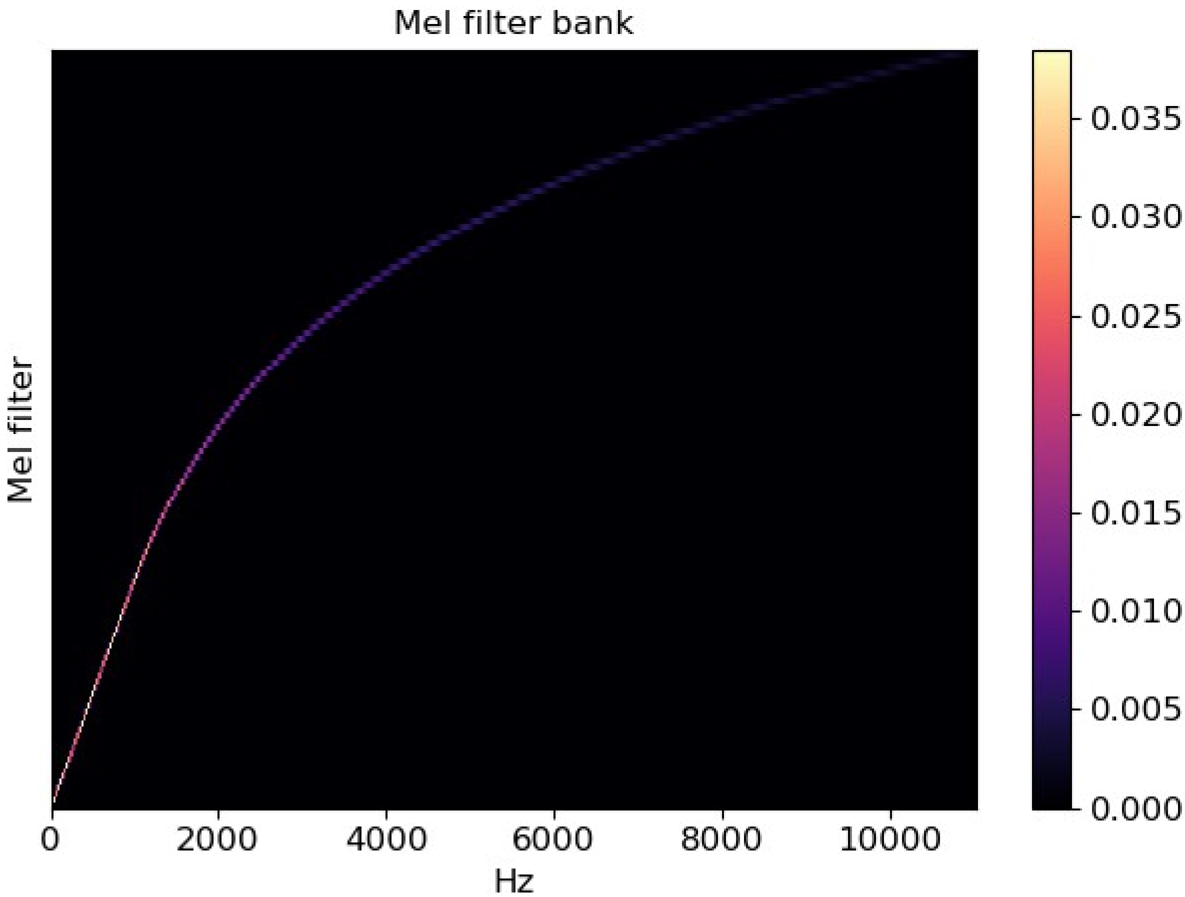
```
>>> melfb = librosa.filters.mel(22050, 2048)
>>> melfb
array([[ 0.   ,  0.016, ...,  0.   ,  0.   ],
       [ 0.   ,  0.   , ...,  0.   ,  0.   ],
       ...,
       [ 0.   ,  0.   , ...,  0.   ,  0.   ],
```

```
       [ 0.   ,  0.   , ...,  0.   ,  0.   ]])
```

Clip the maximum frequency to 8KHz

```
>>> librosa.filters.mel(22050, 2048, fmax=8000)
array([[ 0.  ,  0.02, ...,  0.  ,  0.  ],
       [ 0.  ,  0.  , ...,  0.  ,  0.  ],
       ...,
       [ 0.  ,  0.  , ...,  0.  ,  0.  ],
       [ 0.  ,  0.  , ...,  0.  ,  0.  ]])
```

```
>>> import matplotlib.pyplot as plt
>>> plt.figure()
>>> librosa.display.specshow(melfb, x_axis='linear')
>>> plt.ylabel('Mel filter')
>>> plt.title('Mel filter bank')
>>> plt.colorbar()
>>> plt.tight_layout()
>>> plt.show()
```



librosa.filters.chroma

`librosa.filters.chroma`(*sr*, *n_fft*, *n_chroma=12*, *tuning=0.0*, *A440=<DEPRECATED parameter>*, *ctroct=5.0*, *octwidth=2*, *norm=2*, *base_c=True*, *dtype=<class 'numpy.float32'>*)
[source]

Create a Filterbank matrix to convert STFT to chroma

Parameters: **sr** : number > 0 [scalar]

audio sampling rate

n_fft : int > 0 [scalar]

number of FFT bins

n_chroma : int > 0 [scalar]

number of chroma bins

tuning : float

Tuning deviation from A440 in fractions of a chroma bin.

A440 : float > 0 [scalar] <Deprecated>

Reference frequency for A440

Note

This parameter is deprecated in version 0.7.1. It will be removed in 0.8.0. Use *tuning*= instead.

ctroct : float > 0 [scalar]
octwidth : float > 0 or None [scalar]

ctroct and *octwidth* specify a dominance window - a Gaussian weighting centered on *ctroct* (in octs, A0 = 27.5Hz) and with a gaussian half-width of *octwidth*. Set *octwidth* to *None* to use a flat weighting.

norm : float > 0 or np.inf

Normalization factor for each filter

base_c : bool

If True, the filter bank will start at ‘C’. If False, the filter bank will start at ‘A’.

dtype : np.dtype

The data type of the output basis. By default, uses 32-bit (single-precision) floating point.

Returns: **wts** : ndarray [shape=(n_chroma, 1 + n_fft / 2)]

Chroma filter matrix

See also

`util.normalize`
`feature.chroma_stft`

Notes

This function caches at level 10.

Examples

Build a simple chroma filter bank

```
>>> chromafb = librosa.filters.chroma(22050, 4096)
array([[  1.689e-05,   3.024e-04, ...,   4.639e-17,   5.327e-17],
       [  1.716e-05,   2.652e-04, ...,   2.674e-25,   3.176e-25],
...,
       [  1.578e-05,   3.619e-04, ...,   8.577e-06,   9.205e-06],
       [  1.643e-05,   3.355e-04, ...,   1.474e-10,   1.636e-10]])
```

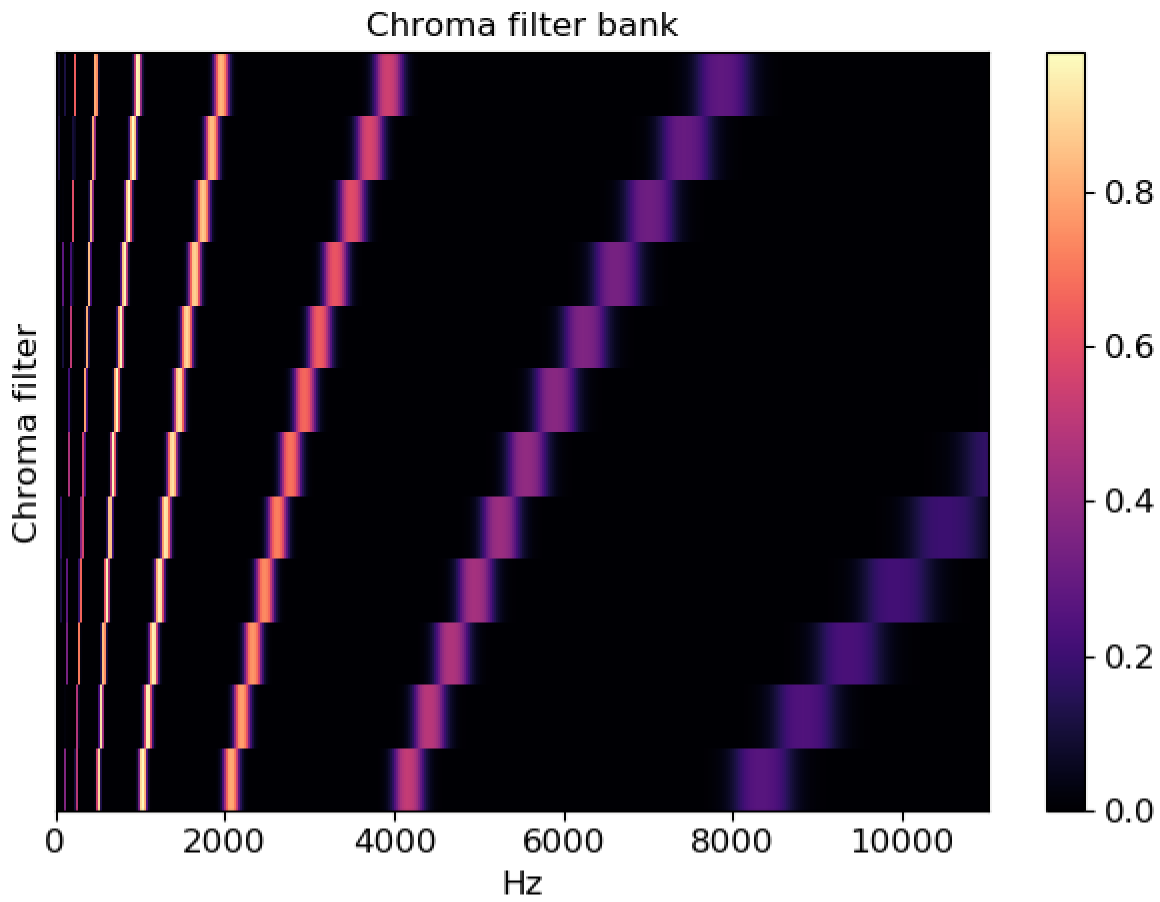
Use quarter-tones instead of semitones

```
>>> librosa.filters.chroma(22050, 4096, n_chroma=24)
array([[  1.194e-05,   2.138e-04, ...,   6.297e-64,   1.115e-63],
       [  1.206e-05,   2.009e-04, ...,   1.546e-79,   2.929e-79],
...,
       [  1.162e-05,   2.372e-04, ...,   6.417e-38,   9.923e-38],
       [  1.180e-05,   2.260e-04, ...,   4.697e-50,   7.772e-50]])
```

Equally weight all octaves

```
>>> librosa.filters.chroma(22050, 4096, octwidth=None)
array([[  3.036e-01,   2.604e-01, ...,   2.445e-16,   2.809e-16],
       [  3.084e-01,   2.283e-01, ...,   1.409e-24,   1.675e-24],
...,
       [  2.836e-01,   3.116e-01, ...,   4.520e-05,   4.854e-05],
       [  2.953e-01,   2.888e-01, ...,   7.768e-10,   8.629e-10]])
```

```
>>> import matplotlib.pyplot as plt
>>> plt.figure()
>>> librosa.display.specshow(chromafb, x_axis='linear')
>>> plt.ylabel('Chroma filter')
>>> plt.title('Chroma filter bank')
>>> plt.colorbar()
>>> plt.tight_layout()
>>> plt.show()
```



librosa.filters.constant_q

`librosa.filters.constant_q`(*sr*, *fmin=None*, *n_bins=84*, *bins_per_octave=12*, *tuning=<DEPRECATED parameter>*, *window='hann'*, *filter_scale=1*, *pad_fft=True*, *norm=1*, *dtype=<class 'numpy.complex64'>*, ***kwargs*)[source]

Construct a constant-Q basis.

This uses the filter bank described by [1].

[1] McVicar, Matthew. "A machine learning approach to automatic chord extraction." Dissertation, University of Bristol. 2013.

Parameters: **sr** : number > 0 [scalar]

Audio sampling rate

fmin : float > 0 [scalar]

Minimum frequency bin. Defaults to *C1 ~= 32.70*

n_bins : int > 0 [scalar]

Number of frequencies. Defaults to 7 octaves (84 bins).

bins_per_octave : int > 0 [scalar]

Number of bins per octave

tuning : float [scalar] <DEPRECATED>

Tuning deviation from A440 in fractions of a bin

Note

This parameter is deprecated in 0.7.1. It will be removed in version 0.8.

window : string, tuple, number, or function

Windowing function to apply to filters.

filter_scale : float > 0 [scalar]

Scale of filter windows. Small values (<1) use shorter windows for higher temporal resolution.

pad_fft : boolean

Center-pad all filters up to the nearest integral power of 2.

By default, padding is done with zeros, but this can be overridden by setting the *mode=* field in *kwargs*.

norm : {inf, -inf, 0, float > 0}

Type of norm to use for basis function normalization. See librosa.util.normalize

dtype : np.dtype

The data type of the output basis. By default, uses 64-bit (single precision) complex floating point.

kwargs : additional keyword arguments

Arguments to *np.pad()* when *pad==True*.

Returns:

filters : np.ndarray, *len(filters) == n_bins*

filters[i] is *i*th time-domain CQT basis filter

lengths : np.ndarray, *len(lengths) == n_bins*

The (fractional) length of each filter

See also

constant_q_lengths
librosa.core.cqt
librosa.util.normalize

Notes

This function caches at level 10.

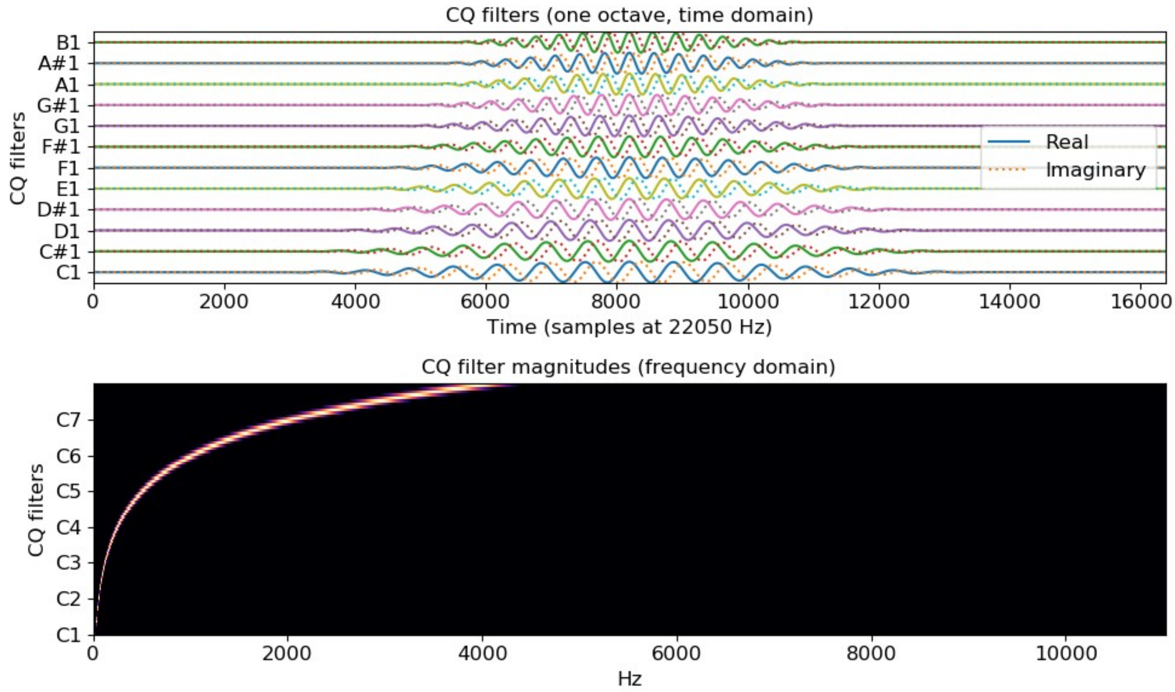
Examples

Use a shorter window for each filter

```
>>> basis, lengths = librosa.filters.constant_q(22050, filter_scale=0.5)
```

Plot one octave of filters in time and frequency

```
>>> import matplotlib.pyplot as plt
>>> basis, lengths = librosa.filters.constant_q(22050)
>>> plt.figure(figsize=(10, 6))
>>> plt.subplot(2, 1, 1)
>>> notes = librosa.midi_to_note(np.arange(24, 24 + len(basis)))
>>> for i, (f, n) in enumerate(zip(basis, notes[:12])):
...     f_scale = librosa.util.normalize(f) / 2
...     plt.plot(i + f_scale.real)
...     plt.plot(i + f_scale.imag, linestyle=':')
>>> plt.axis('tight')
>>> plt.yticks(np.arange(len(notes[:12])), notes[:12])
>>> plt.ylabel('CQ filters')
>>> plt.title('CQ filters (one octave, time domain)')
>>> plt.xlabel('Time (samples at 22050 Hz)')
>>> plt.legend(['Real', 'Imaginary'], frameon=True, framealpha=0.8)
>>> plt.subplot(2, 1, 2)
>>> F = np.abs(np.fft.fftn(basis, axes=[-1]))
>>> # Keep only the positive frequencies
>>> F = F[:, :(1 + F.shape[1] // 2)]
>>> librosa.display.specshow(F, x_axis='linear')
>>> plt.yticks(np.arange(len(notes))[::12], notes[::12])
>>> plt.ylabel('CQ filters')
>>> plt.title('CQ filter magnitudes (frequency domain)')
>>> plt.tight_layout()
```




```
>>> plt.show()
```

librosa.filters._multirate_fb

`librosa.filters._multirate_fb`(*center_freqs=None*, *sample_rates=None*, *Q=25.0*, *passband_ripple=1*, *stopband_attenuation=50*, *ftype='ellip'*, *flayout='sos'*)[source]

Helper function to construct a multirate filterbank.

A filter bank consists of multiple band-pass filters which divide the input signal into subbands. In the case of a multirate filter bank, the band-pass filters operate with resampled versions of the input signal, e.g. to keep the length of a filter constant while shifting its center frequency.

This implementation uses `scipy.signal.iirdesign` to design the filters.

Parameters :

center_freqs : np.ndarray [shape=(n,), dtype=float]

Center frequencies of the filter kernels. Also defines the number of filters in the filterbank.

sample_rates : np.ndarray [shape=(n,), dtype=float]

Samplerate for each filter (used for multirate filterbank).

Q : float

Q factor (influences the filter bandwith).

passband_ripple : float

The maximum loss in the passband (dB) See `scipy.signal.iirdesign` for details.

stopband_attenuation : float

The minimum attenuation in the stopband (dB) See `scipy.signal.iirdesign` for details.

ftype : str

The type of IIR filter to design See `scipy.signal.iirdesign` for details.

flayout : string

Valid *output* argument for `scipy.signal.iirdesign`.

- If *ba*, returns numerators/denominators of the transfer functions, used for filtering with `scipy.signal.filtfilt`. Can be unstable for high-order filters.
- If *sos*, returns a series of second-order filters, used for filtering with `scipy.signal.sosfiltfilt`. Minimizes numerical precision errors for high-order filters, but is slower.
- If *zpk*, returns zeros, poles, and system gains of the transfer functions.

Returns: **filterbank** : list [shape=(n,), dtype=float]

Each list entry comprises the filter coefficients for a single filter.

sample_rates : np.ndarray [shape=(n,), dtype=float]

Samplerate for each filter.

Raises: ParameterError

If *center_freqs* is *None*. If *sample_rates* is *None*. If *center_freqs.shape* does not match *sample_rates.shape*.

See also

`scipy.signal.iirdesign`

Notes

This function caches at level 10.

librosa.filters.semitone_filterbank

`librosa.filters.semitone_filterbank`(*center_freqs=None*, *tuning=0.0*, *sample_rates=None*, *flayout='ba'*, ***kwargs*)[source]

Constructs a multirate filterbank of infinite-impulse response (IIR) band-pass filters at user-defined center frequencies and sample rates.

By default, these center frequencies are set equal to the 88 fundamental frequencies of the grand piano keyboard, according to a pitch tuning standard of A440, that is, note A above middle C set to 440 Hz. The center frequencies are tuned to the twelve-tone equal temperament, which means that they grow exponentially at a rate of 2**(1/12), that is, twelve notes per octave.

The A440 tuning can be changed by the user while keeping twelve-tone equal temperament. While A440 is currently the international standard in the music industry (ISO 16), some orchestras tune to A441-A445, whereas baroque musicians tune to A415.

See [1] for details.

[1] Müller, Meinard. "Information Retrieval for Music and Motion." Springer Verlag. 2007.

Parameters:

center_freqs : np.ndarray [shape=(n,), dtype=float]

Center frequencies of the filter kernels. Also defines the number of filters in the filterbank.

tuning : float [scalar]

Tuning deviation from A440 as a fraction of a semitone (1/12 of an octave in equal temperament).

sample_rates : np.ndarray [shape=(n,), dtype=float]

Sample rates of each filter in the multirate filterbank.

flayout : string

- If *ba*, the standard difference equation is used for filtering with `scipy.signal.filtfilt`. Can be unstable for high-order filters.
- If *sos*, a series of second-order filters is used for filtering with `scipy.signal.sosfiltfilt`. Minimizes numerical precision errors for high-order filters, but is slower.

kwargs : additional keyword arguments

Additional arguments to the private function *_multirate_fb()*.

Returns:

filterbank : list [shape=(n,), dtype=float]

Each list entry contains the filter coefficients for a single filter.

fb_sample_rates : np.ndarray [shape=(n,), dtype=float]

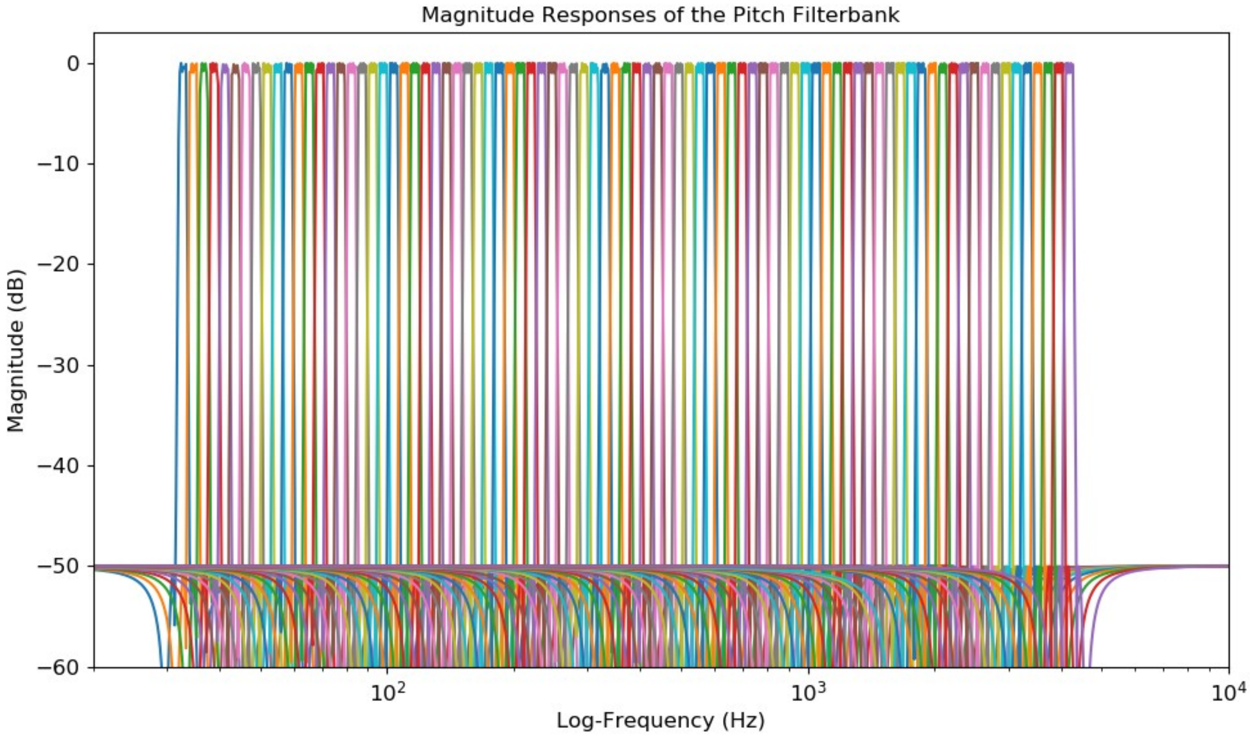
Sample rate for each filter.

See also

[librosa.core.cqt]
[librosa.core.iirt]
[librosa.filters._multirate_fb]
[librosa.filters.mr_frequencies]
[scipy.signal.iirdesign]

Examples

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> import scipy.signal
>>> semitone_filterbank, sample_rates =
librosa.filters.semitone_filterbank()
>>> plt.figure(figsize=(10, 6))
>>> for cur_sr, cur_filter in zip(sample_rates, semitone_filterbank):
...     w, h = scipy.signal.freqz(cur_filter[0], cur_filter[1], worN=2000)
...     plt.plot((cur_sr / (2 * np.pi)) * w, 20 * np.log10(abs(h)))
>>> plt.semilogx()
>>> plt.xlim([20, 10e3])
>>> plt.ylim([-60, 3])
>>> plt.title('Magnitude Responses of the Pitch Filterbank')
>>> plt.xlabel('Log-Frequency (Hz)')
>>> plt.ylabel('Magnitude (dB)')
>>> plt.tight_layout()
>>> plt.show()
```



librosa.filters.window_bandwidth

`librosa.filters.window_bandwidth`(*window*, *n=1000*)[source]

Get the equivalent noise bandwidth of a window function.

Parameters: **window** : callable or string

A window function, or the name of a window function. Examples: - scipy.signal.hann - ‘boxcar’

n : int > 0

The number of coefficients to use in estimating the window bandwidth

Returns: **bandwidth** : float

The equivalent noise bandwidth (in FFT bins) of the given window function

See also

`get_window`

Notes

This function caches at level 10.

librosa.filters.get_window

`librosa.filters.get_window`(*window*, *Nx*, *fftbins=True*)[source]

Compute a window function.

This is a wrapper for `scipy.signal.get_window` that additionally supports callable or pre-computed windows.

Parameters: **window** : string, tuple, number, callable, or list-like

The window specification:

- If string, it’s the name of the window function (e.g., *‘hann’*)
- If tuple, it’s the name of the window function and any parameters (e.g., *(‘kaiser’, 4.0)*)
- If numeric, it is treated as the beta parameter of the *‘kaiser’* window, as in `scipy.signal.get_window`.
- If callable, it’s a function that accepts one integer argument (the window length)
- If list-like, it’s a pre-computed window of the correct length *Nx*

Nx : int > 0

The length of the window

fftbins : bool, optional

If True (default), create a periodic window for use with FFT If False, create a symmetric window for filter design applications.

Returns: **get_window** : np.ndarray

A window of length *Nx* and type *window*

Raises: ParameterError

If *window* is supplied as a vector of length != *n_fft*, or is otherwise mis-specified.

See also

`scipy.signal.get_window`

Notes

This function caches at level 10.

librosa.filters.constant_q_lengths

`librosa.filters.constant_q_lengths`(*sr*, *fmin*, *n_bins=84*, *bins_per_octave=12*, *tuning=<DEPRECATED parameter>*, *window='hann'*, *filter_scale=1*)[source]

Return length of each filter in a constant-Q basis.

Parameters: **sr** : number > 0 [scalar]

Audio sampling rate

fmin : float > 0 [scalar]

Minimum frequency bin.

n_bins : int > 0 [scalar]

Number of frequencies. Defaults to 7 octaves (84 bins).

bins_per_octave : int > 0 [scalar]

Number of bins per octave

tuning : float [scalar] <DEPRECATED>

Tuning deviation from A440 in fractions of a bin

Note

This parameter is deprecated in 0.7.1. It will be removed in version 0.8.

window : str or callable

Window function to use on filters

filter_scale : float > 0 [scalar]

Resolution of filter windows. Larger values use longer windows.

Returns: **lengths** : np.ndarray

The length of each filter.

See also

`constant_q`
`librosa.core.cqt`

Notes

This function caches at level 10.

librosa.filters.cq_to_chroma

`librosa.filters.cq_to_chroma`(*n_input*, *bins_per_octave=12*, *n_chroma=12*, *fmin=None*, *window=None*, *base_c=True*, *dtype=<class 'numpy.float32'>*)[source]

Convert a Constant-Q basis to Chroma.

Parameters: **n_input** : int > 0 [scalar]

Number of input components (CQT bins)

bins_per_octave : int > 0 [scalar]

How many bins per octave in the CQT

n_chroma : int > 0 [scalar]

Number of output bins (per octave) in the chroma

fmin : None or float > 0

Center frequency of the first constant-Q channel. Default: 'C1' ~= 32.7 Hz

window : None or np.ndarray

If provided, the cq_to_chroma filter bank will be convolved with *window*.

base_c : bool

If True, the first chroma bin will start at 'C' If False, the first chroma bin will start at 'A'

dtype : np.dtype

The data type of the output basis. By default, uses 32-bit (single-precision) floating point.

Returns: **cq_to_chroma** : np.ndarray [shape=(n_chroma, n_input)]

Transformation matrix: *Chroma = np.dot(cq_to_chroma, CQT)*

Raises: ParameterError

If *n_input* is not an integer multiple of *n_chroma*

Notes

This function caches at level 10.

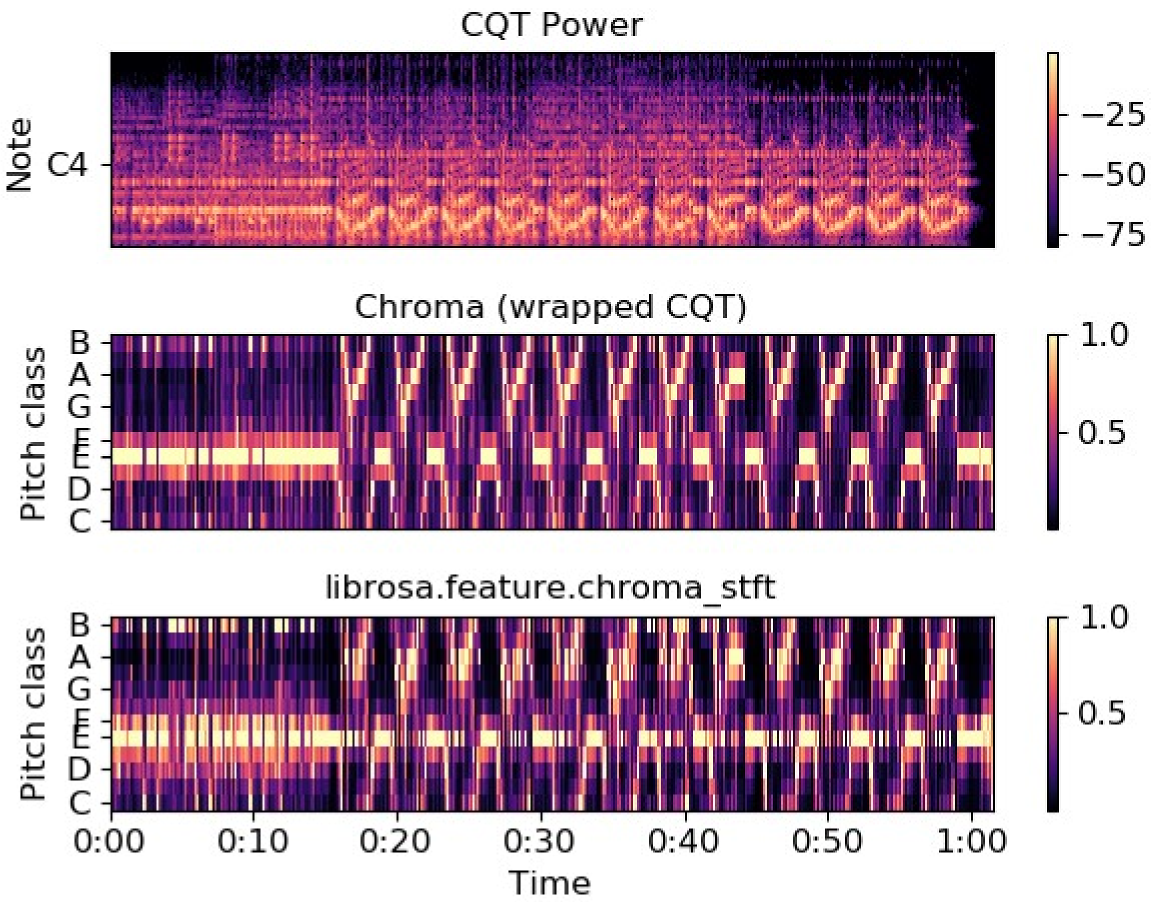
Examples

Get a CQT, and wrap bins to chroma

```
>>> y, sr = librosa.load(librosa.util.example_audio_file())
>>> CQT = np.abs(librosa.cqt(y, sr=sr))
>>> chroma_map = librosa.filters.cq_to_chroma(CQT.shape[0])
>>> chromagram = chroma_map.dot(CQT)
>>> # Max-normalize each time step
>>> chromagram = librosa.util.normalize(chromagram, axis=0)
```

```
>>> import matplotlib.pyplot as plt
>>> plt.subplot(3, 1, 1)
>>> librosa.display.specshow(librosa.amplitude_to_db(CQT,
...                                                  ref=np.max),
...                          y_axis='cqt_note')
>>> plt.title('CQT Power')
>>> plt.colorbar()
>>> plt.subplot(3, 1, 2)
>>> librosa.display.specshow(chromagram, y_axis='chroma')
>>> plt.title('Chroma (wrapped CQT)')
>>> plt.colorbar()
>>> plt.subplot(3, 1, 3)
```

```
>>> chroma = librosa.feature.chroma_stft(y=y, sr=sr)
>>> librosa.display.specshow(chroma, y_axis='chroma', x_axis='time')
>>> plt.title('librosa.feature.chroma_stft')
>>> plt.colorbar()
>>> plt.tight_layout()
>>> plt.show()
```



librosa.filters.mr_frequencies

librosa.filters.mr_frequencies(*tuning*)[source]

Helper function for generating center frequency and sample rate pairs.

This function will return center frequency and corresponding sample rates to obtain similar pitch filterbank settings as described in [1]. Instead of starting with MIDI pitch *A0*, we start with *C0*.

[1] Müller, Meinard. "Information Retrieval for Music and Motion." Springer Verlag. 2007.

Parameters: tuning : float [scalar]

Tuning deviation from A440, measure as a fraction of the equally tempered semitone (1/12 of an octave).

Returns: center_freqs : np.ndarray [shape=(n,), dtype=float]

Center frequencies of the filter kernels. Also defines the number of filters in the filterbank.

sample_rates : np.ndarray [shape=(n,), dtype=float]

Sample rate for each filter, used for multirate filterbank.

See also

`librosa.filters.semitone_filterbank`
`librosa.filters._multirate_fb`

librosa.filters.window_sumsquare

`librosa.filters.window_sumsquare`(*window*, *n_frames*, *hop_length=512*, *win_length=None*, *n_fft=2048*, *dtype=<class 'numpy.float32'>*, *norm=None*)[source]

Compute the sum-square envelope of a window function at a given hop length.

This is used to estimate modulation effects induced by windowing observations in short-time fourier transforms.

Parameters: **window** : string, tuple, number, callable, or list-like

Window specification, as in `get_window`

n_frames : int > 0

The number of analysis frames

hop_length : int > 0

The number of samples to advance between frames

win_length : [optional]

The length of the window function. By default, this matches *n_fft*.

n_fft : int > 0

The length of each analysis frame.

dtype : np.dtype

The data type of the output

Returns: **wss** : np.ndarray, shape=`(n_fft + hop_length * (n_frames - 1))`

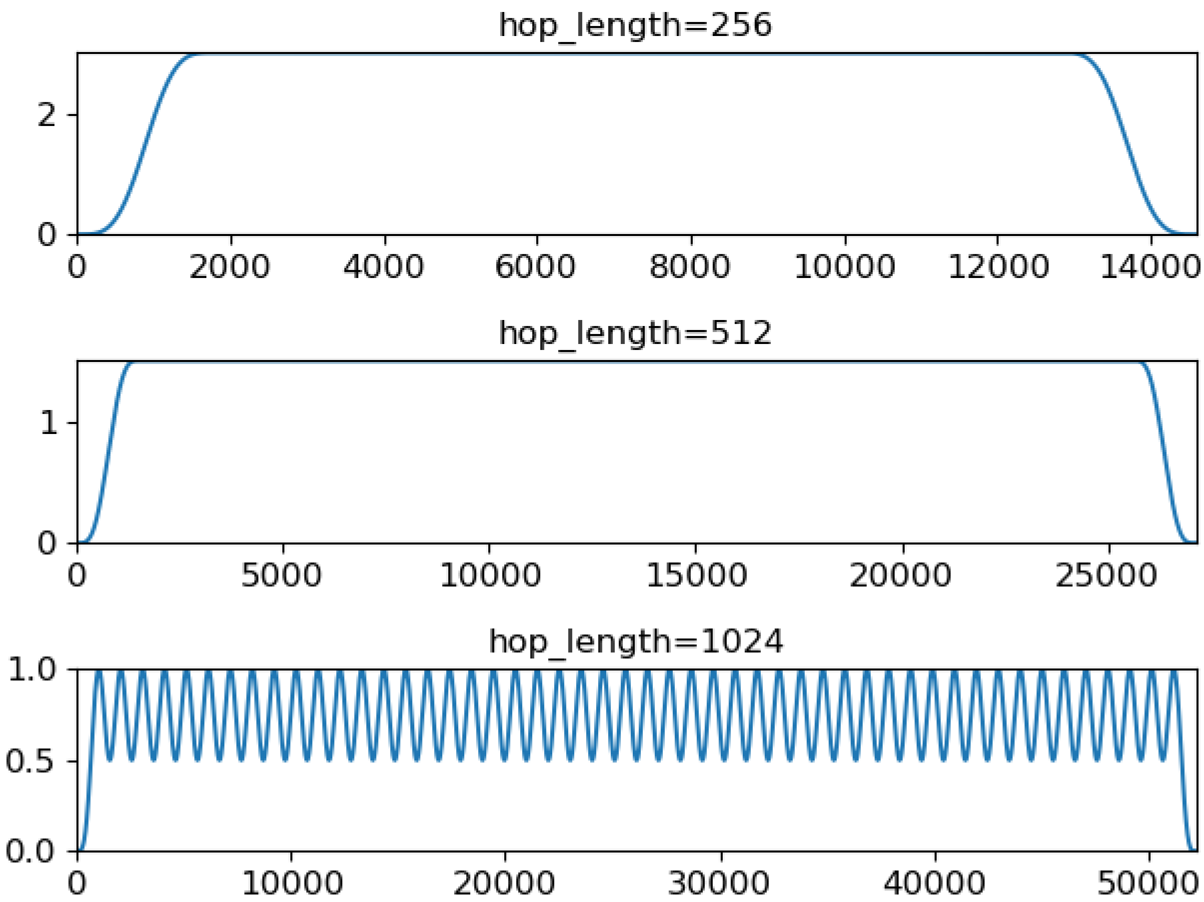
The sum-squared envelope of the window function

Examples

For a fixed frame length (2048), compare modulation effects for a Hann window at different hop lengths:

```
>>> n_frames = 50
>>> wss_256 = librosa.filters.window_sumsquare('hann', n_frames,
hop_length=256)
>>> wss_512 = librosa.filters.window_sumsquare('hann', n_frames,
hop_length=512)
>>> wss_1024 = librosa.filters.window_sumsquare('hann', n_frames,
hop_length=1024)
```

```
>>> import matplotlib.pyplot as plt
>>> plt.figure()
>>> plt.subplot(3,1,1)
>>> plt.plot(wss_256)
>>> plt.title('hop_length=256')
>>> plt.subplot(3,1,2)
>>> plt.plot(wss_512)
>>> plt.title('hop_length=512')
>>> plt.subplot(3,1,3)
>>> plt.plot(wss_1024)
>>> plt.title('hop_length=1024')
>>> plt.tight_layout()
>>> plt.show()
```



librosa.filters.diagonal_filter

`librosa.filters.diagonal_filter`(*window*, *n*, *slope=1.0*, *angle=None*, *zero_mean=False*)[source]

Build a two-dimensional diagonal filter.

This is primarily used for smoothing recurrence or self-similarity matrices.

Parameters: **window** : string, tuple, number, callable, or list-like

The window function to use for the filter.

See `get_window` for details.

Note that the window used here should be non-negative.

n : int > 0

the length of the filter

slope : float

The slope of the diagonal filter to produce

angle : float or None

If given, the slope parameter is ignored, and angle directly sets the orientation of the filter (in radians). Otherwise, angle is inferred as *arctan(slope)*.

zero_mean : bool

If True, a zero-mean filter is used. Otherwise, a non-negative averaging filter is used.

This should be enabled if you want to enhance paths and suppress blocks.

Returns: **kernel** : np.ndarray, shape=[(m, m)]

The 2-dimensional filter kernel

Caching

This section covers the *librosa* function cache. This allows you to store and re-use intermediate computations across sessions.

Enabling the cache

By default, caching is disabled. To enable caching, the environment variable *LIBROSA_CACHE_DIR* must be set prior to loading *librosa*. This can be done on the command line prior to instantiating a python interpreter:

```
$ export LIBROSA_CACHE_DIR=/tmp/librosa_cache
$ ipython
```

or from within python, prior to importing *librosa*:

```
>>> import os
>>> os.environ['LIBROSA_CACHE_DIR'] = '/tmp/librosa_cache'
>>> import librosa
```

Warning

The cache does not implement any eviction policy. As such, it can grow without bound on disk if not purged. To purge the cache directly, call:

```
>>> librosa.cache.clear()
```

Cache configuration

The cache is implemented on top of [joblib.Memory](). The default configuration can be overridden by setting the following environment variables

- *LIBROSA_CACHE_DIR* : path (on disk) to the cache directory

- *LIBROSA_CACHE_MMAP* : optional memory mapping mode *{None, 'r+', 'r', 'w+', 'c'}*
- *LIBROSA_CACHE_COMPRESS* : flag to enable compression of data on disk *{0, 1}*
- *LIBROSA_CACHE_VERBOSE* : controls how much debug info is displayed. *{int, non-negative}*
- *LIBROSA_CACHE_LEVEL* : controls the caching level: the larger this value, the more data is cached. *{int}*

Please refer to the `joblib.Memory` documentation for a detailed explanation of these parameters.

As of 0.7, librosa's cache wraps (rather than extends) the `joblib.Memory` object. The memory object can be directly accessed by `librosa.cache.memory`.

Cache levels

Cache levels operate in a fashion similar to logging levels. For small values of *LIBROSA_CACHE_LEVEL*, only the most important (frequently used) data are cached. As the cache level increases, broader classes of functions are cached. As a result, application code may run faster at the expense of larger disk usage.

The caching levels are described as follows:

- 10: filter bases, independent of audio data (mel, chroma, constant-q)
- 20: low-level features (cqt, stft, zero-crossings, etc)
- 30: high-level features (tempo, beats, decomposition, recurrence, etc)
- 40: post-processing (delta, stack_memory, normalize, sync)

The default cache level is 10.

Example

To demonstrate how to use the cache, we'll first call an example script twice without caching:

```
$ time -p ./estimate_tuning.py ../librosa/util/example_data/Kevin_MacLeod_-
_Vibe_Ace.ogg
Loading  ../librosa/util/example_data/Kevin_MacLeod_-_Vibe_Ace.ogg
Separating harmonic component ...
Estimating tuning ...
+9.00 cents
real 6.74
user 6.03
sys 1.09

$ time -p ./estimate_tuning.py ../librosa/util/example_data/Kevin_MacLeod_-
_Vibe_Ace.ogg
Loading  ../librosa/util/example_data/Kevin_MacLeod_-_Vibe_Ace.ogg
Separating harmonic component ...
Estimating tuning ...
+9.00 cents
real 6.68
user 6.04
sys 1.05
```

Next, we'll enable caching to */tmp/librosa*:

```
$ export LIBROSA_CACHE_DIR=/tmp/librosa
```

and set the cache level to 50:

```
$ export LIBROSA_CACHE_LEVEL=50
```

And now we'll re-run the example script twice. The first time, there will be no cached values, so the time should be similar to running without cache. The second time, we'll be able to reuse intermediate values, so it should be significantly faster.:

```
$ time -p ./estimate_tuning.py ../librosa/util/example_data/Kevin_MacLeod_-
_Vibe_Ace.ogg
Loading  ../librosa/util/example_data/Kevin_MacLeod_-_Vibe_Ace.ogg
Separating harmonic component ...
Estimating tuning ...
+9.00 cents
real 7.60
user 6.79
sys 1.15

$ time -p ./estimate_tuning.py ../librosa/util/example_data/Kevin_MacLeod_-
_Vibe_Ace.ogg
Loading  ../librosa/util/example_data/Kevin_MacLeod_-_Vibe_Ace.ogg
Separating harmonic component ...
Estimating tuning ...
+9.00 cents
real 1.64
user 1.30
sys 0.74
```

Reducing the cache level to 20 yields an intermediate acceleration:

```
$ export LIBROSA_CACHE_LEVEL=20

$ time -p ./estimate_tuning.py ../librosa/util/example_data/Kevin_MacLeod_-
_Vibe_Ace.ogg
Loading  ../librosa/util/example_data/Kevin_MacLeod_-_Vibe_Ace.ogg
Separating harmonic component ...
Estimating tuning ...
+9.00 cents
real 4.98
user 4.17
sys 1.22
```

Advanced I/O Use Cases

This section covers advanced use cases for input and output which go beyond the I/O functionality currently provided by *librosa*.

Read specific formats

librosa uses soundfile and audioread for reading audio. As of v0.7, librosa uses `soundfile` by default, and falls back on `audioread` only when dealing with codecs unsupported by `soundfile` (notably, MP3, and some variants of WAV). For a list of codecs supported by `soundfile`, see the *libsndfile* documentation.

Note

See installation instruction for PySoundFile here.

Librosa's load function is meant for the common case where you want to load an entire (fragment of a) recording into memory, but some applications require more flexibility. In these cases, we recommend using `soundfile` directly. Reading audio files using `soundfile` is similar to the method in *librosa*. One important difference is that the read data is of shape `(nb_samples, nb_channels)` compared to `(nb_channels, nb_samples)` in `librosa.core.load`. Also the signal is not resampled to 22050 Hz by default, hence it would need be transposed and resampled for further processing in *librosa*. The following example is equivalent to `librosa.load(librosa.util.example_audio_file())`:

```
import librosa
import soundfile as sf

# Get example audio file
filename = librosa.util.example_audio_file()

data, samplerate = sf.read(filename, dtype='float32')
data = data.T
data_22k = librosa.resample(data, samplerate, 22050)
```

Blockwise Reading

For large audio signals it could be beneficial to not load the whole audio file into memory. Librosa 0.7 introduces a streaming interface, which can be used to work on short fragments of audio sequentially. `librosa.core.stream` cuts an input file into *blocks* of audio, which correspond to a given number of *frames*, which can be iterated over as in the following example:

```
import librosa

sr = librosa.get_samplerate('/path/to/file.wav')

# Set the frame parameters to be equivalent to the librosa defaults
# in the file's native sampling rate
frame_length = (2048 * sr) // 22050
hop_length = (512 * sr) // 22050

# Stream the data, working on 128 frames at a time
stream = librosa.stream('path/to/file.wav',
                        block_length=128,
                        frame_length=frame_length,
                        hop_length=hop_length)

chromas = []
for y in stream:
   chroma_block = librosa.feature.chroma_stft(y=y, sr=sr,
                                              n_fft=frame_length,
                                              hop_length=hop_length,
                                              center=False)
   chromas.append(chromas)
```

In this example, each audio fragment `y` will consist of 128 frames worth of samples, or more specifically, `len(y) == frame_length + (block_length - 1) * hop_length`.

Each fragment `y` will overlap with the subsequent fragment by `frame_length - hop_length` samples, which ensures that stream processing will provide equivalent results to if the entire sequence was processed in one step (assuming padding / centering is disabled).

For more details about the streaming interface, refer to `librosa.core.stream`.

Read file-like objects

If you want to read audio from file-like objects (also called *virtual files*) you can use `soundfile` as well. (This will also work with `librosa.core.load` and `librosa.core.stream`, provided that the underlying codec is supported by `soundfile`.)

E.g.: read files from zip compressed archives:

```
import zipfile as zf
import soundfile as sf
import io

with zf.ZipFile('test.zip') as myzip:
    with myzip.open('stereo_file.wav') as myfile:
        tmp = io.BytesIO(myfile.read())
        data, samplerate = sf.read(tmp)
```

Warning

This is a example does only work in python 3. For python 2 please use `from urllib2 import urlopen`.

Download and read from URL:

```
import soundfile as sf
import io

from six.moves.urllib.request import urlopen

url = "https://raw.githubusercontent.com/librosa/librosa/master/tests/data/test1_44100.wav

data, samplerate = sf.read(io.BytesIO(urlopen(url).read()))
```

Write out audio files

PySoundFile provides output functionality that can be used directly with numpy array audio buffers:

```
import numpy as np
import soundfile as sf

rate = 44100
data = np.random.uniform(-1, 1, size=(rate * 10, 2))

# Write out audio as 24bit PCM WAV
sf.write('stereo_file.wav', data, samplerate, subtype='PCM_24')

# Write out audio as 24bit Flac
sf.write('stereo_file.flac', data, samplerate, format='flac', subtype='PCM_24')
```

```
# Write out audio as 16bit OGG
sf.write('stereo_file.ogg', data, samplerate, format='ogg', subtype='vorbis')
```

Changelog

v0.7.1

2019-10-09

New Features

- #966 *util.frame* now supports multidimensional data. Includes a new helper function *util.stack* for contiguous concatenation. *Brian McFee*
- #934 *core.griffinlim_cqt*: Phase retrieval from constant-Q magnitude spectra. *Brian McFee*
- #940 Enhanced compatibility with HTK's MFCC implementation: *effects.preemphasis* and *lifter=* parameter in MFCC. *Brian McFee*
- #949 *util.shear* utility for shear-transforming 2D arrays. *Brian McFee*
- #926 *core.reassigned_spectrogram*: time-frequency reassigned spectrogram. *Scott Seyfarth*

Bug fixes

- #983 Added a missing parameter to *griffinlim_cqt*. *Voodoohop*
- #978 Correct FFT normalization discrepancy in rms calculation. *Brian McFee*
- #930 Corrected an error in automatic tuning correction for CQT. *Brian McFee*
- #942 Fixed seeking behavior in `librosa.stream` when operating on file-handle inputs. *Carl Thome*
- #920 Fixed a boundary condition check in full-sequence DTW. *Frank Zalkow*

Documentation

- #976 Fixed a typo in *stream* documentation. *Alastair Porter*
- #958 Visualization of reassigned spectrograms. *Vincent Lostanlen*
- #943 Improved documentation for *core.stft*. *Vincent Lostanlen*
- #939 Expanded documentation of *feature.melspectrogram*. *Vincent Lostanlen*

Other changes

- #1004 Expose frame parameters in *onset* and *chroma_stft* functions. *Brian McFee*
- #1003 Removed warning filter reset, and changed the Python2 deprecation warning to class *FutureWarning*. *Brian McFee, Fabian Keller*
- #1000 Fixed an upstream deprecation warning from *np.asscalar()*. *Vincent Lostanlen*
- #971 Beat and tempo estimation now support prior distributions. *Brian McFee*
- #968 *util.valid_audio* now enforces memory contiguity. *Vincent Lostanlen*
- #963 Improved x-axis decoration types in *display.waveplot*. *Vincent Lostanlen*
- #960 Ensure memory contiguity of audio buffers after resampling. *Brian McFee*
- #957 Code-base audit for proper usage of *times_like*. *Brian McFee*
- #953 Deprecated *core.ifgram* in favor of *reassigned_spectrogram*. *Brian McFee*
- #950 Allow initial phase estimates for *griffinlim* methods. *Brian McFee*
- #949 Accelerated *segment.lag_to_recurrence* and *segment.recurrence_to_lag*. Deprecated *util.roll_sparse*. *Brian McFee*
- #930 *A440=* parameter has been deprecated across the library in favor of a standardized *tuning=* parameter. *Brian McFee*

v0.7.0

2019-07-07

Note: the 0.7 series will be the last to officially support Python 2.7.

New features

- #772 `librosa.core.stream`: Stream generator to process long audio files into smaller pieces. *Brian McFee*
- #845 `librosa.core.load`: Replaced the default audio decoder with *pysoundfile*, and only use `audioread` as backup. *Brian McFee*
- #843 `librosa.core.griffinlim`: Phase retrieval from magnitude spectrograms using the (accelerated) Griffin-Lim method. *Brian McFee*
- #843 `librosa.feature.inverse`: New module for feature inversion, based on the Griffin-Lim phase retrieval algorithm. Includes *mel_to_audio* and *mfcc_to_audio*. *Brian McFee*
- #725 `librosa.core.lpc`: Linear prediction coefficients (LPC). *Adam Weiss*
- #907 `librosa.sequence.rqa`: Recurrence Quantification Analysis (RQA) for sequence alignment. *Brian McFee*
- #739 `librosa.beat.plp`: Predominant local pulse (PLP) for variable-tempo beat tracking. *Brian McFee*
- #894 `librosa.feature.fourier_tempogram`: Fourier Tempogram for representing rhythm in the frequency domain. *Brian McFee*
- #891 `librosa.core.pcen` Per-channel energy normalization (PCEN) now allows logarithmic range compression at the limit power->0. *Vincent Lostanlen*
- #863 `librosa.effects.pitch_shift` supports custom resampling modes. *Taewoon Kim*
- #857 `librosa.core.cqt` and `librosa.core.icqt` Forward and inverse constant-Q transform now support custom resampling modes. *Brian McFee*
- #842 `librosa.segment.path_enhance`: Near-diagonal path enhancement for recurrence, self- or cross-similarity matrices. *Brian McFee*
- #840 `librosa.segment.recurrence_matrix` now supports a keyword argument, *self=False*. If set to *True*, the recurrence matrix includes self-loops. *Brian McFee*
- #776 `librosa.core.piptrack` now supports a keyword argument, *ref=None*, allowing users to override the reference thresholding behavior for determining which bins correspond to pitches. *Brian McFee*
- #770 `librosa.segment.cross_similarity`: Cross-similarity function for comparing two feature sequences. *Rachel Bittner, Brian McFee*
- #709 `librosa.onset.onset_strength_multi` now supports a user-specified reference spectrum via the *ref* keyword argument. *Brian McFee*
- #576 `librosa.core.resample` now supports *mode='polyphase'*. *Brian McFee*
- #519 `librosa.onset.onset_strength_multi`: Setting *aggregate=False* disables the aggregation of onset strengths across frequency bins. *Brian McFee*

Bug fixes

- #900 `librosa.effects.pitch_shift` now preserves length. *Vincent Lostanlen*
- #891 `librosa.core.pcen` Dynamic range compression in PCEN is more numerically stable for small values of the exponent. *Vincent Lostanlen*
- #888 `librosa.core.ifgram` Instantaneous frequency spectrogram now correctly estimates center frequencies when using windows other than *hann*. *Brian McFee*

- #869 `librosa.sequence.dtw` Fixed a bug in dynamic time warping when *subseq=True*. *Viktor Andreevitch Morozov*
- #851 `librosa.core.pcen` now initializes its autoregressive filtering in the steady state, not with silence. *Jan Schlüter, Brian McFee*
- #833 `librosa.segment.recurrence_matrix`: *width* parameter now cannot exceed data length. *Brian McFee*
- #825 Filter bank constructors *mel*, *chroma*, *constant_q*, and *cq_to_chroma* are now type-stable. *Vincent Lostanlen, Brian McFee*
- #802 `librosa.core.icqt` Inverse constant-Q transform has been completely rewritten and is more numerically stable. *Brian McFee*

Removed features (deprecated in v0.6)

- Discrete cosine transform. We recommend using `scipy.fftpack.dct`
- The *delta* function no longer support the *trim* keyword argument.
- Root mean square error (*rmse*) has been renamed to *rms*.
- *iirt* now uses *sos* mode by default.

Documentation

- #891 Improved the documentation of PCEN. *Vincent Lostanlen*
- #884 Improved installation documentation. *Darío Hereñú*
- #882 Improved code style for plot generation. *Alex Metsai*
- #874 Improved the documentation of spectral features. *Brian McFee*
- #804 Improved the documentation of MFCC. *Brian McFee*
- #849 Removed a redundant link in the *util* documentation. *Keunwoo Choi*
- #827 Improved the docstring of *recurrence_matrix*. *Brian McFee*
- #813 Improved the docstring of *load*. *Andy Sarroff*

Other changes

- #917 The *output* module is now deprecated, and will be removed in version 0.8.
- #878 More informative exception handling. *Jack Mason*
- #857 *librosa.core.resample()* now supports *mode='fft'*, equivalent to the previous `scipy` mode. *Brian McFee*
- #854 More efficient length-aware ISTFT and ICQT. *Vincent Lostanlen*
- #846 Nine librosa functions now store jit-compiled, numba-accelerated caches across sessions. *Brian McFee*
- #841 `librosa.core.load` no longer relies on *realpath()*. *Brian McFee*
- #834 All spectral feature extractors now expose all STFT parameters. *Brian McFee*
- #829 Refactored `librosa.cache`. *Brian McFee*
- #818 Thanks to *np.fft.rfft*, functions *stft*, *istft*, *ifgram*, and *fmt* are faster and have a reduced memory footprint. *Brian McFee*

v0.6.3

2019-02-13

Bug fixes

- #806 Fixed a bug in *estimate_tuning*. *@robrib, Monsij Biswal, Brian McFee*
- #799 Enhanced stability of elliptical filter implementation in *iirt*. *Frank Zalkow*

New features

- #766 made smoothing optional in *feature.chroma_cens*. *Kyungyun Lee*
- #760 allow explicit units for time axis decoration in *display*. *Kyungyun Lee*

Other changes

- #813 updated *core.load* documentation to cover bit depth truncation. *Andy Sarroff*
- #805 updated documentation for *core.localmax*. *Brian McFee*

- #801 renamed *feature.rmse* to *feature.rms*. *@nullmightybofo*
- #793 updated comments in *stft*. *Dan Ellis*
- #791 updated documentation for *write_wav*. *Brian McFee*
- #790 removed dependency on deprecated `imp` module. *Brian McFee*
- #787 fixed typos in CONTRIBUTING documentation. *Vincent Lostanlen*
- #785 removed all run-time assertions in favor of proper exceptions. *Brian McFee*
- #783 migrated test infrastructure from `nose` to `pytest`. *Brian McFee*
- #777 include LICENSE file in source distribution. *toddrme2178*
- #769 updated documentation in *core.istft*. *Shayenne Moura*

v0.6.2

2018-08-09

Bug fixes

- #730 Fixed cache support for `joblib>=0.12`. *Matt Vollrath*

New features

- #735 Added *core.times_like* and *core.samples_like* to generate time and sample indices corresponding to an existing feature matrix or shape specification. *Steve Tjoa*
- #750, #753 Added *core.tone* and *core.chirp* signal generators. *Ziyao Wei*

Other changes

- #727 updated documentation for *core.get_duration*. *Zhen Wang*
- #731 fixed a typo in documentation for *core.fft_frequencies*. *Ziyao Wei*
- #734 expanded documentation for *feature.spectrall_rolloff*. *Ziyao Wei*
- #751 fixed example documentation for proper handling of phase in dB-scaling. *Vincent Lostanlen*
- #755 forward support and future-proofing for fancy indexing with `numpy>1.15`. *Brian McFee*

v0.6.1

2018-05-24

Bug fixes

- #677 *util.find_files* now correctly de-duplicates files on case-insensitive platforms. *Brian McFee*
- #713 *util.valid_intervals* now checks for non-negative durations. *Brian McFee, Dana Lee*
- #714 *util.match_intervals* can now explicitly fail when no matches are possible. *Brian McFee, Dana Lee*

New features

- #679, #708 *core.pcen*, per-channel energy normalization. *Vincent Lostanlen, Brian McFee*
- #682 added different DCT modes to *feature.mfcc*. *Brian McFee*
- #687 *display* functions now accept target axes. *Pius Friesch*
- #688 numba-accelerated *util.match_events*. *Dana Lee*
- #710 *sequence* module and Viterbi decoding for generative, discriminative, and multi-label hidden Markov models. *Brian McFee*
- #714 *util.match_intervals* now supports tie-breaking for disjoint query intervals. *Brian McFee*

Other changes

- #677, #705 added continuous integration testing for Windows. *Brian McFee*, *Ryuichi Yamamoto*
- #680 updated display module tests to support matplotlib 2.1. *Brian McFee*
- #684 corrected documentation for *core.stft* and *core.ifgram*. *Keunwoo Choi*
- #699, #701 corrected documentation for *filters.semitone_filterbank* and *filters.mel_frequencies*. *Vincent Lostanlen*
- #704 eliminated unnecessary side-effects when importing *display*. *Brian McFee*
- #707 improved test coverage for dynamic time warping. *Brian McFee*
- #714 *util.match_intervals* matching logic has changed from raw intersection to Jaccard similarity. *Brian McFee*

API Changes and compatibility

- #716 *core.dtw* has moved to *sequence.dtw*, and *core.fill_off_diagonal* has moved to *util.fill_off_diagonal*. *Brian McFee*

v0.6.0

2018-02-17

Bug fixes

- #663 fixed alignment errors in *feature.delta*. *Brian McFee*
- #646 *effects.trim* now correctly handles all-zeros signals. *Rimvydas Naktinis*
- #634 *stft* now conjugates the correct half of the spectrum. *Brian McFee*
- #630 fixed display decoration errors with *cqt_note* mode. *Brian McFee*
- #619 *effects.split* no longer returns out-of-bound sample indices. *Brian McFee*
- #616 Improved *util.valid_audio* to avoid integer type errors. *Brian McFee*
- #600 CQT basis functions are now correctly centered. *Brian McFee*
- #597 fixed frequency bin centering in *display.specshow*. *Brian McFee*
- #594 *dtw* fixed a bug which ignored weights when *step_sizes_sigma* did not match length. *Jackie Wu*
- #593 *stft* properly checks for valid input signals. *Erik Peterson*
- #587 *show_versions* now shows correct module names. *Ryuichi Yamamoto*

New features

- #648 *feature.spectral_flatness*. *Keunwoo Choi*
- #633 *feature.tempogram* now supports multi-band analysis. *Brian McFee*
- #439 *core.iirt* implements the multi-rate filterbank from Chroma Toolbox. *Stefan Balke*
- #435 *core.icqt* inverse constant-Q transform (unstable). *Brian McFee*

Other changes

- #674 Improved *write_wav* documentation with cross-references to `soundfile`. *Brian McFee*
- #671 Warn users when phase information is lost in dB conversion. *Carl Thome*
- #666 Expanded documentation for *load*'s resampling behavior. *Brian McFee*
- #656 Future-proofing numpy data type checks. *Carl Thome*
- #642 Updated unit tests for compatibility with matplotlib 2.1. *Brian McFee*
- #637 Improved documentation for advanced I/O. *Siddhartha Kumar*
- #636 *util.normalize* now preserves data type. *Brian McFee*
- #632 refined the validation requirements for *util.frame*. *Brian McFee*
- #628 all time/frequency conversion functions preserve input shape. *Brian McFee*
- #625 Numba is now a hard dependency. *Brian McFee*
- #622 *hz_to_midi* documentation corrections. *Carl Thome*
- #621 *dtw* is now symmetric with respect to input arguments. *Stefan Balke*

- #620 Updated requirements to prevent installation with (incompatible) sklearn 0.19.0. *Brian McFee*
- #609 Improved documentation for *segment.recurrence_matrix*. *Julia Wilkins*
- #598 Improved efficiency of *decompose.nn_filter*. *Brian McFee*
- #574 *dtw* now supports pre-computed distance matrices. *Curtis Hawthorne*

API changes and compatibility

- #627 The following functions and features have been removed:
    - *real*= parameter in *cqt*
    - *core.logamplitude* (replaced by *amplitude_to_db*)
    - *beat.estimate_tempo* (replaced by *beat.tempo*)
    - *n_fft*= parameter to *feature.rmse*
    - *ref_power*= parameter to *power_to_db*
- The following features have been deprecated, and will be removed in 0.7.0:
    - *trim*= parameter to *feature.delta*
- #616 *write_wav* no longer supports integer-typed waveforms. This is due to enforcing consistency with *util.valid_audio* checks elsewhere in the codebase. If you have existing code that requires integer-valued output, consider using `soundfile.write` instead.

v0.5.1

2017-05-08

Bug fixes

- #555 added safety check for frequency bands in *spectral_contrast*. *Brian McFee*
- #554 fix interactive display for *tonnetz* visualization. *Brian McFee*
- #553 fix bug in *feature.spectral_bandwidth*. *Brian McFee*
- #539 fix *chroma_cens* to support scipy >=0.19. *Brian McFee*

New features

- #565 *feature.stack_memory* now supports negative delay. *Brian McFee*
- #563 expose padding mode in *stft/ifgram/cqt*. *Brian McFee*
- #559 explicit length option for *istft*. *Brian McFee*
- #557 added *show_versions*. *Brian McFee*
- #551 add *norm*= option to *filters.mel*. *Dan Ellis*

Other changes

- #569 *feature.rmse* now centers frames in the time-domain by default. *Brian McFee*
- #564 *display.specshow* now rasterizes images by default. *Brian McFee*
- #558 updated contributing documentation and issue templates. *Brian McFee*
- #556 updated tutorial for 0.5 API compatibility. *Brian McFee*
- #544 efficiency improvement in CQT. *Carl Thome*
- #523 support reading files with more than two channels. *Paul Brossier*

v0.5.0

2017-02-17

Bug fixes

- #371 preserve integer hop lengths in constant-Q transforms. *Brian McFee*
- #386 fixed a length check in `librosa.util.frame`. *Brian McFee*
- #416 `librosa.output.write_wav` only normalizes floating point, and normalization is disabled by default. *Brian McFee*

- #417 `librosa.cqt` output is now scaled continuously across octave boundaries. *Brian McFee*, *Eric Humphrey*
- #450 enhanced numerical stability for `librosa.util.softmask`. *Brian McFee*
- #467 correction to chroma documentation. *Seth Kranzler*
- #501 fixed a numpy 1.12 compatibility error in `pitch_tuning`. *Hojin Lee*

New features

- #323 `librosa.dtw` dynamic time warping. *Stefan Balke*
- #404 `librosa.cache` now supports priority levels, analogous to logging levels. *Brian McFee*
- #405 `librosa.interp_harmonics` for estimating harmonics of time-frequency representations. *Brian McFee*
- #410 `librosa.beat.beat_track` and `librosa.onset.onset_detect` can return output in frames, samples, or time units. *Brian McFee*
- #413 full support for scipy-style window specifications. *Brian McFee*
- #427 `librosa.salience` for computing spectrogram salience using harmonic peaks. *Rachel Bittner*
- #428 `librosa.effects.trim` and `librosa.effects.split` for trimming and splitting waveforms. *Brian McFee*
- #464 `librosa.amplitude_to_db`, `db_to_amplitude`, `power_to_db`, and `db_to_power` for amplitude conversions. This deprecates `logamplitude`. *Brian McFee*
- #471 `librosa.util.normalize` now supports `threshold` and `fill_value` arguments. *Brian McFee*
- #472 `librosa.feature.melspectrogram` now supports `power` argument. *Keunwoo Choi*
- #473 `librosa.onset.onset_backtrack` for backtracking onset events to previous local minima of energy. *Brian McFee*
- #479 `librosa.beat.tempo` replaces `librosa.beat.estimate_tempo`, supports time-varying estimation. *Brian McFee*

Other changes

- #352 removed `seaborn` integration. *Brian McFee*
- #368 rewrite of the `librosa.display` submodule. All plots are now in natural coordinates. *Brian McFee*
- #402 `librosa.display` submodule is not automatically imported. *Brian McFee*
- #403 `librosa.decompose.hpss` now returns soft masks. *Brian McFee*
- #407 `librosa.feature.rmse` can now compute directly in the time domain. *Carl Thome*
- #432 `librosa.feature.rmse` renames `n_fft` to `frame_length`. *Brian McFee*
- #446 `librosa.cqt` now disables tuning estimation by default. *Brian McFee*
- #452 `librosa.filters.__float_window` now always uses integer length windows. *Brian McFee*
- #459 `librosa.load` now supports `res_type` argument for resampling. *CJ Carr*
- #482 `librosa.filters.mel` now warns if parameters will generate empty filter channels. *Brian McFee*
- #480 expanded documentation for advanced IO use-cases. *Fabian Robert-Stoeter*

API changes and compatibility

- The following functions have permanently moved:
  - `core.peak_peak` to `util.peak_pick`
  - `core.localmax` to `util.localmax`

        - `feature.sync` to `util.sync`
    - The following functions, classes, and constants have been removed:
        - `core.ifptrack`
        - `feature.chromagram`
        - `feature.logfsgram`
        - `filters.logfrequency`
        - `output.frames_csv`
        - `segment.structure_Feature`
        - `display.time_ticks`
        - `util.FeatureExtractor`
        - `util.buf_to_int`
        - `util.SMALL_FLOAT`
    - The following parameters have been removed:
        - `librosa.cqt`: *resolution*
        - `librosa.cqt`: *aggregate*
        - `feature.chroma_cqt`: *mode*
        - `onset_strength`: *centering*
    - Seaborn integration has been removed, and the `display` submodule now requires matplotlib >= 1.5.
        - The *use_sns* argument has been removed from *display.cmap*
        - *magma* is now the default sequential colormap.
    - The `librosa.display` module has been rewritten.
        - `librosa.display.specshow` now plots using *pcolormesh*, and supports non-uniform time and frequency axes.
        - All plots can be rendered in natural coordinates (e.g., time or Hz)
        - Interactive plotting is now supported via ticker and formatter objects
- `librosa.decompose.hpss` with *mask=True* now returns soft masks, rather than binary masks.
- `librosa.filters.get_window` wraps `scipy.signal.get_window`, and handles generic callables as well pre-registered window functions. All windowed analyses (e.g., `stft`, `cqt`, or `tempogram`) now support the full range of window functions and parametric windows via tuple parameters, e.g., `window=('kaiser', 4.0)`.
- `stft` windows are now explicitly asymmetric by default, which breaks backwards compatibility with the 0.4 series.
- `cqt` now returns properly scaled outputs that are continuous across octave boundaries. This breaks backwards compatibility with the 0.4 series.
- `cqt` now uses *tuning=0.0* by default, rather than estimating the tuning from the signal. Tuning estimation is still supported, and enabled by default for chroma analysis (`librosa.feature.chroma_cqt`).
- `logamplitude` is deprecated in favor of `amplitude_to_db` or `power_to_db`. The *ref_power* parameter has been renamed to *ref*.

v0.4.3

2016-05-17

Bug fixes

- #315 fixed a positioning error in `display.specshow` with logarithmic axes. *Brian McFee*
- #332 `librosa.cqt` now throws an exception if the signal is too short for analysis. *Brian McFee*
- #341 `librosa.hybrid_cqt` properly matches the scale of `librosa.cqt`. *Brian McFee*
- #348 `librosa.cqt` fixed a bug introduced in v0.4.2. *Brian McFee*
- #354 Fixed a minor off-by-one error in `librosa.beat.estimate_tempo`. *Brian McFee*
- #357 improved numerical stability of `librosa.decompose.hpss`. *Brian McFee*

New features

- #312 `librosa.segment.recurrence_matrix` can now construct sparse self-similarity matrices. *Brian McFee*
- #337 `librosa.segment.recurrence_matrix` can now produce weighted affinities and distances. *Brian McFee*
- #311 `librosa.decompose.nl_filter` implements several self-similarity based filtering operations including non-local means. *Brian McFee*
- #320 `librosa.feature.chroma_cens` implements chroma energy normalized statistics (CENS) features. *Stefan Balke*
- #354 `librosa.core.tempo_frequencies` computes tempo (BPM) frequencies for autocorrelation and tempogram features. *Brian McFee*
- #355 `librosa.decompose.hpss` now supports harmonic-percussive-residual separation. *CJ Carr, Brian McFee*
- #357 `librosa.util.softmask` computes numerically stable soft masks. *Brian McFee*

Other changes

- `librosa.cqt`, `librosa.hybrid_cqt` parameter *aggregate* is now deprecated.
- Resampling is now handled by the `resampy` library
- `librosa.get_duration` can now operate directly on filenames as well as audio buffers and feature matrices.
- `librosa.decompose.hpss` no longer supports `power=0`.

v0.4.2

2016-02-20

Bug fixes

- Support for matplotlib 1.5 color properties in the `display` module
- #308 Fixed a per-octave scaling error in `librosa.cqt`. *Brian McFee*

New features

- #279 `librosa.cqt` now provides complex-valued output with argument *real=False*. This will become the default behavior in subsequent releases.
- #288 `core.resample` now supports multi-channel inputs. *Brian McFee*
- #295 `librosa.display.frequency_ticks`: like `time_ticks`. Ticks can now dynamically adapt to scale (mHz, Hz, KHz, MHz, GHz) and use automatic precision formatting (`%g`). *Brian McFee*

Other changes

- #277 improved documentation for OSX. *Stefan Balke*
- #294 deprecated the `FeatureExtractor` object. *Brian McFee*
- #300 added dependency version requirements to install script. *Brian McFee*

    - #302, #279 renamed the following parameters
        - `librosa.display.time_ticks`: *fmt* is now *time_fmt*
        - `librosa.feature.chroma_cqt`: *mode* is now *cqt_mode*
        - `librosa.cqt`, `hybrid_cqt`, `pseudo_cqt`, `librosa.filters.constant_q`: *resolution* is now *filter_scale*
- #308 `librosa.cqt` default *filter_scale* parameter is now 1 instead of 2.

v0.4.1

2015-10-17

Bug fixes

- Improved safety check in CQT for invalid hop lengths
- Fixed division by zero bug in `core.pitch.pip_track`
- Fixed integer-type error in `util.pad_center` on numpy v1.10
- Fixed a context scoping error in `librosa.load` with some audioread backends
- `librosa.autocorrelate` now persists type for complex input

New features

- `librosa.clicks` sonifies timed events such as beats or onsets
- `librosa.onset.onset_strength_multi` computes onset strength within multiple sub-bands
- `librosa.feature.tempogram` computes localized onset strength autocorrelation
- `librosa.display.specshow` now supports `*_axis='tempo'` for annotating tempo-scaled data
- `librosa.fmt` implements the Fast Mellin Transform

Other changes

- Rewrote `display.waveplot` for improved efficiency
- `decompose.deompose()` now supports pre-trained transformation objects
- Nullified side-effects of optional seaborn dependency
- Moved `feature.sync` to `util.sync` and expanded its functionality
- `librosa.onset.onset_strength` and `onset_strength_multi` support superflux-style lag and max-filtering
- `librosa.core.autocorrelate` can now operate along any axis of multi-dimensional input
- the `segment` module functions now support arbitrary target axis
- Added proper window normalization to `librosa.core.istft` for better reconstruction (PR #235).
- Standardized `n_fft=2048` for `piptrack`, `ifptrack` (deprecated), and `logfsgram` (deprecated)
- `onset_strength` parameter `'centering'` has been deprecated and renamed to `'center'`
- `onset_strength` always trims to match the input spectrogram duration
- added tests for `piptrack`
- added test support for Python 3.5

v0.4.0

2015-07-08

Bug fixes

- Fixed alignment errors with `offset` and `duration` in `load()`
- Fixed an edge-padding issue with `decompose.hpss()` which resulted in percussive noise leaking into the harmonic component.
- Fixed stability issues with `ifgram()`, added options to suppress negative frequencies.
- Fixed scaling and padding errors in `feature.delta()`
- Fixed some errors in `note_to_hz()` string parsing
- Added robust range detection for `display.cmap`
- Fixed tick placement in `display.specshow`
- Fixed a low-frequency filter alignment error in `cqt`
- Added aliasing checks for `cqt` filterbanks
- Fixed corner cases in `peak_pick`
- Fixed bugs in `find_files()` with negative slicing
- Fixed tuning estimation errors
- Fixed octave numbering in to conform to scientific pitch notation

New features

- python 3 compatibility
- Deprecation and moved-function warnings
- added `norm=None` option to `util.normalize()`
- `segment.recurrence_to_lag`, `lag_to_recurrence`
- `core.hybrid_cqt()` and `core.pseudo_cqt()`
- `segment.timelag_filter`
- Efficiency enhancements for `cqt`
- Major rewrite and reformatting of documentation
- Improvements to `display.specshow`: - added the `lag` axis format - added the `tonnetz` axis format - allow any combination of axis formats
- `effects.remix()`
- Added new time and frequency converters: - `note_to_hz()`, `hz_to_note()` - `frames_to_samples()`, `samples_to_frames()` - `time_to_samples()`, `samples_to_time()`
- `core.zero_crossings`
- `util.match_events()`
- `segment.subsegment()` for segmentation refinement
- Functional examples in almost all docstrings
- improved numerical stability in `normalize()`
- audio validation checks
- `to_mono()`
- `librosa.cache` for storing pre-computed features
- Stereo output support in `write_wav`
- Added new feature extraction functions: - `feature.spectral_contrast` - `feature.spectral_bandwidth` - `feature.spectral_centroid` - `feature.spectral_rolloff` - `feature.poly_features` - `feature.rmse` - `feature.zero_crossing_rate` - `feature.tonnetz`
- Added `display.waveplot`

Other changes

- Internal refactoring and restructuring of submodules
- Removed the `chord` module
- input validation and better exception reporting for most functions
- Changed the default colormaps in `display`
- Changed default parameters in onset detection, beat tracking
- Changed default parameters in `cqt`
- `filters.constant_q` now returns filter lengths
- Chroma now starts at `C` by default, instead of `A`
- `pad_center` supports multi-dimensional input and `axis` parameter
- switched from `np.fft` to `scipy.fftpack` for FFT operations
- changed all librosa-generated exception to a new class librosa.ParameterError

Deprecated functions

- `util.buf_to_int`
- `output.frames_csv`
- `segment.structure_feature`
- `filters.logfrequency`
- `feature.logfsgram`

v0.3.1

2015-02-18

Bug fixes

- Fixed bug #117: `librosa.segment.agglomerative` now returns a numpy.ndarray instead of a list
- Fixed bug #115: off-by-one error in `librosa.core.load` with fixed duration
- Fixed numerical underflow errors in `librosa.decompose.hpss`
- Fixed bug #104: `librosa.decompose.hpss` failed with silent, complex-valued input
- Fixed bug #103: `librosa.feature.estimate_tuning` fails when no bins exceed the threshold

Features

- New function `librosa.core.get_duration()` computes the duration of an audio signal or spectrogram-like input matrix
- `librosa.util.pad_center` now accepts multi-dimensional input

Other changes

- Adopted the ISC license
- Python 3 compatibility via futurize
- Fixed issue #102: segment.agglomerative no longer depends on the deprecated Ward module of sklearn; it now depends on the newer Agglomerative module.
- Issue #108: set character encoding on all source files
- Added dtype persistence for resample, stft, istft, and effects functions

v0.3.0

2014-06-30

Bug fixes

- Fixed numpy array indices to force integer values
- `librosa.util.frame` now warns if the input data is non-contiguous

- Fixed a formatting error in `librosa.display.time_ticks()`
- Added a warning if `scikits.samplerate` is not detected

Features

- New module `librosa.chord` for training chord recognition models
- Parabolic interpolation piptracking `librosa.feature.piptrack()`
- `librosa.localmax()` now supports multi-dimensional slicing
- New example scripts
- Improved documentation
- Added the `librosa.util.FeatureExtractor` class, which allows librosa functions to act as feature extraction stages in `sklearn`
- New module `librosa.effects` for time-domain audio processing
- Added demo notebooks for the `librosa.effects` and `librosa.util.FeatureExtractor`
- Added a full-track audio example, `librosa.util.example_audio_file()`
- Added peak-frequency sorting of basis elements in `librosa.decompose.decompose()`

Other changes

- Spectrogram frames are now centered, rather than left-aligned. This removes the need for window correction in `librosa.frames_to_time()`
- Accelerated constant-Q transform `librosa.cqt()`
- PEP8 compliance
- Removed normalization from `librosa.feature.logfsgram()`
- Efficiency improvements by ensuring memory contiguity
- `librosa.logamplitude()` now supports functional reference power, in addition to scalar values
- Improved `librosa.feature.delta()`
- Additional padding options to `librosa.feature.stack_memory()`
- `librosa.cqt` and `librosa.feature.logfsgram` now use the same parameter formats `(fmin, n_bins, bins_per_octave)`.
- Updated demo notebook(s) to IPython 2.0
- Moved `perceptual_weighting()` from `librosa.feature` into `librosa.core`
- Moved `stack_memory()` from `librosa.segment` into `librosa.feature`
- Standardized `librosa.output.annotation` input format to match `mir_eval`
- Standardized variable names (e.g., `onset_envelope`).

v0.2.1

2014-01-21

Bug fixes

- fixed an off-by-one error in `librosa.onset.onset_strength()`
- fixed a sign-flip error in `librosa.output.write_wav()`
- removed all mutable object default parameters

Features

- added option `centering` to `librosa.onset.onset_strength()` to resolve frame-centering issues with sliding window STFT
- added frame-center correction to `librosa.core.frames_to_time()` and `librosa.core.time_to_frames()`
- added `librosa.util.pad_center()`

- added `librosa.output.annotation()`
- added `librosa.output.times_csv()`
- accelerated `librosa.core.stft()` and `ifgram()`
- added `librosa.util.frame` for in-place signal framing
- `librosa.beat.beat_track` now supports user-supplied tempo
- added `librosa.util.normalize()`
- added `librosa.util.find_files()`
- added `librosa.util.axis_sort()`
- new module: `librosa.util()`
- `librosa.filters.constant_q` now support padding
- added boolean input support for `librosa.display.cmap()`
- speedup in `librosa.core.cqt()`

Other changes

- optimized default parameters for `librosa.onset.onset_detect`
- set `librosa.filters.mel` parameter `n_mels=128` by default
- `librosa.feature.chromagram()` and `logfsgram()` now use power instead of energy
- `librosa.display.specshow()` with `y_axis='chroma'` now labels as `pitch class`
- set `librosa.core.cqt` parameter `resolution=2` by default
- set `librosa.feature.chromagram` parameter `octwidth=2` by default

v0.2.0

2013-12-14

Bug fixes

- fixed default `librosa.core.stft, istft, ifgram` to match specification
- fixed a float->int bug in peak_pick
- better memory efficiency
- `librosa.segment.recurrence_matrix` corrects for width suppression
- fixed a divide-by-0 error in the beat tracker
- fixed a bug in tempo estimation with short windows
- `librosa.feature.sync` now supports 1d arrays
- fixed a bug in beat trimming
- fixed a bug in `librosa.core.stft` when calculating window size
- fixed `librosa.core.resample` to support stereo signals

Features

- added filters option to cqt
- added window function support to istft
- added an IPython notebook demo
- added `librosa.features.delta` for computing temporal difference features
- new `examples` scripts: tuning, hpss
- added optional trimming to `librosa.segment.stack_memory`
- `librosa.onset.onset_strength` now takes generic spectrogram function `feature`
- compute reference power directly in `librosa.core.logamplitude`
- color-blind-friendly default color maps in `librosa.display.cmap`
- `librosa.core.onset_strength` now accepts an aggregator
- added `librosa.feature.perceptual_weighting`

- added tuning estimation to `librosa.feature.chromagram`
- added `librosa.core.A_weighting`
- vectorized frequency converters
- added `librosa.core.cqt_frequencies` to get CQT frequencies
- `librosa.core.cqt` basic constant-Q transform implementation
- `librosa.filters.cq_to_chroma` to convert log-frequency to chroma
- added `librosa.core.fft_frequencies`
- `librosa.decompose.hpss` can now return masking matrices
- added reversal for `librosa.segment.structure_feature`
- added `librosa.core.time_to_frames`
- added cent notation to `librosa.core.midi_to_note`
- added time-series or spectrogram input options to `chromagram`, `logfsgram`, `melspectrogram`, and `mfcc`
- new module: `librosa.display`
- `librosa.output.segment_csv` => `librosa.output.frames_csv`
- migrated frequency converters to `librosa.core`
- new module: `librosa.filters`
- `librosa.decompose.hpss` now supports complex-valued STFT matrices
- `librosa.decompose.decompose()` supports `sklearn` decomposition objects
- added `librosa.core.phase_vocoder`
- new module: `librosa.onset`; migrated onset strength from `librosa.beat`
- added `librosa.core.pick_peaks`
- `librosa.core.load()` supports offset and duration parameters
- `librosa.core.magphase()` to separate magnitude and phase from a complex matrix
- new module: `librosa.segment`

Other changes

- `onset_estimate_bpm => estimate_tempo`
- removed `n_fft` from `librosa.core.istft()`
- `librosa.core.mel_frequencies` returns `n_mels` values by default
- changed default `librosa.decompose.hpss` window to 31
- disabled onset de-trending by default in `librosa.onset.onset_strength`
- added complex-value warning to `librosa.display.specshow`
- broke compatibilty with `ifgram.m`; `librosa.core.ifgram` now matches `stft`
- changed default beat tracker settings
- migrated `hpss` into `librosa.decompose`
- changed default `librosa.decompose.hpss` power parameter to `2.0`
- `librosa.core.load()` now returns single-precision by default
- standardized `n_fft=2048`, `hop_length=512` for most functions
- refactored tempo estimator

v0.1.0

Initial public release.

Glossary

time series

Typically an audio signal, denoted by `y`, and represented as a one-dimensional *numpy.ndarray* of floating-point values. `y[t]` corresponds to amplitude of the waveform at sample `t`.

sampling rate
: The (positive integer) number of samples per second of a time series. This is denoted by an integer variable `sr`.

frame
: A short slice of a time series used for analysis purposes. This usually corresponds to a single column of a spectrogram matrix.

window
: A vector or function used to weight samples within a frame when computing a spectrogram.

frame length
: The (positive integer) number of samples in an analysis window (or frame). This is denoted by an integer variable `n_fft`.

hop length
: The number of samples between successive frames, e.g., the columns of a spectrogram. This is denoted as a positive integer `hop_length`.

window length
: The length (width) of the window function (e.g., Hann window). Note that this can be smaller than the frame length used in a short-time Fourier transform. Typically denoted as a positive integer variable `win_length`.

spectrogram
: A matrix `S` where the rows index frequency bins, and the columns index frames (time). Spectrograms can be either real-valued or complex-valued. By convention, real-valued spectrograms are denoted as *numpy.ndarray*s `S`, while complex-valued STFT matrices are denoted as `D`.

onset (strength) envelope
: An onset envelope `onset_env[t]` measures the strength of note onsets at frame `t`. Typically stored as a one-dimensional *numpy.ndarray* of floating-point values `onset_envelope`.

chroma
: Also known as pitch class profile (PCP). Chroma representations measure the amount of relative energy in each pitch class (e.g., the 12 notes in the chromatic scale) at a given frame/time.