acc

Apr 13, 2020

Contents:

CHAPTER 1

Introduction

The acc package aims to extract P-wave reflections and image the lithospheric discontinuty structrues.

CHAPTER 2

Work flow

There are five steps to run the acc package including

0. config the parameter file
1. import raw data of teleseismic events
2. calculate auto-correlograms
3. migrate the P-wave reflectivities to depth domain
4. plot results

2.1 0. config parameter

Before you start to config parameter, it is suggested to know the principles of the approach. Please refer to the following papers.

```
1. Weijia Sun and B. L. N. Kennett, 2016, Receiver structure from teleseisms:␣
↪Autocorrelation and cross correlation, Geophys Res Lett, 43, 6234-6242.
2. Weijia Sun and B. L. N. Kennett, 2017, Mid-lithosphere discontinuities beneath the␣
↪western and central North China Craton, Geophys Res Lett, 44, 1302-1310.
```

Then please read through the configure file carefully. Make sure each parameter has been well set. For example, the raw data should be in **SAC format with the headers of event and station information**. Then you can set *data* to the path where the data is saved. Wildcard is supported.

2.2 1. other steps

If the parameters are given properly, the other four steps can be simply executed via the following commands.

```
>>> acc importdata -p conf.json
>>> acc calevent -p conf.json
>>> acc migration -p conf.json
>>> acc plotprofile -p conf.json
```

The figures will be saved under path_you_set/figures in both pdf and png format.

CHAPTER 3

Template Configuration File

```
### Configure file for the ACC package in json format
# Comments are indicated with "#" and ignored while parsing
# It is strongly suggested to read the self-commented configuration file.
{

# type of data used to image structures
# You can select "event" to use teleseismic events or global events using the␣
↪corresponding phases.
# Only Z-component data would be processed for imaging.
# Correspondingly, the phase would be "P" no matter what you set phase to.
"acc_type": "event", # "event" supported only
"phase": "P", # "P" supported only

# number of CPU cores to processing data, if null then use all available CPUs.
"njobs": 12,

### switches of processing
"rm_resp_on": false, # switch of instrumental response removal. I strongly you remove␣
↪RESP externally.
"whiten_on": true, # switch of spectral whitening
"time_norm_on": false, # switch of temporal normalization

"tt_model": "ak135", # model for travel time calculation
"pp_model": "ak135", # model for pirece point calculation
"depth_unit": "m", # "km", "m" of focal depth. Now automatically selected in the code.

### input/output options ###
"io": {

        # Data can be
        #       a wildcard expression of files. All files are read at once. If you have
        #       a huge dataset, think about splitting data into several small pieces.
        "data": "RAW_DATA/*/*.sac",
        # whether force to overwrite if files already existed. true: overwrite,␣
↪false: skip
```

(continues on next page)

(continued from previous page)

```
        "force": true,

        # roopath for results and figures
        "outpath": "acc"
},


### options of data selection ###
# selection criteria for "event" data
"data_selection_event": {
        ### options for event_type of "event"
        "dist_range": [30, 90], # distace range in degree
        "magnitude": [5.0, 7.0], # select all data, use a wide range, e.g., [-10, 100]
        "snr_threshold": 2.0, # signal-to-noise ration, event greater than 'snr_
↪threshold' will be chosen.
        "signal": [-10, 10], # time window of signal in sec relative to the␣
↪theoretical arrival time
        "noise": [-40, -20], # time window of noise in sec relative to the␣
↪theoretical arrival time
        "waterlevel": 1.0e-8 # used for SNR calculation to avoid zero division
},


### options of pre-processing
"preprocessing": {
        # including basic single station processings: downsampling, instrumental␣
↪response removal, detrend, demean
        "sampling_rate": 20, # sampling rate in Hz, 10 Hz mean 0.1 s in sampling␣
↪interval

        ### options for spectral whitening
        "whiten":{
                "smooth": 0.5, # smoothing window width, if null then do not smooth.
                "filter": [0.01, 1.0], # frequency band, if null then do not filter.
                "corners": 4, # filter parameter
                "zerophase": true, # filter parameter
                "waterlevel": 1e-8 # waterlevel for safe division
        },

        ### options for temporal normalization, please see Bensen (2007, GJI).
        "time_norm":{
                "method": "run_abs_mean", # supporting "1bit", "run_abs_mean"
                "time_length": 5, # smoothing window width in sec
                "filter": [0.01, 0.5], #frequency band in Hz
                "corners": 2, # filter parameter
                "zerophase": true, # filter parameter
                "waterlevel": 1.0e-8 # waterlevel for safe division
        }
},


### options for auto and cross correlation
"correlate":{
        ### options for event_type of "event"
        "window": [-10, 110], # in sec relative to the theoretical arrival time
        "filter": [0.1, 0.5], # frequency band
        "corners": 4, # filter parameter
```

(continues on next page)

(continued from previous page)

```
        "zerophase": true # filter parameter
},


### options for stacking, not used for this example.
"stack": {
        "method": "linear", # "linear", "PWS", "bootstrap_linear", 'bootstrap_PWS'
        "power": 2, # power of PWS, 0 means linear stacking
        # options for bootstrap stacking
        "percentage": 0.9, # percentage of all data used for bootstrap stacking
        "seed": 123456789, # random seed
        "n_iter": 100 # number of iterations
},


### options for migration and profile settings
"migration": {
        "model": null # if null, then use the tt_model
},

"profile": {
        # the start point and the end point to define a profile, (latitude, longitude)
        "latlon0": [-19.96, 134.35], # if null, then use the station coordinate.␣
↪useful for one station
        "latlon1": [-19.77, 134.39], # the end point.
        "binsize": 5, # bin size in km
        "binwidth": 200, # bin width in km
        "profile_id": "WB", # profile id shown on the top right corner.
        "wild_card": "WB" # the wildcard is used to extract those expected stations␣
↪to stack. "*" to select all stations
},

### plotting
"plot": {
        "figsize": [20, 3], # figure size of width and height in inches
        "width_ratios": [1, 9], # width_ratios of the waveform and image
        "depth_range": [0, 200],
        "dist_range": [150, 350], # if null, use the range of bins.
        "image_scale": 4,
        "wavef_scale": 2
}

}
```

CHAPTER 4

API documentations

4.1 acc.core module

Preprocessing and correlation

`acc.core.`**`_fill_array`**(*data*, *mask=None*, *fill_value=None*)

Mask numpy array and/or fill array value without demasking.

Additionally set fill_value to value. If data is not a MaskedArray and mask is None returns silently data.

Parameters

- **`mask`** – apply mask to array
- **`fill_value`** – fill value

`acc.core.`**`_find_start_end_time`**(*stream*)

find the start and end time of a specific stream.

Parameters **`stream`** –

Return starttime, endtime

`acc.core.`**`_run_abs_mean`**(*tr*, *time_length=5*, *filter=(0.01*, *1)*, *corners=4*, *zerophase=True*, *waterlevel=1e-08*)

running absolute mean normalization

Parameters

- **`tr`** – trace to be normalized
- **`time_length`** (*`float`*) – time length to be smoothed, default 5 sec
- **`filter`** (*`tuple`*) – bandpass frequency band, default (0.01, 1) Hz
- **`corners`** (*`int`*) – corners default 4
- **`zerophase`** (*`bool`*) – default True
- **`waterlevel`** (*`float`*) – waterlevel to safe guard division

Returns data after temporal normalization

acc.core.**remove_response**(*trace*, *resp_path*, *pre_filt=(0.01*, *0.02*, *8*, *9)*, *output='VEL'*, *format='XML'*)

Romove instrumental response

Parameters

- **trace** –
- **resp_path** –
- **pre_filt** –
- **output** –
- **format** –

Returns

Note: currently only support XML DATALESS RESP format.

acc.core.**rotation**(*stream*, *method='NE->RT'*, *acc_type='event'*)

Rotatation.

Parameters

- **stream** – obspy stream to be rotated
- **method** – "NE->RT" or "ZNE->LQT"
- **acc_type** – "event", "noise"

Return stream obspy stream after rotation.

Note: The keywords *component_azimuth* and *component_inclination* must be given in the stats. Currently, only acc_type="event" are well debugged. The input 3-component data should be in the roughly same periods or time window.

acc.core.**spectral_whitening**(*tr*, *smooth=None*, *filter=None*, *waterlevel=1e-08*, *corners=2*, *zerophase=True*)

Apply spectral whitening to data

Data is divided by its smoothed (Default: None) amplitude spectrum.

Parameters

- **tr** – trace to manipulate
- **smooth** – length of smoothing window in Hz (default None -> no smoothing)
- **filter** – filter spectrum with bandpass after whitening (tuple with min and max frequency) (default None -> no filter)
- **waterlevel** – waterlevel relative to mean of spectrum
- **mask_again** – weather to mask array after this operation again and set the corresponding data to 0
- **corners** – parameters parsing to filter,
- **zerophase** – parameters parsing to filter

Returns whitened data

`acc.core.`**`time_norm`**(*tr*, *method*, *time_length=5*, *filter=(0.01, 1)*, *corners=2*, *zerophase=True*, *waterlevel=1e-08*)

Calculate normalized data, see e.g. Bensen et al. (2007, GJI)

Parameters

- **`tr`** – Trace to manipulate
- **`method`** (*`str`*) – 1bit: reduce data to +1 if >0 and -1 if <0

  run_abs_mean: running absolute mean normalization
- **`time_length`** (*`float`*) – time length to be smoothed, default 5 sec
- **`filter`** (*`tuple`*) – bandpass frequency band, default (0.01, 1) Hz
- **`corners`** (*`int`*) – corners default 2
- **`zerophase`** (*`bool`*) – default True
- **`waterlevel`** (*`float`*) – waterlevel to safe guard division

Returns normalized data

4.2 acc.io module

I/O modules

`acc.io.`**`_acc_read`**(*file*, *outpath*, *acc_type*, *phase*, *force=True*, *tt_model='ak135'*, *depth_unit='km'*)

Read seismic data.

Parameters

- **`file`** – file name to be read
- **`outpath`** – where to save the data
- **`acc_type`** – event or noise
- **`phase`** – if acc_type='event', then calculate the traveltime of the phase
- **`force`** – force to overwrite when file exists if force is True
- **`tt_model`** – 1-d theoretical model, e.g., ak135 used to calculate traveltime of a specific phase
- **`depth_unit`** – unit of depth, general one is km, but in some cases it could be m.

Returns

`acc.io.`**`_get_event_data`**(*tr*, *tt_model*, *phase*, *acc_type*, *depth_unit='km'*)

Update a sac trace to a obspy trace and update trace header, and calculate theoretical traveltime of a specific model and phase

Parameters

- **`tr`** –
- **`tt_model`** –
- **`phase`** –
- **`acc_type`** –
- **`depth_unit`** –

Returns

Note: The input trace should be read from sac-formatted files.

depth_unit is not used. if depth>1000 then unit should be meter, since no events deeper than 700 km on the earth.

acc.io.**_get_event_id**(*tr*)
Get event id from a sac-formatted trace.

Parameters **tr** –

Return str event_id event id

acc.io.**_get_event_id_tr**(*tr*)
Get event id from a obspy trace. The obspy trace should have the event_time keyword.

Parameters **tr** – a obspy trace

Return str event_id event id in "%Y%m%d%H%M%S", e.g., '20191019200909'

acc.io.**_get_noise_data**(*tr*, *acc_type*)
Get update sac-trace header to obspy trace header station_latitude etc.

Parameters

- **tr** –
- **acc_type** –

Return tr

acc.io.**_get_noise_id**(*tr*)
Get trace id for noise type data. e.g., 'starttime-endtime'

Parameters **tr** – an obspy trace

Return str event_id noise id

acc.io.**_get_sac_origin**(*tr*)
Get the origin time of an event trace in sac format.

Parameters **tr** – A event trace

Return origin origin time of an event

Note: The trace should be sac formatted.

acc.io.**_get_station_id**(*tr*)
Get station id of a given trace.

Parameters **tr** – trace

Return station_id station id formatted as '{newwork}.{station}.{location}'.

acc.io.**_load_json**(*jsonfile*)
Load parameters from the json formatted file

Parameters **jsonfile** (*str*) – json file containing parameters

Return dict kwargs dictionary containing parameters `acc.io.`**`import_data`**`(`*jsonfile*`)`

Import data from external media

Parameters **`jsonfile`** – parameter filename

Returns

4.3 acc.main module

4.4 acc.migration module

`acc.migration.`**`_mig_1`**`(`*path*, *model='ak135'*`)`

`acc.migration.`**`mig_one_station`**`(`*stream*, *model='ak135'*, *earth_radius=6378137.0*, *depth_range=(0, 300, 1)*`)`

`acc.migration.`**`migration_1station`**`(`*jsonfile*`)`

`acc.migration.`**`migration_one_station_stack`**`(`*stream*, *method='PWS'*, *power=2*, *time_range=[8, 16]*, *coeff=0.5*`)`

Stacking of migrated traces for one station. Currently not used.

Parameters

- **`stream`** –
- **`method`** – "PWS" for phase-weighted stacking and "linear" for linear stacking
- **`power`** – used by "PWS" only. power=0 is linear stacking
- **`time_range`** – if None do simple stacking (PWS or linear). if a list contains two elements then

it will select traces with high resemblance with the initial stacking to get final stacked trace. :param coeff: traces with correlation efficient higher than the value will be selected. :return: trace after stacking

Note: The one more procedure is to improve signal-to-noise ratio. You may check the stats header *corrstack*. If the key equals zero, the stacking is the general ones, otherwise the correlated stacking are implemented. The value denotes the number of stacked traces.

4.5 acc.plotting module

Plottings

`acc.plotting.`**`_plot`**`(`*amp, depth, stack, extent, dist_range, depth_range, savepath, profile_id, wclip=1, iclip=1, figsize=(5.2, 3), width_ratios=[0.6, 2]*`)`

`acc.plotting.`**`_plot_1station`**`(`*stmig*, *latlon0*, *azimuth*, *bins*, *width*, *savepath*, *depth_range*, ***kwargs*`)`

`acc.plotting.`**`_plot_stations`**`(`*stmig*, *latlon0*, *azimuth*, *bins*, *width*, *savepath*, *depth_range*, *profile_id*, ***kwargs*`)` `acc.plotting.`**`plot_acc_one_station`**(*stream*, *fname=None*, *fig_width=7.0*, *trace_height=0.5*, *stack_height=0.5*, *scale=2*, *scale_stack=10*, *fillcolors=('red', 'blue')*, *info=(('back_azimuth', 'baz (°)', 'C0'), ('distance', 'dist (°)', 'C3'))*)

Plot auto- or correlogram for one station. Reproduced from the *rf* package.

Parameters

- **`stream`** – stream to plot
- **`fname`** – filename to save plot to. Can be None. In this case the figure is left open.
- **`fig_width`** – width of figure in inches
- **`trace_height`** – height of one trace in inches
- **`stack_height`** – height of stack axes in inches
- **`scale`** – scale for individual traces
- **`fillcolors`** – fill colors for positive and negative wiggles
- **`info`** – Plot one additional axes showing maximal two entries of the stats object. Each entry in this list is a list consisting of three entries: key, label and color. info can be None. In this case no additional axes is plotted.

`acc.plotting.`**`plot_ppoint`**(*stream, fname='pp.pdf', depths=[30, 50, 80, 100, 150, 200]*)

`acc.plotting.`**`plot_profile`**(*jsonfile*)

`acc.plotting.`**`write_netcdf4`**(*pos*, *dep*, *data*, *file*)

4.6 acc.processing module

`acc.processing.`**`_autocorrelation`**(*tr, window=[-20, 70], filter=[0.5, 4], corners=2, zerophase=True*)

Autocorrelation for event type data.

Parameters

- **`tr`** –
- **`window`** –
- **`filter`** – A tuple or a list containing the lower and upper limit of filter.
- **`corners`** –
- **`zerophase`** –

Returns

Note: filter after autocorrelation.

`acc.processing.`**`_cal_event_snr`**(*tr, signal=[-10, 10], noise=[-100, -50], waterlevel=1e-08*)

Calculation of SNR for event data.

Parameters

- **`tr`** –
- **`signal`** –

- **`noise`** –
- **`waterlevel`** –

Returns

`acc.processing.`**`_crosscorrelation`**(*tr1, tr2, window=[-20, 70], filter=[0.5, 4], corners=2, zerophase=True*)

cross-correlation for event type data.

Parameters

- **`tr1`** –
- **`tr2`** –
- **`window`** –
- **`filter`** –
- **`corners`** –
- **`zerophase`** –

Returns

`acc.processing.`**`_proc`**(*tr, sampling_rate=10*)

Basic processing including downsampling, detrend, and demean.

Parameters

- **`tr`** – raw trace
- **`sampling_rate`** –

Return tr trace after processing

`acc.processing.`**`_proc_event_Z`**(*file, **kwargs*)

Processing a single component Z including downsampling, detrend, demean, remove instrumental response, selection of data by SNR>threshold, spectral whitening, temporal normalization and auto-correlation.

For event type data.

Parameters

- **`file`** –
- **`kwargs`** –

Returns

`acc.processing.`**`_proc_event_full`**(*st, **kwargs*)

processings including

Parameters

- **`st`** –
- **`kwargs`** –

Returns

`acc.processing.`**`_proc_event_rst`**(*st, **kwargs*)

processing including rotation, spectral whitening and temporal normalization. and cross-correlation.

Parameters

- **`st`** –
- **`kwargs`** –

Returns 1 if processing successfully.

Note: rst is short as Rotation Spectral whitening and Temporal normalization.

---

acc.processing.**_proc_noise_Z** (*file*, ***kwargs*)
Internal functions of calculate z-comp autocorrelation for noise data.

Parameters

- **file** –
- **kwargs** –

Returns

acc.processing.**_rot** (*st*, *method='NE->RT'*)
rotation.

Parameters

- **st** –
- **method** – "NE->RT", "ZNE->LQT".

Returns

Note: the code can handle 'Z12', '123', 'ZNE'. It requires trace header of component_azimuth and component_inclination. if method == "NE->RT", then the two components rotation will be applied. But while the components are "123" or "Z12", they are rotated to "ZNE" first.

---

acc.processing.**_select_data_event** (*tr*, *dist_range=[30, 90]*, *magnitude=[5.0, 9.0]*, *snr_threshold=2.0*, *signal=[-10, 10]*, *noise=[-100, -50]*, *waterlevel=1e-08*)

acc.processing.**_simple_proc** (*st*, *sampling_rate=10*, *njobs=1*)
A parallel version of *_proc*, i.e., Basic processing including downsampling, detrend, and demean.

Parameters

- **st** – an obspy stream
- **sampling_rate** – expected sampling rate
- **njobs** – number of jobs or CPU to use

Return st stream after processing

acc.processing.**_sliding_autocorrelation** (*tr*, *length=3600*, *overlap=1800*, *filter=[0.5, 4]*, *corners=2*, *zerophase=True*)
Sliding autocorrelation for noise data.

Parameters

- **tr** –
- **length** –
- **overlap** –
- **filter** –
- **corners** –
- **zerophase** –

Returns

`acc.processing.`**`processing_event_Z`**(*jsonfile*)
processing vertical component of event data

Parameters **`jsonfile`** –

Returns

`acc.processing.`**`processing_event_full`**(*jsonfile*)

`acc.processing.`**`processing_noise_Z`**(*jsonfile*)
Parallel processing of noise autocorrelation.

Parameters **`jsonfile`** –

Returns

4.7 acc.profile module

`acc.profile.`**`_find_box`**(*latlon*, *boxes*, *crs=None*)
Return the box which encloses the coordinates.

`acc.profile.`**`_find_box_cartesian`**(*latlon*, *boxes*)
Return the box which encloses the coordinates.

`acc.profile.`**`_get_box`**(*latlon0*, *azimuth*, *length*, *width=2000*, *offset=0*)
Create a single box.

`acc.profile.`**`_get_box_cartesian`**(*latlon0*, *azimuth*, *length*, *width=2000*, *offset=0*)
Create a single box.

`acc.profile.`**`df_depth`**(*depth*, *dfmig*, *boxes*, *depths*, *pos_dist*, *a*, *b*)

`acc.profile.`**`direct_geodetic`**(*latlon*, *azi*, *dist*)
Solve direct geodetic problem with geographiclib.

Parameters

- **`latlon`** (*`tuple`*) – coordinates of first point
- **`azi`** – azimuth of direction
- **`dist`** – distance in km

Returns coordinates (lat, lon) of second point on a WGS84 globe

`acc.profile.`**`get_profile_boxes`**(*latlon0*, *azimuth*, *bins*, *width=2000*)
Create 2D boxes for usage in *profile()* function.

Parameters

- **`latlon0`** (*`tuple`*) – coordinates of starting point of profile
- **`azimuth`** – azimuth of profile direction
- **`bins`** (*`tuple`*) – Edges of the distance bins in km (e.g. (0, 10, 20, 30))
- **`width`** – width of the boxes in km (default: large)

Returns List of box dicts. Each box has the entries 'poly' (shapely polygon with lonlat corners), 'length' (length in km), 'pos' (midpoint of box in km from starting coordinates), 'latlon' (midpoint of box as coordinates)

`acc.profile.`**`internal_profile`**(*tr*)

acc.profile.**profile**(*stream*, *boxes*, *crs=None*)
Stack traces in stream by piercing point coordinates in defined boxes.

Parameters

- **stream** – stream with pre-calculated piercing point coordinates
- **boxes** – boxes created with *get_profile_boxes()*
- **crs** – cartopy projection (default: AzimuthalEquidistant)

Returns profile stream

4.8 acc.stack module

stacking

acc.stack.**_bootstrap**(*st*, *normalize=True*, ***kwargs*)

acc.stack.**_gen_random_number**(*low=0*, *high=100*, *n_iter=100*, *percentage=0.9*, *seed=59*)

acc.stack.**_stack**(*path*, ***kwargs*)

acc.stack.**linear_stack**(*stream*, *normalize=True*)

acc.stack.**pws_stack**(*stream*, *power=2*, *normalize=True*)

acc.stack.**stack**(*jsonfile*)

4.9 acc.util module

Utility functions and classes for Auto and Cross Correlogram calculation.

class acc.util.**IterMultipleComponents**(*stream*, *key=None*, *number_components=None*)
Bases: `object`

Return iterable to iterate over associated components of a stream.

Parameters

- **stream** – Stream with different, possibly many traces. It is split into substreams with the same seed id (only last character i.e. component may vary)
- **key** (`str or None`) – Additionally, the stream is grouped by the values of the given stats entry to differentiate between e.g. different events (for example key='starttime', key='onset')
- **number_components** (`int, tuple of ints or None`) – Only iterate through substreams with matching number of components.

__class__
alias of `builtins.type`

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'acc.util', '__doc__': "\n Return iterable to

__dir__()
Default dir() implementation.

`__eq__`
Return self==value.

`__format__``()`
Default object formatter.

`__ge__`
Return self>=value.

`__getattribute__`
Return getattr(self, name).

`__gt__`
Return self>value.

`__hash__`
Return hash(self).

`__init__``(`*stream*, *key=None*, *number_components=None*`)`
Initialize self. See help(type(self)) for accurate signature.

`__init_subclass__``()`
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

`__iter__``()`

`__le__`
Return self<=value.

`__len__``()`

`__lt__`
Return self<value.

`__module__ = 'acc.util'`

`__ne__`
Return self!=value.

`__new__``()`
Create and return a new object. See help(type) for accurate signature.

`__reduce__``()`
Helper for pickle.

`__reduce_ex__``()`
Helper for pickle.

`__repr__`
Return repr(self).

`__setattr__`
Implement setattr(self, name, value).

`__sizeof__``()`
Size of object in memory, in bytes.

`__str__`
Return str(self).

`__subclasshook__``()`
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

`__weakref__`
list of weak references to the object (if defined)

`acc.util.`**`iter_time`**(*tr*, *length=3600*, *overlap=1800*)

`acc.util.`**`pkl2sac1`**(*directory*, *suffix='pkl'*, *fmt='SAC'*)
Convert file from Pickle format with suffix of pkl to SAC format

Parameters

- **`directory`** – the directory contains files to be converted.
- **`suffix`** – in this case, it should be "pkl".
- **`fmt`** – the target format to be converted. Support SAC, MSEED.

Example: /the/path/hello.pkl to /the/path_SAC/hello.sac

`acc.util.`**`smooth`**(*x*, *window_len=None*, *window='flat'*, *method='zeros'*)
Smooth the data using a window with requested size.

This method is based on the convolution of a scaled window with the signal.

Parameters

- **`x`** – the input signal (numpy array)
- **`window_len`** – the dimension of the smoothing window; should be an odd integer
- **`window`** – the type of window from 'flat', 'hanning', 'hamming', 'bartlett', 'blackman' flat window will produce a moving average smoothing.
- **`method`** – handling of border effects

  'zeros': zero padding on both ends (len(smooth(x)) = len(x))

  'reflect': pad reflected signal on both ends (same)

  'clip': pad signal on both ends with the last valid value (same)

  None: no handling of border effects (len(smooth(x)) = len(x) - len(window_len) + 1)

4.10 Module contents

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index

Symbols