ATom: acoustic tomography of the atomsphere Documentation

Release 1.0

Nicholas Hamilton, James Hansen

Sep 07, 2018

CONTENTS:

CHAPTER

ONE

ATOM: ACOUSTIC TOMOGRAPHY OF THE ATOMSPHERE

ATom is a python package designed for the analysis of acoustic tomography data, including processing raw data from acoustic sources and receivers.

Author: **Nicholas Hamilton** James Hansen

1.1 Acknowledgments

Thanks to anyone who would like to contribute! Please see below.

1.2 License

CHAPTER

TWO

CODE REFERENCE

2.1 atom_functions module

This module contains the fundamental object classes and functions of the package.

ATom.atom_functions.**butter_bandpass**(*lowcut*, *highcut*, *fs*, *order=5*)
create band-pass frequency filter to isolate chirp frequency in mic singals

Parameters:

lowcut: float lower limit of frequency band

highcut: float upper limit of frequency band

fs: float sampling frequency of data

order: int filter order, default = 5

Returns:

b, a: ndarray, ndarray Numerator (b) and denominator (a) polynomials of the IIR filter. Only returned if output='ba'.

ATom.atom_functions.**butter_bandpass_filter**(*data*, *lowcut*, *highcut*, *fs*, *order=5*)
implement fileter on input data

Parameters:

data: np.arary acoustic signal to be filtered (microphone data)

lowcut: float lower limit of frequency band, passed to *butter_bandpass*

highcut: float upper limit of frequency band, passed to *butter_bandpass*

fs: float sampling frequency of data, passed to *butter_bandpass*

order: int filter order, default = 5, passed to *butter_bandpass*

Returns:

y: np.array frequency-filtered data

ATom.atom_functions.**covariance**(*micdat*, *speakerdat*)
Lag-N cross correlation between two signals. Only the correlation between a speaker chirp and its respective signal in each microphone record sample is required.

Parameters:

micdat: pd.DataFrame extracted microphone data containing received acoustic signals

speakerdat: pd.DataFrame extracted speaeker acoustic signals

Returns

covar: np.array time-lag correlation between micdat and speakerdat

ATom.atom_functions.**covariance_depricated**(*micdat*, *speakerdat*)
Lag-N cross correlation. Parameters

micdat: pd.DataFrame extracted microphone data containing received acoustic signals

speakerdat: pd.DataFrame extracted speaeker acoustic signals

Returns covar: float

ATom.atom_functions.**crosscorr_depricated**(*datax*, *datay*, *lag=0*)
Lag-N cross correlation.

Parameters

lag: int default 0

datax, datay: pandas.Series objects of equal length

Returns covar: float

class ATom.atom_functions.**dataset**(*datapath*)
dataset is the object class for raw data. It should contain a directory in where raw data are to be found, data I/O routines, experiement constants, array calibration info, etc.

extract_travel_times(*upsamplefactor=10*, *searchLag=None*, *filterflag=True*, *verbose=False*)
Main processing step of raw data.

Acoustic chirps are identified in speaker and microphone signals. Travel time from each speaker to each mic are calculated.

Parameters:

upsamplefactor: int degree to which acoustic signals are upsampled. This is needed to increase precision of travel time estimate

searchLag: int acoustic signal window width. If none is provided, a default window width is assigned of *searchLag = 3 * self.meta.chirp_record_length * upsamplefactor*

filterflag: bool implement frequency filter to microphone signals to remove spurious spectral contributions. Band-pass filter with acoustic chirp bandwidth around the central frequency of the acoustic chip, with the bandwidth

verbose: bool determine output text. used to debug.

Returns:

ATom_signals: np.ndarray [nspeakers, nmics, searchLag, nrecords] acoustic chirps received by the microphones

travel_times: np.ndarray [nspeakers, nmics, nrecords] travel times (ms) of chirps between each speaker and mic for each record

travel_inds: np.ndarray [nspeakers, nmics, nrecords] travel times (samples) of chirps between each speaker and mic for each record

get_calibration_info(*caldatapath*)
get locations of speakers and mics, signal latency between particular devices, and sound propagation delays from speakers as a function of azimuth

Parameters:

caldatapath: str

path to directory containing raw data

- 'average_latency_yymmdd.csv'
- 'mic_locations_yymmdd.csv'
- 'speaker_locations_yymmdd.csv'

and/or containing processed data to import

- 'offsets.py'

get_constants(*constantspath*)
get values of constants used in experiment

Parameters:

constantspath: str path to directory containing 'constants.py'

get_meta(*constantspath*)
get meta data for experiment

Parameters:

constantspath: directory path to file containing meta data

load_aux(*sampletime*)
load data file into dataset object

Parameters:

aux_dataapath: str path to directory containing aux data

load_data_sample(*fileID*)
load data file into dataset object

Parameters:

fileID: int integer value of main and aux data in respective lists

load_main(*sampletime*)
load data file into dataset object

Parameters:

maindatapath: str path to directory containing main data

signal_ETAs()
imports the current speaker (i) and microphone (j) as well as the temperature array for the current signal period

time_info()
print time resolution of main and aux data

ATom.atom_functions.**freq_filter**(*datasample*, *filter_freq_inds*)
frequency filter to isolate chirp signal in microphones brute force method takes FFT of datasample, sets frequecies outside specified windows to zero, implements IFFT.

Probably produces ringing in data. Should probably use proper filter.

Parameters:

datasample: np.array data to filter

filter_freq_inds: np.array key frequencies used in filter design.

ATom.atom_functions.**get_speaker_signal_delay**(*speakersamp*)
extract the speaker signal delays from a single record

Parameters

speakersamp: pd.DataFrame speaker time series data for a single record

Returns:

speaker_signal_delay: np.array index corresponding to detected speaker signal delays

class ATom.atom_functions.**meta_data**
Base class for instrument or data record meta data. Takes in a list of parameters and values.

from_data(*data*)
pass data from constants file, create attribute/value pairs use to create meta data attributes

from_file(*keys*, *values*)
instantiate meta_data object from lists of file pairs

from_lists(*keys*, *values*)
instantiate meta_data object from lists of attribute/value pairs

to_list()
pretty print list of attributes

ATom.atom_functions.**rollchannel**(*data*, *rollval*)
shifts values of data forward by rollval index

Parameters:

data: pandas.Series data to shift in time

rollval: int default 0

ATom.atom_functions.**signalOnMic**(*micsamp*, *speakersigs*, *signalETAs*, *searchLag*, *chirp_record_length*)
Extract chirps from the microphones. Each microphone receives (nominally 8) chirps emitted by each speaker. Known speaker/mic locations, along with known speaker chirp emission times, together provide expected travel times from each speaker to each mic. Time correlation between signals determines the precise time of chirp arrival and adds to the expected signal travel times.

Parameters:

micsamp: pd.DataFrame microphone signals for a single record

speakersigs: pd.DataFrame extracted acoustic chirps from 'signalOnSpeaker'

searchLag: int length of search window in samples

chirp_record_length: int length of acoustic chirp in samples base value = 116 multiplied by upsample factor

Returns:

micsigs: pd.DataFrame extracted speaker chirps, centered in a window of length searchLag should be nSpeaker signals for each microphone

time_received_record: np.array transit time of each acoustic signal in samples

ATom.atom_functions.**signalOnSpeaker**(*speakersamp*, *searchLag*, *chirp_record_length*, *speaker_signal_delay*)
Extract chirps from the speakers. These are generated signals, and so are clean, consistent, and spaced by known amounts. Speaker signals are compared against microphone signals to determine the actual trasit time of acoustic chirps across the array.

Parameters:

speakersamp: pd.DataFrame speaker signals for a single record

searchLag: int length of search window in samples

chirp_record_length: int length of acoustic chirp in samples base value = 116 multiplied by upsample factor

speaker_signal_delay: array each speaker chirp is delayed by a specified amount to offset the chirps in time base values = [2480, 2080, 4080, 0, 3200, 4000, 800, 2880] multiplied by upsample factor

Returns:

speakersigs: pd.DataFrame extracted speaker chirps, centered in a window of length searchLag

ATom.atom_functions.**upsample**(*datasample*, *upsamplefactor*, *method='cubic'*)
upsample data by desired factor using specified method

Parameters:

datasample: pd.DataFrame microphone or speaker data to upsample

upsamplefactor: int, float factor by which to upsample data upsamplefactor > 1 ==> increase in time resolution upsamplefactor < 1 ==> decrease in time resolution

method: str method by which to interpolate data:

- pandas - built-in interp method, slow
- linear - linear interpolation
- cubic (default) - cubic interpolation

Returns:

newdatasample: pd.DataFrame upsampled data

ATom.atom_functions.**upsample_data_deptricated**(*self*, *upsamplefactor*)
artificially upsample data to provide the desired resolution new_timedelta = oldtimedelta / upsamplefactor upsamplefactor > 1 ==> increase in time resolution upsamplefactor < 1 ==> decrease in time resolution

Parameters:

upsamplefactor: float or int scale by which to resample data

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

INDEX