windtunnel Documentation

Release 0.0.1

Benyamin Schliffke & Jessica Wiedemeier

Aug 08, 2018

CONTENTS

A collection of tools for basic boundary layer and concentration measurements analysis with Python 3.

CHAPTER

ONE

WINDTUNNEL

Python package for basic boundary layer and concentration measurement analysis.

class windtunnel.**PointConcentration**(*time*, *wtref*, *slow_FID*, *fast_FID*, *open_rate*)

PointConcentration is a class that holds data collected during a continuous release point concentration measurement. The class can hold the raw time series, the corresponding wtref and all other quantities necessary to analyse the time series. All the information in a PointConcentration object can be saved to a txt file. @parameter: time, type = np.array @parameter: wtref, type = np.array @parameter: fast_FID, type = np.array @parameter: slow_FID, type = np.array @parameter: open_rate, type = np.array

ambient_conditions(*x*, *y*, *z*, *pressure*, *temperature*, *calibration_curve*, *mass_flow_controller*, *calibration_factor=0*)

Collect ambient conditions during measurement. pressure in [Pa], temperature in [°C].

calc_c_star()

Calculate dimensionless concentration. [-]

calc_full_scale_concentration()

Calculate full scale concentration in [ppmV].

calc_full_scale_flow_rate()

Convert flow rate to full scale flow rate in [m^3/s].

calc_full_scale_time()

Calculate full scale timesteps in [s].

calc_model_mass_flow_rate()

Calculate the model scale flow rate in [kg/s].

calc_net_concentration()

Calculate net concentration in [ppmV].

calc_wtref_mean()

Calculate scaled wtref mean in [m/s].

clear_zeros()

Clear and count zeros in concentration measurements.

convert_temperature()

Convert ambient temperature to °K.

classmethod from_file(*filename*)

Create PointConcentration object from file. open_rate is converted to %.

full_scale_information(*full_scale_wtref*, *full_scale_flow_rate*)

Collect information on desired full scale information. full_scale_wtref in [m/s]. full_scale_flow_rate is automatically adjusted to standard atmosphere conditions. input in [kg/s], output in [m^3/s].

save2file_avg(*filename*, *out_dir=None*)
Save average full scale and model scale data from PointConcentration object to txt file. filename must include '.txt' ending. If no out_dir directory is provided './' is set as standard. @parameter: filename, type = str @parameter: out_dir, type = str

save2file_fs(*filename*, *out_dir=None*)
Save full scale and model scale data from PointConcentration object to txt file. filename must include '.txt' ending. If no out_dir directory is provided './' is set as standard. @parameter: filename, type = str @parameter: out_dir, type = str

save2file_ms(*filename*, *out_dir=None*)
Save model scale data from PointConcentration object to txt file. filename must include '.txt' ending. If no out_dir directory is provided './' is set as standard. @parameter: filename, type = str @parameter: out_dir, type = str

scaling_information(*scaling_factor*, *scale*, *ref_length*, *ref_height*)
Collect data necessary to scale the results. unit: [m], where applicable.

to_full_scale()
Return all quantities to full scale. Requires XXXXXX to be specified.

tracer_information(*gas_name*, *mol_weight*, *gas_factor*)
Collect information on tracer gas used during measurement. Molecular weight in [kg/mol].

class windtunnel.**PuffConcentration**(*time*, *wtref*, *slow_FID*, *fast_FID*, *signal*, *open_rate*)
PuffConcentration is a class that holds data collected during a puff release point concentration measurement. The class can hold the raw time series, the corresponding wtref and all other quantities necessary to analyse the time series. The PuffConcentration class inherits from pandas.DataFrame, thus offers all of the functionality offered by pandas (e.g. DataFrame.plot.hist(), DataFrame.to_excel(), or DataFrame.rolling().mean()) All the information in a PuffConcentration object can be saved to a txt file, as well as all file type offered by pandas. @parameter: time, type = pd.Series @parameter: wtref, type = np.array @parameter: fast_FID, type = pd.Series @parameter: slow_FID, type = pd.Series @parameter: signal, type = np.array @parameter: open_rate, type = np.array

apply_threshold_concentration(*threshold_concentration=0.0*)
Apply a given threshold concentration to peak_concentration to remove weak puffs. The default value for threshold_concentration is 0. (float).

avg_arrival_time
Get average arrival time.

avg_ascent_time
Get average ascent time.

avg_descent_time
Get average descent time.

avg_leaving_time
Get average leaving time.

avg_peak_concentration
Get average peak concentration.

avg_peak_time
Get average peak time.

calc_net_concentration()
Calculate net concentration in [ppmV].

calc_release_length()
Calculate the length of each release period. Returns an np.array containing the duration of each release

period.

check_against_avg_puff()
Check each puff against the average puff of the time series.

detect_arrival_time()
Detects the beginning of each puff. Returns an np.array containing the first timestamp of each puff.

detect_begin_release_index()
Detects the indices of the end of each release period. Returns a list containing the index of the last timestamp of each release period.

detect_begin_release_period()
Detects the beginning of each release period. Returns an np.array containing the first timestamp of each release period.

detect_end_release_index()
Detects the indices of the end of each release period. Returns a list containing the index of the last timestamp of each release period.

detect_end_release_period()
Detects the end of each release period. Returns an np.array containing the last timestamp of each release period.

detect_leaving_time()
Detects the end of each puff. Returns an np.array containing the last timestamp of each puff.

classmethod from_file(*filename*)
Create PuffConcentration object from file. open_rate is converted to %. :type filename: str

get_ascent_time()
Calculate the ascent time between arrrival time and peak time. Returns an np.array.

get_descent_time()
Calculate the ascent time between arrrival time and peak time. Returns an np.array.

get_dosage()
Calculates the dosage of each puff between two release times.

get_peak_concentration()
Acquire peak concentration of each puff. Returns a list.

get_peak_time()
Acquire peak time of each puff. Returns a list.

get_puff_statistics()
Returns DataFrame with all puff information.

get_residence_time()
Calculate the residence time of each puff. Returns an np.array.

max_puffs
Get maximum number of puffs. Deduced from the length of release_length.

offset_correction()
Correct a non-zero offset in the concentration measured.

save2file(*filename*, *out_dir=None*)
Save data from PointConcentration object to txt file. filename must include '.txt' ending. If no out_dir directory is provided './' is set as standard. @parameter: filename, type = str @parameter: out_dir, type = str

class windtunnel.**Timeseries**(*u*, *v*, *x=None*, *y=None*, *z=None*, *t_arr=None*, *t_transit=None*, *tau=10000*)

Timeseries is a class that holds data collected by the BSA software in the standard BSA software output. The class can hold die raw timeseries, the corresponding wtref, the components and coordinates of each measurement as well as the mean wind magnitude and the mean wind direction. The raw timeseries can be processed by nondimensionalising it, adapting the scale, making it equidistant and masking outliers. All the information in a Timeseries object can be saved to a txt file. @parameter: u, type = np.array @parameter: v, type = np.array @parameter: x, type = float @parameter: y, type = float @parameter: z, type = float @parameter: t_arr, type = np.array @parameter: t_transit, type = np.array @parameter: tau, type = int or float - time scale in milliseconds

adapt_scale(*scale*)
Convert timeseries from model scale to full scale. @parameter: scale, type = float

calc_direction()
Calculate wind direction from components.

calc_equidistant_timesteps()
Create equidistant time series.

calc_magnitude()
Calculate wind magnitude from components.

calc_perturbations()
Calculates u' and v' relative to the mean of each tau-long data segment

classmethod from_file(*filename*)
Create Timeseries object from file.

get_wind_comps(*filename*)
Get wind components from filename. @parameter: filename, type = str

get_wtref(*wtref_path*, *filename*, *index=0*, *vscale=1.0*)
Reads wtref-file selected by the time series name 'filename' and scales wtref with vscale. vscale is set to 1 as standard. index accesses only the one wtref value that is associated to the current file. @parameter: path, type = string @parameter: filename, type = string @parameter: index, type = int @parameter: vscale, type = float

mask_outliers(*std_mask=5.0*)
Mask outliers and print number of outliers. std_mask specifies the threshold for a value to be considered an outlier. 5 is the default value for std_mask. @parameter: std_mask, type = float

mean_direction
Calculate mean wind direction from components relative to the wind tunnels axis.

mean_magnitude
Calculate mean wind magnitude from unweighted components.

nondimensionalise()
Nondimensionalise the data. wtref is set to 1 if no wtref is speciefied.

save2file(*filename*, *out_dir=None*)
Save data from Timeseries object to txt file. filename must include '.txt' ending. If no out_dir directory is provided 'C:/Users/[your_u_number]/Desktop/LDA-Analysis/' is set as standard. @parameter: filename, type = str @parameter: out_dir, type = str

set_tau(*milliseconds*)
Give tau a new value

weighted_component_mean
Weigh the u and v component with its transit time through the measurement volume. This is analoguous to the processing of the raw data in the BSA software. Transit time weighting removes a possible bias towards higher wind velocities. Returns the weighted u and v component means.

weighted_component_variance
Weigh the u and v component with its transit time through the measurement volume. This is analoguous to the processing of the raw data in the BSA software. Transit time weighting removes a possible bias towards higher wind velocities. Returns the weighted u and v component variance.

wind_direction_mag_less_180()
Return the wind direction in the range -180 to +180 degrees.

class windtunnel.**Timeseries_nc**(*comp_1*, *comp_2*, *x=None*, *y=None*, *z=None*, *t_arr_1=None*, *t_transit_1=None*, *t_arr_2=None*, *t_transit_2=None*)
Timeseries is a class that holds data collected by the BSA software in non-coincidence mode using the standard BSA software output. The class can hold die raw timeseries, the corresponding wtref, the components and coordinates of each measurement as well as the mean wind magnitude and the mean wind direction. The raw timeseries can be processed by nondimensionalising it, adapting the scale, making it equidistant and masking outliers. All the information in a Timeseries object can be saved to a txt file. @parameter: u, type = np.array @parameter: v, type = np.array @parameter: x, type = float @parameter: y, type = float @parameter: z, type = float @parameter: t_arr, type = np.array @parameter: t_transit, type = np.array

adapt_scale(*scale*)
Convert timeseries from model scale to full scale. @parameter: scale, type = float

calc_direction()
Calculate wind direction from components.

calc_magnitude()
Calculate wind magnitude from components.

equidistant()
Create equidistant time series.

classmethod from_file(*filename*)
Create Timeseries object from file.

get_wind_comps(*filename*)
Get wind components from filename. @parameter: filename, type = str

get_wtref(*wtref_path*, *filename*, *index=0*, *vscale=1.0*)
Reads wtref-file selected by the time series name 'filename' and scales wtref with vscale. vscale is set to 1 as standard. index accesses only the one wtref value that is associated to the current file. @parameter: path, type = string @parameter: filename, type = string @parameter: index, type = int @parameter: vscale, type = float

mask_outliers(*std_mask=5.0*)
Mask outliers and print number of outliers. std_mask specifies the threshold for a value to be considered an outlier. 5 is the default value for std_mask. @parameter: std_mask, type = float

mean_direction
Calculate mean wind direction from components relative to the wind tunnels axis.

mean_magnitude
Calculate mean wind magnitude from unweighted components.

nondimensionalise()
Nondimensionalise the data. wtref is set to 1 if no wtref is speciefied.

pair_components(*atol=1*)
Pair components in comp_1 and comp_2 using atol as absolute tolerance to match a pair of measurements. atol is set to 1 as default, its unit is [ms]. @parameter: atol, type = float or int

save2file(*filename*, *out_dir=None*)
Save data from Timeseries object to txt file. filename must include '.txt' ending. If no out_dir directory is provided './' is set as standard. @parameter: filename, type = str @parameter: out_dir, type = str

weighted_component_mean
Weigh the u and v component with its transit time through the measurement volume. This is analoguous to the processing of the raw data in the BSA software. Transit time weighting removes a possible bias towards higher wind velocities. Returns the weighted u and v component means.

weighted_component_variance
Weigh the u and v component with its transit time through the measurement volume. This is analoguous to the processing of the raw data in the BSA software. Transit time weighting removes a possible bias towards higher wind velocities. Returns the weighted u and v component variance.

windtunnel.**adapt_scale**(*x*, *y*, *z*, *t_arr*, *scale*)
Convert timeseries from model scale to full scale. @parameter: x, type = int or float @parameter: y, type = int or float @parameter: z, type = int or float @parameter: t_arr, type = np.array @parameter: scale, type = float

windtunnel.**calc_acorr**(*timeseries*, *maxlags*)
Full autocorrelation of time series for lags up to maxlags. @parameter timeseries: np.array or list @parameter maxlags: int

windtunnel.**calc_alpha**(*u_mean*, *heights*, *d0=0.0*, *sfc_height=120.0*, *BL_height=600.0*)
Estimate the power law exponent alpha. @parameter: u_mean, type = list or np.array @parameter: heights, type = list or np.array @parameter: d0, type = float @parameter: sfc_height, type = float @parameter: BL_height, type = float

windtunnel.**calc_autocorr**(*timeseries*, *lag=1*)
Autocorrelation of time series with lag. @parameter tiemseries: np.array or list @parameter lag: int

windtunnel.**calc_exceedance_prob**(*data*, *threshold*)
Calculates exceedance probability of threshold in data. Returns threshold and exceedance probability in percent. @parameter data: @parameter threshold: int

windtunnel.**calc_intervalmean**(*indata*, *intervals*, *DD=False*)
Calculates interval means of indata. If DD is set to True the means are calculated for circular quantities. Returns a dictionary with intervals as keys. If intervals has length 1 the function returns an array. @parameter: indata, type = any @parameter: intervals, type = list @parameter: DD, type = boolean

windtunnel.**calc_lux_data**(*dt*, *u_comp*)
Calculates the integral length scale according to R. Fischer (2011) from an equidistant time series of the u component using time step dt. @parameter: t_eq, type = int or float @parameter: u_comp, type = np.array or list

windtunnel.**calc_lux_data_wght**(*transit_time*, *dt*, *u_comp*)
Calculates the integral length scale according to R. Fischer (2011) from an equidistant time series of the u component using time step dt. @parameter: t_eq, type = int or float @parameter: u_comp, type = np.array or list

windtunnel.**calc_normalization_params**(*freqs*, *transform*, *t*, *height*, *mean_x*, *sdev_x*, *num_data_points*)
Calculate the normalized Fourier transform and frequency for the Fourier transform of x Warning: A previous code version normalized segments, while this version normalizes the entire data set at once. The previous version also included a smoothing algorithm, which has been omitted for simplicity. @parameter: freqs, type = list or np.array @parameter: transform, type = list or np.array - this is the non-normalized Fourier transform @parameter: t, type = float - this is time @parameter: height, type = float - z in the Timeseries object @parameter: mean_x, type = float - the mean of the parameter of the Fourier transform F(x) @parameter: sdev_x, type = float - the standard deviation of x @parameter: num_data_points, type = int - the number of elements in x before the transform was found

windtunnel.**calc_ref_spectra**(*reduced_freq*, *a*, *b*, *c*, *d*, *e*)
Calculate dimensionless reference spectra. ??? @parameter: reduced_freq, type = ??? @parameter: a, type = ??? @parameter: b, type = ??? @parameter: c, type = ??? @parameter: d, type = ??? @parameter: e, type = ???

windtunnel.**calc_spectra**(*u_comp*, *v_comp*, *t_eq*, *height*)
Calculate dimensionless energy density spectra from an equidistant time series. @parameter: u_comp, type = np.array or list @parameter: v_comp, type = np.array or list @parameter: t_eq, type = np.array or list

windtunnel.**calc_stats**(*sets*, *DD=False*)
Returns mean, standard deviation and variance of data in sets. If DD is true then the circular equivalents are calculated. TO BE USED WITH CAUTION @parameter sets: iterable set of data @parameter DD: boolean

windtunnel.**calc_turb_data**(*u_comp*, *v_comp*)
Calculate turbulence intensity and turbulent fluxes from equidistant times series of u and v components. @parameter: u_comp: np.array or list @parameter: v_comp: np.array or list

windtunnel.**calc_turb_data_wght**(*transit_time*, *u_comp*, *v_comp*)
Calculate turbulence intensity and turbulent fluxes from equidistant times series of u and v components using transit time weighted statistics. @parameter: transit_time. type = np.array @parameter: u_compy type = np.array @parameter: v_comp, type = np.array

windtunnel.**calc_wind_stats**(*u_comp*, *v_comp*, *wdir=0.0*)
Calculate wind data from equidistant times series of u and v components. wdir is a reference wind direction. @parameter: u_comp: np.array or list @parameter: v_comp: np.array or list @parameter: wdir: int

windtunnel.**calc_wind_stats_wght**(*transit_time*, *u_comp*, *v_comp*, *wdir=0.0*)
Calculate wind data from equidistant times series of u and v components. wdir is a reference wind direction. @parameter: transit_time, type = np.array @parameter: u_comp, type = np.array @parameter: v_comp, type = np.array @parameter: wdir: int

windtunnel.**calc_z0**(*u_mean*, *heights*, *d0=0.0*, *sfc_height=120.0*, *BL_height=600.0*)
Estimate the roughness length z0. @parameter: u_mean, type = list or np.array @parameter: heights, type = list or np.array @parameter: d0, type = float @parameter: sfc_height, type = float @parameter: BL_height, type = float

windtunnel.**check_directory**(*directory*)
Checks if directory exists. If directory doesn't exist, it is created. @parameter: directory, type = string

windtunnel.**convergence_test_1**(*data*, *blocksize=100*)
Conducts a block-wise convergence test on non circular data using blocksize for the size of each increment. Returns a dictionary block_data. Each entry is named after its respective interval. blocksize's default value is 100. @parameter: data, type = np.array or list @parameter: blocksize, type = int or float

windtunnel.**convergence_test_2**(*data*, *interval=100*, *blocksize=100*)
Conducts a block-wise convergence test on non circular data using blocksize for the size of each increment between intervals. Returns a dictionary block_data. Each entry is named after its respective interval. blocksize's and interval's default values are 100. @parameter: data, type = np.array or list @parameter: interval, type = int @parameter: blocksize, type = int

windtunnel.**count_nan_chunks**(*data*)
Counts chunks of NaNs in data. Returns the size of each chunk and the overall number of chunks. @parameter: data, type = np.array or string

windtunnel.**equ_dist_ts**(*arrival_time*, *eq_dist_array*, *data*)
Create a time series with constant time steps. The nearest point of the original time series is used for the corresponding time of the equi-distant time series. @parameter: arrival_time, type = np.array @parameter: eq_dist_array, type = np.array @parameter: data, type = np.array

windtunnel.**equidistant**(*u*, *v*, *t_arr*)
Create equidistant time series. @parameter: u, type = np.array @parameter: v, type = np.array @parameter: t_arr, type = np.array or list

windtunnel.**find_block**(*indata*, *length*, *tolerance*)
Finds block of size length in indata. Tolerance allows some leeway. Returns array. @parameter: indata, type =

np.array (1D) @parameter: length, type = int @parameter: tolerance, type = int

windtunnel.**find_nearest**(*array*, *value*)
Finds nearest element of array to value. @parameter: array, np.array @parameter: value, int or float

windtunnel.**from_file**(*path*, *filename*)
Create array from timeseries in path + file. @parameter: path, string @parameter: filename, string

windtunnel.**get_files**(*path*, *filename*)
Finds files with filename in path as specified. Filename supports the Unix shell-style wildcards (*,?,[seq],[!seq]) @parameter: path, type = string @parameter: filename, type = string

windtunnel.**get_lux_referencedata**(*ref_path=None*)
Reads and returns reference data for the integral length scale (Lux). This function takes no parameters.

windtunnel.**get_pdf_max**(*data*)
Finds maximum of the probability distribution of data. @parameter data: np.array

windtunnel.**get_percentiles**(*data_dict*, *percentile_list*)
Get percentiles from each entry in data_dict specified in percentile_list. Returns a dictionary with the results. @parameter: data_dict, type = dict @parameter: percentile_list, type = list

windtunnel.**get_reference_spectra**(*height*, *ref_path=None*)
Get referemce spectra from pre-defined location.

windtunnel.**get_turb_referencedata**(*component*, *ref_path=None*)
Reads and returns the VDI reference data for the turbulence intensity of component. @parameter: component, type = string

windtunnel.**get_wind_comps**(*path*, *filename*)
Get wind components from filename. @parameter: filename, type = str

windtunnel.**get_wtref**(*wtref_path*, *filename*, *index=0*, *vscale=1.0*)
Reads wtref-file selected by the time series name 'filename' and scales wtref with vscale. vscale is set to 1 as standard. index accesses only the one wtref value that is associated to the current file. @parameter: path, type = string @parameter: filename, type = string @parameter: index, type = int @parameter: vscale, type = float

windtunnel.**mask_outliers**(*u*, *v*, *std_mask=5.0*)
Mask outliers and print number of outliers. std_mask specifies the threshold for a value to be considered an outlier. 5 is the default value for std_mask. @parameter: u, type = np.array @parameter: v, type = np.array @parameter: std_mask, type = float

windtunnel.**mask_outliers_wght**(*transit_time*, *u*, *v*, *std_mask=5.0*)
Mask outliers and print number of outliers. std_mask specifies the threshold for a value to be considered an outlier. 5 is the default value for std_mask. This function usues time transit time weighted statistics. @parameter: u, type = np.array @parameter: v, type = np.array @parameter: std_mask, type = float

windtunnel.**nondimensionalise**(*u*, *v*, *wtref=None*)
Nondimensionalise the data. wtref is set to 1 if no wtref is specified. @parameter: u, type = np.array @parameter: v, type = np.array @parameter: wtref, type = int or float

windtunnel.**plot_DWD_windrose**(*inFF*, *inDD*)
Plots windrose according to DWD classes of 1 m/s for velocity data and 30 degree classes for directional data. The representation of the windrose in this function is less detailed than in plotwindrose(). @parameter inFF: np.array @parameter inDD: np.array

windtunnel.**plot_JTFA_STFT**(*u1*, *v1*, *t_eq*, *height*, *second_comp='v'*, *window_length=3500*, *fixed_limits=(None, None)*, *ymax=None*)
Plots the joint time frequency analysis using a short-time Fourier transform smoothed and raw for both wind components in one figure. Returns the figure. To change overlap, @parameter: u1: array of u-component perturbations @parameter: v1: array of second-component perturbations @parameter: t_eq: as defined by

Timeseries @parameter: height: z as defined by Timeseries @parameter: second_comp, type = string: the name of the second measured

wind component

@parameter: window_length, type = int: window length in ms

windtunnel.**plot_Re_independence**(*data*, *wtref*, *yerr=0*, *ax=None*, ***kwargs*)

Plots the results for a Reynolds Number Independence test from a non- dimensionalised timeseries. yerr specifies the uncertainty. Its default value is 0. @parameter: data, type = np.array or list @parameter: wtref, type = np.array or list @parameter: yerr, type = int or float @parameter: ax: axis passed to function @parameter: **kwargs: additional keyword arguments passed to plt.plot()

windtunnel.**plot_boxplots**(*data_dict*, *ylabel=None*, ***kwargs*)

Plot statistics of concentration measurements in boxplots. Expects input from PointConcentration class. @parameters: data, type = dict @parameters: ylabel, type = string @parameter ax: axis passed to function @parameter **kwargs : additional keyword arguments passed to plt.boxplot()

windtunnel.**plot_convergence**(*data_dict*, *ncols=3*, ***kwargs*)

Plots results of convergence tests performed on any number of quantities in one plot. ncols specifies the number of columns desired in the output plot. **kwargs contains any parameters to be passed to plot_convergence_test, such as wtref, ref_length and scale. See doc_string of plot_convergence_test for more details. @parameter: data_dict, type = dictionary @parameter: ncols, type = int @parameter: **kwargs keyword arguments passed to plot_convergence_test

windtunnel.**plot_convergence_test**(*data*, *wtref=1*, *ref_length=1*, *scale=1*, *ylabel=''*, *ax=None*, ***kwargs*)

Plots results of convergence tests from data. This is a very limited function and is only intended to give a brief overview of the convergence rest results using dictionaries as input objects. wtref, ref_length and scale are used to determine a dimensionless time unit on the x-axis. Default values for each are 1. @parameter: data_dict, type = dictionary @parameter: wtref, type = float or int @parameter: ref_length, type = float or int @parameter: scale, type = float or int @parameter: ylabel, type = string @parameter: ax: axis passed to function

windtunnel.**plot_fluxes**(*data*, *heights*, *yerr=0*, *component='v'*, *lat=False*, *ax=None*, ***kwargs*)

Plots fluxes from data for their respective height with a 10% range of the low point mean. yerr specifies the uncertainty. Its default value is 0. WARNING: Data must be made dimensionless before plotting! If lat is True then a lateral profile is created. @parameter: data, type = list or np.array @parameter: height, type = list or np.array @parameter: yerr, type = int or float @parameter: component, type = string @parameter: lat, type = boolean @parameter ax: axis passed to function @parameter **kwargs : additional keyword arguments passed to plt.plot()

windtunnel.**plot_fluxes_log**(*data*, *heights*, *yerr=0*, *component='v'*, *ax=None*, ***kwargs*)

Plots fluxes from data for their respective height on a log scale with a 10% range of the low point mean. yerr specifies the uncertainty. Its default value is 0. WARNING: Data must be made dimensionless before plotting! @parameter: data, type = list or np.array @parameter: height, type = list or np.array @parameter: yerr, type = int or float @parameter: component, type = string @parameter ax: axis passed to function @parameter **kwargs : additional keyword arguments passed to plt.plot()

windtunnel.**plot_hist**(*data*, *ax=None*, ***kwargs*)

Creates a scatter plot of x and y. @parameter: data, type = list or np.array @parameter ax: axis passed to function @parameter **kwargs : additional keyword arguments passed to plt.plot()

windtunnel.**plot_lux**(*Lux*, *heights*, *err=0*, *lat=False*, *ref_path=None*, *ax=None*, ***kwargs*)

Plots Lux data on a double logarithmic scale with reference data. yerr specifies the uncertainty. Its default value is 0. If lat is True then a lateral profile, without a loglog scale, is created. @parameter: Lux, type = list or np.array @parameter: heights, type = list or np.array @parameter: err, type = int or float @parameter: lat, type = boolean @parameter: ref_path = string @parameter ax: axis passed to function @parameter **kwargs : additional keyword arguments passed to plt.plot()

windtunnel.**plot_perturbation_rose**(*u1*, *v1*, *total_mag*, *total_direction*, *bar_divider=3000*, *second_comp='v'*)

Plots a detailed wind rose using only the perturbation component of the wind. Number of bars depends on bar_divider and length of u1. @parameter: u1: array of u-component perturbations @parameter: v1: array of second-component perturbations @parameter: total_mag: array containing magnitude of wind (not perturbation) @parameter: total_direction: array containing direction of wind (not perturbation) @parameter: bar_divider: inversely proportional to number of bars to be plotted @parameter: second_comp, type = string: the name of the second measured

wind component

windtunnel.**plot_rose**(*inFF*, *inDD*, *ff_steps*, *dd_range*)

Plots windrose according to user specified input from ff_steps and dd_Range. @parameter: inFF, type = np.array @parameter: inDD, type = np.array @parameter: ff_steps, type = list or np.array @parameter: dd_range, type = int or float

windtunnel.**plot_scatter**(*x*, *y*, *std_mask=5.0*, *ax=None*, ***kwargs*)

Creates a scatter plot of x and y. All outliers outside of 5 STDs of the components mean value are coloured in orange. @parameter: x, type = list or np.array @parameter: y, type = list or np.array @parameter: std_mask, float @parameter ax: axis passed to function @parameter **kwargs : additional keyword arguments passed to plt.scatter()

windtunnel.**plot_spectra**(*f_sm*, *S_uu_sm*, *S_vv_sm*, *S_uv_sm*, *u_aliasing*, *v_aliasing*, *uv_aliasing*, *wind_comps*, *height*, *ref_path=None*, *ax=None*, ***kwargs*)

Plots spectra using INPUT with reference data. @parameter: ??? @parameter: ref_path, type = string @parameter ax: axis passed to function @parameter **kwargs : additional keyword arguments passed to plt.plot()

windtunnel.**plot_stdevs**(*u_unmasked*, *t_eq*, *tau*, *comp='u'*)

This method plots the spread of an array based on how many standard deviations each point is from the mean over each tau-long time period @parameter: the array to be analysed @parameter: the times corresponding to the array to be analysied (ms) @parameter: the characteristic time scale (ms)

windtunnel.**plot_turb_int**(*data*, *heights*, *yerr=0*, *component='I_u'*, *lat=False*, *ref_path=None*, *ax=None*, ***kwargs*)

Plots turbulence intensities from data with VDI reference data for their respective height. yerr specifies the uncertainty. Its default value is 0. If lat is True then a lateral profile is created. @parameter: data, type = list or np.array @parameter: heights, type = list or np.array @parameter: yerr, type = int or float @parameter: component, type = string @parameter: lat, type = boolean @parameter: ref_path, type = string @parameter: ax, axis passed to function @parameter **kwargs : additional keyword arguments passed to plt.plot()

windtunnel.**plot_wind_dir_hist**(*data*, *heights*)

Simple wind direction histogram plot @parameter: Timeseries

windtunnel.**plot_winddata**(*mean_magnitude*, *u_mean*, *v_mean*, *heights*, *yerr=0*, *lat=False*, *ax=None*, ***kwargs*)

Plots wind components and wind magnitude for their respective height. yerr specifies the uncertainty. Its default value is 0. If lat is True then a lateral profile is created. @parameter: mean_magnitude, type = list or np.array @parameter: u_mean, type = list or np.array @parameter: v_mean, type = list or np.array @parameter: heights, type = list or np.array @parameter: yerr, type = int or float @parameter: lat, type = boolean @parameter ax: axis passed to function @parameter **kwargs : additional keyword arguments passed to plt.plot()

windtunnel.**plot_winddata_log**(*mean_magnitude*, *u_mean*, *v_mean*, *heights*, *yerr=0*, *ax=None*, ***kwargs*)

Plots wind components and wind magnitude for their respective height on a log scale. yerr specifies the uncertainty. Its default value is 0. @parameter: mean_magnitude, type = list or np.array @parameter: u_mean, type = list or np.array @parameter: v_mean, type = list or np.array @parameter: heights, type = list or np.array @parameter: yerr, type = int or float @parameter: ax, axis passed to function @parameter **kwargs : additional keyword arguments passed to plt.plot()

windtunnel.**plotcdfs**(*sets*, *lablist*, *ax=None*, ***kwargs*)
Plots CDFs of data in sets using the respective labels from lablist @parameter sets: iterable set of data @parameter lablist: list of strings @parameter ax: axis passed to function @parameter **kwargs : additional keyword arguments passed to plt.plot()

windtunnel.**plotpdfs**(*sets*, *lablist*, *ax=None*, ***kwargs*)
Plots PDFs of data in sets using the respective labels from lablist. @parameter sets: iterable set of data @parameter lablist: list of strings @parameter ax: axis passed to function @parameter **kwargs : additional keyword arguments passed to plt.plot()

windtunnel.**plotpdfs_err**(*sets*, *lablist*, *error*, *ax=None*, ***kwargs*)
Plots PDFs of data in sets using the respective labels from lablist with a given margin of error. @parameter sets: iterable set of data @parameter lablist: list of strings @parameter error: int or float @parameter ax: axis passed to function @parameter **kwargs : additional keyword arguments passed to plt.plot()

windtunnel.**plotwindrose**(*inFF*, *inDD*, *num_bars=10*, *ax=None*, *left_legend=False*)
Plots windrose with dynamic velocity classes of each 10% percentile and 10 degree classes for directional data. The representation of the windrose in this function is more detailed than in plot_DWD_windrose(). @parameter inFF: np.array @parameter inDD: np.array @parameter num_bars: how many segments the degree range should be broken

into

@parameter ax: pyplot axes object, must be polar @left_legend: if true, the legend is positioned to the left of the plot

instead of the right

windtunnel.**power_law**(*u_comp*, *height*, *u_ref*, *z_ref*, *alpha*, *d0=0*)
Estimate power law profile. @parameter: u_comp, type = int or float @parameter: height, type = int or float @parameter: u_ref, type = int or float @parameter: z_ref, type = int or float @parameter: alpha, type = int or float @parameter: d0, type = int or float

windtunnel.**transit_time_weighted_flux**(*transit_time*, *component_1*, *component_2*)
Calculate mean flux using transit time weighted statistics. Transit time weighting removes a possible bias towards higher wind velocities. Returns a mean weighted flux. @parameter: transit_time, type = np.arrray([]) @parameter: component_1, type = np.arrray([]) @parameter: component_2, type = np.arrray([])

windtunnel.**transit_time_weighted_mean**(*transit_time*, *component*)
Weigh the flow component with its transit time through the measurement volume. This is analoguous to the processing of the raw data in the BSA software. Transit time weighting removes a possible bias towards higher wind velocities. Returns the weighted component mean. @parameter: transit_time, type = np.arrray([]) @parameter: component, type = np.arrray([])

windtunnel.**transit_time_weighted_var**(*transit_time*, *component*)
Weigh the u and v component with its transit time through the measurement volume. This is analoguous to the processing of the raw data in the BSA software. Transit time weighting removes a possible bias towards higher wind velocities. Returns the weighted u and v component variance. @parameter: transit_time, type = np.arrray([]) @parameter: component, type = np.arrray([])

windtunnel.**trunc_at**(*string*, *delimiter*, *n=3*)
Returns string truncated at the n'th (3rd by default) occurrence of the delimiter.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

W

INDEX

T

W