pydoas Documentation

Release 1.2.2

Jonas Gliss

Feb 21, 2018

Contents

pydoas provides high level functionality for import of text file based results from DOAS analyses (i.e. it cannot perform the actual DOAS retrieval). It supports default import routines for the result file format of the software DOASIS.

Note: pydoas is written for Python 2.7

Contents:

CHAPTER 1

Introduction

pydoas is a Python library for reading, postprocessing and plotting of DOAS (Differential Optical Absorption Spectroscopy) fit results. It supports default import of the result file format of DOASIS. Further import formats for fast data access can easily be defined by the user.

1.1 News

18 Feb 2018: Updated for Python 3 support

1.2 Requirements

- numpy
- matplotlib
- pandas

1.3 Copyright

1.4 Installation

pydoas can be installed from PyPi using:

```
pip install pydoas
```

or from source (hosted on GitHub) by downloading and extracting the latest release or downloading / cloning the repo. If you download manually, call:

```
python setup.py install
```

after download and extraction of the source directory.

1.5 Instructions and code documentation

The code documentation of pydoas is hosted here

1.6 Get started

After installation try running the example scripts in order to test the installation. The scripts are also meant to provide an easy start into the main features of pydoas.

CHAPTER 2

Example scripts

On this page, some exemplary example scripts are presented, the scripts can be downloaded from the webpage.

2.1 Example 1 - Result import and plotting

code

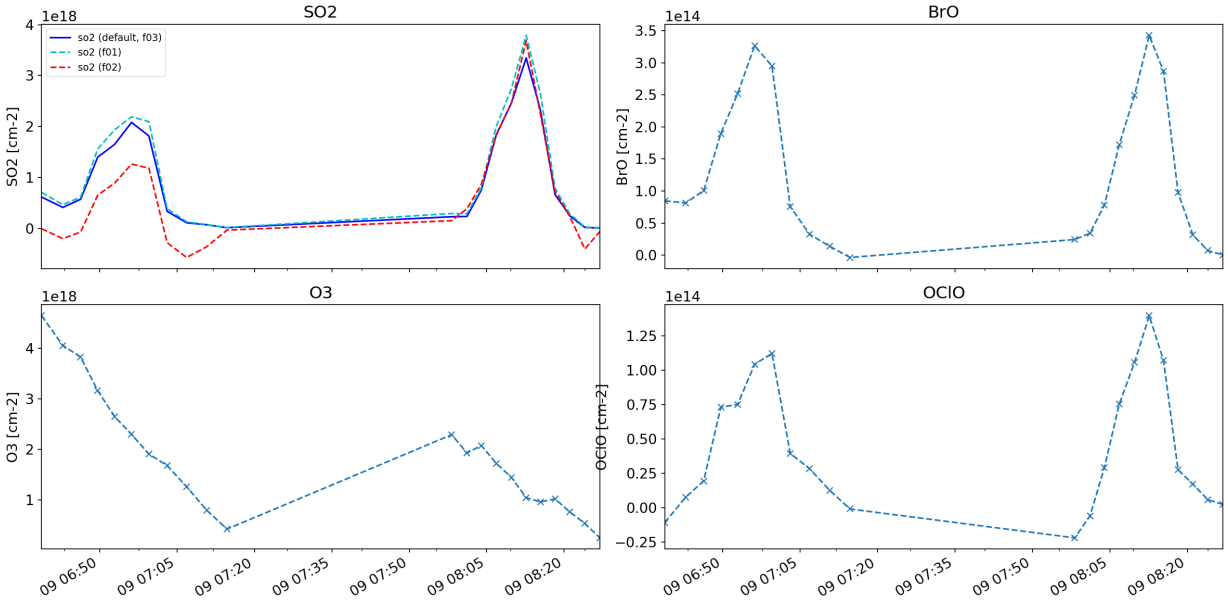
```
# -*- coding: utf-8 -*-
"""pydoas example script 1

Introductory script illustrating how to import data from DOASIS
resultfiles. The DOASIS result file format is specified as default in the
package data file "import_info.txt".
Creates a result data set using DOASIS example data and plots some examples

"""
import pydoas
import matplotlib.pyplot as plt
from os.path import join

from SETTINGS import SAVE_DIR, SAVEFIGS, OPTPARSE, DPI, FORMAT

if __name__=="__main__":
    plt.close("all")
    ### Get example data base path and all files in there
    files, path = pydoas.get_data_files("doasis")

    ### Device ID of the spectrometer (of secondary importance)
    dev_id = "avantes"

    ### Data import type (DOASIS result file format)
    res_type = "doasis"

    ### Specify the the import details
```

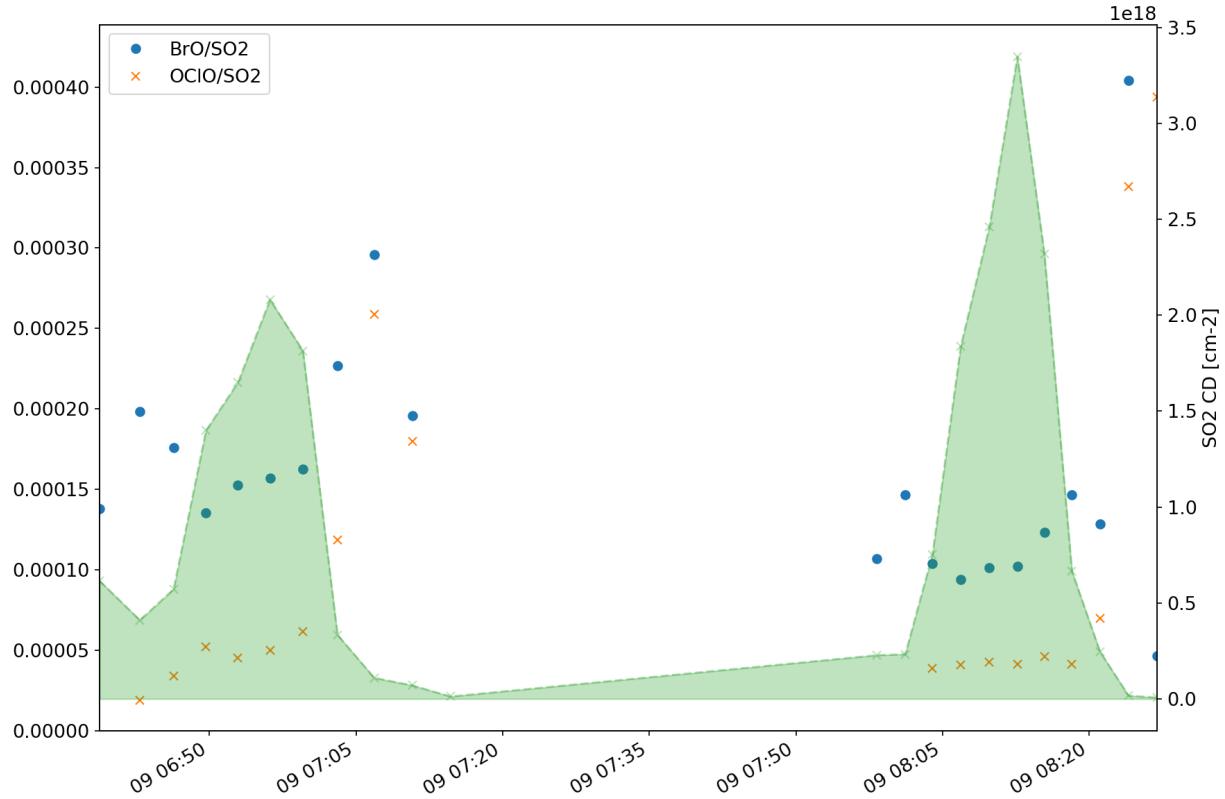
```
# here, 3 x SO2 from the first 3 fit scenario result files (f01, f02, f03)
# BrO from f04, 2 x O3 (f02, f04) and OClO (f04)
import_dict = {'so2'  : ['SO2_Hermans_298_air_conv',\
                                                ['f01','f02','f03']],
               'bro'  : ['BrO_Wil298_Air_conv',['f04']],
               'o3'   : ['o3_221K_air_burrows_1999_conv',\
                                                  ['f02', 'f04']],
               'oclo' : ['OClO_293K_Bogumil_2003_conv',['f04']]}

### Specify the default fit scenarios for each species

# After import, the default fit scenarios for each species are used
# whenever fit scenarios are not explicitely specified
default_dict = {"so2"  :   "f03",
                "bro"  :   "f04",
                "o3"   :   "f04",
                "oclo" :   "f04"}

#: Create import setup object
stp = pydoas.dataimport.ResultImportSetup(path, result_import_dict =\
        import_dict, default_dict = default_dict, meta_import_info = res_type,\
        dev_id = dev_id)

#: Create Dataset object for setup...
ds = pydoas.analysis.DatasetDoasResults(stp)

#: ... and load results
ds.load_raw_results()

### plot_some_examples
fig1, axes = plt.subplots(2, 2, figsize = (16, 8), sharex = True)
ax = axes[0,0]

#load all SO2 results
so2_default = ds.get_results("so2")
so2_fit01 = ds.get_results("so2", "f01")
so2_fit02 = ds.get_results("so2", "f02")

#plot all SO2 results in top left axes object
so2_default.plot(style="-b", ax=ax, label="so2 (default, f03)")
so2_fit01.plot(style="--c", ax=ax, label="so2 (f01)")
so2_fit02.plot(style="--r", ax=ax, label="so2 (f02)").set_ylabel("SO2 [cm-2]")
ax.legend(loc='best', fancybox=True, framealpha=0.5, fontsize=9)
ax.set_title("SO2")
fig1.tight_layout(pad = 1, w_pad = 3.5, h_pad = 3.5)

#now load the other species and plot them into the other axes objects
bro=ds.get_results("bro")
bro.plot(ax=axes[0, 1], label="bro", title="BrO").set_ylabel("BrO [cm-2]")

o3=ds.get_results("o3")
o3.plot(ax=axes[1, 0], label="o3",
        title="O3").set_ylabel("O3 [cm-2]")
oclo=ds.get_results("oclo")
oclo.plot(ax=axes[1, 1], label="oclo",
          title="OClO").set_ylabel("OClO [cm-2]")
```

```
    # Now calculate Bro/SO2 ratios of the time series and plot them with
    # SO2 shaded on second y axis
    bro_so2 = bro/so2_default
    oclo_so2 = oclo/so2_default

    fig2, axis = plt.subplots(1,1, figsize=(12,8))
    bro_so2.plot(ax=axis, style=" o", label="BrO/SO2")
    oclo_so2.plot(ax=axis, style=" x", label="OClO/SO2")
    #axis.set_ylabel("BrO/SO2")
    so2_default.plot(ax=axis, kind="area",
                     secondary_y=True, alpha=0.3).set_ylabel("SO2 CD [cm-2]")
    axis.legend()
    if SAVEFIGS:
        fig1.savefig(join(SAVE_DIR, "ex1_out1.%s" %FORMAT),
                     format=FORMAT, dpi=DPI)
        fig2.savefig(join(SAVE_DIR, "ex1_out2.%s" %FORMAT),
                     format=FORMAT, dpi=DPI)

    ### IMPORTANT STUFF FINISHED (Below follow tests and display options)

    # Import script options
    (options, args) = OPTPARSE.parse_args()

    # If applicable, do some tests. This is done only if TESTMODE is active:
    # testmode can be activated globally (see SETTINGS.py) or can also be
    # activated from the command line when executing the script using the
    # option --test 1
    if int(options.test):
        ### under development
        import numpy.testing as npt
        import numpy as np
        from os.path import basename

        npt.assert_array_equal([len(so2_default),
                                ds.get_default_fit_id("so2"),
                                ds.get_default_fit_id("bro"),
                                ds.get_default_fit_id("oclo")],
                               [22, "f03", "f04", "f04"])

        vals = [so2_default.mean(),
                so2_default.std(),
                so2_fit01.mean(),
                so2_fit02.mean(),
                bro.mean(),
                oclo.mean(),
                bro_so2.mean(),
                oclo_so2.mean(),
                np.sum(ds.raw_results["f01"]["delta"])]

        npt.assert_allclose(actual=vals,
                            desired=[9.626614500000001e+17,
                                     9.785535879339162e+17,
                                     1.0835821818181818e+18,
                                     6.610916636363636e+17,
                                     126046170454545.45,
                                     42836762272727.27,
                                     0.0001389915245877655,
                                     7.579933107191676e-05,
```

```
                                    0.125067],
                            rtol=1e-7)
    print("All tests passed in script: %s" %basename(__file__))
try:
    if int(options.show) == 1:
        plt.show()
except:
    print("Use option --show 1 if you want the plots to be displayed")
```

Code output





2.2 Example 2 - Define new import type and load fake data

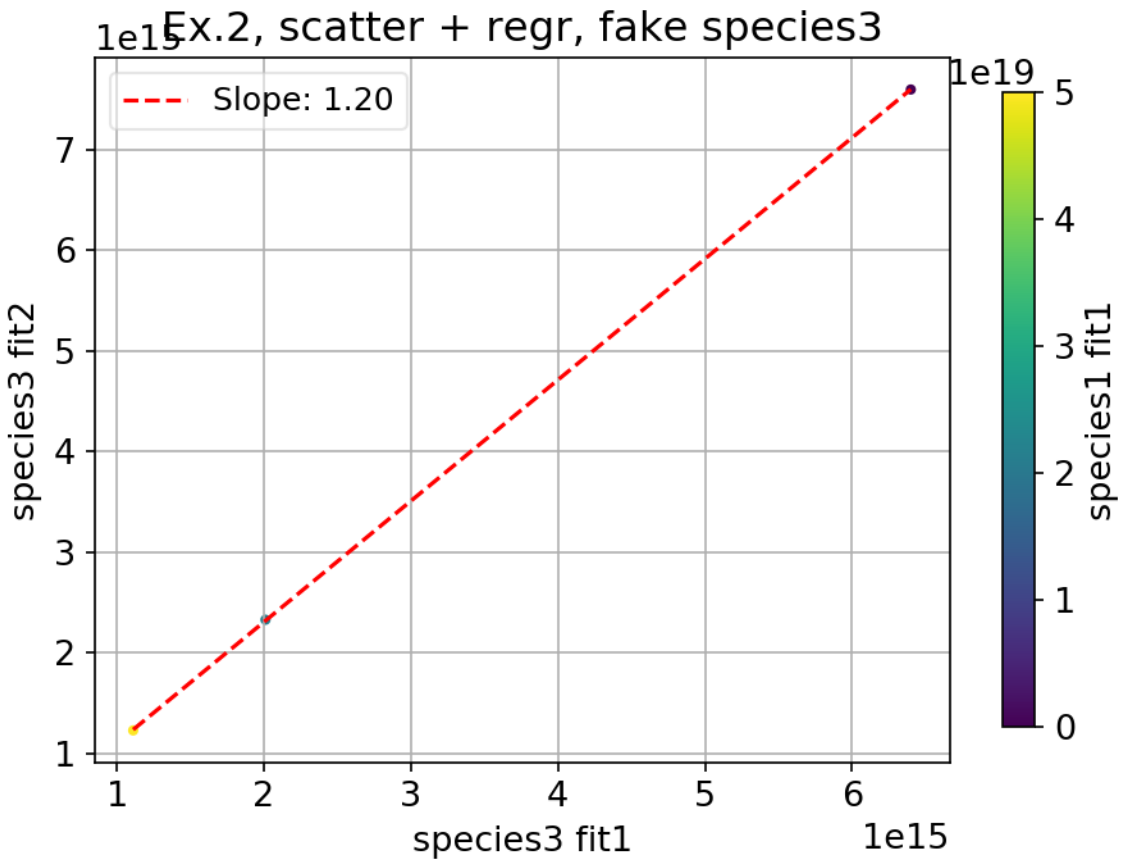
code

```
# -*- coding: utf-8 -*-
"""pydoas example script 2

In this script, a new import type is defined aiming to import the fake
data files in the package data folder "./data/fake_resultfiles
In this example, the import is performed based on specifications of the
data columns rather than based on searching the header lines of the
resultfiles.

After import, the an exemplary scatter plot of fake species 3 is performed
for the two fit IDs specified.

"""
import pydoas
from os.path import join
from collections import OrderedDict as od

from SETTINGS import SAVE_DIR, SAVEFIGS, OPTPARSE, DPI, FORMAT

### Path for output storage
out_path = join(".", "scripts_out")

if __name__ == "__main__":
```

```
    ### create some fake results:

    ### Get data path
    files, path = pydoas.inout.get_data_files(which="fake")

    ### Specify default import parameters
    # (this does not include resultfile columns of DOAS fit results)
    meta_import_info =  od([("type", "fake"),
                            ("access_type", "col_index"),
                            ("has_header_line" , 1),
                            ("time_str_formats", ["%Y%m%d%H%M"]),
                            ("file_type", "csv"),
                            ("delim", ";"),
                            ("start", 0), #col num
                            ("stop", 1), #col num
                            ("bla" , "Blub"), #invalid (for test purpose)
                            ("num_scans", 4)]) #colnum

    # You could create a new default data type now by calling
    # pydoas.inout.write_import_info_to_default_file(meta_import_info)
    # which would add these specs to the import_info.txt file and which
    # would allow for fast access using
    # meta_info_dict = pydoas.inout.get_import_info("fake")

    import_dict = {'species1'   : [2, ['fit1']], #column 2, fit result file 1
                   'species2'   : [2, ['fit2']], #column 2, fit result file 2
                   'species3'   : [3, ['fit1', 'fit2']]} #column 3, fit result file 1␣
↪and 2

    stp = pydoas.dataimport.ResultImportSetup(path, result_import_dict =\
                import_dict, meta_import_info = meta_import_info)

    #: Create Dataset object for setup...
    ds = pydoas.analysis.DatasetDoasResults(stp)

    ax = ds.scatter_plot("species3", "fit1", "species3", "fit2",\
                    species_id_zaxis = "species1", fit_id_zaxis = "fit1")
    ax.set_title("Ex.2, scatter + regr, fake species3")

    if SAVEFIGS:
        ax.figure.savefig(join(SAVE_DIR, "ex2_out1_scatter.%s" %FORMAT),
                          format=FORMAT, dpi=DPI)

    ### IMPORTANT STUFF FINISHED (Below follow tests and display options)

    # Import script options
    (options, args) = OPTPARSE.parse_args()

    # If applicable, do some tests. This is done only if TESTMODE is active:
    # testmode can be activated globally (see SETTINGS.py) or can also be
    # activated from the command line when executing the script using the
    # option --test 1
    if int(options.test):
        import numpy.testing as npt
        from os.path import basename

        npt.assert_array_equal([],
                               [])
```

```
    # check some propoerties of the basemap (displayed in figure)

    npt.assert_allclose(actual=[],
                        desired=[],
                        rtol=1e-7)
    print("No tests implemented in script: %s" %basename(__file__))
try:
    if int(options.show) == 1:
        from matplotlib.pyplot import show
        show()
except:
    print("Use option --show 1 if you want the plots to be displayed")
```

Code output



CHAPTER 3

API

Code documentation of Pydoas API.

3.1 Data import

class pydoas.dataimport.**ResultImportSetup**(*base_dir=None*, *start=datetime.datetime(1900, 1, 1, 0, 0)*, *stop=datetime.datetime(3000, 1, 1, 0, 0)*, *meta_import_info='doasis'*, *result_import_dict={}*, *default_dict={}*, *doas_fit_err_factors={}*, *dev_id=''*, *lt_to_utc_offset=datetime.timedelta(0)*)

Setup class for spectral result imports from text like files

__init__(*base_dir=None*, *start=datetime.datetime(1900, 1, 1, 0, 0)*, *stop=datetime.datetime(3000, 1, 1, 0, 0)*, *meta_import_info='doasis'*, *result_import_dict={}*, *default_dict={}*, *doas_fit_err_factors={}*, *dev_id=''*, *lt_to_utc_offset=datetime.timedelta(0)*)

Parameters

- **base_dir** (*str*) – folder containing resultfiles
- **start** (*datetime*) – time stamp of first spectrum
- **stop** (*datetime*) – time stamp of last spectrum
- **meta_import_info** – Specify the result file format and columns for meta information (see als file import_info.txt or example script 2). Input can be str or dict. In case a string is provided, it is assumed, that the specs are defined in import_info.txt, i.e. can be imported (as dictionary) from this file (using `get_import_info()`, e.g. with arg = `doasis`). If a dictionary is provided, the information is directly set from the provided dictionary.
- **result_import_dict** (*dict*) – specify file and header information for import. Keys define the used abbreveations after import, the values to each key consist of a list with 2 elements: the first specifies the UNIQUE string which is used to identify this species in the header of a given Fit result file, the second entry is a list with arbitrary length containing

the fit scenario IDs defining from which fit scenario result files this specific species is to be extracted.

Example:

```
result_import_dict = {"so2" : ['SO2_Hermans', ['f01','f02']],
                      "o3"  : ['o3_burrows'], ['f01']]}
```

Here `so2` and "o3" are imported, the data column in the result files is found by the header string `'SO2_Hermans'` / `'o3_burrows'` and this species is imported from all fit scenario result files with fit Ids `["f01", "f02"]` (UNIQUE substrings in FitScenario file names.

Exemplary file name:

```
D130909_S0628_i6_f19_r20_f01so2.dat
```

This (exemplary) filename convention is used for the example result files shipped with this package (see folder pydoas/data/doasis_resultfiles) which include fit result files from the software DOASIS.

The delimiter for retrieving info from these file names is "_", the first substring provides info about the date (day), the second about the start time of this time series (HH:MM), 3rd, 4th and 5th information about first and last fitted spectrum number and the corresponding number of the reference spectrum used for this time series and the last index about the fit scenario (fitID).

Each resultfile must therefore include a unique ID in the file name by which it can be identified.

- **default_dict** (*dict*) – specify default species, e.g.:

```
dict_like = {"so2"     :   "f02",
             "o3"      :   "f01"}
```

- **doas_fit_err_factors** (*dict*) – fit correction factors (i.e. factors by which the DOAS fit error is increased):

```
dict_like = {"so2"     :   "f02",
             "o3"      :   "f01"}
```

- **dev_id** (*str*) – string ID for DOAS device (of minor importance)
- **lt_to_utc_offset** (*timedelta*) – specify time zone offset (will be added on data import if applicable).

start
Start time-stamp of data

stop
Stop time-stamp of data

base_path
Old name of base_dir for versions <= 1.0.1

set_start_time(*dt*)
Set the current start time

Parameters **dt** (*datetime*) – start time of dataset

set_stop_time(*dt*)
Set the current start time

Parameters **dt** (*datetime*) – start time of dataset

check_time_stamps()
Check if time stamps are valid and if not, set

complete()
Checks if basic information is available

set_defaults(*dict_like*)
Update default fit IDs for fitted species

Scheme:

```
dict_like = {"so2"     :   "f02",
             "o3"      :   "f01"}
```

set_fitcorr_factors(*dict_like*)
Set correction factors for uncertainty estimate from DOAS fit errors

Parameters **dict_like** (*dict*) – dictionary specifying correction factors for DOAS fit errors (which are usually underestimated, see e.g. Gliss et al. 2015) for individual fit scenarios, e.g.:

```
dict_like = {"f01"   :   4.0,
             "f02"   :   2.0}
```

Default value is 3.0.

xs
Returns list with xs names

get_xs_names()
Set and return the string IDs of all fitted species

get_fit_ids_species(*species_id*)
Find all fit scenarios which contain results of species

Parameters **species_id** (*str*) – string ID of fitted species (e.g. SO2)

fit_ids
Returns list with all fit ids

access_type
Return the current setting for data access type

HEADER_ACCESS_OPT
Checks if current settings allow column identification from file header line

FIRST_DATA_ROW_INDEX

get_fit_ids()
Get all fit id abbreveations

Gets all fit ids (i.e. keys of fit import dict `self.import_info`)

class pydoas.dataimport.**DataImport**(*setup=None*)
A class providing reading routines of DOAS result files

Here, it is assumed, that the results are stored in FitResultFiles, tab delimited whereas the columns correspond to the different variables (i.e. fit results, metainfo, ...) and the rows to the individual spectra.

__init__(*setup=None*)

get_data()
Load all data

load_result_type_info()
Load import information for result type specified in setup

The detailed import information is stored in the package data file import_info.txt, this file can also be used to create new filetypes

base_dir
Returns current basepath of resultfiles

start
Returns start date and time of dataset

stop
Returns stop date and time of dataset

time_str_format
Returns datetime formatting info for string to datetime conversion

This information should be available in the resultfile type specification file (package data: data/import_info.txt)

fit_err_add_col
Return current value for relative column of fit errors

init_result_dict()
Initiate the result dictionary

find_valid_indices_header(*fileheader*, *dict*)
Find positions of species in header of result file

Parameters

- **fileheader** (*list*) – header row of resultfile
- **dict** (*dict*) – dictionary containing species IDs (keys) and the corresponding (sub) strings (vals) to find them in the header

find_all_indices(*fileheader*, *fit_id*)
Find all relevant indices for a given result file (fit scenario)

Parameters

- **fileheader** (*list*) – list containing all header strings from result file (not required if data access mode is from columns see also `HEADER_ACCESS_OPT()` in *ResultImportSetup*)
- **fit_id** (*str*) – ID of fit scenario (required in order to find all fitted species supposed to be extracted, specified in `self.setup.import_info`)

load_results()
Load all results

The results are loaded as specified in `self.import_setup` for all valid files which were detected in *get_all_files()* which writes `self.file_paths`

find_col_index(*substr*, *header*)
Find the index of the column in data

Parameters

- **substr** (*str*) – substr identifying the column in header
- **header** (*list*) – the header of the data in which index of substr is searched

check_time_match(*data*)
Check if data is within time interval set by self.start and self.stop

Parameters **data** (*list*) – data as read by *read_text_file()*

Returns

- bool, Match or no match

first_file
Get filepath of first file match in `self.base_dir`

This can for instance be read with *read_text_file()*

init_filepaths()
Initate the file paths

get_all_files()
Get all valid files based on current settings

Checks `self.base_dir` for files matching the specified file type, and which include one of the required fit IDs in their name. Files matching these 2 criteria are opened and the spectrum times are read and checked. If they match the time interval specified by `self.start` and `self.stop` the files are added to the dictionary `self.file_paths` where the keys specify the individual fit scenario IDs.

Note: This function does not load data but only assigns the individual result files to the fit IDs, the data will then be loaded calling *load_results()*

---

read_text_file(*p*)
Read text file using csv.reader and return data as list

Parameters **p** (*str*) – file path

Returns list data

3.2 Fit result analysis and plotting

class pydoas.analysis.**DatasetDoasResults**(*setup=None*, *init=1*, ***kwargs*)
A Dataset for DOAS fit results import and processing

__init__(*setup=None*, *init=1*, ***kwargs*)
Initialisation of object

Parameters

- **setup** (ResultImportSetup) – setup specifying all necessary import settings (please see documentation of `ResultImportSetup` for setup details)
- ****kwargs** – alternative way to setup `self.setup` (`ResultImportSetup` object), which is only used in case no input parameter **setup** is invalid. Valid keyword arguments are input parameters of `ResultImportSetup` object.

load_input(*setup=None*, ***kwargs*)
Process input information

Writes `self.setup` based on setup

Parameters

- **setup** – is set if valid (i.e. if input is `ResultImportSetup`)

- ****kwargs** –
  - keyword arguments for new `ResultImportSetup`

  (are used in case first parameter is invalid)

base_path
Returns current basepath of resultfiles (from `self.setup`)

start
Returns start date and time of dataset (from `self.setup`)

stop
Returns stop date and time of dataset (from `self.setup`)

dev_id
Returns device ID of dataset (from `self.setup`)

import_info
Returns information about result import details

change_time_ival(*start*, *stop*)
Change the time interval for the considered dataset

Parameters

- **start** (*datetime*) – new start time
- **stop** (*datatime*) – new stop time

Note: Previously loaded results will be deleted

load_raw_results()
Try to load all results as specified in `self.setup`

has_data(*fit_id*, *species_id*, *start=None*, *stop=None*)
Checks if specific data is available

get_spec_times(*fit*)
Returns start time and stop time arrays for spectra to a given fit

set_start_stop_time()
Get start/stop range of dataset

get_start_stop_mask(*fit*, *start=None*, *stop=None*)
Creates boolean mask for data access only in a certain time interval

get_meta_info(*fit*, *meta_id*, *start=None*, *stop=None*)
Get meta info array

Parameters

- **meta_id** (*str*) – string ID of meta information
- **boolMask** (*array*) – boolean mask for data retrieval

Note: Bool mask must have same length as the meta data array

get_results(*species_id*, *fit_id=None*, *start=None*, *stop=None*)
Get spectral results object

Parameters

- **species_id** (*str*) – string ID of species
- **fit_id** (*str*) – string ID of fit scenario (if None, tries to load default fit_id)
- **start** – if valid (i.e. datetime object) only data after that time stamp is considered
- **stop** – if valid (i.e. datetime object) only data before that time stamp is considered

get_default_fit_id(*species_id*)
Get default fit scenario id for species

Parameters **species_id** (*str*) – ID of species (e.g. "so2")

set_default_fitscenarios(*default_dict*)
Update default fit scenarios for species

Parameters **default_dict** (*dict*) – dictionary specifying new default fit scenarios, it could e.g. look like:

```
default_dict = {"so2"   :   "f01",
                "o3"    :   "f01",
                "bro"   :   "f03"}
```

plot(*species_id*, *fit_id=None*, *start=None*, *stop=None*, ***kwargs*)
Plot DOAS results

scatter_plot(*species_id_xaxis*, *fit_id_xaxis*, *species_id_yaxis*, *fit_id_yaxis*, *lin_fit_opt=1*, *species_id_zaxis=None*, *fit_id_zaxis=None*, *start=None*, *stop=None*, *ax=None*, ***kwargs*)
Make a scatter plot of two species

Parameters

- **species_id_xaxis** (*str*) – string ID of x axis species (e.g. "so2")
- **fit_id_xaxis** (*str*) – fit scenario ID of x axis species (e.g. "f01")
- **species_id_yaxis** (*str*) – string ID of y axis species (e.g. "so2")
- **fit_id_yaxis** (*str*) – fit scenario ID of y axis species (e.g. "f02")
- **species_id_zaxis** (*str*) – string ID of z axis species (e.g. "o3")
- **fit_id_zaxis** (*str*) – fit scenario ID of z axis species (e.g. "f01")

:param bool linF

linear_regression(*x_data*, *y_data*, *mask=None*, *ax=None*)
Perform linear regression and return parameters

Parameters

- **x_data** (*ndarray*) – x data array
- **y_data** (*ndarray*) – y data array
- **mask** (*ndarray*) – mask specifying indices of input data supposed to be considered for regression (None)
- **ax** – matplotlib axes object (None), if provided, then the result is plotted into the axes

get_fit_import_setup()
Get the current fit import setup

class pydoas.analysis.**DoasResults**(*data*, *index=None*, *start_acq=[]*, *stop_acq=[]*, *fit_errs=None*, *species_id=''*, *fit_id=''*, *fit_errs_corr_fac=1.0*)

Data time series object inheriting from `pandas.Series` for handling and analysing DOAS fit results

Parameters

- **data** (*arraylike*) – DOAS fit results (column densities)
- **index** (*arraylike*) – Time stamps of data points
- **fit_errs** (*arraylike*) – DOAS fit errors
- **species_id** (*string*) – String specifying the fitted species
- **fit_id** (*string*) – Unique string specifying the fit scenario used
- **fit_errs_corr_fac** (*int*) – DOAS fit error correction factor

---

Todo: Finish magic methods, i.e. apply error propagation, think about time merging etc...

__init__(*data*, *index=None*, *start_acq=[]*, *stop_acq=[]*, *fit_errs=None*, *species_id=''*, *fit_id=''*, *fit_errs_corr_fac=1.0*)

fit_errs = None

fit_id = None

fit_errs_corr_fac = None

start_acq = []

stop_acq = []

start
Start time of data

stop
Stop time of data

species
Return name of current species

has_start_stop_acqtamps()
Checks if start_time and stop_time arrays have valid data

merge_other(*other*, *itp_method='linear'*, *dropna=True*)
Merge with other time series sampled on different grid

Note: This object will not be changed, instead, two new Series objects will be created and returned

---

Parameters

- **other** (*Series*) – Other time series
- **itp_method** (*str*) – String specifying interpolation method (e.g. linear, quadratic)
- **dropna** (*bool*) – Drop indices containing NA after merging and interpolation

Returns

2-element tuple containing

- this Series (merged)
- other Series (merged)

Return type tuple

get_data_above_detlim()
Get fit results exceeding the detection limit

The detection limit is determined as follows:

```
self.fit_errs_corr_fac*self.data_err
```

plot(*date_fmt=None*, ***kwargs*)
Plot time series

Uses plotting utility of `Series` object (pandas)

Parameters ****kwargs** –

- keyword arguments for pandas plot method

shift(*timedelta=datetime.timedelta(0)*)
Shift time stamps of object

Parameters **timedelta** (*timedelta*) – temporal shift

Returns shifted *DoasResults* object

3.3 Supplemental / IO / Helpers

This module contains I/O routines for DOAS result files

pydoas.inout.**get_data_dirs**()
Get directories containing example package data

Returns list of package subfolders containing data files

pydoas.inout.**get_data_files**(*which=u'doasis'*)
Get all example result files from package data

pydoas.inout.**get_result_type_ids**()
Read file import_info.txt and find all valid import types

pydoas.inout.**import_type_exists**(*type_id*)
Checks if data import type exists in import_info.txt

Parameters **type_id** (*str*) – string ID to be searched in import_info.txt

pydoas.inout.**get_import_info**(*resulttype=u'doasis'*)
Try to load DOAS result import specification for default type

Import specifications for a specified data type (see package data file "import_info.txt" for available types, use the instructions in this file to create your own import setup if necessary)

Parameters **resulttype** (*str*) – name of result type (field "type" in "import_info.txt" file)

pydoas.inout.**import_info_file**()
Return path to supplementary file import_info.txt

pydoas.inout.**write_import_info_to_default_file**(*import_dict*)

CHAPTER 4

Changelog

This file keeps track of major changes applied to the code after the first release of pydoas (version 1.0.1)

4.1 06/03/2017

1. Expanded handling of start / stop time specifications in Setup classes and Dataset classes (for initialisation of working environment) -> now, the user can also provide time stamps (datetime.time objects) or dates (datetime.date objects) as input and they will converted automatically to datetime.
2. Included new module helpers.py (helper methods)
3. Included date formatting option in time series plots of DoasResults class
4. Included merging functionality for DoasResults class: method `merge_other`

CHAPTER 5

Remarks for the DOAS analysis

This code does not include any features to perform the DOAS analysis itself but only for reading and visualising DOAS fit results retrieved using a suitable DOAS analysis software (e.g. DOASIS) with fit results being stored in text files (.txt, .csv, .dat, etc.).

The result files are expected to be formatted in a tabular style where columns define individual parameters (e.g. fitted gas column densities of individual species, start acquistion, number of co-added spectra) and the rows correspond to individual spectra. Each result file hence corresponds to a certain *time interval* (containing *N* spectra) and to one *specific DOAS fit scenario* (i.e. fit interval, fitted species, DOAS fit settings, etc). The file names of the result files are therefore required to include a **unique ID which can be used on data import in order to identify the fit scenario** (e.g. f01so2, f02bro). Furthermore, the result files are required to include a header row which is used to identify the individual import columns and which can be specified in the file *import_info.txt* which is part of the package data.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

Symbols

A

B

C

D

F

G