hatyan-2.2.84 documentation, automatically generated from script comments and function docstrings

Contents

Module `hatyan`

hatyan is a Python program for tidal analysis and prediction, based on the FORTRAN version. Copyright (C) 2019-2020 Rijkswaterstaat. Maintained by Deltares, contact: Jelmer Veenstra (jelmer.veenstra@deltares.nl). Source code available at: https://repos.deltares.nl/repos/lib_tide/trunk/src/hatyan_python/hatyan

Installation

Install hatyan OPTION 1: get and install RPM on CentOS/RHEL

- get the latest rpm file (see developer information for building procedure)
- install hatyan on CentOS: `rpm -i hatyan_python-2.2.30-1.x86_64.rpm`
- upgrade hatyan on CentOS: `rpm -U hatyan_python-2.2.31-1.x86_64.rpm`
- installing the RPM results in a hatyan command in linux, this activates a Python virtual environment and sets necessary Qt environment variables. It creates a folder with a python environment hatyan_env, doc en tests (/opt/hatyan_python/hatyan_env/) and a file that provides the hatyan command (/usr/bin/hatyan)
- check version: `hatyan --version`
- test installation: `hatyan /opt/hatyan_python/tests/configfiles/predictie_2019_19Ycomp`
- this should result in several interactive figures popping up, described in chapter 5 (Quick start guide) of the hatyan user manual (gebruikershandleiding).
- if you see the message "RuntimeError: Invalid DISPLAY variable", restart the MobaXterm connection and try again.
- the followning warning can be ignored: "QXcbConnection: XCB error: 145 (Unknown), sequence: 171, resource id: 0, major code: 139 (Unknown), minor code: 20". To avoid it, disable the extension RANDR in Mobaxterm settings (Settings > Configuration > X11)

Install hatyan OPTION 2: no hatyan installation, use existing checkout (this example is only possible on the Deltares network):

- download Anaconda 64 bit Python 3.7 (or higher) from https://www.anaconda.com/distribution/#download-section (miniconda should also be sufficient, but this is not yet tested)
- install it with the recommended settings, but check 'add Anaconda3 to my PATH environment variable' if you want to use conda from the windows command prompt instead of anaconda prompt
- add to the top of your script `sys.path.append(r'n:\Deltabox\Bulletin\veenstra\hatyan_py`

Install hatyan OPTION 3: Create a separate python environment hatyan_env and install from github or even PyPI (not yet possible, this is just an example):

- download Anaconda 64 bit Python 3.7 (or higher) from https://www.anaconda.com/distribution/#download-section (miniconda should also be sufficient, but this is not yet tested)
- install it with the recommended settings, but check 'add Anaconda3 to my PATH environment variable' if you want to use conda from the windows command prompt instead of anaconda prompt
- open command window (or anaconda prompt)
- `conda create --name hatyan_env -c conda-forge python=3.7 git spyder -y` (or higher python version)
- conda activate hatyan_env
- `python -m pip install git+https://github.com/openearth/hatyan.git` (this command installs hatyan and all required packages)
- to update hatyan: `python -m pip install --upgrade git+https://github.com/openearth/`
- conda deactivate
- to remove venv when necessary: `conda remove -n hatyan_env --all`

Example Usage

Copy the code below to your own script to get started.

```
import os, sys
sys.path.append(r'n:\Deltabox\Bulletin\veenstra\hatyan_python')
import datetime as dt

from hatyan import timeseries as Timeseries
from hatyan import components as Components
from hatyan.analysis_prediction import get_components_from_ts, prediction
from hatyan.hatyan_core import get_const_list_hatyan

#defining a list of the components to be analysed (can also be 'half_year' and othe
const_list = get_const_list_hatyan('year')

#reading and editing time series, results in a pandas DataFrame a 'values' column (
file_data_meas = os.path.join(r'n:\Deltabox\Bulletin\veenstra\VLISSGN_waterlevel_2010
times_ext = [dt.datetime(2012,1,1),dt.datetime(2013,1,1)]
```

```
timestep_min = 10
ts_meas = Timeseries.readts_noos(filename=file_data_meas)
ts_meas = Timeseries.resample_timeseries(ts_meas, timestep_min=timestep_min)
ts_meas = Timeseries.crop_timeseries(ts=ts_meas, times_ext=times_ext)

#tidal analysis and plotting of results. commented: saving figure
comp_frommeas = get_components_from_ts(ts=ts_meas, const_list=const_list, nodalfactor
fig,(ax1,ax2) = Components.plot_components(comp=comp_frommeas)
#fig.savefig('components_VLISSGN_4Y.png')

#tidal prediction and plotting of results. commented: saving figure and writing to 
ts_prediction = prediction(comp=comp_frommeas, nodalfactors=True, xfac=True, fu_allti
fig, (ax1,ax2) = Timeseries.plot_timeseries(ts=ts_prediction, ts_validation=ts_meas)
ax1.legend(['prediction','measurement','difference','mean of prediction'])
ax2.set_ylim(-0.5,0.5)
#fig.savefig('prediction_%im_VLISSGN_measurements'%(timestep_min))

#calculation of HWLW and plotting of results. commented: saving figure
ts_ext_prediction = Timeseries.calc_HWLW(ts=ts_prediction)
fig, (ax1,ax2) = Timeseries.plot_timeseries(ts=ts_prediction, ts_ext=ts_ext_predictio
#fig.savefig('prediction_%im_VLISSGN_HWLW'%(timestep_min))
#Timeseries.write_tsnetcdf(ts=ts_prediction, ts_ext=ts_ext_prediction, station='VLI
```

Information For Developers

Create a python environment hatyan_env and install hatyan as developer:

- download Anaconda 64 bit Python 3.7 (or higher) from https://www.anaconda.com/distribution/#download-section (miniconda should also be sufficient, but this is not yet tested)
- install it with the recommended settings, but check 'add Anaconda3 to my PATH environment variable' if you want to use conda from the windows command prompt instead of anaconda prompt
- checkout the hatyan_python folder from the lib_tide repository in a local folder, e.g. `C:\DATA\hatyan_python`: https://repos.deltares.nl/repos/lib_tide/trunk/src/hatyan_python
- open command line and navigate to hatyan local folder, e.g. `C:\DATA\hatyan_python`
- `conda env create -f environment.yml` (This yml file installs Python 3.6.12 since that is the latest available Python on RHEL)
- `conda info --envs` (should show hatyan_env virtual environment in the list)
- conda activate hatyan_env
- `python -m pip install -e . -r requirements_dev.txt` (pip developer mode, also install all packages in requirements_dev.txt containing CentOS tested libraries, linked via setup.py)
- conda deactivate
- to remove hatyan_env when necessary: `conda remove -n hatyan_env --all`

Increase the hatyan version number:

- open command line and navigate to hatyan local folder, e.g. `C:\DATA\hatyan_python`
- conda activate hatyan_env
- bumpversion major or bumpversion minor or bumpversion patch
- the hatyan version number of all relevant files will be updated, as stated in setup.cfg

Running the testbank:

- open command line and navigate to hatyan local folder, e.g. `C:\DATA\hatyan_python`
- conda activate hatyan_env
- pytest
- `pytest -m acceptance`
- `pytest -m systemtest`
- `pytest -m slow`
- `pytest -m "not slow"` (exclude 'slow' testbank scripts for all stations)
- the following arguments are automatically provided via pytest.ini: `-v --tb=short`, add `--cov=hatyan` for a coverage summary

Generate html and pdf documentation:

- in order to generate pdf documentation, miktex needs to be installed and its packages should be updated from its console.
- open command line and navigate to hatyan local folder, e.g. `C:\DATA\hatyan_python`
- conda activate hatyan_env
- `python scripts/generate_documentation.py`

Generate RPM (RHEL/CentOS installer):

- preparation: activate environment, run testbank, check acceptance test output and make backup of results, generate html and pdf documentation, increase minor version number, update history.rst, commit changes
- use the script in scripts/hatyan_rpmbuild.sh (for instance on the CentOS7 Deltares buildserver)
- this script uses the rpmbuild command and the specfile to generate an RPM on a CentOS/RHEL machine with the correct dependencies installed
- rpmbuild uses the specfile scripts/hatyan_python-latest.spec as input
- the dependencies for the RPM are documented in the specfile
- the required Python libraries are documented in requirements_dev.txt: these are fixed, which is at least relevant for sip, since it needs to be compatible with pyqt5==5.7.1 for Qt5 plots
- additionally, the library pyqt5==5.7.1 (specfile) is for interative QT5 plots. There is a newer version but it requires glibc >2.14, while 2.12 is the highest version available on CentOS/RedHat 6)
- to test hatyan on CentOS without installing an RPM: use the script scripts/hatyan_rpmbuild_nobinaries.sh, this creates a comparable setup in the home directory and a ~/hatyan_fromhome.sh file comparable to hatyan command. If you get an error about X11-forwarding, first try the xterm command.

Sub-modules

- hatyan.analysis_prediction

- hatyan.astrog
- hatyan.components
- hatyan.foreman_core
- hatyan.hatyan_core
- hatyan.timeseries
- hatyan.wrapper_RWS

Module `hatyan.analysis_prediction`

analysis_prediction.py contains hatyan definitions related to tidal analysis and prediction.

hatyan is a Python program for tidal analysis and prediction, based on the FORTRAN version. Copyright (C) 2019-2020 Rijkswaterstaat. Maintained by Deltares, contact: Jelmer Veenstra (jelmer.veenstra@deltares.nl). Source code available at: https://repos.deltares.nl/repos/lib_tide/trunk/src/hatyan_python/hatyan

Functions

Function `vectoravg`

```
def vectoravg(
    A_all,
    phi_deg_all
)
```

calculates the vector average of A and phi per constituent, it vector averages over values resulting from multiple periods. A regular average is calculated for the amplitude of A0 (middenstand)

Parameters

`A_i_all` : TYPE DESCRIPTION.
`phi_i_deg_all` : TYPE DESCRIPTION.

Returns

`A_i_mean` : TYPE DESCRIPTION.
`phi_i_deg_mean` : TYPE DESCRIPTION.

Function get_components_from_ts

```
def get_components_from_ts(
    ts,
    const_list,
    nodalfactors=True,
    xfac=False,
    fu_alltimes=True,
    CS_comps=None,
    analysis_peryear=False,
    analysis_permonth=False,
    return_allyears=False,
    source='schureman'
)
```

Wrapper around the analysis() function, it optionally processes a timeseries per year and vector averages the results afterwards, passes the rest of the arguments on to analysis function The timezone of the timeseries, will also be reflected in the phases of the resulting component set, so the resulting component set can be used to make a prediction in the original timezone.

Parameters

`ts` : pandas.DataFrame The DataFrame should contain a 'values' column and a pd.DatetimeIndex as index, it contains the timeseries to be analysed, as obtained from e.g. readts_*.

`const_list` : list, pandas.Series or str list or pandas.Series: contains the tidal constituent names for which to analyse the provided timeseries ts. str: a predefined name of a component set for hatyan_core.get_const_list_hatyan()

`nodalfactors` : `bool/int`, optional Whether or not to apply nodal factors. The default is True.

`xfac` : `bool/int`, optional Whether or not to apply x-factors. The default is False.

`fu_alltimes` : `bool/int`, optional determines whether to calculate nodal factors in middle of analysis period (default) or on every timestep. The default is True.

`analysis_peryear` : `bool/int`, optional DESCRIPTION. The default is False.

`analysis_permonth` : `bool/int`, optional caution, it tries to analyse each month, but skips it if it fails. analysis_peryear argument has priority. The default is False.

`return_allyears` : `bool/int`, optional DESCRIPTION. The default is False.

`CS_comps` : pandas.DataFrame, optional contains the from/derive component lists for components splitting, as well as the amplitude factor and the increase in degrees. The default is None.

Raises

Exception DESCRIPTION.

Returns

`COMP_mean_pd` : pandas.DataFrame The DataFrame contains the component data with component names as index, and colums 'A' and 'phi_deg'.

COMP_all_pd : pandas.DataFrame, optional The same as COMP_mean_pd, but with all years added with MultiIndex

Function analysis

```
def analysis(
    ts,
    const_list,
    nodalfactors=True,
    xfac=False,
    fu_alltimes=True,
    CS_comps=None,
    return_prediction=False,
    source='schureman'
)
```

harmonic analysis with matrix transformations (least squares fit), optionally with component splitting for details about arguments and return variables, see get_components_from_ts() definition

return_prediction : bool/int, optional Whether to generate a prediction for the ts time array. The default is False.

Function split_components

```
def split_components(
    comp,
    CS_comps,
    dood_date_mid,
    xfac=False
)
```

component splitting function for details about arguments and return variables, see get_components_from_ts() definition

Function prediction

```
def prediction(
    comp,
    times_pred_all=None,
    times_ext=None,
    timestep_min=None,
    nodalfactors=True,
    xfac=False,
    fu_alltimes=True,
    source='schureman'
)
```

generates a tidal prediction from a set of components A and phi values. The component set has the same timezone as the timeseries used to create it, therefore the resulting prediction will also be in that original timezone.

Parameters

`comp` : pandas.DataFrame The DataFrame contains the component data with component names as index, and colums 'A' and 'phi_deg'.
`times_pred_all` : pandas.DatetimeIndex, optional Prediction timeseries. The default is None.
`times_ext` : list of datetime.datetime, optional Prediction time extents (list of start time and stop time). The default is None.
`timestep_min` : int, optional Prediction timestep in minutes. The default is None.
`nodalfactors` : `bool/int`, optional Whether or not to apply nodal factors. The default is True.
`xfac` : `bool/int`, optional Whether or not to apply x-factors. The default is False.
`fu_alltimes` : `bool/int`, optional determines whether to calculate nodal factors in middle of the prediction period (default) or on every timestep. The default is True.

Raises

Exception DESCRIPTION.

Returns

`ts_prediction_pd` : pandas.DataFrame The DataFrame should contain a 'values' column and a pd.DatetimeIndex as index, it contains the prediction times and values.

Module `hatyan.astrog`

astrog.py contains all astro-related definitions, previously embedded in a separate program but now part of hatyan.

hatyan is a Python program for tidal analysis and prediction, based on the FORTRAN version. Copyright (C) 2019-2020 Rijkswaterstaat. Maintained by Deltares, contact: Jelmer Veenstra (jelmer.veenstra@deltares.nl). Source code available at: https://repos.deltares.nl/repos/lib_tide/trunk/src/hatyan_python/hatyan

Functions

Function `astrog_culminations`

```
def astrog_culminations(
    tFirst,
    tLast,
    mode_dT='exact',
    tzone='UTC'
)
```

Makes use of the definitions dT, astrab and astrac. Calculates lunar culminations, parallax and declination. By default the lunar culmination is calculated at coordinates 52,0 (Netherlands,Greenwich).

Parameters

`tFirst` : pd.Timestamp, datetime.datetime or `string ("yyyymmdd")` Start of timeframe for output.

`tLast` : pd.Timestamp, datetime.datetime or `string ("yyyymmdd")` End of timeframe for output.

`mode_dT` : string, optional Method to calculate difference between universal time and terrestrial time (dT). Can be 'last' (for fortran reproduction), 'historical' or 'exact' (most accurate). The default is 'exact'.

`tzone` : `string/dt.timezone`, optional Timezone to convert the output dataset to. The default is 'UTC'.

Raises

Exception Checks input times tFirst and tLast.

Returns

`dataCulminations` : pandas DataFrame datetime: lunar culmination at Greenwich in UTC (datetime) type: type of culmination (1=lower, 2=upper) parallax: lunar parallax (degrees) declination: lunar declination (degrees)

Function `astrog_phases`

```
def astrog_phases(
    tFirst,
    tLast,
    mode_dT='exact',
    tzone='UTC'
)
```

Makes use of the definitions dT, astrab and astrac. Calculates lunar phases. The lunar phases are independent of coordinates.

Parameters

`tFirst` : datetime.datetime or `string ("yyyymmdd")` Start of timeframe for output.

`tLast` : datetime.datetime or `string ("yyyymmdd")` End of timeframe for output.

`mode_dT` : string, optional Method to calculate difference between universal time and terrestrial time (dT). Can be 'last' (for fortran reproduction), 'historical' or 'exact' (most accurate). The default is 'exact'.

`tzone` : `string/dt.timezone`, optional Timezone to convert the output dataset to. The default is 'UTC'.

Raises

Exception Checks input times tFirst and tLast.

Returns

`dataPhases` : pandas DataFrame datetime: lunar phase in UTC (datetime) type: type of phase (1=FQ, 2=FM, 3=LQ, 4=NM)

Function astrog_sunriseset

```
def astrog_sunriseset(
    tFirst,
    tLast,
    mode_dT='exact',
    tzone='UTC',
    lon=5.3876,
    lat=52.1562
)
```

Makes use of the definitions dT, astrab and astrac. Calculates sunrise and -set at requested location.

Parameters

`tFirst` : datetime.datetime or `string ("yyyymmdd")` Start of timeframe for output.

`tLast` : datetime.datetime or `string ("yyyymmdd")` End of timeframe for output.

`mode_dT` : string, optional Method to calculate difference between universal time and terrestrial time (dT). Can be 'last' (for fortran reproduction), 'historical' or 'exact' (most accurate). The default is 'exact'.

`tzone` : `string/dt.timezone`, optional Timezone to convert the output dataset to. The default is 'UTC'.

`lon` : float, optional Longitude, defined positive eastward. The default is -5.3876 (Amersfoort).

`lat` : float, optional Latitude, defined positive northward, cannot exceed 59 (too close to poles). The default is 52.1562 (Amersfoort).

Raises

Exception Checks input times tFirst and tLast.

Returns

`dataSun` : pandas DataFrame datetime: time of rise or set in UTC (datetime) type: type (1=sunrise, 2=sunset)

Function astrog_moonriseset

```
def astrog_moonriseset(
    tFirst,
    tLast,
    mode_dT='exact',
    tzone='UTC',
    lon=5.3876,
    lat=52.1562
)
```

Makes use of the definitions dT, astrab and astrac. Calculates moonrise and -set at requested location.

Parameters

tFirst : datetime.datetime or string ("yyyymmdd") Start of timeframe for output.
tLast : datetime.datetime or string ("yyyymmdd") End of timeframe for output.
mode_dT : string, optional Method to calculate difference between universal time and terrestrial time (dT). Can be 'last' (for fortran reproduction), 'historical' or 'exact' (most accurate). The default is 'exact'.
tzone : string/dt.timezone, optional Timezone to convert the output dataset to. The default is 'UTC'.
lon : float, optional Longitude, defined positive eastward. The default is -5.3876 (Amersfoort).
lat : float, optional Latitude, defined positive northward, cannot exceed 59 (too close to poles). The default is 52.1562 (Amersfoort).

Raises

Exception Checks input times tFirst and tLast.

Returns

dataMoon : pandas DataFrame datetime: time of rise or set in UTC (datetime) type: type (1=moonrise, 2=moonset)

Function astrog_anomalies

```
def astrog_anomalies(
    tFirst,
    tLast,
    mode_dT='exact',
    tzone='UTC'
)
```

Makes use of the definitions dT, astrab and astrac. Calculates lunar anomalies. The lunar anomalies are independent of coordinates.

Parameters

tFirst : datetime.datetime or string ("yyyymmdd") Start of timeframe for output.
tLast : datetime.datetime or string ("yyyymmdd") End of timeframe for output.
mode_dT : string, optional Method to calculate difference between universal time and terrestrial time (dT). Can be 'last' (for fortran reproduction), 'historical' or 'exact' (most accurate). The default is 'exact'.
tzone : string/dt.timezone, optional Timezone to convert the output dataset to. The default is 'UTC'.

Raises

Exception Checks input times tFirst and tLast.

Returns

dataAnomaly : pandas DataFrame datetime: lunar anomaly in UTC (datetime) type: type of anomaly (1=perigeum, 2=apogeum)

Function astrog_seasons

```
def astrog_seasons(
    tFirst,
    tLast,
    mode_dT='exact',
    tzone='UTC'
)
```

Makes use of the definitions dT, astrab and astrac. Calculates astronomical seasons. The seasons are independent of coordinates.

Parameters

tFirst : datetime.datetime or string ("yyyymmdd") Start of timeframe for output.
tLast : datetime.datetime or string ("yyyymmdd") End of timeframe for output.
mode_dT : string, optional Method to calculate difference between universal time and terrestrial time (dT). Can be 'last' (for fortran reproduction), 'historical' or 'exact' (most accurate). The default is 'exact'.
tzone : string/dt.timezone, optional Timezone to convert the output dataset to. The default is 'UTC'.

Raises

Exception Checks input times tFirst and tLast.

Returns

dataSeasons : pandas DataFrame datetime: start of astronomical season in UTC (datetime) type: type of astronomical season (1=spring, 2=summer, 3=autumn, 4=winter)

Function astrab

```
def astrab(
    date,
    dT_TT,
    lon=5.3876,
    lat=52.1562
)
```

Python version of astrab.f in FORTRAN 77 Calculates 18 astronomical parameters at requested time.

Parameters

`date` : datetime.datetime or pandas.DatetimeIndex Requested time for calculation.

`dT_TT` : float Difference between terrestrial and universal time in days.

`lon` : float, optional Longitude for altitudes, defined positive eastward. The default is 5.3876 (Amersfoort).

`lat` : float, optional Latitude for altitudes, defined positive northward. The default is 52.1562 (Amersfoort).

Raises

Exception Checks if input is valid.

Returns

`astrabOutput` : dictionary 18 astronomical variables: EHMOON, lunar ephemeris hour angle (degrees between +90 and +450) DECMOO, lunar declination (degrees) PARLAX, lunar horizontal parallax (arcseconds) DPAXDT, time derivative of parallax (arcseconds/day) ALTMOO, lunar altitude (degrees, negative below horizon) ELONG, ecliptic elongation moon-sun (degrees, between +45 and +405) ALTSUN, solar altitude (degrees, negative under horizon) LONSUN, solar longitude (degrees, between 45 and 405) EQELON, equatorial elongaton moon-sun (degrees, between 0 and 360) DECSUN, solar declination (degrees) DISSUN, relative distance earth-sun (astronomical units) EHARI, ephemeris hour angle vernal equinox (degrees, between 0 and 360) RASUN, solar right ascension (degrees, between 0 and 360) EHSUN, solar ephemeris hour angle (degrees, between 0 and 360) LONMOO, lunar longitude (degrees, between 0 and 360) LATMOO, lunar latitude (degrees) RAMOON, lunar right ascension (degrees, between 0 and 360) ANM, mean lunar anomaly (degrees, between 0 and 360)

Function `astrac`

```
def astrac(
    timeEst,
    dT_TT,
    mode,
    lon=5.3876,
    lat=52.1562
)
```

Python version of astrac.f in FORTRAN 77. Calculates exact time of requested astronomical phenomenon.

Parameters

`timeEst` : datetime.datetime or pandas.DatetimeIndex Estimated time for iteration.

`dT_TT` : float Difference between terrestrial and universal time in days.

`mode` **: numpy.array of integer(s)** Requested phenomenon: 1: lunar lower culmination (EHMOON=180 deg.) 2: lunar upper culmination (EHMOON=360 deg.) 3: lunar first quarter (ELONG=90 deg.) 4: full moon (ELONG=180 deg.) 5: lunar last quarter (ELONG=270 deg.) 6: new moon (ELONG=360 deg.) 7: moonrise (ALTMOO=-34 BOOGMIN-SEMIDIAM., ascending) 8: moonset (ALTMOO=-34 BOOGMIN-SEMIDIAM., descending) 9: sunrise (ALTSUN=-50 arcseconds, ascending) 10: sunset (ALTSUN=-50 arcseconds, descending) 11: vernal equinox (LONSUN=360 deg.) 12: summer solstice (LONSUN=90 deg.) 13: autumnal equinox (LONSUN=180 deg.) 14: winter solstice (LONSUN=270 deg.) 15: perigeum (DPAXDT=0, descending) 16: apogeum (DPAXDT=0, ascending)

`lon` **: float, optional** Longitude for rise and set, defined positive eastward. The default is 5.3876 (Amersfoort).

`lat` **: float, optional** Latitude for rise and set, defined positive northward, cannot exceed 59 for modes 7 to 10 (too close to poles). The default is 52.1562 (Amersfoort).

Raises

Exception Checks if latitude is not too close to poles.

Returns

`TIMOUT` **: datetime** Exact time after iteration.

Function `dT`

```
def dT(
    dateIn,
    mode_dT='exact'
)
```

Calculates difference between terrestrial time and universal time. Current hard-coded values valid until 2023, update arrays afterwards.

Parameters

`dateIn` **: datetime.datetime or pandas.DatetimeIndex** Date for correction. Definition makes use of provided year.

`mode` **: string, optional** 'last': using the last hard-coded value (as last FORTRAN version) 'historical': using all (previous) hard-coded values (historical FORTRAN versions) 'exact' (default): determine dT based on number of leap seconds (follows international definition)

Raises

Warning Checks if hard-coded values can still be used.

Returns

`dT_TT` **: float** Difference dT between terrestrial time (TT) and universal time (UT1) in seconds

Function convert_str2datetime

```
def convert_str2datetime(
    datetime_in_list
)
```

Tries to convert datetime_in_list (list of str or datetime.datetime) to list of datetime.datetime

Parameters

datetime_in_list : list of str/dt.datetime/pd.Timestamp DESCRIPTION.

Raises

Exception DESCRIPTION.

Returns

datetime_out_list : list of pd.Timestamp DESCRIPTION.

Function convert2perday

```
def convert2perday(
    dataframeIn,
    timeformat='%H:%M %Z'
)
```

converts normal astrog pd.DataFrame to one with the same information restructured per day

Parameters

dataframeIn : pd.DataFrame with columns 'datetime' and 'type_str'.
timeformat : str, optional format of the timestrings in dataframeOut. The default is '%H:%M %Z'.

Returns

dataframeOut : pd.DataFrame The 'datetime' column contains dates, with columns containing all unique 'type_str' values.

Function plot_astrog_diff

```
def plot_astrog_diff(
    pd_python,
    pd_fortran,
    typeUnit='-',
    typeLab=None,
    typeBand=None,
    timeBand=None
)
```

Plots results of FORTRAN and python verison of astrog for visual inspection. Top plot shows values or type, middle plot shows time difference, bottom plot shows value/type difference.

Parameters

`pd_python` : pandas DataFrame DataFrame from astrog (python) with times (UTC).
`pd_fortran` : pandas DataFrame DataFrame from astrog (FORTRAN) with times (UTC).
`typeUnit` : string, optional Unit of provided values/types. The default is '-'.
`typeLab` : TYPE, optional Labels of provided types. The default is ['rise','set'].
`typeBand` : list of floats, optional Expected bandwith of accuracy of values/types. The default is [-.5,.5].
`timeBand` : list of floats, optional Expected bandwith of accuracy of times (seconds). The default is [0,60].
`timeLim` : list of floats, optional Time limits of x-axis. The default is None (takes limits from pd_python).

Returns

`fig` : figure handle Output figure.
`axs` : axis handles Axes in figure.

Module hatyan.components

components.py contains all the definitions related to hatyan components.

hatyan is a Python program for tidal analysis and prediction, based on the FORTRAN version. Copyright (C) 2019-2020 Rijkswaterstaat. Maintained by Deltares, contact: Jelmer Veenstra (jelmer.veenstra@deltares.nl). Source code available at: https://repos.deltares.nl/repos/lib_tide/trunk/src/hatyan_python/hatyan

Functions

Function plot_components

```
def plot_components(
    comp,
    comp_allyears=None,
    comp_validation=None,
    sort_freqs=True
)
```

Create a plot with the provided analysis results

Parameters

comp : TYPE DESCRIPTION.
comp_allyears : TYPE, optional DESCRIPTION. The default is None.
comp_validation : TYPE, optional DESCRIPTION. The default is None.
sort_freqs : BOOL, optional Whether to sort the component list on frequency or not, without sorting it is possible to plot components that are not available in hatyan. The default is True.

Returns

fig : matplotlib.figure.Figure The generated figure handle, with which the figure can be adapted and saved.
axs : (tuple of) matplotlib.axes._subplots.AxesSubplot The generated axis handle, whith which the figure can be adapted.

Function `write_components`

```
def write_components(
    comp,
    filename,
    metadata=None
)
```

Writes the provided analysis results to a file

Parameters

comp : TYPE DESCRIPTION.
filename : TYPE DESCRIPTION.
metadata : TYPE, optional DESCRIPTION. The default is None.

Returns None.

Function `merge_componentgroups`

```
def merge_componentgroups(
    comp_main,
    comp_sec,
    comp_sec_list=['SA', 'SM']
)
```

Merges the provided component groups into one

Parameters

comp_main : TYPE DESCRIPTION.
comp_sec : TYPE DESCRIPTION.
comp_sec_list : TYPE, optional DESCRIPTION. The default is ['SA','SM'].

Raises

Exception DESCRIPTION.

Returns

COMP_merged : TYPE DESCRIPTION.

Function **read_components**

```
def read_components(
    filename,
    get_metadata=False
)
```

Reads analysis results from a file.

Parameters

filename : TYPE DESCRIPTION.
get_metadata : TYPE, optional DESCRIPTION. The default is False.

Raises

Exception DESCRIPTION.

Returns

TYPE DESCRIPTION.

Function **components_timeshift**

```
def components_timeshift(
    comp,
    hours
)
```

Module `hatyan.foreman_core`

foreman.py contains all foreman definitions now embedded in hatyan. The dataset is derived from "M.G.G. Foreman (2004), Manual for Tidal Heights Analysis and Prediction, Institute of Ocean Sciences (Patricia Bay, Sidney B.C. Canada)"

hatyan is a Python program for tidal analysis and prediction, based on the FORTRAN version. Copyright (C) 2019-2020 Rijkswaterstaat. Maintained by Deltares, contact: Jelmer Veenstra (jelmer.veenstra@deltares.nl). Source code available at: https://repos.deltares.nl/repos/lib_tide/trunk/src/hatyan_python/hatyan

Functions

Function `get_foreman_content`

```
def get_foreman_content()
```

Function `get_foreman_doodson_nodal_harmonic`

```
def get_foreman_doodson_nodal_harmonic(
    lat_deg=51.45
)
```

Omzetten van het tweede deel van de foremantabel in een pandas DataFrame met harmonische (+satellite) componenten.

Parameters

`const_list` : TYPE DESCRIPTION.

`lat_deg` : TYPE, optional 0 in degrees from equator (pos N, neg S). For R1 and R2, maar variatie heeft niet erg veel invloed op A en phi. The default is 51.45.

Returns

foreman_harmonic_doodson_all : TYPE DESCRIPTION.
foreman_harmonic_nodal_all : TYPE DESCRIPTION.
foreman_harmonic_doodson : TYPE DESCRIPTION.
foreman_harmonic_nodal : TYPE DESCRIPTION.

Function get_foreman_shallowrelations

```
def get_foreman_shallowrelations()
```

Omzetten van het derde deel van de foremantabel in een pandas DataFrame met shallow water relations.

Returns

foreman_shallowrelations : TYPE DESCRIPTION.

Function get_foreman_v0freq_fromfromharmonicdood

```
def get_foreman_v0freq_fromfromharmonicdood(
    dood_date=None,
    mode=None
)
```

Zoekt de frequentie of v0 voor alle harmonische componenten, in geval van v0 op de gegeven datum (dood_date). Hiervoor zijn de harmonische doodson getallen (foreman_harmonic_doodson_all) nodig, afkomstig uit get_foreman_harmonic uit foreman.py

Function get_foreman_v0_freq

```
def get_foreman_v0_freq(
    const_list,
    dood_date
)
```

Zoekt voor iedere component uit de lijst de v op basis van harmonische doodson getallen en de frequentie rechtstreeks uit de foreman tabel. Shallow water componenten worden afgeleid met de relaties beschreven in de foreman tabel.

Function get_foreman_nodalfactors_fromharmonic_oneconst

```
def get_foreman_nodalfactors_fromharmonic_oneconst(
    foreman_harmonic_nodal_const,
    dood_date
)
```

Function get_foreman_nodalfactors

```
def get_foreman_nodalfactors(
```

```
        const_list,
        dood_date
    )
```

Zoekt voor iedere component uit de lijst de u en f (nodal factors) op basis van satellite doodson getallen uit de foreman tabel. Shallow water componenten worden afgeleid met de relaties beschreven in de foreman tabel.

Module `hatyan.hatyan_core`

hatyan_core.py contains definitions with data for the hatyan constituents.

hatyan is a Python program for tidal analysis and prediction, based on the FORTRAN version. Copyright (C) 2019-2020 Rijkswaterstaat. Maintained by Deltares, contact: Jelmer Veenstra (jelmer.veenstra@deltares.nl). Source code available at: https://repos.deltares.nl/repos/lib_tide/trunk/src/hatyan_python/hatyan

Functions

Function `get_v0uf_sel`

```
def get_v0uf_sel(
    const_list
)
```

get_v0uf_sel

Parameters

`const_list` : **TYPE** DESCRIPTION.

Returns

`v0uf_sel` : **TYPE** DESCRIPTION.

Function `get_const_list_hatyan`

```
def get_const_list_hatyan(
    listtype,
    return_listoptions=False
```

```
)
```

Definition of several hatyan components lists, taken from the tidegui initializetide.m code, often originating from corresponcence with Koos Doekes

Parameters

`listtype` : str The type of the components list to be retrieved, options: - 'all': all available components in hatyan_python - 'all_originalorder': all 195 hatyan components in original hatyan-FORTRAN order - 'year': default list of 94 hatyan components - 'halfyear': list of 88 components to be used when analyzing approximately half a year of data - 'month': list of 21 components to be used when analyzing one month of data. If desired, the K1 component can be splitted in P1/K1, N2 in N2/Nu2, S2 in T2/S2/K2 and 2MN2 in Labda2/2MN2. - 'month_deepwater': list of 21 components to be used when analyzing one month of data for deep water (from tidegui). - 'springneap': list of 14 components to be used when analyzing one spring neap period (approximately 15 days) of data - 'day': list of 10 components to be used when analyzing one day - 'day_tidegui': list of 5 components to be used when analyzing one day ir two tidal cycles (from tidegui) - 'tidalcycle': list of 6 components to be used when analyzing one tidal cycle (approximately 12 hours and 25 minutes)

Raises

Exception DESCRIPTION.

Returns

`const_list_hatyan` : list of str A list of component names.

Function `get_doodson_eqvals`

```
def get_doodson_eqvals(
    dood_date,
    mode=None
)
```

Berekent de doodson waardes T, S, H, P, N en P1 voor het opgegeven tijdstip.

Parameters

`dood_date` : datetime.datetime or pandas.DateTimeIndex Date(s) on which the doodson values should be calculated.

`mode` : str, optional Calculate doodson values with Schureman (hatyan default) or Sea Level Science by Pugh. The default is False.

Returns

`dood_T_rad` : TYPE Bij hatyan 180+, maar in docs niet. In hatyan fortran code is +/-90 in v ook vaak omgedraaid in vergelijking met documentation.

`dood_S_rad` : TYPE Middelbare lengte van de maan. Mean longitude of moon

`dood_H_rad` : TYPE Middelbare lengte van de zon. mean longitude of sun

dood_P_rad : TYPE Middelbare lengte van het perieum van de maan. Longitude of lunar perigee
dood_N_rad : TYPE Afstand van de klimmende knoop van de maan tot het lentepunt. Longitude of moons node
dood_P1_rad : TYPE Middelbare lengte van het perieum van de zon. Longitude of solar perigee

Function get_hatyan_constants

```
def get_hatyan_constants(
    dood_date
)
```

get_hatyan_constants

Parameters

dood_date : TYPE DESCRIPTION.

Returns

DOMEGA : TYPE DESCRIPTION.
DIKL : TYPE DESCRIPTION.
DC1681 : TYPE DESCRIPTION.
DC5023 : TYPE DESCRIPTION.
DC0365 : TYPE DESCRIPTION.

Function calcwrite_baseforv0uf

```
def calcwrite_baseforv0uf()
```

Calculate and write table for all constituents. Alternatively (faster), this data can be read from data_components_hatyan.pkl. Whether to calculate or read the table is controlled with the v0uf_calculatewrite boolean in the top of this script

Returns

v0uf_all : TYPE DESCRIPTION.

Function get_hatyan_freqs

```
def get_hatyan_freqs(
    const_list,
    dood_date=None,
    sort_onfreq=True,
    return_allraw=False
)
```

Returns the frequencies of the requested list of constituents. Source: beromg.f

Parameters

const_list : list or pandas.Series contains the tidal constituent names.

Raises

Exception DESCRIPTION.

Returns

t_const_freq : TYPE DESCRIPTION.

Function get_hatyan_v0

```
def get_hatyan_v0(
    const_list,
    dood_date
)
```

Returns the v-values of the requested list of constituents for the requested date(s)

Parameters

const_list : list or pandas.Series contains the tidal constituent names.
dood_date : TYPE DESCRIPTION.

Returns

v_0i_rad : TYPE DESCRIPTION.

Function get_hatyan_u

```
def get_hatyan_u(
    const_list,
    dood_date
)
```

Returns the u-values of the requested list of constituents for the requested date(s)

Parameters

const_list : list or pandas.Series contains the tidal constituent names.
dood_date : TYPE DESCRIPTION.

Returns

u_i_rad_HAT : TYPE DESCRIPTION.

Function get_hatyan_f

```
def get_hatyan_f(
    const_list,
    dood_date,
    xfac
)
```

Returns the f-values of the requested list of constituents for the requested date(s)

Parameters

const_list : list or pandas.Series contains the tidal constituent names.
dood_date : TYPE DESCRIPTION.

Returns

f_i_HAT : TYPE DESCRIPTION.

Function correct_fwith_xfac

```
def correct_fwith_xfac(
    f_i_pd,
    f_i_M2_pd,
    xfac
)
```

Correct f-values with xfactor, this definition is only ran when xfac=True.

Parameters

f_i_pd : TYPE DESCRIPTION.
f_i_M2_pd : TYPE DESCRIPTION.

Returns

f_i_pd : TYPE DESCRIPTION.

Function robust_daterange_fromtimesextfreq

```
def robust_daterange_fromtimesextfreq(
    times_ext,
    timestep_min
)
```

Generate daterange. Pandas pd.date_range and pd.DatetimeIndex only support times between 1677-09-21 and 2262-04-11, because of its ns accuracy. For dates outside this period, a list is generated and converted to a pd.Index instead.

Parameters

times_ext : list of datetime.datetime DESCRIPTION.
timestep_min : int DESCRIPTION.

Returns

times_pred_all_pdDTI : pd.DatetimeIndex or pd.Index DESCRIPTION.

Function robust_timedelta_sec

```
def robust_timedelta_sec(
    dood_date,
    refdate_dt=None
)
```

Generate timedelta. Pandas pd.DatetimeIndex subtraction only supports times between 1677-09-21 and 2262-04-11, because of its ns accuracy. For dates outside this period, a list subtraction is generated.

Parameters

dood_date : pd.Index or pd.DatetimeIndex DESCRIPTION.
refdate_dt : dt.datetime, optional DESCRIPTION. The default is None.

Returns

dood_tstart_sec : np.array DESCRIPTION.
fancy_pddt : bool DESCRIPTION.

Module `hatyan.timeseries`

timeseries.py contains all definitions related to hatyan timeseries.

hatyan is a Python program for tidal analysis and prediction, based on the FORTRAN version. Copyright (C) 2019-2020 Rijkswaterstaat. Maintained by Deltares, contact: Jelmer Veenstra (jelmer.veenstra@deltares.nl). Source code available at: https://repos.deltares.nl/repos/lib_tide/trunk/src/hatyan_python/hatyan

Functions

Function `calc_HWLW`

```
def calc_HWLW(
    ts,
    calc_HWLW345=False,
    calc_HWLW345_cleanup1122=True,
    debug=False
)
```

Calculates extremes (high and low waters) for the provided timeseries. This definition uses scipy.signal.find_peaks() with arguments 'distance' and 'prominence'. The minimal 'distance' between two high or low water peaks is based on the M2 period: 12.42/1.5=8.28 hours for HW and 12.42/1.7=7.30 hours for LW (larger because of aggers). The prominence for local extremes is set to 0.01m, to filter out very minor dips in the timeseries. If there are two equal high or low water values, the first one is taken. There are no main high/low waters calculated within 6 hours of the start/end of the timeseries (keyword buffer_hr), since these can be invalid. This function can deal with gaps. Since scipy.signal.find_peaks() warns about nan values, those are removed first.

Parameters

ts : pandas.DataFrame The DataFrame should contain a ‘values’ column and a pd.DatetimeIndex as index, it contains the timeseries with a tidal prediction or water level measurements.

calc_HWLW345 : boolean, optional Whether to also calculate local extremes, first/second low waters and ‘aggers’. The default is False, in which case only extremes per tidal period are calculated. When first/second low waters and aggers are calculated, the local extremes around highwater (eg double highwaters and dips) are filtered out first.

calc_HWLW345_cleanup1122 : boolean, optional Whether to remove HWLWcodes 11 and 22 from DataFrame. The default is True.

debug : boolean, optional Whether to print debug information. The default is False.

Raises

Exception DESCRIPTION.

Returns

data_pd_HWLW : pandas.DataFrame The DataFrame contains colums ‘times’, ‘values’ and ‘HWLWcode’, it contains the times, values and codes of the timeseries that are extremes. 1 (high water) and 2 (low water). And if calc_HWLW345=True also 3 (first low water), 4 (agger) and 5 (second low water).

Function `calc_HWLWnumbering`

```
def calc_HWLWnumbering(
    ts_ext,
    station=None,
    corr_tideperiods=None
)
```

For calculation of the extremes numbering, w.r.t. the first high water at Cadzand in 2000 (occurred on 1-1-2000 at approximately 9:45). The number of every high and low water is calculated by taking the time difference between itself and the first high water at Cadzand, correcting it with the station phase difference (M2phasediff). Low waters are searched for half an M2 period from the high waters. By adding a search window of half the period of M2 (searchwindow_hr), even strong time variance between consecutive high or low waters should be caputered.

Parameters

ts_ext : pandas.DataFrame The DataFrame should contain a ‘values’ and ‘HWLW-code’ column and a pd.DatetimeIndex as index, it contains the times, values and codes of the timeseries that are extremes.

station : string, optional The station for which the M2 phase difference should be retrieved from data_M2phasediff_perstation.txt. This value is the phase difference in degrees of the occurrence of the high water generated by the same tidal wave as the first high water in 2000 at Cadzand (actually difference between M2 phases of stations). This value is used to correct the search window of high/low water numbering. The default is None.

corr_tideperiods : integer, optional Test keyword to derive HWLWnumbering with a n*360 degrees offset only, but this does not work properly. The default is None.

Raises

Exception DESCRIPTION.

Returns

ts_ext : pandas.DataFrame The input DataFrame with the column 'HWLWno' added, which contains the numbers of the extremes.

Function timeseries_fft

```
def timeseries_fft(
    ts_residue,
    prominence=1000,
    plot_fft=True
)
```

Function plot_timeseries

```
def plot_timeseries(
    ts,
    ts_validation=None,
    ts_ext=None,
    ts_ext_validation=None
)
```

Creates a plot with the provided timeseries

Parameters

ts : pandas.DataFrame The DataFrame should contain a 'values' column and a pd.DatetimeIndex as index, it contains the timeseries.

ts_validation : pandas.DataFrame, optional The DataFrame should contain a 'values' column and a pd.DatetimeIndex as index, it contains the timeseries. The default is None.

ts_ext : pandas.DataFrame, optional The DataFrame should contain a 'values' and 'HWLW_code' column and a pd.DatetimeIndex as index, it contains the times, values and codes of the timeseries that are extremes. The default is None.

ts_ext_validation : pandas.DataFrame, optional The DataFrame should contain a 'values' and 'HWLW_code' column and a pd.DatetimeIndex as index, it contains the times, values and codes of the timeseries that are extremes. The default is None.

Returns

fig : matplotlib.figure.Figure The generated figure handle, with which the figure can be adapted and saved.

axs : (tuple of) matplotlib.axes._subplots.AxesSubplot The generated axis handle, whith which the figure can be adapted.

Function plot_HWLW_validatestats

```
def plot_HWLW_validatestats(
    ts_ext,
    ts_ext_validation,
    create_plot=True
)
```

This definition calculates (and plots and prints) some statistics when comparing extreme values. This is done by calculating the extreme number (sort of relative to Cadzand 1jan2000, but see 'warning') and subtracting the ts_ext and ts_ext_validation dataframes based on these numbers (and HWLWcode). It will only result in values for the overlapping extremes, other values will be NaN and are not considered for the statistics. Warning: the calculated extreme numbers in this definition are not corrected for the real phase difference with the M2phasediff argument, the calculated extreme are fine for internal use (to match corresponding extremes) but the absolute number might be incorrect.

Parameters

ts_ext : pandas.DataFrame The DataFrame should contain a 'values' and 'HWLW_code' column and a pd.DatetimeIndex as index, it contains the times, values and codes of the timeseries that are extremes.

ts_ext_validation : pandas.DataFrame The DataFrame should contain a 'values' and 'HWLW_code' column and a pd.DatetimeIndex as index, values and codes of the timeseries that are extremes.

create_plot : boolean, optional Whether to plot the time/value differences or only print the statistics. The default is True.

Returns

fig : matplotlib.figure.Figure The generated figure handle, with which the figure can be adapted and saved.

axs : (tuple of) matplotlib.axes._subplots.AxesSubplot The generated axis handle, whith which the figure can be adapted.

Function write_tsnetcdf

```
def write_tsnetcdf(
    ts,
    station,
    vertref,
    filename,
    ts_ext=None,
    tzone_hr=1
)
```

Writes the timeseries to a netCDF file

Parameters

ts : pandas.DataFrame The DataFrame should contain a 'values' column and a pd.DatetimeIndex as index, it contains the timeseries.
station : str DESCRIPTION.
vertref : str DESCRIPTION.
filename : str The filename of the netCDF file that will be written.
ts_ext : pandas.DataFrame, optional The DataFrame should contain a 'values' and 'HWLW_code' column and a pd.DatetimeIndex as index, it contains the times, values and codes of the timeseries that are extremes. The default is None.
tzone_hr : int, optional The timezone (GMT+tzone_hr) that applies to the data. The default is 1 (MET).

Returns None.

Function `write_tsdia`

```
def write_tsdia(
    ts,
    station,
    vertref,
    filename
)
```

Writes the timeseries to an equidistant dia file

Parameters

ts : pandas.DataFrame The DataFrame should contain a 'values' column and a pd.DatetimeIndex as index, it contains the timeseries.
station : TYPE DESCRIPTION.
vertref : TYPE DESCRIPTION.
filename : TYPE DESCRIPTION.

Raises

Exception DESCRIPTION.

Returns None.

Function `write_tsdia_HWLW`

```
def write_tsdia_HWLW(
    ts_ext,
    station,
    vertref,
    filename
)
```

writes the extremes timeseries to a non-equidistant dia file

Parameters

ts_ext : pandas.DataFrame The DataFrame should contain a 'values' and 'HWLW_code' column and a pd.DatetimeIndex as index, it contains the times, values and codes of the timeseries that are extremes.
station : TYPE DESCRIPTION.
vertref : TYPE DESCRIPTION.
filename : TYPE DESCRIPTION.

Raises

Exception DESCRIPTION.

Returns None.

Function writets_noos

```
def writets_noos(
    ts,
    filename,
    metadata=None
)
```

Writes the timeseries to a noos file

Parameters

ts : pandas.DataFrame The DataFrame should contain a 'values' column and a pd.DatetimeIndex as index, it contains the timeseries.
filename : TYPE DESCRIPTION.

Returns None.

Function crop_timeseries

```
def crop_timeseries(
    ts,
    times_ext,
    onlyfull=True
)
```

Crops the provided timeseries

Parameters

ts : pandas.DataFrame The DataFrame should contain a 'values' column and a pd.DatetimeIndex as index, it contains the timeseries.
times_ext : TYPE DESCRIPTION.

Raises

Exception DESCRIPTION.

Returns

`ts_pd_out` : TYPE DESCRIPTION.

Function `resample_timeseries`

```
def resample_timeseries(
    ts,
    timestep_min,
    tstart=None,
    tstop=None
)
```

resamples the provided timeseries, only overlapping timesteps are selected, so no interpolation. with tstart/tstop it is possible to extend the timeseries with NaN values.

Parameters

`ts` : pandas.DataFrame The DataFrame should contain a 'values' and 'HWLW_code' column and a pd.DatetimeIndex as index, it contains the timeseries to be resampled.
`timestep_min` : int the amount of minutes with which to resample the timeseries.
`tstart` : dt.datetime, optional the start date for the resampled timeseries, the default is None which results in using the start date of the input ts.
`tstop` : dt.datetime, optional the stop date for the resampled timeseries, the default is None which results in using the stop date of the input ts.

Returns

`data_pd_resample : pandas.DataFrame with a 'values' column and a pd.DatetimeIndex as`
the resampled timeseries.

Function `check_ts`

```
def check_ts(
    ts
)
```

prints several statistics of the provided timeseries

Parameters

`ts` : pandas.DataFrame The DataFrame should contain a 'values' column and a pd.DatetimeIndex as index, it contains the timeseries to be checked.

Returns

`print_statement` : str For printing as a substring of another string.

Function **get_diablocks_startstopstation**

```
def get_diablocks_startstopstation(
    filename
)
```

Gets information about the data blocks present in a dia file

Parameters

filename : TYPE DESCRIPTION.
station : TYPE DESCRIPTION.

Raises

Exception DESCRIPTION.

Returns

block_starts : TYPE DESCRIPTION.
data_starts : TYPE DESCRIPTION.
data_ends : TYPE DESCRIPTION.
block_stations : TYPE DESCRIPTION.
block_id : TYPE DESCRIPTION.

Function **get_diablocks**

```
def get_diablocks(
    filename
)
```

Function **convertcoordinates**

```
def convertcoordinates(
    coordx_in,
    coordy_in,
    epsg_in,
    epsg_out=28992
)
```

Function **readts_dia_nonequidistant**

```
def readts_dia_nonequidistant(
    filename,
    diablocks_pd,
    block_id
)
```

Function readts_dia_equidistant

```
def readts_dia_equidistant(
    filename,
    diablocks_pd_extra,
    block_id
)
```

Function readts_dia

```
def readts_dia(
    filename,
    station=None,
    block_ids=None
)
```

Reads an equidistant or non-equidistant dia file, or a list of dia files. Also works for diafiles containing multiple blocks for one station.

Parameters

filename : TYPE DESCRIPTION.
station : TYPE DESCRIPTION. The default is None.
block_ids : int, list of int or 'allstation', optional DESCRIPTION. The default is None.

Raises

Exception DESCRIPTION.

Returns

data_pd : pandas.core.frame.DataFrame DataFrame with a 'values' column and a pd.DatetimeIndex as index in case of an equidistant file, or more columns in case of a non-equidistant file.

Function readts_dia_HWLW

```
def readts_dia_HWLW(
    filename,
    station
)
```

Reads a non-equidistant dia file (wrapper around readts_dia). This definition will be phased out.

Function readts_noos

```
def readts_noos(
    filename,
    datetime_format='%Y%m%d%H%M',
```

```
        na_values=None
    )
```

Reads a noos file

Parameters

`filename` : TYPE DESCRIPTION.
`datetime_format` : TYPE, optional DESCRIPTION. The default is '%Y%m%d%H%M'.
`na_values` : TYPE, optional DESCRIPTION. The default is None.

Returns

`data_pd` : TYPE DESCRIPTION.

Module `hatyan.wrapper_RWS`

wrapper_RWS.py contains wrapper functions around the hatyan process for RWS related calculations.

hatyan is a Python program for tidal analysis and prediction, based on the FORTRAN version. Copyright (C) 2019-2020 Rijkswaterstaat. Maintained by Deltares, contact: Jelmer Veenstra (jelmer.veenstra@deltares.nl). Source code available at: https://repos.deltares.nl/repos/lib_tide/trunk/src/hatyan_python/hatyan

Functions

Function `init_RWS`

```
def init_RWS(
    file_config,
    argvlist=[None],
    interactive_plots=True,
    silent=False
)
```

Initializes the hatyan process for RWS related calculations. Besides the return variables, it prints a header for the print output (shows up in the hatyan diagnostics file)

Parameters

`file_config` : TYPE DESCRIPTION.
`argvlist` : TYPE DESCRIPTION.
`interactive_plots` : `bool/int`, optional sets the correct matplotlib backend so plots are (not) displayed on both RedHat and windows. The default is True.

Raises

Exception DESCRIPTION.

Returns

dir_output : path/str the output directory for the hatyan process, the current directory is set to this folder.

timer_start : datetime.datetime provides a start time with which exit_RWS calculates the total time of the process.

Function exit_RWS

```
def exit_RWS(
    timer_start
)
```

Provides a footer to the print output (shows up in the hatyan diagnostics file)

Parameters

timer_start : TYPE The start time of the hatyan process, which is used to calculate the total time of the process.

Returns None.

Function get_outputfoldername

```
def get_outputfoldername(
    file_config
)
```

Creates an output folder based on the start time of the filename of the configfile and the current time.

Parameters

file_config : str or path path to the configuration file.

Raises

Exception DESCRIPTION.

Returns

dir_output : str or path path to the output directory.

Generated by *pdoc* 0.9.2 (https://pdoc3.github.io).