SELENE: *SEa LEvel NEar-real time quality control processing*

DESIGN & USER'S GUIDE

Version 1.0

Powered by: ***Puertos del Estado***

Puertos del Estado

GOBIERNO DE ESPAÑA MINISTERIO DE FOMENTO

Content

1. Introduction

This document contains a description of the software SELENE, intended to update the software operational at Puertos del Estado (PdE) for near-real time quality control and processing of tide gauge data. The update has mainly consisted in the implementation in Python programming language from the original Fortran codes developed at the end of the 90's.

It also aims to be a guide for developers to install, use and even customize the new software.

Currently, SELENE is already running successfully for the 40 tide gauges of the REDMAR network in Puertos del Estado (Spain).

1.1. Objectives and advantages of SELENE

The original software was created two decades ago based on FORTRAN programming language and C-Shell scripts, that allowed near-real time (each 15 min) quality control and processing of tide gauge data before entering the Nivmar Sea Level Forecasting system of PdE (Álvarez-Fanjul et al, 2001, Pérez-Gómez et al, 2013). The characteristics of this software became later adopted as part of the international standards on quality control of tide gauge data, described in several publications and reports such as the ones of the ESEAS-RI and the MyOcean projects (García et al, 2005, Pérez-Gómez and de Alfonso et al., 2010, Pouliquen, 2011).

Technology and programming languages have progressed significantly during the last years, along with an increasing use of tide gauge data in near-real time. The interest of the international tide gauge community (e.g. GLOSS, EuroGOOS) in having access to this software for other tide gauge networks yielded PdE to this upgrade to a modern, open-source and well-documented code.

The original software was completely embedded in Puertos del Estado operative and it was therefore difficult to adapt to other systems. One of the objectives was, moreover, to improve the ease of maintenance of the code and to facilitate its future upgrades by other institutions and experts within the GLOSS community.

The main objective of SELENE consists therefore in being a clear, simple, independent and portable software, keeping a good performance and easy maintenance.

2. Description of the original software

The algorithms described below perform the near-real time quality control before archiving and use of the data by means of an automatic procedure applied to a moving window of several days of raw data, with a latency varying from 15 min to 1 hour, depending on the implementation or needs. At Puertos del Estado, the software is applied each 15 min to all the REDMAR network stations since 1998 and, since 2011, to IBIROOS sea level stations (tide gauges in the European Atlantic coast, integrated in the Copernicus Marine Service IBI In-Situ TAC).

It is very important to emphasize that this software applies just the **L1 quality control level,** that according to the **standards defined within ESEAS-RI, MyOcean, GLOSS and EuroGOOS documents** mentioned above, is the one possible in a near-real time process. An L2 (Delayed Mode) quality control must be applied at least annually, including control of datum shifts and extremes, tides and mean sea levels evolution for the historical record.

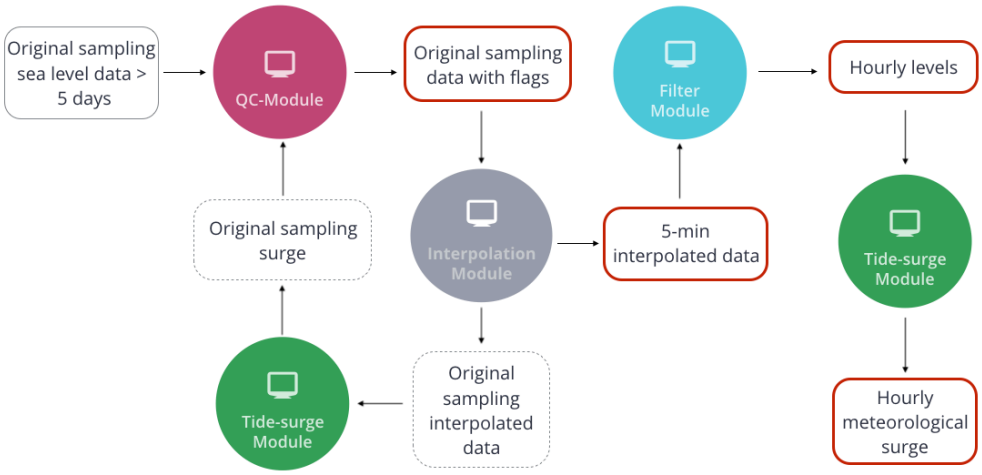

Figure 1: scheme with main modules (QC, Interpolation, Tide-Surge and Filter) and data flow of the software. Final output products framed in red: original sampling data with flags, 5-min interpolated data, hourly sea levels and residuals (Pérez-Gómez et al, 2013).

The algorithms perform the following basic actions: spikes detection, stability test, date and time control, flagging and interpolation of short gaps, data sampling adjustment and filter to hourly values. Also, if harmonic constants for a tide gauge are available, computation of the tide and residuals for the same period(subtracting the tide from the raw data) allows to apply the spike detection algorithm to the residuals, to detect less

obvious outliers, and the generation of the hourly residuals in near-real time. Applied to a moving window of around 5 days of raw data, the following products (files) are generated in the process: original sampling data with flags, 5-min interpolated data, hourly sea levels and hourly residuals (see figure 1 and table 1).

QC MODULE (flagging data)	• Format and date control • Out of range values • Algorithm for detection of spikes • Stability test: flagging values when there is no change in the magnitude of sea level after several time steps (stuck limit)
INTERPOLATION MODULE (resampling and short gaps)	• Adjusting and re-sampling of the time interval • Interpolation of very short gaps (smaller than 10-25 mins, depending on the tidal range) • Interpolation of wrong values previously flagged on the QC-module
TIDE-SURGE MODULE	• Computation of tide and residuals
FILTER MODULE (hourly values)	• Application of a filter to obtain hourly values

Table 1: role/function of the main modules of the software.

The QC module mainly flags bad data. An algorithm based on iterative fits to a spline is applied for spikes detection and can be configurated depending on the tidal range and the sea level variability of the station (more details in Pérez-Gómez et al., 2013). With respect to the stability test, the number of data values or time steps to begin to flag depends on the time interval and can be configurated: a typical value, for example, is 3 for 5-minute data, or 10 for 1-min data.

If harmonic constants for the station we are processing are available, which is possible if we have at least one year of good quality data, we can apply a second quality control to the residual time series and repeat the scheme in an iterative process. The tide-surge module computes the tide and the residuals for the period of data (with the original time interval) and goes back to apply the first step (QC module) to the residual time series.

If new errors are detected here (for example, less obvious spikes, not so clear in the total sea level time series), the corresponding flags are annotated to the flagged file, and the whole process including the interpolation module is repeated.

The tidal package used is the one developed by Foreman, 1977. The software includes only the tidal prediction Fortran code. The harmonic constants should be computed apart with the Foreman harmonic analysis program, for consistency.

For some applications use of filtered hourly values is enough or even preferred, as high frequency data may contain other phenomena we are not interested in. In this case, from the 5-min generated time series we obtain hourly values by means of a symmetrical filter of 54 points, following the expression:

$$X_f(t) = F_0.X(t) + \sum_{m=1}^{M} F_m\left[X(t+m) + X(t-m)\right]$$

Where $X_f(t)$ is the hourly filtered value and $F_0 \ldots F_m$ the weights applied to the high frequency values. Details of this filter can be found in Pugh, 1987 (original Fortran code provided by Ian Vassie, 1992).

3. Requirements of the new version SELENE

SELENE complies with the following requirements:

- Developed in Python.
- Foreman Fortran77 code to calculate predictions in the actual version.
- Developed as a "core" independent of any operative to ease the exportation of the software to any other system and/or institution.
- Following the same philosophy of the original software in terms of terminology, data flows and products generated.
- Used in a near-real environment: Puertos del Estado REDMAR tide gauge network.
- Multi-platform: Linux, MacOS, Windows.
- Open to addition of new modules/QC tools in the future

4. Design overview

SELENE is structured around four main sections/folders (see figure 2 detailed structure):

- ***configuration***: basic configuration files of the customization component
- ***foreman***: Foreman's prediction program
- ***modules***: main modules of SELENE, mainly based on the ones described in section 2, translated to Python.
- ***utils***: additional utilities and functionalities to support the modules.

Input raw data files are stored in ***datafiles*** directory, and output files in the ***output*** directory. The main python command is selene.py. When run interactively, the output files can be visualized to check the quality control process with selenevis.py. Messages with information about processing steps and bad data detection can be followed on in **selene.log** (***logs*** directory).

4.1. Configuration

The ***configuration*** folder contains the customization and configuration files of the software.

If available, **harmonic constant files** should be under "***configuration/harmonics***" folder. Another subfolder, ***configuration/auxfiles***, contains two fixed files: **astodata**, required to build the input for Foreman's prediction program, and **pughtaps**, required by filterpugh.py to build the lowpass FIR filter used to get the hourly values. This is the default filter applied to hourly values at this moment, although use of other filters should be feasible and easy with this structure.

SELENE can be configured in the three levels described below.

4.1.1. Constants

constans.py: this file contains several constants that in principle should not change. It is not recommended to modify this file therefore although, in some cases, it could be done by an advanced user.

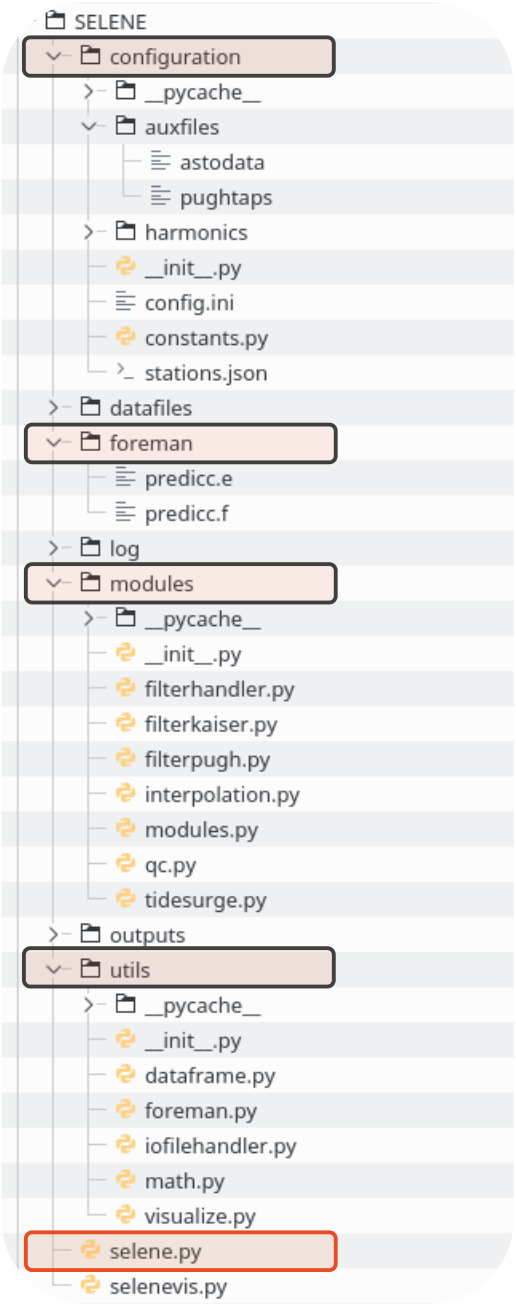

Figure 2: SELENE files structure: input and output files in ***datafiles*** and ***output*** directories. Main script: **selene.py**. Visualization script: **selenevis.py**

4.1.2. General configuration file

config.ini: general configuration file that should be modified according to the needs of the user. The parameters/information contained in this file are general for all the stations. Example:

```
[general]
outputfolder = outputs/

[foreman]
program = foreman/predicc.e
harmonicsfolder = configuration/harmonics/

[interpolation]
maxgapinminutes = 25
subsamplingmethod = first

[filterhandler]
cutoff = 0.00013889
```

- *outputfolder*: folder to write the outputs.
- *program*: compiled Fortran77 Foreman's prediction program.
- *harmonicsfolder*: folder where the harmonic files are located.
- *maxgapinminutes*: value used by the interpolator module as the maximum gap to be ignored. If the time series has a gap bigger than this value it will be not filled in with interpolated data.
- *subsamplingmethod*: value used by the interpolator module to decide which method is used during resampling. It could be 'first' (by default) or 'mean'. More information in:

  https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.resample.html
- *cutoff*: cutoff frequency used by the filter module if the Kaiser filter is used. This filter has not been tested in this version, the original Pugh filter is applied and this parameter not needed in SELENE 1.0.

4.1.3. Stations configuration file

stations.json: JSON file with the parameters that can be customized or defined independently for each station, if needed. For example, the following information should be provided in this standard JSON file for each tide gauge (make sure symbols like "{" and "}" are included at the beginning and end of the file, the commas, etc):

```
{
"3853":
        {
        "name": "Alcudia",
        "shortname": "Alcu",
        "latitude": 39.8333,
        "longitude": 3.14,
        "seriesfile": "datafiles/3853.data",
        "seriesseparator": "",
        "seriesdatecolumns": "1,2",
        "seriesdateformat": "%Y%m%d%H%M%S",
        "seriesvaluecolumn": 3,
        "seriesqccolumn": 4,
        "qc_level_nsigma": 4,
        "qc_level_winsize": 200,
        "qc_level_splinedegree": 2,
        "qc_surge_nsigma": 5,
        "qc_surge_winsize": 1000,
        "qc_surge_splinedegree": 3,
        "qc_stucklimit": 5,
        "foremanharmfile": "Alcudia/harm853.10"
        },                                  *Note: omit this "," if last station in the file
}
```

- The station id is the main label used for identification of the station (usually a number), in this case "3853".
- *name:* human-readable name of the station (*short-name* equivalent short version, used in the REDMAR network).
- *latitude*: latitude of the station (float).
- *longitude*: longitude of the station (float).
- *seriesfile*: path and file name with input raw data (ascii column file).
- *seriesseparator*: column separator in the input raw data file.
- *seriesdatecolumns*: comma separated values with the columns containing the date and time considering the separator indicated in *seriesseparator*.
- *seriesdateformat*: date format considering the date as the concatenation of values of the columns indicated in *seriesdatecolumns*. It follows the python datetime format:

  https://docs.python.org/3/library/datetime.html#strftime-strptime-behavior
- *seriesvaluecolumn*: column of raw sea level data considering the separator indicated in *seriesseparator*.
- *seriesqccolumn*: column containing the quality control flag considering the separator indicated in *seriesseparator*.

- *qc_level_nsigma*: sigma value used to flag data by the spline fit algorithm, applied to raw sea level data
- *qc_level_winsize*: window size of the spline fit algorithm for raw sea level data
- *qc_level_splinedegree*: spline degree of the spline fit algorithm applied to raw sea level data
- *qc_surge_nsigma*: sigma value used to flag data by the spline fit algorithm, applied to residual data
- *qc_surge_winsize*: window size of the spline fit algorithm for residual data
- *qc_surge_splinedegree*: spline degree of the spline fit algorithm applied to residual data
- *qc_stucklimit*: maximum number of data with same value (stuck value) allowed in the quality control module
- foremanharmfile: harmonic constants file (Foreman program). The path is relative to the *harmonicsfolder* element included in config.ini. If this field is empty (""), the tide and residuals are not computed (only the first step of the quality control is applied to total sea level).

4.2. Utils

Package utils contains a collection of utilities and functions to support the tasks carried out by the different modules.

dataframe.py: contains two functions related with pandas dataframe.

- mininterval(dataframe): calculates the minimum interval in the dataframe
- save(dataframe,filename): saves a dataframe to a file.

foreman.py: two main functions:

- createinputfile(targetfile,astofile,harmfile,startdate,enddate,mode,interval, idstation,name,latitude,longitude,logger): prepares the input file to foreman's prediction program (notice the format expected with the examples attached for the harmonic constant files):
  - *targetfile*: file to write the result, the Foreman's input file.
  - *astofile*: predefined file located in *configuration/auxfiles/astodata*. The location is defined in constants.py (foremanastofile)

- *harmfile*: harmonics file for the station, defined in stations.json (foremanharmfile)
- *startdate*: starting date of tide prediction
- *enddate:* ending date of tide prediction
- *mode*: it could be 'EQUI' or 'EXTR' to define equidistant or extremes (highs and lows) mode in tide prediction. SELENE uses only 'EQUI' mode so it is set in the constants.py (foremanmode)
- *interval*: time interval of the prediction returned by the Foreman prediction program, in hours. Intervals accepted are defined in constants.py:

```
foremanmininterval = 0.1
foremanhourlyinterval = 1.0
```

0.1 → 0.1 hours = 6 minutes

1.0 → hourly values

- *idstation*: station id, defined in stations.json (main label)
- *name*: station name, defined in stations.json (name)
- *latitude*: latitude, defined in stations.json (latitude)
- *lontigude*: longitude, defined in stations.json (longitude)
- *logger*: internal attribute to pass the python logger.

- run(infile,outfile,logger,config): run foreman's prediction program. The program is set in config.ini (foreman→program)
  - *infile*: input file created with *createinputfile* method
  - *outfile*: output file generated
  - *logger*: internal attribute to pass the python logger
  - *config*: configuration file instance (internal)

iofilehandler.py: utility to read and process text files.

- txtfile2dataframe(txtfile,separator,datecolumns,datepattern,datacolumn, qualitycolumn): utility to read the sea level data series input file.
  - *txtfile*: sea level data text file, defined in stations.json (seriesfile)

- *separator:* columns separator of the text file, defined in stations.json (seriesseparator)
- *datecolumns:* comma separated values with the columns containing the date and time considering the separator indicated in *separator*, defined in stations.json (seriesdatecolumns)
- *datepattern*: date format considering the date as the concatenation of values of the columns indicated in *datecolumns*. It follows the python datetime format:

  https://docs.python.org/3/library/datetime.html#strftime-strptime-behavior

  It is defined in stations.json (seriesdateformat)
- *datacolumn*: column containing the sea level datum considering the separator indicated in *seriesseparator*, defined in stations.json (seriesvaluecolumn)
- *qualitycolumn*: : column containing the quality control flag considering the separator indicated in *seriesseparator*, defined in stations.json (seriesqccolumn)

- filterwindow(filfile): utility to read the pughtaps file (filter weigths) to apply the lowpass FIR filter in the filtherpugh.py module.
  - *filfile*: predefined file located in *configuration/auxfiles/pughtaps*. The location is defined in constants.py (pughtaps)

selenemath.py: additional math functions used by the software, for example:

- rmse(predictions, targets): calculate the root mean square error between two arrays.
  - *predictions*: array with result of the data fitted to spline.
  - *targets:* array with original data.

visualize.py: utility to plot the results.

- plotsubplots(dfs): plots the data frames in different subplots.
  - *dfs*: array of tuples with the form [data frame, legend text, color, subplot]
- plots(dfs): plots the data frames in the same plot.
  - *dfs*: array of tuples with the form [data frame, legend text, color]

4.3. Modules

This package contains the main modules used in the program.

4.3.1. Quality Control module - qc.py

qc.py: main quality control and flagging of the raw sea level/residual data.

- qc(originaldf,nsigma,winsize,splinedegree,stucklimit,logger,c): the only method of this module. It performs the quality control processing and returns a flagged data frame.
    - *originaldf*: sea level data or tide surge data frame.
    - *nsigma*: number of sigmas to identify an event
    - *winsize:* number of data of the window to fit the spline
    - *splinedegree*: degree of the spline
    - *stucklimit*: steps with same value to consider stuck value
    - *logger*: internal attribute to pass the python logger
    - *c*: internal attribute to pass the constants

Note: in some cases, the polyfit method of numpy employed to fit the spline can rise a RankWarning:

```
#qc.py:63: RankWarning: Polyfit may be poorly conditioned splinefit =
np.polyfit(winx, windata, splinedegree)
```

According to the documentation, it indicates that the rank of the coefficient matrix in the least-squares fit is deficient. It happens mostly when a tide surge data series is processed, but it is the expected behavior. To avoid this warning messages it is used:

```
warnings.simplefilter('ignore', np.RankWarning)
```

How it works?

1. Copy of the original data frame to work with
2. Deletes original data flagged as bad data
3. Performs the stability/stuck test.
4. Bad data detected during stuck test are deleted.

5. Performs the spike detection test: the algorithm for detection of spikes is the main component of the qc module: it is based on the fit of a spline to a moving window.

   The degree of the spline (usually 2 or 3) and the size of the window can be selected and determined depending on the characteristics of the tide, the data sampling, etc. It is configured in stations.json. The algorithm flags as spikes the values that differ more than n sigmas from the fit (usually larger than 3, this can also be defined in the stations.json file). Repeating the process for non-tidal residuals is needed sometimes to detect less obvious spikes not detected in the first step; in this case, the qc module is applied again when the residuals are obtained.

   The algorithm loops until no new spikes are detected. After each iteration, the bad data detected are deleted.

6. Data not marked during the quality control process are set to good data.

4.3.2. Interpolation module - interpolation.py

interpolation.py: module that performs resampling and interpolation of short gaps.

- interpolate(originaldf,intervalinterpolate,mininterval,logger,config,c): the only method of interpolation module. It performs an interpolation and return a data series with no gaps and the final nominal time sampling.
  - *originaldf*: data frame to be interpolated
  - *intervalinterpolate*: target time sampling. It can be None. In that case, the target time sampling is the original time sampling of the data series (*mininterval*).
  - *mininterval*: raw data series time sampling. The minimum interval between two points of the data series is the time sampling of the data series.
  - *logger*: internal attribute to pass the python logger
  - *config*: configuration file instance (internal)
  - *c*: internal attribute to pass the constants

How it works?

1. Copy of the original data frame to work with, in this case the output of the QC module (raw data with flags)

2. If *intervalinterpolate* is not set (None), the target time sampling will be the original data frame time sampling (*mininterval*).
3. Original data flagged as bad are deleted.
4. The data frame is processed in fragments when there are gaps bigger than *maxgapinminutes* value (configured in config.ini → interpolation). If gaps are smaller, they are skipped and treated as no gap. Each fragment is resampled and interpolated to its original time sampling.
5. Then, once the data frame has no gaps in its original time sampling, it is resampled to the target time sampling using the method set in *subsamplingmethod*, in config.ini → interpolation. By default, the method used is 'first'.

4.3.3. Filter module - filterhandler.py

A handler was created to allow use of different filters in future versions (even others implemented by the user). SELENE 1.0 provides two filters, **pugh** (filterpugh.py) and **kaiser** (filterkaiser.py) the latter not tuned and tested properly, so its use not recommended as it is.

filterhandler.py: main module to invoke a filter to hourly values, a kind of interface to manage the different filters behind it.

- filt(filtername,df,minsampling,logger,config,c): the only method. It is responsible to call the selected lowpass FIR filter.
    - *filtername*: filter to use, e.g. 'pugh' or 'kaiser'
    - *df*: data frame to be filtered
    - minsampling: data frame's time sampling
    - logger: internal attribute to pass the python logger
    - config: configuration file instance (internal)
    - c: internal attribute to pass the constants

filterpugh.py: 'pugh' filter.

- filt(df,logger,c): It calculates hourly values by means of a symmetrical filter of 54 points (M), following the expression:

$$X_f(t) = F_0.X(t) + \sum_{m=1}^{M} F_m \left[X(t+m) + X(t-m) \right]$$

Where $X_f(t)$ is the hourly filtered value and $F_0 \ldots F_m$ the weights applied to the high frequency values. Details of this filter can be found in Pugh, 1987.

The weights are located in **pughtaps** (under configuration/auxfiles folder). To translate the content of this file to a filter window:

```
utils.iofilehandler.filterwindow(pughtaps)
```

- *df*: data frame to filter
- *logger*: internal attribute to pass the python logger
- *c*: internal attribute to pass the constants

filterkaiser.py: 'kaiser' filter.

- filt(df,minsampling,logger,config): It calculates hourly values using **scipy.signal** package, and its function firwin. Some work must be done to polish the Nyquist rate, the cutoff frequency…
  - *df*: data frame to filter
  - *minsampling*: data frame's time sampling
  - *logger*: internal attribute to pass the python logger
  - *config*: internal attribute to pass the constants

How it works?

1. filterhandler invoked using 'pugh' or 'kaiser'.
2. Using 'pugh' filter:
   - The filter window is created from pughtaps file.
   - The formula is applied using the weights of the previous step.
   - Data is resampled to hourly values with 'first' method.
3. Using 'kaiser' filter:
   - sample rate, Nyquist rate, cutoff frequency parameters are calculated from the time sampling.

- The order N and the term beta, Kaiser window required parameters, are calculated.
- The FIR filter is applied using the previously generated Kaiser window.
- Data is resampled to hourly values with 'first' method.

4.3.4. Tide surge module - tidesurge.py

The foreman.py utility has been created and located in ***utils*** package. This module is in charge of running the foreman prediction program, written in FORTRAN77 (not translated to Python). Future versions of SELENE will include possibility of using other tidal analysis and prediction packages.

tidesurge.py: module that computes the tide and the sea level residual.

- tidesurge(infile,outfile,astofile,harmfile,startdate,enddate,mode,interval,stationid,stationname,latitude,longitude,originalseries,logger,config): it calculates and returns a data frame with residuals from a sea level data frame. The residuals are obtained by subtracting the predicted sea level data obtained by foreman.py.
  - *infile*: input file for Foreman's prediction program.
  - *outfile:* output file produced by Foreman's prediction program.
  - *astofile*: predefined file located in *configuration/auxfiles/astodata*. The location is defined in constants.py (foremanastofile)
  - *harmfile*: harmonics file for the station, defined in stations.json (foremanharmfile)
  - *startdate*: starting date to calculate the prediction
  - *enddate:* ending date to calculate the prediction
  - *mode*: it could be 'EQUI' or 'EXTR' to define equidistant or extremes (highs and lows) mode. SELENE uses only 'EQUI' mode, as set in the constants.py (foremanmode)
  - *interval*: time interval (hours) of the prediction returned by the Foreman prediction program. Intervals accepted are defined in constants.py:

    ```
    foremanmininterval = 0.1
    foremanhourlyinterval = 1.0
    ```

    0.1 → 0.1 hours = 6 minutes

    1.0 → hourly values
  - *idstation*: station id, defined in stations.json (main label)

- *name*: station name, defined in stations.json (name)
- *latitude*: latitude, defined in stations.json (latitude)
- *lontigude*: longitude, defined in stations.json (longitude)
- *originalseries*: original data frame
- *logger*: internal attribute to pass the python logger.

How it works?

1. Creates the input file for Foreman's prediction software.
2. Run Foreman's prediction program to get the tide predictions.
3. Reads and parses the Foreman's output file.
4. Resamples the prediction to the same interval as the original data frame.
5. Subtracts the prediction to the original sea level data to get the residuals.

5. How SELENE works

The main program is selene.py. It is the basic controller that invokes the different modules and orchestrate the whole process, according to the following steps:

1. Command: the only argument accepted is a **station id**. It must match one of main labels in the stations.json. Example: `python selene.py 3114`
2. Logger initialized: this object will be shared with the different modules and utilities during the process.
3. Parsing the configuration file config.ini, configuration object that will be shared during the whole process.
4. Loading stations.json file.
5. If a station is not configured in stations.json, the process is terminated.
6. Parsing the input raw sea level data series file (stations[stationid]['seriesfile']):

```
originaldf =
iofilehandler.txtfile2dataframe(stations[stationid]['seriesfile'],stations[st
ationid]['seriesseparator'],list(map(int,stations[stationid]['seriesdatecolum
ns'].split(','))),stations[stationid]['seriesdateformat'],stations[stationid]
['seriesvaluecolumn'],stations[stationid]['seriesqccolumn'])
```

7. Quality control check to get a data frame with original sampling and flags:

```
originaldfwithflags=
qc.qc(originaldf,stations[stationid]['qc_level_nsigma'],stations[stationid]['
qc_level_winsize'],stations[stationid]['qc_level_splinedegree'],stations[stat
ionid]['qc_stucklimit'],logger,c)
```

8. If harmonic file exists for this station: 9-12, else: go to 13.
9. Original sea level data series flagged are interpolated to its time sampling:

```
originaldfinterpolated=
inter.interpolate(originaldfwithflags,None,originalmininterval,logger,conf,c)
```

10. Tide and surge computation:

```
tidesurgedf = surge.tidesurge('foreman_min_' + stationid + '.in',
'foreman_min_' + stationid + '.out',c.foremanastofile,
config['foreman']['harmonicsfolder']+stations[stationid]['foremanharmfile'],f
oremanstartdate,foremanenddate,c.foremanmode,c.foremanmininterval,stationid,s
tations[stationid]['name'],stations[stationid]['latitude'],stations[stationid
]['longitude'],originaldfinterpolated,logger,config)
```

11. Quality control check applied to surge/residuals time series:

```
tidesurgedfwithflags=
qc.qc(tidesurgedf,stations[stationid]['qc_surge_nsigma'],stations[stationid][
'qc_surge_winsize'],stations[stationid]['qc_surge_splinedegree'],stations[sta
tionid]['qc_stucklimit'],logger,c)
```

12. Bad data detected in previous step marked in original sea level data series.

```
tidesurgebadflags = tidesurgedfwithflags[tidesurgedfwithflags.quality ==
c.badqc]
  if not tidesurgebadflags.empty:
    for ix in tidesurgebadflags.index.values:
      originaldfwithflags.loc[ix,'quality'] = c.badqc
```

13. Flagged sea level data resampled to 5 min sampling:

```
fivemindfinterpolated =
inter.interpolate(originaldfwithflags,5,originalmininterval,logger,config,c)
```

14. 5 min time series filtered to get hourly values:

```
hourlyfiltered = filt.filt('pugh',fivemindfinterpolated,5,logger,config,c)
```

15. If harmonic file exists for the station: hourly surge/residuals time series is obtained:

```
hourlytidesurgedf = surge.tidesurge('foreman_hour_' + stationid + '.in',
'foreman_hour_' + stationid + '.out',
c.foremanastofile,config['foreman']['harmonicsfolder']+stations[stationid]['f
oremanharmfile'],foremanstartdate,foremanenddate,c.foremanmode,c.foremanhourl
yinterval,stationid,stations[stationid]['name'],stations[stationid]['latitude
'],stations[stationid]['longitude'],hourlyfiltered,logger,config)
```

16. End program.

6. Validation / visualization of the process

Additionally, SELENE includes a visualization tool, selenevis.py, to validate and display the different products produced by selene.py, and verify the correct performance and detection of wrong data. This is always important in an automatic process that cannot perform correctly in the 100% of the cases. It allows the user to see an only product (one of the time series generated) or a combined view with all the products of the process for a specific station.

There are two options:

1. Display one of the main outputs or products stored in ***outputs*** folder:

It must be invoked with argument -files followed by a list of comma-separated files, including the path, for example, if interested on original sampling with flags of bad data, for Bilbao (3114):

```
python selenevis.py -files outputs/3114_original_sampling_flags.out:
```

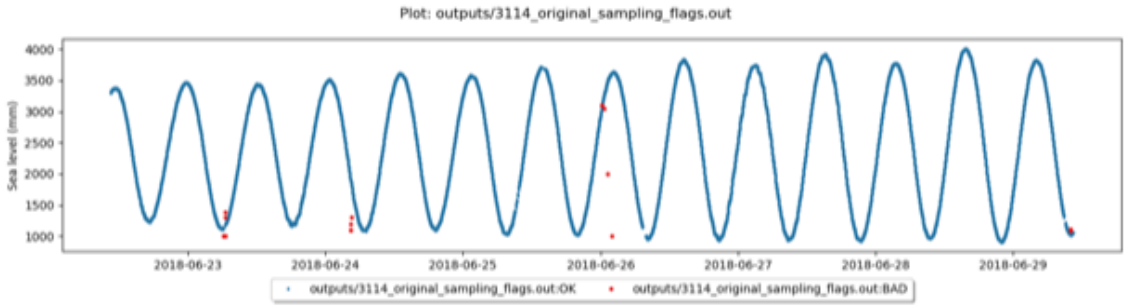

Figure 3: output of visualization script **selenevis.py** for a specific file in ***outputs.*** (Bad data display in red)

2. Display a **combined view** of all the products for a station:

It must be invoked with arguments -station followed by the code of the station:

`python selenevis.py -station 3510` (figure …., for Melilla station)

`python selenevis.py -station 3114` (figure …., for Bilbao station)

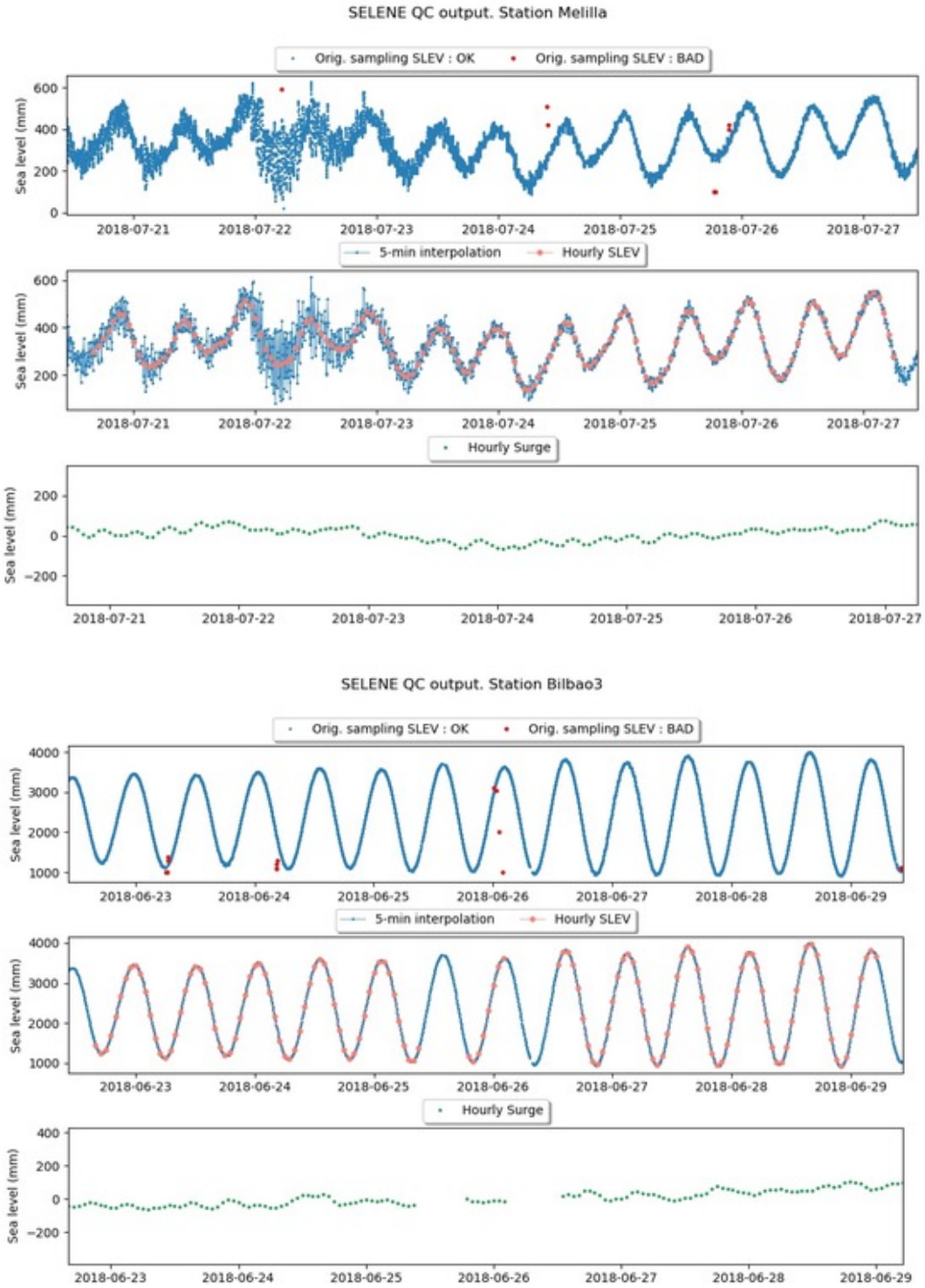

Figure 4: combined visualization with script **selenevis.py** for Melilla tide gauge (top) and for Bilbao tide gauge (bottom), including flags of wrong data (red), 5-min and hourly products, and residuals).

7. Get Started

7.1. Environment

- Python 3.6.X
- Anaconda 3:

  https://www.anaconda.com/download/

  https://docs.anaconda.com/anaconda/

7.1.1. Libraries used:

All the packages are included in Anaconda 3 distribution:

Scientific:

```
import pandas
import scipy.signal
import numpy as np
```

Auxiliary:

```
import configparser
import time
import datetime
import sys
import os
import logging
import json
import warnings
```

7.2. Requisites

Gfortran compiler, available for Linux, MacOS and Windows systems.

https://gcc.gnu.org/wiki/GFortran

7.3. Installation

Once the required python version, the python packages (Anaconda 3) and gfortran compiler are installed:

1. Copy SELENE folder to your file system. Destiny path is left at user choice.

2. Go to <SELENE_PATH>/SELENE/foreman

3. Compile predicc.f

   Linux & MacOS:

   ```
   gfortran -fno-automatic -ffixed-line-length-132 -O2 predicc.f -o predicc.e
   ```

   Windows:

   ```
   gfortran -fno-automatic -ffixed-line-length-132 -O2 predicc.f -o predicc.exe
   ```

   The executable predicc.e or predicc.exe is generated.

4. Give execution permissions to the new compiled program - only Linux & MacOS:

   ```
   chmod 755 predicc.e
   ```

7.4. Configuration

Configure SELENE:

1. Modify <SELENE_PATH>/SELENE/configuration/config.ini

   By default:

   ```
   [general]
   outputfolder = outputs/

   [foreman]
   program = foreman/predicc.e
   harmonicsfolder = configuration/harmonics/
   ```

```
[interpolation]
maxgapinminutes = 25
subsamplingmethod = first

[filterhandler]
cutoff = 0.00013889
```

[foreman] program must be modified to indicate the path to the executable Fortran created during the installation. The path could be relative or absolute.

Optionally, **[general] outputsfolder** and **[foreman] harmonicsfolder** paths could be modified although is not recommended.

Advanced users could be interested in modifying the other values. More details in 4.1.2 General configuration file section.

2. Configure stations.json, including information of the stations or tide gauges.

   More details in 4.1.3. Stations configuration file section.

7.5. Run

Once the application is configured, and the input data ready in the folder ***datafiles***, just run the following command for one of the stations:

```
python selene.py <station_id>
```

*** Notice that <station_id> must match the element label in the JSON stations.json.

7.6. Visualization

To visualize one product:

```
python selenevis.py -files <outputsfolder>/<product_file_name>
```

To visualize one station (combined view of its products):

```
python selenevis.py -station <station_id>
```

ANNEX A: Use case - Puertos del Estado

SELENE Software is integrated in Puertos del Estado operative and running automatically. This component is very important and highly dependent on the institution and operative system. Therefore, the example for Puertos del Estado is shown here.

One of the main goals of SELENE was to keep the 'core' software completely independent with the objective to export it easily to other systems / institutions.

In this annex a description is presented of how SELENE can work integrated in a full operative, including data download, quality control and consolidation of flagged data in a data base.

The structure is as follows:

Figure 5: structure of the additional modules for integration of SELENE in the automatic procedure and operative, example at Puertos del Estado.

A python launcher, selenelauncher.py, was created to guide the process. This is in fact the command that will be included in the *crontab* file to be run automatically every 15 minutes.

selenelauncher.py → main program / launcher

1. No arguments are required:

   ```
   python selenelauncher.py
   ```

2. Configuration inputs are directly set at the beginning:

   ```
   #inputs
   logfile='log/selenelauncher.log'
   stationsfile='SELENE/configuration/stations.json'
   processors = 0
   ```

   - *logfile*: logs file

- *stationsfile:* stations.json path (absolute or relative)
- *processors:* it is the most interesting parameter. The program is configured to be executed with multiprocessing. This number indicates how many CPUs will be dedicated. 0 means all CPUs available:

3. Loads stations.json file.
4. Downloads data from the data base: selenedatadownload.py (more details next subsection)
5. Create multiprocessing.Pool indicating the number of dedicated CPUs
6. Map the list of stations to process in the pool:

```
p = mp.Pool(processes=processors)
p.map(selene,list(stations.keys()))
```

7. Consolidate data: selenedbconsolide.py (more details next subsection). Once all the stations are processed, quality control flags and final products are stored in the data base.

selenedatadownload.py → download data from data base

1. No arguments are required:

```
python selenedatadownload.py
```

2. Configuration inputs are directly set at the beginning:

```
logfile='log/selenedatadownload.log'
dbconfigfile='dbconfig.json'
dbconfigdatabase='doris'
daystosubstract=7
stationsfile='SELENE/configuration/stations.json'
stationsdatafilesfolder='SELENE/datafiles/'
```

- *logfile*: logs file
- *dbconfigfile*: data base connection configuration file. It is a JSON with this content:

```
{
"<database_connection_id>":["<database_host>","<database_nam
e>","PostgreSQL","<user>","<passwd>"]
}
```

- *dbconfigdatabase*: data base connection id to use. It must be defined in *dbconfigfile*.
- *daystosubstract*: number of days to download.
- *stationsfile*: stations.json path (absolute or relative)
- *stationsdatafilesfolder*: target folder to download data. It is important this folder is coherent with the values specified in stations.json in its elements 'seriesfile'.

3. It is fundamental to use a proprietary package to connect to data base (more documentation on demand):

```
import mftoolspy.database.mfdbhandler
```

4. Data base connection is created and initialized:

```
postgres =
mftoolspy.database.mfdbhandler.mfdbhandler(dbconfigdatabase,logf
ile,logging.INFO,dbconfigfile)
```

5. Loads stations.json file.

6. Before downloading data, all flags are reset to good data flag:

```
sqlupdate = "update mareas.redmar_mir_tr set qc_nivel=1, qc_ol=1
where codigo in (" + ','.join(list(stations.keys())) + ") and
qc_nivel=4 and fecha >= (now() - interval '" +
str(daystosubstract) + " day')"

postgres.query(sqlupdate)
```

7. Finally, for each station, data are downloaded and written to a file with the format SELENE is expecting:

```
sql = "select codigo,fecha,nivel,qc_nivel from
mareas.redmar_mir_tr where fecha > '" + str(inidate) + "' and
fecha <= '" + str(enddate) + "' and codigo = " + station + "
order by fecha ASC"

for row in postgres.query(sql):
  f.write(station + ' ' + row[1].strftime('%Y%m%d %H%M%S') + ' '
  + str(row[2]) + ' ' + str(row[3]) + '\n')
```

selenedbconsolide.py → update data in data base

1. No arguments are required:

```
python selenedbconsolide.py
```

2. Configuration inputs are directly set at the beginning:

```
logfile='log/selenedatadownload.log'
dbconfigfile='dbconfig.json'
dbconfigdatabase='doris'
stationsfile='SELENE/configuration/stations.json'
stationsdatafilesfolder='SELENE/datafiles/'
```

   - *logfile*: logs file
   - *dbconfigfile*: data base connection configuration file. It is a JSON with this content:

```
{
"<database_connection_id>":["<database_host>","<database_name>","PostgreSQL","<user>","<passwd>"]
}
```

   - *dbconfigdatabase*: data base connection id to use. It must be defined in *dbconfigfile*.
   - *stationsfile:* stations.json path (absolute or relative)
   - *stationsdatafilesfolder:* target folder to download data. This folder must be coherent with the values specified in stations.json, in its element 'seriesfile'.

3. It is fundamental to use a proprietary package to connect to the data base (more documentation on demand):

```
import mftoolspy.database.mfdbhandler
```

4. Data base connection is created and initialized:

```
postgres =
mftoolspy.database.mfdbhandler.mfdbhandler(dbconfigdatabase,logfile,logging.INFO,dbconfigfile)
```

5. Load stations.json file.

6. Finally, for each station, the product with flagged data is accessed and read. For each line, an update of the database is executed:

```
if os.path.isfile(stationsoutputsfolder + station +
'_original_sampling_flags.out'):
    with open(stationsoutputsfolder + station +
    '_original_sampling_flags.out','r') as f:
            for row in enumerate(f):
                values = row[1].split()
                if int(values[3]) == 4:
                   sql = "update mareas.redmar_mir_tr set
                   qc_nivel = 4, qc_ol = 4 where codigo = " +
                   station+ " and fecha = " + (values[0] + ' ' +
                   values[1]).replace('"','\'')

                   postgres.execute(sql)
```

REFERENCES

Foreman, F.G.G. Manual for tidal heights analysis and predictions. Pacific Marine Report No. 77. Technical Report, Institute of Ocean Sciences, Patricia Bay, 1977.

Álvarez-Fanjul, E., Pérez-Gómez, B. and Rodríguez Sánchez-Arévalo, I., 2001. Nivmar: a storm surge forecasting system for Spanish Waters. Scientia Marina, 65 (Suppl. 1), 145-154.

García, M.J., Pérez-Gómez, B., Raicich, F., Rickards, L., Bradshaw, E., Plag, H.P., Zhang, X., Bye, B.L. and Isaksen, E., 2005. European Sea Level Monitoring: Implementation of ESEAS Quality Control, in: Dynamic Planet: Monitoring and Understanding a Dynamic Planet with Geodetic and Oceanographic Tools, Chapter 11, p. 67. IAG Symposium, Cairns, Australia. Ed: Paul Tregoning and Chris Rizos.

Pérez-Gómez, B., de Alfonso, M., Huess, V. and Rickards, L., 2010. Near-real time quality control and validation of sea level in-situ data within MyOcean. MyOcean Document. Technical report, Puertos del Estado.

Pouliquen, S., 2011. Recommendations for in-situ data Real Time Quality Control. EuroGOOS Publication Report No. 27. Technical report, EuroGOOS, Norrkoping, Sweden.

Pérez-Gómez, B., Alvarez Fanjul, E., Pérez, S., de Alfonso, M. and Vela, J., 2013. Use of tide gauge data in operational oceanography and sea level hazard warning systems. Journal of Operational Oceanography, 6(2), 1–18.

Contact for support:

Dr. Begoña Pérez Gómez: bego@puertos.es

Puertos del Estado

Avda. Del Partenón, 10 28042 Madrid - Spain