SWOT Simulator Documentation

Release 2.3.0

Lucile Gaultier, Clement Ubelmann and Lee-Lueng Fu

Mar 15, 2017
## CONTENTS

### 1 SWOT Simulator for Ocean Science
1.1 Description ......................................................... 1
1.2 Licence ............................................................ 1
1.3 Installation ......................................................... 1
1.4 Running ............................................................ 2
1.5 Testing ............................................................. 2
1.6 Documentation ................................................... 2

### 2 SWOT Simulator for Ocean Science
2.1 Abstract: ............................................................ 3
2.2 Simulation of the SWOT sampling over synthetic Sea Surface Height ........................................... 3
2.3 Simulation of errors ................................................. 5
2.4 Simulation of errors for the nadir altimeter ................. 15
2.5 The software ....................................................... 16
2.6 References: ......................................................... 23

### 3 Evolution of the simulator
3.1 Changelog for swotsimulator ..................................... 25

### 4 Auto Generated Documentation
4.1 Main program: run_simulator .................................... 29
4.2 Read data: rw_data ............................................... 29
4.3 Build swath: build_swath ....................................... 33
4.4 Build errors: build_error ....................................... 33
4.5 Useful tools: mod_tools .......................................... 34

### 5 Open Source LICENCE
5.1 Software licence and copyright ................................ 37
5.2 Model data copyright ............................................ 37

### 6 Indices and tables

Python Module Index ..................................................... 41
Index ................................................................. 43
SWOT SIMULATOR FOR OCEAN SCIENCE

1.1 Description

This software simulates SWOT sea surface height (SSH) observations that can be applied to an ocean general circulation model (OGCM), allowing the exploration of ideas and methods to optimize information retrieval from the SWOT Mission in the future. From OGCM SSH inputs, the software samples SWOT-like outputs on a swath along the orbit ground track and adds measurement error and noise, which are generated according to technical characteristics published by the SWOT project team. The simulator can also be used to simulate nadir-like data for any altimeters.

The swotsimulator code can be downloaded at http://swot.jpl.nasa.gov/science/resources/

1.2 Licence

The swotsimulator is under LGPL licence (see COPYING)

1.3 Installation

The code runs with python and uses the following libraries:

- Numpy
- Scipy
- NetCDF4 if you need to read netcdf4 model files (not included in Canopy)

If you don’t have python and the needed python libraries on your machine you can download the enthought python distribution canopy at <https://store.enthought.com/> and follow the installation instructions.

To install swotsimulator:

-> If you have installed canopy:

Open a canopy terminal (Tools - Canopy Terminal)

> cd [yourpath]/swotsimulator/
> python setup.py install

-> If you have your own python and libraries:

- global installation (requires root privilege):

> sudo python setup.py install
1.4 Running

Run the SWOT simulator:

```
> python run.py [your params file]
```

Run the nadir alone:

```
> python run_nadiralone.py [your params file]
```

For example, to run the example do:

- `open ./example/params_example.txt` and check that the directory path are correct, if needed modify `path of dir_setup, indatadir, outdatadir` with your own path (the path of the simulator directory).

- Run the simulator: `python run.py ./example/params_example.txt`

Note that if you have installed python using enthought Canopy, you should either use the Canopy terminal or run the canopy python instead of python (direct path or use an alias).

The data provided in the example were produced by the Regional Ocean Modeling System (ROMS) off the Oregon coast developed by Dr. Yi Chao and his team.

1.5 Testing

Outputs of the simulator can be tested using algorithm from the test directory.

To run all tests, go to the test directory:

```
> python run_all.py [your params file]
```

1.6 Documentation

- To build the documentation, in the doc directory:
  - Build html: `make html`
  - Build pdf: `make latexpdf`

  The build documentation files are located in `doc/build/html` and in `doc/build/latex`.
- for a complete description: see the doc directory or just run `pydoc PyDom`
2.1 Abstract:

This software simulates sea surface height (SSH) synthetic observations of the proposed SWOT mission that can be applied to an ocean general circulation model (OGCM), allowing the exploration of ideas and methods to optimize information retrieval from the SWOT Mission in the future. From OGCM SSH inputs, the software generates SWOT-like outputs on a swath along the orbit ground track, as well as outputs from a nadir altimeter. Some measurement error and noise are generated according to technical characteristics published by the SWOT project team. Not designed to directly simulate the payload instrument performance, this SWOT simulator aims at providing statistically realistic outputs for the science community with a simple software package released as an open source in Python. The software is scalable and designed to support future evolution of orbital parameters, error budget estimates from the project team and suggestions from the science community.

2.2 Simulation of the SWOT sampling over synthetic Sea Surface Height

From a global or regional OGCM configuration, the software generates SSH on a 120–km wide swath at typically 1–km resolution. An illustration of outputs for a global ECCO (MITgcm) configuration is shown on Fig. 1.

2.2.1 Proposed SWOT orbits

The software uses as an input the ground-tracks of the satellite orbit. The user can choose between different orbits such as the fast sampling orbit (1-day repeat), the science orbit (21-day repeat with a 10-day subcycle) and also the contingency orbit (21-day repeat with 1-day subcycle). The table below shows the characteristics of these 3 orbits:

<table>
<thead>
<tr>
<th>Orbits</th>
<th>Repeat Cycle (days)</th>
<th>Repeat Cycle (Orbits)</th>
<th>Sub-cycles (days)</th>
<th>Inclination</th>
<th>Elevation (km)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast Sampling orbit</td>
<td>0.99349</td>
<td>14</td>
<td>N.A.</td>
<td>77.6</td>
<td>857</td>
</tr>
<tr>
<td>Science Orbit</td>
<td>20.8646</td>
<td>292</td>
<td>1, 10</td>
<td>77.6</td>
<td>891</td>
</tr>
<tr>
<td>Contingency orbit</td>
<td>20.8639</td>
<td>293</td>
<td>1</td>
<td>77.6</td>
<td>874</td>
</tr>
</tbody>
</table>
The ground-track coordinates corresponding to these orbits are given as input ASCII files of 3 columns (longitude, latitude, time) for one complete cycle sampled at every ~5–km. The ascending node has been arbitrarily set to zero degree of longitude, but the user can shift the orbit by any value in longitude.

Orbit files have been updated with the one provided by AVISO on September 2015 (http://www.aviso.altimetry.fr/en/missions/future-missions/swot/orbit.html). There are two additional orbit files available in the last version of the simulator. Input files are also ASCII with 3 columns (time, longitude, latitude). Orbits are provided at low resolution and are interpolated automatically by the simulator. ‘ephem_calval_june2015_ell.txt’ contains the updated fast sampling orbit and ‘ephem_science_sept2015_ell.txt’ the updated science orbit.

Other orbit files of the same format (time, longitude, latitude) can also be used as an input. To avoid distortions in the SWOT grid, we recommend a minimum of 10km sampling between the ground-track points of the orbit.

### 2.2.2 The SWOT swath

From the orbit nadir ground track the software generates a grid covering the SWOT swath over 1 cycle. In the across-swath direction, the grid is defined between 10–km and 60–km off nadir. The grid size is 2 kilometers in the along-track and across-track directions by default, but can be set at any other value (e.g. 500–m or 250–m). The longitude and latitude coordinates are referenced for each grid point, and the time coordinate (between 0 and t_cycle) is referenced in the along-track direction only. A scheme of the SWOT grid is presented on Fig. 2. The SWOT grid is stored by pass (e.g. 292 ascending passes and 292 descending passes for the science orbit). A pass is defined by an orbit starting at the lowest latitude for ascending track and at the highest latitude for descending track (+/-77.6 for the considered SWOT orbits). The first pass starts at the first lowest latitude crossing in the input file, meaning that ascending passes are odd numbers and descending passes are even numbers.

### 2.2.3 Interpolation of SSH on the SWOT grid and nadir track

The input SSH must be given at regular time step, over any period of time. By default the absolute time of the first time step is zero and corresponds to the beginning of pass 1. The SSH is interpolated on the SWOT grid and nadir track for each pass and successive cycles if the input data exceeds 1 cycle. The nadir track has the same resolution as the SWOT grid in the along-track direction, but it is possible to compute it separately with a different along track resolution. On the SWOT grid, the 2D interpolation is linear in space. No interpolation is performed in time: the SSH on the SWOT grid at a given time corresponds to the SSH of the closest time step. This avoids contaminations of the rapid signals.
(e.g. internal waves) if they are under-sampled in the model outputs. However, note that locally, sharp transitions of the SSH along the swath may occur if the satellite happens to be over the domain at the time of transition between two time steps. Fig. 3a shows an input SSH as an example. Fig 3b is the interpolated SSH on a 400km long segment of the SWOT grid crossing the domain.

2.3 Simulation of errors

The software generates random realizations of instrument errors and noise over the interpolated SSH, as well as simulated geophysical errors. These error simulations can be adjusted to match updates of the error budget estimation from the SWOT project team. Fig. 4 shows a realization of a SWOT error field generated by the software. It is the sum of random realizations of multiple error components described in the following. Fig. 3c shows the “observed” SSH when simulated noise is added to the interpolated SSH.

2.3.1 Instrumental errors

The following components of instrumental errors are implemented in the software: the KaRIN noise, the roll errors, the phase errors, the baseline dilation errors and the timing errors. Random realizations of the noise and errors are performed following the statistical descriptions of the SWOT error budget document (Esteban-Fernandez et al., 2014). While cross-track calibration algorithms are not finalized, the long-wavelength components of the instrument errors are filtered beyond 2000–km by default in the software (to simulate approximately the residual errors after cross-track calibrations). The user interested to test algorithms for cross track calibration of instrument error must deactivate the filtering of the long wavelengths. Error drifts may then be as large as 20 meters at the edges of the swath.

The KaRIN noise

The KaRIN noise is random from cell to cell, defined by a Gaussian zero-centered distribution of standard deviation inversely proportional to the square root of the cell surface. In the simulator, the KaRIN noise varies with the distance to the nadir and the Significant Wave Height (SWH) specified as a constant value between 0 and 8 meters. For a grid cell of 1 km², the standard deviation of the KaRIN noise follows the curve shown on Fig. 5 with SWH varying from...
Fig. 2.3: FIG. 3: (a) SSH (in meters) produced by the Regional Ocean Modeling System (ROMS) off the Oregon coast developed by Dr. Yi Chao and his team. (b) SSH_model simulator output interpolated on the SWOT grid. (c) “Observed” SSH, which is the sum of SSH_model and a random realization of the total SWOT noise with the default parameters of the software.

Fig. 2.4: FIG. 4: Random realization of the error field (in meters). Swath coordinates are in km.
0 to 8-m (Esteban-Fernandez et al., 2014). Fig. 6 shows a random realization produced by the software with $1km^2$ grid cells and SWH=2-m.

Fig. 2.5: FIG. 5: The example curves of the standard deviation (cm) of the KaRIN noise as a function of cross-track distance (km).

Fig. 2.6: FIG. 6: Random realization of the KaRIN noise (m) following the standard deviation shown Fig. 5, with 2-km by 2-km grid cells and a 2-m SWH.

Note that the present version of the simulator only supports a constant value in time and space for SWH. In future versions, we may implement the use of time/space-varying fields of SWH as an input.

**Roll knowledge and control errors**

As detailed in Esteban-Fernandez et al., 2014, the roll error signal is the sum of two components: the roll error knowledge (also called gyro error) and the roll control errors. An estimation of the along-track power spectral density of the
two roll angles is given in the input file ‘global_sim_instrument_error.nc’ from the SWOT project. It is represented on Fig. 7.

![Fig. 7: Estimation of the power spectral density of the gyro error angle (blue) and the roll control error angle (green).](image)

Following these spectra, random realizations of an along-track roll angle $\theta_{roll}(al)$ are performed with uniform phase distribution. The algorithm of the random realization is described in APPENDIX A. From $\theta_{roll}(al)$ in arcsecond unit, the power spectrum of the gyro knowledge error plus the roll control error, the total roll error $h_{roll}$ (in meters) at a distance $ac$ (in km) from the nadir is given by (see Esteban-Fernandez et al., 2014):

$$h_{roll}(al, ac) = (1 + \frac{H}{Re})\theta_{roll}(al) \frac{\pi}{648}ac$$

where $H$ is the altitude of the satellite and $Re$ the earth radius. An example of realization is shown on Fig. 8.

![Fig. 8: Random realization of the roll error following the power spectra of the roll angle shown Fig. 7 (with filtering of long wavelengths).](image)

**Phase errors**

An estimation of the along-track power spectrum of phase error is also given in the input file ‘global_sim_instrument_error.nc’. It is represented on Fig. 9.

Following this power spectrum, random realizations of an along-track phase error $\theta$ (al) are performed with uniform phase distribution. From $\theta$ (al) in deg. unit, the phase error on the height $h_\theta$ (in meters) at a distance $ac$ (in km) from the nadir is given by (see Esteban-Fernandez et al., 2014):

$$h_\theta(al, ac) = \frac{1}{K_{Ka}B}(1 + \frac{H}{Re})\theta(al) \frac{100\pi}{18}ac$$

An independent realization of $\theta$ is chosen for the left ($ac<0$) and right ($ac>0$) swaths. As a result, the error is decorrelated between the 2 sides (as opposed to the case of roll error), as illustrated on the random realization shown on Fig. 10.
Baseline dilation errors

The baseline dilation and its resulting height measurement error is also implemented, although the errors are significantly less important than the roll and phase errors. The along-track power spectrum of the dilation $\delta B$ is also given in the input file ‘global_sim_instrument_error.nc’. It is represented on Fig. 11.

Following this power spectrum, random realizations of an along-track baseline dilation $\delta B$ are performed with uniform phase distribution. From $\delta B$ in $\mu m$, the baseline dilation error on the height $h_{\delta B}$ (in meters) at a distance $ac$ (in km) from the nadir is given by the following formula (see Esteban-Fernandez et al., 2014):

$$h_{\delta B}(al, ac) = -(1 + \frac{H}{Re}) \frac{\delta B(al)}{HB} ac^2$$
Timing errors

The timing errors are also minor compared to roll and phase errors, but are implemented in the software. The along-track power spectrum of the timing error $\tau$ is also given in the input file ‘global_sim_instrument_error.nc’. It is represented on Fig. 13.

Following this power spectrum, random realizations of an along-track timing error $\tau$ are performed with uniform phase distribution. From $\tau$ (al) in pico seconds, the timing error on the height $h_\tau$ (in meters) at a distance $ac$ (in km) from the nadir is given by (see Esteban-Fernandez et al., 2014):

$$h_\tau(al, ac) = \frac{c}{2} \tau(al) \times 10^{-12}$$

Where $c$ is the speed of light in m/s. The timing errors are constant in the across swath direction. An example is shown on Fig. 14.
2.3.2 Geophysical errors

So far, only the major geophysical source of error, the wet troposphere error, has been implemented in the software in a quite simple way. More realistic simulation will be hopefully implemented in the future versions.

Wet troposphere errors

The software simulates errors in the water vapor path delay retrieval with the option of a 1-beam radiometer configuration or a 2-beam radiometer configuration. First, a 2D random signal is generated around the swath following a 1D input spectrum, with uniform phase distribution as described in APPENDIX A. By default in the software, the 1D spectrum is the global average of estimated path delay spectrum from the AMSR-E instrument and from the JPL’s High Altitude MMIC Sounding Radiometer (Brown et al.) for the short wavelength. This spectrum is expressed by the following formula (in cm²/(cy/km)):

\[
S_{\text{wd}} = 3.156 \times 10^{-5} f^{-8/3} \quad \text{for} \quad 3000 \text{km} > \lambda > 100 \text{km} \\
S_{\text{wd}} = 1.4875 \times 10^{-4} f^{-2.33} \quad \text{for} \quad \lambda \leq 100 \text{km}
\]

Fig. 15 shows a random realization of the path delay following the above spectrum. By modifying the code, the user can change the power spectrum to match the water vapor characteristics of a particular region, by using for example the global climatology provided in Ubelmann et al., 2013.

![Random realization of wet-tropospheric path delay without correction](image)

Fig. 2.15: FIG. 15: Random realization of wet-tropospheric path delay without correction (in meters).

From the 2D random signal, the software simulates the residual error after correction for the estimated path delay from the radiometer. By default, the number of radiometer beams is set to 1. We considered that the radiometer (with 1 or 2 beams) measure the path delay averaged over a 2D Gaussian footprint with standard deviation \( \sigma_0 \) (in km). \( \sigma_0 \) is set at 8-km by default (corresponding to an overall 20-km diameter beam, close to the characteristic of the AMR radiometer on Jason-2), but can be modified by the user since the beam characteristics are not yet fixed by the project team. An additional radiometer instrument error is considered, given by the following characteristics (in cm²/(cy/km), see Esteban-Fernandez et al., 2014):

\[
S_{\text{wd, instr}} = 9.510^{-5} f^{-1.70} \quad \text{for} \quad 10^{-3} \leq f < 0.0023 \\
S_{\text{wd, instr}} = 0.036f^{-0.814} \quad \text{for} \quad 0.0023 \leq f < 0.0683 \\
S_{\text{wd, instr}} = 0.32 \quad \text{for} \quad f \geq 0.0683
\]

The high frequencies of instrument error (below 25km wavelength) have been filtered in the simulator. Indeed, this high-frequency signal can be easily removed since it exceeds significantly the spectral characteristics of a water vapor spectrum averaged over a 25~km diameter beam. The scheme on Fig. 16 shows how the residual error with a 1-beam or 2-beam radiometer is calculated. In the 1-beam case, the single beam measurement around the nadir plus a random realization of the radiometer instrument error is the estimate applied across the swath. In the 2-beam case, the estimation across the swath is a linear fit between the two measurements. Fig. 17 shows an example of residual error after a 1-beam and a 2-beam correction.
Fig. 2.16: FIG. 16: Scheme showing the simulation of the path delay estimation and the residual error for a 1-beam (left) and 2-beam (right) radiometer configuration.

Fig. 2.17: FIG. 17: (a) Residual error after wet-tropospheric correction with the simulation of a 2-beam radiometer at 35-km away from nadir, from the simulated path delay on Fig. 15. (b) Residual error with the simulation of a 1-beam radiometer at nadir.
Sea state bias

The Sea State Bias (or Electromagnetic bias) and its estimation are not implemented in the software yet. In later versions, inputs of SWH and surface wind speed may be proposed to simulate SSB and the residual error after SSB estimation.

Other geophysical errors

The other geophysical errors (Dry-troposphere, Ionosphere) are not implemented in the software since they have a minor impact on the mesoscales to be observed by SWOT.
2.3.3 Total error budget

The along-track power spectra of the different error components have been computed to check the consistency with the baseline requirements. These spectra, averaged across-track between 10–km and 60–km off nadir, are represented on Fig. 18. The total error budget with a 1-beam radiometer (thick black curve) is indeed slightly below the error requirement (thick red curve).

Note that the along-track power spectrum of the KaRIN noise (dark pink thick curve) sampled on a 2–km by 2–km grid is about $6 \text{ cm}^2/(\text{km}/\text{cy})$, which exceeds the requirements for short wavelength. However, these requirements have been defined for wavelength exceeding 15–km in the plan (2 dimensions). Sampled at the Niquist frequency in the across swath direction (7.5–km), the noise drops down to $2 \text{ cm}^2/(\text{km}/\text{cy})$ (thick dark pink curve).

![Fig. 2.18: FIG. 18: Error budget in the spectral domain, computed from a random realization of the simulator. The spectral densities have been averaged across-swath between 10–km and 60–km off nadir, consistently with the definition of the requirements.](image-url)
2.4 Simulation of errors for the nadir altimeter

Two main components of the nadir altimetry error budget are simulated: the altimeter noise and residual wet-tropo errors. For the altimeter noise, the noise follow a spectrum of error consistent with global estimates from the Jason-2 altimeter. The wet tropo residual errors are generated using the simulated wet tropo signal and radiometer beam convolution described above for the KaRIN data.
2.5 The software

The software is written in Python, and uses Numpy and Scipy python libraries. All the parameters that can be modified by the user are read in a params file (e.g. params.txt) specified by the user. These parameters are written in yellow later on and are linked to their location in the params file example.

The software is divided in 6 modules:

- `run_simulator.py` is the main program that runs the simulator.
- `build_swath.py` generates the SWOT swath and save several swath variables in a netcdf file.
- `build_error.py` generates all the errors on the swath.
- `rw_data.py` contains all the classes to read and write model and SWOT swath data (in netcdf).
- `mod_tools.py` contains miscellaneous functions (algebraic functions and generation of random coefficients).
- `run_nadir.py` enables the user to run the nadir alone.

2.5.1 Inputs

The inputs are SSH model outputs in netcdf (to read netcdf4 files, the python module netCDF4 is needed). Each time step is stored in one file so that there are as many files as time steps. A list of file (in .txt format) is read by the software. It contains the grid file and all SSH model outputs. The first file in this list is the grid, therefore, if the grid is contained in the SSH files, the first SSH file should be repeated. The directory that contains input (`indatadir`) and the name of the list of files (`file_input`) are specified in the params file.

It is possible to generate the noise alone, without using any SSH model as an input. To generate the noise alone, the name of the list of files (`file_input`) should be set to `None`.

The module `rw_data.py` is used to read model data. For any standard netcdf format, the data can be read using `model = MODEL_NETCDF`, which is the `model` default value. The user needs to specify the latitude (`lat`), longitude (`lon`), and SSH(`var`) variable names. Netcdf data that follow NEMO or ROMS format can automatically be read using `model = NEMO` or `model = ROMS` and there is no need to specify the longitude, latitude or SSH variables name. The coordinates are supposed to be in degrees and SSH variable in m in the program, if the SSH is not in m, specify the conversion factor in `SSH_factor` (so that SSH*SSH_factor is in m). The time step between two inputs (`timestep`) and the number of steps that have to be processed (`nstep`) can be modified in the params file. The value corresponding to not a number can be specified in `model_nan`.

2.5.2 Generation of the SWOT grid

The SWOT grid is generated in the `build_swath.py` module. The orbit file (`filesat`) is located in `dir_setup` and contains longitude, latitude and the corresponding time for each point of the orbit (see section Proposed SWOT orbits for more details on the provided orbit). The orbit is interpolated at the along track resolution specified by the user (in `delta_al`) and the swath is computed at each points of the orbit with an across track resolution specified by the user (in `delta_ac`). The width of the swath (`halfswath`) and the gap between the nadir and the swath (`halfgap`) can also be defined according to Fig. 2. The generation of the SWOT grid can be made on the whole region of the model (`modelbox` =None) or on a subdomain (`modelbox` = [lon_min, lon_max, lat_min, lat_max]). To compute the noise
2.5. The software

A netcdf file containing SWOT grid information is stored for each pass in the output directory (outdatadir) under the name filesgrid_p[pass].nc. It contains along track distances (x_al) from the first point of the orbit and across track distances (x_ac) from the nadir (in km), longitudes (lon) and latitudes (lat) at each point of the SWOT grid, the number of days in a cycle (cycle) and the distance crossed by the satellite in a cycle (al_cycle). Once the SWOT grid has been created, it is stored in outdatadir. As long as the domain (modelbox parameter) does not change, the grid does not have to be recomputed and makesgrid can be set to False. The nadir grid is stored separately under the name filesgrid nadir_p[pass].nc. For a more convenient use, in the example the name of the output files are a concatenation of a config name and satname <params-file> orbit name.

2.5.3 Sampled SSH and error fields

At each pass, for each cycle, an output netcdf file containing the SSH interpolated from the model (if file_input is set to None) and the different errors are created. The output file names are file_output _[cycle]_[p]_pass].nc for the swath and file_output nadir_[c]_cycle]_[p]_pass].nc for the nadir. The SSH is interpolated at the SWOT grid resolution. If the model grid is regular, option grid can bet set to regular and RectBivariateSpline interpolation from scipy is used. In all cases, grid option can be set to irregular and pyresample is used for the interpolation if the module is installed. If the grid is irregular and pyresample is not installed, griddata from scipy interpolates data using either the ‘linear’ (interpolation = ‘linear’) or ‘nearest’ neighbor (interpolation = ‘nearest’). In case of large domain, this last solution for the interpolation can be slow and even trigger memory error. The use of the ‘nearest’ interpolation is necessary to avoid memory error though the gradient of the SSH is significantly altered using this interpolation method.

To compute each error, set the corresponding parameter to True (karin, roll, baseline_dilation, phase, timing, wet_tropo, ssh). The KaRIN noise levels are read in karin_file, and are a function of the Significant Wave Height (SWH) and of the across track distance to the nadir. The SWH can be defined in swh (in m). Note that all swh greater than 8.0~m are set to 8.0~m. The KaRIN noise is randomly computed using coefficients produced on a nrandkarin km long grid (suggested value is nrandkarin =1000~km). Note that two files are actually provided for the KaRIN noise, the old one karin_noise_v1.nc for those who need consistency with older run, and the new one karin_noise_v2.nc which contains more levels of noise and the noise is defined up to 5~km from the nadir (vs 10 for the old file). By default the simulator takes the newest file to compute KaRIN noise. As explained in section Instrumental errors, the instrumental errors (roll, baseline_dilation, phase, timing) are randomly computed according to a spectrum specified in the file file_inst_err. A cut-off wavelength (lambda_cut) of 40000~km is recommended in model studies so that only errors below the cut-off wavelength are represented, however lambda_cut should be set to None for cross-calibration studies (see section 2.1). The number of random coefficients is specified in ncomp1d for 1d spectrum and (ncomp1d =2000 is recommended) and ncomp2d for 2d spectrum (mainly the wet_tropo error, ncomp2d =2000 is recommended). Having different numbers of random coefficients for 1d and 2d spectrum enables to reduce the computing cost when one needs a high number of random coefficients for the instrument error (only 1d spectrum). The computation of the wet_tropo error is costly so that on needs to keep the number of random coefficients for 2d spectrum as small as possible. Several parameters can be modified to compute the residual wet_tropo error. Following the subsection Wet troposphere errors, the residual path delay error can be computed using one beam (nbeam =1), two beams (nbeam =2), or both of these residual path delay error can be computed (nbeam = ‘both’). Each beam has a Gaussian footprint of sigma km. If there are two beams, the left beam is located at beam_pos_l km from the nadir and the right one at beam_pos_r km from the nadir. If only one beam is used, it is located at the nadir. As for the Sea State bias error (ssh), it has not been implemented yet.

All the computed errors are saved in the output netcdf file. The observed SSH (SSH_obs) is also computed by adding all the computed errors to the SSH from the model (SSH_model) when model data are provided. Note that if nbeam =’both’ (residual error due to path delay using one beam and two beams are both computed), only the residual error due to path delay using one beam is considered in the observed SSH.

Two errors are considered in the nadir. The first one is the instrument error, which follows the 1d spectrum computed from the current altimeters. You have to set (nadir) to True to compute this error. The second error is the path delay
due to the wet troposphere and this error is computed with the residual path delay error in the swath. The observed SSH (SSH_obs) is computed by adding these two errors to the SSH interpolated from the model (SSH_model).

### 2.5.4 Reproducible runs

It is possible to reproduce the exact same run by saving the random coefficients (phase phi, amplitude A and frequency fr) for every error but the KaRIN noise. Random coefficients on a nrandkarin km long and x_ac km large are saved for KaRIN noise. All these variables are saved during the first run in file_coeff. The random coefficients for the nadir are stored in a separate file file_coeff_nadir.nc. If you don’t want the runs to be reproducible, set file_coeff = None. Warning: If the size of the across track grid (x_ac) or the numbers of random coefficient (ncomp, nrandkarin) are changed, the random coefficients must be recomputed as their shape has been modified.

### 2.5.5 Simulate nadir data alone

When the simulator is run, the altimeter is simulated at the nadir but with the same along track resolution as the one in SWOT swath. It is possible to compute the altimeter data alone so that the along track resolution is independant of the simulator, or to compute Observation-like observation from any altimeter. To do so, run:

```python
>> run_nadiralone.py [your params file]
```

This run uses the module run_nadir.py, which simulates the nadir along track data at a resolution of delta_al that can be specified in the nadir params file. This parameter file is very similar to the one used for SWOT simulator. See the params_example_nadir.txt txt in the example directory for an example. The only difference between the swot simulator parameter file and the nadir alone parameter file is that a list of orbit file can be provided in filesat instead of a single file, enabling the user to simulate several altimeters.

### 2.5.6 Getting started

All information regarding the installation and running of the software are in the SWOT Simulator for Ocean Science file. An example of a params.txt file is given below. A ready to launch kit is also provided in swotsimulator/example/. It contains model files produced by the Regional Ocean Modeling System (ROMS) off the Oregon coast developed by Dr. Yi Chao and his team (stored in swotsimulator/example/input_fields/) and a params file (params_example.txt). To test the Simulator using the provided example, install python and the simulator as it is advised in the SWOT Simulator for Ocean Science file, modify dir_setup, indatadir, outdatadir with your own path in swotsimulator/example/params_example.txt. The outputs of the example are going to be saved in outdatadir = swotsimulator/example/swot_output/. To print help regarding the simulator or one of the modules, type in a python or ipython window:

```python
>>>import swotsimulator.M
>>>help(swotsimulator.M)
```

with M the name of the module.

To run the example, type in any terminal:

```bash
>> run.py ./example/params_example.txt
```

for the SWOT simulator and

```bash
>> run_nadiralone.py ./example/params_example_nadir.txt
```

to compute the nadir alone.

Note that if no params file is provided, the default one is ./example/params_example.txt.
Example of Params.txt for SWOT-like data

```
# -----------------------#
# Files and directories
# -----------------------#

# ------ Directory that contains orbit file:
dir_setup='[yourpath]/SWOT_simulator/data/'
# ------ Directory that contains your own inputs:
indatadir='[yourpath_to_yourdata]/'
# ------ Directory that contains your outputs:
outdatadir='[yourpath_to_outputs]/'
# ------ Orbit file:
satname=[chosenorbit]
filesat=dir_setup+'/'+orbit292.txt'

# ------ Name of the configuration (to build output files names)
config='[nameofyourconfig]'
# -----------------------#
# SWOT swath parameters
# -----------------------#

# ------ Satellite grid file root name:
# (Final file name is root_name_p[numberofpass].nc)
filesgrid=outdatadir+'/'+config+'-'+satname+'_grid'
or filesgrid=outdatadir+'/'+your_grid_root_name'
# ------ Force the computation of the satellite grid:
makesgrid=True or True
# ------ Give a subdomain if only part of the model is needed:
# (If modelbox is None, the whole domain of the model is considered)
modelbox=None or [yourlon_min, yourlon_max, yourlat_min, yourlat_max]
# ------ Distance between the nadir and the end of the swath (in km):
halfswath=60.
# ------ Distance between the nadir and the beginning of the swath (in km):
halfgap=10.
# ------ Along track resolution (in km):
delta_al=2.
# ------ Across track resolution (in km):
delta_ac=2.
# ------ Shift longitude of the orbit file if no pass is in the domain (in degree):
# Default value is None (no shift)
shift_lon=None
# ------ Shift time of the satellite pass (in day):
# Default value is None (no shift)
shift_time=None

# -----------------------#
# Model input parameters
# -----------------------#

# ------ List of model files:
# (The first file contains the grid and is not considered as model data)
# To generate the noise alone, file_input=None and specify region in modelbox
file_input=indatadir+'/'+[your_list_of_file_name.txt] or None
# ------ Type of model data:
# (Optional, default is NETCDF_MODEL and reads netcdf3 and netcdf4 files)
model='NETCDF_MODEL'
```

2.5. The software
grid='irregular'

---

# Type of grid:
# 'regular' or 'irregular', if 'regular' only 1d coordinates are extracted from model

grid='irregular'

---

# Specify SSH variable:
var='H'

---

# Specify factor to convert SSH values in m:
SSH_factor=1.

---

# Specify longitude variable:
lon='lon_rho'

---

# Specify latitude variable:
lat='lat_rho'

---

# Time step between two model outputs (in days):
timestep=1.

---

# Number of outputs to consider:
# (timestep*nstep=total number of days)

---

model_nan=0.

---

# Output file root name:
# (Final file name is root_name_c[cycle]_p[pass].nc
file_output=outdatadir+'/'+config+'_'+satname
or file_output=outdatadir+'/[your_output_root_name]'

---

# Interpolation of the SSH from the model (if grid is irregular and

---

interpolation='nearest' or 'linear'

---

# Random error coefficients so that runs are reproducible:
# If file_coef is specified and does not exist, file is created
# If you don't want runs to be reproducible, file_coef is set to None
file_coef=outdatadir+'/Random_coeff.nc'

---

# KarIN noise (True to compute it):
karin=True or False

---

# KarIN file containing spectrum for several SWH:
karin_file=dir_setup+'/karin_noise.nc'

---

# SWH for the region:
# if swh greater than 7 m, swh is set to 7
swh=2.0

---

# Number of km of random coefficients for KarIN noise
# (recommended nrandkarin=1000):
nrandkarin=1000

---

# Other instrument error (roll, phase, baseline dilation, timing)

---

# Compute nadir (True or False):
nadir=True

---

# File containing spectrum of instrument error:
```
file_inst_error=dir_setup+"/global_sim_instrument_error.nc"
# ------ Number of random realisations for instrumental and geophysical error:
# (recommended ncomp=2000), ncomp1d is used for 1D spectrum, and ncomp2d is used for 2D spectrum (wet troposphere computation):
ncomp1d=2000
ncomp2d=2000
# ------ Cut off frequency:
# (Use lambda_cut=40000km for cross-calibration)
lambda_cut=40000
lambda_max=lambda_cut*50
# ------ Roll error (True to compute it):
roll=True or False
# ------ Phase error (True to compute it):
phase=True or False
# ------ Baseline dilation error (True to compute it):
baseline_dilation=True or False
# ------ Timing error (True to compute it):
timing=True or False

## -- Geophysical error
""
## ----------------------
# ------ Wet tropo error (True to compute it):
wet_tropo=True or False
# ------ Beam print size (in km):
# Gaussian footprint of sigma km
sigma=8.
# ------ Number of beam used to correct wet_tropo signal (1, 2 or 'both'):
nbeam=1 or 2 or 'both'
# ------ Beam position if there are 2 beams (in km from nadir):
beam_pos_l=-35.
beam_pos_r=35.
# ------ Sea State Bias error (Not implemented yet):
ssb=False
```
# -----------------------#
# ------ Satellite grid file root name:
# (Final file name is root_name_p[numberofpass].nc)
filesgrid=outdatadir+'/'+config+'_'+satname+'_grid'
or filesgrid=outdatadir+'/'+[your_grid_root_name]' 
# ------ Force the computation of the satellite grid:
makesgrid=True or False
# ------ Give a subdomain if only part of the model is needed:
# (modelbox=[lon_min, lon_max, lat_min, lat_max])
# (If modelbox is None, the whole domain of the model is considered)
modelbox=None or [yourlon_min, yourlon_max, yourlat_min, yourlat_max]
# ------ Distance between the nadir and the end of the swath (in km):
# default 60.
# ------ Distance between the nadir and the beginning of the swath (in km):
# default 10.
# ------ Along track resolution (in km):
delta_al=6.
# ------ Shift longitude of the orbit file if no pass is in the domain (in degree):
# Default value is None (no shift)
shift_lon=None
# ------ Shift time of the satellite pass (in day):
# Default value is None (no shift)
# -----------------------#

# Model input parameters
# -----------------------#
# ------ List of model files:
# (The first file contains the grid and is not considered as model data)
# To generate the noise alone, file_input=None and specify region in modelbox
file_input=indatadir+'/'+[your_list_of_file_name.txt]' or None
# ------ Type of model data:
# (Optional, default is NETCDF_MODEL and reads netcdf3 and netcdf4 files)
model='NETCDF_MODEL'
# ------ Type of grid:
# 'regular' or 'irregular', if 'regular' only 1d coordinates are extracted from model
grid='irregular'
# ------ Specify SSH variable:
var='H'
# ------ Specify factor to convert SSH values in m:
SSH_factor=1.
# ------ Specify longitude variable:
lon='lon_rho'
# ------ Specify latitude variable:
lat='lat_rho'
# ------ Time step between two model outputs (in days):
timestep=1.
# ------ Number of outputs to consider:
model_nan=0.
# -----------------------#
#------ Output file root name:
# (Final file name is root_name_c[cycle]_p[pass].nc
file_output=outdatadir+'/'+config+'_'+satname
or file_output=outdatadir+'/'+[your_output_root_name]'

# pyresample is not installed:
# (either 'linear' or 'nearest', use 'nearest' for large region
# as it is faster and use less memory)
interpolation='nearest' or 'linear'

#----------------------#
# NADIR error parameters
#----------------------#
#------ File containing random coefficients to compute and save
# random error coefficients so that runs are reproducible:
# If file_coeff is specified and does not exist, file is created
# If you don't want runs to be reproducible, file_coeff is set to None
file_coeff=outdatadir+'/Random_coeff.nc'
#------ Numbers of random realisations for instrumental and geophysical error
# recommended ncomp=2000), ncomp1d is used for 1D spectrum, and ncomp2d is used for
# 2D spectrum (wet troposphere computation):
#ncomp1d=3000
#ncomp2d=2000

##-- Geophysical error
##----------------------
##------ Wet tropo error (True to compute it):
wet_tropo=True or False
##------ Beam print size (in km):
#sigma=8.
##------ Across track resolution of the beam for the correction of the wet tropo (in
#delta_ac=6.

### References:


EVOLUTION OF THE SIMULATOR

3.1 Changelog for swotsimulator

This file lists the changes in each swotsimulator version

3.1.1 2.3

12/19/2016 - Updating Orbit File with those from http://www.aviso.altimetry.fr/en/missions/future-missions/swot/orbit.html. Two new files have been gitted: ephem_science_sept2015_ell.txt for the science orbit and ephem_calval_june2015_ell. for the 1-day (for calval purpose) orbit.

3.1.2 2.2.1

08/20/2016 - Bug fixed:

-> Spatial lag (params key ... ) was not implemented properly

3.1.3 2.2

09/25/2016 - Upgrade of the code to be pep8 compliant.

09/23/2016 - Adding new variables and global attribute to improve CF1.6 compliancy of netcdf outputs from the simulator

08/20/2016 - Minor modification:

-> Karin noise is updated with the version provided by th SWOT technical team, swath is now from 5km to 60 from Nadir and SWH can go up to 8~m.

07/29/2016 - Add warnings into rw_data to user-proof the code.

3.1.4 2.1

06/15/2016 Version available on github
3.1.5 2.0.0

04/15/2016

• Major modifications:
  -> The code is now compatible with Python3
  -> There are now 2 numbers to define random realisations in the parameter file: ncomp1d for the number of 1d realisations (all the instrument errors) and ncomp2d for the number of 2d realisations (wet-tropospheric error). This enable the user to increase the number of 1d realisations for longer wavelengths without increasing the computing time for the computation of the wet-tropospheric error.
  -> Nadir data are saved in a separate file for the grid and for the observation-like outputs
  -> Passes are now splitted in ascending (odd numbers) and descending (even number) passes.
  -> Netcdf format of outputs is improved
  -> Interpolations of the model on SWOT swath have evolved so that the simulator can run faster: If the model grid is regular (options grid in parameter file), spline interpolation can be used. Regular or irregular grid can be interpolated using pyresample if it is installed or griddata (from scipy.interpolate). Note that griddata can be very slow or even run out of memory for large number of data.

• Minor modifications:
  -> Main scripts run_simulator.py and run_nadir.py have been rewritten to make the code more flexible and easily readable.
  -> Improve printing of error messages and detection of potential errors
  -> The nadir instrument errors were a random noise. It has been modified so that the nadir instrument errors are now following the spectrum of errors of current nadir altimeters.
  -> Orbit interpolated at high resolution if the provided orbit has a low temporal resolution (more than 0.5s)

3.1.6 1.0.0

10/31/2014

• Major modifications:
  -> Implementation of SWH (Sea Wave Height) dependant KaRIN noise: New file karin_noise.nc added to data. The value of SWH is specified in the params file and is constant in time and space.
  -> Save normalized KaRIN noise every nrandkarin km, and random coefficient for other noises in file_coeff specified in params so that runs are reproducible (random coefficients are loaded from that file when it exists).
  -> Set up an option to compute the noise alone if no model file is provided (set ‘file_input’ to None in params)
  -> Karin noise is now define on a swath from 10km to 60 km from the nadir. The parameter halfgap should be change to 10km in the params.txt file
  -> Compute data at the nadir using a noise similar to the altimeter ... WARNING: the along track sampling is the same as the one in SWOT data.
  -> Possibility to compute nadir-like data for any altimeter using run_nadir.py. It runs exactly like run.py. An example of the parameter file is in example/params_example_nad.txt, the along track sampling can be set in the params file (recommended value is 6km).
• Minor modifications:
  -> Number of random frequencies is now coded as a parameter and can be modified in params.
  -> Change name of variable indice by index in output file.
  -> renamed the module to swotsimulator, according to Python’s PEP standard naming convention
  -> Reading Netcdf4 files is now possible if netCDF4 python module is installed

• Fixed bugs:  -> problem around 0deg longitude due to the discontinuity 0-360 is fixed
  -> Last pass missing: fixed

3.1.7 1.0 beta 1

09/01/2014"
4.1 Main program: run_simulator

Main program: Usage: run_simulator(file_param)

If no param file is specified, the default one is exemple/params_exemple.txt

In the first part of the program, model coordinates are read and the SWOT swath is computing accordingly.

The SWOT grid parameters are saved in netcdf files, if you don’t want to recompute them, set maksgrid (in params file) to False.

In the second part of the program, errors are computed on SWOT grid for each pass, for each cycle. The error free SSH is the SSH interpolated from the model at each timestep. Note that there is no temporal interpolation between model files and thus if several files are used in the SSH interpolation, some discontinuities may be visible.

OUTPUTS are netcdf files containing the requested errors, the error free SSH and the SSH with errors. There is one file every pass and every cycle.

#———————————————————————– # Additional Documentation # Authors: Lucile Gaultier and Clement Ubelmann # # Modification History: # - Jul 2014: Original by Clement Ubelmann and Lucile Gaultier, JPL # - Nov 2014: Beta version # - Feb 2015: Version 1.0 # - Dec 2015: Version 2.0 # # Notes: # - Written for Python 2.3, Python 3.4, tested with Python 2.7, Python 3.4 # # Copyright (c) 2002-2014, California Institute of Technology. # All rights reserved. Based on Government Sponsored Research under # contracts NAS7-1407 and/or NAS7-03001. # #———————————————————————–

swotsimulator.run_simulator.create_SWOTlikedata(cycle, ntotfile, list_file, modelbox, sgrid, ngrid, model_data, modeltime, err, errnad, p, progress_bar=True)

Create SWOT and nadir errors err and errnad, interpolate model SSH model_data on swath and nadir track, compute SWOT-like and nadir-like data for cycle, SWOT grid sgrid and ngrid.

swotsimulator.run_simulator.load_error(p)

Initialize random coefficients that are used to compute random errors following the specified spectrum.

If a random coefficient file is specified, random coefficients are loaded from this file.

swotsimulator.run_simulator.load_sgrid(sgridfile, p)

Load SWOT swath and Nadir data for file sgridfile

4.2 Read data: rw_data

Module to read and write data Contains functions to read variables and coordinates from a netcdf files.

Contains model classes:
-ROMS
-NEMO
-NETCDF_MODEL

Contains satellite class: Sat_SWOT
Contains file instrumentation class: file_instr

class swotsimulator.rw_data.CLS_MODEL(file=None, var='H', lon='lon_rho', lat='lat_rho',
depth=0, time=0)

Class to read CLS data model type.

USAGE is NETCDF_MODEL(file=name of file ,var= variable name, lon=variable longitude, lat=variable latitude, units=).

Argument file is mandatory, arguments var, lon, lat are specified in params file.

calc_box()
    Calculate subdomain coordinates from netcdf file Return minimum, maximum longitude and minimum, maximum latitude

read_coordinates(index=None)
    Read coordinates from netcdf file
    Argument is index=index to load part of the variable.

read_var(index=None)
    Read variables from netcdf file
    Argument is index=index to load part of the variable.

class swotsimulator.rw_data.MITgcm(file=None, var='Eta', lon='XC', lat='YC', depth=0, time=0)

Class to read MITgcm binary data.

USAGE is MITgcm(file=name of file ,var= variable name, lon=variable longitude, lat=variable latitude, units=).

Argument file is mandatory, arguments var, lon, lat are specified in params file.

calc_box()
    Calculate subdomain coordinates from netcdf file Return minimum, maximum longitude and minimum, maximum latitude

read_coordinates(index=None)
    Read MITgcm output coordinates saved in XC.data and YC.data
    Argument is index=index to load part of the variable.

read_var(index=None)
    Read variables from binary file
    Argument is index=index to load part of the variable.

class swotsimulator.rw_data.NEMO(file=None, var='sossheig', lon='nav_lon', lat='nav_lat',
time='time', depth='depth')

Class to read NEMO data

USAGE is NEMO(file=name of file ,var= variable name, lon=longitude name, lat=latitude name, depth= depth name, time=time name).

Argument file is mandatory, other arguments have default values var='sossheig', lon='nav_lon', lat='nav_lat', depth='depth', time='time.
**calc_box()**
Calculate subdomain coordinates from NEMO file Return minimum, maximum longitude and minimum, maximum latitude

**read_coordinates(index=None)**
Read coordinates from NEMO file
Argument is index=index to load part of the variable.

**read_var(index=None)**
Read variables from NEMO file
Argument is index=index to load part of the variable.

**class swotsimulator.rw_data.NETCDF_MODEL(file=None, var='H', lon='lon_rho', lat='lat_rho', depth=0, time=0)**
Class to read any netcdf data.

_USAGE is NETCDF_MODEL(file=name of file, var=variable name, lon=variable longitude, lat=variable latitude, units=).

Argument file is mandatory, arguments var, lon, lat are specified in params file.

**calc_box()**
Calculate subdomain coordinates from netcdf file Return minimum, maximum longitude and minimum, maximum latitude

**read_coordinates(index=None)**
Read coordinates from netcdf file
Argument is index=index to load part of the variable.

**read_var(index=None)**
Read variables from netcdf file
Argument is index=index to load part of the variable.

**class swotsimulator.rw_data.ROMS(file=None, var='rho', depth='depth', time='time', lon='x_rho', lat='y_rho')**
Class to read ROMS data

_USAGE is ROMS(file=name of file, var=variable name, lon=longitude name, lat=latitude name, depth=depth name, time=time name).

Argument file is mandatory, other arguments have default values var='rho', lon='x_rho', lat='y_rho', depth='depth', time='time'.

Variable units is specified in params file and default value is True (coordinates in degree).
If units is False (coordinates in km), specify left low corner of the domain (lon0, lat0) in params file.

**calc_box()**
Calculate subdomain coordinates from ROMS file Return minimum, maximum longitude and minimum, maximum latitude

**read_coordinates(index=None)**
Read coordinates from ROMS file
Argument is index=index to load part of the variable.

**read_var(index=None)**
Read variables from ROMS file
Argument is index=index to load part of the variable.
class swotsimulator.rw_data.Sat_SWOT(file=None, lon=None, lat=None, lon_nadir=None, lat_nadir=None, time=None, cycle=None, al_cycle=None, x_al=None, x_ac=None, timeshift=None)

Sat_SWOT class: to read and write data that has been created by SWOT simulator

load_swath(**kwargs)
Load swath variables stored in Satellite grid file sgridfile.
(longitude, latitude, number of days in a cycle, crossed distance during a cycle, time, along track and across track position).

write_data(**kwargs)
Write SWOT data in output file file_output Dimensions are x_al (along track distance), x_ac (across track distance).
Variables are longitude, latitude, index (file number), error-free SSH (SSH interpolated from the model), selected errors (karin, wet_tropo, roll, baseline_dilation, phase, timing) and SSH with errors.

write_swath(**kwargs)
Write swath location in Satellite grid file sgridfile.
Dimensions are time (i.e. along track), x_ac (across track) and cycle (1).
Variables are longitude, latitude, number of days in a cycle, distance crossed in a cycle, time, along track and across track distances are stored.

class swotsimulator.rw_data.file_instr(file=None)
Class to open file containing instrumentation errors.

USAGE: file_instr(file=file name)
Mandatory argument is the file name.

read_var(**kwargs)
Read variables in instrumentation file.
Possible arguments are the variable names in the instrument file.

class swotsimulator.rw_data.file_karin(file=None)
Class to open file containing SWH and corresponding karin noise.

USAGE: file_karin(file=file name)
Mandatory argument is the file name.

read_karin(swh)
Read and interpolate karin noise.
Possible arguments is SWH.

swotsimulator.rw_data.read_coordinates(ifile, nlon, nlat, twoD=True)
General routine to read coordinates in a netcdf file.
Inputs are file name, longitude name, latitude name.
Outputs are longitudes and latitudes (2D arrays).

swotsimulator.rw_data.read_params(params_file)
Read parameters from parameters file and store it in p.
This program is not used in swot_simulator.

swotsimulator.rw_data.read_var(ifile, var, index=None, time=0, depth=0, model_nan=None)
General routine to read variables in a netcdf file.
Inputs are file name, variable name, index=index to read part of the variables, time=time to read a specific time, depth=depth to read a specific depth, model_nan=nan value

\texttt{swotsimulator.rw_data.write_params(params, pfile)}

Write parameters that have been selected to run \texttt{swot_simulator}.

### 4.3 Build swath: build\_swath

\texttt{swotsimulator.build\_swath.makeorbit(modelbox, p, orbitfile='orbit\_292.txt', filealtimeter=None)}

Computes the orbit nadir on a subdomain. The path of the satellite is given by the orbit file and the subdomain corresponds to the one in the model. Note that a subdomain can be manually added in the parameters file.

Inputs are satellite orbit (p.filesat), subdomain (modelbox), Along track sampling, along track resolution).

Outputs are Sat\_Nadir object containing Nadir track (along track distance x\_al, longitude lon and latitude lat, number of days in a cycle cycle, distance crossed in a cycle cycle\_al, time, time shift and time of pass passtime

\texttt{swotsimulator.build\_swath.orbit2swath(modelbox, p, orb)}

Computes the swath of SWOT satellites on a subdomain from an orbit. The path of the satellite is given by the orbit file and the subdomain corresponds to the one in the model. Note that a subdomain can be manually added in the parameters file.

Inputs are satellite orbit (p.filesat), subdomain (modelbox), Swath parameters (half gap distance p.halfgap, half swath distance p.halfswath, along track resolution p.delta\_al, across track resolution p.delta\_ac).

Outputs are netcdf files containing SWOT grid (along track distance x\_al, across track distance from nadir x\_ac, longitude lon and latitude lat, number of days in a cycle cycle, distance crossed in a cycle cycle\_al, time

### 4.4 Build errors: build\_error

\texttt{class swotsimulator.build\_error.error(p, roll=None, ssb=None, wet_tropo=None, phase=None, baseline_dilation=None, karin=None, timing=None, SSH=None, wt=None)}

Class error define all the possible errors that can be computed using SWOT simulator. Random realisation of errors can be initialized using \texttt{init\_error}. If the random realisations have already been computed and stored in file file\_coeff, the random realisations are read directly using \texttt{load\_coeff}. The corresponding errors on a swath can be computed using \texttt{make\_error}.

\texttt{init\_error(p, nac)}

Initialization of errors: Random realisation of errors are computed using a known power spectrum. The outputs are the amplitude, the phase and the frequency of each random realisation. By default, there are ncomp1d=2000 random realisations for the instrumental errors (1d spectrum) and ncomp2d=2000 random realisations for the geophysical errors (2d spectrum) and nrandkarin*x\_ac km of random number for KaRIN noise.

\texttt{load\_coeff(p)}

Load existing random realisations that has been stored in file\_coeff. The outputs are the amplitude, the phase and the frequency of each random realisation. There are ncomp1d random realisations for instrumental and ncomp2d for geophysical errors and nrandkarin*x\_ac random number for KaRin noise.

\texttt{make\_SSH\_error(SSH\_true, p)}

Compute observed SSH adding all the computed error to the model SSH. If residual path delay errors after 2-beams and 1-beam radiometer correction are both computed (nbeam=’both’), only the path delay error after 1-beam radiometer correction is considered in the observed SSH.

4.3. Build swath: build\_swath
**make_error** \((\text{sgrid}, \text{cycle}, \text{SSH\_true}, p)\)

Build errors corresponding to each selected noise among the effect of the wet_tropo, the phase between the two signals, the timing error, the roll of the satellite, the sea surface bias, the distortion of the mast, the karin noise due to the sensor itself.

**save_coeff** \((p, nac)\)

Save random realisations to enable runs to be reproducible. The ncomp1d random phase phi, amplitude A and frequency fr in 1D and ncomp2d random phase phi, amplitude A and frequencies frx and fry in 2D are saved in file_coeff for each error (except KaRIN) and can be loaded using load_coeff. Random numbers on a grid rrandkarin km long and x_ac large are stored for KaRIN noise.

**class** swotsimulator.build_error.errornadir \((p, \text{nadir=None, wet\_tropo1=None, wt=None})\)

Class errornadir defines the error on the nadir. Random realisation of errors can be initialized using init_error. If the random realisations have already been computed and stored in file file_coeff, the random realisations are read directly using load_coeff. The corresponding errors on a swath can be computed using make_error.

**init_error** \((p)\)

Initialization of errors: Random realisation of errors are computed using a known power spectrum. The outputs are the amplitude, the phase and the frequency of each random realisation. By default, there are ncomp2d=2000 random realisations for the wet tropo and ncomp1d=2000 random realisations for the nadir 1d spectrum error.

**load_coeff** \((p)\)

Load existing random realisations that has been stored in nadir+file_coeff. The outputs are the amplitude, the phase and the frequency of each random realisation. There are ncomp random realisations.

**save_coeff** \((p)\)

Save random realisations to enable runs to be reproducible. The ncomp1d random phase phi, amplitude A and frequency fr for 1D spectrum and ncomp2d random phase phi, amplitude A and frequencies frx and fry for 2D spectrum are saved in nadirfile_coeff for each error and can be loaded using load_coeff.

### 4.5 Useful tools: mod_tools

Spectral and algebra tools for swot simulator.

Contains the following functions:

- gen_signal1d: compute 1d random error from spectrum
- gen_signal2d: compute 2d random error from spectrum
- rotationmat3d: rotate data
- spher2cart: convert spherical to cartesian coordinates
- cart2spher: convert cartesian to spherical coordinates
- update_progress: Progress bar

**swotsimulator.mod_tools.cart2spher** \((x, y, z)\)

Convert cartesian coordinates to spherical coordinates.

Inputs are cartesian coordinates \(x, y, z\).

Return lon, lat.

**swotsimulator.mod_tools.gen_coeff_signal1d** \((f, PS, nc)\)

Generate nc random coefficient from a spectrum PS with frequencies f.

Return Amplitude, phase and frequency of nc realisations
swotsimulator.mod_tools.gen_coeff_signal2d \((f, PS, nc)\)
Generate nc random coefficient from a spectrum PS with frequencies \(f\).
Inputs are: frequency \([f]\), spectrum \([PS]\), number of realisation \([nc]\) Return Amplitude, phase and frequency in 2D \((frx, fry)\) of nc realisations

swotsimulator.mod_tools.rotationmat3D \((\theta, axis)\)
Creates a rotation matrix: Slow method.
Inputs are rotation angle \(\theta\) and rotation axis \(axis\). The rotation matrix correspond to a rotation of angle \(\theta\) with respect to axis \(axis\).
Return the rotation matrix.

swotsimulator.mod_tools.spher2cart \((lon, lat)\)
Convert spherical coordinates to cartesian coordinates.
Inputs are longitude, latitude.
Return x, y, z

swotsimulator.mod_tools.update_progress \((progress, arg1, arg2)\)
Creation of a progress bar: print on screen the progress of the run
CHAPTER FIVE

OPEN SOURCE LICENCE

This is the open source licence regarding the Open Source SWOT Simulator the Ocean.

5.1 Software licence and copyright

Copyright (c) 2002-2014, California Institute of Technology. All rights reserved. Based on Government Sponsored Research under contracts NAS7-1407 and/or NAS7-03001.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the California Institute of Technology (Caltech), its operating division the Jet Propulsion Laboratory (JPL), the National Aeronautics and Space Administration (NASA), nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE CALIFORNIA INSTITUTE OF TECHNOLOGY BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

5.2 Model data copyright

The SSH data were produced by the Regional Ocean Modeling System (ROMS) off the Oregon coast developed by Dr. Yi Chao and his team.
INDICES AND TABLES

- genindex
- modindex
- search
S
swotsimulator.build_error, 33
swotsimulator.build_swath, 33
swotsimulator.mod_tools, 34
swotsimulator.run_simulator, 29
swotsimulator.rw_data, 29
INDEX

C
calc_box() (swotsimulator.rw_data.CLS_MODEL method), 30

calc_box() (swotsimulator.rw_data.MITgcm method), 30

calc_box() (swotsimulator.rw_data.NEMO method), 30

calc_box() (swotsimulator.rw_data.NETCDF_MODEL method), 31

calc_box() (swotsimulator.rw_data.ROMS method), 31

cart2spher() (in module swotsimulator.mod_tools), 34

CLS_MODEL (class in swotsimulator.rw_data), 30

create_SWOTlikedata() (in module swotsimulator.run_simulator), 29

E
error (class in swotsimulator.build_error), 33

ererrornadir (class in swotsimulator.build_error), 34

F
file_instr (class in swotsimulator.rw_data), 32

file_karin (class in swotsimulator.rw_data), 32

G
gen_coeff_signal1d() (in module swotsimulator.mod_tools), 34

gen_coeff_signal2d() (in module swotsimulator.mod_tools), 34

I
init_error() (swotsimulator.build_error.error method), 33

init_error() (swotsimulator.build_error.errornadir method), 34

L
load_coeff() (swotsimulator.build_error.error method), 33

load_coeff() (swotsimulator.build_error.errornadir method), 34

load_error() (in module swotsimulator.run_simulator), 29

load_sgrid() (in module swotsimulator.run_simulator), 29

load_swath() (swotsimulator.rw_data.Sat_SWOT method), 32

M
make_error() (swotsimulator.build_error.error method), 33

make_SSH_error() (swotsimulator.build_error.error method), 33

makeorbit() (in module swotsimulator.build_swath), 33

MITgcm (class in swotsimulator.rw_data), 30

N
NEMO (class in swotsimulator.rw_data), 30

NETCDF_MODEL (class in swotsimulator.rw_data), 31

R
read_coordinates() (in module swotsimulator.rw_data), 32

read_coordinates() (swotsimulator.rw_data.CLS_MODEL method), 30

read_coordinates() (swotsimulator.rw_data.MITgcm method), 30

read_coordinates() (swotsimulator.rw_data.NEMO method), 31

read_coordinates() (swotsimulator.rw_data.NETCDF_MODEL method), 31

read_coordinates() (swotsimulator.rw_data.ROMS method), 31

read_karin() (swotsimulator.rw_data.file_karin method), 32

read_params() (in module swotsimulator.rw_data), 32

read_var() (in module swotsimulator.rw_data), 32

read_var() (swotsimulator.rw_data.CLS_MODEL method), 30

read_var() (swotsimulator.rw_data.file_instr method), 32

read_var() (swotsimulator.rw_data.MITgcm method), 30

read_var() (swotsimulator.rw_data.NEMO method), 31

read_var() (swotsimulator.rw_data.NETCDF_MODEL method), 31

read_var() (swotsimulator.rw_data.ROMS method), 31

ROMS (class in swotsimulator.rw_data), 31

rotationmat3D() (in module swotsimulator.mod_tools), 34

43
S
Sat_SWOT (class in swotsimulator.rw_data), 31
save_coeff() (swotsimulator.build_error.error method), 33
save_coeff() (swotsimulator.build_error.errornadir method), 34
spher2cart() (in module swotsimulator.mod_tools), 35
swotsimulator.build_error (module), 33
swotsimulator.build_swath (module), 33
swotsimulator.mod_tools (module), 34
swotsimulator.run_simulator (module), 29
swotsimulator.rw_data (module), 29

U
update_progress() (in module swotsimulator.mod_tools), 35

W
write_data() (swotsimulator.rw_data.Sat_SWOT method), 32
write_params() (in module swotsimulator.rw_data), 33
write_swath() (swotsimulator.rw_data.Sat_SWOT method), 32