alphaBetaLab: software architecture

**Lorenzo Mentaschi**

**European Commission, Joint Research Centre (JRC), Directorate for Space, Security and Migration, via Fermi 2749, 21027 Ispra (VA), Italy**

**Contact email:** *[lorenzo.mentaschi@ec.europa.eu]*

For an explanation of the science that led to the development of alphaBetaLab, the reader is referred to (Mentaschi et al., 2015, 2018a). For a short overview on the fluxes and the architecture the reader is referred to (Mentaschi et al., 2018b). Here the software architecture is explored in more detail, better explaining the advantages, the limits and the possible future improvements of this software.

The library alphaBetaLab is developed in python2.7, and depends on the following python packages: shapely, basemap, netcdf4, proj4. The instructions to obtain the code and install the library can be found in sections 1.1.1 and 1.1.2. Below a quick overview of the main modules/classes/algorithms of the library.

## Model mesh definition

The UOST/alphaBetaLab approach relies on the hypothesis that any mesh can be considered as a set of polygons, called cells, and that the model estimates the average value of the unknowns in each cell. In alphaBetaLab the generic mesh type is defined by in the module abGrid. For the definition of the polygons, and related logics, alphaBetaLab relies on the library shapely (<https://pypi.python.org/pypi/Shapely>).

### abGrid objects

The class \_abGrid (in module abGrid) represents a grid as a collection of polygons, each representing a cell. The grid can be unstructured. The constructor takes as argument: cells: a list of n cells as shapely polygon objects, and a list of n tuples of integer indices (cellCoordinates), of the same length as cells, where each tuple represents the matrix indices of each cell in the model. The class is private, and the instances can be generated using the functions getLandSeaGrid, that builds a \_abGrid instance with all the polygons contained in the list cells, and the function getSeaGrid, that filters out the polygons on land and along the resolved coasts. From the point of view of the user, the difference between these two functions is that the second takes as further arguments a high-resolution matrix of alpha (see section 2.2) and a coastalCellDetector object (see section 2.6.2). The class \_abGrid implements logics to find, for each cell, the neighboring cells (methods buildNeighCache and getNeighbors).

It is worth mentioning that the sole cells containing unresolved obstacles are relevant to the algorithms implemented in alphaBetaLab, and although the presence of empty cell-polygons in the abGrid object does not change the output, filtering them out will speed up considerably the computational time.

### Utilities to build abGrid objects for known types of mesh

abGrid builders are components to build abGrid objects from model meshes.

**abRectangularGridBuilder:** class for building a abGrid instance for lon-lat grids. The constructor takes as an input the coordinates of the grid and the grid paces, a mask defining the land points, and high-resolution matrix of alpha (see section 2.2). In the buildGrid, the cell-polygons are instantiated, and the abGrid object is created. For the sake of reusability, this class does not load directly the grid files of ww3. This is done instead by the regularGridSpecWW3 method, in module abEstimateAndSave (section 2.8). Moreover, this method is not meant to be invoked directly by the final user, but handled by function abEstimateAndSaveRegularEtopo1 in module abEstimateAndSave (section 2.8).

**abTriangularMeshGridBuilder:** class for building a abGrid instance for a triangular mesh (Roland, 2008). For the sake of its re-usability, this class does not load directly gmesh files, but instead rely on class \_abTriMeshSpec, in module abTriangularMesh, that defines the logical structure of a generic triangular mesh. Currently abTriangularMesh supports loading from gmesh format (msh files) used in WW3 and form the SCHISM-specific format Gr3. Moreover, this class is not meant to be handled directly by the final user, but is used in function abEstimateAndSaveTriangularEtopo1 in module abEstimateAndSave (section 2.8).

**abSMCGridBuilder:** to-be-implemented class for building a abGrid instance for a Spherical Multi-Cell (SMC) mesh (Li, 2011, 2012). For the sake of its re-usability, this class should not load directly gmesh files, but instead rely on some half-way class defining the logical structure of a SMC mesh. Moreover, it should not be called directly by the user, but handled by proper methods in module abEstimateAndSave (section 2.8).

## High resolution matrix of alpha

To be able to understand what unresolved obstacles/features are placed in each cell-polygon, and to estimate α and β, alphaBetLab needs a high-resolution definition of the obstacles/features. This definition is provided through the class abHighResAlphaMatrix, in terms of a high resolution regular matrix of transparency coefficients. The elements of the matrix can assume value between 1 (no obstruction) and 0 (land cell). At high resolution, the transparency coefficients are assumed to be independent from the direction. The constructor of abHighResAlphaMatrix takes as arguments a 2d matrix of alphas, and lists of x and y coordinates. In principle, this class also supports transparency coefficients dependent of frequency (the matrix would be in this case a 3d one). This option could be useful to model unresolved frequency-dependent energy dissipation, like for ice floes. However, this option is currently unsupported by other classes. Method getAlphaSubMatrix of this class returns an instance of abHighResAlphaMatrix corresponding to the minimum matrix covering the polygon passed as argument. This method is invoked within alphaBetaLab, when the α/β coefficients are evaluated for each polygon.

Instances of abHighResAlphaMatrix based on etopo1 can be built using the function loadBathy, in module abEtopo1BathyLoader. In these instances, the transparency coefficient has value 1 for sea points, and 0 for land points.

## Upstream polygon

To be able to estimate the Shadow Effect (SE) component of UOST, an upstream polygon needs to be computed for each cell-polygon and spectral component. The upstream polygon of a cell A is defined as the intersection between the joint cells neighboring A and the flux of the spectral component upstream of A. In alphaBetaLab it is computed within the module abUpstreamPolyEstimator, by function getUpstreamPoly. This function takes as arguments a cell-polygon, a neighborhood polygon and a direction. The neighborhood polygon is estimated in alphaBetaLab outside of this function, and is given by the union of the neighbor polygons, that can be retrieved with the function \_abGrid.getNeighbors.

## Estimation of α and β in a single polygon

### Class abSingleCellAlphaEstimator

For each single polygon (that can be a cell or an upstream polygon), the value of the directional α transparency coefficients are computed by abSingleCellAlphaEstimator objects. The constructor of this class takes as an input a shapely polygon and a abHighResAlphaMatrix instance.

The method computeAlpha computes the α transparency coefficient for a given spectral component, as a function of the cross section of the unresolved features inside the polygon, along the propagation direction. Inside the method, the problem is first rotated and reflected, to refer it to the first octant (class abFirstOctantTransformation in module abFirstOctantTransformation). Then the computation is done on the first octant problem (private method \_getAlphaFirstOctant of class abSingleCellAlphaEstimator).

### Class abSingleCellBetaEstimator

For each single polygon (that can be a cell or an upstream polygon), the value of the directional β transparency coefficients are computed by abSingleCellBetaEstimator objects. The constructor of this class takes as an input a shapely polygon and a abHighResAlphaMatrix instance.

The method computeBeta computes the β transparency coefficient for a given spectral component. Similar to the method computeAlpha of abSingleCellAlphaEstimator, the problem is reflected and rotated to reduce it to the first octant, then the method \_getBetaFirstOctant estimates solves the first octant problem. The algorithm implemented in \_getBetaFirstOctant follows the definition of β: estimates n polygon slices starting from the upstream side of the cell, for each of these slices estimates the α coefficient. The value of β is then given by the average of the slices’ α coefficients (Mentaschi et al., 2015, 2018a). This approach provides accurate estimations of β, but it is computationally expensive.

A future, more optimized implementation will start from the value of α, and change it as a function of the distribution of the obstacles in the polygon: if the distribution mean lies at the upstream side of the cell, then β = α. If it lies at the downstream side, then β = 1. If distribution mean lies at a certain position between the two sides, β will be increased with respect to α by the corresponding fraction.

## Cell sizes

UOST source term depends on the size of the cell-polygon along the propagation direction of the spectral component. These distances (that must be estimated in meters, and not in degrees) are computed within the class abCellSize, in the homonym module. The algorithm finds the main axes of the cell-polygon, and approximates it as an ellipse.

## Assembling altogether

### Class abCellsEstimator

Class abCellsEstimator assembles together the concepts of mesh (abGrid), upstream polygons, high resolution matrix of transparency coefficients (abHighResAlphaMatrix), spectral grid (although in the current implementation only direction is supported), estimators of α and β and sizes for the single polygons (classes abSingleCellAlphaEstimator, abSingleCellBetaEstimator and abCellSize).

This class has two blocks of methods, the first for the computation of α and β coefficients local to the cell-polygon, the second for the computation of α and β coefficients for the shadow, i.e. computed on the cell upstream polygons. These blocks of methods are organized to comply with the needs of the python multiprocessing framework, exploited in module abCellsEstimatorParallel.

The main method of the first block is computeLocalAlphaBeta. It loops the cells of the mesh, and for each direction estimates the α and β coefficients. The main method of the second block is computeShadowAlphaBeta. Similar to computeLocalAlphaBeta, it loops the cells of the mesh, but for each cell it estimates the α and β coefficients on the upstream polygon of the cell for the given directional bin.

In computeLocalAlphaBeta and computeShadowAlphaBeta, to speed up the computation, a few directional bins are considered (8 in Mentaschi et al., 2018), and the estimated α and β are subsequently interpolated to the directions of the spectral grid.

### Excluding coastal cells

In cells intersecting large land bodies a parameterization of unresolved obstacles would conflict with the resolved portion of the coastal dynamics. Coastal cells are therefore excluded from the computation of both local dissipation and shadow effect. A coastal cell can, however, be affected by the shadow projected by an obstructed, non-coastal cell. The logics for the detection of coastal cells are implemented in class abCoastalCellDetector (module abCoastalCellDetector). The algorithm treats differently bodies of different size. If a land body is more than 10 (by default) times larger than the size of a cell (the threshold can change), the body is treated as large, and if the cell intersects it for more than 10% (by default) of its surface is excluded from the computation. If a land body is between 2 (by default) and 10 times larger than a cell, it is considered as small body, and the cell is excluded from the computation if it intersects the body for more than 50% (by default) of its surface. Finally, if the land body is smaller than twice the cell, it is considered as unresolved. The application of this algorithm in (Mentaschi et al., 2018a) showed its limits, especially in proximity of small resolved bodies.

### Long breakwaters

If an unresolved breakwater intersects several consecutive cells, the approach implemented in abSingleCellAlphaEstimator and abSingleCellBetaEstimator, based on the estimation of the cross section of the obstacle in each single cell, is not enough. In such case, for example, abSingleCellAlphaEstimator would detect a total block only for spectral components normal to the breakwater, while if you consider all the cells crossed by the breakwater, the block should be total for all the spectral components propagating from one side of the breakwater to the other. Tackling this problem becomes important at high resolution.

A prototype of algorithm to deal with this problem is developed in module abLongBreakWaterLocAlphaAdjust. However, it was not applied in the case study analyzed by (Mentaschi et al., 2018a), and will require further testing before being available for use. Moreover, it is unoptimized and not parallelized.

### Parallelization

To attain better performances, alphaBetaLab is extensively parallelized using the python multiprocessing approach (<https://docs.python.org/2/library/multiprocessing.html>), that allows the dynamical allocation to worker-processes of tasks enqueued by an iterator. This kind of parallelization can be successfully employed in shared memory environments.

### Options

Each class/algorithm/component of alphaBetaLab can have its own settings and parameters. A centralized system was developed to make it possible to set these parameters in a unique place, without breaking the modular architecture of alphaBetaLab. The functions that each class uses to get its parameters are in the module abOptionManager. This module is very small and simple, and should not depend on other modules, to preserve code modularity. Whenever an alphaBetaLab object needs a parameter, it asks it to an option object through the function getOption of abOptionManager. This method is developed in such a way, that if the option is not found, alphaBetaLab will run with its default value. As a result, the object containing the parameters of alphaBetaLab can be of whatever type (not necessarily an instance of the abOptions class, in the same module), and should contain the sole options that are needed different from the default value. At the end of each run of alphaBetaLab, all the options are printed in the standard output with their actual value.

An example of use of this approach, from the point of view of the final user, is given in the sample script obstFileBuilder.py, in /mygit/alphaBetaLab/ww3/samples/obstructionCaribbeans/bathyAndObstructions/. An abOption object is instantiated at line 48 of the script, and passed to the function abEstimateAndSaveRegularEtopo1 (see section 2.8).

#### List of parameters

The majority of the parameters should not be touched by the final users. The ones that may be modified are:

* computationDirs: to reduce the amount of computation, only a subset of directions (8 by default) are considered. The user can pass customized directions.
* llcrnr, urcrnr: low-left and up-right corners of the sub-domain where *α* and *β* should be computed. By default the whole domain is considered.
* minSizeKm: for cells below this length no subscale transparency will be computed. Can be useful with unstructured meshes with a lot of very small cells.
* verbose: if active a more detailed output is print on the screen during the execution. True by default.

## Output: UOST configuration files

The output of the algorithms described in the previous sections consists of two sets of α and β coefficients, one for the local dissipation LD, the other for the shadow effect SE. It need to be mentioned that if a cell contains unresolved obstacles, but has no shadow projected on it, it will be present in the first set, but not in the second. If a cell does not contain unresolved obstacles, but some neighbor cell projects a shadow on it, it will be present in the second set but not in the first.

These coefficients should be saved to the configuration files that are input of the UOST source term in WW3. This is done by the class abWwiiiObstrFileSaver, in the homonym module. Method saveLocFile saves the coefficients for the LD component, method saveShdFile saves the coefficients for the SE component. Method saveFiles invokes in one command both the methods.

Below, a short explanation of the file format.

The character ‘$’ at the beginning of a row identifies comments.

The first non-comment line indicates the number of cells affected (by LD or SE).

Then, cell-by-cell:

* A line indicating the matrix-coordinates of the cell in the model.
* A line indicating the size of the cell in km along each direction.
* The matrix of α for each spectral component. Each row (line) is a frequency, each column is a direction. Currently all the rows will result equal, because alphaBetaLab supports only transparency coefficients changing with directions.
* The matrix of β for each spectral component. Each row (line) is a frequency, each column is a direction. Currently all the rows will result equal, because alphaBetaLab supports only only transparency coefficients changing with directions.

## Modelers’ commands

Wave modelers should not stick directly with the modules/classes/algorithms described in the previous sections, but should invoke simple end-user functions with simple parameters, that accomplish all the requested activities, e.g. building the mesh, building the high-resolution matrix of alpha (for example loading the ETOPO1 bathymetry), estimating the alpha-beta (parallelizing if requested) and saving the results to the UOST configuration file, all using the desired options for the algorithm. These end-user commands are implemented in the abEstimateAndSave module (currently only the implementation of abEstimateAndSaveRegularEtopo1, working for latlon grids, is finalized). An example of script invoking abEstimateAndSaveRegularEtopo1 to generate the UOST configuration files is obstFileBuilder.py, of the example on the Caribbean Sea, mentioned in section 1.1.3. Stubs for other similar methods, for other types of meshes, are also present in module abEstimateAndSave.

## Further steps and issues

1. The most important future development is the implementation and testing of the algorithm to generate \_abGrid objects from triangular SMC meshes, allowing the first applications on non-regular grids.
2. A practical issue of alphaBetaLab working on large domains is that it is rather slow. This is partly due to memory management issues, as many parts of the algorithm (and in particular, library shapely) use a no-caching lazy-instancing approach, that creates and re-creates objects whenever these are needed for specific computations. This problem has been partly overcome building caching systems (for example, the cache of neighbor polygons in the \_abGrid class). But further optimizations are possible. Another development that could significantly boost the execution speed is the optimization of the computation of β, now rather expensive, as explained in section 2.4.2.
3. Properly dealing with long breakwaters (see section 2.6.3) will become more and more important as the resolution increases, together with the possibility of having long breakwaters between two neighboring cells.
4. A better detection of what cells intersecting land bodies need a parameterization of unresolved obstacles is an important point to improve the accuracy of UOST, especially in regular and SMC grid (for triangular meshes this may be not an issue, as alphaBetaLab should just exclude from computation the boundary nodes). The limitations of the algorithm currently implemented in alphaBetaLab (section 2.6.2) are shown in (Mentaschi et al., 2018a).

***References*:**

Li, J.-G., 2011. Global Transport on a Spherical Multiple-Cell Grid. Mon. Weather Rev. 139, 1536–1555. doi:10.1175/2010MWR3196.1

Li, J.G., 2012. Propagation of ocean surface waves on a spherical multiple-cell grid. J. Comput. Phys. 8, 262–277.

Mentaschi, L., Kakoulaki, G., Vousdoukas, M., Voukouvalas, E., Feyen, L., Besio, G., 2018a. Parameterizing unresolved obstacles with source terms in wave modeling: A real-world application. Ocean Model. 126, 77–84. doi:10.1016/j.ocemod.2018.04.003

Mentaschi, L., Kakoulaki, G., Vousdoukas, M.I., Voukouvalas, E., Feyen, L., Besio, G., 2018b. alphaBetaLab: automatic estimation of subscale transparencies for the Unresolved Obstacles Source Term in ocean wave modelling. SoftwareX.

Mentaschi, L., Pérez, J., Besio, G., Mendez, F.J., Menendez, M., 2015. Parameterization of unresolved obstacles in wave modelling: A source term approach. Ocean Model. 96, 93–102. doi:10.1016/j.ocemod.2015.05.004

Roland, A., 2008. Development of WWM II: Spectral wave modelling on unstructured meshes. Technische Universitat Darmstadt, Institute of Hydraulic and Water Resources Engineering.