WW3 Tutorial: Grid Generation

Purpose

The purpose of this exercise is to introduce users to a grid generation software called GRIDGEN. It has been developed specifically for generating grids that can be used for WAVEWATCH III applications. In this case, we will create a grid for South Africa and define its boundary with a global grid at 0.5° resolution.

Please note: the grid generation software is a series of MATLAB routines that are used to create ASCII grid files. These files are then provided to the **ww3_grid** *program to generate the specific model definition files. Some limited proficiency in MATLAB would be helpful in following along with the exercises, but is not necessary.*

I. Inputs

Copy the **TUTORIAL_GRIDGEN** folder and the bathymetry data onto your home directory:

```
cd ~
cp -r $WAVE_COURSE/TUTORIALS/TUTORIAL_GRIDGEN .
ln -sf  $WAVE_DATA/BATHY/*    ~/TUTORIAL_GRIDGEN/reference/
or (if you are working remotely)
wget -mnH --cut-dirs=5 ftp://ftp.ifremer.fr/ifremer/ww3/COURS/WAVES_SHORT_COURSE/TUTORIALS/TUTORIAL_GRIDGEN/
wget -mnH --cut-dirs=4 ftp://ftp.ifremer.fr/ifremer/ww3/COURS/WAVE_DATA/BATHY
ln -sf BATHY/* ~/TUTORIAL_GRIDGEN/reference/
```

Let's go now in ***~/TUTORIAL_GRIDGEN***.

1) Content of the tutorial folder

At the start of this tutorial exercise, you will find there the following files :

bin		(all matlab programs to make up the gridgen software)
	generate_grid.m	check the manual and/or the documentation
	read_mask.m	to learn more about each function
	split_boundary.m	
	create_grid.m	to create all the grids
	create_boundary.m	to modify the mask
	…	
reference		(data needed to create the grid)
	etopo*.nc, gebco.nc	high resolution bathymetric data
	coastal_bound_*.mat	coastal boundary polygons
	…	

namelist (pre-defined namelists to use the grid generation script)

gridgen.GLOB-30M_gebco.nml
gridgen.ZA-7M.nml
grigden.BENG-3M.nml

data (folder where the output files of gridgen are created)

GLOB-30M.bot
GLOB-30M.mask_nobound
GLOB-30M.obst
GLOB-30M.meta

2) Notes on the reference data

Coastal boundaries

The **coastal boundaries** are Matlab binary files, stored in a data structure form: loading any one of these files will provide the user with the form of the data structure. Users can use their own coastal polygons with this package as long as they are stored using the same format. For each boundary, the coastal polygon is defined in an **anti-clockwise pattern** with the first and last points being the same (to effectively close the boundary).

Reference bathymetry

The **reference bathymetric data** provided with this package (in the 'reference' folder) are the global ETOPO1 and ETOPO2 data files, a global GEBCO data file and a regional SRTM file covering the South Africa. All of them are in NetCDF format.

Users can use their own bathymetric data sets but they will have to do one of two things:

- change the algorithm of the bathymetry generating subroutine (*generate_grid.m* or *generate_grid_xyz.m*) to accurately read the reference bathymetry
  or
- store the data in the same format as the provided NetCDF bathymetric file; that is the solution that was chosen for the file gebco.nc.

In that case, users must be careful to respect the following main requirements for the bathymetry data sets: the file must be in NetCDF format with a '*.nc' extension, the elevation variable must depend upon the 'longitude' and 'latitude' dimensions (whatever their actual names), the longitude and latitude variables must be provided as 1D arrays and have an 'actual_range' attribute.

As regards bathymetry data itself, generally speaking there are 4 types of bathymetry references:

- Lower astronomical tide
- Upper astronomical tide
- Mean lower low water: average of the lower low water height of each tidal day
- Mean sea level: arithmetic mean of hourly heights

… and two sign conventions:

- depth as a positive number, increasing downwards from the surface of the water towards the bottom of the ocean (oceanographic convention)
- or the opposite: depth as a negative number, increasing upwards to reach 0 at the surface

For GRIDGEN, it is very important that the bathymetry data set uses the **Mean Sea Level** as the bathymetry level of reference (see OHI convention: http://www.iho.int/srv1/index.php?lang=en), because the WW3 model must have the same bathymetry reference as the currents and level forcings it will be using. Using other references would lead to over- or underestimating wet points at the coast (e.g. using LAT would mark some points 'dry' when they are actually under water all the time except for very low tides).

The convention for depth must also be that **depths are negative numbers downwards**. (Otherwise the depth array created in the first step of the GRIDGEN toolbox will return wrong values).

Below is a list of a few bathymetry sets available with their distribution country and spatial extension:

- etopo1 1' US (global): https://www.ngdc.noaa.gov/mgg/global/global.html
- globe 30'' US (global): https://www.ngdc.noaa.gov/mgg/topo/globe.html
- srtm 1'' US (global): http://www2.jpl.nasa.gov/srtm/
- gebco 30'' GB (global): http://www.gebco.net/data_and_products/gridded_bathymetry_data/
- shom 100m FR (France): data.shom.fr

If you choose to use such data sets, make sure that they match the requirements listed above.

II. Generate a grid: South Africa

From the tutorial directory, go to the 'bin' directory and launch matlab:

```
cd ~/TUTORIAL_GRIDGEN/bin
matlab &
```

From here on the commands will be in the matlab environment. The commands that we will go through step by step in this exercise are also provided in a function in the bin directory.

1) Define basic parameters for the desired grid

Purpose: Set up the paths and the grid spatial extension as follows:

```
set(groot,'DefaultFigureColormap',jet)
bin_dir = '~/TUTORIAL_GRIDGEN/bin';
ref_dir = '~/TUTORIAL_GRIDGEN/reference';
nml_dir = '~/TUTORIAL_GRIDGEN/namelist';
data_dir = '~/TUTORIAL_GRIDGEN/data';
dx = 0.125;  dy = 0.125;
lon_west = 12;
lon_east = 23;
lat_south = -38.9;
lat_north = -27.2;
lon1d = (lon_west:dx:lon_east);
lat1d = (lat_south:dy:lat_north);
[lon,lat] = meshgrid(lon1d,lat1d);
```

The *lon* and *lat* arrays are 2D arrays that provide longitudes and latitudes for each cell of the desired grid. Units are in degrees.

To allow for the GRIDGEN functions to be called from anywhere, add them to the path:

```
addpath(bin_dir,'-END');
```

Load the boundary mat file. The user has the option of choosing from several resolutions, with the grids being generated faster with the coarser boundaries. Best practice is to build grids with the full resolution boundaries, even though it can take much more time.

```
load([ref_dir,'/coastal_bound_full.mat']);
```

Loading this boundary file should have generated a data structure array called 'bound'. You can check the nature of the structure and the size of the array by typing bound without the semicolon at the end:

```
bound
```

The total number of polygons and the size of individual polygons will depend on the resolution set chosen. To get an idea of what this particular set looks like plot the 1000 first polygons on a figure:

```
figure(1);clf;
for i = 1:1000
plot(bound(i).x,bound(i).y,'.','MarkerSize',0.5);
hold all;
end;
```

The full plot (188 603 polygons) should look like:

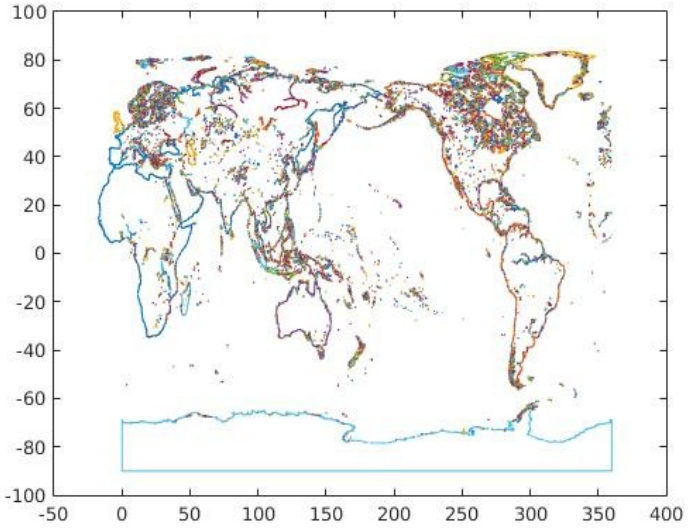


Figure 1: Boundary Polygons

2) Create a bathymetry grid

Purpose: compute depth for the final grid and create **m1**, the initial land-sea mask

This is done using the function **generate_grid**. The raw bathymetric data is read from the NetCDF file gebco.nc in the 'reference_data' directory.
From the bathymetry data file, depth is either averaged or interpolated at each point of the desired output grid (depending on whether the required resolution is coarser or finer than the file resolution). The value **DRY_VAL** is attributed to dry cells. Theses dry points are defined with 2 criteria:

- the variable **CUT_OFF** identifies the water depth below which the cells should be identified as 'wet' (and their depths computed);
- the variable **LIM_BATHY** identifies the minimum proportion of reference bathymetry cells corresponding to the target cell that should be 'wet' to identify the target cell as 'wet'. (to then add obstruction corresponding to the percentage of land on the grid cell)

NB: The value chosen for LIM_BATHY is low because the *generate_grid* function creates a depth array which is used to build the initial mask. It is important at this early stage to emphasize the 'wet' cells while building the initial mask because later in the routine, other functions will refine this mask by checking only 'wet' cells and determining if they should be switched to 'dry.' It does not work the other way round (from 'dry' to 'wet') because the corresponding bathymetry is not available for 'dry' cells.

```
CUT_OFF = 0.0;        % Cut-off depth to distinguish between dry & wet cells
LIM_BATHY = 0.4;   % Base bathymetry cells needing to be wet for the target cell to be considered wet.
DRY_VAL = 999999;  % Depth value set for fry cells
```

The function also needs information on the bathymetry file from which depth values will be computed:

```
ref_grid = 'gebco';  % Name of the file without the '.nc' extension
xvar = 'lon';  % Name of the variable defining longitudes in file
yvar = 'lat';  % Name of the variable defining latitudes in file
zvar = 'elevation';  % Name of the variable defining depths in file
```

```
depth = generate_grid('rect',lon,lat,ref_dir,ref_grid,LIM_BATHY,CUT_OFF,DRY_VAL,xvar,yvar,zvar);
figure(1);clf;
d=depth;d(d==DRY_VAL)=nan; pcolor(lon,lat,d); shading flat; colorbar
```

The function returns a 2D array with the bathymetry values: **depth.** The bathymetric data is in meters and everything else is in degrees.

Plotting the bathymetric data should look like Figure 2. For aesthetic purposes, dry cells have been marked as NaNs, which are not plotted in matlab. The negative values refer to depth below Mean Sea Level (MSL).

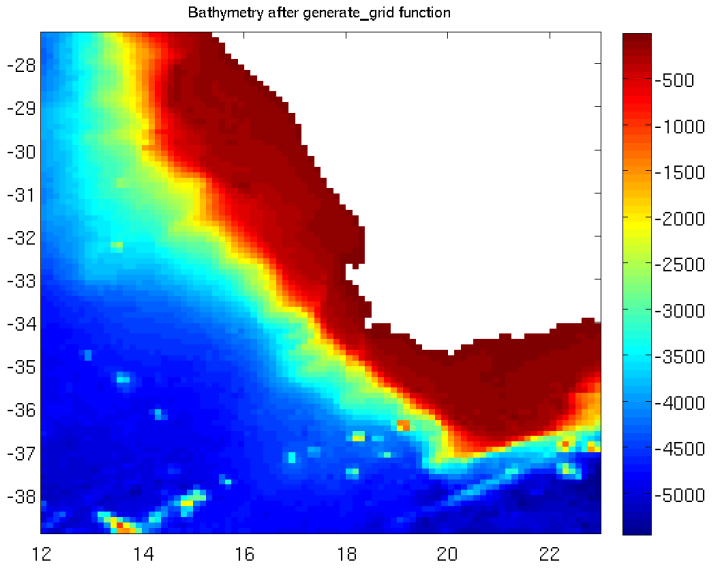


Figure 2: Initial bathymetry

The next step is to set up the land – sea mask, using the newly created bathymetry data set **depth**, to identify the initial set of 'wet' and 'dry' cells:

```
m1 = ones(size(depth));
m1(depth == DRY_VAL) = 0;
```

Plotting the initial mask should look like Figure 3:

```
figure(1);clf;
pcolor(lon,lat,m1);shading flat;caxis([0 3]);colorbar
```

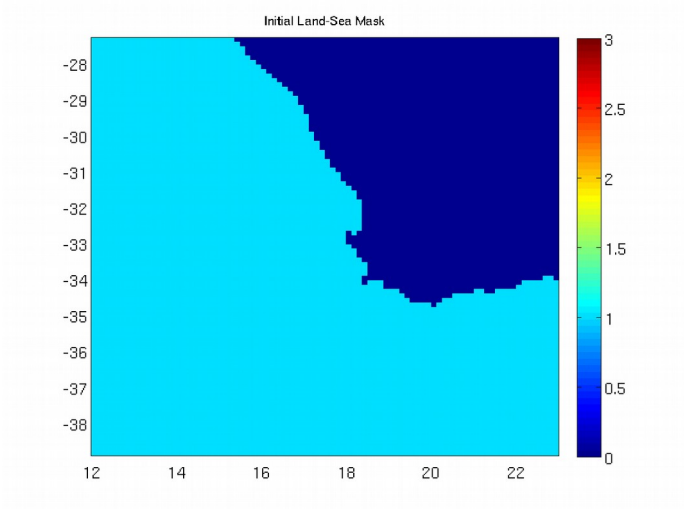


Figure 3: Initial land - sea mask

This initial set **m1** will not be used as the final land – sea mask because the reference bathymetry and the coastal boundaries are not always consistent. That is why, when building the bathymetric data set, we over emphasized the wet cells (determined by the variable 'LIM_BATHY' in the bathymetric generation section).
In the following part of the tutorial, we will see how to take into account the coastal boundary polygons.

3) Compute boundaries

Purpose: identify the coastal boundary polygons in the 'bound' database corresponding to the geographical extent of the desired grid

First, the coastal boundaries need to be identified within the computational domain using the GSHHS database (or an optional user-defined database), to properly account for boundary closure and splitting of boundaries. This is done using the *compute_boundary* function.

We start with identifying the domain on which the function must apply. It is chosen a little larger than the actual grid considered, to account for all the cells at the edges of the domain:

```
lon_start = min(min(lon))-dx;
lon_end = max(max(lon))+dx;
lat_start = min(min(lat))-dy;
lat_end = max(max(lat))+dy;
coord = [lat_start lon_start lat_end lon_end];
MIN_DIST = 4;  % minimum distance between edge of polygon and boundary
```

Now, we compute the subset of boundaries within the domain using the command *compute_boundary*:

```
[b,N] = compute_boundary(coord,bound);
```

Here, **N** is the total number of boundaries generated and **b** returns a data structure array of the subset of boundaries generated using **N**. (Note that **bound** are the original boundary polygons that have been loaded from the reference directory). Plotting the boundaries should yield a plot like Figure 4

```
figure(1);clf;
for i = 1:N
plot(b(i).x,b(i).y);
hold on;
end;
```

NB1: The 'bound' structure of coastal polygons only accounts for polygons with a longitude range of 0°-360°. To compare them with the grid, only positive longitude values can be given as arguments in 'coord'. This is not a problem here since lon_west = 12° and lon_east = 23°, but for other grids it might. In that case, an offset of +360 should be added to the longitudes in the 'coord' variable.
NB2: The **MIN_DIST** variable inside the function can sometimes cause trouble. It is a threshold defining the minimum distance between the edge of a polygon and the inside/outside boundary. A low value reduces computation time but can raise errors if the grid is too coarse. If faced with a crash at this stage of the grid creation process ("*Attempted to access out2in_xcross(0); index must be a positive integer or logical.*"), consider increasing MIN_DIST.

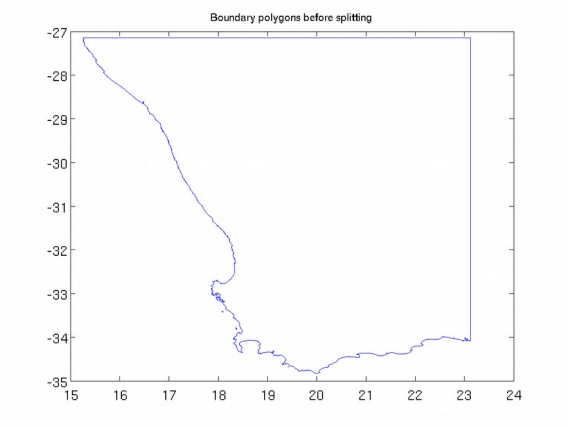


Figure 4: Coastal polygon in the boundary domain

4) Split up boundary polygons

Purpose: split up the 'b' structure of boundary polygons previously created into a structure of smaller polygons 'b_split', to reduce computation time for the next step (mask cleaning).

At this stage, we want to improve the land – sea mask by making it consistent with the coastal boundaries **b**, using the *clean_mask* function. For a given boundary polygon, this function checks all the wet cells in that polygon and determines if they should be switched from 'wet' to 'dry'. However, this is one of the most computationally intensive parts of the software: the time taken for any search depends upon the number of points making up a particular polygon (the more points, the longer the search). To save some time, a boundary splitting routine has been developed that splits up the boundary polygons to more manageable levels:

```
SPLIT_LIM = 0.5;
b_split = split_boundary(b,SPLIT_LIM,MIN_DIST);
```

where **SPLIT_LIM** in *split_boundary* determines the width (or height) cut off limit (in degrees) above which the boundaries are split up into smaller chunks. SPLIT_LIM is usually defined from 5 to 10 times the maximum value of the array [dx dy]. The split up boundary polygons should look like Figure 5:

```
Nb = length(b_split);
figure(1);clf;
for i = 1:Nb
plot(b_split(i).x,b_split(i).y);
hold on;
end;
```

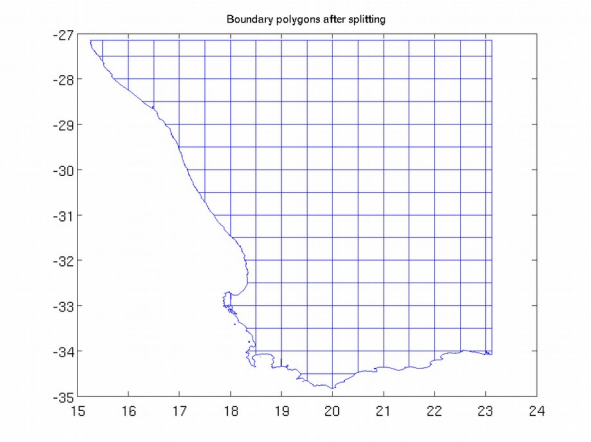


Figure 5: Split-up boundary polygons

5) Clean up the initial mask

Purpose: clean up the initial land-sea mask **m1** by checking all the wet cells and determining if they lie outside the boundary polygons (**b_split**) or not.

Now, we are ready to run the clean up routine *clean_mask*. This function uses the initial land – sea mask **m1** and the structure of split boundary polygons **b_split**, and returns a cleaned mask **m2**.
The *clean_mask* function checks what portion of each wet cell lies within the boundary polygons. Comparing this value to **LIM_VAL**, it determines if that cell should be switched from 'wet' to 'dry'. (The function works only for switching the wet cells to dry and not the other way round, since for each wet cell

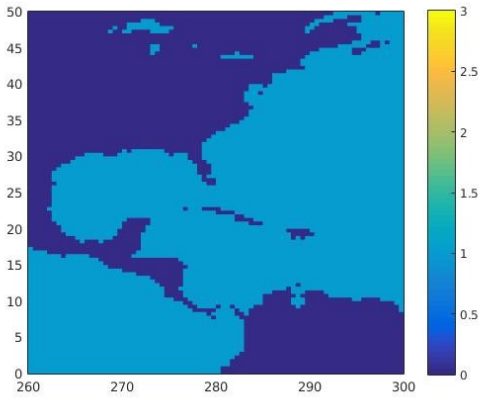


Figure 6: Initial Land - Sea mask around the Gulf of Mexico

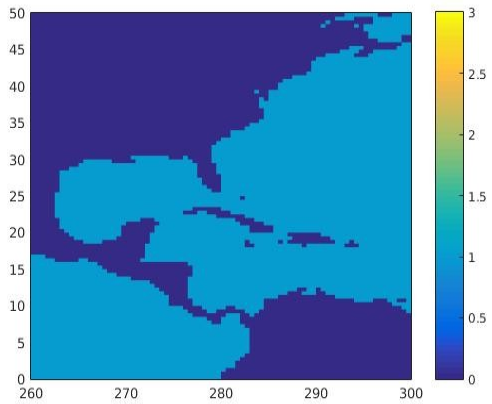


Figure 7: Clean mask: the Great Lakes are missing but some other small lakes were artificially created

a corresponding depth would have to be determined as well.)

The parameters for *clean_mask* are:

- **LIM_VAL**: cut-off value below which, if the cell domain lies inside the polygon, it is marked 'dry';
- **OFFSET**: additional buffer around boundary, set to check if the cell is crossing the boundary; usually set to max([dx dy]).

Important note: this part of the grid generation routine should be skipped if building **inundation grids**, since then the users deliberately want cells that lie within the coastal polygons to be marked wet.

```
LIM_VAL = 0.5;
OFFSET = 0.125;   % max([dx dy])
m2 = clean_mask(lon,lat,m1,b_split,LIM_VAL,OFFSET);
```

Plotting this cleaned up version of the mask looks like Figure 8. Because of the way the function works, the cleaned up mask m2 gets rid of the water bodies that are inside the coastal boundary polygons (ie lakes). It does not result in any visible change in our case, but if there were some lakes in the considered grid, they would have disappeared by now. Thus, one needs to be careful in trying to use this software for creating grids for lakes.

```
figure(1);clf;
pcolor(lon,lat,m2);shading flat;caxis([0 3]);colorbar
```

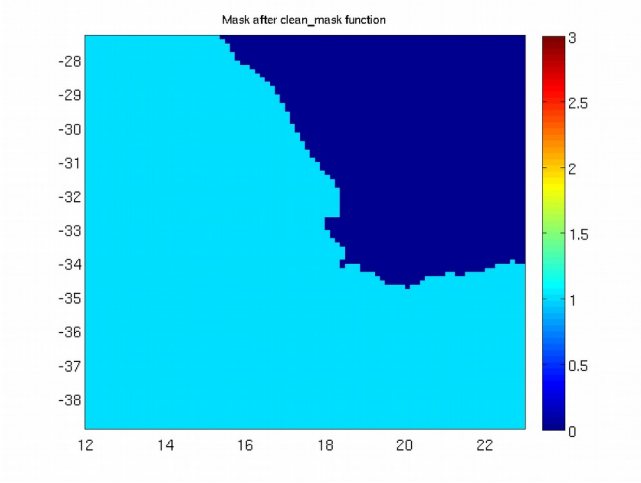


Figure 8: Mask after clean up

Another example of the action of *clean_mask* in an area containing lakes and different oceans:

6) Remove artificially generated lakes

Purpose: use the *remove_lake* function to clean the **m31** and **m32** masks from the lakes artificially generated in the previous steps (or to keep only the largest water body).

The mask clean up is still not complete. Depending on the grid, the *clean_mask* function may have artificially generated lakes (isolated wet cell(s) that are not connected to the main body of water). These typically arise because either the grid resolution is not fine enough to adequately resolve land-sea margin, or the domain includes other water bodies. This is not the case for the South Africa grid, but in Figure 7 you have an example of a grid where there are artificially generated lakes To remove these lakes we use a function called *remove_lake*, taking as inputs the cleaned mask **m2** and two other variables, LAKE_TOL and IS_GLOBAL, defined as follows:

- **LAKE_TOL**: Tolerance value determining if all the wet cells corresponding to a particular water body should be flagged 'dry' or not. If LAKE_TOL > 0, all water bodies having less than this value of total 'wet' cells will be flagged 'dry'. If LAKE_TOL = 0, the output and input masks are unchanged. If LAKE_TOL < 0, all but the largest water body are flagged 'dry'.
- **IS_GLOBAL**: Flag set to 1 for global grids, else to 0. Determines if cells wrap around longitudes.

```
LAKE_TOL = 100;
IS_GLOBAL = 0;
[m4,mask_map] = remove_lake(m2,LAKE_TOL,IS_GLOBAL);
```

The *remove_lake* function finds all the different water bodies, and then uses the value of the variable **LAKE_TOL** to determine what to do to the different water bodies. If LAKE_TOL is a negative number, then all but the largest water body are marked dry. If on the other hand it is a positive number, then all water bodies having less than this value of total 'wet' cells are flagged 'dry'. If LAKE_TOL = 0, the output and input masks are unchanged. The **IS_GLOBAL** variable determines if the mask array is global (cells wrap around). A value of 0 indicates it is not. The function returns two arrays, a modified mask **m4** and a two dimensional array **mask_map** providing unique ids for the different water bodies. After running through this function, the new mask looks like:

```
figure(1);clf;
pcolor(lon,lat,m4);shading flat;caxis([0 3]);colorbar
```

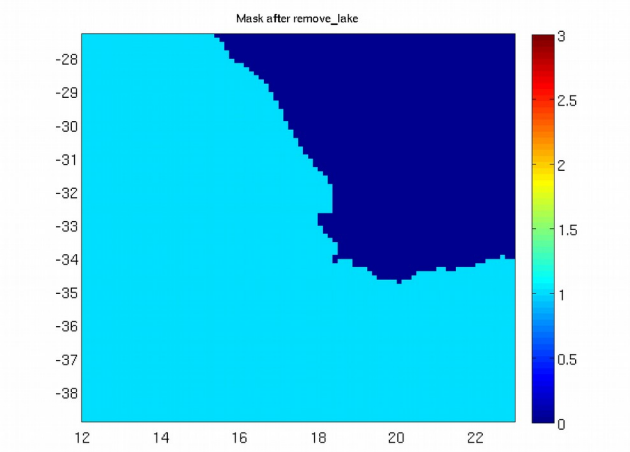


Figure 9: Land-Sea mask after clean up of separate water bodies

In our case, there is not much change between the cleaned up mask **m2** (Figure 8) and the mask **m4** (Figure 9) after applying the *remove_lake* function, because there were not any artificially lakes in m2 and there is only one water body. You can check that by plotting the water bodies identified in mask_map, as shown in Figure 10:

```
figure(1);clf;
pcolor(lon,lat,mask_map);shading flat;caxis([-1 6]);colorbar
```

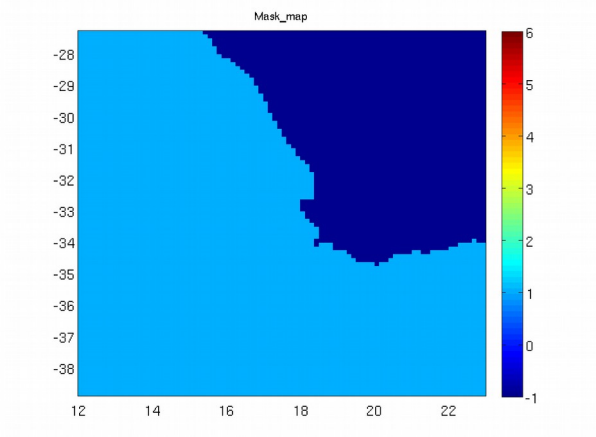


Figure 10: Water bodies identified in mask_map: here, there is only one

However this is not always the case. If we consider the area around the Gulf of Mexico, for which we saw earlier that small lakes were artificially created (Figure 6), the *clean_mask* function has much more visible effects. With LAKE_TOL=-1, the function actually removes every wet cell except the ones belonging to the Atlantic Ocean, as shown in Figure 11. That is because it corresponds to the largest water body (see Figure 12). Artificial lakes are gone, too. If wishing to keep a particular water body, it is possible to get its ID number in mask_map, then switch the corresponding points in m4 back to 1 with the command line “m4(mask_map == ID) = 1”.

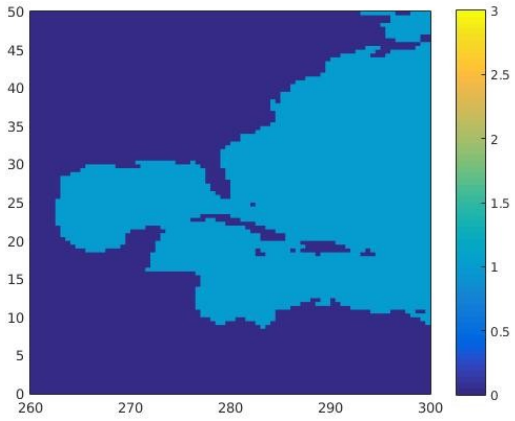


Figure 11: Land - Sea mask after clean up of separate water bodies

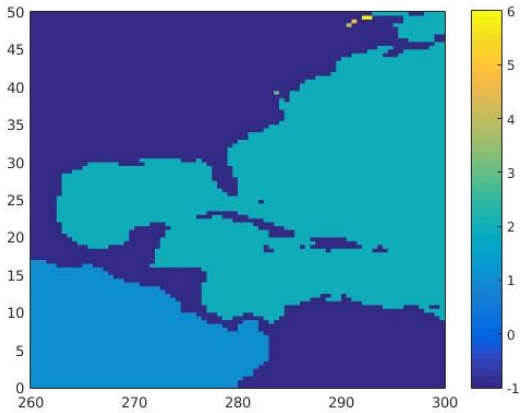


Figure 12: Different water bodies with unique ids

7) Generating obstruction grids

Purpose: from the final land-sea mask **m4** and the coastal boundaries **b**, create the obstruction grids needed for ww3_grid.

Once the land – sea mask **m4** has been adequately defined, together with the bathymetric data, it is time to create the obstruction grid(s). From the morning lectures, we know that the obstruction grids can be generated by either considering only the obstructions in the cell itself, considering obstructions in one neighbor, or considering obstructions in both neighbors. The *create_obstr* function needs mainly **m4** and **OBSTR_OFFSET** as arguments. OBSTR_OFFSET is the flag deciding whether neighbors should be considered (1) or not (0) and is required twice: once for left/down neighbors, once for right/up neighbors.

```
OBSTR_OFFSET = 1;
[sx1,sy1] = create_obstr(lon,lat,b,m4,OBSTR_OFFSET,OBSTR_OFFSET);
```

Here we have built the obstruction grids using both neighbors. The resulting obstruction grids are plotted in Figure 13 and Figure 14.

```
sx1(find(m4==0))=NaN;
sy1(find(m4==0))=NaN;
figure(1);clf;
pcolor(lon,lat,sx1);shading flat;caxis([0 1]); colorbar
figure(2);clf;
pcolor(lon,lat,sy1);shading flat;caxis([0 1]); colorbar
```

NB: Like the *compute_boundary* and *split_boundary* functions, *create_obstr* takes **b** as an argument. As we saw before, the longitudes in **b** range from 0° to 360° only. That means the longitudes array **lon** needs to be temporarily modified if you are working with the etopo2 convention (longitudes from -180° to 180°). This is not the case here as our bounds are lon_west=12° and lon_east=23°, but you must keep it in mind for other grids.

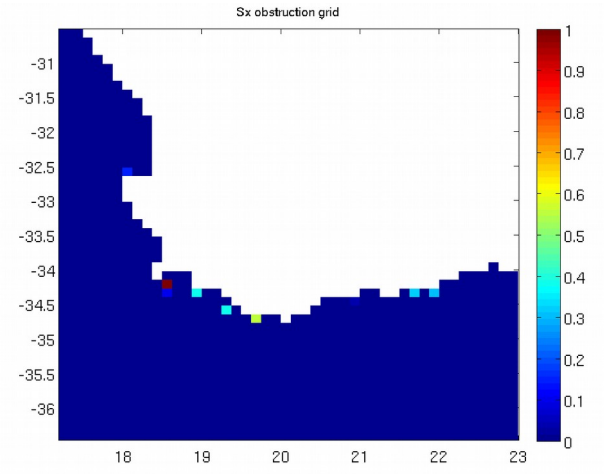


Figure 13: Obstruction grid along x

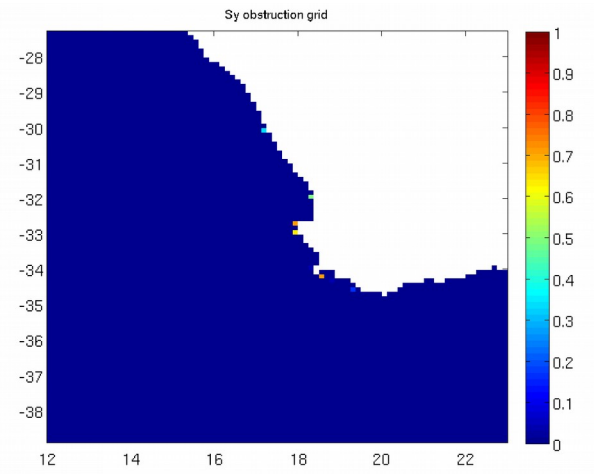


Figure 14: Obstruction grid along y

8) Save files

Once the grids have been generated, they need to be saved in ASCII files, that will later be read by **ww3_grid** to generate binary model definition files. Since **ww3_grid** allows for a scaling factor, we save these variables as integers. Typically we round bathymetric data to the third decimal place and the obstruction values to the second decimal place.

```
fname='ZA-7M';
depth_scale = 1000;
obstr_scale = 100;
d = round((depth)*depth_scale);
write_ww3file([data_dir,'/',fname,'.bot'],d);
write_ww3file([data_dir,'/',fname,'.mask_nobound'],m4);
d1 = round((sx1)*obstr_scale);
d2 = round((sy1)*obstr_scale);
write_ww3obstr([data_dir,'/',fname,'.obst'],d1,d2);
write_ww3meta([data_dir,'/',fname],[nml_dir,'/gridgen.',fname,'.nml'],'RECT',lon,lat,1/depth_scale,1/obstr_scale,1.0);
```

In the 'data' folder, there should now be 3 ASCII data files for bathymetry (*.bot), mask (*.mask_nobound) and obstruction grids (*.obst) as well as one ASCII *.meta file that shall be used to create the **ww3_grid.nml** file later.

III. Generate grids for the multi_grid run: create_grid

This “test grid” was easy to create manually and step by step because it had the simplest possible features: the grid did not cross the Greenwich meridian and all longitudes were positive. However, depending upon your bathymetry data set and the grid you want to create, this might not always be the case. In a general way, you have to be careful with the *generate_grid*, *compute_boundary*, *split_boundary* and *create_obstr* functions. They only accept longitudes ranging from 0° to 360° so for these special steps, you may have to add an offset to all or part of your longitude array, or even split your grid in two parts. Thus, repeating the individual steps of the previous section to create another grid can be a cumbersome task.

Fortunately, the GRIDGEN toolbox comes with a generic script to help you do this in a more convenient and secure manner: the ***create_grid*** function takes care of the grid generation for you in an automatic way, provided you gave it the right parameters. The only argument it needs is a namelist file containing all the information on the desired grid. All that is left for you to do is to fill in this file with the correct values – which remains a crucial step.

The previous section was a step-by-step introduction. Now, to prepare the multi-grid run that you will achieve in another tutorial, we are going to use the automatic routine to create a second grid around Benguela with 3’ resolution: 'BENG-3M'. All the information needed about this grid is stored in the 'gridgen.BENG-3M.nml' file. This is the only argument taken by the ***create_grid*** function.
Now, create this grid using the automatic toolbox:

```
create_grid('~/TUTORIAL_GRIDGEN/namelist/gridgen.BENG-3M.nml');
```

After this step you should have 4 new ASCII file in your data folder, starting with 'BENG-3M'.

IV. Set up boundary conditions for multiple grids

At this stage, we have generated the bathymetry, mask, obstruction & meta files for the 'ZA-7M' and the 'BENG-3M' grids, so the user's working directory should have these 2 sets of grid files: ZA-7M.* and BENG-3M.* in addition to the GLOB-30M.* files.

To run the wave model properly, we must set up the boundary conditions for 'BENG-3M' and 'ZA-7M'. Indeed, each of these two grids needs to receive information from a coarser one: the 'BENG-3M' grid receives data from the 'ZA-7M' grid, which itself gets information from the global grid. In the following part of the tutorial, we will see how to properly define the boundary points for these two grids.

Until now, we have been working with mask values of 0 (land) and 1 (sea). However, there are two additional values for the mask file that can be set: 2 (boundary points) and 3 (excluded points). These values should define the boundary points in the inner, finer grid, so the mask for the 'ZA-7M' and the 'BENG-3M' grids need to be updated.

First, let us set up the boundaries for the 'BENG-3M' grid. It takes information from the coarser 'ZA-7M grid'. In this section, the target grid is defined as the fine resolution grid whose mask values shall be changed to determine boundary and excluded points (ie BENG-3M), while the base grid is defined as the coarser outer grid from which the target grid is going to get information (ZA-7M). The aim here is to define appropriate points where boundary data from the base grid is provided to the target grid. Since version 3, WW3 allows for boundary points (for the finer grid) to be defined inside the grid, thus allowing for features such as coast line following grids, even though we are using regular grids.

The GRIDGEN toolbox provides a function called ***create_boundary*** that takes care of the creation of boundary point in the inner mask. This function only needs the ***gridgen.BENG-3M.nml*** namelist file, more specifically the 'GRID_INIT' namelist inside this file. The namelist provides information on the name of target and base grids (here 'BENG-3M' and 'ZA-7M' respectively), and on the type of boundary selection you want to use.

There are 3 options available, defined by the 'SELECT_BOUND' parameter in the namelist file:

- SELECT_BOUND = 0 : select the boundary polygon(s) manually. This will plot the current mask (with no boundary) for the target grid. You will then click on the plot to define the vertices of your boundary polygon(s). Be aware that the polygon **must be defined in a counter clockwise direction** and be closed (this allows us to easily identify points inside, on and outside the polygon).
- SELECT_BOUND = 1 : select automatically around the target grid's borders. This is especially useful if the aim is to use the entire target grid.
- SELECT_BOUND = 2 : select from a .poly file

From there, all you have to do is run the function:

```
create_boundary('~/TUTORIAL_GRIDGEN/namelist/gridgen.BENG-3M.nml');
```

The outputs of the *create_boundary* function are:

- the file '**BENG-3M.mask**', which corresponds to the new mask for the inner grid, taking into account the presence of boundaries (some values are set to 2 and/or 3).
- the files '**BENG-3M.bound**' and '**BENG-3M.fullbound**', which provide the position and name of the boundaries so that the coarser grid knows where to compute the spectra, and the finer grid knows where to look for it. The 'BENG-3M.fullbound' file gives **all** the inner grid's boundary points with the target grid's resolution (which is most often not necessary), whereas the 'BENG-3M.bound' file provides boundary points at the base grid's resolution. This is only useful in case of single grid implementation where a coarse grid needs to process the spectral boundaries in advance for a fine grid. These files are not used in multigrid implementation.

Inside ***create_boundary***, we use a function called *modify_mask*. It sets all the mask values outside the polygon to 'undefined' in the target grid's mask (these cells are not used in the computation), and follows along the polygon to make sure that for every potential boundary cell in the target grid, there are active (wet) cells in the base grid from which data can be received.

For the automatic generation of polygon at the borders of the target grid (SELECT_BOUND=1), the final mask is given by Figure 15. An example of final mask with manual selection of polygon (SELECT_BOUND=0) is given by Figure 16.

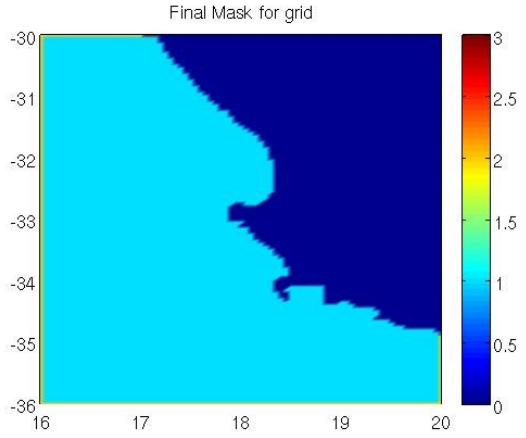


Figure 15: Final mask with boundary points, automatic selection of boundaries

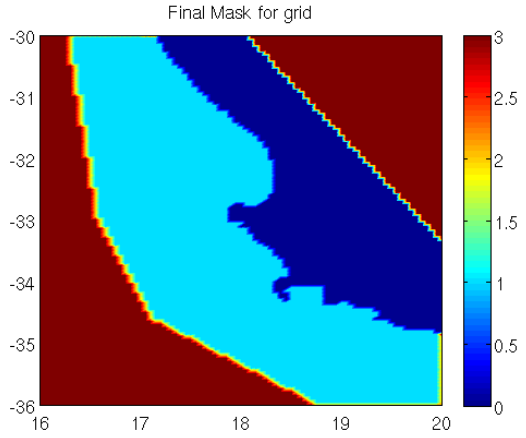


Figure 16: Final mask with boundary points, manual selection of boundaries

Note that the active boundary cells in the sea along the edges of the grid in Figure 15 are 2. The red cells in Figure 16 correspond to excluded points (where mask=3), there are active boundary points on land, this is just an artifact on the plot.

In a second phase, as the South Africa grid (providing data to the Benguela grid) gets boundary information from the global grid, you need to run ***compute_boundary*** one more time for this grid. Here, the target grid is 'ZA-7M' and the base grid is 'GLOB-30M', as defined in the *gridgen.ZA-7M.nml* file.

```
create_boundary('~/TUTORIAL_GRIDGEN/namelist/gridgen.ZA-7M.nml');
```

Note : if you don't have a base grid, you can still define the active boundaries by setting the base grid to the same name as your grid for the namelist FNAMEB.

V. Conclusion

In this tutorial we generated the bathymetry and obstruction files, as well as the mask file with boundary information for both grids. The toolbox also provided a list of the boundary points with their name and position for each grid and a *.meta file with information on the grid type, extension and resolution. The next step will be to use these files to run the **ww3_grid** program for both grids.

Now that you know how it works you can use it in the future to create your own grids and define their boundaries.

To conclude, 3 kinds of grids can be generated using GRIDGEN :

-The most usually case is a regular grid based on a regular bathymetric data file which was the case of the previous example ZA-7M and BENG-3M.

-You can create a curvilinear grid based on a regular bathymetric data file but you will need to define a file with all the latitudes depending on the longitude and the same for the longitude, and example is given for the ARC-12K grid with lat/lon files in the reference folder.

-The last specific case is using a curvilinear bathy file to generate a curvilinear grid, a dedicated program is available “create_grid_curv.m”, an example is given for the NORWAY-8K.

References

Chawla, A., and Tolman, H..L. (2007). “Automated grid generation for WAVEWATCH III”. Technical Bulletin 254, NCEP/NOAA/NWS, National Center for Environmental Prediction, Washington, DC

Chawla, A. and Tolman, H. L. (2008) “Obstruction grids for spectral wave models”, *Ocean Mod.*, **22**, 12 – 25

Tolman, H. L. (2003). “Treatment of unresolved islands and ice in wind wave models”, *Ocean Mod*,.**5**, 219–231.

More information:
Arun Chawla (Arun.Chawla-at-noaa.gov)
Fabrice Ardhuin (Fabrice.Ardhuin-at-ifremer.fr)
Mickael Accensi (Mickael.Accensi-at-ifremer.fr)
Marion Huchet (Marion.Huchet-at-ifremer.fr)

Appendix A – Converting a XYZ bathy file to Mean Sea Level reference

The xyz bathy file must respect the Mean Sea Level reference with depths negative downwards.

In case of a bathy defined in Lower astronomical tide reference with depths positive downwards, knowing the tide shift to the Mean Sea Level, here is an example of script to convert the file :

Lower Astro Tide bathy file :

1	-1.4604994	44.3423289	52.9
2	-1.4575266	44.3423289	52.9
3	-1.4545538	44.3423289	52.9
4	-1.451581	44.3423289	52.9
5	-1.4486082	44.3423289	52.9
6	-1.4456354	44.3423289	54.0

Mean Sea Level bathy file :

-1.4604994	44.3423289	-55.25
-1.4575266	44.3423289	-55.25
-1.4545538	44.3423289	-55.25
-1.451581	44.3423289	-55.25
-1.4486082	44.3423289	-55.25
-1.4456354	44.3423289	-56.35

Script to convert from Lower Astro Tide to Mean Sea Level :

```
#!/bin/tcsh -e

# Define the shift between Lower Astro Tide and Mean Sea Level
shift=2.35
# Loop on each line of the file
cat bathy_LowerAstroTide.xyz | while read line; do
# Split the line and get the longitude as second argument
  lon=$(echo ${line} | cut -d ' ' -f2)
# Split the line and get the latitude as third argument
  lat=$(echo ${line} | cut -d ' ' -f3)
# Split the line and get the elevation as fourth argument
  ele=$(echo ${line} | cut -d ' ' -f4)
# Inverse the elevation sign and remove the water level shift
  ele=$(echo "scale=4;-($ele) - $shift" | bc)
# Write the longitude, latitude and new elevation to the output file
  echo $lon $lat $ele >> bathy_MeanSeaLevel.xyz
done
```

Appendix B – Converting a bathy file from Shapefile to ASCII format

Converting the Mean Sea Level bathy from a Shapefile "shp" format to a ASCII "xyz" format can be done using the GDAL tools hosted by the Open Source Geospatial Foundation http://www.gdal.org/index.html

Install the GDAL tool and convert the shapefile to ascii :

```
apt-get install gdal-bin
ogr2ogr -f CSV bathy_MeanSeaLevel.csv  bathy_MeanSeaLevel.shp -lco GEOMETRY=AS_XYZ
remove the first line of the output .csv file (x,y,z header)
```

The ASCII file can now be converted with gmt tools. Please Appendix C for more details.

Appendix C – Converting a bathy file from ASCII to netCDF format

Converting the Mean Sea Level bathy from a ASCII "xyz" format to a netCDF format can be done using the GMT tools from the University of Hawaii at Manoa http://gmt.soest.hawaii.edu/

Install the GMT tool and set the environment variables accordingly to your paths :

```
apt-get install gmt gmt-gshhs-full
export NETCDFHOME=/usr/lib
export GMTHOME=/usr/lib/gmt
export PATH=$PATH:$GMTHOME/bin
```

Many tools are available in this toolbox, the one you will need to convert the ASCII bathy file to a netCDF file is named **xyz2grd**. You can also combine multiple netCDF bathy files using **grdpaste**.

You can retrieve the grid boundaries with these commands :

```
cat bathy_MeanSeaLevel.xyz | cut -d ',' -f1 | sort -u | head -n1   % min longitude
cat bathy_MeanSeaLevel.xyz | cut -d ',' -f1 | sort -u | tail -n1   % max longitude
cat bathy_MeanSeaLevel.xyz | cut -d ',' -f2 | sort -u | head -n1   % min latitude
cat bathy_MeanSeaLevel.xyz | cut -d ',' -f2 | sort -u | tail -n1   % max latitude
```

Defining the bathy input file in first argument, the attribute names with option -D, the output netcdf file with option -G, the resolutions along x / along y with option -I, the west/east/south/north min/max domain coordinates with option -R :

```
gmt xyz2grd  bathy_MeanSeaLevel.xyz     Gbathy_MeanSeaLevel.nc        -I0.003/0.002
-R-1.46/-0.95/44.34/44.89        -Ddegree/degree/elevation/1/0"Bathymetry"/= -
```

If you want to combine two netCDF bathy file, do :

```
 grdpaste regional_bathy.nc global_bathy.nc -Gfull_bathy.nc
```

Appendix D – Define a polyline as domain boundaries with google-earth

Once you have run the create_grid program, you may want to define a specific boundary area. This can be done with google-earth application to define your polygon.

google earth
select the polyline tool
draw your polygon around your area, it can be an opened polygon
save as kml file
remove the first line of kml file

Then you can convert the kml file into a polygon file.
See Appendix C to install gmt tool.

run kml2gmt to create .poly file
rename output file to fname.poly
put it in data directory
set SELECT_BOUND =2 in namelist file
run create_boundary program

Appendix E – Create ww3_grid.nml from the GRIDGEN .meta file

Once the GRIDGEN toolbox has generated all the grid and boundaries files, you have to set the ww3_grid.nml which will be used by WAVEWATCHIII to create the model definition file mod_def.ww3

Start by copying the ww3_grid.nml file from the ww3 templates into your data folder :

```
cp $WW3/model/nml/ww3_grid.nml  data/
```

Open the ww3_grid.nml file to fill in the configuration of your grid. Every section will described.

The **first namelist** is about the frequency and directional definition. You can reduce the lower frequency to depending the largest wave period you can expect on your domain, we usually set it to 0.0373 which to correspond to 27s wave period, in that case you have to also increase the discretization in frequency to 32:

```
&SPECTRUM_NML
  SPECTRUM%XFR         =  1.1
  SPECTRUM%FREQ1       =  0.0373
  SPECTRUM%NK          =  32
  SPECTRUM%NTH         =  24
/
```

The **second section** is about the activation of the wave propagation, refraction and source terms. If you are running the model with strong currents, you have to activate the wavenumbers shifts:

```
&RUN_NML
  RUN%FLCX          = T
  RUN%FLCY          = T
  RUN%FLCTH         = T
  RUN%FLSOU         = T
/
```

The **third section** is used to defined the model time steps. The first time step to calculate is the maximum CFL time step which depend on the lowest frequency Fmin previously set to 0.0373Hz and the lowest spatial grid resolution along x in meter at the latitude closest to the pole, here -38.9S.
dx = min(reslon * cos(maxlat*Pi/180)*1852*60, reslon * cos(minlat*Pi/180)*1852*60)
dx = 0.12 * cos(-38.9*3.14/180)*1852*60 = 2074m
with reslon the resolution along x in degrees
with maxlat the northern latitude in degrees
with minlat the southern latitude in degrees
reminder : 1 degree=60minutes // 1minute=1mile // 1mile=1.852km

The formula for the CFL time is :
Tcfl = dx / (g / (Fmin*4*Pi)) with the constants Pi=3,14 and g=9.8m/s²;
Tcfl = 2074 / (9.8 / (0.0373*4*3.14)) = 99s
maxTcfl ~= 90% Tcfl = 90s

Knowing the maxTcfl, the maximum global time step is usually set to 3 times bigger, it can be set up to 6 times bigger to save computation time despite a loss of precision :
Tglob = 3*maxTcfl = 3*90 = 270s
The refraction time step depends on how strong can be the wind or current on your grid, it is usually set to 6 times less the global time step but it can be decreased to 10 times less in case of strong wind or current:
Tref = Tglob / 6 = 270 / 6 = 45s
The minimum source terms time step is usually defined between 1s and 10s.
Tsrc = 1s

```
&TIMESTEPS_NML
 TIMESTEPS%DTMAX       =  270.
 TIMESTEPS%DTXY        =  90.
 TIMESTEPS%DTKTH       =  45.
 TIMESTEPS%DTMIN       =   1.
/
```

The **fourth section** to define is the grid information, it's usually the domain name followed by the spatial resolution, it must not exceed 30 characters.

```
&GRID_NML
 GRID%NAME          =  'ZA-7M'
 GRID%NML           =  'namelists_ZA-7M.nml'
 GRID%TYPE          =  'RECT'
 GRID%COORD         =  'SPHE'
 GRID%CLOS          =  'NONE'
 GRID%ZLIM          =  -0.10
 GRID%DMIN          =  2.5
/
```

The file namelists.nml contains the definition of the namelist corresponding to the switch used to compile the model. For a first test, you can leave it empty to use the default values. In any case it must ends up with line END OF NAMELISTS. For more details about how are defined the wet/dry points and the map status, refer to Appendix G.

The **sixth section** is the rectilinear grid definition, it contains the number of points, the grid resolution and the south-west corner of the grid.

```
&RECT_NML
 RECT%NX            =  89
 RECT%NY            =  94
!
 RECT%SX            =   0.12
 RECT%SY            =   0.12
 RECT%X0            =   12.0000
 RECT%Y0            =  -38.9000
/
```

The **last sections** correspond to the depth file, the mask file and the obstruction file :

```
&DEPTH_NML
  DEPTH%SF           =  0.00
  DEPTH%FILENAME     = '../data/ZA-7M.bot'
/

&MASK_NML
  MASK%FILENAME      = '../data/ZA-7M.mask'
/

&OBST_NML
  OBST%SF            =  0.01
  OBST%FILENAME      = '../data/ZA-7M.obst'
/
```

More namelists can be added for advanced features, like slope file for reflexion or sedimentary bottom file for friction, input boundaries, excluded points and bodies, ...

Appendix F – Create spec.list for ww3_bounc.nml from the GRIDGEN .bound file

To define the wave spectrum at the active boundaries of your grid, you need to define the list of spectral points to use from the base grid. These points are listed in the .bound file created by the create_boundary function from the GRIDGEN toolbox.

1- The simplest way is to download the spectra from our global hindcast, for example in 2015 with CFSR wind forcing for the South East regions :

```
wget -mnH --cut-dirs=4 ftp://ftp.ifremer.fr/ifremer/ww3/HINDCAST/GLOBAL/2015_CFSR/SPECTRA_SE
```

Then, to create the spec.list, run the script make_bounc.sh with the zone prefix in first argument and parent directory of the global spectra folder in second argument :

```
bin/make_bounc.sh  ZA-2M  GLOBAL/2015_CFSR
```

2- Another possibility is to first run a global model and output the spectra points listed in the .bound file and then list these spectra files into the spec.list

```
find $run_dir/GLOB-30M/work/SPEC_NC/GLOB-30M/  -name “spec.nc” >> spec.list
```

3- The last way to have wave spectra on the active boundaries is to run the model on a multigrid implementation using the ww3_multi. In this case, the active boundaries will be implicitly linked to the coarser grid points around.

Appendix G – How the map status and the wet/dry points are defined

Reading the bathy file, the depth value must have negative values under the mean sea level, a scale factor is applied (4th argument).

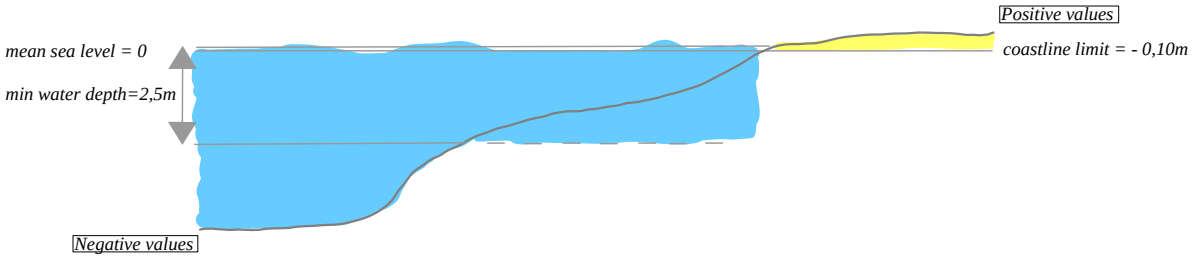
```
-0.10   2.50  40  0.001000   1  1 '(....)'  NAME  '../data/ZA-2M.bot'
```

The coastline limit (1st argument) is the value which distinguish the sea points to the land points. All the points with depth values (ZBIN) greater than this limit (ZLIM) will be considered as excluded points and will never be wet points, even if the water level grows over.

The minimum water depth allowed to compute the model (2nd argument) is the absolute depth value used in the model if the input depth is lower to avoid the model to blow up.

Note : The scale factor is not applied on the coastline limit and the minimum water depth.

Note : The depth value is not modified by the coastline limit and the minimum water depth.



Reading the mask file, a map status will be defined to force the grid points to wet or dry points.

```
                60                  1  1  '(....)'  NAME  '../data/ZA-2M.mask'
```

The input map status (MAPSTA) value in the mask file can be :

- -2 → excluded boundary points (sea points covered by ice)
- -1 → excluded sea points (sea points covered by ice)
- 0 → excluded points (land)
- 1 → sea points (ocean)
- 2 → active boundary points
- 3 → excluded
- 7 → ice

Note : The coastline limit can only overwrite the status of a sea point to a land point.

Example coastline on sea : for a negative or positive depth and a sea point status in the mask, if the coastline limit is about under this depth then it will be a land point.

Example coastline on land : for a negative or positive depth and a land point status in the mask, even if the coastline limit is above this depth it will stay as land point.

Example min depth : for a negative depth and a sea point status in the mask, if the min water depth is greater than the absolute value of this depth then the min water depth will be used to compute the model at this point.

An additional map status (MAPST2) is used to distinguish why the point is excluded in MAPSTA array:
IF MAPSTA = 0 :
MAPST2 = 0 → land point
MAPST2 = 1 → excluded point
IF MAPSTA < 0 :
MAPST2 = 1 → ice coverage
MAPST2 = 2 → point dried out
MAPST2 = 3 → land in moving grid or inferred in nesting
MAPST2 = 4 → masked in two-way nesting

To have only one map status in the output, the final map status (MAPTMP) will be a combination of these two maps status defined as :

```
MAPTMP = MAPSTA + 8*MAPST2
```

So, to retrieve original map status:

```
MASPSTA = MOD(MAPTMP+2, 8) - 2
MAPST2 = MAPTMP - MAPSTA
```

The final possible values of the output map status MAPTMP are :
-5 Other disabled point
-4 Point masked in the two-way nesting
-3 dry point covered by ice
-2 dry point, not covered by ice
-1 point covered by ice, but wet
0 land point
1 active sea point
2 active boundary point
8 excluded sea/ice point
7 = -1 + 8*1 -> mapsta=-1 et mapst2=1 -> excluded sea point, considered iced
15 = -1 + 8*2 -> mapsta=-1 et mapst2=2 -> excluded sea point, considered dried
31 = -1 + 8*3 -> mapsta=-1 et mapst2=3 -> excluded sea point, inferred in nesting
63= -1 + 8*4 -> mapsta=-1 et mapst2=4 -> excluded sea point, masked in 2-way nesting

Note : even if the point is set as excluded (7 or 15) in the mapsta, it can still be calculated if level or ice forcing change enough to set this point as sea point again.

Example wet to dry : for a negative depth in the bathy and a sea point status in the mask, the mapsta is 1 but it can be considered as a land point if the level forcing reduce enough.

Example dry to wet : for a negative depth in the bathy and a sea point status in the mask, the mapsta can be 15 if the level forcing at the first time step reduce enough the sea water level to dry this point.

Example dry to dry : for a negative depth in the bathy and a land point status in the mask, the mapsta is 0 and even if the level forcing grows, it will stay as an excluded point from the computation.

In the output of the model, the depth (DPT) is described as :

DEPTH = LEV – BATHY

in which the bathy is negative in the sea and positive on land, so the depth will be positive in the sea and a fillvalue on land. When the input water level (LEV) increase, it will increase the output depth (DPT) value. By the way, the input water level (LEV) forcing value will be stored in the water level above the sea level (WLV) output variable, this gives the possibility to retrieve the input bathy value at each grid point.

BATHY = WLV - DEPTH