WW3 Tutorial 2.1: Grid Generation

Purpose

The purpose of this exercise is to introduce users to a grid generation software called GRIDGEN (Chawla and Tolman, 2007) that has been developed specifically for generating grids that can be used for WAVEWATCH III applications. *Please note: The grid generation software is a series of MATLAB routines that are used to create ASCII grid files that are then provided to* **ww3_grid** *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*

Software

The GRIDGEN software is distributed as a tar compressed file.

```
gridgen.tar.gz
```

The first step is to change to home directory and move this file there. Uncompress and untar the file using the following commands

```
gunzip gridgen.tar.gz
tar -xf gridgen.tar
```

When uncompressed and untarred a gridgen directory should be created. This consists of three directories.

`reference_data/`
This directory contains the reference data that is used to create the grid. It consists of two types of data – the high resolution bathymetric data that are stored in netcdf format and the coastal boundary polygons that are stored (at different resolutions) as MATLAB binary files (.mat format).(***Note:*** The codes assume that the MATLAB version used has the ability to read netcdf files. This is part of the internal capability of MATLAB since v2009 (I think). If the particular version of MATLAB cannot read netcdf files then external programs for reading netcdf files can be found. Also the boundary polygons are currently limited to the ocean domain. Grids for lakes will have to be generated manually).

`bin/`
This directory contains the individual matlab functions that make up the gridgen software package. See the manual and/or documentation in each of the functions to learn more about what each individual function does.

`examples/`
This directory contains example matlab template files that show how the functions can be used to create the necessary grid files. Detailed comments are provided with the template files. These can be used as the starting points for creating grids. For the first exercise we shall step through the individual files for creating a grid.

The reference bathymetric data provided with this package are the global ETOPO1 and ETOPO2 data files in netcdf format. Users can use their own bathymetric data sets but will have to do one of two things. Either store the data in the same format as the provided netcdf bathymetric files, or change the algorithm of the bathymetry generating subroutine to accurately read the reference bathymetry. The coastal boundaries are stored Matlab binary files 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).

Generating a grid

From the home directory create a working directory where the grids for this exercise shall be generated. Change to this directory and launch matlab

```
mkdir test_grid
cd test_grid
matlab &
```

From here on the commands will be in the matlab environment. Most of the commands that we will go through step by step in this exercise are provided as template scripts in the examples directory.

Step 1 Defining the parameters for generating a grid

In all of the following steps $HOME shall refer to the users home directory. Individual users should replace $HOME with their full path. Set up the variables /path as follows

```
>> bin_dir = '$HOME/gridgen/bin';
>> ref_dir = '$HOME/gridgen/reference_data';
>> out_dir = pwd;
>> grid_box = [0 260 50 300];
>> dx = 0.5;
>> dy = 0.5;
```

The array `grid_box` identifies the lower left corner (in lat, lon) and upper right corners of the desired grid. In GRIDGEN the coordinate system for longitudes goes from 0 to 360 to comply with the way the coastal boundary polygons are defined. The variables `dx` and `dy` refer to the grid resolution in longitude and latitude respectively. All 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. To facilitate faster computations we shall use coastal

boundaries with intermediate resolution in this exercise. (Best practice is to build grids with the full resolution boundaries)

```
>> load([ref_dir,'/coastal_bound_inter.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 polygons on a figure

```
>> figure(1);clf;
>> for i = 1:41518
   plot(bound(i).x,bound(i).y);
   hold on;
   end;
```

The plot should look like Figure 1

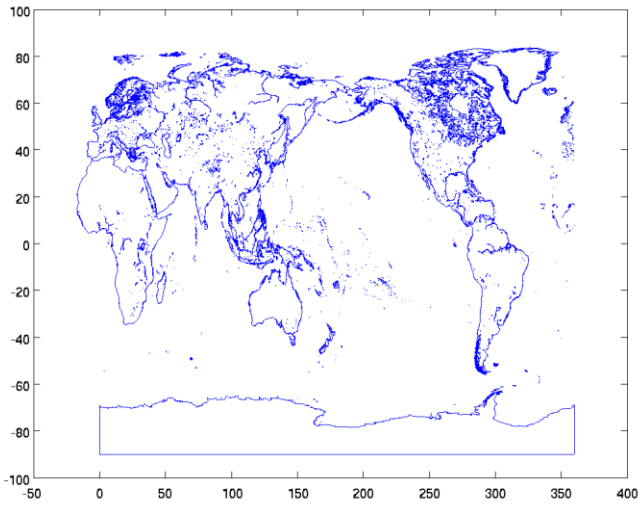

Figure 1: Boundary Polygons

Step 2 Generating a bathymetry grid

This is done using the function `generate_grid`. The raw bathymetric data is read from netcdf files in reference_data directory. The package comes with two sources of bathymetric data – **etopo1.nc** and **etopo2.nc**

```
>> prop=0.1;
>> cut_off=0;
>> dry_val=999;
>>[lon,lat,depth]=generate_grid(ref_dir,'etopo1',grid_box,dx,dy,p
rop,cut_off,dry_val);
```

Bathymetry data has been generated here using the **etopo1.nc** dataset. The variable `prop` identifies the proportion of reference cells that should be 'wet' to identify the bathymetric cell as 'wet' (and hence compute their depths). The variable `cut_off` identifies the water depth below which the cells should be identified as 'wet' and the variable `dry_val` identifies the bathymetric value assigned to all the dry cells. The function returns two one dimensional arrays with the longitudes (`lon`) and latitudes (`lat`) and one two dimensional array with the depth values (`depth`). The bathymetric data is in m and everything else is in degrees.

```
>> figure(1);clf;
>> d=depth;d(d==999)=nan;pcolor(lon,lat,d);shading flat;colorbar
```

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)

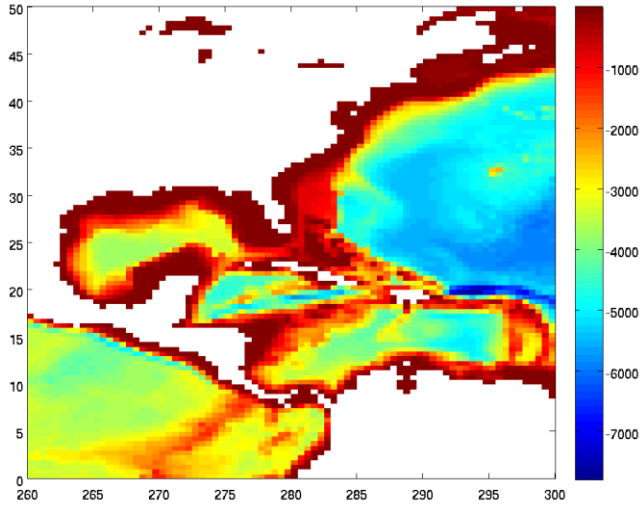

Figure 2: Initial Bathymetry

Step 3 Computing boundaries

The boundaries now need to be identified within the computational domain, properly accounting for boundary closure and splitting of boundaries. We start with identifying a domain that is a little bigger than the actual domain (to account for the cells at the edges of the domain).

```
>> lon_start = lon(1)-dx;
>> lon_end = lon(end)+dx;
>> lat_start = lat(1)-dy;
>> lat_end = lat(end)+dy;
>> coord = [lat_start lon_start lat_end lon_end];
```

Now we compute the subset of boundaries within the domain using the command `compute_boundary`

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

Here `b` returns a data structure array of the subset of boundaries that are generated using `N` and are the total number of boundaries generated. 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 3

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

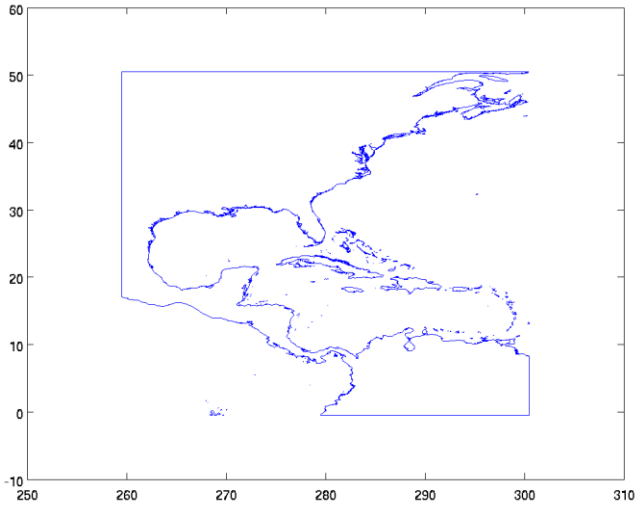

Figure 3: Coastal polygons within the boundary domain

Step 4 Computing the mask

The next step is to setup the land – sea mask. We start by using the bathymetry data set to identify the initial set of wet and dry cells.

```
>> m = ones(size(depth));
>> loc = depth == 999;
>> m(loc) = 0;
```

Plotting the initial mask should look like Figure 4

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

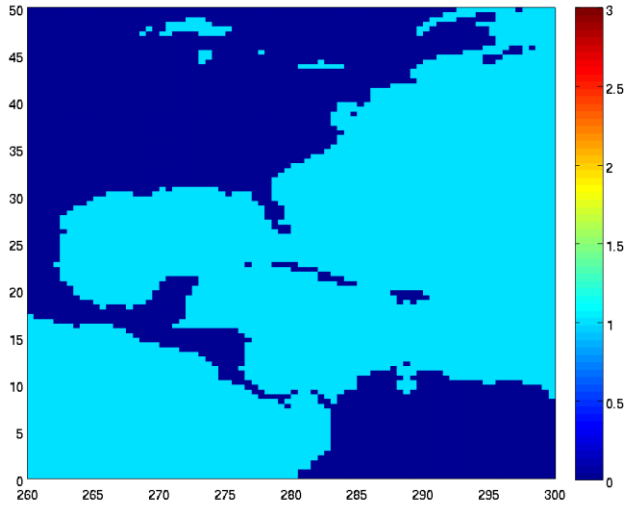

Figure 4: Initial Land - Sea mask

This initial set is not used as the final land – sea mask because the reference bathymetry and the coastal boundaries are not always consistent. That is why when we build the bathymetric data set we over emphasize the wet cells (determined by the variable `prop` in the bathymetric generation section). At this stage we clean up the land – sea mask using the coastal boundaries `b`. This is done using the `clean_mask` function. This function checks all the wet cells that lie inside a coastal boundary and determines if it should be switched from wet to dry. The function works only for switching the wet cells to dry and not the other way around since for each wet cell a corresponding depth would have to be determined as well. It is important to note that 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.

The function checks what portion of each wet cell lies within the boundary polygons. This is one of the more computationally intensive parts of the software and the time taken for any search depends upon the number of points making up a particular polygon (the more the points the longer the search). To avoid that a boundary splitting routine has been developed that splits up the boundary polygons to more manageable levels

```
>> b_split = split_boundary(b,2);
```

Where the second argument in `split_boundary` determines the width (or height) cut off limit (in degrees) above which the boundaries are split up into smaller chunks. Typical rule of thumb is to use a cut off limit that is at least 3 – 4 times the grid resolution. 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;
```

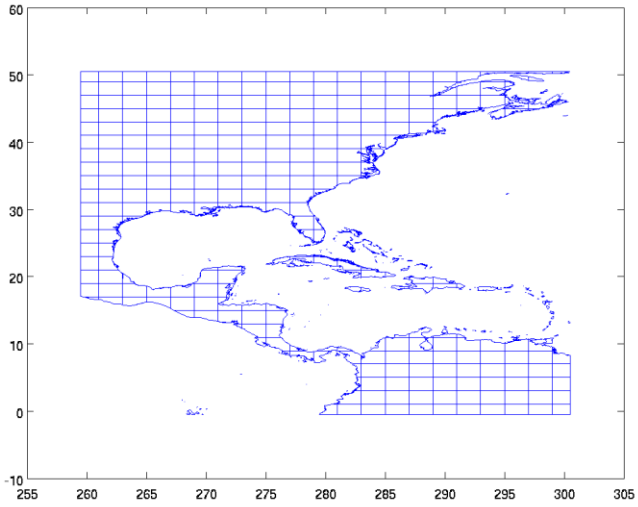

Figure 5: Split up boundary polygons

Now we are ready to run the clean up routine using the initial land – sea mask and the split boundary polygons. Once again like in `generate_grid` the function uses a variable to identify a cut-off below which if the cell domain lies inside the polygon it is marked dry.

```
>> prop = 0.5;
>> m2 = clean_mask(lon,lat,m,b_split,prop);
```

Plotting this cleaned up version of the mask looks like Figure 6. Note that compared to the initial mask (Figure 4) the cleaned up mask gets rid of the water bodies that make up the Great Lakes. This is because the Great Lakes are inside one of the coastal boundary polygons. Thus, one needs to be careful in trying to use this software for creating grids for lakes (such as Lake Victoria in the day 1 exercises).

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

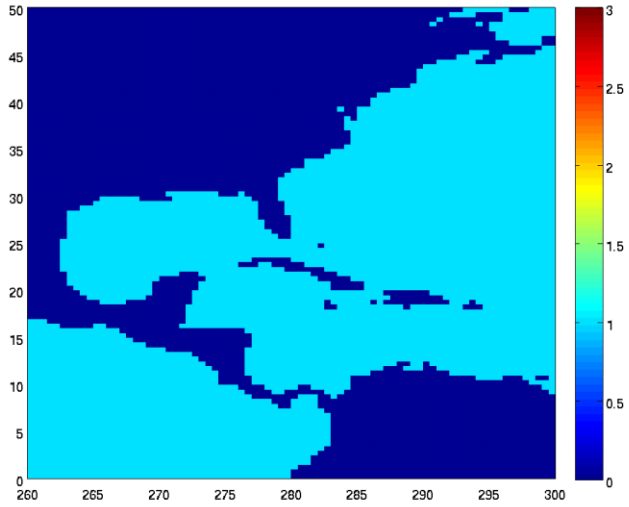

Figure 6: Mask after clean up

To get an idea of how much faster it is to generate the clean up routine using the split boundaries try to generate the mask using the original boundaries. (You can either take this time to catch up on e-mail, or after waiting for 10 minutes use CTRL-C to quit and proceed to the next step).

```
m3 = clean_mask(lon,lat,m,b,prop);
```

Step 5 Remove artificially generated lakes

The mask clean up is still not complete. Note that in Figure 6 we 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 (in this case part of the Pacific Ocean). To remove these we use a function called

```
>> cell_limit = -1;
>> glo = 0;
>> [m3,mask_map] = remove_lake(m2,cell_limit,glo);
```

This function finds all the different water bodies, and then uses the value of the variable `cell_limit` to determine what to do to the different water bodies. If `cell_limit` 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 with cells less than this are marked dry. The `glo` 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 and a two dimensional array with unique ids for the different water bodies. After running through this function the new mask looks like Figure 7

```
>> figure(1);clf;
>> pcolor(lon,lat,m3);shading flat;caxis([0 3]);colorbar
```

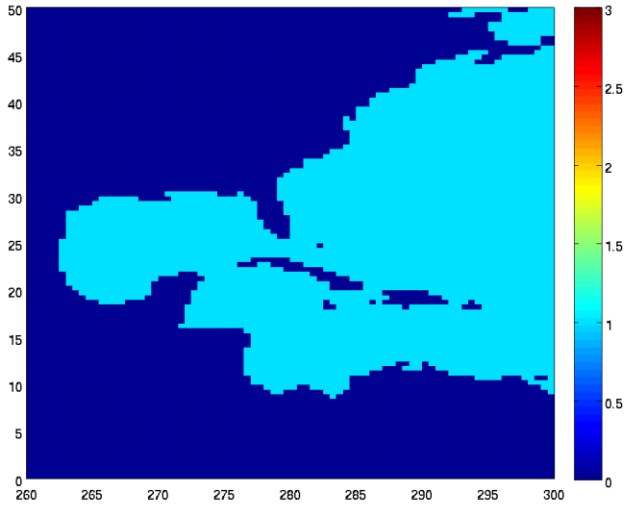

Figure 7: Land - Sea mask after clean up of separate water bodies

In this case by removing all but the largest water body, the Pacific Ocean has also been removed. This is not a problem if the focus of the computation is the Atlantic Ocean. However, there may be applications where both basins are desired (for example, if this grid is nested with a global grid). You can then use the water bodies identified in `mask_map` to turn specific ones back to water. Plotting the `mask_map` yields Figure 8

```
>> pcolor(lon,lat,mask_map);shading flat;colorbar
```

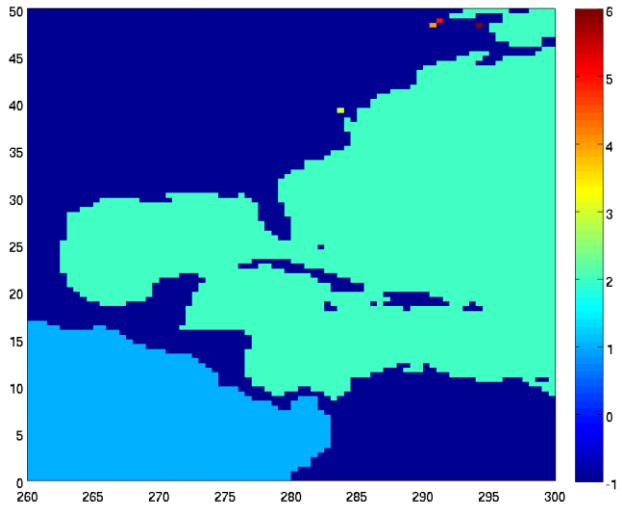

Figure 8: Different water bodies with unique ids

Here the Pacific Ocean has a value of 1. The Pacific Ocean can then be switched back to 1 (However for this application we shall keep the Pacific Ocean masked out)

```
>> m3(mask_map == 1) = 1;
```

Step 6 Generating obstruction grids

Obstruction grids are needed to remove energy from unresolved islands (Tolman, 2003). Once the land – sea mask has been adequately defined, together with the bathymetric data, the obstruction grid(s) can be determined for all the wet points. This is done using an algorithm that combines the land – sea mask with the boundary polygons (Chawla and Tolman, 2008). From the morning lectures we know that the obstruction grids can be generated by either just considering the obstructions in the cell itself, considering obstructions in one neighbor, or considering obstructions in both neighbors.

```
>> left = 1;
>> right = 1;
>> [sx1,sy1] = create_obstr(lon,lat,b,m3,left,right);
```

Here we have built the obstruction grids using both neighbors. Setting the variables (`left, right`) to 0 ignores the respective neighbors. The obstruction grids are given by Figure 9

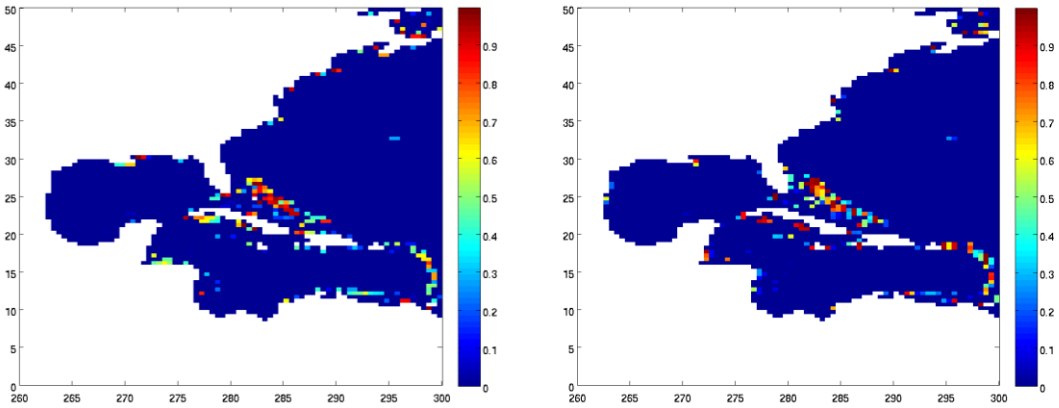

Figure 9: Obstruction grids along x (left panel) and y (right panel)

Step 7 Saving 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='watl_s';
>> depth_scale = 1000;
>> obstr_scale = 100;
>> d = round((depth)*depth_scale);
>> write_ww3file([out_dir,'/',fname,'.depth_ascii'],d);
>> write_ww3file([out_dir,'/',fname,'.maskorig_ascii'],m3);
```

```
>> d1 = round((sx1)*obstr_scale);
>> d2 = round((sy1)*obstr_scale);
>> write_ww3obstr([out_dir,'/',fname,'.obstr_lev1'],d1,d2);
>> write_ww3meta([out_dir,'/',fname],lon,lat,1/depth_scale,...
                                          1/obstr_scale);
```

There should now be 3 ascii data files for bathymetry, obstruction grids and mask. And one ascii meta file that shall be used to provide information to **ww3_grid.inp** file later. If you are building only a single grid model then this is the end of the GRIDGEN part. However, if you are building multiple grids then you will repeat this operation for each grid, and if the grids are communicating with each other, an additional step for boundary information.

Building another grid using the template script

Depending upon the application you are designing, you may have multiple grids that you need to work with (our operational global model uses 9 grids). Repeating the individual steps of the previous section can be a cumbersome task. Thus, GRIDGEN comes with example scripts to help do this in a convenient manner. For this exercise copy the `create_grid_regional.m` script from the examples directory to the working directory. We shall use this script to make a Northern Atlantic grid. Make the following changes to the script

- Change the path names to the appropriate directories
- Change the grid resolutions to 1 degree
- Change the boundary type to ‘inter’ (if you prefer you can experiment with different resolution types)
- Set the domain from (-10,250) to (75,355)
- Change the boundary splitting resolution to an appropriate number
- Change the file name prefix to ‘natl_1d’ (this will set the names for all saved files)

Run the script and sit back for the grid to be generated. You can try different boundary types, set the domain differently, make the Pacific basin wet, etc. Figure 10 shows the land – sea mask for the natl_1d grid that was generated using `create_grid_regional.m`

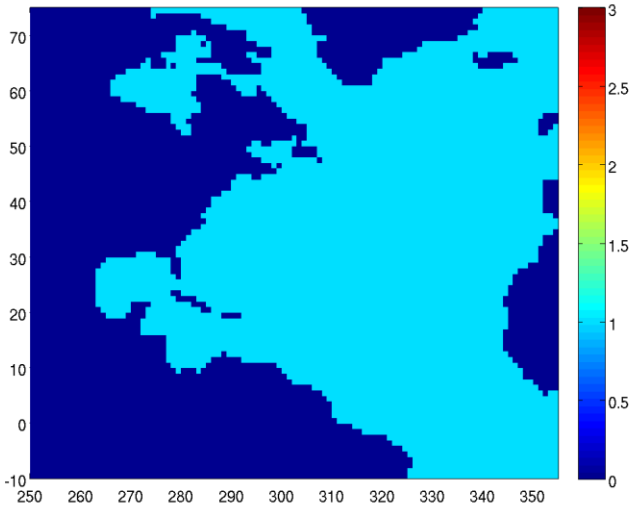

Figure 10: Mask for natl_1d

Setting up boundary conditions for multiple grids

In the multi-grid approach, the inner nest grids need the mask files changed to appropriately set up the boundary cells (where the data from the coarser grids is obtained). Till now we have been working with mask values of 0 (land) and 1 (water). However, there are two additional values for the mask file that can be set – 2 (boundary points) and 3 (excluded points). [***Note*** : The excluded points options are available only from version 3 and higher].At this point the user working directory should have two sets of grid files – `natl_1d.*` and `watl_s.*`.

Our aim is to develop appropriate points where boundary data from the coarser outer grid is provided to the finer inner grid. (Remember, that for the feedback from the finer grids to the coarser grids, the coarser grid values are overwritten by the corresponding inner grid values averaged over the whole coarser grid cell). 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. GRIDGEN provides a wrapper script in the `examples/` directory (`maskmod.m`). However, we shall proceed step by step in this exercise. 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, while the base grid is defined as the coarser grid from which the target grid is going to get information.

We can use the `*.meta` files to get information of the different grids and a function called `read_mask` to read in the actual mask arrays

```
>> [lon,lat] = read_ww3meta('watl_s.meta');
>> Nx = length(lon);
>> Ny = length(lat);
>> m = read_mask('watl_s.maskorig_ascii',Nx,Ny);
>> [lonb,latb] = read_ww3meta('natl_1d.meta');
>> Nxb = length(lonb);
>> Nyb = length(latb);
>> mb = read_mask('natl_1d.maskorig_ascii',Nxb,Nyb);
```

We now need to define the polynomial in the target grid along which the boundary information needs to be passed. If the aim is to use the entire grid then the polynomial can be defined outside the grid, in which case the boundary values shall be assigned to the edges of the grid. The polynomials can be complex to follow along the coastline or simple shapes. Here, we are going to define the polynomial such that the boundary points are defined at an angle inside the grid (there is no particular reason to do this, other than why not. You are more than welcome to try a different polygon if you so prefer).

```
>> px = [260 290 300 300 260 260];
>> py = [5 5 25 50 50 5];
```

Note that the only criterion is that the polygon be defined in a counter clockwise direction and be closed (this allows us to easily identify points inside, on and outside the polygon). We now use a function called `modify_mask` that sets all the mask values outside the polygon to undefined in the target grid (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.

```
>> glo = 0;
>> m_new = modify_mask(m,lon,lat,px,py,mb,lonb,latb,glo);
```

Where the `glo` flag is set to 1 if the base grid is global. For the polygon considered in this example the final mask is given by Figure 11. Note that the active cells along the eastern edge of the grid above 25 N are also 2 but cannot be seen because of the way the colors have been rendered in MATLAB. Compare this mask with Figure 7. You can now save this mask data using the function `write_ww3file` and saving the mask data to a new file name `watl_s.mask` (see previous section).

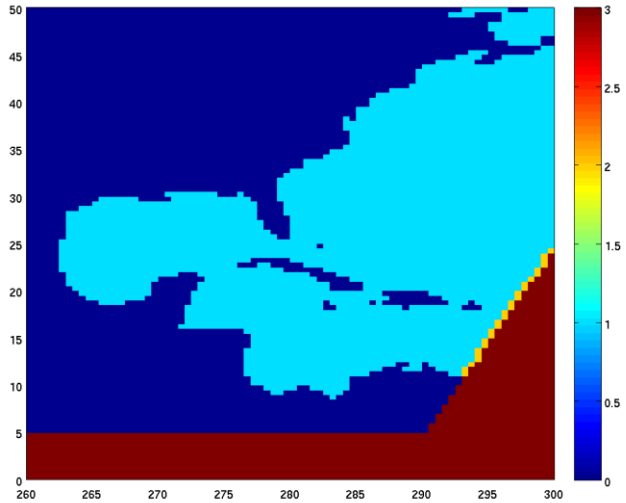

Figure 11: Final Mask with boundary points

Building overlapping grids of similar resolution

Till now we have learnt how to build individual grids and (for nested grids) how to set boundary conditions using the GRIDGEN software. WAVEWATCH also provides the capability of exchanging information across grids of the same (or similar) resolution. However, this is slightly more tricky than building the traditional set of outer (coarser) and inner (finer) grids. Since numerical computations inside the domain are third order and first order at the edge of the domain, for grids of the same resolution it is vital that boundary data from one grid do not overwrite the inner solution of the other grid. Thus, WAVEWATCH requires that in areas where grids of the same resolution are exchanging information, there is enough overlap to preclude that from occurring (otherwise the model will crash). In this section we shall determine how to compute the required overlap width.

The WW3 model uses a fractional time stepping technique to march forward in time. There are 4 different time steps in the model – a global time step (for the over all solution), a time step for spatial propagation, a time step for spectral propagation, and finally, a time step for source term integration. Details about the different time steps and how they are set will be covered in day 4. They are being mentioned here because to determine the overlap area we need to know the first two time steps. Of these two, the spatial time step is limited by the CFL stability criteria. For regular lat-lon grids this means that the time step will be determined by the cells at the highest (or lowest) latitudes.

Step 1 Deciding the grid parameters

One of the grids was already generated in the previous section (`watl_s`). This grid extends to 50^o N and has a spatial resolution of 0.5^o . Assuming that the first frequency component (longest resolved wave) in our modeling simulations is 0.035 s^{-1} (~ 30 sec wave period), and using the CFL stability criterion, yields that

$$\Delta t_p < 1392 \text{ s}$$

Where the subscript p has been used to denote time step for spatial propagation.

We shall now design a second grid for the northern part of the Western Atlantic (`watl_n`), covering the Canadian waters. Since we want this grid also to be nested inside the `natl_1d` grid that was designed in the previous section, let us limit the Northern edge of the grid to 73^o N. Again using the same stability criterion yields for

$$\Delta t_p < 633 \text{ s}$$

Step 2 Determining the overlap region

To determine the overlap region we need to set the time steps for the grids. The global time step is the time step at which the overall solution propagates and input forcings (wind/ice/water levels etc.) are updated . For this exercise we shall set

$$\Delta t_g = 1800 \text{ s}$$

for both the grids. And the spatial time step Δt_p for `watl_s` and `watl_n` to 1200 s and 600 s respectively. Computing the number of grid cells the boundary solution will propagate in one global time step for `watl_s` we get

$$\Delta t_g / \Delta t_p * \text{stencil\_width} = 4.5$$

and similarly for `watl_n`

$$\Delta t_g / \Delta t_p * \text{stencil\_width} = 9$$

where the `stencil_width` is 3 since we are using the third order propagation scheme in the model (see the manual). Thus the overlap width should include at least 9+4.5+1 (the additional cell is needed because we should have at least one common row/column for both grids where the solution from the two can be averaged).

Step 3 Building the grid

Since the overlap width needs to be at least 14.5 grid points, we shall set it at 16. This sets the bottom edge of `watl_n` at 42° N. We can now use `create_grid_regional.m` again to create the grids. Figure 12 shows the mask for `watl_n` that was generated by setting the western and eastern limits of the grid at 260 and 330 respectively.

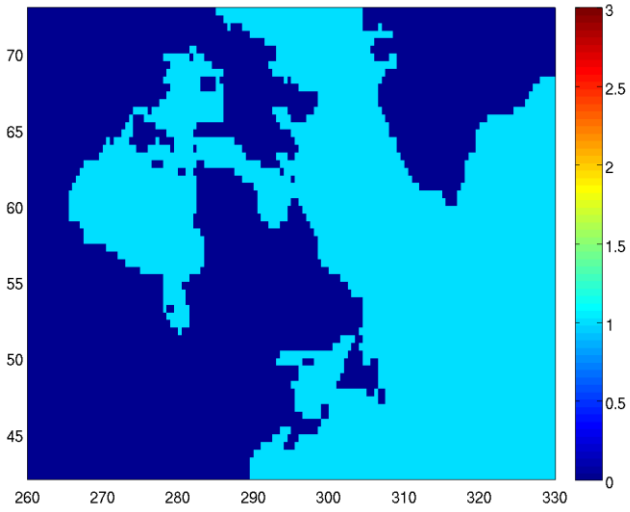

Figure 12: Mask for watl_n

A final check that needs to be done is that in the area of overlap the two grids should have similar masks (i.e. wet point in one should not be a dry point in the other). This matchup should be done before the boundary masks are set in the grids. In our exercise, we had already set the boundary masks for `watl_s` grid. Hopefully, the users saved the mask file to a different name so that they have the original mask file (if not then they can copy that from the tutorial section). Just like the previous section we can use the meta data and mask information to read array details.

```
>> [lon1, lat1] = read_ww3meta('watl_s.meta');
>> m1= read_mask('watl_s.maskorig_ascii',...
      length(lon1),length(lat1));
>> [lon2, lat2] = read_ww3meta('watl_n.meta');
>> m2= read_mask('watl_n.maskorig_ascii',...
      length(lon2),length(lat2));
>> [m1_out, m2_out] = reconcile_masks(m1,lon1,lat1,m2,lon2,lat2);
```

You can now compare the reconciled masks with the earlier masks to see where they are different. Even though the same software is used masks sometime in one grid or the other can be different if the cells are

close to being marked wet or dry (in our case if almost half the cell lies inside the polygons). Save these masks to new names.

```
>> write_ww3file('watl_s.mask_reconcile',m1_out);
>> write_ww3file('watl_n.mask_reconcile',m2_out);
```

At this point however the boundary information has to be generated again. You can either use the wrapper script `maskmod.m` from the `examples` directory, or repeat the steps in the section on boundary conditions. Just remember that the starting point for the mask files are the `*.mask_reconcile` files.

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.