Connectivity Modeling System User's Guide CMS v 2.0

Authors:

Claire B. Paris, Ana C. Vaz, Judith Helgers

August 2015

University of Miami

Rosenstiel School of Marine and Atmospheric Science

Department of Ocean Sciences

Paris' Lab: Physical-Biological Interactions Laboratory

Miami Florida 33149 USA

1. Introduction

Most coastal and benthic marine organisms have a pelagic larval phase, which might be their only opportunity to disperse with the currents and connect geographically separated populations. During the last decade, numerical models of the early life stages of marine organisms have emerged as a powerful tool for investigating the linkages and mixing between sedentary populations and the spatial history of successful migrants. These models are generally referred to as coupled bio-physical models. In such approaches, the early life of target species and their interaction with the environment is simulated by combining a stochastic biological model with ocean circulation models. These models typically use a Lagrangian particle-tracking framework to deal with explicit individuals, and use information on currents and environmental conditions from ocean circulation models to track the movement of a large number of individuals through space and time. In many practical applications, several hundred millions of abiotic or biotic particles need to be simulated by coupling their varying traits with an ocean circulation model. In addition, many particles (larvae) can be advected with the currents along very long distances. Tracking particles for longer time periods, from shallow coastal areas to the deep ocean and back, is typically solved by a tradeoff between domain extent and spatial resolution. It becomes thus critical to use simultaneously several distinct circulation models to evaluate coastal and ocean scale conditions to improve the accuracy of Lagrangian predictions. These requirements are often computationally complex and very demanding, with large data management efforts that limit the scope of the applications.

Connectivity Modeling System

CMS is a community multi-scale biophysical modeling system, based on a stochastic Lagrangian framework. It has been developed to study complex larval migrations and give probability estimates of population connectivity. In addition, CMS can also provide a Lagrangian descriptions of oceanic phenomena (advection, dispersion, retention) and can be used in a broad range of applications, from the dispersion and fate of pollutants to marine spatial conservation.

To facilitate community contributions to CMS release, a Beta version was created and made available to external collaborators and developers on August 1, 2011. The Beta version underwent extensive application and evaluation. The final release was made in December of 2012 with CMS version 1.0. This user guide covers some of the new developments since that official release, in what is tentatively called v2.

CMS operates off-line using archived ocean velocity data, and includes the following modules designed to gain efficiency, flexibility, and applicability:

Parallel Module

CMS is a parallel implementation of a stochastic particle-tracking model and provides the computational efficiency of evaluating a full range of transport and fate variability. It runs in parallel ensemble simulations of millions of passive and/or active particles over large time scales.

Multiscale Nesting Module

Particles can be tracked seamlessly over a series of nested multiple nested-grid ocean model domains. The nesting is independent from the ocean modeling system (e.g., nest 1 can be HYCOM data, while nest 2 is ROMS).

Landmask Boundary Module

CMS is particularly apt for moving particles along complex topography such as steep slopes and convoluted coastlines, avoiding the boundary with a variable spatial interpolation scheme of the velocity field around the particle. Alternatively, CMS provides the option to stop integration of the particles at the boundary (i.e., make landfall).

Vertical Movement Module

CMS has great flexibility in vertical movement by simulating various behavior and transport processes:

1) buoyancy (particle size/density, fluid temperature/salinity), 2) ontogenic vertical migration, 3) diel vertical migration, 4) tidal stream transport, 5) upwelling and subduction, and 6) in any combinations of these types of vertical behavior in the terminal velocity of individual particles.

Connectivity Matrix Module

CMS is a true matrix-based model as it couples units of the benthic and coastal seascape (Habitat Module) to the Lagrangian Stochastic Particle Module (LSPM) and particle behavior (Biological Module) to generate a transition probability connectivity matrix as output of the ensemble simulation.

Flexible Input

CMS can access on-the-fly ocean circulation data via OPeNDAP (Open-source Project for a Network Data Access Protocol) protocol or access locally stored files of archived blocks of data. CMS is also model-independent and can be coupled to ocean circulation model data on Arakawa grids (A, B, and C-grid), in a Cartesian or tilted format.

Flexible Output

CMS is designed to speed up the post-process of both oceanographic and connectivity applications by providing two types of output data 1) connectivity matrix output (source/sink pairs), and 2) trajectory output (location/status). The model can produce output on NETCDF or ASCII formats. In addition, CMS has the option to generate a single file of ensemble simulations (e.g., oceanographic application) or as multiple files corresponding to different initial conditions (e.g., connectivity application).

User's Guide

This User's Guide describes the model architecture and provides information for the setup and implementation of the OSS package. The basic idea is that the User can tell CMS where, when and how many particles he/she wants to release. In turn, CMS moves the particles using hydrodynamic fields and behaviors provided by the User, and outputs individual trajectories, fate, and a connectivity matrix of the ensemble of particles.

CMS is composed of two main executable programs to eliminate dependencies and duplications across the oceanographic and biological modules: 1) getdata accesses the ocean circulation data by reading the nest input files, and 2) cms moves and track the particles by reading the input parameter and configuration files. CMS package also provides some Matlab® code for output visualization and matrix-based post-processes.

Chapter 2 of this userguide describes the general workings and algorithms of the advective code, Chapter 3 explains how to install the code, and Chapter 4 how to run the code. Chapters 5 and 6 deal with setting up the parameters for the oceanographic data and for the IBM and other modules, respectively, while the output files are described in Chapter 7. Finally, the individual files in the code are briefly discussed in Chapter 8.

Developers

The Connectivity Modeling System (CMS) is the results of an interdisciplinary effort at the University of Miami by Claire Paris, division of Applied Marine Physics (AMP) of the Rosenstiel School of Marine and Atmospheric Science (RSMAS), Judith Helgers and Ashwanth Srinivasan from the Center for Computational Science (CCS) of the University of Miami, and Erik Van Sebille, division of Meteorology and Physical Oceanography (MPO), RSMAS.

Citation

We ask to make proper acknowledgement of the Rosenstiel School of Marine and Atmospheric Science, individual developers, agencies and funding agencies. One way to do this is to cite the following publication:

Paris CB, Helgers J, Van Sebille E, Srinivasan A (2013) Connectivity Modeling System (CMS): A probabilistic modeling tool for the multiscale tracking of biotic and abiotic variability in the ocean, Environmental Modelling & Software, http://dx.doi.org/10.1016/j.envsoft.2012.12.006.

Acknowledgements

We thank Matthieu Le Hénaff, Andrew Kough, Ana Vaz, Sally Wood, Dan Holstein, Erica Staaterman, and Karlissa Callwood for code testing and valuable suggestions. We thank the CCS for their help in releasing the OSS of CMS. Funding was provided by the National Science Foundation Biological Oceanography Program (OCE-1048697, OCE- 0928423, OCE-0825625) to C. B. Paris.

Open Source Licence

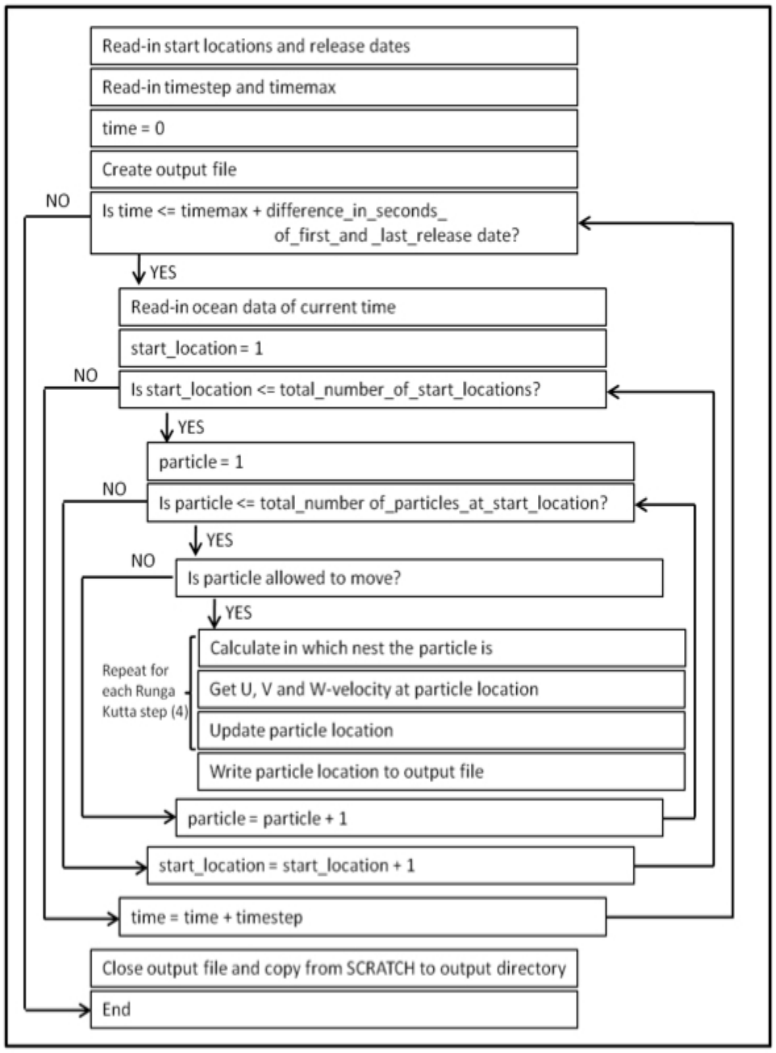
The CMS is an open-source model and licensed under the GNU Lesser General Public License. Here is a copy of the CMS model license file:

```
*                                                                         *
* and/or relevant publications listed at:                                 *
* http://www.rsmas.miami.edu/personal/cparis/publications.html            *
***************************************************************************
```

2. Algorithms

2.1 Flowchart

The basic flow chart of CMS looks as follows:


Read-in start locations and release dates
Read-in timestep and timemax
time = 0
Create output file
NO
Is time <= timemax + difference_in_seconds_of_first_and_last_release date?
YES
Read-in ocean data of current time
start_location = 1
NO
Is start_location <= total_number_of_start_locations?
YES
particle = 1
NO
Is particle <= total_number of_particles_at_start_location?
YES
NO
Is particle allowed to move?
YES
Repeat for each Runga Kutta step (4)
Calculate in which nest the particle is
Get U, V and W-velocity at particle location
Update particle location
Write particle location to output file
particle = particle + 1
start_location = start_location + 1
time = time + timestep
Close output file and copy from SCRATCH to output directory
End

2.2 Multiple Nests

CMS can track the three-dimensional movement of particles across nested domains by using different sets of overlapping circulation model domains (see Figure 1).

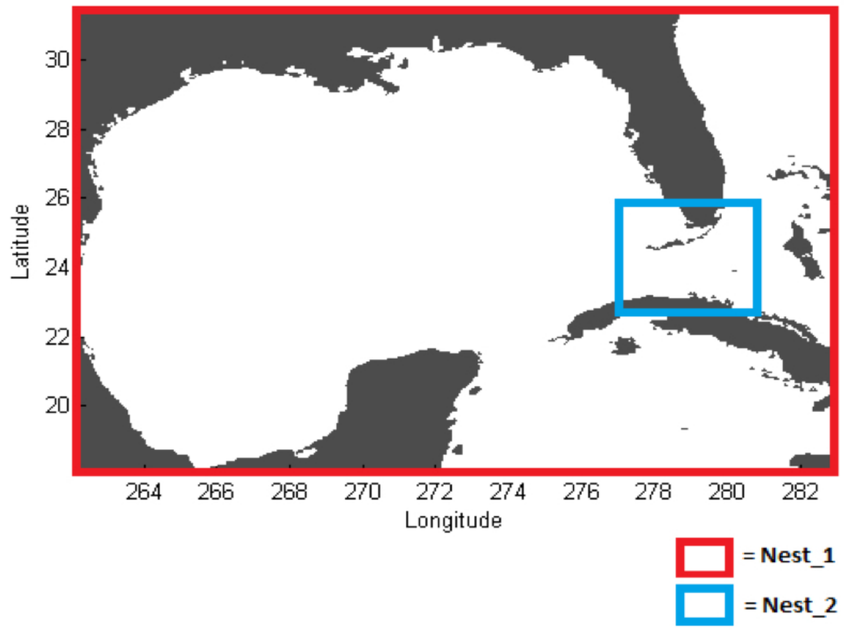


Figure 1. An example domain with two overlapping grids, which are called “nests” by CMS.

Motivation: In order to track the pathways of particles originating from shallow coastal areas or the deep benthic environment to the open ocean and back, it is important to use a series of multiple grids from ocean and coastal circulation models in a single application. Similarly, assessment of the transport requires that processes extending across oceanic scales and scales relevant to coastal dynamics be taken into account simultaneously. Requirements: the use of multiple nested-grid ocean models is useful if there is a high resolution circulation model domain overlapping a lower resolution domain. The “Multiple Nests” option requires that the models are at least one-way nested (i.e., the high resolution nest uses the boundary conditions of the larger domain); better performance can be achieved when models are two-way nested (i.e., information is exchanged between the nested-grid system).

2.3 Integration Method

The movement of particles in CMS are integrated using 3D velocity field of ocean circulation models. The velocity field can present three components: U (the zonal component or x-direction), V (the meridional component or y-direction), and W (the vertical component or z-direction). In essence, the distance travelled by a particle is calculated by multiplying each velocity component by a user-prescribed time step. Here, we use a 4th order Runge-Kutta scheme to calculate particle advection, and this scheme is applied both in space and time. Each 4th order Runge-Kutta step uses four local velocities: one at the initial particle position, and three others at trial positions. From these velocities the final particle position is calculated (Fig. 2). Suppose a particle starts at location (xold, yold, zold) for longitude, latitude and depth, respectively. The following algorithm calculates the new position (xnew, ynew, znew) of the particle:

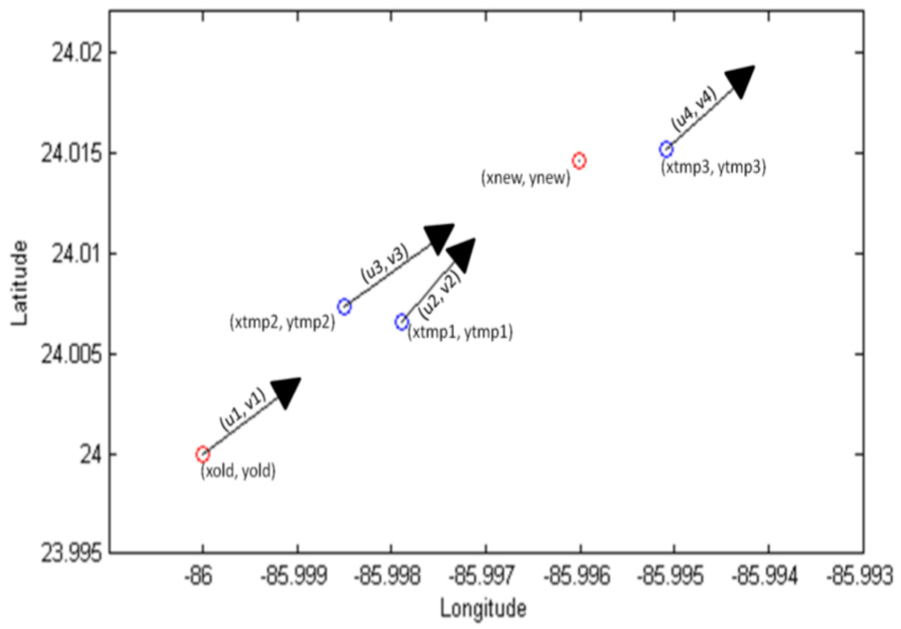


Figure 2. Example of the 4th order Runga-Kutta method in a 2D velocity field.

First step:

- Obtain u1, v1 and w1 from appropriated nest at position (xold, yold, zold) and time t

Second step:

- Calculate first trial position (xtmp1, ytmp1, ztmp1) from (xold, yold,zold) using u1, v1, w1 and 0.5*timestep
- Obtain u2, v2 and w2 from appropriated nest at position (xtmp1, ytmp1, ztmp1) and time t + 0.5*time step

Third step:

- Calculate second trial position (xtmp2, ytmp2, ztmp2) from (xold, yold,zold) using u2, v2, w2 and 0.5*timestep
- Obtain u3, v3 and w3 from appropriated nest at position (xtmp2, ytmp2, ztmp2) and time t + 0.5*timestep

Fourth step:

- Calculate third trial position (xtmp3, ytmp3, ztmp3) from (xold, yold,zold) using u3, v3, w3 and timestep
- Obtain u4, v4 and w4 from appropriated nest at position (xtmp3, ytmp3, ztmp3) and time t + time step

Final step:

- Calculate the final velocities uf, vf, wf:
    - uf=(u1 + 2*u2 + 2*u3 + u4)/6

    - vf=(v1 + 2*v2 + 2*v3 + v4)/6
    - wf=(w1 + 2*w2 + 2*w3 + w4)/6
- Calculate new position (xnew, ynew, znew) from (xold, yold,zold) using uf, vf, wf and at time t

The Fortran code of the Runga-Kutta method can be found in the module move.f90.

2.4 Interpolation Method for Position

There are two interpolation methods used to interpolate the water properties to the position of the particle, depending on the particle location on the model grid.

The CMS first choice is to use a tricubic interpolation (Fig. 3), where 64 neighboring nodes (4x4x4) are used. The tricubic interpolation is done for each dimension of the model output, by using a third-degree (cubic) polynomial. The cubic polynomial is fitted to the circulation model values found on the four closest grid points of each dimension. The resulting polynomial is then used to calculate the variable value on any point between grid points.

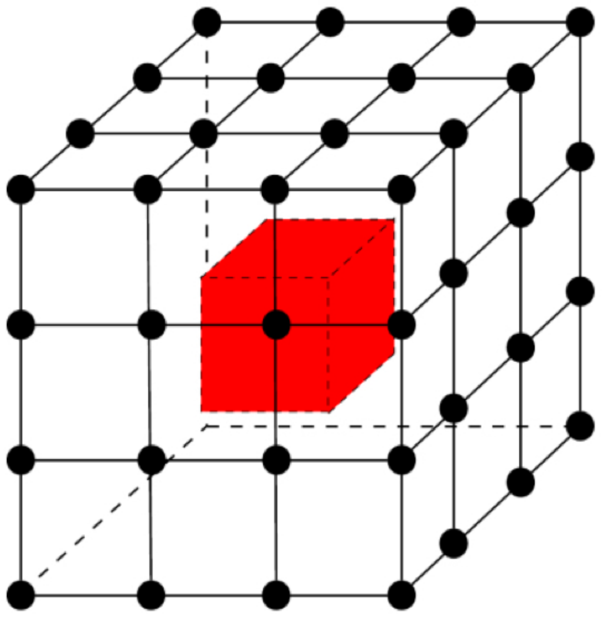


Figure 3. The 64 neighboring grid points used in the tricubic interpolation if the particle is located within the red area.

Note that, by definition, the tricubic interpolation can only be performed if the 64 neighboring points are located on ocean. If at least one of the points lies on land, then a trilinear interpolation will be used, where only 8 neighboring points (2x2x2) are needed (Fig. 4).

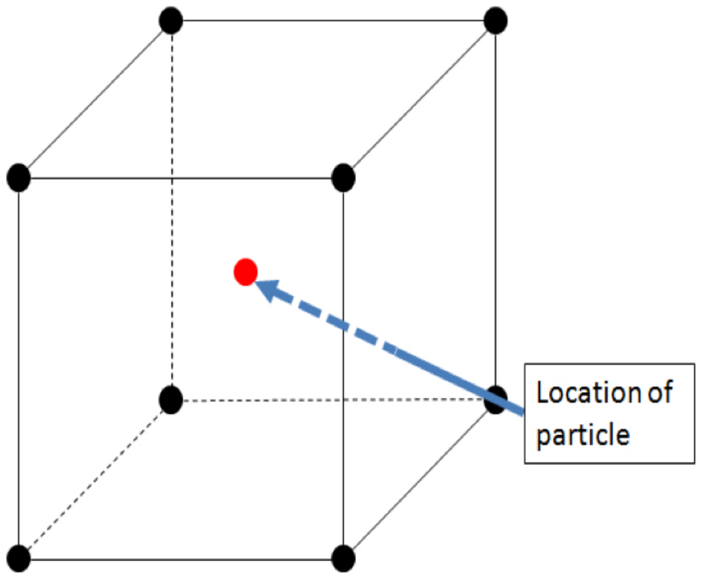


Figure 4. The 8 neighboring points used in the trilinear interpolation if the particle is located at the red dot.

Since CMS can be used for different applications, ranging from simulating abiotic particles to marine organisms, it is necessary that particles can have the ability to avoid being washed ashore. For example, fish larvae are able to avoid getting stranded on the bathymetry, while plastic or oil particles would reach the coast and be washed ashore or settle at the bottom. Thus, the user can chose if their particles will be able to avoid reaching land by setting up the flag avoidcoast in the input file runconfig.list. If this flag is set to false, particles will be able to reach land, and will be removed from the simulation if more than one of the 8 neighboring points required for the triliniar interpolation is located on land. However, if the flag is set to true, the particles will continue to move, by using the nearest velocities available to the particle (as will be described on Chapter 6.1).

If using 2D velocity fields (without a depth dimension), then a faster interpolation is used. In this case, CMS will perform a bicubic interpolation (Fig. 5) if the 16 nearest grid points to the particle are located on water. Otherwise, a bilinear interpolation is applied, for each only 4 neighboring grid points located on water are needed.

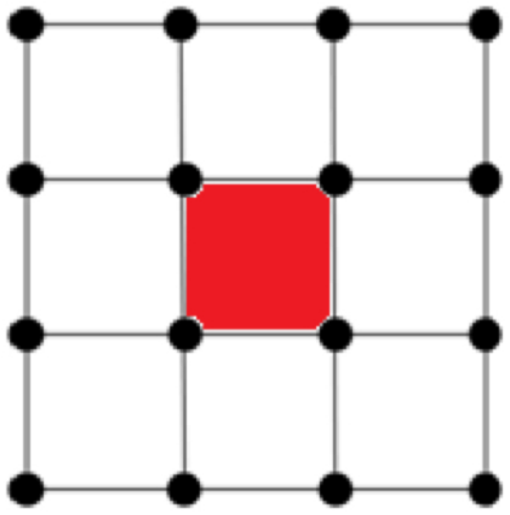


Figure 5. The 16 neighboring points if the particle is located within the red area.

The Fortran code for the interpolation can be found in the module fldinterp.f90.

2.5 Interpolation Method over Time

Oceanographic model files can be stored at different time intervals, thus it is necessary to interpolate the velocity files considering their time resolution, so the advection of particles on times in-between different oceanographic files can be adequately represented. If the velocity files are available at time intervals smaller than months (e.g., weeks, days, hours our seconds), CMS interpolates velocity fields from different files linearly over time.

For example, if a particle was advected for 35640 seconds since uvel1 was loaded (as marked by a red cross on Fig. 6), and the time interval among velocity files is one day (86400 seconds), the U-velocity at the present time-step will be given by:

```
uvel = 35640/86400 * uvel1 + (86400-35640)/86400 * uvel2
```

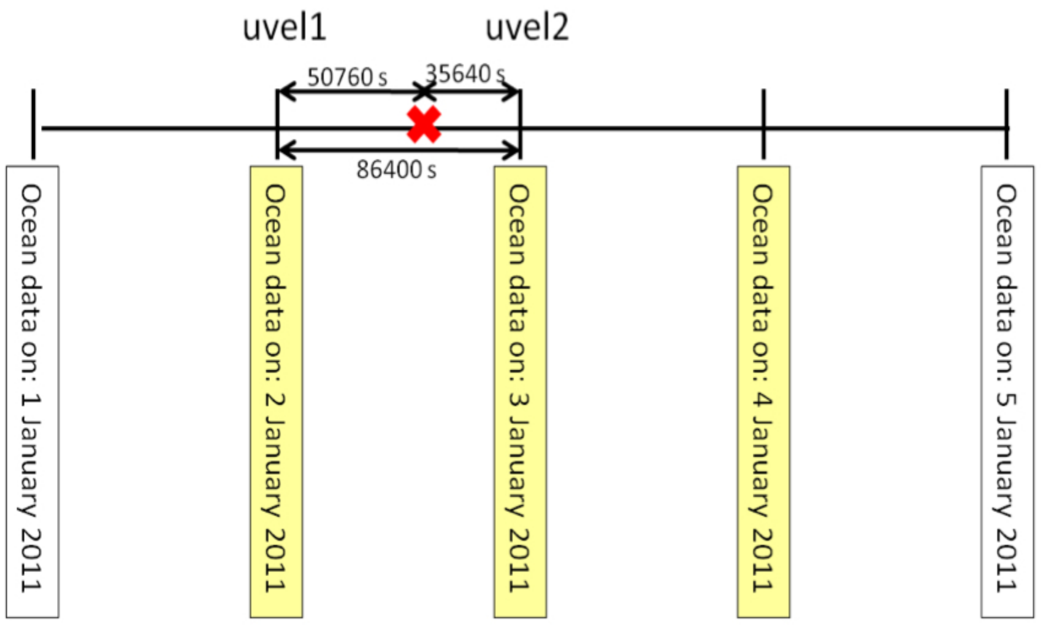

uvel1
uvel2
50760 s
35640 s
86400 s
Ocean data on: 1 January 2011
Ocean data on: 2 January 2011
Ocean data on: 3 January 2011
Ocean data on: 4 January 2011
Ocean data on: 5 January 2011

Figure 6. Example of the linear interpolation scheme for instantaneous velocity fields between consecutive snapshots

The Runge-Kutta method uses velocities from the present time and from future time-steps. Thus at each CMS time-step, three velocity files for each nest are stored in the computer's memory. In the example above, these three files would be for January 2, 3 and 4. If one of these files does not exist, CMS cannot move the particle, which will then be removed from the simulation with an exitcode of -5.

If the nest files are stored in monthly intervals, then four files for each nest will be used by the CMS to interpolate oceanographic properties.

If the particle is on the time represented by the red cross on Figure 7, then the U-velocity at that time is given by:

```
x   = 20/31 (fraction of days advected)
x1  = 11/31 (fraction of days to advect until next file)
uvel= (x1*(1+x*(1-1.5*x))*uvel2) + (x*(1+x1*(1-1.5*x1))*uvel3) + (-0.5*x*x1*x1*uvel1) +
(-0.5*x1*x*x*uvel4)
```

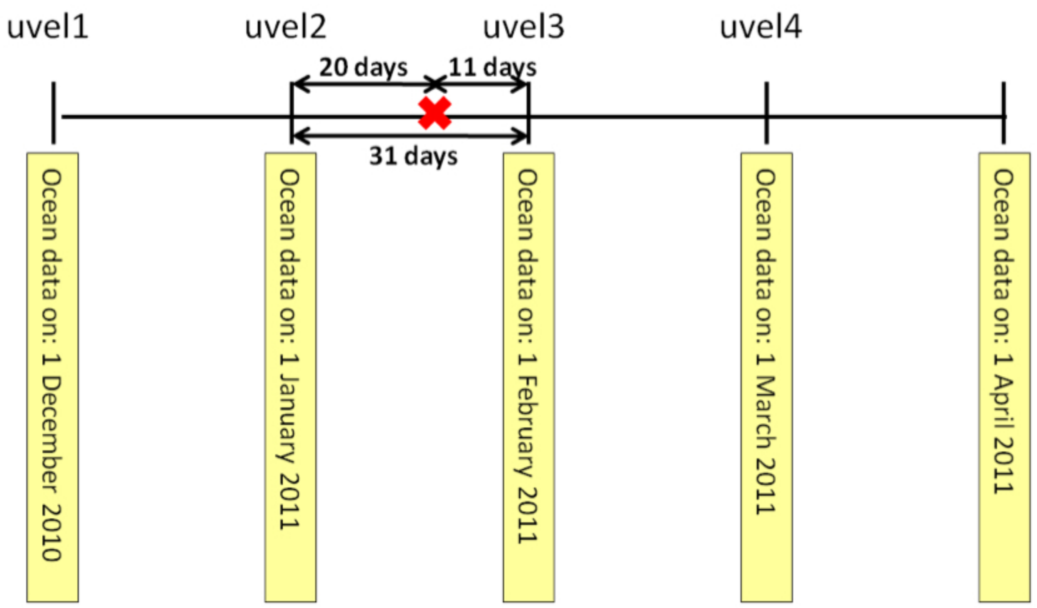

uvel1
uvel2
uvel3
uvel4
20 days
11 days
31 days
Ocean data on: 1 December 2010
Ocean data on: 1 January 2011
Ocean data on: 1 February 2011
Ocean data on: 1 March 2011
Ocean data on: 1 April 2011

Figure 7. Example of the scheme used to interpolate between ocean velocity files stored at monthly intervals.

When using monthly output, CMS will store in memory five temporal files of each nest. Again, if one of the five files does not exist, the particle will be removed from the simulation with an exitcode of -5.

The Fortran code for the interpolation in time can be found in the module getphysicaldata.90.

3. How to install CMS

Before running CMS you have to have some programs installed:

Fortran 90 compiler with MPI binding (mpif90):

Most of CMS code is written in Fortran 90. The Fortran code uses the MPI Message Passing Interface so you need to install the mpif90 compiler. MPI is used to run the CMS on different processors. The MPI code that CMS uses is:

```
! MPI include file
use mpi
! initialise MPI
call MPI_INIT(ierr)
! how many processors are there in total?
call MPI_COMM_SIZE(MPI_COMM_WORLD, npes, ierr)
! what is the id of my processor?
call MPI_COMM_RANK(MPI_COMM_WORLD, my_id, ierr)
! quit MPI
```

```
call MPI_FINALIZE(ierr)
```

If you only have a Fortran 90 compiler that does not support MPI then it is not possible to run the CMS on different processors but you still can use the CMS. To use the CMS with a Fortran compiler without MPI remove all the above MPI code in the file cms.f90.

The CMS is tested with the mpif90, gfortran and ifort compiler.

C-compiler:

One file written in C so a C-compiler is also required. The CMS is tested with the GCC compiler.

NetCDF version 4.1 or later with DAP support enabled

CMS uses NetCDF (.nc) files as input and output, thus requires to install the appropriate NetCDF libraries. The libraries can be found at: http://www.unidata.ucar.edu/software/netcdf/ CMS also can use OPeNDAP (Open-source Project for a Network Data Access Protocol) to download ocean model output.

With the OPeNDAP software, you access ocean model outputs using a URL. Versions of netCDF 4.1 (or higher) enable to access data through OPeNDAP servers. For this, use the --enable-dap flag to enable DAP support when configuring netCDF. DAP support is automatically enabled if a usable curl library can be located using the config flag --with-curl-config, or it can be enabled using the --enable-dap flag, in which case you still need to provide access to the curl library.

After the above programs are installed and working properly, the CMS package can be copied to the computer.

CMS directory structure

CMS consists of three directories:

- /arch : the directory with a file that is the part of the Makefile with the compiler options.
- /src: the directory with the CMS source code.
- /expt: the directory containing the experiments (input and output files of CMS)

The directory */src* contains the source files, and you will need to build the CMS. Included with the source it is a file called *Makefile*, which specify how to derive the target program. You will need to adjust *Makefile* to your computer settings (as described bellow). After configured, you only need to use the command make, which automatically builds executable programs and libraries from source code.

Configuring Makefile

Follow the following steps to make the CMS ready for use: Open the file *mpi_compiler* in the */arch* directory. If necessary, change the values of the following parameters:

```
FC: fill in the Fortran compiler that is used to create the object files
FCFLAGS: fill in the flags that the Fortran compiler needs to create the object files
CC: fill in the C-compiler that is used to create the object files
CCFLAGS: fill in the flags that the C-compiler needs to create the object files
LD: fill in the Fortran compiler that is used to create the executable. In most cases
this will be the same Fortran compiler as you filled in at FC.
LDFLAGS: fill in the flags that the Fortran compiler needs to create the executable from
the object files.
```

```
EXTRALIBS: fill in the extra libraries needed to compile the program
```

Go to the directory */src*. In this directory there is a *Makefile* file.

Type the command make to compile the code, and to create all the object files and executables. If needed, use the command make clean to remove all object files and executables.

Next you need to create a link to the executable file in the */expt* directory, by using the command:

```
ln -s ../src/cms
```

Also, create a link to the program that downloads circulation model outputs by typing:

```
ln -s ../src/getdata
```

CMS is now ready to run from the /expt directory.

How to run CMS

The first step to run CMS is to prepare all input files on the formats required by CMS. Here it is a brief overview of CMS files organization, necessary input files and output formats. Detailed information about these is available on Chapters 5 and 6.

Input and output files for different experiments are stored on the directory */expt*. Each experiment should have a corresponding input_ and expt_ directories (as illustrated on Figure 8). Input directories store the experiment configuration files, while expt stores the oceanographic files (nest files) and the CMS output (the results of your experiment). The corresponding experiment_name directories are automatically created by running *getdata*, or the directories can be created by the user.

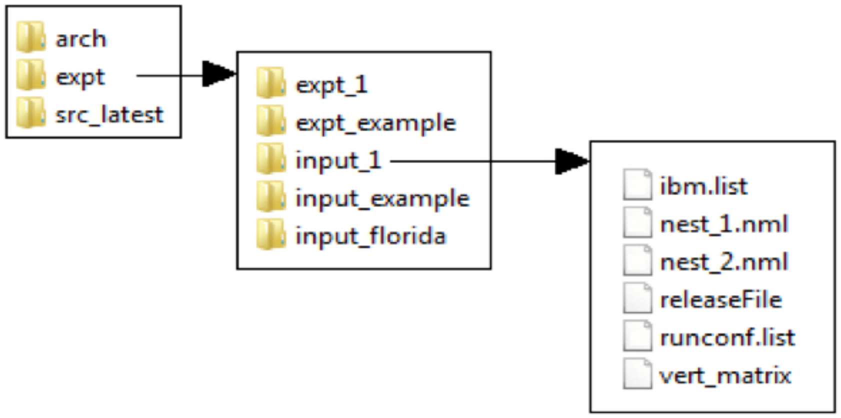


Figure 8. A tree structure for the the /input_name directory.

There are 4 input files that must be present on the input directory to run CMS: 1) configuration files for the oceanographic files, 2) configuration file for the experiment, 3) file with behavior option for particles, and 4) a release file:

- *nest_number.nml* (for example *nest_1.nml*) contains the parameters for downloading oceanographic files and creating nest files which will be used by CMS (see Chapter 5).
- runconf.list is the file with the configuration for running CMS (see Chapter 6).

- ibm.list: it is the file with behavior options for particles (see Chapter 6).
- The release file contains the positions, times and depths where particles will be released (See Chapter 6).

There are some input files which are optional and are also explained in Chapter 6.

The files found on the /expt_ directory are stored on three sub-directories:

- /nests: circulation model files
- /output: CMS output files
- /SCRATCH : output files that failed to finish. This directory should be empty if the CMS run was successful (i.e, no errors were found)

After the input files are set up, you can run the model. For this go to the directory /expt and type:

```
./cms name_of_inputfile_directory
```

The name of the input file directory tells CMS which input files to use.

For example:

```
./cms example
```

In this case, CMS will use the input files stored on *input_example* to run the simulation, and the output files will be saved on the directory */expt_example/output/*.

It is also possible to run the model on multiple processors to reduce the running time. To achieve this, instead of typing the command

```
./cms name_of_inputfile_directory
```

you should type :

```
mpirun -np number_of_processes ./cms name_of_inputfile_directory
```

For example:

```
mpirun –np 8 ./cms example
```

In which case CMS will run on 8 processors.

CMS handles multiprocessing by at startup distributing the lines in the release file (see Chapter 6, ReleaseFilename) over the number of processors. If the number of lines is less than the number of processors, the CMS will not run.

5. Oceanographic files

Before you can run CMS you must obtain circulation model files, which will be prepared in a format specific for the CMS. The hydrodynamic files will be stored in nest files, while the parameters for creating and reading the nest files is contained on *nest_x.nml* files (for example *nest_1.nml*).

There are two ways to create the nest files. You can use the program included in the CMS package *./getdata*, or you can adjust the *nest_1.nml* file for the parameters of nest files that you have created. Using the *./getdata* method is straightforward, and particularly useful if you wish to download files stored on online servers. However, it can only handle files written in Arakawa A and B grids. Moreover, *./getdata* will interpolate any circulation model output stored on B grids onto A grids.

The manual method is more versatile, as it doesn't require regridding the files to an Arakawa A grid. However, it is also slightly more difficult to set up, and can't be used to download files over OPeNDAP. The method is quite similar to the Local Files method of section 5.3, but there are some small differences. These are explained in the last section of this chapter.

5.1 Using getdata

Included with the CMS package is a program called *getdata*, which can download and save oceanographic files in formats suitable for CMS. You can use getdata to:

- download oceanographic files from a internet server using OpenDAP, or
- convert oceanographic files stored in your computer to the CMS format.

To use *./getdata* you need to create an input file with the parameters for downloading the oceanographic data (or with the parameters of the local files), as it will be described bellow.

5.2 OPeNDAP

The program *./getdata* can use OPeNDAP to access the ocean data. The files will be downloaded in separate nestfiles in the directory */expt_name/nests/* (for example/ expt_example/nests)

The names of the downloaded files are:

```
nest_nestnumber_yyyymmddhhmmss.nc
yyyy    = year
mm  = month
dd  = day
hh  = hour
mm  = minutes
ss  = seconds
```

For example:

nest120101004010000.nc is the oceanographic data for the first nest at 10/4/2010 at 1am.

5.3 Local files

If the data is already downloaded, you do not need OPeNDAP. In this case, you have to place the data in the directory: */expt_name/nests/raw* (for example */expt_example/nests/raw*). It is important that the files are in NetCDF format and that the names of the files are in the same format as described at OPeNDAP (*nestnestnumberyyyymmddhhmmss.nc*). So each file should only contain the data of one snapshot.

./getdata will adapt the files for the CMS format. All modified files will be written in */expt_name/nests/* (for example */expt_example/nests/*). If all your fields are in different files, you have to add an extra letter to the file to tell the model which field is in the file.

For example:

```
nest120101004010000u.nc
u = u-velocity
v = v-velocity
w = w-velocity
s = salinity
t = temperature
d = density
ssh =sea surface height
```

5.4 Raw oceanographic files

Raw oceanographic files on the OPeNDAP server or in the local files must meet the following constraints:

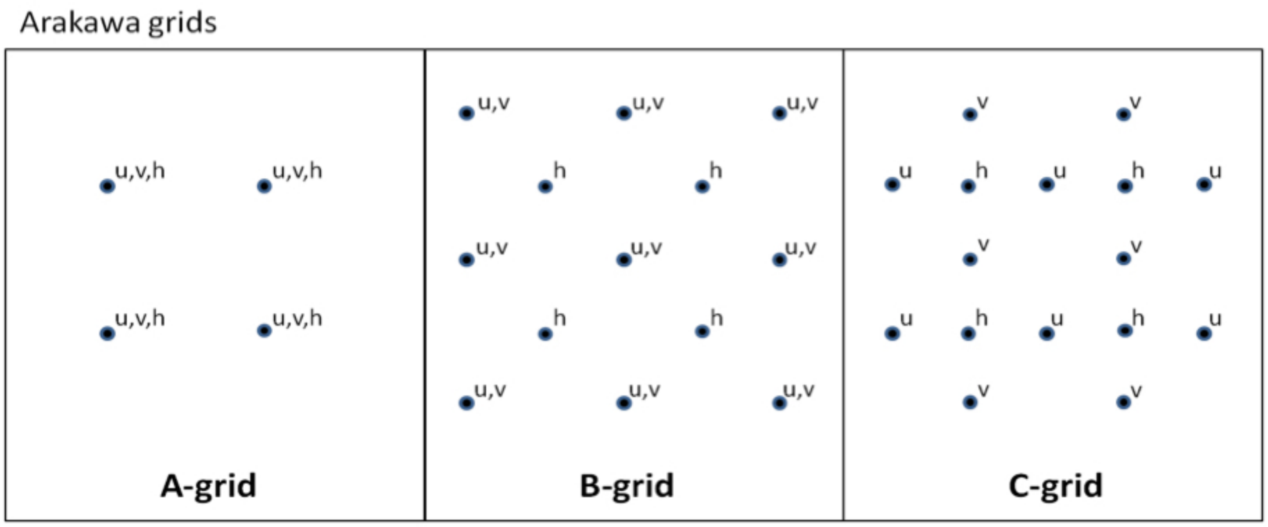


Figure 9. Three different Arakawa grids. The native grid on which the CMS works is the Arakawa A grid,

where all variables are given on the same location. However, the CMS can automatically regrid B-grids to A-grids.

The variable **agrid** in *nest_x.nml* determines whether the oceanographic variables are provided on an A grid. When you use *getdata*, raw oceanographic files have to be in A-grid or B-grid and **agrid** has to be set to **.true.**

Getdata automatically convert B-Grid variables to A-grid. You can also use oceanographic files with variables stored on C-grids on CMS. In this case, set the **.agrid.** to **.false.**, and the files will be read directly by the CMS (not using *getdata*, as explained on the last section of this chapter).

Longitude (required)

- unit: degrees
- difference between the smallest and largest values should be less than 360 degrees
- can present one or two dimensions, ordered as (Latitude, Longitude)
- must be on ascending order

Latitude (required)

- unit: degrees
- values varying between -90 and 90;
- can present one or two dimensions, ordered as (Latitude, Longitude)
- if latitude is two dimensional, values must be on ascending order

Depth

- unit: meters, centimeters or kilometers
- accepted as positive or negative;
- must be ordered from surface to bottom;

Time

- unit: seconds, days, hours or months from a specific date;
- must present regular time-step;
- get data cannot handle missing data;

U and V-velocities (required)

- if using four dimensions, must be ordered as (Time, Depth, Latitude, Longitude)
- if using three dimensions, must be ordered as (Time, Latitude, Longitude)

Optional fields: w-velocity, salinity (PSU only), temperature (degree Celsius only), density (kg/m^3 only), and sea surface height (meters only). If used they must present the same structure as the U- and V-velocity.

5.5 Regridding two-dimensional files

The CMS code works with one dimensional longitudes and latitudes. When using latitudes and longitudes with two dimensions,*./getdata* will regrid then to one dimension (as shown on Figure 10).

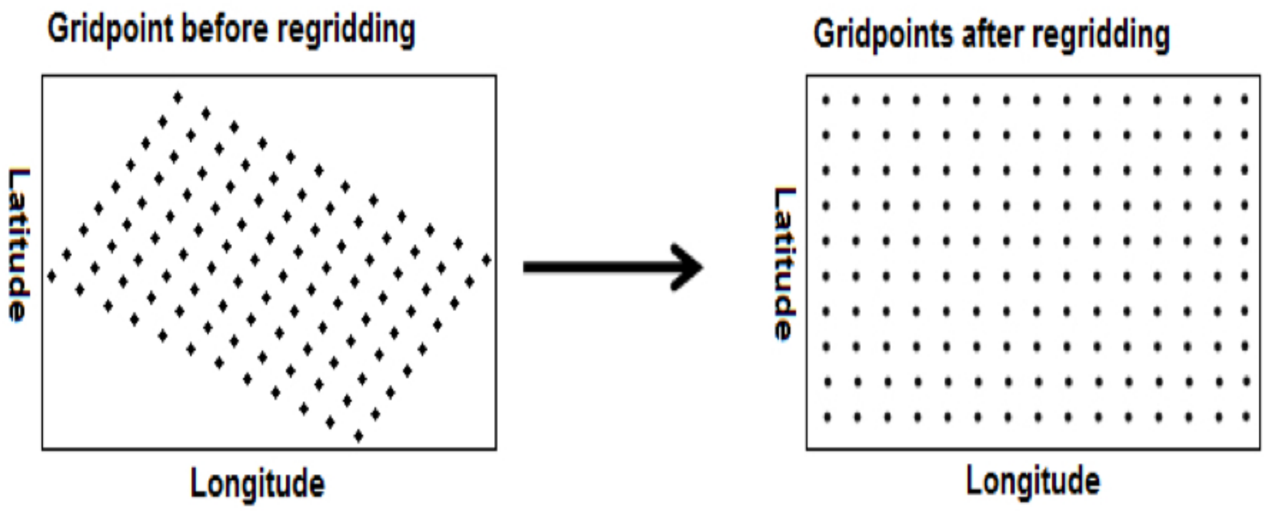


Figure 10. Example of the re-gridding done by ./getdata if the original grid is tilted.

The Fortran code for re-gridding can be found in the module **mod_getdata.90**.

5.6 Nest_x.nml using OPeNDAP

The parameters for downloading the ocean data are stored on the NEST files:

```
nest_x.nml
```

It is possible to use multiple nested-grids from ocean models, there is no limit on number of grids. However, there has to be one *nest_x.nml* for each nested-grid domain.

The following is an example for nest_1.nml:

```
&nest_input
filename = "http://tds.hycom.org/thredds/dodsC/glb_analysis"
xaxis = "X"
yaxis = "Y"
zaxis = "Depth"
taxis = "MT"
xstart= 270
xend   = 290
ystart= 20
yend   = 29
zstart= 0
zend   = 100.00
zaxis_positive_direction= "down"
```

```
tstart_yy = 2010
tstart_mm = 1
tstart_dd = 1
tend_yy   = 2010
tend_mm   = 2
tend_dd   = 1
time_step = 86400
time_units= "days"
jdate_yy  = 1900
jdate_mm  = 12
jdate_dd  = 31
lon_name  = "Longitude"
lat_name  = "Latitude"
depth_name= "Depth"
depth_conversion_factor=1
time_name = "MT"
uvel_name = "u"
vvel_name = "v"
wvel_name = ""
wvel_positive_direction = ""
velocity_conversion_factor=1
dens_name = ""
temp_name = ""
saln_name = ""
ssh_name  = ""
angle_file= ""
fill_value= 1.2676506E30
agrid     = .true.
$end
```

Nest input fields description

```
filename
```

The url of the server where the oceanographic files are stored, written between double quotation marks. The CMS will use the OPeNDAP protocol (through getdata) to access the oceanographic files.

If you want to read the oceanographic files from your computer (stored on nests/raw), then you would remove this line.

```
xaxis, yaxis, zaxis, taxis
```

The names of the file dimensions

- xaxis : X dimension. E.g.: Longitude, lon, i, X
- yaxis: Y dimension. E.g. : Latitude, lat, j, Y
- zaxis: Z dimension. E.g.: Depth, Layer, Z

- taxis: T dimension. E.g.: Time, MT, T

The names of the dimensions can be found by typing in a web browser the name of the server where the data is archived with .html at the end.

For example:

```
http://tds.hycom.org/thredds/dodsC/GOMl0.04/expt_20.1/2004.html
```

will result on:

```
u[MT = 1464][Depth = 7][Latitude = 361][Longitude = 437]
```

thus:

```
xaxis = Longitude, yaxis = Latitude, zaxis = Depth, taxis = MT
```

If there is no depth, the line with the zaxis has to be removed from the nest.nml file.

The names of the dimensions always need to be written between double quotation marks.

You need to make sure that this range is on the server or you will get the following error message: *'The data for longitude is not available. Choose a different value in the nest file'*.

To see which values there are on the server, type: name of the server plus .asc?lon_name.

For example:

```
http://tds.hycom.org/thredds/dodsC/GOMl0.04/expt_20.1/2004.asc?Longitude
```

```
xstart,xend
```

Give the range of the longitudes to download your variables

```
ystart, yend
```

Give the range of the latitudes to download your variables

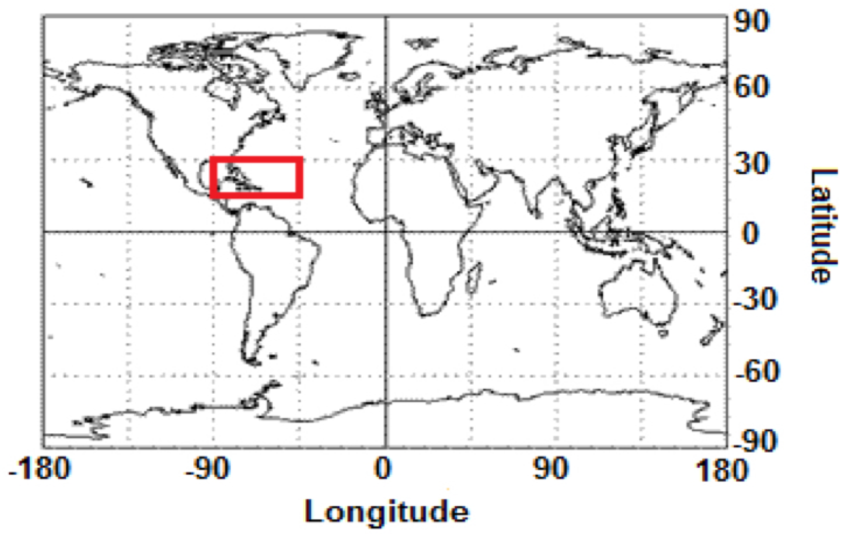


Figure 11. Area (red square) that will be downloaded if xstart = -90, xend = -60, ystart =-15, yend =30. If xstart > xend then another part of the world will be downloaded (see Figure 12)

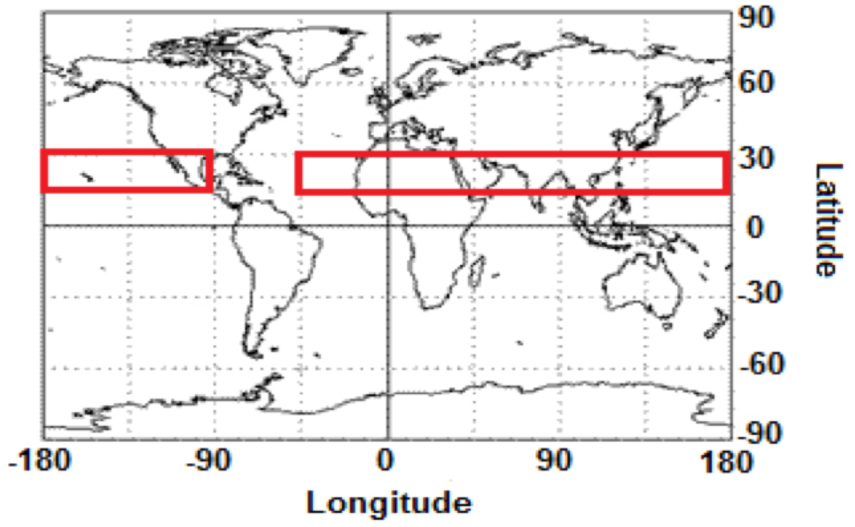


Figure 12. Area (red square) that will be downloaded if xstart = -60, xend = -90, ystart =-15, yend =30

```
zstart, zend
```

Depth range to download the oceanographic variables

Must be given in meters and ordered from surface to bottom. Thus zstart will always be smaller than zend.

If the depths are negative then use positive numbers for zstart and zend but set the parameter *zaxispositivedirection* to *“up”*.

```
zaxis_positive_direction
```

Gives the direction of the depth axis. The only accepted values are *“down”* or *“up”*

If depths are positive (0,100,250,500): *zaxispositivedirection* = *“down”*

If depths are negative (-0,-100,-250,-500): *zaxispositivedirection = "up"*

```
time_units
```

Gives the interval between stored oceanographic files. Four values are accepted: *"months", "days", "hours" or "seconds"*;

> For example:
>
> http://tds.hycom.org/thredds/dodsC/GOMl0.04/expt_20.1/2004.htm
>
> you see that at the *time_name* it says:
>
> *"units: days since 1900-12-31 00:00:00"*
>
> so this means that the *time_units* are *"days"*

```
time_step
```

the *time_step* is the time difference between oceanographic variables time_step is always given in seconds and must be uniform.

To see which values are on the server, type the name of the server plus .asc?time_name.

> For example:
>
> http://tds.hycom.org/thredds/dodsC/GOMl0.04/expt_20.1/2004.asc?MT
>
> you will obtain
>
> 39083.0, 39083.25, 39083.5, 39083.75, 39084.0, 39084.25,
>
> meaning that the oceanographic data is stored on quarter day intervals.
> The time_step in this example is equal to 0.25*86400 = 21600.

If the time_units is one month, then use a time step of 2635200.

```
tstart_yy, tstart_mm, tstart_dd, tend_yy, tend_mm, tend_dd
```

Time to be downloaded

> For example:
> if
> tstart_yy = 2008,
> tstart_mm = 8,
> tstart_dd = 1,

> tend_yy = 2008,
> tend_mm = 8,
> tend_dd = 30
> then oceanographic files will be downloaded from 1 August 2008 to 30 August 2008.

```
jdate_yy, jdate_mm, jdate_dd
```

The time of the oceanographic file given by time_name is calculated from an "origin" data

> For example:
> Looking at:
>
> > http://tds.hycom.org/thredds/dodsC/flkeys.html
>
> you can see that at the time_name says:
>
> > "days since 1900-12-31 00:00:00"
>
> This means that jdate*yy = 1900, jdate*mm =12 and jdate_dd=31
> When using local data these three parameters are not used, and these three lines can be removed from the nest file.

```
lon_name, lat_name, depth_name, time_name
```

The names of the longitude, latitude, depth and time fields on the server.

If there is no depth in the data then the line with the *depth_name* has to be removed, or *depth_name* ="".

Names of variables always have to be written between double quotation marks.

```
depth_conversion_factor
```

Unit of depth must be given in meters.

If the unit is different, the depth need to be recalculated to meters.

The recalculation is done using the parameter depth*conversion*factor:

```
depth_new = depth_old * depth_conversion_factor
```

> For example:
> If the unit of depth is m then the depth*conversion*factor is 1.
> If the unit of depth is cm then the depth*conversion*factor is 0.01.

```
uvel_name, vvel_name, wvel_name
```

uvel_name: name of the zonal velocity component

vvel_name: name of the meridional velocity component

wvel_name: name of the vertical velocity

The names of the velocities need to be written between double quotation marks.

Velocities are necessary to move the particle, thus *uvel_name* and *vvel_name* must be given for CMS to run a simulation. The value of the *wvel_name* is optional, and if not using w-velocity you can remove the line with the *wvel_name*.

```
wvel_positive_direction
```

wvelpositivedirection is defined by either *"upward"* or *"downward"*.

the *wvelpositivedirection* is *"upward" *if a particle will move towards the surface with a positive W-velocity;

the *wvelpositivedirection* is *"downward"* if a particle will move toward the bottom of the sea with a positive W-velocity;

```
velocity_conversion_factor
```

u, v and w velocities must be given in meters per second (m/s).

If the velocity unit is different, the velocities will be recalculated in m/s using the parameter *velocityconversionfactor*:

```
velocity_new = velocity_old * velocity_conversion_factor
```

For example:

Unit velocity	**velocity conversion factor**
m/s	1
cm/s	0.01
km/s	1000
mph	0.44704

```
dens_name, temp_name, saln_name, ssh_name
```

dens_name, *temp_name* and *saln_nanme* are the names of the density field, temperature field and salinity field on the server, respectively;

Variable names need to be written between double quotation marks.

These are optional variables, and you can remove the lines with the variable names that you do not want to download.

It is important to note that temperature and salinity fields are used to compute the particle's buoyancy. Thus if using the buoyancy or mass spawning flags the nest files need to include these variables.

The density field of the ocean is calculated with the temperature and salinity fields, by the following formula (REF?):

```
c1=-1.36471E-01, c2= 4.68181E-02, c3= 8.07004E-01, c4=-7.45353E-03, c5=-2.94418E-03, c6=
3.43570E-05, c7= 3.48658E-05

density = (c1+c3*saln+temp*(c2+c5*saln+temp*(c4+c7*saln+c6*temp)))
```

The coefficients can be changed in the Fortran module util.f90.

```
angle_file
```

If the grid of the original file has an angle with respect to the Cartesian grid, then the grid will be re-gridded to a straight Cartesian grid, by using the original angle. The inclination (angle) of each grid point must be given in a netCDF file. The parameter angle_file is the name of this netCDF file without the nc.

```
fill_value
```

The value of the points with no velocity (for example land or landmask points).

5.7 Nest_x.nml if using local files

If you already have all oceanographic files stored on your computer, you have to place the oceanographic files in the directory:

```
/expt_name/nests/raw
```

> For example: /expt_example/nests/raw.

NOTE: It is important that the files are netCDF files and that the names of the files have the correct format. There must be one file for each snapshot. The nest.nml follows almost the same configuration as when using the OPeNDAP method, however, the line containing the filename has to be removed when using local files

the field *time_units* are not used for local data;

the fields *jdate_yy*, *jdate_mm* and *jdate_dd* are not used, therefore they can also be removed.

5.8 Download.txt

In the same directory where all the *nest* files are stored, there is also a file called download.txt, which is created for each nest. This file saves a list of the NEST files that were downloaded by *./getdata*. This way, if there is a server crash while downloading the oceanographic files,*./getdata* can start downloading only new files. If you wish to remove your nest files, remember to remove this file as well.

> For example, if you see the message: “Data file: 2007-4-25 06:00:00 already exists”, then this data file will not be downloaded again.

5.9 Not using getdata

As of CMS v1.1, users can also directly provide their hydrographic data in the nests folder (so not in raw, and not using getdata). Advantage of this method is that the hydrographic files in any grid (A, B or C) can be used. If you are using native B or C grids, set the **agrid *variable in nest_x.nml to *.false.**, and if using an A grid, set it to **.true.**

Since on C grids, the zonal and meridional components of the velocity are given at different locations, if using this method you will need to provide at least two nest files for each time step, one for u and one for v, named *nest1yyyymmddhhmmssu.nc* and *nest1yyyymmddhhmmssv.nc*.

The date format yyyymmddhhmmss is the same date string as also used by getdata.

You can also add vertical (w), temperature (t), and salinity (s) files as needed. So for each time step for each oceanographic files are available, you will need at least two files (and up to 5 different files). Each file will store only one field (u, v, w, t or s).

For each of your variables, you can define different variable names for longitude, latitude and depth. The names of the variables dimensions can be set in the *nest_1.nml* file as *lon_nameU*, *lat_nameU*, *dep_nameU*, *lon_nameV, lat_nameV*, *dep_nameV*, *lon_nameW*, *lon_nameT,* etc.

The names of the variables themselves can also be set in *nest_1.nml* as *uvelname, vvelname* etc. If not using agrid, CMS will use all these values in nest_1.nml to determine how to read the fields.

CMS uses the *tstartyy, tstartmm*, and *tstart_dd* in *nest_1.nml* to determine when to start the run, so it is important to also provide these (see earlier in the section).

In agrid-is-false-mode, CMS can also handle netcdf files with multiple snapshots in a file. Easiest way to get CMS to work with files that have for instance five snapshots, is to create five symbolic links to the same file, following the normal CMS calendar convention. Do this for every file. CMS will then automatically iterate through the individual snapshots in a file. It is important to note, however, that the **tsart** values (see above) point to the first snapshot of a file, and the **tend** values point to the last snapshot.

Currently, work is underway to generalize the grids that CMS can handle even further. As of yet, the grid needs to be “orthogonal”, meaning that the values for longitude have to be independent of latitude and vice versa. A common Mercator grid will work in CMS but more complicated tri-polar grids do not yet work. Use *getdata* if you want to work with these.

6. Other Input Files

6.1 runconf.list

The run_conf file contains the values for variables that are necessary to run the model. It also contains a list of options for turbulence, forward or backward advection mode, behavior of the particle at boundaries, and output format.

The following is an example of the file runconf.list:

```
&runconf
!===============================================================!
nnests          = 1
timeMax         = 8640000
timestep        = 2700
outputFreq      = 43200
releaseFilename = "releaseFile"
!===============================================================!
!Turbulence Module
turb            = .false.
horDiff         = 1          ! horizontal diffusivity (m2/s2)
vertDiff        = 0.5        ! vertical diffusivity (m2/s2)
turbTimestep    = 2700       ! in seconds
!===============================================================!
!Periodic Boundary Condition
periodicbc      = .false.
!===============================================================!
!Landmask Boundary Condition
avoidcoast      = .false.
!===============================================================!
!Backward Tracking Module
backward        = .false.
!===============================================================!
!Output files in ASCII instead of netCDF
ascii           = .false.
!===============================================================!
!Flag for limit particle vertical movement to upper layer
upperlevelsurface    = .true.
!===============================================================!
!Flag for looping through velocity files
loopfiles            = .false.
loopfilesstartyear   = 1980
loopfilesstartmonth  = 1
loopfilesstartday    = 5
loopfilesendyear     = 2006
loopfilesendmonth    = 12
loopfilesendday      = 29
!===============================================================!
!Options for save restart files
restartfromfile    = .false.
restartwritefreq   = 432000
!===============================================================!
```

```
!Options for mixed layer physics
mixedlayerphysics  = .false.
mixedlayerwmax     = 0.1
!==============================================================!
&end
```

Description of runconfig.list

Number of nests

```
nnests
```

nnests is the number of nests to be used

- For each nest there has to be a nest.nml file.
- nnests needs to be a positive integer number.
- There is no limit on the number of nests to be used

Time of simulation

```
timeMax (in seconds)
```

Defines how long particles from each release should move.

- timeMax should be given in seconds
- must be a positive integer number

Simulation time step

```
timestep (in seconds)
```

The time step used for the calculation of new particle position

- timestep must be given in seconds and be a positive integer number
- Make sure that 86400s (1 day) divided by timestep is also an integer number.

Output frequency

```
outputfreq (in seconds)
```

Defines the time interval to save trajectories information (longitude, latitude, depth, and other parameters of traj_ files).

- Must be given in seconds
- Must be a positive integer number
- **outputfreq** should be equal to or a multiple of **timestep**

File with release information

```
releaseFilename (mandatory)
```

The name of the file with the release information.

Without this file there is no simulation

The release file is a separate file containing the information to set up particle release, including the exact position (longitude, latitude and depth), time (year, month, day, seconds), and the number of particles to be released.

It can also include the polygon where the particle is released (if using the flag **.polygon.**).

For example:

Release polygon	Longitude	Latitude	Depth	Number of releases
1	277.2	24.6	1	10
10	277.4	24.6	1	20

Release year	Release month	Release day	Release seconds
2008	08	02	3600
2008	08	10	0

Make sure the following is correct:

- the depth is given in meters
- there must be a oceanographic file (nest file) for the release year, month and day
- If you are not using polygons then the first column is a number

Turbulence Module

```
turb
```

A random component can be added to the motion of the particles to represent the subgrid-scale motion unresolved by the model. This “random kick” can be added by setting the flag **turb** to **.true.** in the file *runconf.list*. The turbulent velocity is added to the horizontal velocities (U and V-velocities), following the random walk or random displacement method (RDM) described by Okubo (1980, Diffusion and Ecological Problems: Mathematical Models. Springer, Berlin):

```
uturb = ((2 * horDiff)/turbtimestep)^0.5 * random_number
```

- horDiff : horizontal diffusivity (m^2 /s). You can enter a different horizontal diffusivity for each nest, by separating different values with a comma. This value should be scaled by the grid size of your model (e.g., see Okubo 1971 DSR)
- vertDiff: the vertical diffusivity (m^2 /s). You can enter a different vertical diffusivity for each nest and separate the different values with a comma.
- turbtimestep: the time (seconds) after which the random component of the velocity is added to the particle movement.
- random_number: a Gaussian random number between 0 and 1 (calculated by CMS)

turbTimestep tells CMS how often the turbulent velocity is calculated. It can be defined as equal to or a multiple of **timestep**. If **turbTimestep** is not specified, CMS will use a **turbTimestep** equal to **timestep**.

If the flag **turb** is set to **.false.**, turbulence will not be added to the velocities and the model will run deterministically. In this case, a single particle can be released for each initial condition (IC).

Periodic Boundary Condition

```
periodicbc
```

Motivation: For global water mass transport studies where particles need to recirculate or for population connectivity studies in the Pacific, the particles need to cross the boundaries of the global nest file.

The flag **periodicbc** can be added to the file *runconf.list* and can have two values: **.true.** or **.false.**

If **periodicbc** is **true** then the x direction is periodic and a particle can loop around the nest in the longitude direction.

Avoid landmask boundary

```
avoidcoast
```

Motivation: Marine larvae of fish and of some invertebrates do not get stranded on the topography but are capable to swim away. Alternatively, pollutants or organic matter can settle on the seafloor or be washed ashore.

When is a particle on land?

Before explaining how CMS prevents particles to come on land, it is necessary to define when a particle is on land. The land gridpoints always have the same U, V and W-velocity values, defined as 2100 (= fillvalue). `CMS checks the surrounding eight points of the location of the particle in a trilinear interpolation (see Figure 3)

The variable countLand stores how many of the eight grid points are land points. So countLand can have a value between 0 and 8.

CMS also defines the distance from each of the eight grid points to the location of the particle. A particle will be considered on land if the closest grid point is on land, or if more than 2 points are located on land.

How does CMS prevent a particle to come on land?

If the new position of a particle is on land and the flag **ckavoidcoast** is set to **.true.**, then CMS will find a new position located on water for this particle, following the steps outlined bellow. If the particle is still on land, CMS will move on to the next step. If a particle is on water after a step, this will be the new position of the particle.

- CMS places the particle back to its old location and moves the particle with U and W-velocity.
- CMS places the particle back to its old location and moves the particle with V and W-velocity.
- CMS places the particle back to its old location and moves the particle with U and V-velocity.
- CMS places the particle back to its old location and does not move the particle.

How does CMS make it possible for a particle to reach land?

If a particle reaches the land-mask at the coastline or anywhere on the topography, it will stop moving and exit with an exitcode = -2 in the trajectory output file; the model will continue to run with the next particle. If this is the desired behavior, the flag **avoidcoast** in the file runconf.list should be set to **.false.**

If the particles should not come on land you can set the flag **avoidcoast** in the file *runconf.list* to **.true.**

Backward Tracking Module

```
ckbackward
```

Motivation: This module is useful to estimate spawning locations of larvae caught in plankton studies.

If **ckbackward** is set to **.true.** in the file *runconf.list* then particle displacement will be calculated b integrating the velocity field backward in time, answering the question “Where did a particle come from?”

If **ckbackward** is **.false.** then the trajectories are moving forward in time so it answers the question “Where did a particle go?”. This is CMS default mode.

To calculate the backward trajectories CMS reads the nest files in opposite order and changes the sign of the velocity components. When using the backward module, you will not be able to use **buoyancy**,

massSpawning, and **tidalMovement**. If you wish to use the ontogenic vertical matrix migration with backward trajectory, you need to reverse the vertical matrix.

Output files in ASCII

```
ascii
```

CMS standard format output is netCDF. If ascii is set to true in the file runconf.list then output files will be in ascii format (see Chapter 7 for more details).

Prevent Particle from Dying at the Uppermost Level

```
upperlevelsurface
```

In the default CMS mode, the vertical movement of particles is confined to the uppermost layer (typically the sea surface), preventing particles from “flying into the air.”

If, however, particles should be killed when reaching surface, the flag upperlevelsurface has to be set to .false. in runconf.list.

Loop Through the Nest Files

```
loopfiles
```

Motivation: In certain applications it is necessary to advect particles for periods of time longer than oceanographic files are available. On its default mode, CMS remove particules from the simulation when there are no more nest files (exitcode -5).

By setting the **loopfiles .true.** in runconf.list, particle advection will continue until the maximum integration time. CMS will loop through the available files, using the first oceanographic file once it gets past the last file.

Create restart files

```
restartfromfile
```

CMS will create a binary raw file with all the information necessary to restart a run for passive particles, that is, not using the *ibm.list* options.

It will save this file at a frequency specified by the variable *restartwritefreq*. The restart files are written in the SCRATCH directory. To restart from a saved restartfile, set the restartfromfile variable to **.true.**. Note that the traj_file is now overwritten with new data from the restart time.

Mixed layer parameterization

```
mixedlayerphysics
```

If information on the depth of the mixed layer base is available, CMS can randomly move particles that are above that depth vertically through the mixed layer at each time step. Set the parameter to **.true.**.

The field **mixedlayerwmax** defines the maximum vertical velocity that a particle can get in the mixed layer.

6.2 Individual Based Model ibm.list

The *ibm.list* is an input file that allows to give particles specific behaviors. If using CMS on passive-particle mode (physical applications), you can delete this file from your input directory.

```
&ibm
!===============================================================!
!Buoyancy Module
Buoyancy         = .false.
buoyancyFilename = "buoyancytest" !buoyancy filename
!===============================================================!
!Seascape Module
polygon          = .false.
polyFilename     = "Caribbean.xyz"
settlementStart  = 160            !in days
!===============================================================!
!Probability Matrix of Vertical Migration
ibio             = .true.
ibioFilename     = "vert_matrix"
ibioTimeStep     = 259200        !in seconds
!===============================================================!
!Mortality Rate
mort             = .false.
halflife         = 90000         !in seconds
!===============================================================!
!Different Particle Attributes Module
diffpart         = .false.
diffpartFilename = "diffpart_matrix"
!===============================================================!
!Combined Buoyancy and Vertical Migration
massSpawning     = .false.
larvaStart       = 7             !in days
!===============================================================!
!Selective Tidal Stream Transport
tidalmovement    = .false.
tstStart         = 14            !in days
!===============================================================!
!Adding strata for 3D polygons
strata           = .false.
strataFilename   = "strataFile"  !file with strata information
!===============================================================!
```

```
!Output interpolated temperature in netcdf file
outputtemp       = .false.
outputsaln       = .false.
!===============================================================!
$end
```

Buoyancy

```
buoyancy
```

Motivation: Transport of eggs and early larval stages is considered to be one of the major factors impacting recruitment success. The eggs of most marine fish are positively buoyant at time of release. How millions of eggs are initially transported and where they end up before hatching depends on their buoyancy, flow stratification and turbulence. The terminal velocity of particles (eggs) is computed as the force balance between the particle specific density, water density and the kinematic viscosity.

So, besides using vertical velocities to move particles up and down, CMS can also add the particle buoyancy to the particle velocity.

To move particles taking their buoyancy into account, set the flag **buoyancy** in the file *ibm.list* to **.true.**. You will need to specify the name of file which stores the particle diameter and density information.

Eggs' diameter and density changes with time, so you can define these characteristics at time discreet intervals. Also, variability in size and density of eggs can be observed for the same species (or even the same breeder), thus it is possible to set a range of densities and diameters to be used in your experiments.

The buoyancy depends on the particle's attributes, and also on the environmental conditions, so your nests files must store the oceanographic temperature and density fields.

The buoyancy input file should have the following structure:

```
4
432000 864000 1728000 2160000
1010 1015 1020 1027
1012 1017 1022 1029
0.00025 0.00020 0.00015 0.00010
0.00027 0.00022 0.00017 0.00012
```

- Line 1 = number of columns, which represents the time steps at each the eggs' properties will be changed
- Line 2 = time (in seconds) that each property column is valid
- Lines 3 and 4 = lower and upper boundaries of the particle density (kg/m^3)
- Lines 5 and 6 = lower and upper boundaries of the particle diameter range (meters)

To simulate the transport not considering a range of values, simply add the same values for the lower and upper boundary. To use the same values for all time steps, create a file with one column, with a time interval equal to your simulation maximum time, e.g..

```
1
2160000
1010
1010
0.00010
0.00010
```

The velocity determined by buoyancy is calculated by the Stoke's formula, which is then added to the W-velocity (from the velocity field):

```
(9.81*diam_particle2 *(dens_particle - dens_water))/(18*viscosity)
```

Where:

- diam_particle : the particle diameter (meters)
- viscosity : the seawater viscosity computed with the following formula:

  ```
  1.88e-3 - (T*4.e-5)
  ```

  where T is the temperature of the water.

- density_particle: density of the particle (kg/m^3)
- dens_water: the density of the seawater (kg/m^3).
- Water temperature and density must be stored in the oceanographic files.

Seascape Module

```
polygon
settlementStart
```

Motivation: Polygons are the representation of marine habitat, describing the location of both the spawning and the nursery areas. Inside their boundaries two essential processes will be simulated: release and retention. Therefore, the polygons must be carefully defined accordingly to the region, species and processes you are interested to study.

By setting the flag **polygon** to **.true.** in the file *ibm.list* you can add marine habitat. Polygons are described in a separate polygon file, which name should be added to the *ibm.list* file.

A single polygon in the polygon file looks as follows:

```
Longitude    Latitude    Polygonnumber
277.2766     24.6665     1
277.2715     24.6235     1
277.2507     24.5921     1
```

```
277.2243    24.5786     1
277.1763    24.5777     1
277.1600    24.5564     1
277.1417    24.5851     1
277.1531    24.6684     1
277.2766    24.6665     1
```

- Polygon numbers must start from one, and be consecutive and in ascending order.
- Particle settlement in the habitat represented by polygons will start once particles reach the age defined on settlementStart in the input file ibm.list.
- If using strata (3D polygons) you will need to add a 4th column with the depth of each polygon vertex (as it will be described on the strata section)

How to Create the polygon file?

Polygon files can be generated in a GIS-based software. You need to begin with projected habitat location data. Then create a vector grid to overlay the habitat data. Set the grid size you want for the size of the polygons. Perform a spatial join with the grid and reef polygons. The next step is to remove all polygons which fall completely within the land mask (Fig. 13).

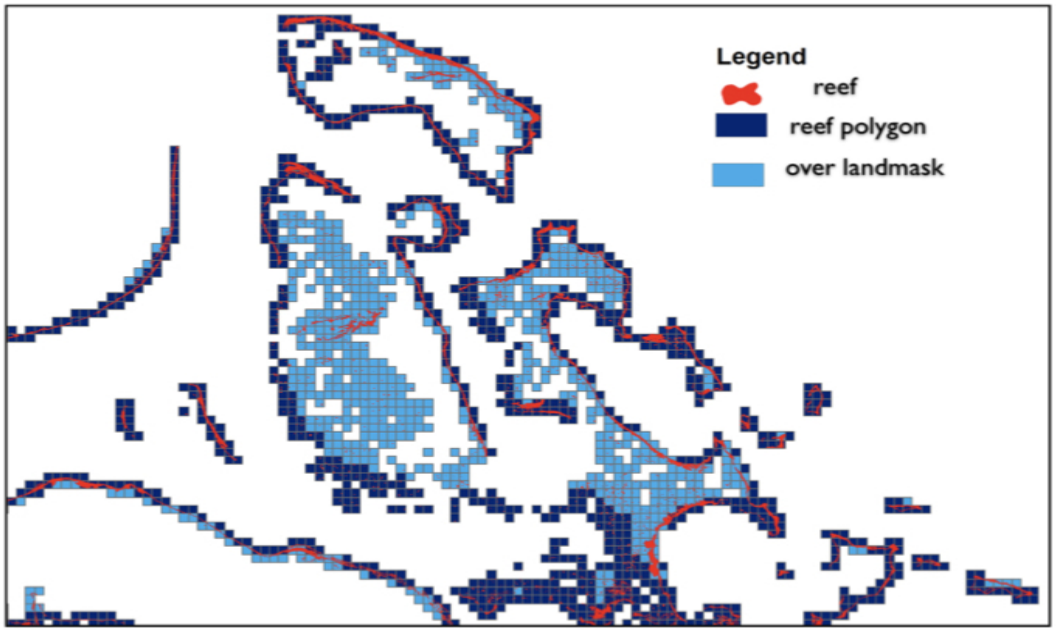


Figure 13. An example of 8km x 8km grid polygon in the Bahamian Archipelago.

Another particle way to create polygons is to create a buffer around the habitat and split the resulting polygons in equal segments using a tension factor representing the scale of the unit polygons (Fig.14).

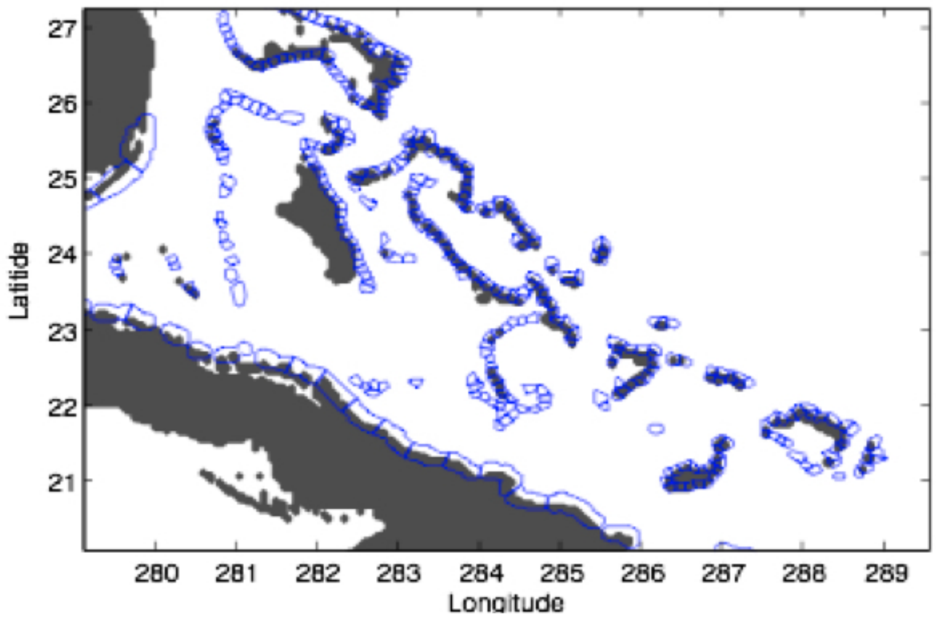


Figure 14. Example of a non-grid polygon file composed of irregular ca. 5km x 10km units for the Bahamian Archipelago and ca. 9km x 50km units for North Central Cuba.

You can assign each polygon a unique number which will match the release locations.

Finally, to obtain the latitudes and longitudes of your polygon vertices for input into the CMS, convert polygon to points. This gives you a point feature layer with the vertices for each polygon preserved with their original polygon number. The attribute table can be exported directly into a text file.

How to Create Release Points?

Find the new centroids of the polygon file to ensure the release point is within the polygon itself. This should ensure all your release points are in the 'ocean'.

Vertical Migration

```
ibio
ibioFilename
ibioTimestep
```

Motivation: Most marine larvae undergo ontogenetic vertical migration whereby they swim or adjust they buoyancy to move downward from the upper part of the water column to deeper layers or reversely, upward into the neuston as they age. As a result, vertical distribution patterns can be observed via plankton surveys and can be statistically described as stage-specific probability density distributions with depth. This imprinted behavior is species-specific and the center of mass and vertical spread of a particular organism can be constrained by varying environmental conditions; yet on average the observed distributions are typically robust.

To add vertical migration, set the flag **ibio** on *ibm.list* with **.true.**, and CMS will vertically move particles by using a probability matrix, which should represent the observed distribution of larvae through ontogenetic or diel vertical migration.

The probability matrix is described in a separate file, which name should be added to the *ibm.list* file.

The probability matrix should look as follows:

```
11
7
10 20 30 40 50 60 70
86400 86400 86400 432000 432000 432000 259200 259200 259200 259200      259200
20 10 10 00 00 00 00 00 00 00 00
20 10 10 05 05 05 05 00 00 00 00
20 10 10 05 05 05 05 00 00 00 00
40 70 60 20 20 20 20 10 10 10 10
00 00 10 40 40 40 40 50 50 50 50
00 00 00 20 20 20 20 30 30 30 30
00 00 00 10 10 10 10 10 10 10 10
```

- Line 1 = number of columns (x), representing how many times the vertical distribution of larvae will change along a simulation
- Line 2 = number of rows (y), representing the number of depth levels used
- Line 3 = maximum depth of each row in meters
- Line 4 = number of seconds each column of vertical distributions is valid
- Lines 5 to 5+y-1 = array with probabilities (x by y). The sum of probabilities for each column should be equal to 100

Each row (y) in the array represents a depth and each column (x) corresponds to a time interval. The value at each point (x,y) refers to the percentage of larvae that can be found at that depth at that time interval. The model randomly chooses based on the matrix what the depth of a particle is.

Particles will change vertical layers at the period defined by **ibioTimestep**. The **ibioTimestep** can be equal or an integer multiple of **timestep**. If **ibioTimestep** is not defined, then CMS will use **timestep** instead.

If you are using **ibio**, you are not able to use **buoyancy** (which will be ignored).

Mortality Rate

```
mort
```

Motivation: Larval mortality is an important parameter that can change the dispersal distance of an organism, influencing the structure of the metapopulation. Larval mortality rates are not constant throughout larval life and larvae acquire and lose competence at varying time/age. As a result, the pelagic larval duration (PLD) is often plastic. Similarly, abotic particles may have important fate processes that need to be taken into account during dispersion. Here we introduce a simple half-life mortality function that can be further modified to match experimental/ observational survivorship curves.

If the flag mort in the file *ibio.list* is set to true, then a particle can die. The mortality rate of particles is based on an exponential decay formula, following their half-life time.

Half-life is the period of time it requires for a quantity to fall to half its value. For larvae, it is the time it takes for a larval cohort to be reduced by half; for a chemical compound undergoing decay, it corresponds to a the time it takes to decrease by half. The decay constant is given by:

```
lambda = ln(2) / half-life
```

The half-life time of the particle can be entered in the file *ibm.list*, in seconds.

This equation also can be changed in the file **mod_ mort_larva.f90**.

Different Particle Attributes Module

```
diffpart
```

Motivation: this module allows defining variable attributes such as size, density, and mortality rates observed in the early life history traits. Similarly, this can be used for the dispersion of abiotic particles with varying characteristics and fate.

If **diffpart** in the file *ibm.list* is set to **.true.** then the particles can have different sizes, densities and half-life times. This is only useful if you like to add buoyancy or/and use mortality rates.

If the **diffpart** is **.true.** then the values at the parameters **diam_particle**, **density_particle** and **half_life** in the file *ibm.list* will be ignored.

The different attributes are described in a separate diffpart file, the name of this file has to be added to the *ibm.list* file. The diffpart file looks as follows:

```
3
33  33  34
820 840 860
840 860 880
2e-6    2e-4    1e-3
2e-4    1e-3    1e-2
86400   432000  900000
```

- Line 1: number of different categories (x)
- Line 2: the x probabilities that a particle sits in a particular category.
- Line 3: the x different minimum densities.
- Line 4: the x different maximum densities. The density of a particle will be a random number between the minimum density and maximum density.
- Line 5: the x different minimum sizes.
- Line 6: the x different maximum sizes. The size of a particle will be a random number between the minimum size and maximum size.
- Line7: the x half life times in seconds .

Spawning aggregations

```
massSpawning
```

Motivation: Many organisms gather in large aggregations for synchronized mass spawning. These gatherings produce enormous numbers of gametes concentrated in space and time, that disperse initially as a cloud of buoyant particles, become less buoyant after fertilization, and increase in specific gravity prior to hatching into swimming larvae.

This module resolves the transition from the egg to the larval stages. When using the flag **massSpawning** in the file *ibm.list*, particles will move in the vertical using sequentially buoyancy and vertical migration.

Initially, particles (eggs) will move in the vertical following the parameters defined by **buoyancy**.

When eggs reach the age of hatch, which is defined by **larvaStart** (in days) in the file *ibm.list*, larvae will move vertically following the file with the probabilistic vertical migration matrix. So when using mass spawning, it is necessary to provide a **ibioFilename** and corresponding file.

Selective Tidal Stream Transport

```
tidalMovement
```

Motivation: Many estuarine fish and invertebrate species undergo vertical migrations that are coordinated with phases of the tide in order to achieve horizontal movement. This general mechanism is known as Selective Tidal Stream Transport (STST), and more than one behavior has been associated with it. A flood-tide transport occurs when organisms use the flood phase for shoreward transport and immigration to estuaries. An ebb-tide transport or ebb-phased migration is used for seaward transport and out-migration from estuaries.

If **tidalMovement** (*ibm.list*) is set to **.true.**, particles will only move if the sea surface height is rising. When particles do not move, their depth in the trajectory output file will be set as -999.99.

The days after which the STS movement starts depends on the value of the **tstStart** (in days) defined on *ibm.list*.

To use **tidalMovement**, the variable sea surface height will need to be present on the nest files. The rise or decrease of the SSH is given by the gradient between the SSH from the two nest files closes to the current time (before and after). This rule can be changed in the source code.

3-Dimensional polygons

```
strata
```

Motivation: Marine organisms can be distributed over large ranges of depths, however, their young might need specific conditions met at a smaller range of depths to survive. Also, understand how habitats located a different depths are connected is becoming a central challenge for the demographic quantification of populations and the design of conservation measures.

To account for discreet 3D habitats a new module was added to CMS which allow users to use 3D polygons and control depths at which settlement will take place.

When using the flag **strata** in the file *ibm.list*, polygons will be defined at different vertical strata in the water column, and larvae will settle at the nearest polygon after their competency period

- The strata boundaries are flexible and defined by the user using the strata file
- The depths of polygons must be present in the polygon file to use the strata option
- The spawning and settlement strata will be recorded on the connectivity file when using this option

The strata file have the following structure

```
6
0 21 41 61 81 101
```

- Line 1: number of vertical strata to be used + bottom boundary of vertical strata
- Line 2: initial depth of each vertical strata (meters).

The maximum depth of each strata is assumed to be the next strata initial limit - 1 meter. The last strata depth represents the bottom boundary of vertical strata: larvae found deeper than this value will therefore not settle.

In example file above, there are 5 vertical strata, ranging from:

1) 0-20 m
2) 21-40 m
3) 41-60 m
4) 61-80 m
5) 81-100 m

Larvae found at depths bigger than 101 meters will not settle at any polygon.
A larvae located at 15 m will settle only in a polygon located between 0-20 m, as a larvae located at 41 m will settle between 41-60 m.
These calculations will be done by the model, which will define at which vertical strata larvae and polygons are located based on the user defined strata file.

The polygon file will therefore have the following configuration, where the forth column is the depth of the polygon in meters:

```
Longitude   Latitude    Polygonnumber   Depth
277.2766    24.6665     1               20
277.2715    24.6235     1               20
277.2507    24.5921     1               20
277.2243    24.5786     1               20
277.1763    24.5777     1               20
277.1600    24.5564     1               20
277.1417    24.5851     1               20
277.1531    24.6684     1               20
277.2766    24.6665     1               20
```

Polygons with the same configuration (same vertices), located at different depths, must be given unique polygon numbers. The connectivity file in this option will record the spawning strata and the settlement strata.

7. Output files

7.1 Trajectory output file traj_locname

The trajectory output file can have two format options. The standard output is on NetCDF format. You can choose to set the format to ascii.

Option 1: NetCDF output

Dimensions:

```
Particle : number of particles in the NetCDF file
Time     : number of time steps saved in the NetCDF file
```

Variables:

```
int time(time)
    units     = "seconds"
    long_name = "Time after release"
    FillValue = -1

int location(particle)
    long_name = "Line number in the release file"
    FillValue = -1

float lon(particle,time)
    units     = "degrees_east"
    long_name = "Longitude"
    FillValue = 1.267651e+30f

float lat(particle,time)
    units     = "degrees_north"
    long_name = "Latitude"
    FillValue = 1.267651e+30f

float depth(particle,time)
    units     = "meter"
    long_name = "Depth"
    FillValue = 1.267651e+30f
```

The following variables will be only present if a ibm.list file is present in the input directory:

```
float distance(particle,time)
    units     = "kilometer"
    long_name = "Cumulative distance"
    FillValue = 1.267651e+30f

int exitcode(particle)
    long_name = "Status with which the particle exits"

double releasedate(particle)
    units     = "Julian date"
    long_name = "Date the particle is released"
```

Only in output file if the flag polygon is turned on:

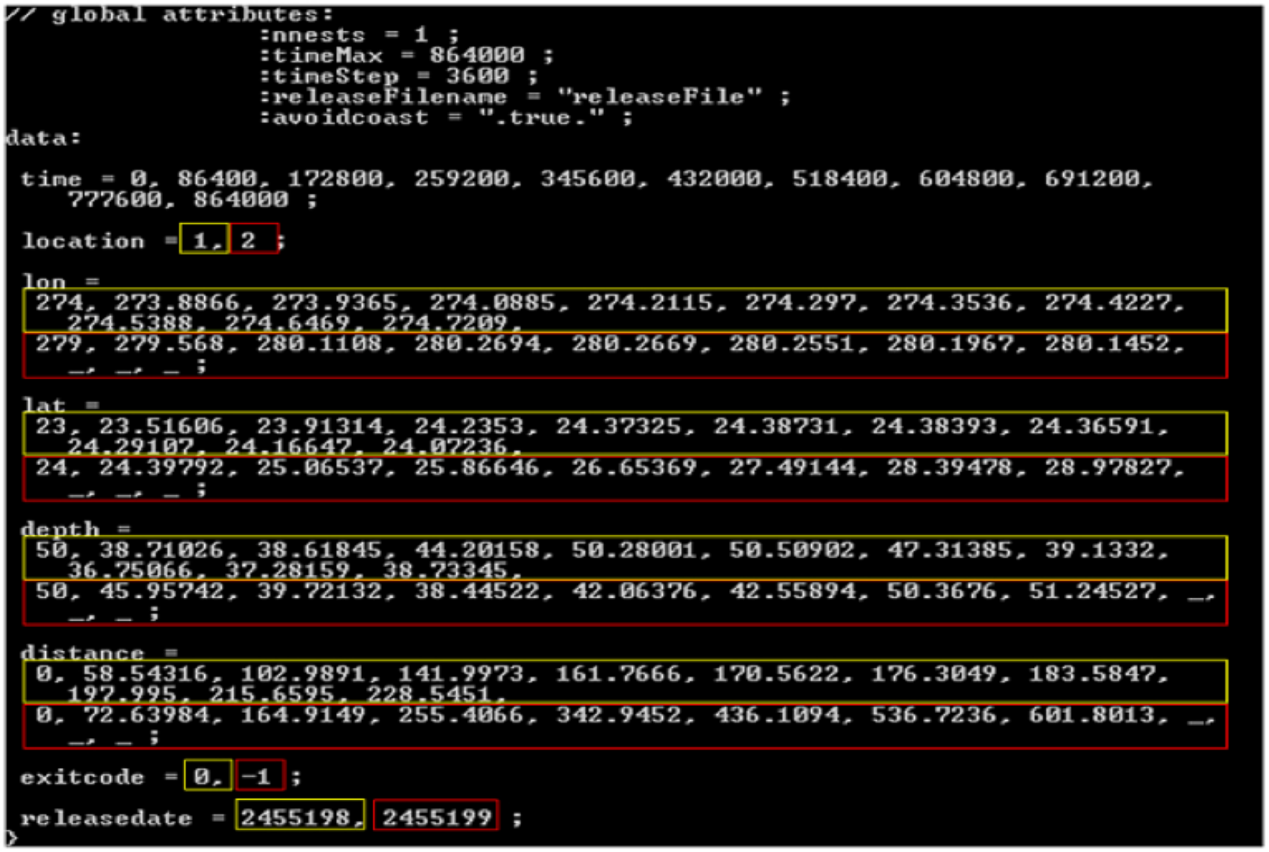
```
int releasepolygon(particle)
    long_name = "Number of release polygon"
```

Only in output file if the flag buoyancy is turned on:

```
float density(particle)
    units     = "kg/m3"
    long_name = "Density of particle"

float diameter(particle)
    units     = "meter"
    long_name = "Diameter of particle"
```

Example:
Two particles are released at different locations and days, and are dispersed for 10 days, as seen on Figure 15. The output in the yellow box is for particle 1, while on the red box is for particle 2. Particle 2 leaves the model area on day 7 (as denoted by the exit code -1) and does not move after this day, while particle 1 moves until the end of the simulation.



```
// global attributes:
                :nnests = 1 ;
                :timeMax = 864000 ;
                :timeStep = 3600 ;
                :releaseFilename = "releaseFile" ;
                :avoidcoast = ".true." ;
data:

 time = 0, 86400, 172800, 259200, 345600, 432000, 518400, 604800, 691200,
    777600, 864000 ;

 location = 1, 2 ;

 lon =
  274, 273.8866, 273.9365, 274.0885, 274.2115, 274.297, 274.3536, 274.4227,
    274.5388, 274.6469, 274.7209,
  279, 279.568, 280.1108, 280.2694, 280.2669, 280.2551, 280.1967, 280.1452,
    _, _, _ ;

 lat =
  23, 23.51606, 23.91314, 24.2353, 24.37325, 24.38731, 24.38393, 24.36591,
    24.29107, 24.16647, 24.07236,
  24, 24.39792, 25.06537, 25.86646, 26.65369, 27.49144, 28.39478, 28.97827,
    _, _, _ ;

 depth =
  50, 38.71026, 38.61845, 44.20158, 50.28001, 50.50902, 47.31385, 39.1332,
    36.75066, 37.28159, 38.73345,
  50, 45.95742, 39.72132, 38.44522, 42.06376, 42.55894, 50.3676, 51.24527, _,
    _, _ ;

 distance =
  0, 58.54316, 102.9891, 141.9973, 161.7666, 170.5622, 176.3049, 183.5847,
    197.995, 215.6595, 228.5451,
  0, 72.63984, 164.9149, 255.4066, 342.9452, 436.1094, 536.7236, 601.8013, _,
    _, _ ;

 exitcode = 0, -1 ;

 releasedate = 2455198, 2455199 ;
}
```

Figure 15. Example of an output file in NetCDF format.Option 2: Ascii output

Location	Particle	Time	Longitude	Latitude	Depth
1	1	0	282.380	24.480	0.000
1	2	0	277.034	24.615	1.000
2	1	0	278	25	100.0
1	1	43200	282.310	24.654	2.000
1	1	86400	282.321	24.833	4.000

Distance	Exit code	Release date	Release polygon	Density	Diameter
0	0	2453431	5	820	0.0005
0	0	2453431	5	820	0.0005
0	0	2453432	1	840.5	0.00025

20.64	0	2453431	5	820	0.0005
40.44	-1	2453431	5	820	0.0005

Example of a line of the output file:

```
Location id
```

Line number in the release file.

```
Particle id
```

The number of the particle at a release location.

```
Time
```

The time in seconds from release.

```
Longitude, Latitude, Depth
```

Location of the particle in degrees, and depth in meters.

```
Distance
```

The total distance the particle travelled from its starting point (cumulative distance), in kilometers. If not using the *ibm.list* file, this column is not present.

```
ExitCode
```

Status with which the particle exits

```
Releasedate
```

Date the particle is released in Julian date. If not using the *ibm.list* file, this column is not present.

```
Release polygon
```

The number of the release polygon, only added if the flag polygon is turned on.

```
Density
```

The density of the particle in kg/m^{3,} only added if the flag buoyancy is turned on.

```
Diameter
```

The diameter of the particle in meters, only added if the flag buoyancy is turned on.

Distance calculation

The distance is calculated only using the longitudes and latitudes, so not the depth.

The formula used is:

```
dist = sin((lat2-lat1)/2)^2 + sin((lon2-lon1)/2)^2 * cos(lat1) * cos(lat2)

distance= R * 2 * atan2(sqrt(dist), sqrt(1-dist))
```

where R is the is earth's radius (mean radius = 6371.22 km)

This formula calculates the distance between position 1 (lon1, lat1) and position 2 (lon2, lat2).

Possible options of exit codes:

- **0** Particle was moving.
- **-1** Particle left the model area.
- **-2** Particle was too close to land.
- **-3** Particle died. This exit code appears only if using mortality.
- **-4** Particle settled on a polygon. This exit code appears only if using polygons.
- **-5** There were no oceanographic data files for this date.

Connectivity output Con_locname

The other output file is a file that contains the connectivity of the particles. This output file will only be created if the flag **polygon** in the file *ibm.list* is set to **.true.**.

Settlement polygon	Settlement polygon	Settlement year	Settlement month	Settlement day
1	4	2008	8	2

Age in seconds	Settlement Depth	Spawning year	Spawning month	Spawning day
95040	1.00	2008	7	20

Each line of the outputfile gives information about where and when a particle settled.

```
Release polygon
```

The number of the polygon where the particle is released

```
Retention polygon
```

The number of the polygon where the particle is settled

```
Retention Year, month, day
```

The year, month and day that the particle is settled

```
Longitude, Latitude, Depth
```

Location of the particle in degrees and meters for depth.

```
Age
```

The age of the particle in seconds at the moment the particle settled.

```
Depth
```

The depth in meters at the moment the particle settled.

```
Release Year, month, day
```

The year, month and day that the particle is released

When using the flag **.strata.** two extra columns will be recorded in this file: spawning and settlement strata:

```
Spawning strata
```

The strata at the moment the particle was spawn

```
Settlement strata
```

The strata at the moment the particle settled

Post processing

The CMS package also provides Matlab® code for output visualization.

- draw_velocities.m: draws either the U-velocity, the V-velocity or the speed.
- draw*traj*ascii.m: draws the trajectories and land when the outputfile is in Ascii format
- draw*traj*netcdf.m: draws the trajectories and land when the outputfile is in netCDF format
- make_mtx.m: computes and draws a square connectivity matrix

All four codes have parameters at the start of the file, these parameters have to be filled in before you run the code.

8. Code

Following is a complete listing of source code files with a short description of their content. The subroutines that define the most important variables are the following:

General

def_nests.f90 Defines types to store the information of the nests, such as boundaries, dates, and others

def_particle.f90 Defines types to store particles information, such as release position and date.

def_globalvariables.f90 Defines global variables

def_constants.f90 Defines constants

Modules

mod_kinds.f90 Specifies options for kind types

mod_netcdf.f90 Defines error messages related to NetCDF

mod_iounits.f90 Routines for automatic allocation of i/o units

mod_random.f90 Gaussian random number generator

mod_netcdfoutput.f90 Routines that create and write output to the NetCDF trajectory file

mod_directory.c Creates directories and copies files from one directory to another

mod_calendar.f90 Compute calendar date from Julian date and Julian date from calendar date

mod_nciorw.f90 Routines that create, read and write output to NetCDF files

mod_getdata.f90 Routines that read oceanographic files from an OPeNDAP server or from locally stored files

Modules used accordingly to flags:

mod_ibio.f90 Routines for ontogenetic vertical migration (flag ibio is set to true)

mod_turb.f90 Routines for turbulence movement

mod*mort*larva.f90 Routines for mortality

mod*buoyancy*larva.f90 Routines for buoyancy

mod_reef.f90 Routines for using polygon

mod_diffpart.f90 Routines if using diffpart