Croco Tutorials

Release 1.2

S.Jullien, M.Caillaud, R. Benshila, L. Bordois ,

G. Cambon, F. Dufois, F. Dumas, J. Gula ,

M. Le Corre, S. Le Gentil, F. Lemarié ,

P. Marchesiello, G. Morvan, and S. Theetten

Jan 17, 2022

TUTORIALS

1 Disk space **1**

2 Compilers and Libraries **3**

3 Environment variables **5**

4 Download **7**
4.1 Downloading CROCO 7
4.2 Getting other codes (coupling) 8

5 Contents & Architecture **11**
5.1 Architecture 11
5.2 Contents 11

6 Summary of essential steps **17**

7 Test Cases **19**
7.1 BASIN 19
7.2 Set up you own test case 21

8 Regional: Preparing your configuration **25**

9 Regional: Preprocessing (Matlab) **27**
9.1 Contents of the `croco_tools` 29
9.2 Philosophy of the `croco_tools` 30
9.3 Climatological pre-processing 30
9.4 Interannual pre-processing 35

10 Compiling **39**
10.1 cppdefs.h 40
10.2 param.h 43
10.3 jobcomp 43
10.4 Compilation options 45
10.5 Tips in case of errors during compilation 45

11 Running the model **47**
11.1 Edit croco.in 47
11.2 Run the model 50
11.3 Tips in case of BLOW UP or ERROR 51

12 Increasing the resolution: BENGUELA_VHR **53**

13 Running with interannual forcing **55**
13.1 Run after classical interannual pre-processing 55
13.2 Alternative method: online interpolation of atmospheric bulk forcing 57

14 Nesting Tutorial **59**

15 Adding Rivers **63**
15.1 Constant flow and concentration . . . 63
15.2 Variable flow read in a netCDF file and **constant concentration** . . . 64
15.3 Variable flow and **variable concentration** from a netCDF file . . . 66
15.4 Using a nest . . . 66

16 Adding tides **69**
16.1 Pre-processing (Matlab) . . . 69
16.2 Compiling . . . 70
16.3 Running . . . 71

17 Visualization (Matlab) **73**

18 Visualization (Python) **75**
18.1 Setup your Miniconda environment . . . 75
18.2 Croco_visu directory . . . 75
18.3 Launch visualization . . . 75
18.4 How to customize for your own history files . . . 84
18.5 How to add new variables . . . 85

19 NBQ Tutorial **89**
19.1 Some important points about Large-Eddy Simulations (LES) . . . 89
19.2 KH_INST Test Case . . . 92
19.3 Set up your own NBQ configuration . . . 93
19.4 NBQ OPTIONS . . . 94
19.5 Appendix : some words on CROCO-NBQ kernel . . . 95

20 Coupling tutorial **97**
20.1 Summary of steps for coupling . . . 97
20.2 Compiling in coupled mode . . . 98
20.3 Simple CROCO-TOY coupled example . . . 105
20.4 Advanced coupling tutorial . . . 108

21 Littoral dynamics tutorial **133**

22 Realistic coastal configuration **137**

23 XIOS **139**

24 Tips **143**

25 CROCO/MUSTANG tutorial & tips **145**
25.1 Get to know the CROCO/MUSTANG coupling . . . 145
25.2 Run a test case . . . 145
25.3 Create your own configuration . . . 146

26 TRAINING 2019: DATARMOR specific **151**
26.1 Getting the good environment . . . 151
26.2 Creating your work architecture . . . 151
26.3 DATA FILES . . . 152
26.4 BASIN configuration for XIOS tutorial . . . 152
26.5 SOURCES for coupling tutorial . . . 152

27 Ifremer specific **155**
27.1 *Croco training in the framework of datarmor* . . . 155

Bibliography **175**

CHAPTER

ONE

DISK SPACE

CROCO and CROCO_TOOLS source codes require less than 500 MB of disk space. Climatological datasets, provided for regional configuration, require about 18 GB of disk space.

CHAPTER

TWO

COMPILERS AND LIBRARIES

CROCO uses Fortran routines as well as cpp-keys. The I/O are in netcdf. It therefore requires to have:

- a C compiler
- a Fortran compiler
- a Netcdf library
- MPI libraries and compilers if running in parallel

CROCO_TOOLS use Matlab, and Python scripts.

CHAPTER

THREE

ENVIRONMENT VARIABLES

A few environment variables for compilers and libraries should be declared to avoid issues when compiling and running CROCO. If you are using Intel compilers for instance, you should declare the followings (in your .bashrc file):

```
export CC=icc
export FC=ifort
export F90=ifort
export F77=ifort
```

For Netcdf, you should also declare your netcdf path, and add it to the `PATH` and `LD_LIBRARY_PATH` environment variables. Here is an example:

```
export NETCDF=$HOME/softs/netcdf
export PATH=$NETCDF/bin::${PATH}
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}::${NETCDF}/lib
```

Note: Common errors associated with Netcdf are usually solved by checking that Netcdf is correctly declared in your `LD_LIBRARY_PATH`

CHAPTER

FOUR

DOWNLOAD

4.1 Downloading CROCO

To perform a regional simulation using CROCO, the modeler needs:

- the croco source code
- the croco_tools scripts, which are tools for pre- and post-processing
- some utilities for the croco_tools (additional libraries)
- datasets to force the model at the boundaries, at the surface, and at the inital time

4.1.1 Source code

CROCO source code and croco_tools stable releases are available in the Download section of https://www.croco-ocean.org/croco-project/

Both are available for download in a compressed tar files. To install them:

```
tar -zxvf croco-v1.0.tar.gz
tar -zxvf croco_tools-v1.0.tar.gz
```

Also `tar -zxvf` all the climatological datasets

Check if fixes for reported bugs have been released on: https://www.croco-ocean.org/croco-project/. If there are some, download them and copy the routines in the appropriate directory.

Example at the date of 10th of July 2019:

```
cp analytical.F_fix17juin2019 croco-v1.0/OCEAN/analytical.F
cp get_psource_ts.F_fix04september2018 croco-v1.0/OCEAN/get_psource_ts.F
cp read_inp.F_fix30july2018 croco-v1.0/OCEAN/read_inp.F

cp check_domain_runoff.m_fix17juin2019 croco_tools-v1.0/Rivers/check_domain_runoff.
↪m
cp make_runoff.m_fix17juin2019 croco_tools-v1.0/Rivers/make_runoff.m
```

4.1.2 Matlab utilities

For the Matlab toolbox, additional packages as m_map, air-sea, etc, are required (named UTILITIES).

Note: UTILITIES are now provided directly within croco_tools, and do not need to be downloaded separately

Otherwise, (if the provided files are not working on your platform) they are avaiable for download here: https://www.croco-ocean.org/download/utilities/, or the user can download them on the original repositories:

- the NetCDF Matlab Mex file is needed to read and write into NetCDF files and it can be found at the web location: http://mexcdf.sourceforge.net/.
- The LoadDAP Matlab Mex file is used to download data from OpenDAP servers for inter-annual and forecast simulations. It can be found at the web location: http://www.opendap.org/download/ml-structs.html. The Matlab LoadDAP Mex file provides a way to read any OpenDAP-accessible data into Matlab. Note that the LibDAP library must be installed on your system before installing LoadDAP. Details can be found at the web location: http://www.opendap.org.

4.1.3 Forcing datasets

Finally, forcing **datasets** are required (initial, surface, and boundary conditions). Climatological global datasets are provided here: https://www.croco-ocean.org/download/datasets/

croco_tools also provide pre-processing scripts for the use of interannual datasets as:

- CFSR, ERA-interim, ... for atmospheric forcing
- SODA, ECCO2, MERCATOR, ... for the ocean boundaries and initialization

4.2 Getting other codes (coupling)

- **OASIS coupler**

To use CROCO in coupled mode (coupling with atmosphere and/or waves), **OASIS3-MCT version 3** is required.

Note: Older versions of OASIS do not include all the necessary functions as grid generation in parallel mode. If you want to use an older version, you need to create your grids.nc, masks.nc, and areas.nc files first, and comment the call to `cpl_prism_grids` in `cpl_prism_define.F`

To download OASIS3-MCT, you need to register on OASIS website: https://portal.enes.org/oasis/

Then, you can download the code from the website or use the git repository:

```
git clone https://gitlab.com/cerfacs/oasis3-mct.git
```

- **WW3**

WaveWatch3 is now hosted on github on a public repository: https://github.com/NOAA-EMC/WW3

You can thus clone the repository:

```
git clone https://github.com/NOAA-EMC/WW3
```

- **WRF**

Currently the distributed version of WRF does not include coupling with waves, and some other functionalities we have recently implemented. We therefore suggest to use the fork including modifications for coupling with WW3 and CROCO through the OASIS coupler, but note that this is a development version... https://github.com/wrf-croco/WRF/tree/WRF-CROCO

You can clone it with git :

```
git clone https://github.com/wrf-croco/WRF.git
```

Other versions of WRF are available here: http://www2.mmm.ucar.edu/wrf/users/download/get_source.html https://github.com/wrf-model/WRF

- **WPS**

WRF pre-processing system is also needed to prepare WRF configurations. It is available on the following github repository: https://github.com/wrf-model/WPS

You can clone it with git:

```
git clone https://github.com/wrf-model/WPS.git
```

You need to use the same WPS version than the WRF version you use. Currenlty the WRF version on the WRF-CROCO fork is WRF4.2.1. You should therefore use the WPS 4.2 version. To do so, with git you can move to the appropriate tag:

```
cd WPS
git checkout tags/v4.2
```

CHAPTER

FIVE

CONTENTS & ARCHITECTURE

5.1 Architecture

A classical work architecture consists in:

```
croco/croco
croco/croco_tools
CONFIGS
```

To run a CROCO simulation, you need to follow these 3 steps:

- complete the pre-processing (for realistic cases) => see Pre-processing tutorial
- set-up the parameters and setting files (`param.h` and `cppdefs.h`) and compiled the model
- set-up the input file `croco.in` and run the model

CROCO contents, main inputs and setting files are described in the following:

5.2 Contents

CROCO and its tools are distributed in separate directories `croco` and `croco_tools`.

5.2.1 croco

AGRIF	Agrif library for nesting
CVTK	Regression test library
DOC_SHINX	Model documentation
MPI_NOLAND_preprocessing	Fortran utility to determine the optimal MPI decomposition to supress land computation
MUSTANG	MUSTANG sediment model
OCEAN	CROCO source files
PISCES	PISCES biogeochemical model source files
`README.md`	Informations on CROCO version
SCRIPTS	Scripts for plurimonth runs, online analysis tools, and coupled simulations
TEST_CASES	Test cases namelists and useful files
XIOS	XIOS I/O server library
`create_config.bash`	Script to setup your configuration. It creates a configuration directory, and copy useful files in it from **croco** and **croco_tools** sources

5.2.2 croco_tools

CROCO preprocessing tools have been primarily developed under Matlab software by IRD researchers (former Roms_tools). Note: These tools have been made to build easily regional configurations using climatological data. To use interannual data, some facilities are available (NCEP, CFSR, QuickScat data for atmospheric forcing, SODA and ECCO for lateral boundaries). However, to use other data, you will need to adapt the scripts. All utilities/toolbox requested for matlab crocotools programs are provided within UTILITIES directory, or can be downloaded here: http://www.croco-ocean.org/download/utilities/.

Scripts

Aforc_CFSR	Scripts for the recovery of surface forcing data (based on CFSR reanalysis) for interannual simulations
Aforc_ECMWF	Scripts for the recovery of surface forcing data (based on ECMWF-ERAinterim simulations) for interannual simulations
Aforc_ERA5	Scripts for the recovery of surface forcing data (based on ECMWF-ERA5 simulations) for interannual simulations
Aforc_NCEP	Scripts for the recovery of surface forcing data (based on NCEP2 reanalysis) for interannual simulations
Aforc_QuikSCAT	Scripts for the recovery of wind stress from satellite scatterometer data (QuickSCAT)
Coupling_tools	Scripts for preparing coupled simulations
Diagnostic_tools	A few Matlab scripts for animations and basic statistical analysis
Forecast_tools	Scripts for the generation of an operational oceanic forecast system
Nesting_tools	Preprocessing tools used to prepare nested models
Oforc_OGCM	Scripts for the recovery of initial and lateral boundary conditions from global OGCMs (SODA (Carton et al., 2005) or ECCO (Stammer et al., 1999)) for inter-annual simulations
Opendap_tools	LoadDAP mexcdf and several scripts to automatically download data over the Internet
Preprocessing_tools	Preprocessing Matlab scripts (make_grid.m, make_forcing, etc…)
Rivers	Scripts to prepare time-varying runoff forcing file and compute the runoff location
Tides	Matlab routines to prepare CROCO tidal simulations. Tidal data are derived from the Oregon State University global models of ocean tides TPXO6 and TPXO7 (Egbert and Erofeeva, 2002): http://www.oce.orst.edu/research/po/research/tide/global.html
Visualization_tools	Matlab scripts for the CROCO visualization graphic user interface
croco_pyvisu	Python toolbox for CROCO visualization graphic user interface

UTILITIES

mask	Land mask edition toolbox developed by A.Y. Shcherbina.
mex60	Matlab NetCDF interface for 32 & 64 bits Linux architectures and old matlab version: 6 and before
mexcdf/mexnc	Matlab NetCDF interface for 32 & 64 bits Linux architectures, MatlabR14sp1 until R2008a (http://mexcdf.sourceforge.net/downloads/mexcdf-R2008a.r2691.zip). For next releases of Matlab, R2008b, R2009a, it is more simpler, either use the native NetCDF toobox of matlab or use the last release of mexcf at the same url for version after R2008a. (http://mexcdf.sourceforge.net/downloads/mexcdf.r2802.zip)
mexcdf/netcdf_toolbox	The Matlab NetCDF toolbox available in the same mexcdf package.
m_map	The Matlab mapping toolbox (http://www2.ocgy.ubc.ca/rich/map.html).
netcdf_x86_64	The NetCDF Fortran library for Linux, compiled with ifort on a 64 bits architecture.

DATASETS

CARS2009	CSIRO Atlas of Regional Seas database. Annual, seasonal and monthly climatology for temperature, salinity, nitrate, phosphate and oxygen
COADS05	Directory of the surface fluxes global monthly climatology at resolution (Da Silva et al., 1994)
GOT99.2	Atlas of the loading tide for M2 S2 N2 K2 K1 O1 P1 Q1
QuikSCAT_clim	QuickSCAT monthly climatology of wind stress
RUNOFF_DAI	River discharge monthly climatology in $m.s^{-3}$ for the 925 largest rivers reaching the ocean (from Dai en Trenberth, 2000)
SST_pathfinder	SST global monthly climatology at a finer resolution (9.28 km) than COADS05, computed from AVHRR-Pathfinder observations from 1985 to 1997 (Casey and Cornillon, 1999)
SeaWifs	Surface chlorophyll-a climatology based on SeaWifs observations
TPX07	Directory of the global model of ocean tides TPXO7 (Egbert and Erofeeva, 2002)
Topo	Location of the global topography dataset at 2° resolution (Smith and Sandwell, 1997). Original data can be found at: http://topex.ucsd.edu/cgi-bin/get_data.cgi
WOA2009	World Ocean Atlas 2009 global datase References list: http://www.nodc.noaa.gov/OC5/WOA09/pubwoa09.html
WOAPISCES	A global dataset for biogeochemical PISCES data (annual and seasonal climatology). References are : Fe and DOC : Aumont et Bopp, 2006 Si, O2, NO3, PO4 from WOA2005, DIC and Alkalinity come from Goyet et al.

CHAPTER

SIX

SUMMARY OF ESSENTIAL STEPS

1. **Compilation** CROCO needs to be compiled for each configuration (grid, MPI decomposition, paramterizations...). The files that need to be edited are (available in croco/OCEAN directory):

`cppdefs.h`	CPP-keys* allowing to select configuration, numerical schemes, parameterizations, forcing and boundary conditions * *CROCO extensively uses the C preprocessor (cpp) during compilation to replace code statements, insert files into the code, and select relevant parts of the code depending on its directives.*
`param.h`	Grid settings: the values of the model grid size are: LLm0 points in the X direction MMm0 points in the Y direction N vertical levels For realistic regional cases, LLm0 and MMm0 are given by running `make_grid.m`, and N is defined in `crocotools_param.m` `param.h` also contains: Parallelisation settings Tides, Wetting-Drying, Point sources, Floats, Stations specifications
`jobcomp`	the compilation script (including settings for paths, compilers, libraries, etc)

2. **Namelist** CROCO namelist input file `croco.in` contains several configurations settings such as: the time stepping, the vertical coordinate settings, the I/O settings and paths, some parameters for the model, ... It has to be edited before running. It is available in croco/OCEAN directory for regional configurations, and in croco/TEST_CASES directory for test cases.

3. **Input files** CROCO needs the following input files to run:

   - CROCO grid file: `croco_grd.nc`
   - CROCO surface forcing file: `croco_frc.nc` (or `croco_blk.nc`)
   - CROCO vertical boundary conditions: `croco_bry.nc` (or `croco_clim.nc`)
   - CROCO initial conditions: `croco_ini.nc`

They can be created using the Preprocessing croco_tools, see dedicated tutorial. These files are eventually not mandatory in test cases for which the useful settings are defined analytically within the CROCO code.

4. **Run** CROCO can be run in serial or parallel mode. See the run tutorial.
5. **Outputs** CROCO usual outputs are:
   - CROCO restart file: `croco_rst.nc`
   - CROCO instantaneous output file: `croco_his.nc`
   - CROCO averaged output file: `croco_avg.nc`
   - CROCO log file: `croco.log` if you have defined the LOGFILE key in cppdefs.h : # define LOGFILE

   Other output files can be generated depending on the settings provided in `croco.in`.

CHAPTER

SEVEN

TEST CASES

7.1 BASIN

1. **Create a configuration directory:**

```
mkdir ~/CONFIGS/BASIN
```

2. **Copy the input files for compilation from croco sources:**

```
cd ~/CONFIGS/BASIN
cp ~/croco/croco/OCEAN/cppdefs.h .
cp ~/croco/croco/OCEAN/param.h .
cp ~/croco/croco/OCEAN/jobcomp .
```

3. **Edit `cppdefs.h` for using BASIN case**

```
# define BASIN

# undef REGIONAL
```

   You can also explore the CPP options selected for BASIN case.

   You can check the BASIN settings in `param.h`.

4. **Edit the compilation script `jobcomp`:**

```
# set source, compilation and run directories
#
SOURCE=~/croco/croco/OCEAN
SCRDIR=./Compile
RUNDIR=`pwd`
ROOT_DIR=$SOURCE/..
#
# determine operating system
#
OS=`uname`
echo "OPERATING SYSTEM IS: $OS"

#
# compiler options
#
FC=$FC

#
# set MPI directories if needed
#
MPIF90=$MPIF90
MPIDIR=$(dirname $(dirname $(which $MPIF90) ))
MPILIB="-L$MPIDIR/lib -lmpi -limf -lm"
```

(continues on next page)

(continued from previous page)

```
MPIINC="-I$MPIDIR/include"

# set NETCDF directories
#
#------------------------------------------------------------
# Use :
#-lnetcdf           : version netcdf-3.6.3                --
#-lnetcdff -lnetcdf : version netcdf-4.1.2                --
#-lnetcdff          : version netcdf-fortran-4.2-gfortran --
#------------------------------------------------------------
#
#NETCDFLIB="-L/usr/local/lib -lnetcdf"
#NETCDFINC="-I/usr/local/include"
NETCDFLIB=$(nf-config --flibs)
NETCDFINC=-I$(nf-config --includedir)
```

5. Compile the model:

   - By using classical launch command (on individual computers):

     ```
     ./jobcomp > jobcomp.log
     ```

   - **OR** by using a batch script (*e.g.* PBS) to launch the model (in clusters): For DATARMOR:

     ```
     cp $CROCO_DIR/job_comp_datarmor.pbs .
     qsub job_comp_datarmor.pbs
     ```

   If compilation is successful, you should have a `croco` executable in your directory.

   You will also find a `Compile` directory containing the model source files:

   - `.F` files: original model source files that have been copied from `$croco/OCEAN`
   - `_.f` files: pre-compiled files in which only parts defined by cpp-keys are kept
   - `.o` object files

6. **Copy the namelist input file for BASIN case:**

   ```
   cp ~/croco/croco/TEST_CASES/croco.in.Basin croco.in
   ```

   Eventually edit it.

7. **Run the model:**

   ```
   ./croco croco.in > croco.out
   ```

   If your run is successful you should obtain the following files:

   ```
   basin_rst.nc # restart file
   basin_his.nc # instantaneous output file
   ```

8. **Have a look at the results:**

   ```
   ncview basin_his.nc
   ```

9. **Test: some questions:**

   - What is the size of the grid (see param.h)?
   - What are the name of the horizontal directions?
   - What is the spatial resolution in both horizontal directions?
   - How many vertical levels do you have?

- How are the vertical levels distributed (look for the cpp key `NEW_S_COORD`)?
- What are the initial dynamical conditions (see both cppdefs.h and croco.in)?
- What do the air-sea exchanges look like?

10. Re-run this case in parallel on 4 CPUs:

    To run in parallel, your first need to edit `cppdefs.h`, `param.h`, and to recompile.

    - Edit `cppdefs.h`:

      ```
      # define MPI
      ```

    - Edit `param.h`:

      ```
      #ifdef MPI
      integer NP_XI, NP_ETA, NNODES
      parameter (NP_XI=2,  NP_ETA=2,  NNODES=NP_XI*NP_ETA)
      parameter (NPP=1)
      parameter (NSUB_X=1, NSUB_E=1)
      ```

      Note: MPI tiles should be at least 20x20 points.

    - Recompile.
    - Run the model in parallel:
      - By using classical launch command (on individual computers):

        ```
        mpirun -np NPROCS croco croco.in
        ```

        where NPROCS is the number of CPUs you want to allocate. `mpirun -np NPROCS` is a typical mpi command, but it may be adjusted to your MPI compiler and machine settings.

      - **OR** by using a batch script (*e.g.* PBS) to launch the model (in clusters), examples are provided:

        ```
        cp ~/croco/croco_tools/job_croco_mpi.pbs .
        ```

        Edit `job_croco_mpi.pbs` according to your MPI settings in `param.h` and launch the run:

        ```
        qsub job_croco_mpi.pbs
        ```

Warning: NPROCS needs to be consistent to what you indicated in `param.h` during compilation

7.2 Set up you own test case

Example: set up a convection test case: test case that mimic the winter convection happening in the North-Western Mediterranean sea

1. **Create a configuration directory:**

   ```
   mkdir ~/CONFIGS/CONVECTION
   ```

2. **Copy the input files from croco sources:**

```
cd ~/CONFIGS/CONVECTION
cp ~/croco/croco/OCEAN/cppdefs.h .
cp ~/croco/croco/OCEAN/param.h .
cp ~/croco/croco/OCEAN/jobcomp .
cp ~/croco/croco/OCEAN/croco.in .
```

3. Edit `cppdefs.h`, `param.h`, and `croco.in` for your new CONVECTION case:
   - Add a dedicated key for this test case CONVECTION (in `cppdefs.h`)
   - Set up a flat bottom ; 2500 m deep (variable depth in `ana_grid.F` and follow what is perform under the key BASIN for instance)
   - Set up your grid: 1000x1000x200 grid points (respectively in xi, eta and vertical directions) (parameters LLm0, MMm0 and N in `param.h`)
   - Specify a length and width of 50km in both directions (xi, eta) (variables Length_XI,Length_ETA in `ana_grid.F`)
   - Set up an almost cold start with velocity component fields set to a white noise (see in `ana_initial.F` what is performed for other test cases and fill in arrays u,v) around 0.1 mm/s
   - Set up the initial ssh fields to zero (arrays zeta in `ana_initial.F`)
   - Set up the initial stratification (i.e. the temperature and salinity fields) (in `ana_initial.F`: array t)
   - Set up the wind stress forcing (svstr, sustr in `analytical.F` ; you may follow what is set for INNERSHELF ; not necessary)
   - Set the permanent heat surface flux (stflx= -500 w/m2 (-500/rho0*Cp) in `analytical.F` in subroutine `ana_stflux_tile`)

> **Warning:** In cppdefs.h define your own cpp key CONVECTION which might be a clone of the key BASIN ; in case we add the salinity (with respect to the BASIN case) do not forget to add the keys ANA_SSFLUX and ANA_BSFLUX .

> **Warning:** In croco.in in case we add (with respect to the BASIN case) the salinity do not forget to modify the number of tracers written 2*T and the number of Akt (2*.1.0e-6)

> **Warning:** In croco.in adjust the time step and ndtfast to reach the stability

4. **Edit the compilation script `jobcomp`:**

```
# set source, compilation and run directories
#
SOURCE=~/croco/croco/OCEAN
SCRDIR=./Compile
RUNDIR=`pwd`
ROOT_DIR=$SOURCE/..

#
# compiler options
#
FC=$FC

#
```

(continues on next page)

(continued from previous page)

```
# set MPI directories if needed
#
MPIF90=$MPIF90
MPIDIR=$(dirname $(dirname $(which $MPIF90) ))
MPILIB="-L$MPIDIR/lib -lmpi -limf -lm"
MPIINC="-I$MPIDIR/include"

# set NETCDF directories
#
#------------------------------------------------------------
# Use :
#-lnetcdf           : version netcdf-3.6.3                --
#-lnetcdff -lnetcdf : version netcdf-4.1.2                --
#-lnetcdff          : version netcdf-fortran-4.2-gfortran --
#------------------------------------------------------------
#
#NETCDFLIB="-L/usr/local/lib -lnetcdf"
#NETCDFINC="-I/usr/local/include"
NETCDFLIB=$(nf-config --flibs)
NETCDFINC=-I$(nf-config --includedir)
```

5. **Compile the model:**

```
./jobcomp > jobcomp.log
```

   If compilation is successful, you should have a `croco` executable in your directory.

6. **Run the model:**

   - Classical launch command is (but should probably be launched in a dedicated submission job in clusters... see next item):

     ```
     mpirun -np NPROCS croco croco.in
     ```

     where NPROCS is the number of CPUs you want to allocate. `mpirun -np NPROCS` is a typical mpi command, but it may be adjusted to your MPI compiler and machine settings.

   - **OR** by using a batch script (*e.g.* PBS) to launch the model (in clusters), examples are provided:

     ```
     cp ~/croco/croco_tools/job_croco_mpi.pbs .
     ```

     Edit `job_croco_mpi.pbs` according to your MPI settings in `param.h` and launch the run:

     ```
     qsub job_croco_mpi.pbs
     ```

     Warning: NPROCS needs to be consistent to what you indicated in `param.h` during compilation

7. **If you want to try another mixing parameterization:**

   - Add in `cppdefs.h`, in your CONVECTION case, the following cpp keys dedicated to the closure:

```
#define GLS_MIXING
```

   - In croco.in add this lines for GLS history and averages fields

```
gls_history_fields:   TKE  GLS  Lscale
                       F     F    F
gls_averages:   TKE  GLS  Lscale
                F     F    F
```

- Recompile, and re-run the model

8. **If you want to add stflux as tanh signal:**

   - in `analytical.F`:

```
real*4 r2,RR2
!  Set kinematic surface heat flux [degC m/s] at horizontal
!  RHO-points.
!
    r2= 10**2
    ic = LLm0/2.
    jc = MMm0/2.
    do j=JstrR,JendR
      do i=IstrR,IendR
        RR2 =  (i+iminmpi-ic)*(i+iminmpi-ic)+(jminmpi-
↪jc)*(jminmpi-jc)
        stflx(i,j,itemp)=(-200. -200. * tanh((r2-RR2)/1000.))/
↪rho0/Cp
      enddo
    enddo
```

   - Recompile and re-run the model.

CHAPTER

EIGHT

REGIONAL: PREPARING YOUR CONFIGURATION

To prepare your configuration working directory, you can use the script `create_config.bash` provided in CROCO sources:

```
cp ~/croco/croco/create_config.bash ~/CONFIGS/.
```

Edit your paths and settings in `create_config.bash`:

```
↪#=========================================================================================
# BEGIN USER MODIFICATIONS

# Machine you are working on
# Known machines: Linux DATARMOR IRENE JEANZAY
# -------------------------------------------
MACHINE="Linux"

# CROCO parent directory
# (where croco_tools directory and croco source directory can be found)
# ------------------
CROCO_DIR=~/croco/croco
TOOLS_DIR=~/croco/croco_tools

# Configuration name
# ------------------
MY_CONFIG_NAME=BENGUELA_LR

# Home and Work configuration directories
# ---------------------------------------
MY_CONFIG_HOME=~/CONFIGS
MY_CONFIG_WORK=~/CONFIGS

# Options of your configuration
models_incroco=( all-prod )
```

Run `create_config.bash`:

```
./create_config.bash
```

A directory named BENGUEAL_LR should be created.

You can also manually create your configuration directory, by copying the required files from croco sources:

```
mkdir ~/CONFIGS/BENGUELA_LR
cd ~/CONFIGS/BENGUELA_LR

# For pre-processing:
cp ~/croco/croco_tools/crocotools_param.m .
cp ~/croco/croco_tools/start.m .
```

(continues on next page)

(continued from previous page)

```
# For compiling
cp ~/croco/croco/OCEAN/cppdefs.h .
cp ~/croco/croco/OCEAN/param.h .
cp ~/croco/croco/OCEAN/jobcomp .

# For running
cp ~/croco/croco/OCEAN/croco.in .
```

In your configuration working directory, you need at least the following files:

- For preprocessing:
  - crocotools_param.m
  - start.m
- For compiling:
  - param.h
  - cppdefs.h
  - jobcomb
- For running:
  - croco.in

CHAPTER

NINE

REGIONAL: PREPROCESSING (MATLAB)

CROCO preprocessing tools have been developed under Matlab software by IRD researchers (former Roms_tools). Note: These tools have been made to build easily regional configurations using climatological data. To use interannual data, some facilities are available (NCEP, CFSR, QuickScat data for atmospheric forcing, SODA and ECCO for lateral boundaries). However, to use other data, you will need to adapt the scripts. All utilities/toolbox requested for matlab crocotools programs are provided within the UTILITIES directory, or can be downloaded here: http://www.croco-ocean.org/download/utilities/

9.1 Contents of the `croco_tools`

Aforc_CFSR	Scripts for the recovery of surface forcing data (based on CFSR reanalysis) for interannual simulations
Aforc_ECMWF	Scripts for the recovery of surface forcing data (based on ECMWF-ERAinterim simulations) for interannual simulations
Aforc_ERA5	Scripts for the recovery of surface forcing data (based on ECMWF-ERA5 simulations) for interannual simulations
Aforc_NCEP	Scripts for the recovery of surface forcing data (based on NCEP2 reanalysis) for interannual simulations
Aforc_QuikSCAT	Scripts for the recovery of wind stress from satellite scatterometer data (QuickSCAT)
Coupling_tools	Scripts for preparing coupled simulations
Diagnostic_tools	A few Matlab scripts for animations and basic statistical analysis
Forecast_tools	Scripts for the generation of an operational oceanic forecast system
Nesting_tools	Preprocessing tools used to prepare nested models
Oforc_OGCM	Scripts for the recovery of initial and lateral boundary conditions from global OGCMs (SODA (Carton et al., 2005) or ECCO (Stammer et al., 1999)) for inter-annual simulations
Opendap_tools	LoadDAP mexcdf and several scripts to automatically download data over the Internet
Preprocessing_tools	Preprocessing Matlab scripts (make_grid.m, make_forcing, etc. . .)
Rivers	Scripts to prepare time-varying runoff forcing file and compute the runoff location
Tides	Matlab routines to prepare CROCO tidal simulations. Tidal data are derived from the Oregon State University global models of ocean tides TPXO6 and TPXO7 (Egbert and Erofeeva, 2002): http://www.oce.orst.edu/research/po/research/tide/global.html
Visualization_tools	Matlab scripts for the CROCO visualization graphic user interface
croco_pyvisu	Python toolbox for CROCO visualization graphic user interface
UTILITIES	utilities/toolbox requested for matlab crocotools programs

9.2 Philosophy of the `croco_tools`

- 2 scripts are used to set-up your Matlab environment and your configuration settings:

start.m	useful paths for croco_tools Matlab scripts
crocotools_param.m	namelist file for Matlab pre-processing

  - `start.m`: has to be launched at the beginning of any matlab session to set the path to utilities and croco tools routines. Edit mypath and myutilpath
  - `crocotools_param.m`: defines all the parameters and paths needed to build the grid, forcing and boundary files. Edit the different sections.

Note: In the `croco_tools` toolbox, the native Matlab Netcdf library is not used. A dedicated Netcdf library is provided and used. Its path is added to your Matlab environment through the the `start.m` script.

- Steps for creating a configuration are:
  - build the grid
  - build the atmospheric forcing (not necessary when coupling with an atmospheric model)
  - build the lateral boundary conditions (3D currents, temperature and salinity, barotropic currents, surface elevation)
  - build the initial conditions

9.3 Climatological pre-processing

First we will start by preparing surface and boundary conditions from climatological datasets. Those datasets can be downloaded on CROCO website:

:: https://www.croco-ocean.org/download/datasets/

CARS2009	CSIRO Atlas of Regional Seas database. Annual, seasonal and monthly climatology for temperature, salinity, nitrate, phosphate and oxygen
COADS05	Directory of the surface fluxes global monthly climatology at resolution (Da Silva et al., 1994)
GOT99.2	Atlas of the loading tide for M2 S2 N2 K2 K1 O1 P1 Q1
QuikSCAT_clim	QuickSCAT monthly climatology of wind stress
RUNOFF_DAI	River discharge monthly climatology in $m.s^{-3}$ for the 925 largest rivers reaching the ocean (from Dai en Trenberth, 2000)
SST_pathfinder	SST global monthly climatology at a finer resolution (9.28 km) than COADS05, computed from AVHRR-Pathfinder observations from 1985 to 1997 (Casey and Cornillon, 1999)
SeaWifs	Surface chlorophyll-a climatology based on SeaWifs observations
TPX07	Directory of the global model of ocean tides TPXO7 (Egbert and Erofeeva, 2002)
Topo	Location of the global topography dataset at 2° resolution (Smith and Sandwell, 1997). Original data can be found at: http://topex.ucsd.edu/cgi-bin/get_data.cgi
WOA2009	World Ocean Atlas 2009 global datase References list: http://www.nodc.noaa.gov/OC5/WOA09/pubwoa09.html
WOAPISCES	A global dataset for biogeochemical PISCES data (annual and seasonal climatology). References are : Fe and DOC : Aumont et Bopp, 2006 Si, O2, NO3, PO4 from WOA2005, DIC and Alkalinity come from Goyet et al.

1. **First you may need to edit `start.m`, which contains the path to all useful croco_tools Matlab scripts:**

```
disp(['Add the paths of the different toolboxes'])
tools_path=['~/croco/croco_tools/'];
croco_path=['~/croco/croco/'];
```

Note: You can use these env variables in matlab by using `getenv('ENVVAR')`, example: if you have a `$tools` environment variables for croco_tools, you can write in start.m: `tools_path=[getenv('tools') '/'];`

2. **Then edit `crocotools_param.m`, which is the namelist file for Matlab pre-processing:**
`crocotools_param.m` is separated into several sections:

1 - Configuration parameters	used by make_grid.m (and others..)
2 - Generic file and directory names	need to match your work architecture
3 - Surface forcing parameters	used by make_forcing.m and by make_bulk.m
4 - Open boundaries and initial conditions parameters	used by make_clim.m, make_biol.m, make_bry.m make_OGCM.m and make_OGCM_frcst.m
5 - Parameters for tidal forcing	used by make_tides.m
6 - Reference date and simulation times	used for make_tides, make_CFSR (or make_NCEP), make_OGCM
7 - Parameters for Interannual forcing	SODA, ECCO, CFSR, NCEP, ...
8 - Parameters for the forecast system	used by make_forecast.m
9 - Parameters for the diagnostic tools	used by scripts in `Diagnostic_tools`

The first section is already set for BENGUELA_LR configuration, so you just need to change the second section: directory names:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% 2 - Generic file and directory names
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%
%  CROCOTOOLS directory
%
CROCOTOOLS_dir = ['~/croco/croco_tools/'];
%
%  Run directory
%
RUN_dir=[pwd,'/'];
%
%  CROCO input netcdf files directory
%
CROCO_files_dir=[RUN_dir,'CROCO_FILES/'];
%
%  Global data directory (etopo, coads, datasets download from ftp, etc..)
%
DATADIR=['~/DATA/DATASETS_CROCOTOOLS/'];
%
%  Forcing data directory (ncep, quikscat, datasets download with opendap,␣
↪etc..)
%
FORC_DATA_DIR = ['~/DATA/'];
```

Note: The `crocotools_param.m` is called at the beginning of all Preprocessing script. You do not have to launch it independently.

3. **Now you are ready to launch pre-processing in Matlab:**

Note: All the pre-processing scripts used for climatological forcing are in the `Preprocessing_tools` directory

Launch Matlab, and set up paths:

```
matlab
start
```

Build the grid:

```
make_grid
```

During the grid generation process, the question "Do you want to use editmask ? y,[n]" is asked. The default answer is n (for no). If the answer is y (for yes), editmask, the graphic interface developed by A.Y.Shcherbina, will be launched to manually edit the mask. Otherwise the mask is generated from the unfiltered topography data. A procedure prevents the existence of isolated land (or sea) points.

Finally, a figure illustrates the obtained bottom topography. Note that at his low resolution (1/3°), the topography has been strongly smoothed.

Build the atmospheric forcing: 2 options are available:

- create a forcing file with wind stress (zonal and meridional components), surface net heat flux, surface freshwater flux (E-P), solar shortwave radiation, SST, SSS, surface net sensitivity to SST (used for heat flux correction dQdSST for nudging towards model SST and model SSS)
- or create a bulk file which will be read during the run to perform bulk parameterization of the fluxes using COAMPS or Fairall 2003 formulation. This bulk file contains: surface air temperature, relative humidity, precipitation rate, wind speed at 10m, net outgoing longwave radiation, downward longwave radiation, shortwave radiation, surface wind speed (zonal and meridional components). It also contains surface wind stress (zonal and meridional components), but it is not requested and used in the model (except for specific debugging work). The bulk formulation computes its own wind stress.

```
make_bulk
```

or:

```
make_forcing
```

The settings relative to surface forcing are in section 3 of `crocotools_param.m`. In the case of climatological forcing, the variables are cycled. You can see that here, for the sake of simplicity, we are running the model on a repeating climatological year of 360 days.

A few figures illustrate the wind stress vectors and norm at 4 different periods of the year.

Note: `make_bulk` creates a forcing file that will be used with the cpp key `BULK_FLUX`, while `make_forcing` creates a forcing file containing wind stress directly and will be used when `undefined BULK_FLUX`. This second option is relevant if your atmospheric forcing comes from an atmospheric model with sufficient output frequency, or/and if your are comparing forced and coupled runs. Otherwise it is suggested to use `make_bulk`.

Build the lateral boundary conditions: 2 options are available:

```
make_bry
```

Note: `make_bry` requests that you have previously run `make_forcing` to compute Ekman forcing at the surface.

or:

```
make_clim
```

The settings relative to boundary conditions are in section 4 of `crocotools_param.m`.

A few figures illustrate vertical sections of temperature.

Note: `make_clim` interpolates the oceanic forcing fields over the whole domain: only boundary points + the 10 next points are actually used for sponge + nudging. *Advantage*: sponge + nudging layers at the boundaries, *Disadvantage*: large amount of unused data.

`make_bry` interpolates the oceanic forcing fields at the boundary points only. *Advantage*: light files (useful for long simulations), *Disadvantage*: no nudging layers (only a sponge layer for smooth transition between the boundaries and the interior values).

Build the initial conditions:

```
make_ini
```

4. **You can look at your generated input files in `CROCO_FILES` directory:** You should have:

```
croco_grd.nc
croco_ini.nc
croco_blk.nc # or croco_frc.nc
croco_bry.nc # or croco_clm.nc
```

5. **Summary to create a simple configuration from climatology files:** In Matlab, execute the following:

```
start
make_grid
make_forcing
make_bulk
make_bry       # or make_clim
make_ini
```

This will create:

```
croco_grd.nc
croco_frc.nc (or croco_blk.nc)
croco_bry.nc (or croco_clim.nc)
croco_ini.nc
```

9.4 Interannual pre-processing

Dedicated scripts for interannual pre-processing can be found for the different forcing datasets in:

Aforc_CFSR	Scripts for the recovery of surface forcing data (based on CFSR reanalysis) for interannual simulations
Aforc_ECMWF	Scripts for the recovery of surface forcing data (based on ECMWF-ERAinterim simulations) for interannual simulations
Aforc_ERA5	Scripts for the recovery of surface forcing data (based on ECMWF-ERA5 simulations) for interannual simulations
Aforc_NCEP	Scripts for the recovery of surface forcing data (based on NCEP2 reanalysis) for interannual simulations
Aforc_QuikSCAT	Scripts for the recovery of wind stress from satellite scatterometer data (QuickSCAT)
Forecast_tools	Scripts for the generation of an operational oceanic forecast system
Oforc_OGCM	Scripts for the recovery of initial and lateral boundary conditions from global OGCMs (SODA (Carton et al., 2005), ECCO (Stammer et al., 1999) or CMEMS-GLORYS12) for inter-annual simulations

1. **Edit `crocotools_param.m`** First section should already be set if you have completed the previous tutorial.

   In the second section, check the path to forcing data directory:

```
% 2 - Generic file and directory names

%  Forcing data directory (ncep, quikscat, datasets download with opendap,␣
↪etc..)
%
FORC_DATA_DIR = ['~/DATA/'];
```

   In section 4, select only ini and bry (but no clim files, set: `makeclim = 0;`) to avoid too long pre-processing, and as it is the most usual set up:

```
%  initial/boundary data options (1 = process)
%  (used in make_clim, make_biol, make_bry,
%   make_OGCM.m and make_OGCM_frcst.m)
%
```

(continues on next page)

(continued from previous page)

```
makeini     = 1;    % initial data
makeclim    = 0;    % climatological data (for boundaries and nudging layers)
makebry     = 1;  @ % lateral boundary data
```

Edit section 6 for running January to March 2005:

```
% 6 - Reference date and simulation times

Ymin          = 2005;          % first forcing year
Ymax          = 2005;          % last  forcing year
Mmin          = 1;             % first forcing month
Mmax          = 3;             % last  forcing month
```

Note: An important aspect is the definition of time and especially the choice of a time origin. The origin of time Yorig should be kept the same for all the preprocessing and postprocessing steps.

Edit section 7 for using CFSR and SODA forcing sets:

```
% 7 - Parameters for Interannual forcing (SODA, ECCO, CFSR, NCEP, ...)

Download_data = 0;    % Get data from OPENDAP sites
level         = 0;    % AGRIF level; 0 = parent grid
%
NCEP_version  = 3;    % NCEP version:
                      % [ CFSR up-to-date product are recommandated ]
                      %  1: NCEP/NCAR Reanalysis, 1/1/1948 - present
                      %  2: NCEP-DOE Reanalysis, 1/1/1979 - present
                      %  3: CFSR (Climate Forecast System Reanalysis),
                      %           1/1/1979 - 31/3/2011

NCEP_dir = [FORC_DATA_DIR,'CFSR_',CROCO_config,'/']; % CFSR data dir.␣
↪[croco format]
makefrc       = 0;       % 1: create forcing files
makeblk       = 1;       % 1: create bulk files
QSCAT_blk     = 0;       % 1: a) correct NCEP frc/bulk files with
                         %         u,v,wspd fields from daily QSCAT data
                         %    b) download u,v,wspd in QSCAT frc file
add_tides     = 0;       % 1: add tides

% ...

OGCM        = 'SODA';        % Select the OGCM: SODA, ECCO
%
OGCM_dir    = [FORC_DATA_DIR,OGCM,'_',CROCO_config,'/'];  % OGCM data dir.␣
↪[croco format]
```

2. **Then you can run the Matlab pre-processing for these interannual forcing:** You should already have you grid set up. Otherwise, run `make_grid`

   To build your interannual atmospheric forcing, the useful script is `make_CFSR`

   To build your interannual ocean forcing, he useful script is `make_OGCM`

```
start
make_CFSR
make_OGCM
```

Warning:

As this pluri-month preprocessing can be longer and uses more CPU ressources, you may need to submit it as a

```
cp ~/croco/croco_tools/example_job_prepro_matlab.pbs .
```

Launch your pre-processing job:

```
qsub example_job_prepro_matlab.pbs
```

3. **Check your generated files in `CROCO_FILES`** You should have:

```
croco_blk_CFSR_Y????M?.nc
croco_bry_SODA_Y????M?.nc
croco_ini_SODA_Y????M?.nc
```

CHAPTER

TEN

COMPILING

The files that you need to edit for compilation are:

cppdefs.h	CPP-keys* allowing to select configuration, numerical schemes, parameterizations, forcing and boundary conditions ** CROCO extensively uses the C preprocessor (cpp) during compilation to replace code statements, insert files into the code, and select relevant parts of the code depending on its directives.*
param.h	Grid settings: the values of the model grid size are: LLm0 points in the X direction MMm0 points in the Y direction N vertical levels For realistic regional cases, LLm0 and MMm0 are given by running `make_grid.m`, and N is defined in `crocotools_param.m` `param.h` also contains: Parallelisation settings Tides, Wetting-Drying, Point sources, Floats, Stations specifications
jobcomp	the compilation script (including settings for paths, compilers, libraries, etc)

Warning: CROCO needs to be compiled for each configuration (domain, coupled, uncoupled, parameterizations...), *i.e.*, each time you change something in `cppdefs.h` or `param.h`

Let's explore, check, and edit the 3 aforementionned files:

10.1 cppdefs.h

Let's explore, check, and edit: `cppdefs.h`

1. **First section of `cppdefs.h` defines your configuration (test case or realistic regional case):**

```
#undef  BASIN           /* Basin Example */
#undef  CANYON          /* Canyon Example */
#undef  EQUATOR         /* Equator Example  */
#undef  INNERSHELF      /* Inner Shelf Example */
#undef  RIVER           /* River run-off Example */
#undef  OVERFLOW        /* Graviational/Overflow Example */
#undef  SEAMOUNT        /* Seamount Example */
#undef  SHELFRONT       /* Shelf Front Example */
#undef  SOLITON         /* Equatorial Rossby Wave Example */
#undef  THACKER         /* Thacker wetting-drying Example */
#undef  UPWELLING       /* Upwelling Example */
#undef  VORTEX          /* Baroclinic Vortex Example */
#undef  INTERNAL        /* Internal Tide Example */
#undef  IGW             /* COMODO Internal Tide Example */
#undef  JET             /* Baroclinic Jet Example */
#undef  SHOREFACE       /* Shoreface Test Case on a Planar Beach */
#undef  RIP             /* Rip Current Test Case */
#undef  SANDBAR         /* Bar-generating Flume Example */
#undef  SWASH           /* Swash Test Case on a Planar Beach */
#undef  TANK            /* Tank Example */
#undef  ACOUSTIC        /* Acoustic wave Example */
#undef  GRAV_ADJ        /* Graviational Adjustment Example */
#undef  ISOLITON        /* Internal Soliton Example */
#undef  KH_INST         /* Kelvin-Helmholtz Instability Example */
#undef  TS_HADV_TEST    /* Horizontal tracer advection Example */
#define REGIONAL        /* REGIONAL Applications */
```

   For the BENGUELA_LR case we are running, you should have:

```
#define REGIONAL        /* REGIONAL Applications */
```

2. **Then, in `cppdefs.h`, you have one section for each case. Let's explore the REGIONAL case section:**

   - First is the name of your configuration:

```
#if defined REGIONAL
/*
!======================================================================
!               REGIONAL (realistic) Configurations
!======================================================================
!
!----------------------
! BASIC OPTIONS
!----------------------
!
*/
                      /* Configuration Name */
# define BENGUELA_LR
```

   - Then, you can set parallelization option (you can set `define MPI` if you want to run in parallel):

```
/* Parallelization */
# undef  OPENMP
# undef  MPI
```

- Then, you can set I/O options (XIOS server, netcdf 4 parallel option, NB: we will have a dedicated tutorial on XIOS):

```
/* I/O server */
# undef  XIOS
```

- Non-hydrostatic option:

```
/* Non-hydrostatic option */
# undef  NBQ
```

- Nesting settings:

```
/* Nesting */
# undef  AGRIF
# undef  AGRIF_2WAY
```

- Coupling with other models (atmosphere, waves):

```
/* OA and OW Coupling via OASIS (MPI) */
# undef  OA_COUPLING
# undef  OW_COUPLING
```

- Including wave-current interactions:

```
/* Wave-current interactions */
# undef  MRL_WCI
```

- Managing open boundaries (you can choose to close one of the boundaries, useful in coastal cases):

```
/* Open Boundary Conditions */
# undef  TIDES
# define OBC_EAST
# define OBC_WEST
# define OBC_NORTH
# define OBC_SOUTH
```

- Activating applications:

```
/* Applications */
# undef  BIOLOGY
# undef  FLOATS
# undef  STATIONS
# undef  PASSIVE_TRACER
# undef  SEDIMENT
# undef  BBL
```

- Defining a dedicated log file for CROCO standard output (default is undef but you can `define LOGFILE` to facilitate the reading of model output, particularly useful for coupled simulations):

```
/* dedicated croco.log file */
# undef  LOGFILE
```

> **Warning:** Keep `undef LOGFILE` is you use Plurimonth run scritps as: `run_croco_inter.bash` because it already re-direct the CROCO output, and check it...

- Time reference setting:

> **Warning:** By default no reference time is used, and time is referred to the beginning of the simulation only

```
/* Calendar */
# undef USE_CALENDAR
```

3. **Then you have detailed settings (you can find a description of all cpp keys in `Contents and Architecture` sectio**

   - In grid configuration:

```
/* Grid configuration */
# define CURVGRID
# define SPHERICAL
# define MASKING
# undef  WET_DRY
# define NEW_S_COORD
```

> **Warning:** you should check that the vertical coordinate setting `NEW_S_COORD` is in adequation with your pre-processing setting (`vtransform=2` in `crocotools_param.m`)

   - In surface forcing subsection:
     - if you have prepared `croco_frc.nc` file (using `make_frc.m`):

```
/* Surface Forcing */
# undef BULK_FLUX
```

     - if you have prepared `croco_blk.nc` file (using `make_blk.m`):

```
/* Surface Forcing */
# define BULK_FLUX
```

   - Then, you have to set your lateral forcing according to your pre-processing as well:
     - If you have prepared `croco_clm.nc` file (using `make_clim.m`):

```
/* Lateral Forcing */
# define CLIMATOLOGY

and

# undef  FRC_BRY
```

     - Or, if you have prepared `croco_bry.nc` file (using `make_bry.m`):

```
/* Lateral Forcing */
# undef CLIMATOLOGY

and

# define FRC_BRY
```

The other CPP-keys will be explored in other tutorials.

10.2 param.h

`param.h` is composed of the following sections:

- Dimensions of Physical Grid and array dimensions
- MPI related variables
- Number maximum of weights for the barotropic mode
- OA-Coupling, Tides, Wetting-Drying, Point sources, Floast, Stations
- Derived dimension parameters
- I/O : flag for type sigma vertical transformation
- Number of tracers
- Tracer identification indices

Most of the time you only need to check/edit the 2 first sections:

1. **Check the grid settings:**

```
#  elif defined  BENGUELA_LR
      parameter (LLm0=41,    MMm0=42,    N=32)    ! BENGUELA_LR
```

   - **LLm0**: Dimension (ghost points included) in the ξ direction.
   - **MMm0**: Dimension (ghost points included) in the η direction.
   - **N**: Number of ρ-vertical points, in the vertical grid.

2. **Check and eventually edit the parallelization settings:**

```
#ifdef MPI
      integer NP_XI, NP_ETA, NNODES
      parameter (NP_XI=1,  NP_ETA=4,  NNODES=NP_XI*NP_ETA)
      parameter (NPP=1)
      parameter (NSUB_X=1, NSUB_E=1)
#elif defined OPENMP
      parameter (NPP=4)
```

   - In the case of OpenMP parallelization, NPP is the number of cpu used in the computation
   - In the case of MPI parallelization, it is equal to to NNODES.
   - AUTOTILING (implemented by L. Debreu): cpp-key that enable to compute the optimum sub-domains partition in terms of computation time.

Note: MPI tiles should be at least 20x20 points.

10.3 jobcomp

Now that your input files are set up, you can proceed to compilation:

Here we assume that you have set a few environment variables for compilers and libraries. Here is an example with Intel compilers and a netcdf library located in $HOME/softs/netcdf. Adapt these to your own settings (in your .bashrc file):

```
# compilers
export CC=icc
export FC=ifort
export F90=ifort
export F77=ifort
export MPIF90=mpiifort

# netcdf library
export NETCDF=$HOME/softs/netcdf
export PATH=$NETCDF/bin::${PATH}
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}::${NETCDF}/lib
```

1. **Edit the compilation script `jobcomp`:**

```
# set source, compilation and run directories
#
SOURCE=~/croco/croco/OCEAN
SCRDIR=./Compile
RUNDIR=`pwd`
ROOT_DIR=$SOURCE/..
#
# determine operating system
#
OS=`uname`
echo "OPERATING SYSTEM IS: $OS"

#
# compiler options
#
FC=$FC

#
# set MPI directories if needed
#
MPIF90=$MPIF90
MPIDIR=$(dirname $(dirname $(which $MPIF90) ))
MPILIB="-L$MPIDIR/lib -lmpi -limf -lm"
MPIINC="-I$MPIDIR/include"

# set NETCDF directories
#
#------------------------------------------------------------
# Use :
#-lnetcdf           : version netcdf-3.6.3               --
#-lnetcdff -lnetcdf : version netcdf-4.1.2               --
#-lnetcdff          : version netcdf-fortran-4.2-gfortran --
#------------------------------------------------------------
#
#NETCDFLIB="-L/usr/local/lib -lnetcdf"
#NETCDFINC="-I/usr/local/include"
NETCDFLIB=$(nf-config --flibs)
NETCDFINC=-I$(nf-config --includedir)
```

2. **Compile the model:**

```
./jobcomp > jobcomp.log
```

If compilation is successful, you should have a `croco` executable in your directory.

You will also find a `Compile` directory containing the model source files:

- `.F` files: original model source files that have been copied from `~/croco/croco/OCEAN`
- `_.f` files: pre-compiled files in which only parts defined by cpp-keys are kept

- `.o` object files

10.4 Compilation options

A very summarized information on compilation options is given here. For further details, search information on the web, or with your cluster assistance team. Useful informations can also be found on this page: http://www.idris.fr/jean-zay/cpu/jean-zay-cpu-comp_options.html

- Optimization options:
  - `-O0`, `-O1`, `-O2`, `-O3`, `-fast` : optimization level. `-O0` is no optimization, use it for debug. `-O3` and `-fast` are more agressive optimization options that can lead to problems in reproducibility of your run (especially it is better to avoid -fast).
  - `-xCORE-AVX2` : vectorization option, very agressive optimization => non-reproducibility of CROCO
  - `-fno-alias`, `-no-fma`, `-ip` : other optimization options, commonly used
  - `-ftz`: set to 0 denormal very small numbers. It is set by default with `-O1`, `-O2`, `-O3` (can be a problem in calculation precision)
- Debug options: `-O0 -g -debug -fpe-all=0 -no-ftz -traceback -check all -fbacktrace -fbounds-check -finit-real=nan -finit-integer=8888`
- Precision and writing options:
  - `-fp-model precise`: important to have good precision and reproducibility of your calculations
  - `-assume byterecl`: way of writing: byte instead of bit
  - `-convert big_endian`: way of writing binaries (important for avoiding huge negative numbers)
  - `-i4`, -r8``: way of writing integers and reals (important also for reproducibility between different clusters)
  - `-72`: specifies that the statement field of each fixed-form source line ends at column 72.
  - `-mcmodel=medium -shared-intel` : do not limit memory to 2Go for data (useful for writing large output files)

10.5 Tips in case of errors during compilation

In case of strange errors during compilation (*e.g.* "catastrophic error: could not find ..."), try one of these solutions:

- check your home space is not full ;-)
- check your paths to compilers and libraries (especially Netcdf library)
- check that you have the good permissions, and check that your executable files (configure, make...) do are executable
- check that your shell scripts headers are correct or add them if necessary (*e.g.* for bash: `#!/bin/bash`)
- try to exit/log out the machine, log in back, clean and restart compilation

Errors and tips related to netcdf library:

- with netcdf 4.3.3.1: need to add the following compilation flag for all models: `-mt_mpi`

  > The error associated to a missing `-mt_mpi` flag is of this type: " /opt/intel//impi/4.1.1.036/intel64/lib/libmpi_mt.so.4: could not read symbols: Bad value "

- with netcdf 4.1.3: do NOT add `-mt_mpi` flag

- with netcdf4, need to place hdf5 library path in your environment:

```
export LD_LIBRARY_PATH=YOUR_HDF5_DIR/lib:$LD_LIBRARY_PATH
```

- with netcdf 4, if you use the library splitted in 2: C part and Fortran part, you need to place links to C library before links to Fortran library and need to put both path in this same order in your LD_LIBRARY_PATH

In case of 'segmentation fault' error:

- try to allocate more memory with "unlimited -s unlimited"
- try to launch the compilation as a job (batch) with more allocated memory

CHAPTER

ELEVEN

RUNNING THE MODEL

To run the model, you need to have completed pre-processing (for realistic cases) and compilation phases. In your working directory you need to have:

- For an idealized simulation (*e.g.* test cases):

```
croco # model executable
croco.in # namelist file (available for each test case croco source␣
↪directory: TEST_CASES)
```

- For a realistic simulation:

```
croco # model executable
croco.in # namelist file (available in croco source directory: OCEAN)

# in CROCO_FILES:
croco_grd.nc # grid file
croco_bdy.nc or croco_clm.nc # lateral boundary condition file
croco_frc.nc or croco_blk.nc # surface forcing file
croco_ini.nc # initial condition file
```

11.1 Edit croco.in

You first need to set all time, I/O, and different parameters in the CROCO namelist file: `croco.in`.

CROCO namelist file `croco.in` is set by default for the BENGUELA_LR case. So you should have nothing to change.

The detail of all `croco.in` sections can be found here: croco.in

However you can check some settings:

- Time stepping:

```
time_stepping: NTIMES   dt[sec]  NDTFAST  NINFO
                720      3600      60      1
```

`NTIMES`: number of time steps `dt[sec]`: baroclinic time step `NDTFAST`: number of barotropic time steps in one baroclinic time step)

Note: Your time steps should be set according to the stability constraints:

– **Barotropic mode**

$$\frac{\Delta t}{\Delta x}\sqrt{gH} \leq 0.89$$

Note that considering an Arakawa C-grid divides the theoretical stability limit by a factor of 2.

So for instance for a maximum depth of 5000 m and a resolution of 30 km:

$$\Delta t \leq \frac{0.89 \Delta x}{2.\sqrt{gH}}$$
$$\Delta t \leq 60s$$

– **3D advection**

With 60 barotropic time steps in one baroclinic time step, this results in a baroclinic time step of:

$$\Delta t \leq 3600s$$

You can check that this time step does not violate your CFL condition for your advection scheme. Typical CFL values for with Croco time-stepping algorithm are

Advection scheme	Max Courant number
C2	1.587
UP3	0.871
SPLINES	0.916
C4	1.15
UP5	0.89
C6	1.00

In the present BENGUELA_LR case, we use UP3:

$$\frac{\Delta t}{\Delta x}.V_{max} \leq 0.871$$
$$V_{max} \leq 0.871 \frac{30000}{3600}$$
$$V_{max} \leq 7.25 m/s$$

which is a very large allowed maximum horizontal velocity.

- Vertical coordinate parameters:

> **Warning:** These parameters should be set accordingly to pre-processing.

```
S-coord: THETA_S,   THETA_B,    Hc (m)
         7.0d0      2.0d0      200.0d0
```

- By default no reference time is used, and time is referred to the beginning of the simulation only using `NTIMES`. If you want to define the start and stop of the model by dates, you first need to edit *cppdefs.h*, define this key, and recompile the model:

```
#define USE_CALENDAR
```

Then edit *croco.in* input file:

```
run_start_date:
01/01/2005 00:00:00
run_end_date:
01/03/2005 00:00:00
output_time_steps: DT_HIS(H), DT_AVG(H), DT_RST(H)
                     1.0        6        48
```

> **Warning:** this replaces **NTIMES** definition which is implicitly calculated.

As we are running a climatological simulation, this is not very relevant (as the model is cycled on a idealized 360-days period). This is more useful for interannual simulations.

- Check the paths to your input files (they should be properly set by default):

```
grid:  filename
    CROCO_FILES/croco_grd.nc
forcing: filename
    CROCO_FILES/croco_frc.nc
bulk_forcing: filename
    CROCO_FILES/croco_blk.nc
climatology: filename
    CROCO_FILES/croco_clm.nc
boundary: filename
    CROCO_FILES/croco_bry.nc
```

- Indicate if you are starting from an initial or restart file and its path:

```
initial: NRREC  filename
          1
    CROCO_FILES/croco_ini.nc
```

`NRREC`: Switch to indicate start or re-start from a previous solution. `NRREC` is the time index of the initial or restart NetCDF file assigned for initialization.

  - If `NRREC=1` you are starting from an initial file.
  - If `NRREC=X with X a positive number`, you are starting from the Xth time record in the restart file.
  - If `NRREC` is negative (say `NRREC=-1`), the model will start from the most recent time record. That is, the initialization record is assigned internally.

- Indicate the frequency of restart files, and their paths:

```
restart:          NRST, NRPFRST / filename
                   720    -1
    CROCO_FILES/croco_rst.nc
```

`NRST`: frequency of restarts in number of time steps

`NRPFRST=-1`: overwrite old restarts at each restart time,

`NRPFRST=0`: store all restarts in one file,

`NRPFRST=X with X a positive number`: store X restarts in one file

- Indicate if you want history (*e.g.* instantaneous) and/or averaged output files, their output frequency, and path:

```
history: LDEFHIS, NWRT, NRPFHIS / filename
            T      72     0
    CROCO_FILES/croco_his.nc
averages: NTSAVG, NAVG, NRPFAVG / filename
            1      72     0
    CROCO_FILES/croco_avg.nc
```

`LDEFHIS`: T or F: do you want history files

`NWRT`: frequency of output in number of time steps

`NRPFHIS` is the way outputs will be stored:

  - NRPFHIS=-1: overwrite old history outputs at each output time,
  - NRPFHIS=0: store all history outputs in one file,
  - NRPFHIS=X with X a positive number: store X history outputs in one file

  NTSAVG: starting time step for the accumulation of output time-averaged data

  NAVG: frequency of averaged outputs in number of time steps

  NRPFAVG: same as for history files

- Choose which variables to output (T or F flag):

```
primary_history_fields: zeta UBAR VBAR  U  V   wrtT(1:NT)
                          T    T   T   T  T    30*T
auxiliary_history_fields:   rho Omega  W  Akv  Akt  Aks Visc3d Diff3d␣
↪HBL HBBL Bostr Wstr Ustr Vstr Shfl Swfl rsw rlw lat sen  HEL
                            F   F     T   F    T    F    F     F      ␣
↪ T   T    T     T    T    T    T    T   10*T
gls_history_fields:   TKE  GLS  Lscale
                       T    T    T

primary_averages: zeta UBAR VBAR  U  V   wrtT(1:NT)
                   T    T    T    T  T   30*T
auxiliary_averages: rho Omega  W  Akv  Akt  Aks  Visc3d Diff3d HBL␣
↪HBBL Bostr Wstr Ustr Vstr Shfl Swfl rsw rlw lat sen HEL
                     F   T     T   F    T    F     F     F       T   T ␣
↪  T     T   T    T     T    T   10*T
gls_averages:   TKE  GLS  Lscale
                 T    T    T
```

Other parameters in croco.in file will be explored in the next tutorials.

11.2 Run the model

- To run the BENGUELA_LR simulation in serial mode (1 CPU), just do:

```
./croco croco.in
```

  Where croco is your executable compiled with all your chosen options and parameterizations and croco.in is your namelist file for croco.

- To run the BENGUELA_LR simulation in parallel (if you have compiled CROCO with #define MPI):
  - By using classical launch command (on individual computers):

```
mpirun -np NPROCS croco croco.in
```

  where NPROCS is the number of CPUs you want to allocate. mpirun -np NPROCS is a typical mpi command, but it may be adjusted to your MPI compiler and machine settings.

  - **OR** by using a batch script (*e.g.* PBS) to launch the model (in clusters), examples are provided:

```
cp ~/croco/croco/SCRIPTS/example_job_croco_mpi.pbs .
```

  Edit example_job_croco_mpi.pbs according to your MPI settings in param.h and to your machine MPI command, and launch the run:

```
qsub example_job_croco_mpi.pbs
```

> **Warning:** NPROCS needs to be consistent to what you indicated in `param.h` during compilation

- If your run is successful you should obtain the following files:

```
croco_rst.nc # restart file
croco_his.nc # instantaneous output file
croco_avg.nc # averaged output file
croco.log    #if you have defined the LOGFILE key in cppdefs.h : #␣
↪define LOGFILE
```

`croco.log` contains the standard output of your run (informations on your settings, input files, evolution of the time stepping). `croco.log` is also useful when your run blows up to search for the error.

You can explore your model outputs (croco_his.nc, croco_avg.nc) using different frameworks (ncview, ferret, etc). In the croco_tools, a matlab interface is offered to explore your data: `croco_gui`, as well as a Python interface `croco_pyvisu`. These are explored in other tutorials.

- Have a quick look at the results:

```
ncview croco_his.nc
```

- Test: some questions:
  - What is the size of the grid (see param.h)?
  - What is the spatial resolution in both horizontal directions?
  - How many vertical levels do you have?
  - How are the vertical levels distributed (look for the cpp key `NEW_SCOORD`)?
  - What are the initial dynamical conditions (see both cppdefs.h and croco.in)?
  - What do the air-sea exchanges look like?

11.3 Tips in case of BLOW UP or ERROR

- Check your time steps
- Eventually increase NDTFAST and/or decrease the baroclinic time step
- Check the location of your boundaries (in particular if your blow up point is located close to them): it should not be placed on a too strong topographic gradient, or coastline particular shape (it is usually better to have a boundary normally crossing the coastline)
- Check the thickness and value of the sponge

CHAPTER

TWELVE

INCREASING THE RESOLUTION: BENGUELA_VHR

Now that you have successfully run the default configuration, you can try running another configuration: BENGUELA_VHR.

1. Create a new configuration directory for `BENGUELA_VHR`
2. As for the previous configuration, edit the paths in `start.m` and `crocotools_param.m` (or copy `start.m` and `crocotools_param.m` from `BENGUELA_LR`)
3. Make the appropriate changes in `crocotools_param.m` to increase the resolution to 1/12°
4. Re-run preprocessing for this new configuration (grid, bulk, forcing, bry, ini)
5. Make the appropriate changes in `cppdefs.h` (define `BENGUELA_VHR`, `MPI`, `BULK_FLUX`, `FRC_BRY`, undef `CLIMATOLOGY`), and `param.h` for running BENGUELA_VHR in parallel on 16 CPUs
6. As for the previous configuration, edit the paths in `jobcomp` (or copy `jobcomp` from `BENGUELA_LR`). And re-compile the model
7. Make the appropriate changes in `croco.in`: change the time step!
8. As in the previous configuration, copy the batch job:

   ```
   cp ~/croco/croco/SCRIPTS/example_job_croco_mpi.pbs .
   ```

   Edit it (notably change the number of CPUs used), and run the model:

   ```
   qsub example_job_croco_mpi.pbs
   ```

9. Test questions:

- On how much CPUs could you run the model (max # of CPUs)?

CHAPTER

THIRTEEN

RUNNING WITH INTERANNUAL FORCING

13.1 Run after classical interannual pre-processing

Before running you should prepare your interannual inputs files following the Interannual Preprocessing tutorial.

To run a plurimonth simulation, we provide the following scripts in `~/croco/croco/SCRIPTS/Plurimonths_scripts`:

- `run_croco.bash`: Plurimonth run with climatological forcing
- `run_croco_inter.bash`: Plurimonth run with interannual forcing

These scripts:

- get the grid, the forcing, the initial and the boundary files
- run the model for 1 month
- store the output files in a specific form: *e.g.* `croco_avg_Y????M?.nc`
- replace the initial file by the restart file (`croco_rst.nc`) which has been generated at the end of the month
- re-launch the model for next month

A dedicated namelist input file is also requested and provided `~/croco/croco/OCEAN/croco_inter.in`

All these files are already copied to your configuration directory if you have used `create_config.bash`. Otherwise, copy them from the source directory to your configuration directory.

1. **Edit `run_croco_inter.bash`:** Paths should already be correct.

   Number of MPI CPUs and command for running:

   ```
   # number of processors for MPI run
   NBPROCS=4

   # command for running the mode : ./ for sequential job, mpirun -np NBPROCS␣
   ↪for mpi run
   RUNCMD="mpirun -np $NBPROCS"
   ```

   Type of forcings:

   ```
   # Define which type of inputs are used
   #
   BULK_FILES=1
   FORCING_FILES=0
   CLIMATOLOGY_FILES=0
   BOUNDARY_FILES=1
   RUNOFF_FILES=0
   ```

   Names of forcings:

```
# Atmospheric surface forcing dataset used for the bulk formula (NCEP)
#
ATMOS_BULK=CFSR
#
# Atmospheric surface forcing dataset used for the wind stress (NCEP,␣
↪QSCAT)
#
ATMOS_FRC=CFSR
#
# Oceanic boundary and initial dataset (SODA, ECCO,...)
#
OGCM=SODA
```

Time stepping and outputs settings:

```
# Model time step [seconds]
#
DT=3600
#
# Output frequency [days]
#   average
ND_AVG=3
#   history (if = -1 set equal to NUMTIMES)
ND_HIS=-1
#   restart (if = -1 set equal to NUMTIMES)
ND_RST=-1
#
# Number of barotropic time steps within one baroclinic time step [number],
↪ NDTFAST in croco.in
#
NFAST=60
#
```

Dates settings (according to `crocotools_param.m`):

```
NY_START=2005
NY_END=2005
NM_START=1
NM_END=3
```

2. **Launch the simulation** Copy the adequate job script:

```
cp ~/croco/croco/SCRIPTS/example_job_run_croco_inter.pbs .
```

Check the MPI settings and launch the job:

```
qsub example_job_run_croco_inter.pbs
```

3. **Check at your outputs:** You should have:

```
croco_his_Y2000M1.nc
croco_his_Y2000M2.nc
croco_his_Y2000M3.nc
croco_avg_Y2000M1.nc
croco_avg_Y2000M2.nc
croco_avg_Y2000M3.nc
croco_rst.nc
```

Warning: If you have an error while you run did not BLOW UP, maybe it is because you have `define LOGFILE` in your `cppdefs.h`. For using `run_croco_inter.in` it should be

```
undef.
```

13.2 Alternative method: online interpolation of atmospheric bulk forcing

Instead of pre-processing your atmospheric bulk forcing, you can use online interpolation of atmospheric bulk forcing.

To do so:

1. **Your atmospheric files need to be in a format readable by CROCO:** At the moment the following forcing are implemented for online interpolation:
   - CFSR data pre-formatted using the script `Process_CFSR_files_for_CROCO.sh` available in `croco_tools`
   - ERAI data pre-formatted using `reformat_ECMWF.m` (used in `make_ECMWF.m` in the `croco_tools`)
   - AROME data formatted in Meteo France framework

   Warning: we need to make the pre-formatting scripts available somewhere in croco_tools

2. **Edit `cppdefs.h`** In `Surface forcing` section:

```
#  define ONLINE
#  ifdef ONLINE
#   undef  AROME
#   undef  ERA_ECMWF
#  endif
```

   Note: for `ONLINE` interpolation, default is CFSR format. `AROME` and `ERA_ECMWF` are also available by defining the cpp-keys.

3. **Re-compile the model** First copy your old executable to keep it, then re-compile:

```
cp croco croco.bck
./jobcomp > jobcomp.log.online
```

4. **Link or copy the CFSR files to your `DATA` directory**

```
mkdir DATA/CFSR_Benguela_LR/
#ln -s ~/DATA/METEOROLOGICAL_FORCINGS/CFSR/BENGUELA/CROCO_format/*2005*.nc␣
↪DATA/CFSR_Benguela_LR/.
cp ~/DATA/METEOROLOGICAL_FORCINGS/CFSR/BENGUELA/CROCO_format/*2005*.nc␣
↪DATA/CFSR_Benguela_LR/.
```

5. **Check and eventually edit `croco_inter.in` last section**

```
online:    byear  bmonth recordsperday byearend bmonthend / data path
           NYONLINE   NMONLINE      4             2011     3
          ../DATA/CFSR_Benguela_LR/
```

6. **Re-run the model**

```
qsub example_job_run_croco_inter.sh
```

Note: In case of errors while using ONLINE, it is probably associated to time issues: check the time in your CFSR input files, and check your time origin Yorig.

CHAPTER

FOURTEEN

NESTING TUTORIAL

Nesting is performed in the model through the AGRIF library.

To create a nested configuration:

1. First build the parent domain configuration as in previous section
2. Then in matlab, you need to use the nestgui utility

```
nestgui
```

   The nestgui will appear :

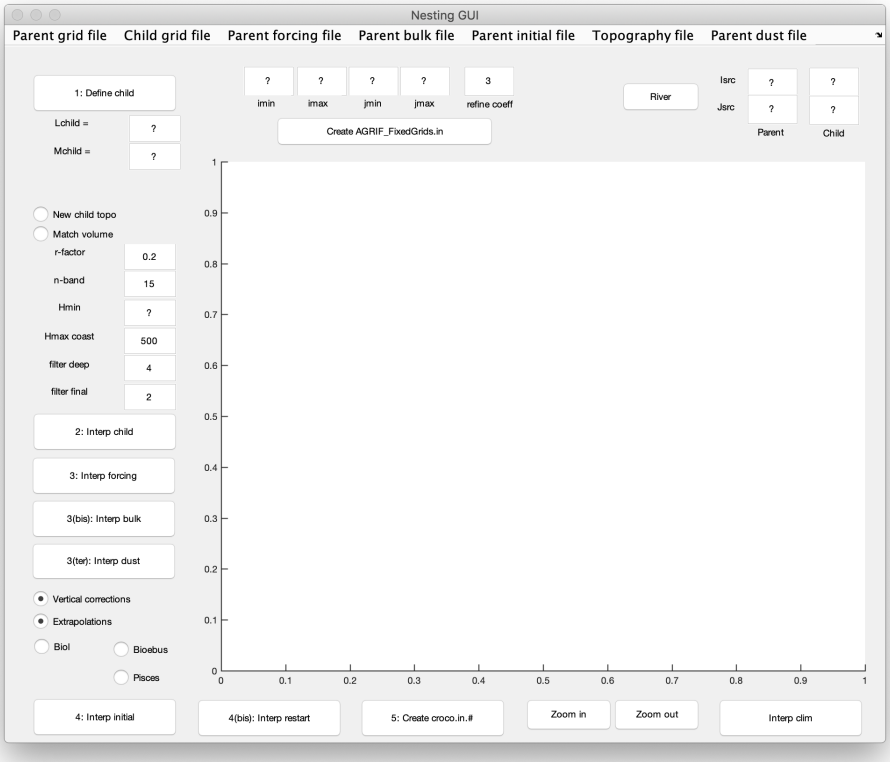


Fig. 1: Entrance window of nestgui

3. First choose the grid file of your parent domain: `CROCO_FILES/croco_grd.nc`.
4. Click `1. Define child` and create the child domain on the main window. The size of the grid child (`Lchild` and `Mchild`) is now visible. This operation can be redone until you are satisfied with the size and the position of the child domain. The child domain can be finely tuned using the `imin`, `imax`, `jmin` and `jmax` boxes.

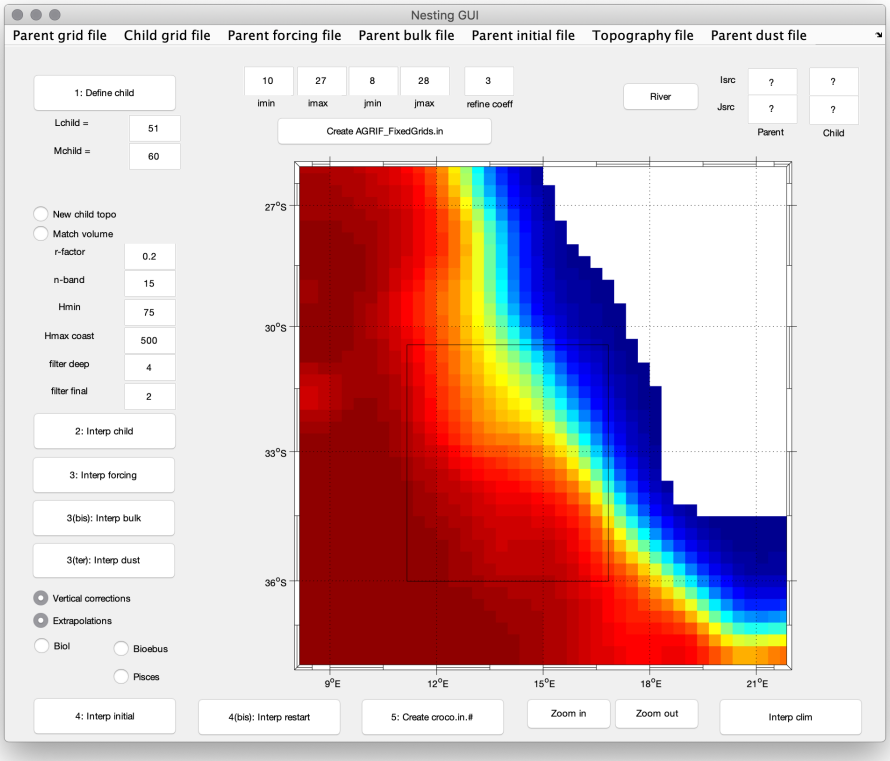


Fig. 2: Main window of nestgui

> **Warning:** Be aware that the mask interpolation from the parent grid to the child grid is not optimal close to corners. Parent/Child boundaries should be placed where the mask is showing a straight coastline. A warning will be given during the interpolation procedure if this is not the case.

5. (If you want to change the topography input file for the child domain, click `new child topo`, choose your new input topo file and edit `n-band` which is the number of grid points on which you will connect the parent and child topography)

   Click `2. Interp child` to create the child grid. It generates the child grid file. Before, you should select if you are using a new topography (`New child topo` button) for the child grid or if you are just interpolating the parent topography on the child grid. In the first case, you should defines what topography file will be used (e.g. ~/Roms_tools/Topo/etopo2.nc or another dataset). You should also define if you want the volume of the child grid to match the volume of the parent close to the parent/child boundaries (`Match volume` button, it should be "on" by default). You should also define the `r-factor` (Beckmann and Haidvogel, 1993) for topography smoothing ("r-factor", 0.25 is safe) and the number of points to connect the child topography to the parent topography (`n-band`, it follows the relation hnew = α. hnew + (1 - α).hparent , where α is going from 0 to 1 in "n-band" points from the parent/child boundaries). You should also select the child minimum depth (`Hmin`, it should be lower or equal to the parent minimum depth), the maximum depth at the coast (`Hmax coast`), the number of selective hanning filter passes for the deep regions (`n filter deep`" and the number of final hanning filter passes (`n filter final`).

6. Click `3. Interp forcing` or `3. Interp bulk` to interpolate the forcing or bulk file on the child grid. It interpolates the parent surface forcing on the child grid. Select the parent forcing file to be interpolated (*e.g.* `Run_BENGUELA_LR/CROCO_FILES/croco_frc.nc`). The child forcing file `croco_frc.nc.1` will be created. The parent surface fluxes are interpolated on the child grid.

You can use `Interp bulk` if you are using a bulk formula. In this case, the parent bulk file (*e.g.* `Run_BENGUELA_LR/CROCO_FILES/croco_blk.nc`) will be interpolated on the child grid.

(If you have changed the topography, Click `Vertical interpolations`)

7. Click `4. Interp initial` or `Interp restart` to create initial or restart file. It interpolates parent initial conditions on the child grid. Select the parent initial file (*e.g.* `Run_BENGUELA_LR/CROCO_FILES/croco_ini.nc`). The child initial file `croco_ini.nc.1` will be created. If the topographies are different between the parent and the child grids, the child initial conditions are vertically re-interpolated. In this case you should check if the options `vertical corrections` and `extrapolations` are selected. It is preferable to always use these options. If there are parent biological fields in the initial files, they can be processed automatically, we have to define the type of biological models: either NChlPZD or N2ChlPZD2, then click on the `Biol` button, either BioEBUS, then click on the `Bioebus`, either PISCES biogeochemical model, then click on the `Pisces` button. The fields needed for the initialization of these biological model will be processed. For information, in the case of NPZD-type (NChlPZD or N2ChlPZD2) model, there are 5 additional fields, in the case of BIOEBUS, there are 8 additional fields and in the case of PISCES biogeochemical model, there is 8 more fields.
8. Click `5. Create croco.in` to create croco.in file for child domain
9. Click `Create AGRIF_FixedGrids.in` to create input file for AGRIF

> **Note:** `Interp clim` button can be used to create a climatology file (*i.e.* boundary conditions) for the child to domain, to test the child domain alone or to compare 1-way online nested run and offline nested run.

10. This will create:

```
CROCO_FILES/croco_grd.nc.1
CROCO_FILES/croco_frc.nc.1   (or croco_blk.nc.1)
CROCO_FILES/croco_ini.nc.1
croco.in.1
AGRIF_FixedGrids.in
```

11. Once the input files had been build, you need to compile the model in nesting mode. You need to `define AGRIF` in `cppdefs.h` and re-compile.
12. You will then be able to launch `croco` as usual. It will run as an individual binary, with an internal loop on the number of grids. The child grid will use the `*.1` files, this suffix will also be added to the output file of the nest. You can also define more than one child grid.

```
qsub job_croco_mpi.pbs
```

CHAPTER

FIFTEEN

ADDING RIVERS

If you want to include rivers in your simulation domain, there are several variables to define as:

- the number of rivers: Nsrc
- the position of the rivers on the model grid: **Isrc** and **Jsrc**
- the zonal or meridional axis of the river flow: **Dsrc**
- **if flow (and concentration) is constant**, the flow rate of the river (in m3/s): **Qbar** (positive or negative)
- **if flow (and concentration) is variable, and read from a netCDF file**, the direction of the flow **qbardir** :
  - 1 for west-east / south-north
  - -1 for east-west / north-south
- the type of tracer advected by the river: **Lsrc**
- the value/concentration: **Tsrc**

15.1 Constant flow and concentration

For this you need to define the cpp-keys in cppdefs.h

```
#define PSOURCE
```

And re-compile.

Then in the croco.in file

```
psource:   Nsrc  Isrc  Jsrc  Dsrc  Qbar [m3/s]    Lsrc        Tsrc
         2
                  3    54     1    200.       T T      20. 15.
                  3    40     0    200.       T T      20. 15.
```

where Nsrc=2 is the number of rivers processed, then each line describes a river. Let's describe the parameter for river #1:

- Isrc=3, Jsrc=54 are the i, j indices where the river is positioned
- Dsrc=1 indicates the orientation (here meridional => along V direction)
- 200 is the runoff flow value in m3/s, oriented to the east
- T T are true/false indications for reading or not the following variables (here temperature and salinity)
- 20 and 15 are respectively the temperature and salinity of the river. You can edit these parameters.

Warning: The sources points must be placed on U or V points on the C-grid and not on rho-points

You can then run the model:

```
qsub job_croco_mpi.pbs
```

15.2 Variable flow read in a netCDF file and constant concentration

Instead of using a constant flow, you can use variable flow. For that you need read it from a netcdf file. First define the dedicated cpp-key in cppdefs.h

```
#define PSOURCE_NCFILE
```

And re-compile the model.

Then you also need to prepare the netcdf river runoff input file.

For that, you can use in CROCO_TOOLS `make_runoff` (Rivers/make_runoff.m) which detect the main rivers located in your domain (from **RUNOFF_DAI** runoff climatology).

Note: RUNOFF_DAI is a global monthly runoff climatology containing the 925 first rivers over the world, from *Dai and Trenberth, 2000*

After asking you some specifications for each detected river in your domain, for the selected rivers:

- It will compute the right location on the croco_grid regarding the direction and orientation you defined
- It will create the river forcing netCDF file croco_runoff.nc containing the various river flow time series.

To do so, in CROCO_TOOLS, edit `make_runoff.m` and define the following flags:

```
%% Choose the monthly runoff forcing time and cycle in days

clim_run=1

%     - times and cycles for runoff conditions:
%           - clim_run = 1 % climato forcing experiments with climato calendar
%                     qbar_time=[15:30:365];
%                     qbar_cycle=360;
%
%           - clim_run = 0 % interanual forcing experiments with real calendar
%                     qbar_time=[15.2188:30.4375:350.0313];
%                     qbar_cycle=365.25;
```

```
psource_ncfile_ts=0;

%        - psource_ncfile_ts = 0 => Constant analytical runoff tracers␣
↪concentration no processing
%                            It reads analytical values in croco.in
%                            or use default value defined in
%                            analytical.F
```

For the BENGUELA test case, you will have 2 rivers detected, Orange and Doring. We recommend to define them as zonal (0) and oriented from east to west (-1). It will give you the lines to enter in the croco.in file in the psource_ncfile section.

```
psource_ncfile:   Nsrc  Isrc  Jsrc  Dsrc qbardir  Lsrc  Tsrc   runoff file name
                         CROCO_FILES/croco_runoff.nc
                2
                      25  34  0  -1   30*T   20 15
                      31  19  0  -1   30*T   20 15
```

where Nsrc=2 is the number of rivers, then each line describe a river. Let's describe the parameter for the river #1

- Isr=25, Jsrc=34 are the i, j indices where the river is positioned
- Dsrc=0 indicates the orientation (here zonal)
- qsbardir= -1 indicates the direction (here towards the west)
- Lstrc=30*T are true/false flags for reading or not the following variables (here temperature and salinity)
- Tsrc=20 15 are respectively the temperature and salinity of the river.

You can edit these parameters.

Temperature and salinity can also be variable and read from a netCDF file, it is described in the next section.

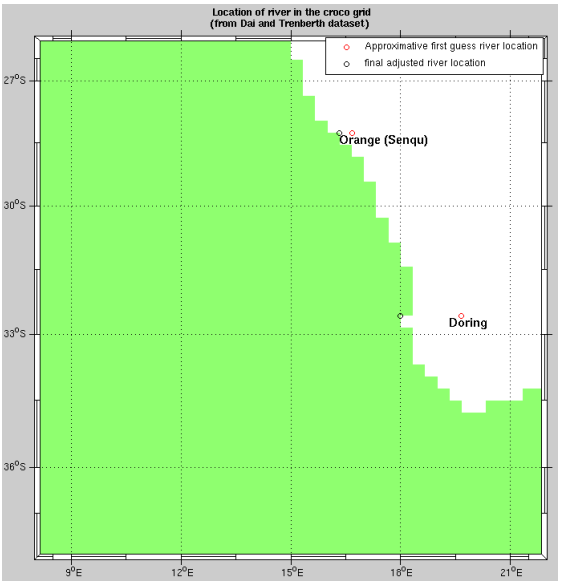


Fig. 1: First and final guess rivers positions

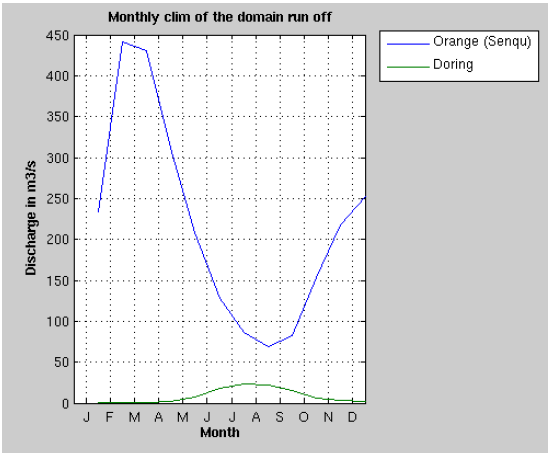


Fig. 2: Rivers flow seasonal cycle

15.3 Variable flow and variable concentration from a netCDF file

To run CROCO with a variable concentration of river tracers, you need to define the following cpp-key in cppdefs.h

```
#define PSOURCE_NCFILE_TS
```

You also need to prepare your netcdf input file. Using the CROCO_TOOLS: edit `make_runoff.m` and change the following flags:

```
psource_ncfile_ts=1;

if psource_ncfile_ts
    psource_ncfile_ts_auto=1 ;
    psource_ncfile_ts_manual=0;
end

%        - pource_ncfile_ts = 1  => Variable runoff tracers
%                                   concentration  processing is activated.
%
%        In this case, either choose:
%           - psource_ts_auto : auto definition
%                               using the nearest point in the climatology
%                               file croco_clm.nc to fill the tracer
%                               concentration time serie in croco_runoff.nc
%
%           - psource_ts_manual : manually definition the
%                                 variable tracer concentration to fill
%                                 the tracer concentration time serie in
%                                 croco_runoff.nc
```

After asking you some specifications of each detected river in your domain, for the selected rivers, in addition to river flow as in previous section, it will also put the tracers concentration (temp,salt, no3, et ...) time series into the river forcing netCDF file croco_runoff.nc

```
psource_ncfile:   Nsrc  Isrc  Jsrc  Dsrc qbardir  Lsrc  Tsrc   runoff file name
                          CROCO_FILES/croco_runoff.nc
                2
                       25  34  0  -1   30*T   16.0387 25.0368
                       30  19  0  -1   30*T   16.1390 25.1136
```

You also can edit these parameters.

Warning: The Tsrc value reported in croco.in are the annual-mean tracer values, the are just for information. The real tracer concentration (Tsrc) are read in the runoff netCDF file created.

15.4 Using a nest

The above procedure can be applied to a nested grid. For this, edit `make_runoff` and change the `gridlevel` variable to the adhoc grid level.

```
%Choose the grid level into which you ant to set up the runoffs
gridlevel=1
if ( gridlevel == 0 )
  % -> Parent / zoom #0
  grdname  = [CROCO_files_dir,'croco_grd.nc'];
  rivname =  [CROCO_files_dir,'croco_runoff.nc']
  clmname = [CROCO_files_dir,'croco_clm.nc'];    % <- climato file for runoff
```

(continues on next page)

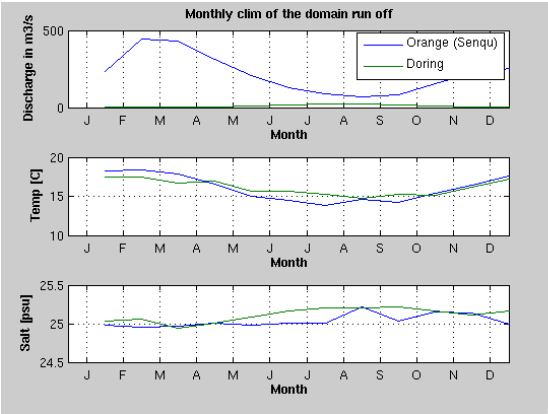


Fig. 3: Rivers tracer concentration seasonal cycle

(continued from previous page)

```
else
  %  -> Child / zoom #XX
  grdname  = [CROCO_files_dir,'croco_grd.nc.',num2str(gridlevel)];
  rivname =  [CROCO_files_dir,'croco_runoff.nc.',num2str(gridlevel)];
  clmname = [CROCO_files_dir,'croco_clm.nc.',num2str(gridlevel)]; % <- climato␣
↪file for runoff
end
```

and run `make_runoff` again to generate

```
croco_runoff.nc.1
```

Note: The runoff has a default vertical profile defined in CROCO as an exponential vertical distribution of velocity. It is in `analytical.F`, subroutine `ana_psource` if you need to change it.

CHAPTER

SIXTEEN

ADDING TIDES

Using the method described by Flather (1976), CROCO is able to propagate the different tidal constituents from its lateral boundaries.

To do so, you will need to add the tidal components to the forcing file, and define the following cpp keys `TIDES`, `SSH_TIDES` and `UV_TIDES` and recompile the model using jobcomp. To work correctly, the model should use the characteristic method open boundary radiation scheme (cpp key `OBC_M2CHARACT` defined).

Warning: To get a clean signal you need to provide harmonic components from both tide elevation and tide velocity. In case you don't have velocity harmonics (not defined `UV_TIDES`) a set of reduced equation is available to compute velocity from SSH (`OBC_REDUCED_PHYSICS`)

16.1 Pre-processing (Matlab)

1. **Edit `crocotools_param.m` section 5**

```
%%%%%%%%%%%%%%%%%%%%%%
%
% 5-Parameters for tidal forcing
%
%%%%%%%%%%%%%%%%%%%%%%
%
% TPXO file name (TPXO7)
%
tidename=[CROCOTOOLS_dir,'TPXO7/TPXO7.nc'];
%
% Number of tides component to process
%
Ntides=10;
%
% Chose order from the rank in the TPXO file :
% "M2 S2 N2 K2 K1 O1 P1 Q1 Mf Mm"
% " 1 2 3 4 5 6 7 8 9 10"
%
tidalrank=[1 2 3 4 5 6 7 8 9 10];
%
% Compare with tidegauge observations
%
lon0=18.37;
lat0=-33.91; % Cape Town location
Z0=1; % Mean depth of the tidegauge in Cape Town
```

2. **Launch pre-processing of tides in Matlab:**

```
start
make_tides
```

3. Check your `croco_frc.nc` file

16.2 Compiling

1. **Edit `cppdefs.h` for defining tides:**

```
/* Open Boundary Conditions */
# define TIDES

/* Open Boundary Conditions */
# ifdef TIDES
#  define SSH_TIDES
#  define UV_TIDES
#  define POT_TIDES
#  undef  TIDES_MAS
#  ifndef UV_TIDES
#   define OBC_REDUCED_PHYSICS
#  endif
#  define TIDERAMP
# endif
# define OBC_M2CHARACT
# undef  OBC_M2ORLANSKI
# define OBC_M3ORLANSKI
# define OBC_TORLANSKI
# undef  OBC_M2SPECIFIED
# undef  OBC_M3SPECIFIED
# undef  OBC_TSPECIFIED
```

2. **Check/Edit `param.h`:**

```
#if defined SSH_TIDES || defined UV_TIDES
      integer Ntides              ! Number of tides
                                  ! ====== == =====
# if defined IGW || defined S2DV
      parameter (Ntides=1)
# else
      parameter (Ntides=10)
# endif
#endif
```

> **Warning:** The number of tide components must be coherent with the one defined in `crocotools_param.m`

3. **Re-compile the model:**

```
./jobcomp > jobcomp_tide.log
```

16.3 Running

Run the model:

```
qsub job_croco_mpi.pbs
```

CHAPTER

SEVENTEEN

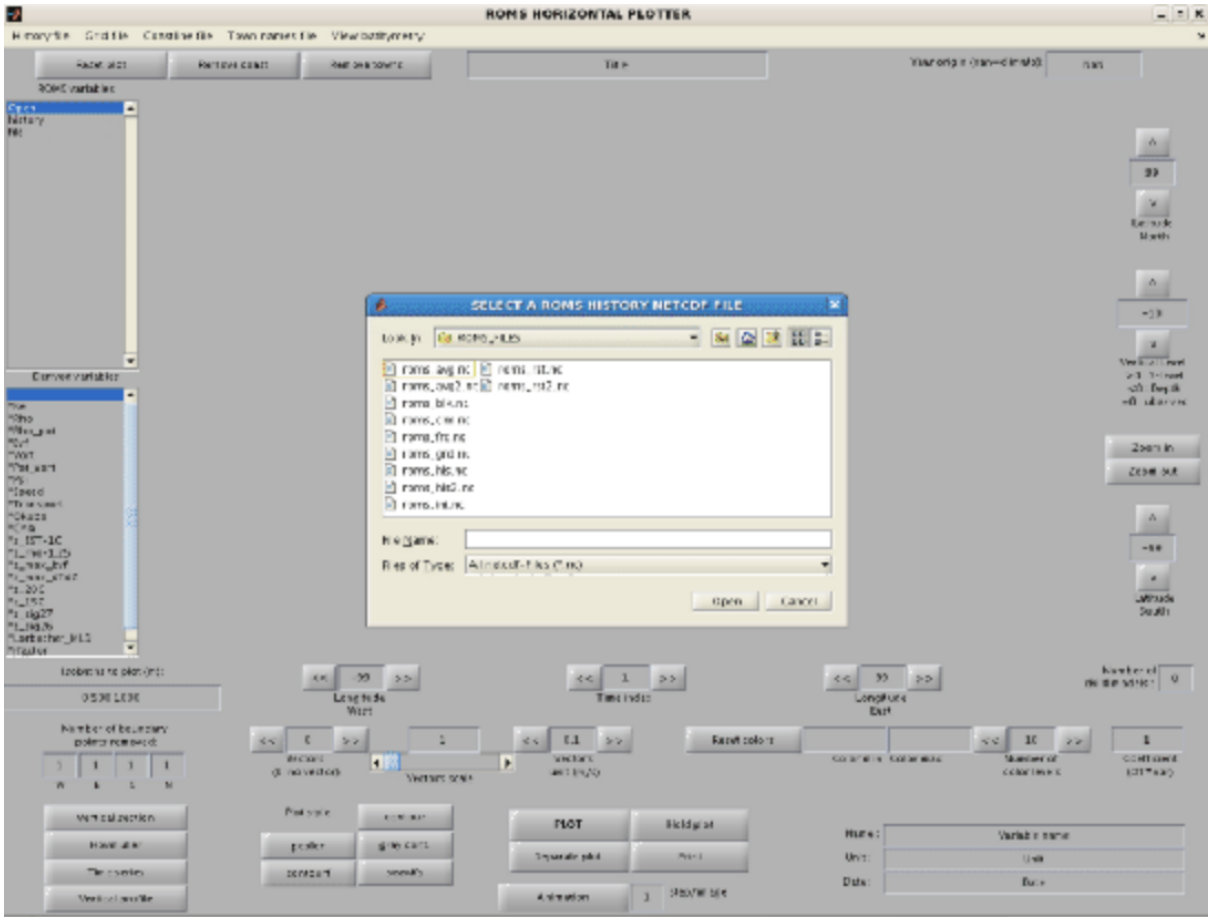
VISUALIZATION (MATLAB)

The `croco_gui` utility has been developped under Matlab software to visualize CROCO outputs.

In Matlab:

```
start
croco_gui
```

A window pops up, asking for a CROCO history NetCDF file (see screen captions below). You should select croco_his.nc (history file) or croco_avg.nc (average file) and click “open”.

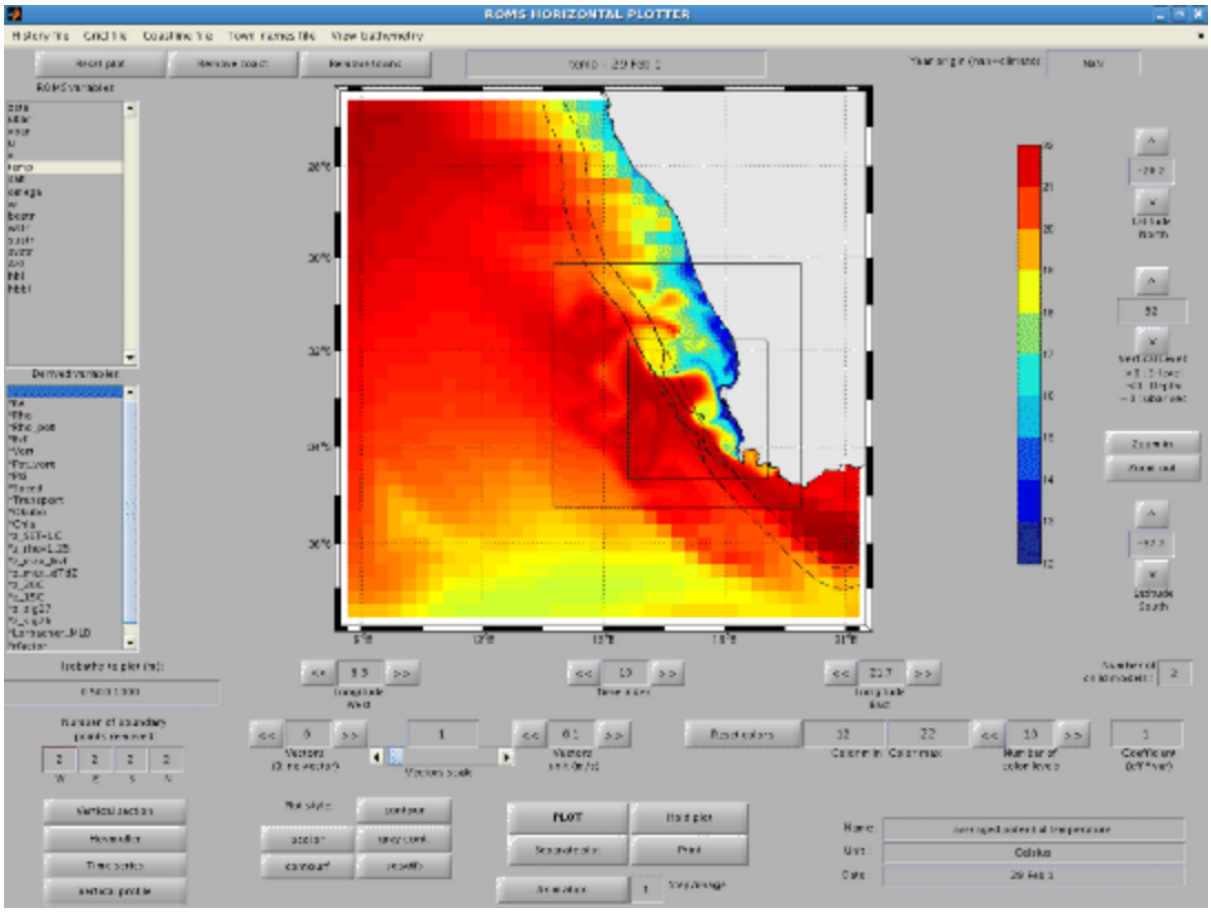


The main window appears, variables can be selected to obtain an image such as Figure below. On the left side, the upper box gives the available CROCO variable names and the lower box presents the variables derived from the CROCO model outputs :

- Ke : Horizontal slice of kinetic energy
- Rho : Horizontal slice of density using the non-linear equation of state for seawater of Jackett and McDougall (1995)

- Pot_Rho : Horizontal slice of the potential density
- Bvf : Horizontal slice of the Brunt-Väisäla frequency
- Vort : Horizontal slice of vorticity
- Pot_vort : Horizontal slice of the vertical component of Ertel's potential vorticity. In our case, **:mat:`\lambda=\rho`**
- Psi : Horizontal slice of stream function. This routine might be costly since it inverses the Laplacian of the vorticity (using a successive over relaxation solver)
- Speed : Horizontal slice of the ocean currents velocity
- Transport : Horizontal slice of the transport stream function
- Okubo : Horizontal slice of the Okubo-Weiss parameter
- Chla : Compute a chlorophyll-a from Large and Small phytoplankton concentrations
- z_SST_1C : Depth of 1°C below SST
- z_rho_1.25 : Depth of 1.25 kg/m^3 below surface density
- z_max_bvf : Depth of the maximum of the Brunt-Väisäla frequency
- z_max_dtdz : Depth of the maximum vertical temperature gradient
- z_20C : Depth of the 20°C isotherm
- z_15C : Depth of the 15°C isotherm
- z_sig27 : Depth of the 1027 kg/m^3 density layer

It is possible to add arrows for the horizontal currents by increasing the "Current vectors spatial step". It is also possible to obtain vertical sections, time series, vertical profiles and Hovmüller diagrams by clicking on the corresponding targets in croco_gui.



CHAPTER

EIGHTEEN

VISUALIZATION (PYTHON)

Croco_visu is a tool written in python to visualize history files genarated by the Croco model.

18.1 Setup your Miniconda environment

- Download and install miniconda: download Miniconda from the Conda website. See the documentation for installation.
- Put the path in your shell environment file (ex:.cshrc, .bashrc)

```
source path_to_your_miniconda/etc/profile.d/conda.csh
```

- Create your conda croco_pyvisu environment

```
conda update conda
conda create -n croco_pyvisu -c conda-forge python=3.7 wxpython xarray␣
↪matplotlib netcdf4 scipy ffmpeg
conda create --name toto --clone
```

18.2 Croco_visu directory

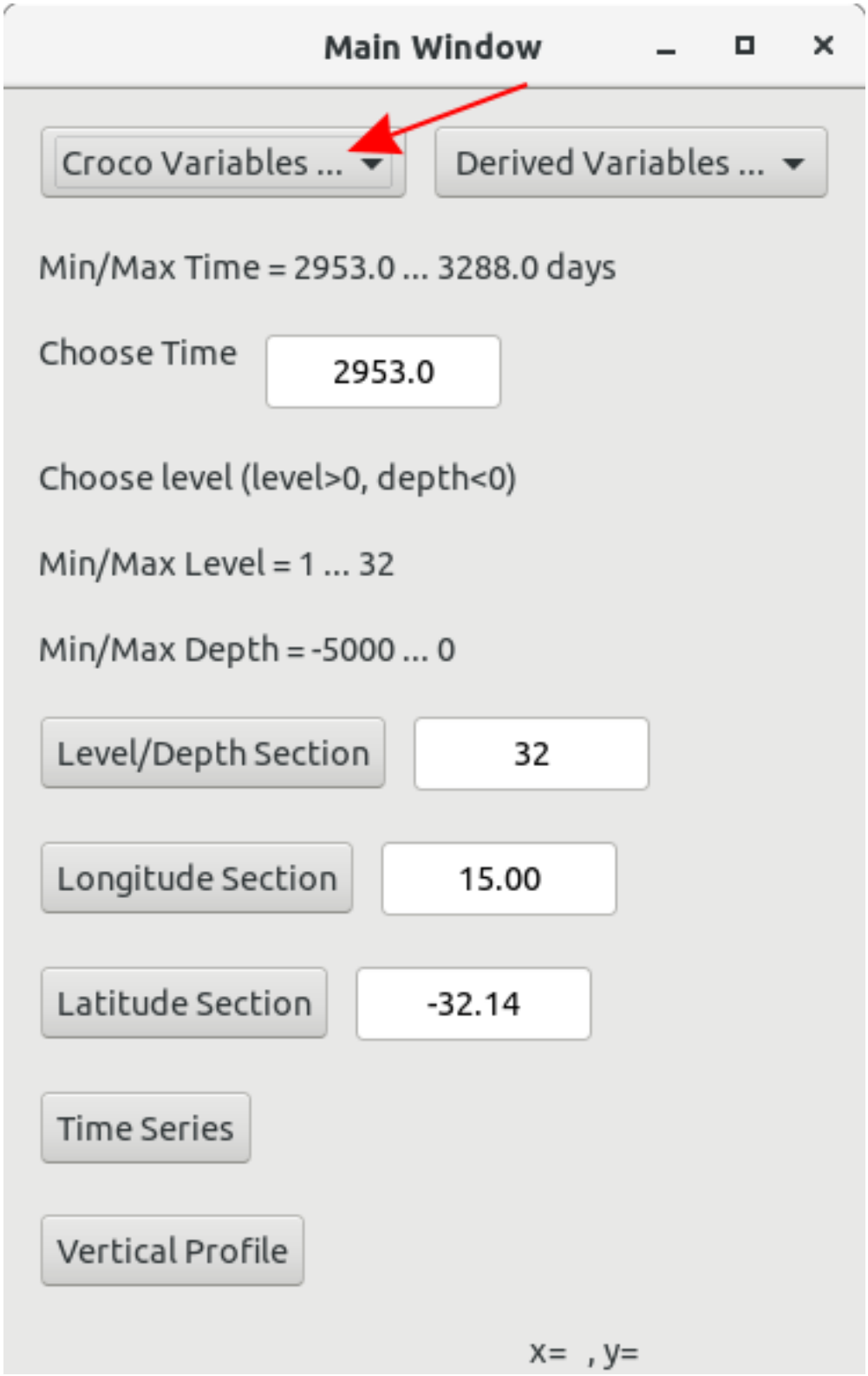
The `croco_pyvisu` directory is provided in the CROCO_TOOLS.

18.3 Launch visualization

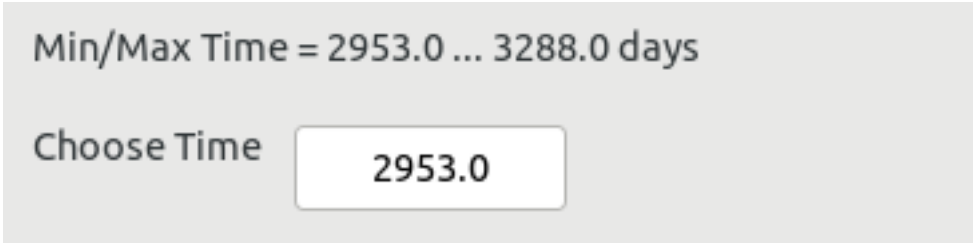
To start croco_visu:

```
conda activate croco_pyvisu
croco_gui_xarray.py
```

The main window is opened.

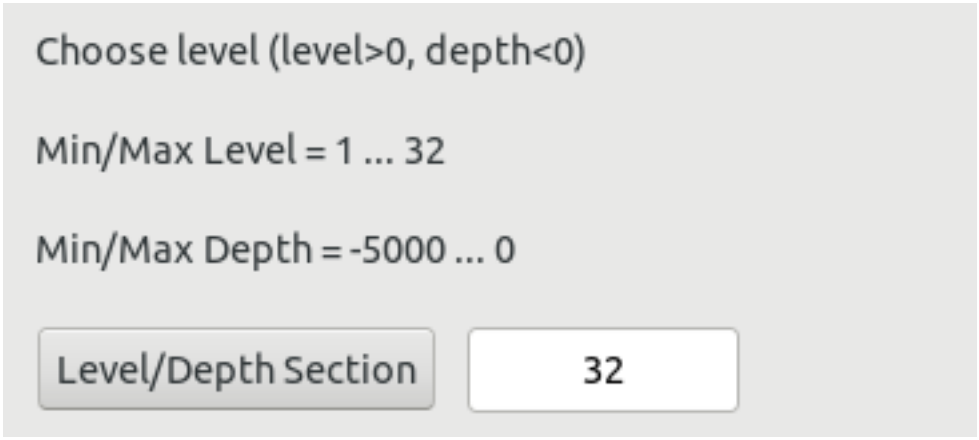


First, you have to choose a variable through the *Croco Variables...* menu, which list all the 2D/3D variables of the history file.



Min/max Time of the file are filled out
You can type a time in the *Choose Time* input box. The nearest time of the history file will be retrieved.

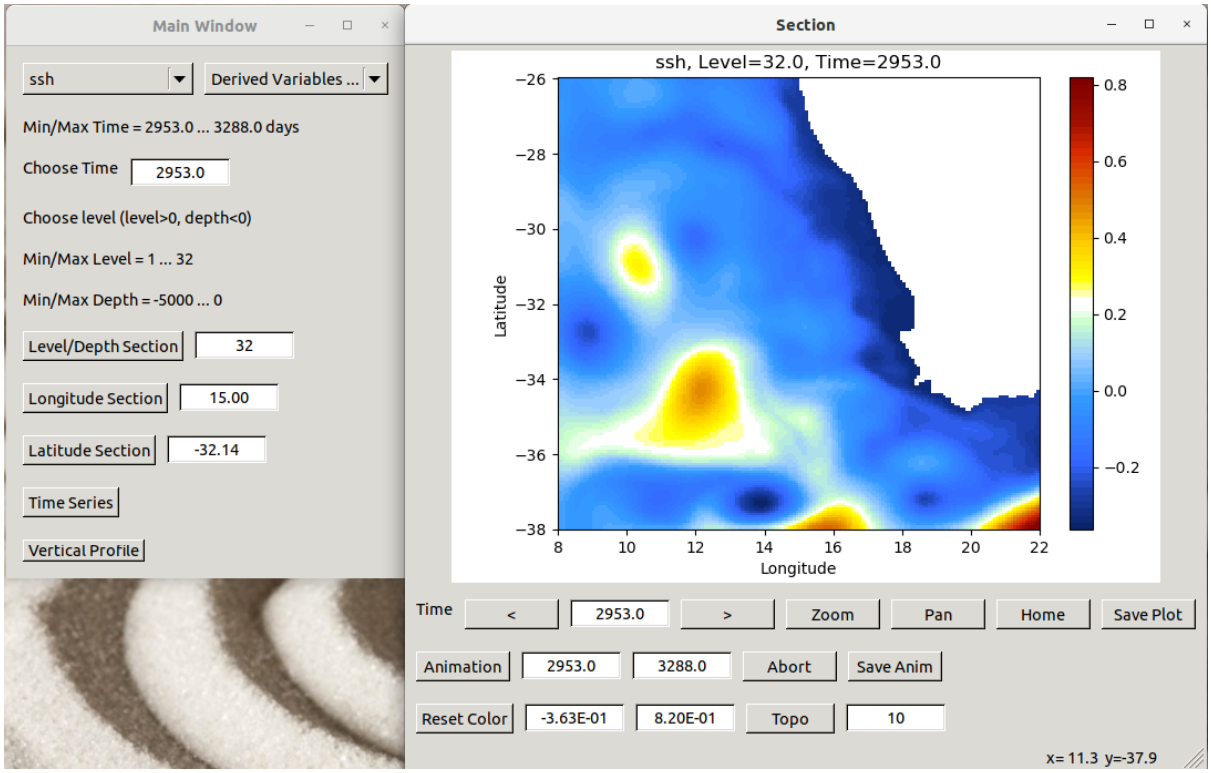
Warning: Each time you type something in an input text box, you must validate the input with the *Enter* key.



Min/max level and min/max depth of the file are indicated
You can type a level or depth in the input level/depth box (>0 means level, <=0 means depth), by default, the value is the highest level

You can type a longitude/latitude in the input longitude/latitude box (default is the mean longitude/latitude)

Now you can click on the *Level/Depth Section* button and a new window appears.



In this new window, you can only plot the current variable at the current level/depth. If you want another variable, or another level/depth, you have to choose first another variable or another level/depth and click again on the Level/Depth Section button. Then you will have another new window.

This new window offers several buttons and information:

- to change the current time, you can type a new time in the text box or use the arrows

- to zoom, translate and save the plot

To zoom, you have to first click the *Zoom* button and after select the region to zoom.
To translate the plot, you have to first click the *Pan* button and then move the plot with the left mouse button.
The *Home* button is to go back to the default view.
The *Save Plot* button will open a new window for you to save the current plot.

- to create animations

You have first to choose the start time and the end time in the two input text boxes. Then you can click on the *Animation* button to start the animation. You can abort the animation with the *Abort* button and if you select the *Save Anim* button, your animation is saved in a sub-directory *Figures_*.

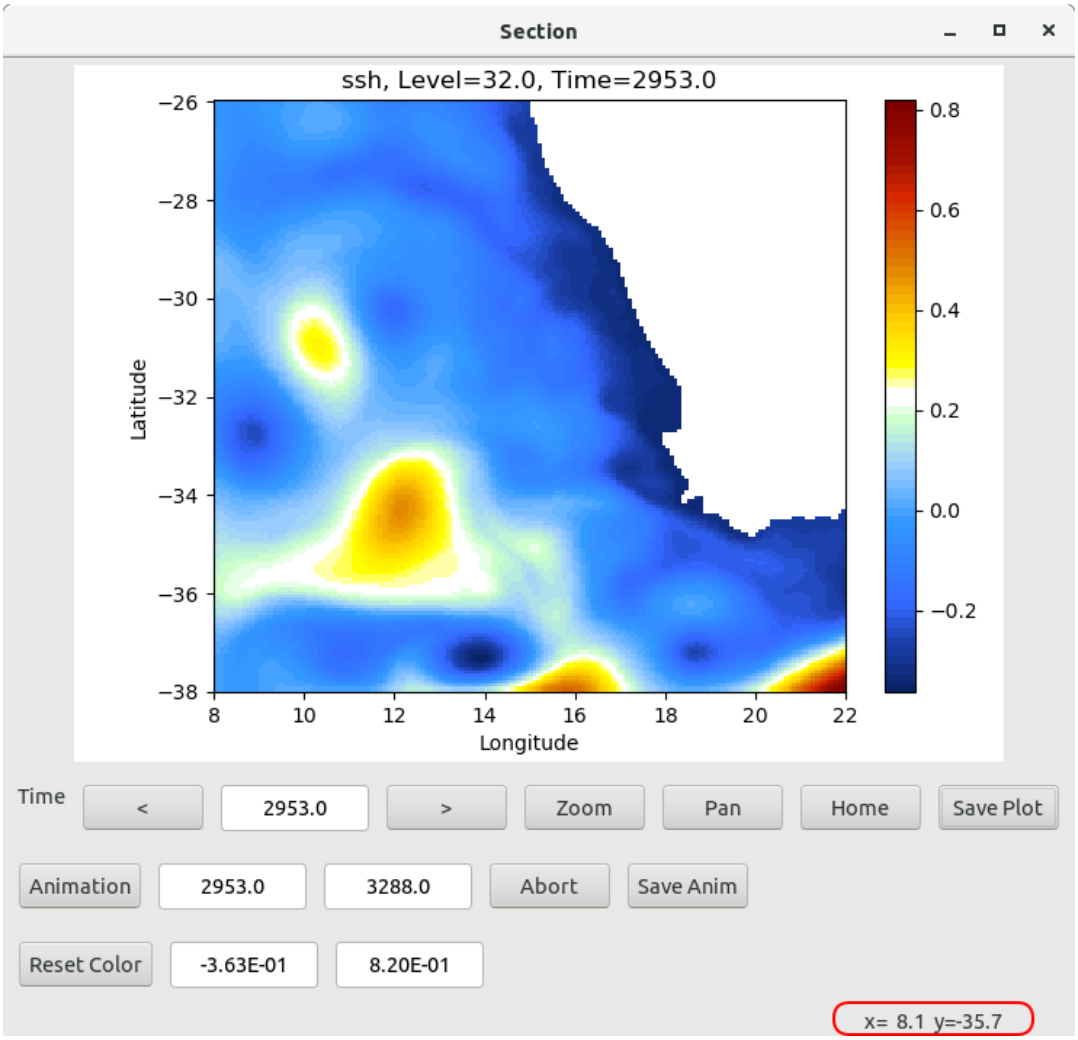
- change the colorbar

You can choose new limits for the colorbar (return to validate the input) or you can go back to the default colorbar with the *Reset Color* button.

- show contours of the topography

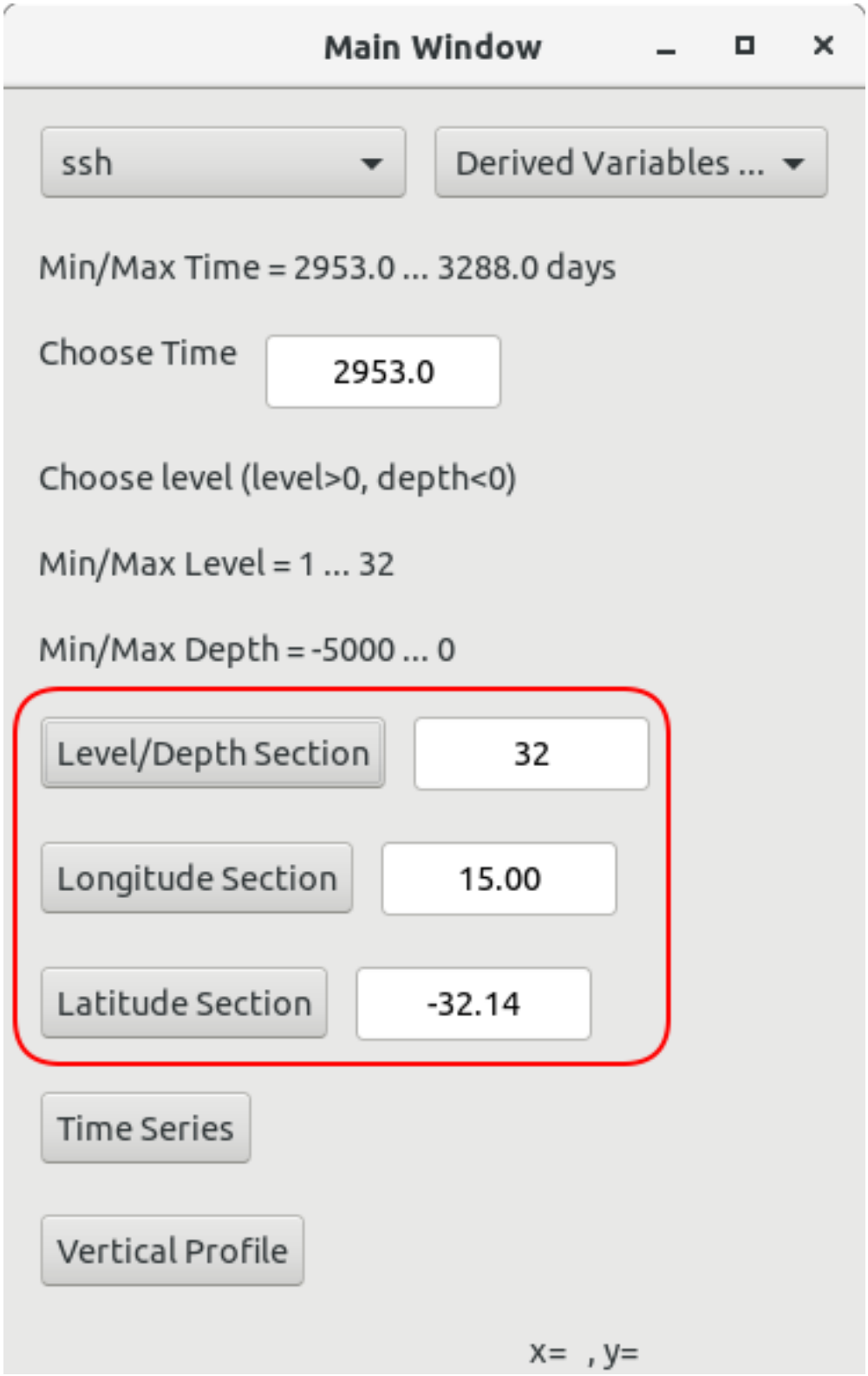
You can show the contours of the topography by clicking on the *Topo* button (on/off) and you can change the number of contours shown in the text input box (return to validate the input, default is 10).

- see the coordinates of the current point



At the right bottom corner of the window, you have the coordinates of the cursor.

On the main window, the two others buttons *Longitude Section* and *Latitude Section* will open the same kind of window than the *Level/Depth Section* but at a given longitude or latitude.

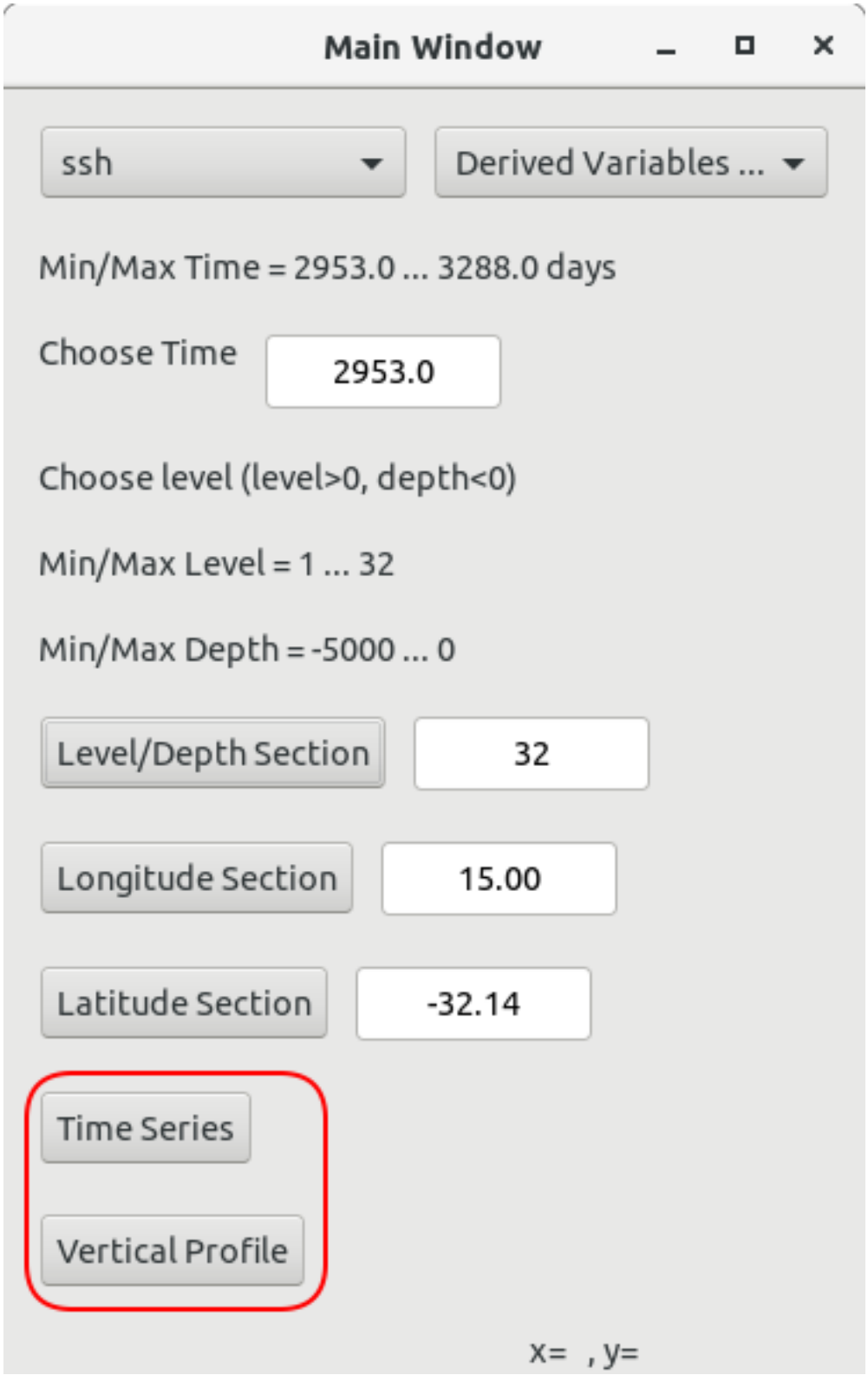


Of course, to have a longitude/latitude section, you must choose a 3D variable.
You have to choose first the longitude/latitude before clicking the buttons. The longitude/latitude can be typed in the text input boxes near the buttons, or you can click on a plot to select a new point. If you click on the plot of a

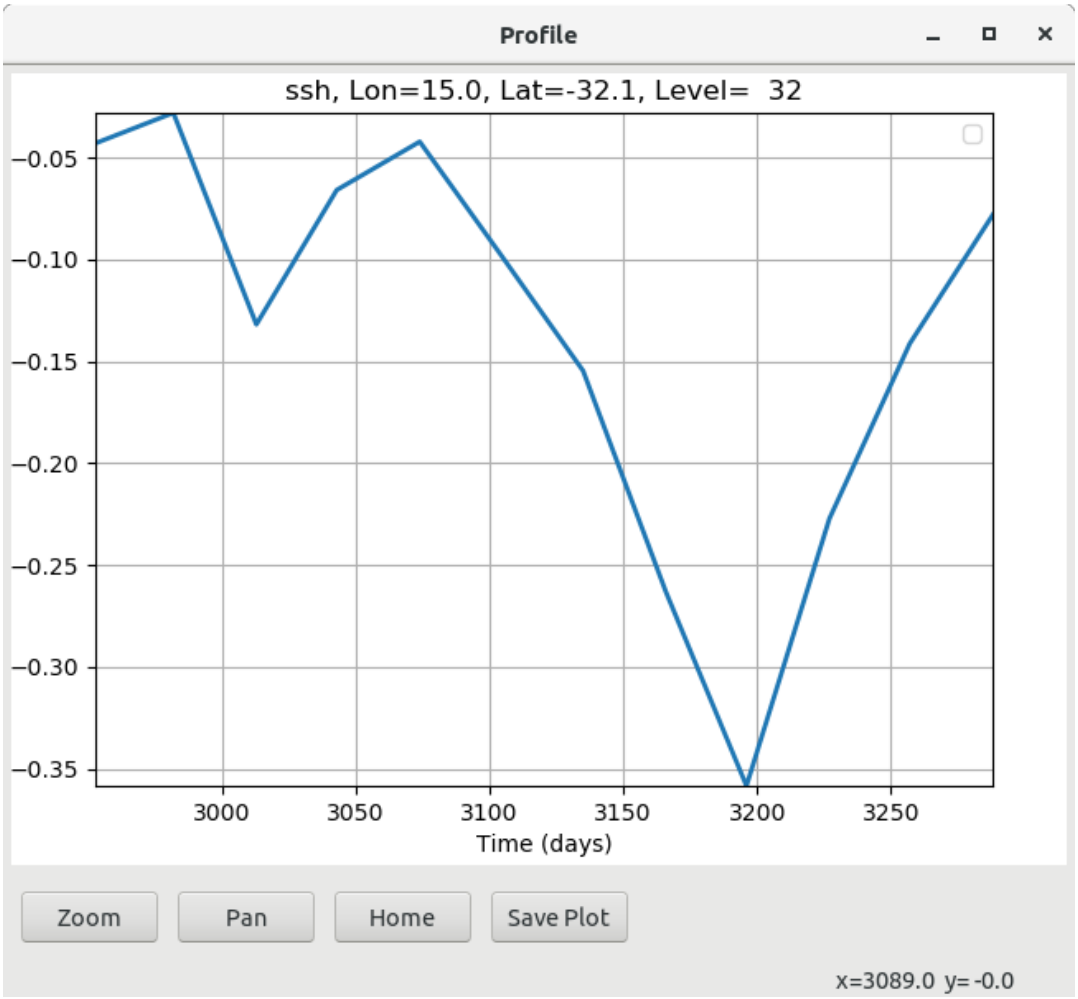
- *Level/Depth Section*, you select new longitude and latitude.

- *Longitude Section*, you select new depth and latitude.
- *Latitude Section*, you select new depth and longitude.

The two last buttons of the main window *Time Series* and *Vertical Profile* create new windows to plot curves.



Both *Time Series* and *Vertical Profile* have the same possibilities.



- *Zoom* a part of the curve: you must first select the *Zoom* button and then select the region to zoom.
- *Pan* the curve: you must first select the *Pan* button and then translate the curve.
- *Home* to go back to the default view
- *Save Plot* : when you click on the *Save Plot* button, a popup window is opened for you to choose the name of the file.

The time series is plotted at the current level/depth, longitude and latitude. The vertical profile is plotted at the current longitude/latitude.
You also have the coordinates of the cursor at the right bottom of the window.

18.4 How to customize for your own history files

The only file you have to change is the file *croco_wrapper.py*. You have two files as examples in the repository:

1. croco_wrapper.py.benguela for the Benguela test case, where history files are created through the classic method.
2. croco_wrapper.py.moz where the history files are created through XIOS.

Choose the right one to start

```
cp croco_wrapper.py.benguela croco_wrapper.py
```

Change `path` and `keymap_files` to load the right files:

```
path = "./"
keymap_files = {
    'coordinate_file': path + "croco_his_Y2008.nc",
    'metric_file': path + "croco_his_Y2008.nc",
    'mask_file': path + "croco_his_Y2008.nc",
    'variable_file': path + "croco_his_Y2008.nc"
}
```

- coordinate_file : containing lon_r, lat_r, time
- metric_file : containing pm, pn, theta_s, theta_b, Vtransform, hc, h, f
- mask_file : containing mask_rho
- variable_file : containing ssh, u, v, w, temp, salt, rho

Change the `keymap_*` dictionnaries to suit your files, you must only change the keys, that is the values on the left before the ":":

```
keymap_dimensions = {
    'xi_rho': 'x_r',
    'eta_rho': 'y_r',
    'xi_u': 'x_u',
    'y_u': 'y_r',
    'x_v': 'x_r',
    'eta_v': 'y_v',
    'x_w': 'x_r',
    'y_w': 'y_r',
    's_rho': 'z_r',
    's_w': 'z_w',
    'time': 't'
}


keymap_coordinates = {
    'lon_rho': 'lon_r',
    'lat_rho': 'lat_r',
    'lon_u': 'lon_u',
    'lat_u': 'lat_u',
    'lon_v': 'lon_v',
    'lat_v': 'lat_v',
    'scrum_time': 'time'
}


keymap_variables = {
    'zeta': 'ssh',
    'u': 'u',
```

(continues on next page)

(continued from previous page)

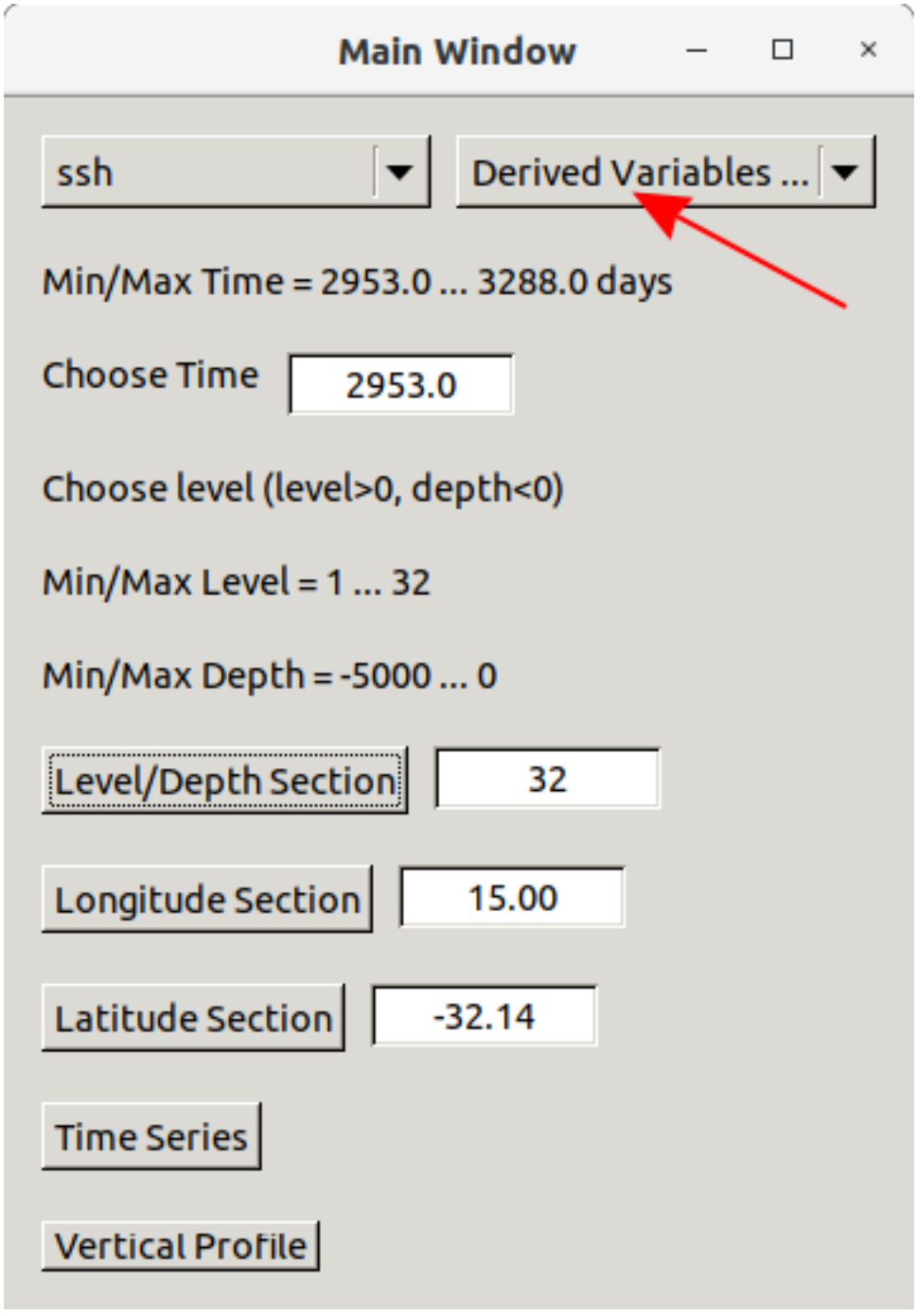
```
    'v': 'v',
    'w': 'w',
    'temp': 'temp',
    'salt': 'salt',
    'rho': 'rho'
}

keymap_metrics = {
    'pm': 'dx_r',
    'pn': 'dy_r',
    'theta_s': 'theta_s',
    'theta_b': 'theta_b',
    'Vtransform': 'scoord',
    'hc': 'hc',
    'h': 'h',
    'f': 'f'
}

keymap_masks = {
    'mask_rho': 'mask_r'
}
```

18.5 How to add new variables

In the main window, you have another menu called *Derived Variables...*, which contains calculated variables, derived from the base fields found in the history file.



Right now, available variables are

- pv_ijk

  Ertel potential vorticity is given by $\frac{curl(u)+f}{rho}$

- zeta_k

  zetak is given by $\frac{\partial v/\partial x - \partial u/\partial y}{f}$

- dtdz

  dtdz in given by $\frac{\partial T}{\partial z}$

- log(Ri)

  Ri is given by $\frac{N}{(\partial u/\partial z)-(\partial v/\partial z)}$ with $N = \sqrt{\frac{-g}{rho0} * \frac{\partial rho}{\partial z}}$

You can add new variables by:

- in the file *CrocoXarray.py*, add a line in the method *list_of_derived*:

```
def list_of_derived(self):
    ''' List of calculated variables implemented '''
    keys = []
    keys.append('pv_ijk')
    keys.append('zeta_k')
    keys.append('dtdz')
    keys.append('log(Ri)')
    return keys
```

- in the file *derived_variables.py*, add two functions *get_newvar* and *calc_newvar* to calculate the new variable
- in the file *croco_gui_xarray.py*, add the calls to the new function *get_newvar* in *updateVariableZ*, *onTimeSeriesBtn* and *onVerticalProfileBtn*

CHAPTER

NINETEEN

NBQ TUTORIAL

CROCO-NBQ kernel solves the compressible and non-hydrostatic Navier-Stokes equations. This kernel can be used to simulate complex nonlinear, nonhydrostatic physics in a realistic but computationally-affordable configuration. Non-hydrostatic effects become important when the horizontal and vertical scales of motion are similar. In oneanic models this typically arises with horizontal scales of the order of 1 km resolved with grid intervals of order 100 m. For motions of larger scale that are resolved with grid intervals of order 1 km, the hydrostatic approximation is well satisfied.

Accurate simulation of nonhydrostatic effects requires to resolve very small horizontal scales. The explicit representation of fine-scale turbulent processes requires a significant number of fundamental numerical choices, such as adapted advective schemes, adaptaed parametrizations, adapted boundary conditions ... In the sections you will find some recommendations about the most adapted numerical schemes for Large-Eddy Simulations (LES).

19.1 Some important points about Large-Eddy Simulations (LES)

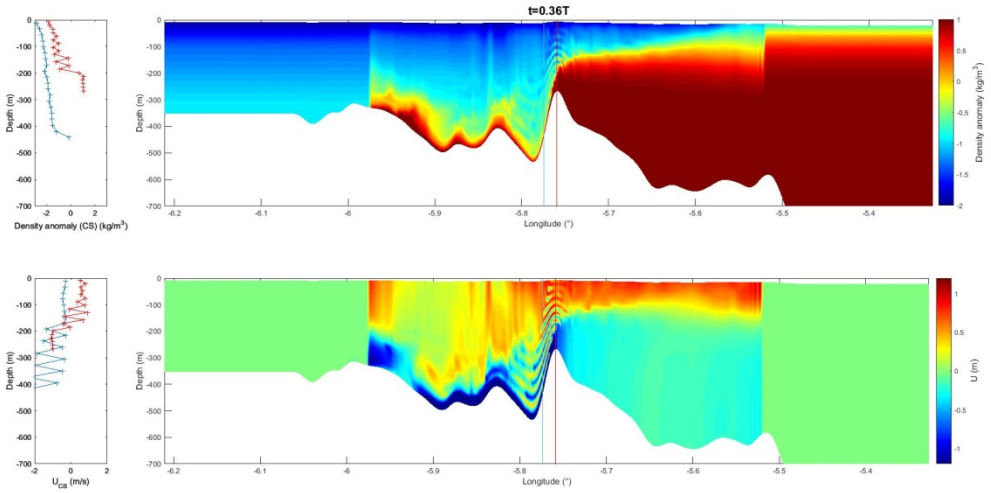
- **Momentum and tracer advection schemes :**

Many advection schemes are now implented in CROCO (i.e. section 4.3 of the model documentation) leading to one recurrent question: which scheme is the most apropriate for my configuration? Unfortunately, no advection scheme is perfect for all applications.

In eddy-resolving ocean simulations significant gradients due to detached eddies or upwelling can be found. More particularly, in coastal eddy-resolving configurations, salinity may vary from river concentration to ocean concentration within a few kilometres in horizontal leading to the formation of even more stronger gradients. In such cases (if your solution has strong gradients, shocks or propagating fronts), it is recommend to use a total variation bounded (4.3.6.5) or a monotonicity-preserving scheme (section 4.3.6.6).

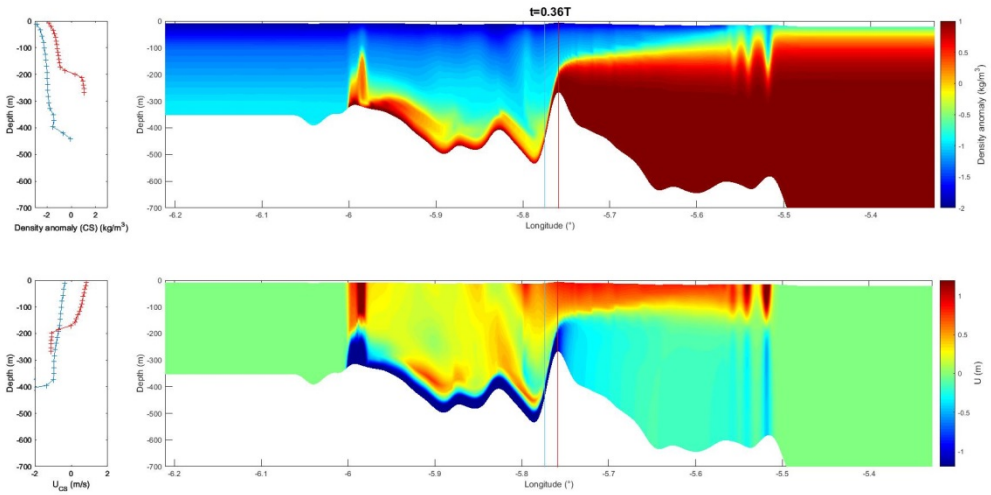
An exemple of numerical artefacs related to the advection schemes: Gibbs phenomenon

Numerical experiments with non-monotonic scheme show artificial oscillations of the solution near regions of sharp gradients (figure below).



Non-monotonic vertical advection schemes (Akima for TS, Spline for UV)

On the contrary, TVD and WENO5 schemes enable sharper shock predictions and as they preserve monotonicity they do not generate spurious oscillations in the solution (figure below).



Monotonic or quasi-monotonic vertical advection schemes (WENO5 for TS, TVD for UVW)

Recommended advection schemes for LES :

CPP options of Momentum Advection

UV_HADV_WENO5	Activate 5th-order WENOZ quasi-monotone lateral advection scheme for UV
UV_VADV_WENO5	Activate 5th-order WENOZ quasi-monotone vertical advection scheme for UV
W_HADV_WENO5	Activate 5th-order WENOZ quasi-monotone lateral advection scheme for W (in NBQ simulation)
W_VADV_WENO5	Activate 5th-order WENOZ quasi-monotone vertical advection scheme for W (in NBQ simulation)

or

UV_HADV_TVD	Activate Total Variation Diminushing lateral advection scheme for UV
UV_VADV_TVD	Activate Total Variation Diminushing vertical advection scheme for UV
W_HADV_TVD	Activate Total Variation Diminushing lateral advection scheme for W (in NBQ simulation)
W_VADV_TVD	Activate Total Variation Diminushing vertical advection scheme for W (in NBQ simulation)

CPP options of Tracer advection

TS_HADV_WENO5	Activate 5th-order WENOZ quasi-monotone lateral tracer advection scheme
TS_VADV_WENO5	Activate 5th-order WENOZ quasi-monotone vertical tracer advection scheme

- **Turbulence schemes : MILES & LES approaches**

In LES, direct transfer ends at the lowest scale resolved, and subgrid dissipation of energy is accomplished by implicit mixing of advection schemes, as well as by explicit parametrization provided by turbulent closure schemes. The choices of advection schemes and/or turbulent closure schemes are thus critical to represent correctly the turbulent energy cascade.

Small scales tend to be more isotropic and homogeneous than the large ones, thus LES requires 3D turbulent closure schemes. Two options of 3D turbulent closure schemes are available in CROCO : A generic two-equation turbulence closure model called Generalized Length Scale (GLS) scheme & a Smagorinsky model (i.e. model documentation). An alternative approach is monotonically integrated LES (MILES). In MILES, the dissipative nature of monotonic advection schemes is exploited to provide an implicit model of turbulence.

Related CPP options (for users):

GLS_MIX2017_3D	Activate 3D Generic Length Scale scheme
UV_VIS_SMAGO_3D	Activate 3D Smagorinsky SGS model

- **Options of Bottom boundary layer**

In coastal seas, the bottom mixed layers may occupy a considerable fraction of the water depth. In contrast, bottom mixed layers in ocean bassins cover only a small portion of the total depth of several thousands of meters. Moreover the strong dissipation of kinetic energy generated by the bed friction can be enhanced in shallow water. Hence the parametrization of the bottom boundary layer dynamic is particularly imortant in coastal large eddy simulations. Some new parametrization options are under development in CROCO to potentially improve the representation of the bottom boundary layers.

BSTRESS_FAST allows solving the bottom friction term of the momentum equations at the fast time step (using part of the code structure inherited from the Non-Boussinesq solver). It avoids reducing the slow-mode (baroclinic) time step for cases with high bottom friction or/and high near bottom vertical resolution. This is not yet a default option as it needs further evaluation in various configurations.

NBQ_FREESLIP imposes a free-slip boundary condition on the bottom (the normal component of the fluid velocity field is set to zero at the bottom level but the tangential component is unrestricted). This is not a default option, by default a no-slip condition is imposed on the bottom. Further evaluation in various configurations is needed.

Related CPP options (for users):

NBQ_FREESLIP	Activate free-slip boundary condition on the bottom
BSTRESS_FAST	solve the bottom friction term of the momentum equations at the fast time step

19.2 KH_INST Test Case

1. **Create a configuration directory:**

```
mkdir ~/CONFIGS/KH_INST
```

2. **Copy the input files for compilation from croco sources:**

```
cd ~/CONFIGS/KH_INST
cp ~/croco/croco/OCEAN/cppdefs.h .
cp ~/croco/croco/OCEAN/param.h .
cp ~/croco/croco/OCEAN/jobcomp .
```

3. **Edit `cppdefs.h` for using KH_INST case**

```
# define KH_INST

# undef REGIONAL
```

Explore the CPP options selected for KH_INST case and undef MPI

```
after #elif defined KH_INST
# undef MPI
```

You can check the KH_INST settings in `param.h`.

4. **Edit the compilation script `jobcomp`:**

```
# set source, compilation and run directories
#
SOURCE=~/croco/croco/OCEAN
SCRDIR=./Compile
RUNDIR=`pwd`
ROOT_DIR=$SOURCE/..
#
# determine operating system
#
OS=`uname`
echo "OPERATING SYSTEM IS: $OS"

#
# compiler options
#
FC=$FC

#
# set MPI directories if needed
#
MPIF90=$MPIF90
MPIDIR=$(dirname $(dirname $(which $MPIF90) ))
MPILIB="-L$MPIDIR/lib -lmpi -limf -lm"
MPIINC="-I$MPIDIR/include"

# set NETCDF directories
#
#------------------------------------------------------------
# Use :
#-lnetcdf           : version netcdf-3.6.3                --
#-lnetcdff -lnetcdf : version netcdf-4.1.2                --
#-lnetcdff          : version netcdf-fortran-4.2-gfortran --
#------------------------------------------------------------
#
```

(continues on next page)

(continued from previous page)

```
#NETCDFLIB="-L/usr/local/lib -lnetcdf"
#NETCDFINC="-I/usr/local/include"
NETCDFLIB=$(nf-config --flibs)
NETCDFINC=-I$(nf-config --includedir)
```

5. **Compile the model:**

```
./jobcomp > jobcomp.log
```

If compilation is successful, you should have a `croco` executable in your directory.

You will also find a `Compile` directory containing the model source files:

- `.F` files: original model source files that have been copied from `~/croco/croco/OCEAN`
- `_.f` files: pre-compiled files in which only parts defined by cpp-keys are kept
- `.o` object files

6. **Copy the namelist input file for KH_INST case:**

```
cp ~/croco/croco/TEST_CASES/croco.in.KH_INST croco.in
```

Eventually edit it.

7. **Run the model:**

```
./croco croco.in > croco.out
```

If your run is successful you should obtain the following files:

```
khinst_rst.nc # restart file
khinst_his.nc # instantaneous output file
```

8. **Have a look at the results:**

```
ncview khinst_his.nc
```

9. Test: some questions:
   - What is the impact of the relaxation of the non-hydrostatic hypothesis?
   - What are the impacts of the advection schemes?
   - What is the impact of adding a turbulent scheme?

19.3 Set up your own NBQ configuration

- In **cppdefs.h** you should activate
  - *NBQ* : activate the non-Boussinesq and non-hydrostatic kernel

```
/* Non-Boussinesq */
# define NBQ
```

- To set up adapted time steps to your NBQ configuration (**dt** & **NDTFAST** in **croco.in** file), you can activate in **cppdefs_dev.h**
- *DIAG_CFL* : activate diagnostics of the CFL criteria

```
# define DIAG_CFL
```

If DIAG_CFL is defined, at each NINFO during the run, CFL criteria are indicated in your output file :

- **INT_3DADV** : Slow (baroclinic) mode CFL criterion. This parameter depends on your mesh grid size and your ocean current intensity (time-varying diagnostic). It should be inferior to approximately 1 (depending on the advection scheme, i.e. section 4.2.5).
- **EXT_GWAVES** : CFL criterion based on the barotropic wave speed. It should be inferior to 0.89 (i.e. section 4.2.5).
- **NBQ_HADV** : CFL criterion based on the pseudo-acoustic wave speed. This parameter should be inferior to 1.7.
- Compile your model
- Edit **croco.in** file, add the following line

```
time_stepping_nbq: NDTNBQ     CSOUND_NBQ     VISC2_NBQ
                   1           "5xsqrt(gHmax)"          1.e-2
```

  - **NDTNBQ** : irrelevant parameter
  - **CSOUND_NBQ** : Pseudo-acoustic waves speed. This parameter should be at least superior to five times the barotropic wave speed (sqrt(gHmax)) in your domain and inferior or equal to the acoustic waves speed (1500 m/s).
  - **VISC2_NBQ** : Bulk Viscosity (i.e. section 1.4)
- Run your simulation.

If it's blow-up, change your time steps to respect the CFL criteria (increase NDTFAST such as NBQ_HADV<1.7). Relaxing the hydrostatic hypothesis, change the dynamic of the small-scale processes and thus potentially the intensity of the small-scale currents leading to more drastic baroclinic CFL conditions. So if it's still blow up, reduce the baroclinic time step (dt).

19.4 NBQ OPTIONS

- **In which cases, do I need to activate the NBQ Precise option?**

Two versions of CROCO-NBQ are currently available: NBQ_PERF & NBQ_PRECISE. NBQ_PERF solve the compressible and non-hydrostatic Navier-Stokes equations and conserve precisely the volume. This version is the most efficient in terms of computational time. NBQ_PRECISE solve also the compressible and non-hydrostatic Navier-Stokes equations and conserve precisely the mass. However, this version is more time consuming (to conserve precisely the mass, an update of the sigma vertical grid at each fast time step is needed). By default, the NBQ_PERF option is defined in the cppdefs_dev.h file. In regional or coastal configurations (resolution ranges of 50-300m), no significant differences in terms of oceanic dynamics have been observed so far. However, specific care is needed when surface waves are explicitly represented. In such configurations, the fluctuations of the vertical grid are more pronounced and NBQ_PRECISE considerably improve the representation of the surface waves. At resolution ranges of 1m, further investigations are needed.

Related CPP options (for users):

NBQ_PERF	The most efficient version in terms of computational time
NBQ_PRECISE	The most precise version in terms of mass conservation

- **Options of open boundaries conditions**

An Orlanski radiation condition (OBC_NBQORLANSKI) has been applied to the internal mode velocities, temperature, and salinity at the open boundaries. Whereas barotropic and acoustic waves are radiated through the boundary using the methods of characteristics. It is recommended to use a sponge layer to deal with strong nonlinearities (as for example to avoid reflexion of solitary waves at the lateral boundaries).

Related CPP options (for users):

OBC_NBQ	OBC
OBC_NBQORLANSKI	Radiative conditions
NBQ_NUDGING	interior/bdy forcing/nudging
NBQCLIMATOLOGY	interior/bdy forcing/nudging
NBQ_FRC_BRY	bdy forcing/nudging

19.5 Appendix : some words on CROCO-NBQ kernel

In CROCO-NBQ, the "fast mode" includes in addition to the external (barotropic) mode, the pseudo-acoustic mode that allows computation of the nonhydrostatic pressure within a non-Boussinesq approach (Auclair et al., 2018). A two-level time-splitting kernel is thus conserved, but the fast time step integrates a 3D-compressible flow. Hence, acoustic waves or "pseudo-acoustic" waves have indeed been re-introduced to avoid Boussinesq-degeneracy which inevitably leads to a 3D Poisson-system in non-hydrostatic Boussinesq methods and to reduce computational costs. As long as "pseudo-acoustic" waves remain faster than the fastest physical processes in the domain, their phase-velocity can artificially be slowed down rendering unphysical high-frequency processes associated with bulk compressibility but preserving a coherent slow non-hydrostatic dynamics with a softening of the CFL criterion. More details are given on *http://poc.omp.obs-mip.fr/auclair/WOcean.fr/SNH/Pub/Tutorials/CROCO/Html_maps/Croco2018_map.html*.

Auclair, F., Bordois, L., Dossmann, Y., Duhaut, T., Paci, A., Ulses, C., Nguyen, C., 2018. A non-hydrostatic non-Boussinesq algorithm for free-surface ocean modelling. Ocean Modelling 132, 12–29. https://doi.org/10.1016/j.ocemod.2018.07.011

Related CPP options (for developers):

NBQ_IMP	The equation of motion for vertical velocity is solved implicitly in the vertical direction.
NBQ_THETA_IMP	The semi-implicit theta method is used to reduce the numerical dissipation iduced by the implicitation of the vertical velocity equation in the vertical direction (i.e. Fringer et al. 2006)
NBQ_HZ_Prognostic	Prognostic the grid evolution
NBQ_AM4	Classical fourth-order Adams-Moulton (AM4) time-stepping method
NOT_NBQ_AM4	Forward-Backward time-stepping method
NBQ_MASS	Perfect conservation of mass (undef NBQ_MASS : perfect conservation of volume)
NBQ_HZCORRECT	The sigma vertical grid is updated at each fast time step to reflect the newly solved elevations (as the free surface is now explicitly resolved at each fast time step).
NBQ_GRID_SLOW	The sigma vertical grid is updated only at each slow time step (reduce the computational time).
HZR Hzr	Trick to change the name of a variable in the equation of mass conservation

Related CPP options (for users):

NBQ	Solving the compressible and non-hydrostatic Navier-Stokes equations

CHAPTER

TWENTY

COUPLING TUTORIAL

Here you will be guided to build a configuration and run it in forced and coupled modes using the tools provided in `croco_tools/Coupling_tools` and `croco/SCRIPTS/SCRIPTS_COUPLING`.

20.1 Summary of steps for coupling

1. **Compilation**
   - Compile OASIS
   - Compile your models in coupled mode **with the same compilers and netcdf libraries**
2. **Namelists**
   - Define the namelist for OASIS: `namcouple`
   - Check/edit the namelists and input files of the different models (CROCO= `croco.in`, WW3: `ww3_grid.inp`, `ww3_shel.inp`, WRF: `namelist.input`, MNH: `EXSEG1.nam`, TOY: `TOYNAMELIST.nam`)
3. **Restart files**
   - Create restart files for the coupler
   - If you are coupling nested models to CROCO, create a `cplmask` file
   - Create restart/input files for the different models (see Preprocessing)
4. **Run**
   - Launch the models simultaneously, e.g.:

```
mpirun -np 4 wwatch : -np 4 crocox
```

5. **Outputs**
   - Check logs and ouptuts, especially:
     - The debug.root.0? files
     - The model log files (e.g. croco.log)
     - If you have problems in your coupled run, first check the dimensions of the grids in all grid files (models grid files and OASIS grids and masks files)

The coupling tools provided with the model will perform steps 2-4. In the following tutorial, you will be guided through all the steps. First, you will try a simple coupling example to help you understand the coupling philosophy and steps to run a coupled simulation, then you can go to the advanced tutorial to perform coupled simulations using the provided coupling tools and scripts.

20.2 Compiling in coupled mode

Warning: You need to compile OASIS **before** compiling the models

Note: In case of error during compilation, refer to the "Tips in case or error during compilation" below

20.2.1 Compiling OASIS

1. You need to have dowloaded the OASIS sources (see Download section). They are assumed, in the following, to be under: `$HOME/oasis/oasis3-mct`
2. Then, explore the `oasis3-mct` directory, you will find:
   - `doc`: oasis documentation
   - `lib`: `mct`, `psmile`, and `scrip` libraries folders
   - `util`: with notably `make_dir` folder containing `TopMakefileOasis3`, `make.inc`, and several `make.*` for different machines
   - `examples`
3. Enter the `make_dir` directory:

```
cd ~/oasis/oasis3-mct/util/make_dir
```

4. Edit the configure file for your machine `make.*`.
5. Edit the `make.inc` file to point to your `make.yourmachine` (examples for machines we have tested can be found here: `$HOME/croco/croco/SCRIPTS/SCRIPTS_COUPLING/OASIS_IN/make.*`)

```
include  $(home)/oasis/oasis3-mct/util/make_dir/make.YOURMACHINE
```

Warning: Absolute paths are mandatory in `make.*` files!

6. Clean your directory and launch compilation:

```
make realclean -f TopMakefileOasis3 > oasis_clean.out
make -f TopMakefileOasis3 > oasis_make.out
```

If compilation is successful, you should find in `~/oasis/` a `compile_oasis3-mct` directory including:

- `lib` containing `libmct.a libmpeu.a libpsmile.MPI1.a libscrip.a`
- `build`

Note: In case of error during compilation, note that classical errors are associated to:

- files missing executable permission
- issues in the paths given in `make.yourmachine`
- compilation options that have to be set carefully (in `make.YOURMACHINE`)

20.2.2 Compiling CROCO

1. To work in coupled mode you need to activate `OA_COUPLING` and/or `OW_COUPLING` in `cppdefs.h`:

```
#define OA_COUPLING
#define OW_COUPLING
```

   You also need to define `MPI`:

```
#define MPI
```

> **Warning:** `MPI` is mandatory for coupling, even if the run is launched on 1 CPU. Indeed the MPI communicator is used to communicate with OASIS.

2. Edit all the usual paths, compilers, libraries in `jobcomp`, and notably OASIS path `PRISM_ROOT_DIR`:

```
# set OASIS-MCT (or OASIS3) directories if needed
#
PRISM_ROOT_DIR=~/oasis/compile_oasis3-mct
```

   You may also eventually need to set/change compilation options.

> **Warning:** `-O3` compilation option is quite agressive and may result in some errors on some machines and with some compilers during coupled run (e.g. stokes velocities set to 0). To avoid such errors, set optimization to `-O2`.

3. And compile:

```
./jobcomp >& compile_coupled.log
```

   If compilation aborts (netcdf errors in oasis functions), you may need to change the following lines to:

```
LDFLAGS1="$LDFLAGS1 $LIBPSMILE $NETCDFLIB"
CPPFLAGS1="$CPPFLAGS1 ${PSMILE_INCDIR} $NETCDFINC"
FFLAGS1="$FFLAGS1 ${PSMILE_INCDIR} $NETCDFINC"
```

   Then try to compile again.

20.2.3 Compiling the TOY model

A toy model is available in the `croco/SCRIPTS/SCRIPTS_COUPLING/TOY_IN`. It consists of a few fortran routines, that exchange variables with OASIS to mimic a wave or atmosphere model. The toy model is compiled using a Makefile. See the `readme` in `croco/SCRIPTS/SCRIPTS_COUPLING/TOY_IN` for instructions.

1. Copy the TOY model in your configuration directory and check/edit the Makefile.YOURMACHINE for your machine (examples for a few clusters are provided), and link it to `Makefile`:

```
cp -r ~/croco/croco/SCRIPTS/SCRIPTS_COUPLING/TOY_IN ~/CONFIGS/BENGUELA_
↪LR_cpl/.
cd ~/CONFIGS/BENGUELA_LR_cpl/TOY_IN
ln -sf Makefile.YOURMACHINE Makefile
```

2. Then clean and compile:

```
make clean &> toy_clean.out
make &> toy_make.out
```

If the compilation is successfull you should have the TOY executable `toy_model`

20.2.4 Compiling WRF

Note: Currently the distributed version of WRF does not include coupling with waves, if you want to use such functionality you can use the fork including modifications for coupling with WW3 and CROCO through the OASIS coupler, but note that this is a development version... https://github.com/wrf-croco/WRF/tree/WRF-CROCO

WRF needs to be compiled both in forced and coupled modes.

1. Enter WRF directory, and configure your compilation:

```
cd ~/wrf/WRFV4.2.1
# cleaning before configure (must be done if you re-compile)
./clean -a
# Then launch configure
./configure
```

   Choose distributed memory option (dm) and compiler option in adequation with your machine setup (in our case it will be #24).

   Note:

   - For creating model output files larger than 2Go, you should consider using netcdf large file support function. It is activated through the `WRFIO_NCD_LARGE_FILE_SUPPORT` environment variable (set to 1).
   - WRF is strict on netcdf dependencies, meaning that problems during compilation are often due to netcdf settings. WRF uses:
     - `NETCDF` environment variable that can be set before launching configure, otherwise configure will ask you to provide your netcdf full path
     - `NETCDF4` environment variable that can be set to 1 if you want to use netcdf 4 facilities (if your netcdf library allows it). When using netcdf4 library, check if all dependencies are properly set, they are usually found with `nf- config --flibs` command
     - always check all the lines associated to netcdf library and dependencies in the generated `configure.wrf`: `NETCDF4_IO_OPTS`, `NETCDF4_DEP_LIB`, `INCLUDE_MODULES` (last line should be netcdf inlcude path), `LIB_EXTERNAL` (last line should be netcdf library and its dependencies).

2. Check and edit the generated `configure.wrf` file. Notably edit the parallel compiler lines:

```
DM_FC           =       mpiifort
DM_CC           =       mpiicc
```

3. First compile in uncoupled mode:

```
./compile em_real >& compile_uncoupled.log
```

   Note: WRF supports using multiple processors for compilation. The default number of processors used is 2. But you can compile with more processors by using the `J` environment variable set (example for 8 processors: `J=-j 8`).

   Note: WRF compilation will take a while (about 1h) and may take a lot of memory. You may need to launch compilation in a job. Examples for a few machines are provided here, along with a script to help you compile:

```
~/croco/croco/SCRIPTS/SCRIPTS_COUPLING/WRF_IN/*.compile.wrf.*
~/croco/croco/SCRIPTS/SCRIPTS_COUPLING/WRF_IN/make_WRF_compil
```

If compilation is successful, you will find in `main` the following executables:

`wrf.exe` `real.exe` `ndown.exe` `tc.exe`

4. Copy them to dedicated directory (as well as your configure.wrf, in case you need to recompile):

```
mkdir exe_uncoupled
cp configure.wrf exe_uncoupled/.
cp main/*.exe exe_uncoupled/.
cp compile_uncoupled.log exe_uncoupled/.
```

5. To compile in coupled mode, you need to edit `configure.wrf`, first copy it to `configure.wrf.coupled`:

```
cp configure.wrf configure.wrf.coupled
```

And then edit `configure.wrf.coupled`:

```
# Just before: #### Architecture specific settings ####, add for OASIS:
OA3MCT_ROOT_DIR = $(OASISDIR)

# In: #### Architecture specific settings ####, add -Dkey_cpp_oasis3 :
ARCH_LOCAL      =       -DNONSTANDARD_SYSTEM_FUNC -DWRF_USE_CLM $(NETCDF4_IO_
↪OPTS) -Dkey_cpp_oasis3

# In: # POSTAMBLE, add includes and libraries associated to OASIS before␣
↪netcdf ones, as follows:
INCLUDE_MODULES =       $(MODULE_SRCH_FLAG) \
                        $(ESMF_MOD_INC) $(ESMF_LIB_FLAGS) \
                        -I$(WRF_SRC_ROOT_DIR)/main \
                        -I$(WRF_SRC_ROOT_DIR)/external/io_netcdf \
                        -I$(WRF_SRC_ROOT_DIR)/external/io_int \
                        -I$(WRF_SRC_ROOT_DIR)/frame \
                        -I$(WRF_SRC_ROOT_DIR)/share \
                        -I$(WRF_SRC_ROOT_DIR)/phys \
                        -I$(WRF_SRC_ROOT_DIR)/chem -I$(WRF_SRC_ROOT_DIR)/inc \
                        -I$(OA3MCT_ROOT_DIR)/build/lib/mct \
                        -I$(OA3MCT_ROOT_DIR)/build/lib/psmile.MPI1 \
                        -I$(NETCDFPATH)/include \

 LIB_EXTERNAL    = \
                     -L$(WRF_SRC_ROOT_DIR)/external/io_netcdf -lwrfio_nf \
                     -L$(OA3MCT_ROOT_DIR)/lib -lpsmile.MPI1 -lmct -lmpeu -
↪lscrip \
                     -L$(NETCDF)/lib -lnetcdff -lnetcdf
```

Examples of `configure.wrf.uncoupled` and `configure.wrf.coupled` are provided in `$HOME/croco/croco/SCRIPTS/SCRIPTS_COUPLING/WRF_IN/CONFIGURE_WRF/`.

Warning: Compiling WRF in coupled mode required a lot of memory (>3.5Go). If needed, submit a job with extra-memory to compile.

6. To compile:

```
./clean -a # clean before compilation
cp configure.wrf.coupled configure.wrf
./compile em_real >& compile.coupled.log
```

If compilation is successful, you will find in `main` the following executables:

```
wrf.exe real.exe ndown.exe tc.exe
```

7. Copy them to dedicated directory (as well as your configure.wrf, in case you need to recompile):

```
mkdir exe_coupled
cp configure.wrf exe_coupled/.
cp main/*.exe exe_coupled/.
cp compile.coupled.log exe_coupled/.
```

WPS compilation

1. Enter WPS directory, and configure your compilation:

```
cd ~/wrf/WPS # note that you should use the WPS version consistent with your␣
↪WRF version!
./clean -a
./configure
```

Choose distributed memory option (dm) and compiler option in adequation with your machine setup.

2. Check and edit `configure.wps`, notably `WRF_DIR` and compilers:

```
WRF_DIR                 =       ../WRF

DM_FC               = mpiifort
DM_CC               = mpiicc
```

3. Compile WPS:

```
./compile >& compile_wps.log
```

If compilation is successful, you will find in your WPS directory:

```
geogrid.exe
ungrib.exe
metgrid.exe
```

Alternatively, a `compile_wps.bash` and examples of `configure.wps` are provided in the `Coupling_tools/WRF_WPS`.

20.2.5 Compiling WW3

1. Go to the model bin directory to perform the compilation:

```
cd ~/ww3/model/bin
```

WW3 compilation requests 3 files:

- a switch file which contains the parallelisation, and the numerical parameterization setting. These swhiches are keywords listed in a so-called switch file. Many templates are provided by institutions with a suffix `switch_*`. This file is used during compilation. Open one of the coupled example switch file: `switch_OASOCM` (for coupling with an ocean model) or `switch_OASACM` (for coupling with an atmospheric model)
  - For running in coupled mode, some switches are mandatory:

    `DIST MPI COU OASIS` and `OASOCM` (for coupling with an ocean model) and/or `OASACM` (for coupling with an atmospheric model)

  - Also, the switches to interpolate in time current or wind need to be set to 0 in coupled case mode (and forced cases used to compare to coupled mode):

    CRT0 WNT0

- a comp.COMPILER file
- a link.COMPILER file

The 2 later files contain useful options and links for compilation. You therefore need to check the ones that you will use depending on you compiler and machine settings.

In this tutorial, let's take the example of `comp.Intel` and `link.Intel` files.

2. You can edit the compilation options in `comp.Intel`, for instance:

```
opt="-c $list -O3 -ip -xHost -no-fma -fp-model precise -assume byterecl -fno-
↪alias -fno-fnalias -module $path_m"
```

3. First we will compile WW3 in uncoupled mode. To do that, create an equivalent switch file than `switch_OASOCM` but without coupling switches:

```
cp switch_OASOCM switch_UNCOUPLED
```

In `switch_UNCOUPLED`, erase the following switches: `COU OASIS OASOCM`

4. Now you are ready to setup and compile WW3:

```
./w3_setup .. -c Intel -s UNCOUPLED
./w3_automake
```

If compilation is successful, you will find your executables in `../exe`, you should move these executables to a dedicated directory:

```
mkdir ../exe_UNCOUPLED
mv ../exe/* ../exe_UNCOUPLED/.
```

5. To compile in coupled mode, check that the `$OASISDIR` variable correctly refers to your OASIS compile directory, and re-setup and re-launch your compilation:

For coupling with the ocean:

```
./w3_clean -c
./w3_setup .. -c Intel -s OASOCM
./w3_automake
```

If compilation is successful, you should move your executable to a proper directory:

```
mkdir ../exe_OASOCM
mv ../exe/* ../exe_OASOCM/.
```

For coupling with the atmosphere:

```
./w3_clean -c
./w3_setup .. -c Intel -s OASACM
./w3_automake
```

If compilation is successful, you should move your executable to a proper directory:

```
mkdir ../exe_OASACM
mv ../exe/* ../exe_OASACM/.
```

For coupling with both the ocean and the atmosphere, first create a `switch_OASOCM_OASACM`:

```
cp switch_OASOCM switch_OASOCM_OASACM
```

Edit it to have both OASOCM and OASACM switches:

```
F90 NOGRB NC4 TRKNC DIST MPI PR3 UQ FLX0 LN1 ST4 STAB0 NL1 BT4 DB1 MLIM TR0␣
↪BS0 IC2 IS0 REF1 XX0 WNT2 WNX1 RWND CRT0 CRX1 TIDE COU OASIS OASOCM OASACM␣
↪O0 O1 O2 O2a O2b O2c O3 O4 O5 O6 O7
```

And compile:

```
./w3_clean -c
./w3_setup .. -c Intel -s OASOCM_OASACM
./w3_automake
```

If compilation is successful, you should move your executable to a proper directory:

```
mkdir ../exe_OASOCM_OASACM
mv ../exe/* ../exe_OASOCM_OASACM/.
```

Note: a script to help you compile the various mode is also available in: `$HOME/croco/croco/SCRIPTS/SCRIPTS_COUPLING/WW3_IN/make_WW3_compil`

20.2.6 Tips in case of errors during compilation

In case of strange errors during compilation (*e.g.* "catastrophic error: could not find ..."), try one of these solutions:

- check your home space is not full ;-)
- check your paths to compilers and libraries (especially Netcdf library)
- check that you have the good permissions, and check that your executable files (configure, make...) do are executable
- check that your shell scripts headers are correct or add them if necessary (*e.g.* for bash: `#!/bin/bash`)
- try to exit/log out the machine, log in back, clean and restart compilation

Errors and tips related to netcdf library:

- with netcdf 4.3.3.1: need to add the following compilation flag for all models: `-mt_mpi`

  The error associated to a missing `-mt_mpi` flag is of this type: " /opt/intel//impi/4.1.1.036/intel64/lib/libmpi_mt.so.4: could not read symbols: Bad value "

- with netcdf 4.1.3: do NOT add `-mt_mpi` flag
- with netcdf4, need to place hdf5 library path in your environment:

```
export LD_LIBRARY_PATH=YOUR_HDF5_DIR/lib:$LD_LIBRARY_PATH
```

- with netcdf 4, if you use the library splitted in 2: C part and Fortran part, you need to place links to C library before links to Fortran library and need to put both path in this same order in your LD_LIBRARY_PATH

In case of 'segmentation fault' error:

- try to allocate more memory with "unlimited -s unlimited"
- try to launch the compilation as a job (batch) with more allocated memory

20.3 Simple CROCO-TOY coupled example

For this first step towards coupling, we will just use the BENGUELA_LR configuration and add coupling to a toy model that mimics a wave model. The toy model is available in the `croco/SCRIPTS/SCRIPTS_COUPLING/TOY_IN`. It consists of a few fortran routines, that exchange variables with OASIS to mimic a wave or atmosphere model. For a more advanced coupling with actual atmospheric and wave models, you can go to the other sections of the coupling tutorial.

1. First copy the `BENGUELA_LR` configuration that you have already run in forced mode:

```
cp -r ~/CONFIGS/BENGUELA_LR ~/CONFIGS/BENGUELA_LR_cpl
```

2. For running in coupled mode, you first need to compile OASIS, and then re-compile CROCO in coupled mode, and compile the TOY model. Follow the instructions in the Compilation section of the coupling tutorial.

3. Set up the TOY model:

   The toy model can send either fields from a model file (for instance generated by running a model in forced mode previously), or constant or sinusoidal fields. Check the readme in toy_in for more informations. In every cases, you will need to provide a grid to the toy model, here named `grid_wav.nc`. The toy model will read and exchange variables specified in the `TOYNAMELIST.nam` from an input file, here named `toy_wav.nc`. First edit the `TOYNAMELIST.nam` file: exchanged field names and number of time steps In the current example, the toy model is set to run 72 time steps of 3600s.

   For this tutorial, we will thus use the `toy_wav.nc` and `grid_wav.nc` files provided. You should have the following executable, namelist, and input files to use the TOY model:

   - toy_model
   - TOYNAMELIST.nam
   - grid_wav.nc
   - toy_wav.nc

4. Set up CROCO:

   Edit the `croco.in` to run over the same duration:

```
time_stepping: NTIMES   dt[sec]  NDTFAST  NINFO
                72      3600      60      1
```

   You can also change the frequency of outputs:

```
history: LDEFHIS, NWRT, NRPFHIS / filename
            T      24     0
    CROCO_FILES/croco_his.nc
averages: NTSAVG, NAVG, NRPFAVG / filename
            1      24     0
    CROCO_FILES/croco_avg.nc
```

   And set to True the outputs for waves fields:

```
wave_history_fields: hrm  frq  action  k_xi  k_eta  eps_b  eps_d Erol␣
↪eps_r
                     20*T
wave_average_fields: hrm  frq  action  k_xi  k_eta  eps_b  eps_d Erol␣
↪eps_r
                     20*T
```

5. Edit OASIS namelist, `namcouple`, to specify which fields will be coupled. A basis of namcouple files can be found in the `croco/SCRIPTS/SCRIPTS_COUPLING/OASIS_IN` directory. Copy the relevant namcouple:

```
cp ~/croco/croco/SCRIPTS/SCRIPTS_COUPLING/OASIS_IN/namcouple.base.ow.
↪toywav ~/CONFIGS/BENGUELA_LR_cpl/namcouple
```

In this namcouple, you will have to edit all the fields denoted into brackets <...>. Let's browse the namcouple file. It has several sections:

- A first section with general settings:
  - the number of fields to exchange (in our case 7: 3 from the ocean to the wave model (SSH, UOCE, VOCE), and 4 from the wave to the ocean model (HS, T0M1, SDIR, CDIR))
  - the number and names of model executables: here names must be of 6 characters exactly, so you need to move your model executable names to these 6-character names:

```
mv croco crocox
mv toy_model toyexe
```

  - the duration of the run in seconds: you need to change <runtime> to your actual duration (3days * 24h * 3600s): 259200
  - the debug level (see detailed explanation in the comments in the namcouple file)
- A second section, with the informations on exchanged fields. A typical sub-section for one exchanged field looks like:

```
SRMSSHV0 TOY__SSH 1 <cpldt> 1 oce.nc EXPORTED
<ocenx> <oceny> <wavnx> <wavny> ocnt toyt LAG=<ocedt>
R  0  R  0
SCRIPR
DISTWGT LR SCALAR LATLON 1 4
```

line 1: field in sending model, field in target model, unused, coupling period, number of transformations (here 1 interpolation), restart file, field status

line 2: nb of pts for sending model grid (without halo) first dim, and second dim, for target grid first dim, and second dim, sending model grid name, target model grid name, lag = time step of sending model

line 3: sending model grid periodical (P) or regional (R), and nb of overlapping points, target model grid periodical (P) or regional (R), and number of overlapping points

line 4: list of transformations performed (here only grid interpolation SCRIPR keyword, see OASIS documentation for more informations)

line 5: parameters for each transformation (here distributed weight interpolation, see OASIS documentation for more informations)

You need to edit all the fields denoted into brackets: <...>:

<cpldt>	the coupling frequency in seconds for each field you will exchange
<ocenx>	the number of points in xi direction for CROCO (see param.h)
<oceny>	the number of points in eta direction for CROCO (see param.h)
<wavnx>	the number of points in x direction for the TOY model (see grid_wav.nc file)
<wavny>	the number of points in y direction for the TOY model (see grid_wav.nc file)
<ocedt>	the CROCO time step
<wavdt>	the TOY model time step (see TOYNAMELIST.nam)

6. Finally, you need to prepare restart files for the coupler (in addition to model initial/restart files). To do so, two scripts are provided in the Coupling tools to start from calm conditions or previously existing files. In our case we will start from calm conditions. Note that this script uses the nco library, so that you should have it installed/loaded to run the script:

```
# Copy the useful script
cp ~/croco/croco/SCRIPTS/SCRIPTS_COUPLING/SCRIPTS_TOOLBOX/ROUTINES/
↪OASIS_SCRIPTS/create_oasis_restart_from_calm_conditions.sh ~/CONFIGS/
↪BENGUELA_LR_cpl/.

# launch the creation of restart file for OASIS for the toy model:
#    first argument: grid name
#    second argument: restart file name
#    third argument: type of model
#    fourth argument: list of variables to initialize to 0
./create_oasis_restart_from_calm_conditions.sh grid_wav.nc wav.nc toy
↪"TOY_T0M1 TOY___HS TOY_CDIR TOY_SDIR"

# launch the creation of restart file for OASIS for CROCO model:
./create_oasis_restart_from_calm_conditions.sh CROCO_FILES/croco_grd.
↪nc oce.nc croco "SRMSSHV0 SRMVOCE0 SRMUOCE0"
```

   You should have now in your configuration directory `wav.nc` and `oce.nc`, which are the OASIS restart files.

7. You are now ready to run CROCO in coupled mode with the toy model:

```
mpirun -np 2 toyexe : -np 4 crocox
```

   Or edit and launch a job to run the coupled models.

   If the run went well, you should have in your configuration directory the following files:

```
grids.nc # grid file for OASIS (created automatically)
areas.nc # areas of cells used by some OASIS interpolations (created␣
↪automatically)
masks.nc # masks file for OASIS (created automatically)
rmp_ocnt_to_toyt_DISTWGT.nc
rmp_toyt_to_ocnt_DISTWGT.nc
rmp_ocnu_to_toyt_DISTWGT.nc
rmp_ocnv_to_toyt_DISTWGT.nc # weight files for OASIS interpolation␣
↪(one for each grid interpolation)
nout.000000 # OASIS log file
toyexe.timers_0000 # OASIS log file for time statistics
crocox.timers_0000 # OASIS log file for time statistics
debug.root.01 # OASIS log file for the master processor for model #1␣
↪(toy in our case)
debug.root.02 # OASIS log file for the master processor for model #2␣
↪(CROCO in our case)
debug.notroot.01 # OASIS log file for other processors for model #1␣
↪(toy in our case)
debug.notroot.02 # OASIS log file for other processors for model #2␣
↪(CROCO in our case)
OUTPUT_TOY.txt # log file for the toy
croco.log # log file for CROCO (if you have define the LOGFILE cpp-key,
↪ otherwise croco log output is in CPL.o???????)
CPL.o??????? # log file for the batch job
```

Note: If you have problems running the coupled model, you need to check:

- The dimensions of the grids in all grid files (models grid files and OASIS grids and masks files)

- The `debug.root.0?` files
- The model log files (*e.g.* `croco.log`)

You can then check your new CROCO outputs in CROCO_FILES (you can see that you have the additional wave fields outputs (*e.g.* hrm) and if you can see small differences of the surface currents for example if you do a difference of coupled and non-coupled CROCO outputs).

8. If you want then to use actual coupling with an atmospheric or wave model, and run production simulation in coupled mode, follow the next steps of the Coupling tutorial. It uses the full Coupling toolbox provided in croco_tools/Coupling_tools and croco/SCRIPTS/SCRIPTS_COUPLING. It will help you create a dedicated architecture for coupled runs, and it will provide you a set of scripts for running coupled simulation without managing all the files one by one. Basically, the Coupling toolbox will manage:
   - CROCO compilation if requested
   - Copying the model executables to your configuration directory
   - Getting models input files
   - Preparing OASIS restart files
   - Editing namelists, that is replacing automatically all the fields into brackets `<...>` in the different namelist files (for all models and for OASIS)
   - Launching the run
   - Putting output files in a dedicated output directory
   - Putting restart files for a future run in a dedicated restart directory
   - Eventually launching the next job if requested

20.4 Advanced coupling tutorial

If you have successfully run the simple CROCO-TOY coupled example, and you want to perform more advanced coupled simulation, you can follow this advanced coupling tutorial.

Note that it requires to be quite familiar with the various models to couple.

A set of coupling tools has been designed to help building and runing coupled configurations. It is provided within the `croco/SCRIPTS/SCRIPTS_COUPLING` directory.

Some pre-processing tools are also provided in the `croco_tools/Coupling_tools` directory.

First the contents of the SCRIPTS_COUPLING toolbox will be described, and then the different steps for running a coupled simulation.

20.4.1 Coupling tools contents

The `croco/SCRIPTS/SCRIPTS_COUPLING` toolbox contains several sub-directories:

- SCRIPTS_TOOLBOX: contains all the scripts, namelists, and routines
- OASIS_IN: contains base namelists, and compilation file examples
- CROCO_IN: contains base namelist
- WW3_IN: contains base namelists, and useful files for compilation
- WRF_IN: contains base nameslist, and usesul files for compilation
- TOY_IN: contains the toy mode sources, and base namelists

submitjob.sh	Script to create and launch job
mynamelist.sh	Namelist for the run (models, dt, output,. . .)
myjob.sh	Informations about the job (date, job duration,. . .)
myenv_mypath.sh	Machine environment and path to models

In `OASIS_IN`:

make.ADA	Example file for OASIS compilation on ADA IDRIS cluster
make.DATARMOR	Example file for OASIS compilation on DATARMOR cluster
namcouple.base.*	Namelist files for the different coupled modes in which <. . .> will be replaced by `cpl_nam.sh` from `SCRIPTS_TOOLBOX`
namcouple.base.aw.debug	Example of a namelist files with debug options
namcouple.base.aw.nointerp	Example of namelist with given interpolation file

In `CROCO_IN`:

croco.in.base	Base namelist file for CROCO (timestepping, input, output. . .), in which <. . .> will be replaced by `oce_nam.sh` from `SCRIPTS_TOOLBOX`

In `WRF_IN`:

configure.namelist.real	Configure file to edit for running real
run_real.bash	Script to run real (wrf pre-processing)
job.real.*	Job script to run real
make_WRF_compil	Script to compile wrf
MACHINE.compile.wrf.*	Jobs to launch `make_WRF_compil` on some MACHINES
namelist.input.base.complete	Namelist base in which <...> will be replaced by `run_real` and `atm_nam.sh` from `SCRIPTS_TOOLBOX`
README.namelist	Readme to know all the namelist options available (also available in WRF)
myoutfields.txt	Example of file that can be prescribed in wrf namelist to add/remove variable outputs
CONFIGURE_WRF/MACHINE	
configure.wrf.coupled	Example of configure file for compiling wrf in coupled mode
configure.wrf.uncoupled	Example of configure file for compiling wrf in forced mode

In `WW3_IN`:

switch_*	Switches for the different modes
ww3_grid.inp.base	Grid input file in which <...> (timesteps, etc) will be replaced by `wav_getfile.sh` script
ww3_prnc.inp.*	prnc input file for prerpating ww3 input files
ww3_strt.inp	strt input file for running `ww3_strt`
ww3_shel.inp.base.*	shel input files for the different modes in which <...> (dates, etc) will be replaced by `wav_getfile.sh`
ww3_ounf.inp.base	ounf input file in which dates will be replaced
ww3_bounc.inp	boundary input file for running `ww3_bounc`

In `SCRIPTS_TOOLBOX`

*_nam.sh	Update pre-filled namelist with `mynamelist.sh` informations
_get.sh	Get input files for the models
*_putfile.sh	Retrieve output and restart files and put them where it is specified in header_*.sh
chained_job.sh	Submit all jobs at the beginning with the following having condition on the previous
caldat.sh	Return the calendar date and time given julian date
julday.sh	Calculate the Julian Day Number for a given month, day, and year
caltools.sh	Compute dates for the experiment
getversion.sh	Return model's version used (and write it in the log file)
MACHINE	
header.MACHINE	Job header for different machines, paths toward model's executables, input directories, namelist but also execution, output and restart directories
launch.MACHINE	Script to create app.conf file for launching coupled runs with MPMD[1]
myenv.MACHINE*	Necessary modules on the different MACHINES to compile and run the models
NAMELIST	
namelist_*	Different namelists which are concatenated, in create_config, to build `mynamelist.sh`
PATHS	
path_*.sh	Script used in `create_config` to build `myenv_mypath.sh`
OASIS_SCRIPTS	
create_oasis_grids_for_wrf.sh	Script to create grids.nc and masks.nc files for OASIS for WRF (useful only if you are using a version of WRF in which the oasis function is not implemented. In the wrf-croco fork the function is implemented and this script is not used).
create_oasis_restart_from_cal...	

The `croco_tools/Coupling_tools` toolbox contains:

CROCO	
README_preprocess_croco	Readme to use croco_tools classic pre-processing (in matlab)
README_nest_cpl	Readme to prepare nests in coupled runs
make_grid_from_WRF.m	Script to generate a grid for CROCO from WRF grid with eventually a refinement coefficient
find_childgrid_inparentgrid.m	Script to Find the position of a nested grid in the parent before using AGRIF tools
job_prepro_matlab.pbs	Example job to run matlab preprocessing on a super-computer
prepro_*.m	Example scripts used by the job script
WW3	
make_ww3_grd_input_i..._grd.m	Script to generate coord. and bathy. file for WW3 from croco_grd.nc file
script_make_CFSR_wind_for_ww3.sh	Script to create wind input file for WW3 from CFSR
script_make_WRF_wind_for_ww3.sh	Script to create wind input file for WW3 from WRF
script_make_CROCO_current....sh	Script to create current and level input files for WW3
UV2T.sh	Useful functioni to change from U,V to T grid, used in above-mentionned scripts
WRF_WPS	
README_download_CFSR_data	Some useful readme for WPS
README_wps	Some useful readme for WPS
README.Vtable	Some useful readme for WPS
configure.namelist.wps	Configure file to edit for running WPS
Vtable.CFSR_sfc_flxf06	Vtables for CFSR data
Vtable.CFSR_press_pgbh06	Vtables for CFSR data
Vtable.GDAS_4soillevel_my	Vtable for GFS/GDAS data
METGRID.TBL.GDAS	Table for Metgrid
job.wps.*	Job scripts to run WPS pre-processing
run_wps.bash	Script to run wps (wrf pre-processing)
CONFIGRE_WPS	Examples of configure files for compiling WPS

[1] MPMD (Multiple Program Multiple Data) is supported on some machines. Different executables are launched and communicate with each other using MPI; all MPI processes are included within the same MPI_COMM_WORLD communicator. This execution method uses a text file (call here app.conf) which contains the mapping between MPI processes and executables

Footnote

20.4.2 Coupling tools philosophy and workflow

The idea of the coupling tools is to facilitate the management of coupled configurations, the run, and displacement of I/O.

First step is to create a configuration with the usual `create_config.bash` script, by specifying wich models you want to use in the `models` options.

From there a configuration architecture will be built:

```
HOMEDIR/CONFIGS/MY_CONFIG_NAME
              create_config.bash.bck
              myenv_mypath.sh
              mynamelist.sh
              myjob.sh
              submitjob.sh
              - SCRIPTS_TOOLBOX
              - PREPRO
              - OASIS_IN
              - CROCO_IN
              - WW3_IN
              - WRF_IN
              - XIOS_IN
WORKDIR/CONFIGS/MY_CONFIG_NAME
              - OASIS_FILES
              - CROCO_FILES
              - WW3_FILES
              - WRF_FILES
              - DATA
```

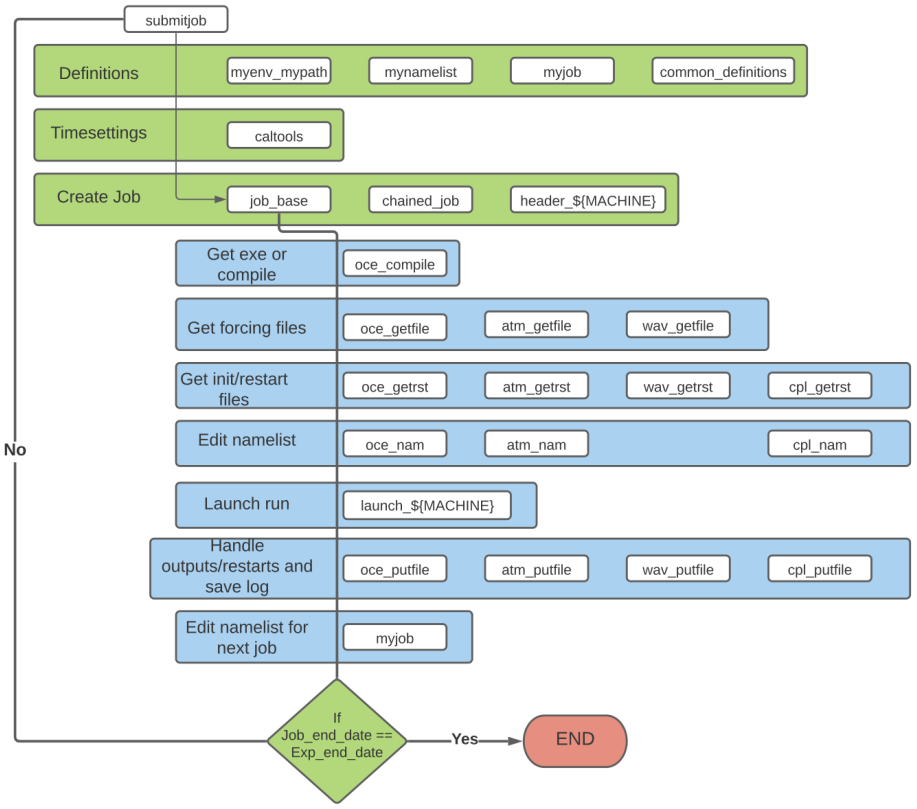
The user will provide:

- the environment settings, and paths within the `myenv_mypath.sh` script
- the settings for the experiment (which models, time stepping, input files...) in `mynamelist.sh`
- the settings for the job (dates notably) in `myjob.sh`

Then the user launch the job with `./submitjob.sh`.

The coupling toolbox manage:

- CROCO compilation if requested
- Copying the model executables to your configuration directory
- Getting models input files
- Preparing OASIS restart files
- Editing namelists, that is replacing automatically all the fields into brackets <...> in the different namelist files (for all models and for OASIS)
- Launching the run
- Putting output files in a dedicated output directory
- Putting restart files for a future run in a dedicated restart directory
- Eventually launching the next job if requested



20.4.3 Create your configuration

To prepare your configuration working directory, you can use the script `create_config.bash` provided in CROCO sources:

```
cp ~/croco/croco/create_config.bash ~/CONFIGS/.
```

Edit your paths and settings in `create_config.bash`:

```
#==========================================================================================
# BEGIN USER MODIFICATIONS

# Machine you are working on
# Known machines: Linux DATARMOR IRENE JEANZAY
# ------------------------------------------
MACHINE="Linux"

# CROCO parent directory
# (where croco_tools directory and croco source directory can be found)
# -----------------
CROCO_DIR=~/croco/croco
TOOLS_DIR=~/croco/croco_tools

# Configuration name
# ------------------
MY_CONFIG_NAME=BENGUELA_cpl

# Home and Work configuration directories
# ---------------------------------------
MY_CONFIG_HOME=~/CONFIGS
```

(continues on next page)

(continued from previous page)

```
MY_CONFIG_WORK=~/CONFIGS

# Options of your configuration
models_incroco=( all-prod )
```

Run `create_config.bash`:

```
./create_config.bash
```

Go into your configuration directory, open, check and eventually edit paths in `myenv_mypath.sh`, and source it (you need to be in a bash environment):

```
source myenv_mypath.sh
```

It will set a few useful paths and environment variables.

20.4.4 Pre-processing for coupled run

CROCO preprocessing

You can run CROCO pre-processing as usual in the `$HOME/CONFIGS/BENGUELA_cpl/PREPRO/CROCO` directory. See the usual Pre-processing tutorial.

WW3 pre-processing

WW3 GRIDGEN

Preprocessing tools for WW3 have been developed under Matlab software. They are available in the GRIDGEN matlab package (a tutorial is available here: ftp://ftp.ifremer.fr/ifremer/ww3/COURS/WAVES_SHORT_COURSE/TUTORIALS/TUTORIAL_GRIDGEN/waves-workshop-exercise-gridgen.pdf).

Basic steps for regular grids are summarized here:

1. Define your grid parameters:

```
dx= ... # in degrees
dy= ... # in degrees
lon1d=[...:dx:...] # in degrees
lat1d=[...:dy:...] # in degrees
[lon,lat]=meshgrid(lon1d,lat1d);
```

2. Coastline (defined as polygons in coastal bound ....mat) and bathy (e.g., etopo1.nc) files are used. Some threshold values are set up:

```
lim_wet=... ; # proportion of cell from which it is considered " wet"
cut_off=0; # depth at which cell is considered as "wet"
dry_val=999; # value given to "dry" cells
```

3. Grid can then be generated:

```
depth=generate_grid(lon,lat,ref_dir,'etopo1','lim_wet,cut_off, dry_val)
```

4. Definition of boundaries:

```
lon_start=min(min(lon))-dx;
lon_end=max(max(lon))+dx;
lat_start=min(min(lat))-dy;
lat_end=max(max(lat))+dy;
```

(continues on next page)

(continued from previous page)

```
coord=[lat_start lon_start lat_end lon_end];
[b,n]=compute_boundary(coord,bound,1);
```

5. Mask generation (use of bathy and coastline):

```
m=ones(size(depth));
m(depth==dry_val)=0;
b_split=split_boundary(b,5*max([dx dy])); # splitting to make computation more␣
↪efficient
lim_wet=0.5;
offset=max([dx,dy]);
# mask cleaning remove lonely wet cells close to the coastline:
m2=clean_mask(lon,lat,m,b_split,lim_wet,offset);
cell_limit=-1 ; # if this value is negative all water bodies except the larger␣
↪are considered dry (\ie remove all lakes or closed seas), if positive: has␣
↪to be the minimum number of cells to consider a body as water
glob=0 ; # if global or not
[m4,mask_map]=remove_lake(m2,cell_limit,glob);
```

6. To make a grid from another model grid:
   - read bathymetry and mask from your model file
   - write the bathymetry thanks to write ww3file function, note that WW3 is expecting negative depth in the ocean:

```
write_ww3file([data_dir,'/','bottomm2','.inp'],depth'.*(-1));
```

   - build the mask for WW3: mask=1 is water, mask=0 is for points which won't be computed, mask=2 for active boundary points
   - write the mask file:

```
write_ww3file([data_dir,'/','mapsta','.inp'],mm');
```

Alternative

Alternatively, you can build the grid input files from a CROCO grid file. A script is provided in `Coupling_tools/WW3`: `make_ww3_grd_input_files_from_croco_grd.m`

Warning: Do not put the mask to 0 all around your domain, it will create problems in OASIS interpolations. You can either set 1 for sea points or 2 for boundary points.

Wind, current, and water level forcings

Eventually, wind, current, and water level forcing files with a valid time axis have to be prepared (if you need them as forcing for your WW3 run, not requested in full ocean-wave-atmosphere coupled mode).

A few scripts for preparing ww3 forcing files from CROCO (current and water level, WRF (wind) and CFSR (wind) files already processed through `Process_CFSR_files_for_CROCO.sh` are provided in `croco_tools/Coupling_tools/WW3`:

- `script_make_CROCO_current_and_level_for_ww3.sh`
- `script_make_WRF_wind_for_ww3.sh`
- `script_make_CFSR_wind_for_ww3.sh`

WW3 routines are named ww3_ROUTINENAME and take as input file by default: ww3_ROUTINENAME.inp. You have to set parameters in these .inp input files before running.

Steps for WW3 pre-processing are:

```
./ww3_grid # To prepare the grid and run (NB: timesteps are defined in ww3_grid.
↪inp file)
./ww3_prnc # To prepare wind forcing if you want to use one (not mandatory)
./ww3_strt # To prepare initialisation (not mandatory, will take defalut rest␣
↪state if not runned)
./ww3_bounc # To prepare spectral boundary conditions (not mandatory, will take␣
↪initial state as boundary conditions if not runned)
```

These steps will be performed automatically by the coupling scripts, when you submit the job.

Note: Note on mask/mapsta and bathy in WW3: The input map status (MAPSTA) value in the mask file can be :

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

The final possible values of the output map status MAPSTA are :

- -5 : other disabled point
- -4 : point masked in the two-way nesting
- -3 : dry point covered by ice
- -2 : dry point, not covered by ice
- -1 : wet point covered by ice
- 0 : land point
- 1 : active sea point
- 2 : active boundary point
- 8 : excluded sea/ice point
- 7 : excluded sea point, considered iced
- 15 : excluded sea point, considered dried: can become wet
- 31 : excluded sea point, inferred in nesting
- 63 : excluded sea point, masked in 2-way nesting

Coastline limiting depth (m, negative in the ocean) defined in ww3 grid.inp will also affect your MAPSTA: points with depth values above this coastline limit will be transformed to land points and therefore considered as excluded points (never become wet points, even if the water level grows over). In the output of the model, the depth (`dpt`) is described as : `DEPTH = LEV - BATHY`, in which the bathy is negative in the sea and positive on land, so the depth will be positive in the sea and a fillvalue on land. When the input water level (LEV) increases, it increases the output depth (`DPT`) value. The input water level forcing value is stored in WLV output variable, thus it gives the possibility to retrieve the input bathy value at each grid point : `BATHY = WLV - DPT`.

WRF preprocessing

WRF pre-processing system is WPS.

Warning: It should be downloaded in the same version than WRF.

Instructions, and scripts are provided in `~/croco/croco_tools/Coupling_tools/WRF_WPS`. You can follow the instructions given in `readme_wps`, and use the provided scripts:

`run_wps.bash`, `job.wps.*`

Note: you will need to have WPS compiled before (see preivous compilation tuto).

Running WPS

WRF pre-preocessing with WPS contains 3 steps:

- `geogrid`: defining the horizontal domain and interpolating geographical static data
- `ungrib`: decoding Grib meteorological data from reananlyses (or so)
- `metgrid`: interpolating meteorological data on the model grid

To run WPS, you therefore need:

- Geographical data

  Geographical data for WRF are available on WRF users website http://www2.mmm.ucar.edu/wrf/users/download/get_source.html. Geographical data will be available following the link "here" under WPS download section. You can download the full complete set, but note that topo files are not all in it. Download them individually in addition (e.g. topo_30s). Note that Geographical data file is a VERY LARGE file (49 Go uncompressed). Uncompress them (tar xvjf or tar -zxvf).

- Reanalysis data in grib format (from CFSR for example) to build the boundary and initial conditions

  For example, CFSR data can be downloaded from: https://rda.ucar.edu/datasets/ds093.0/index.html#!description

  A dedicated readme for CFSR data download is provided in `croco_tools/Coupling_tools/WRF_WPS`.

  You can use `g1print.exe` or `g2print.exe` (depending on you grib data format) available in `WPS/ungrib/` to check the variables in your data files. Usage is:

  ```
  ./g2print.exe YOURDATAFILE
  ```

- Vtable to read the grib data: exsiting Vtables can be found in WPS source directory under `WPS/ungrib/Variable_Tables`, and informations to choose Vtables can be found here: http://www2.mmm.ucar.edu/wrf/users/download/free_data.html

  Note: For CFSR, you will need 2 Vtables: one for the fields on pressure levels, one for the fields on surface level. Both Vtables are available in `croco_tools/Coupling_tools/WRF_WPS` directory:

  ```
  Vtable.CFSR_press_pgbh06
  Vtable.CFSR_sfc_flxf06
  ```

  `ungrib` therefore needs to be run twice (once for each type). This is done in `run_wps.bash` (see below).

A few scripts have been made to help you run WPS. You can find them in your croco_tools/Coupling_tools/WRF_WPS directory:

- configure.namelist.wps
- run_wps.bash
- job.wps.*

1. You should find them in YOURCONFIG/PREPRO/WRF_WPS. Edit all the required lines in configure.namelist.wps, and edit all the required paths in run_wps.bash
2. Run WPS directly (or using job.wps.pbs if you need to submit it in batch):

```
./run_wps.bash configure.namelist.wps NBPROCS >& run_wps.log
```

   If WPS is successful, you will obtain in ~/CONFIGS/BENGUELA_cpl/WRF_FILES/WPS_DATA:

```
geo_em.d01.nc
geo_em.d02.nc
met_em.d01.....nc # numerous files where '...' are dates
met_em.d02.....nc # numerous files where '...' are dates
```

3. Check your metgrid files by looking at some variables with ncview (*e.g.* LANDMASK, PSFC, PSML, SKINTEMP, TT ...)

   If some variables are missing, it is probably because you did not process ungrib and metgrid for all your input data.

   If something appears weird, it may be due to a bad interpolation (for example due to a too coarse land-sea mask in the original data). If so, re-run WPS with an updated METGRID.TBL

Running real.exe

After running WPS pre-processing, you need to run real.exe program which actually creates WRF input files for realistic cases from WPS generated files.

Warning: You need to use real.exe from uncoupled compilation even for a coupled run

A script has been made to help you run real.exe: run_real.bash. You can find it in your ~/CONFIGS/BENGUELA_cpl/WRF_IN directory or in the croco/SCRIPTS/SCRIPTS_COUPLING/WRF_IN. It also uses:

- configure.namelist.wps
- namelist.input.base.complete

1. Edit user settings in run_real.bash: paths, MPI settings...
2. Eventually edit namelist.input.base.complete with you choice of parameterization. *DO NOT EDIT* the stuff placed into brackets: <...>, it will be replaced by run_real.bash with appropriate values.

   Warning: For coupling with waves and currents, only YSU surface and boundary layer schemes are possible at the moment. Be sure to select these.

3. Run run_real.bash (eventually using a batch job as job.real.pbs):

```
./run_real.bash configure.namelist.wps NBPROCS >& run_real.log
```

   If real is successful, you will obtain in ~/CONFIGS/BENGUELA_cpl/WRF_FILES/YYYY:

```
wrfinput_d01_DATE
wrfbdy_d01_DATE
wrflowinp_d01_DATE # if sst_update is set to 1
wrfdda_d01_DATE # if nudging is activated i
wrf*_d02_DATE # if you have 2 domains
```

Additional pre-processing for coupled runs

In addition to traditional WRF pre-processing, you will need to:

- edit options in `namelist.input`:
  - in `&physics`: isftcflx = 5 if your are coupling with a wave model
  - in `&physics`: sst_update = 1 if your are coupling with an ocean model
  - in `&domains`: num_ext_model_couple_dom = X : number of domains of the other model you are coupling to WRF
- edit `CPLMASK` variable in `wrfinput_d0X` for all your coupled domains:
  - CPLMASK=1 where you want to couple
  - CPLMASK=0 when you do no want to couple
- you may need to create a WRF grid file for OASIS, if you are using the distributed version of WRF (at the date of 2021-Nov). If you are using the github WRF-CROCO version, you don't need to create this grid file, it will be created automatically. If necessary, a script is provided in `croco/SCRIPTS/SCRIPTS_COUPLING/SCRIPTS_TOOLBOX/ROUTINES/OASIS_SCRIPTS`:

  Edit and run `create_oasis_grids_for_wrf.sh`

Note that the CPLMASK creation may also be performed automatically in the coupling tools.

OASIS pre-processing

In `oasis_in` you have several scripts to help you prepare:

- WRF grid files for OASIS: `create_oasis_grids_for_wrf.sh`
- and eventually create oasis restart files from preexisting model outputs: `create_oasis_restart_from_preexisting_output_files.sh`

This step is also performed automatically by the coupling tools when launching the run with submitjob.sh. .

20.4.5 Running in COUPLED mode

To run models in coupled mode, you need to have completed the compilation and the preprocessing phases for each model. Then choose the case you desire in the list below

CROCO-TOY (wav or atm)

CROCO-WRF

CROCO-WW3

CROCO-WRF-WW3

In this case you should have in your $CHOME repository:

- `myenv_mypath.sh`
- `mynamelist.sh`
- `myjob.sh`

- CROCO_IN
- TOY_IN
- OASIS_IN
- SCRIPTS_TOOLBOX

myenv_mypath.sh should already have been filled in before the compilation. In TOY_IN, you must have the executable toy_model

To make the run you need to modify the files myjob.sh and mynamelist.sh.

- In myjob.sh , you will have to fill in information about jobs:

```
# Real job duration in sec (converted to MACHINE format in submit job)
export TIMEJOB=1800

#------------------------------------------------------------------------------
↪-
# Run date settings
#------------------------------------------------------------------------------
↪-
# Your run can be divided into several jobs (e.g.: 1 year run into 12 jobs of␣
↪1 month)

# Start date of the first Job
export YEAR_BEGIN_JOB=2005
export MONTH_BEGIN_JOB=1
export DAY_BEGIN_JOB=1

# Duration of each Job
export JOB_DUR_MTH=1
export JOB_DUR_DAY=0

# How many jobs do you want to launch?
export NBJOB=1

# Do we start from a restart?
export RESTART_FLAG="FALSE"
```

Along with the number of cpu you will use for each model:

```
# nb of CPUs for each model
export NP_OCEX=2
export NP_OCEY=2
export NP_TOY=2
```

There are other more advanced options, but we will not focus on them for now.

- In mynamelist.sh, specify the name of the experiment, the run type (frc, oa, ow, owa), and which models are used. From here, we will consider the toy model as an atmospheric model:

```
export CEXPER=BENGUELA_oa_toyatm
export RUNtype=oa
#
export USE_OCE=1
export USE_TOYATM=1
export USE_TOYOCE=0
export USE_TOYWAV=0
#
```

Set the exe path (for croco it is usually CROCO_IN, corresponding to OCE_NAM_DIR in myenv_mypath.sh):

```
#-------------------------------------------------------------------------------
# Exe paths
#-------------------------------------------------------------------------------
export OCE_EXE_DIR="${CHOME}/CROCO_IN"
export TOY_EXE_DIR="${CHOME}/TOY_IN"
```

Then edit the model setting:

```
#-------------------------------------------------------------------------------
# CPL
#-------------------------------------------------------------------------------
# namelist
export namcouplename=namcouple.base.${RUNtype}${istoy}

# coupling frequency
export CPL_FREQ=21600

#-------------------------------------------------------------------------------
# OCE
#-------------------------------------------------------------------------------
# namelist [Info: grid size is directly read in oce_compile.sh and cpl_nam.sh ]

# Online Compilation
export ONLINE_COMP=1

# Time steps
export TSP_OCE=800
export TSP_OCEF=60

# Parameter
export hmin=75; # minimum water depth in CROCO, delimiting coastline in WW3

# domains
export AGRIFZ=0
export AGRIF_2WAY="FALSE"

# forcing files
export ini_ext='ini_SODA' # ini extension file (ini_SODA,...)
export bdy_ext='bry_SODA' # bry extension file (bry_SODA,...)
export surfrc_flag="FALSE" # Flag if surface forcing is needed (FALSE if cpl)
export interponline=0 # switch (1=on, 0=off) for online surface interpolation
export frc_ext='blk_CFSR' # surface forcing extension(blk_CFSR, frc_CFSR,...). If␣
↪interponline=1 just precise the type (ECMWF, CFSR,AROME,...)
export tide_flag="FALSE" # the forcing extension must be blk_??? otherwise tide␣
↪forcing overwrites it

# output settings
#!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
↪!!!!!!!
#                                          WARNING                                 ␣
↪      !
# When XIOS is activated the following values (for the model) are not taken into␣
↪account !
#!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
↪!!!!!!!
export oce_nhis=27     # history output interval (in number of timesteps)
export oce_navg=27     # average output interval (in number of timesteps)
```

Since the toy model mimics an atmospheric model, put `"atm"` in the list of models. You must also provide the input file you put in `toy_files`:

```
#-------------------------------------------------------------------------------
# TOY
#-------------------------------------------------------------------------------
# type
export toytype=("atm") #oce,atm,wav

# forcing files
export toyfile=("$CWORK/TOY_FILES/wrfout_d01_20050101_20050131.nc")
export timerange=('2,125')
```

Now that you have completed the necessary files, you are ready to run your simulation. To do so, simply do:

```
./submitjob.sh
```

In your ${CHOME} directory, you should have already filled in `myenv_mypath.sh`.

To make the run, you need to modify the files `myjob.sh` and `mynamelist.sh`.

- In `myjob.sh` , you will have to fill in information about jobs:

```
# Real job duration in sec (converted to MACHINE format in submit job)
export TIMEJOB=1800

#-------------------------------------------------------------------------------
↪-
# Run date settings
#-------------------------------------------------------------------------------
↪-
# Your run can be divided into several jobs (e.g.: 1 year run into 12 jobs of␣
↪1 month)

# Start date of the first Job
export YEAR_BEGIN_JOB=2005
export MONTH_BEGIN_JOB=1
export DAY_BEGIN_JOB=1

# Duration of each Job
export JOB_DUR_MTH=1
export JOB_DUR_DAY=0

# How many jobs do you want to launch?
export NBJOB=1

# Do we start from a restart?
export RESTART_FLAG="FALSE"
```

Along with the number of cpu you will use for each model:

```
# nb of CPUs for each model
export NP_OCEX=2
export NP_OCEY=2
export NP_ATM=12

# additional MPI Settings for ATM (WRF)
export atm_nprocX=-1       # -1 for automatic settings
export atm_nprocY=-1       # -1 for automatic settings
export atm_niotaskpg=0     # 0 for default settings
export atm_niogp=1         # 1 for default settings
```

There are other more advanced options, but we will not focus on them for now.

- In `mynamelist.sh`, specify the name of the experiment, the run type (frc, oa, ow, owa), and which models are used.:

```
#
export CEXPER=BENGUELA_oa
export RUNtype=oa
#
export USE_ATM=1
export USE_OCE=1
```

Set the exe path (for croco it is usually CROCO_IN, corresponding to OCE_NAM_DIR in myenv_mypath.sh):

```
#-------------------------------------------------------------------------------
# Exe paths
# ------------------------------------------------------------------------------
export OCE_EXE_DIR="${CHOME}/CROCO_IN"
export ATM_EXE_DIR="${ATM}/exe_coupled"
```

Then edit the model setting:

```
#-------------------------------------------------------------------------------
# CPL
#-------------------------------------------------------------------------------
# namelist
export namcouplename=namcouple.base.${RUNtype}${istoy}

# coupling frequency
export CPL_FREQ=21600

#-------------------------------------------------------------------------------
# OCE
#-------------------------------------------------------------------------------
# namelist [Info: grid size is directly read in oce_compile.sh and cpl_nam.sh ]

# Online Compilation
export ONLINE_COMP=1

# Time steps
export TSP_OCE=800
export TSP_OCEF=60

# Parameter
export hmin=75; # minimum water depth in CROCO, delimiting coastline in WW3

# domains
export AGRIFZ=0
export AGRIF_2WAY="FALSE"

# forcing files
export ini_ext='ini_SODA' # ini extension file (ini_SODA,...)
export bdy_ext='bry_SODA' # bry extension file (bry_SODA,...)
export surfrc_flag="FALSE" # Flag if surface forcing is needed (FALSE if cpl)
export interponline=0 # switch (1=on, 0=off) for online surface interpolation
export frc_ext='blk_CFSR' # surface forcing extension(blk_CFSR, frc_CFSR,...). If␣
↪interponline=1 just precise the type (ECMWF, CFSR,AROME,...)
export tide_flag="FALSE" # the forcing extension must be blk_??? otherwise tide␣
↪forcing overwrites it

# output settings
#!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
↪!!!!!!!
#                                        WARNING                                 ␣
↪      !
# When XIOS is activated the following values (for the model) are not taken into␣
↪account !
```

(continues on next page)

(continued from previous page)

```
#!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
↪!!!!!!!
export oce_nhis=27     # history output interval (in number of timesteps)
export oce_navg=27     # average output interval (in number of timesteps)

#-------------------------------------------------------------------------------
# ATM
#-------------------------------------------------------------------------------
# namelist
export atmnamelist=namelist.input.base.complete

# Time steps
export DT_ATM=150

# Grid size
#[ Grid size should be already put in the namelist. When coupled it is directly␣
↪read in cpl_nam.sh ]

# domains
export NB_dom=1 # Number of coupled domains
export wrfcpldom='d01'

# Boundaries interval
export interval_seconds=21600 # interval ( in sec ) of the latteral input
export auxinput4_interval=360 # interval ( in min ) of bottom input

# output settings
#!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
↪!!!!!!!
#                                          WARNING                                ␣
↪       !
# When XIOS is activated the following values (for the model) are not taken into␣
↪account !
#!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
↪!!!!!!!
export atm_his_h=6                        # output interval (h)
export atm_his_frames=1000 # $((31*24))          # nb of outputs per file
export atm_diag_int_m=$((${atm_his_h}*60))  # diag output interval (m)
export atm_diag_frames=1000     # nb of diag outputs per file
```

Now that you have completed the necessary files, you are ready to run your simulation. To do so, simply do:

```
./submitjob.sh
```

In your ${CHOME} repository you should have already filled in `myenv_mypath.sh`.

To make the run, you need to modify the files `myjob.sh` and `mynamelist.sh`.

- In `myjob.sh` , you will have to fill in information about jobs:

```
# Real job duration in sec (converted to MACHINE format in submit job)
export TIMEJOB=1800

#-------------------------------------------------------------------------------
↪-
# Run date settings
#-------------------------------------------------------------------------------
↪-
# Your run can be divided into several jobs (e.g.: 1 year run into 12 jobs of␣
↪1 month)

# Start date of the first Job
```

(continues on next page)

(continued from previous page)

```
export YEAR_BEGIN_JOB=2005
export MONTH_BEGIN_JOB=1
export DAY_BEGIN_JOB=1

# Duration of each Job
export JOB_DUR_MTH=1
export JOB_DUR_DAY=0

# How many jobs do you want to launch?
export NBJOB=1

# Do we start from a restart?
export RESTART_FLAG="FALSE"
```

Along with the number of cpu you will use for each model:

```
# nb of CPUs for each model
export NP_OCEX=2
export NP_OCEY=2
export NP_WAV=14
```

There are other more advanced options, but we will not focus on them for now.

- In `mynamelist.sh`, specify the name of the experiment, the run type (frc, oa, ow, owa), and which models are used.:

```
#
export CEXPER=BENGUELA_ow
export RUNtype=ow
#
export USE_OCE=1
export USE_WAV=1
```

Set the exe path (for croco it is usually CROCO_IN, corresponding to OCE_NAM_DIR in myenv_mypath.sh):

```
#-------------------------------------------------------------------------------
# Exe paths
# ------------------------------------------------------------------------------
export OCE_EXE_DIR="${CHOME}/CROCO_IN"
export WAV_EXE_DIR="${WAV}/exe_ow_BENGUELA"
```

Then edit the model setting:

```
#-------------------------------------------------------------------------------
# CPL
#-------------------------------------------------------------------------------
# namelist
export namcouplename=namcouple.base.${RUNtype}${istoy}

# coupling frequency
export CPL_FREQ=21600

#-------------------------------------------------------------------------------
# OCE
#-------------------------------------------------------------------------------
# namelist [Info: grid size is directly read in oce_compile.sh and cpl_nam.sh ]

# Online Compilation
export ONLINE_COMP=1

# Time steps
```

(continues on next page)

(continued from previous page)

```
export TSP_OCE=800
export TSP_OCEF=60

# Parameter
export hmin=75; # minimum water depth in CROCO, delimiting coastline in WW3

# domains
export AGRIFZ=0
export AGRIF_2WAY="FALSE"

# forcing files
export ini_ext='ini_SODA' # ini extension file (ini_SODA,...)
export bdy_ext='bry_SODA' # bry extension file (bry_SODA,...)
export surfrc_flag="TRUE" # Flag if surface forcing is needed (FALSE if cpl)
export interponline=0 # switch (1=on, 0=off) for online surface interpolation
export frc_ext='blk_CFSR' # surface forcing extension(blk_CFSR, frc_CFSR,...). If␣
↪interponline=1 just precise the type (ECMWF, CFSR,AROME,...)
export tide_flag="FALSE" # the forcing extension must be blk_??? otherwise tide␣
↪forcing overwrites it

# output settings
#!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
↪!!!!!!!
#                                          WARNING                                  ␣
↪       !
# When XIOS is activated the following values (for the model) are not taken into␣
↪account !
#!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
↪!!!!!!!
export oce_nhis=27     # history output interval (in number of timesteps)
export oce_navg=27     # average output interval (in number of timesteps)

#---------------------------------------------------------------------------------
# WAV
#---------------------------------------------------------------------------------
# namelist

# Time steps
export DT_WAV=3600     # TMAX = 3*TCFL
export DT_WW_PRO=1200  # TCFL --> ww3.grid to see the definition
export DT_WW_REF=1800  # TMAX / 2
export DT_WW_SRC=10

# Grid size
export wavnx=41 ; export wavny=42

# forcing files
export forcin=() # forcing file(s) list (leave empty if none)
export forcww3=() # name of ww3_prnc.inp extension/input file

# output settings
export flagout="TRUE" # Keep (TRUE) or not (FALSE) ww3 full output binary file␣
↪(out_grd.ww3)
export wav_int=21600            # output interval (s)
# ww3 file to be used for creating restart file for oasis
export wavfile=$CWORK/outputs_frc_ww3_CFSR/ww3.200501.nc # Usually done by running␣
↪a frc mode on the area
```

Now that you have completed the necessary files, you are ready to run your simulation. To do so, simply do:

```
./submitjob.sh
```

In your ${CHOME} repository you should have already filled in `myenv_mypath.sh`.

To make the run, you need to modify the files `myjob.sh` and `mynamelist.sh`.

- In `myjob.sh` , you will have to fill in information about jobs:

```
# Real job duration in sec (converted to MACHINE format in submit job)
export TIMEJOB=1800

#------------------------------------------------------------------------------
# Run date settings
#------------------------------------------------------------------------------
# Your run can be divided into several jobs (e.g.: 1 year run into 12 jobs of 1 month)

# Start date of the first Job
export YEAR_BEGIN_JOB=2005
export MONTH_BEGIN_JOB=1
export DAY_BEGIN_JOB=1

# Duration of each Job
export JOB_DUR_MTH=1
export JOB_DUR_DAY=0

# How many jobs do you want to launch?
export NBJOB=1

# Do we start from a restart?
export RESTART_FLAG="FALSE"
```

Along with the number of cpu you will use for each model:

```
# nb of CPUs for each model
export NP_OCEX=2
export NP_OCEY=2
export NP_WAV=14
export NP_ATM=12

# additional MPI Settings for ATM (WRF)
export atm_nprocX=-1       # -1 for automatic settings
export atm_nprocY=-1       # -1 for automatic settings
export atm_niotaskpg=0     # 0 for default settings
export atm_niogp=1         # 1 for default settings
```

There are other more advanced options, but we will not focus on them for now.

- In `mynamelist.sh`, specify the name of the experiment, the run type (frc, oa, ow, owa), and which models are used.:

```
#
export CEXPER=BENGUELA_owa
export RUNtype=owa
#
export USE_ATM=1
export USE_OCE=1
export USE_WAV=1
```

Set the exe path (for croco it is usually CROCO_IN, corresponding to OCE_NAM_DIR in myenv_mypath.sh):

```
#------------------------------------------------------------------------------
# Exe paths
```

(continues on next page)

(continued from previous page)

```
# ------------------------------------------------------------------------------
export OCE_EXE_DIR="${CHOME}/CROCO_IN"
export WAV_EXE_DIR="${WAV}/exe_owa_BENGUELA"
export ATM_EXE_DIR="${ATM}/exe_coupled"
```

Then edit the model setting. If WW3 grid was created using `make_ww3_grd_input_files_from_croco_grd.m`, wavx (respectively wavy) is exactly the values of xi_rho (respectively eta_rho) in the croco_grd.nc file used:

```
#------------------------------------------------------------------------------
# CPL
#------------------------------------------------------------------------------
# namelist
export namcouplename=namcouple.base.${RUNtype}${istoy}

# coupling frequency
export CPL_FREQ=21600

#------------------------------------------------------------------------------
# OCE
#------------------------------------------------------------------------------
# namelist [Info: grid size is directly read in oce_compile.sh and cpl_nam.sh ]

# Online Compilation
export ONLINE_COMP=1

# Time steps
export TSP_OCE=800
export TSP_OCEF=60

# Parameter
export hmin=75; # minimum water depth in CROCO, delimiting coastline in WW3

# domains
export AGRIFZ=0
export AGRIF_2WAY="FALSE"

# forcing files
export ini_ext='ini_SODA' # ini extension file (ini_SODA,...)
export bdy_ext='bry_SODA' # bry extension file (bry_SODA,...)
export surfrc_flag="FALSE" # Flag if surface forcing is needed (FALSE if cpl)
export interponline=0 # switch (1=on, 0=off) for online surface interpolation
export frc_ext='blk_CFSR' # surface forcing extension(blk_CFSR, frc_CFSR,...). If␣
↪interponline=1 just precise the type (ECMWF, CFSR,AROME,...)
export tide_flag="FALSE" # the forcing extension must be blk_??? otherwise tide␣
↪forcing overwrites it

# output settings
#!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
↪!!!!!!!
#                                        WARNING                                  ␣
↪      !
# When XIOS is activated the following values (for the model) are not taken into␣
↪account !
#!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
↪!!!!!!!
export oce_nhis=27     # history output interval (in number of timesteps)
export oce_navg=27     # average output interval (in number of timesteps)

#------------------------------------------------------------------------------
# WAV
#------------------------------------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
# namelist

# Time steps
export DT_WAV=3600      # TMAX = 3*TCFL
export DT_WW_PRO=1200   # TCFL --> ww3.grid to see the definition
export DT_WW_REF=1800   # TMAX / 2
export DT_WW_SRC=10

# Grid size
export wavnx=41 ; export wavny=42

# forcing files
export forcin=() # forcing file(s) list (leave empty if none)
export forcww3=() # name of ww3_prnc.inp extension/input file

# output settings
export flagout="TRUE" # Keep (TRUE) or not (FALSE) ww3 full output binary file␣
↪(out_grd.ww3)
export wav_int=21600            # output interval (s)
# ww3 file to be used for creating restart file for oasis
export wavfile=$CWORK/outputs_frc_ww3_CFSR/ww3.200501.nc # Usually done by running␣
↪a frc mode on the area

#------------------------------------------------------------------------------
# ATM
#------------------------------------------------------------------------------
# namelist
export atmnamelist=namelist.input.base.complete

# Time steps
export DT_ATM=150

# Grid size
#[ Grid size should be already put in the namelist. When coupled it is directly␣
↪read in cpl_nam.sh ]

# domains
export NB_dom=1 # Number of coupled domains
export wrfcpldom='d01'

# Boundaries interval
export interval_seconds=21600 # interval ( in sec ) of the latteral input
export auxinput4_interval=360 # interval ( in min ) of bottom input

# output settings
#!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
↪!!!!!!!
#                                        WARNING                                  ␣
↪      !
# When XIOS is activated the following values (for the model) are not taken into␣
↪account !
#!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
↪!!!!!!!
export atm_his_h=6                          # output interval (h)
export atm_his_frames=1000 # $((31*24))           # nb of outputs per file
export atm_diag_int_m=$((${atm_his_h}*60))  # diag output interval (m)
export atm_diag_frames=1000     # nb of diag outputs per file
```

Now that you have completed the necessary files, you are ready to run your simulation. To do so, simply do:

```
./submitjob.sh
```

Handle the outputs/restarts

Once your job is launched, two repositories should have appeared:

- `${CHOME}/job_${CEXPER}`
- `${CWORK}/rundir`

`${CHOME}`, `${CWORK}` being variables you specified in myenv_mypath.sh. The first is where logs are put. The second is where your job is running and where outputs/restarts are stored:

- `${CEXPER}_execute`
- `${CEXPER}_output`
- `${CEXPER}_restart`

In those repositories, you will find one folder per job. Meaning if the simulation is 12 jobs long, you will have 12 folders.

Note: If one of your model "blow up" reduce the time_step ('TSP_*') in `mynamelist.sh`

CHAPTER

TWENTYONE

LITTORAL DYNAMICS TUTORIAL

Note: This configuration is based on rip_current test case

The aim of this tutorial is to investigate gradually the capability of CROCO to deal with the nearshore dynamics. It is built on some test-cases that are packaged within the CROCO release and will be thoroughly analysed. The various aspects that will be adressed are the following :

- Compute a test-case,
- Modify a test-case (including a new bathymetry, modifying the forcings, . . .),
- Use of the CROCO embedded WKB wave model,
- Parametrisation of the Bottom Boundary Layer combining wave and circulation,
- Account for the sediment compartment,
- Morphodynamics.

The tutorial is based on the rip_current, sandbar, shoreface, swash test cases. For a description of the wave-averaged equations and WKB wave model see wci.

Rip currents are strong, seaward flows forced by longshore variation of the wave-induced momentum flux. They are responsible for the recirculation of water accumulated at a beach by a weaker and broader shoreward flow due to Stokes drift.

Here, we consider longshore variation of the wave-induced momentum flux due to breaking at barred bottom topography with an imposed longshore perturbation, as in [YU2003] or [WEIR2011]. The basin is rectangular (768 m by 768 m) and the topography is constant over time and based on field surveys at Duck, North Carolina. Shore-normal, monochoromatic waves (1m, 10s) are imposed at the offshore boundary and propagate through the WKB wave model coupled with the 3D circulation model (Uchiyama et al., 2011). The domain is periodic in the alongshore direction. We assume that the nearshore boundary is reflectionless, and there is no net outflow at the offshore boundary.

The tutorial starts by implementing and running the rip_current test case. It can be activated with the cpp key RIP that can be followed throughout the source code to gather the main informations about the setup. The following figure picke up in [YU2003] shows what the bathymetry looks for.

Answer the basic following questions in order to charaterize the set up:

- what is that analytical formulation of the topography, the basin size, the resolution
- characterize the wave forcing
- what are the interaction between wave and currents
- what is the formulation of the drag coefficient

Related CPP options:

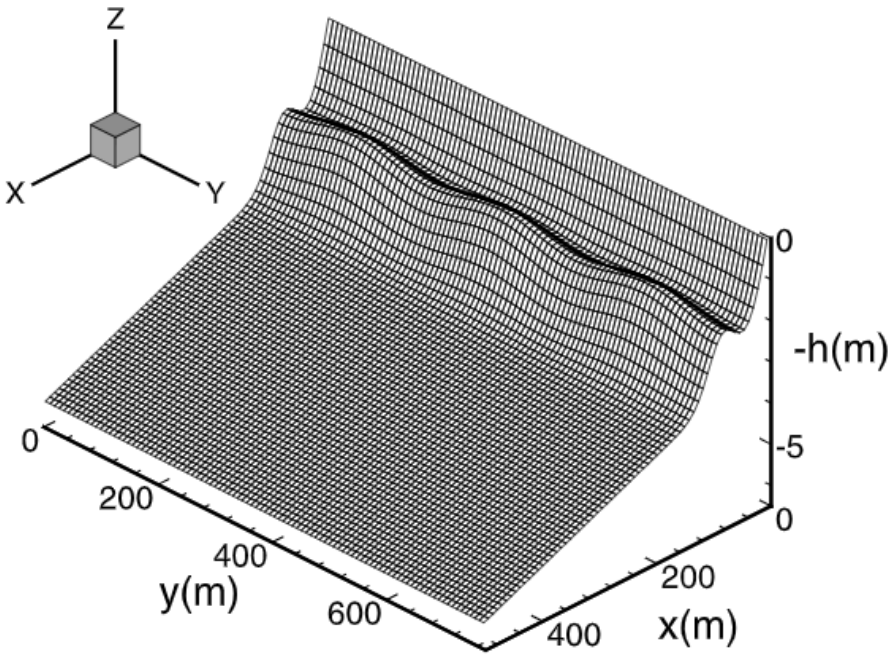


Figure 1. Beach topography with a longshore bar at $x = 80$ m and rip channels. $\epsilon = 0.1$ and $\lambda = 256$ m.

RIP	Idealized Duck Beach with 3D topography (default)
BISCA	Semi-realistic Biscarosse Beach (needs input files)
RIP_TOPO_2D	Idealized Duck with longshore uniform topography
GRANDPOPO	Idealized longshore uniform terraced beach (Grand Popo, Benin)
ANA_TIDES	Adds idealized tidal variations
WAVE_MAKER & NBQ	Wave resolving rather than wave-averaged case (#undef MRL_WCI)

CPP options:

```
# define RIP
```

```
# undef  OPENMP
# undef  MPI
# define SOLVE3D
# define NEW_S_COORD
# define UV_ADV
# define BSTRESS_FAST
# undef  NBQ
# ifdef NBQ
#  define NBQ_PRECISE
#  define WAVE_MAKER
#  define WAVE_MAKER_SPECTRUM
#  define WAVE_MAKER_DSPREAD
#  define UV_HADV_WENO5
#  define UV_VADV_WENO5
#  define W_HADV_WENO5
#  define W_VADV_WENO5
#  define GLS_MIXING_3D
#  undef  ANA_TIDES
#  undef  MRL_WCI
#  define OBC_SPECIFIED_WEST
```

(continues on next page)

(continued from previous page)

```
#  define FRC_BRY
#  define ANA_BRY
#  define Z_FRC_BRY
#  define M2_FRC_BRY
#  define M3_FRC_BRY
#  define T_FRC_BRY
#  define AVERAGES
#  define AVERAGES_K
# else
#  define UV_VIS2
#  define UV_VIS_SMAGO
#  define LMD_MIXING
#  define LMD_SKPP
#  define LMD_BKPP
#  define MRL_WCI
# endif
# define WET_DRY
# ifdef MRL_WCI
#  define WKB_WWAVE
#  define WKB_OBC_WEST
#  define WAVE_ROLLER
#  define WAVE_FRICTION
#  define WAVE_FRICTION
#  define WAVE_STREAMING
#  define MRL_CEW
#  ifdef RIP_TOPO_2D
#   define WAVE_RAMP
#  endif
# endif
# ifndef BISCA
#  define ANA_GRID
# endif
# define ANA_INITIAL
# define ANA_SMFLUX
# define ANA_STFLUX
# define ANA_SSFLUX
# define ANA_SRFLUX
# define ANA_SST
# define ANA_BTFLUX
# if !defined BISCA && !defined ANA_TIDES
#  define NS_PERIODIC
# else
#  define OBC_NORTH
#  define OBC_SOUTH
# endif
# define OBC_WEST
# define SPONGE
# ifdef ANA_TIDES
#  define ANA_SSH
#  define ANA_M2CLIMA
#  define ANA_M3CLIMA
#  define ZCLIMATOLOGY
#  define M2CLIMATOLOGY
#  define M3CLIMATOLOGY
#  define M2NUDGING
#  define M3NUDGING
# endif
# ifdef BISCA
#  define BBL
# endif
# undef SEDIMENT
```

(continues on next page)

(continued from previous page)

```
# ifdef SEDIMENT
#  define ANA_SEDIMENT
#  undef  ANA_SPFLUX
#  undef  ANA_BPFLUX
# endif
# undef  DIAGNOSTICS_UV
```

CHAPTER

TWENTYTWO

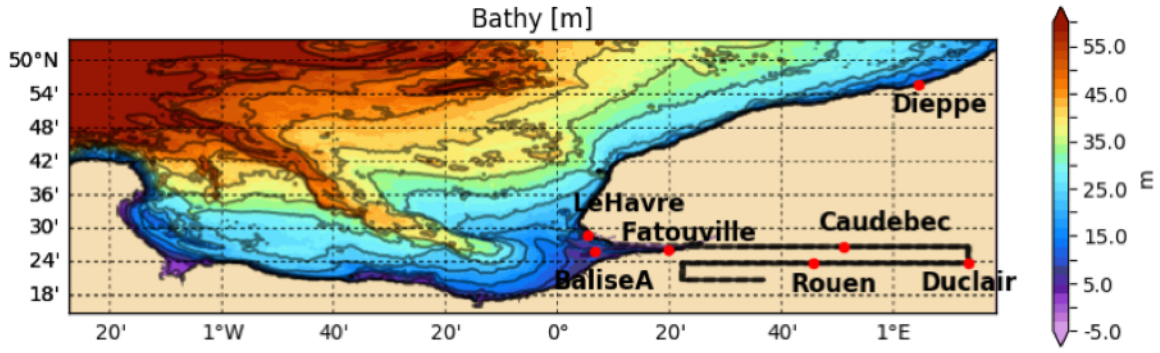
REALISTIC COASTAL CONFIGURATION

Warning: This part is only given as an example of coastal configuration (resolution ~500m) in macro tidal environment. So you have an example of which set of cpp keys to use

All inputs files (both source code, inputs text files and forcing netcdf files) can be found there :

ftp://ftp.ifremer.fr/ifremer/croco/BDS_EXAMPLE/BDS_CONFIG.tar.gz

Here we describe a coastal configuration in bay of Seine (dx,dy=500m) with realistic forcing fields :



- Grid : 481*181 and 20 vertical levels with bathymetry from SHOM (HOMONIM MNT)

Note: In shallow water conditions with Wetting and Drying meshes, vertical sigma coordinates should be equally spaced (no refinement on bottom to avoid too small layers). To do so, put critical depth **hc** in croco.in to a big value (let's say 1e16)

- Schematic river canal for the Seine river : The resolution of the meshes is locally refined in this canal to take into account a the section reduction based on 500m grid size
- Tidal forcing from PREVIMER atlas (700m resolution with both elevation and velocity)
  - 16 waves : M2 N2 S2 K2 2N2 O1 K1 Q1 P1 M4 MS4 MN4 MK4 M6
  - Harmonic composition with MAS (Simon-SHOM) method for elevation (no need to compute wave arguments in pre-processing)
- Realistic 3D open boundary conditions (T,S) from a 2.5km grid-mesh regional model
- Online atmospheric forcing from METEO-FRANCE
  - Bulk flux with Fairall formulation
  - Atm pressure gradient taken into account in the equations
  - Inverse barometer effect from atm pressure : correction added to ssh from harmonic composition at OBC

- River inputs : realistic time series for outflow,temperature and salinity of 9 sources
- Physics
  - Wetting and drying scheme
  - High order advection scheme WENO5 on both horizontal and vertical dimensions for momentum and tracers
  - Bartropic and baroclinic coupling with *M2_FILTER_NONE* option and with myalpha=0.3
  - Vertical turbulent fluxes from GLS (k-epsilon)

CHAPTER

TWENTYTHREE

XIOS

As a start point for this tutorial, we will use the BASIN test case (see section 5.1)

```
cd ~/CONFIGS/BASIN
```

Is everything ok ? Compiling ? Running ? Are the 2 files basin_rst.nc and basin_his.nc created ?

What is the walltime (or real time)?

Now add the XIOS functionnality in the croco executable:

If the XIOS is installed on your target machine (it is the case on Datarmor), there are only 2 new simple steps to follow :

1. Edit `cppdef.h`:

   Need to define 2 news cpp keys fot this test case:

```
/*
!                        Basin Example
!                        ===== =======
*/
# define XIOS
# undef  OPENMP
.....
```

2. Edit the compilation script `jobcomp`:

   Need to add the XIOS library path

```
#
# set XIOS directory if needed
#
XIOS_ROOT_DIR=$HOME/xios-2.5
#
```

   For this tutorial, we need to comment three lines (217, 218 and 219) in `jobcomp`:

```
#        $CPP1 -P -traditional -imacros cppdefs.h  ${ROOT_DIR}/XIOS/
↪field_def.xml_full $RUNDIR/field_def.xml
#        $CPP1 -P -traditional -imacros cppdefs.h  ${ROOT_DIR}/XIOS/
↪domain_def.xml $RUNDIR/domain_def.xml
#        $CPP1 -P -traditional -imacros cppdefs.h  ${ROOT_DIR}/XIOS/
↪iodef.xml $RUNDIR/iodef.xml
```

   For this tutorial, we need to modify the routine `send_xios_diags.F` :

```
cp ~/croco/croco/XIOS/send_xios_diags.F .
```

   Edit send_xios_diags.F and comment lines 2077, 2078 and 2133:

```
!      call xios_send_field("uwnd",uwnd)
!      call xios_send_field("vwnd",vwnd)

!      call xios_send_field("bvf",bvf)
```

Compile the model again:

```
./jobcomp
```

Before running the model with XIOS module, we need three xml files (field_def.xml, domain_def.xml and iodef.xml) :

```
cp /home/datawork-croco/datarmor-only/CONFIGS/TUTO20/BASIN_WITH_XIOS/
↪field_def.xml .
cp /home/datawork-croco/datarmor-only/CONFIGS/TUTO20/BASIN_WITH_XIOS/
↪domain_def.xml .
cp /home/datawork-croco/datarmor-only/CONFIGS/TUTO20/BASIN_WITH_XIOS/
↪iodef.xml.OneFile iodef.xml
```

Have a look at iodef.xml file :

- selected fields to output,
- output frequency `output_freq`,
- what kind of output (instantaneous, average) `operation`,
- ...

At the end of the iodef.xml file, look at the line

```
<variable id="using_server" type="bool">false</variable>
```

The boolean `false` means that croco will run with XIOS in "attached mode". **Each computing processor will write in the output file**. In this "attached mode", XIOS behaves like a netcdf4 layer.

In this `iodef.xml` file, the configuration for outputs is the same as in croco.in file.

Run the model in "attached mode":

```
qsub job_croco_mpi.pbs
```

Compare the new file `Basin_Example_10d_inst_0001-01-01-0001-04-30.nc` with the previous one `basin_his.nc` :

```
ncview basin_his.nc & ; ncview Basin_Example_10d_inst_0001-01-01-0001-04-
↪30.nc
```

Note: If your output file start with a `?`, it is due to a tab before the configuration title in croco.in: `Basin Example`. Just replace the tab by a blank space.

—> For large configuration, XIOS is very efficient in netcdf parallel writting.

Edit iodef.xml file and add new 2D and 3D fields to be written in the output file by uncommenting lines :

```
<field field_ref="w" name="w" />
<field field_ref="salt" name="salt" />
<field field_ref="sustr" name="sustr" />
<field field_ref="svstr" name="svstr" />
<field field_ref="rho" name="rho" />
```

Run the model:

```
qsub job_croco_mpi.pbs
```

Have a look at the new file `Basin_Example_10d_inst_0001-01-01-0001-04-30.nc`

Add an extra file for average output in editing iodef.xml (or you can get an example there):

```
cp /home/datawork-croco/datarmor-only/CONFIGS/TUTO20/BASIN_WITH_XIOS/
↪iodef.xml.Twofiles iodef.xml
```

Have a look at the iodef.xml file to understand how to simply add a new output file

run the model:

```
qsub job_croco_mpi.pbs
```

Have a look at the new netcdf file `Basin_Example_5d_aver_0001-01-01-0001-04-30.nc`

What is the walltime (or real time)?

Run the model in "detached mode":

edit iodef.xml and modify boolean at ``true`` in line:

```
<variable id="using_server" type="bool">true</variable>
```

The boolean `true` means that croco will run with XIOS in "detached mode". **Each computing processor will send fields to one or several XIOS servers which will be in charge of writing the outputs files**.

Edit job_croco_mpi.pbs to add one XIOS server

```
##PBS -l select=1:ncpus=28:mpiprocs=4:mem=8g
#PBS -l select=1:ncpus=28:mpiprocs=5:mem=8g

#time $MPI_LAUNCH croco croco.in >& croco.out
time $MPI_LAUNCH -n 4 croco croco.in : -n 1 xios_server.exe  >& croco.out
```

There will be 4 computing processors sending fields to 1 xios server writting in output files.

Run the model:

```
qsub job_croco_mpi.pbs
```

Theorically, computing processors will run faster (**keep in mind that reading and writting files is slow, computing is fast!**).

What is the walltime (or real time)?

Is it worth to use detached mode in this case?

Adding an online dignostic using ONLY xios:

In the output file, we need to have a new variable computed from already defined variables. For instance, we want to have zeta*zeta ...

Edit `field_def.xml` and add the new variable zeta2:

```
<field id="zeta" long_name="free-surface" unit="meter" />
<field id="zeta2" long_name="squared free-surface" unit="meter2" >␣
↪(zeta*zeta) </field>
```

Then edit `iodef.xml` and add the new variable to be written in the output file:

```
<field_group id="inst_fields" operation="instant">
<field field_ref="zeta" name="zeta" />
<field field_ref="zeta2" name="zeta2" />
```

No need to compile, just run the model:

```
qsub job_croco_mpi.pbs
```

If you have time, add xios in the previous BENGUELA_LR

```
cd $confs/Run_BENGUELA_LR
cp /home/datawork-croco/datarmor-only/CONFIGS/TUTO20/BENGUELA_LR_XIOS/* .
```

Compile once:

```
./jopbcomp
```

Run :

```
qsub job_croco_mpi.pbs
```

Explore files, edit and modify iodef.xml, and run again . . .

CHAPTER

TWENTYFOUR

TIPS

CHAPTER

TWENTYFIVE

CROCO/MUSTANG TUTORIAL & TIPS

25.1 Get to know the CROCO/MUSTANG coupling

Read the documentation on CROCO/MUSTANG (CROCO_MUSTANG)

Note: The MUSTANG learning curve is a steep one. Understanding the documentation strongly benefits from reading the code itself.

Note: In this tutorial $croco refers to the main directory of CROCO source code

25.2 Run a test case

- Choose a test case (sediment)
- Copy the various configuration files that you need

```
cp -r $croco/MUSTANG/MUSTANG_NAMELIST/ ./MUSTANG_NAMELIST
cp -r $croco/TEST_CASES/ ./TEST_CASES
cp $croco/OCEAN/cppdefs.h .
cp $croco/OCEAN/param.h .
cp $croco/OCEAN/Makefile .
cp $croco/OCEAN/jobcomp .
```

- Modify your jobcomp to point to the location of your CROCO source code
- Edit the cppdefs.h file, e.g.:

```
# define DUNE
# define MUSTANG
```

Make sure MUSTANG is activated. For some test cases SEDIMENT (USGS sediment model) is activated by default in cppdefs.h.

25.3 Create your own configuration

1. First choice: V1 or V2 ?

   If you need bedload - you dont't have the choice:

   ```
   # define key_MUSTANG_V2
   ```

   What MUSTANG V2 has to offer:

   - Bedload
   - A new conceptual model for sediment mixture erosion
   - A new model to compute porosity

   http://www.ifremer.fr/docmars/html/doc_MUSTANG/doc.MUSTANG.process.html

2. Modify the param.h file

   Define the number of substances *ntrc_subs* Define the number of layers *ksdmin*,*ksdmax*

3. Create your SUBSTANCE & MUSTANG input files

   Write down the name and location of your files in the croco.in file

   Create the substance file by copying **parasubstance_MUSTANG_full_example.txt**. Keep only the sections that matter (e.g. don't keep a listing of sand and gravel parameters if your run only includes muds). Remember that the number of variables should correspond to **ntrc_subs** in param.h

   Create a user-defined MUSTANG namelist by copying the default one (paraMUSTANG_default.txt). Keep only the parameters that matter for your configuration. If MUSTANG does not find a parameter in the user-defined namelist file, it will use the value defined in the default namelist file.

4. Modify the cppdefs.h file

   Choose what you want to model with the main CPP keys:

   - Without special CPP key, the model is morpho-static. The seabed evolution does not impact the bathymetry seen by the ocean model. If you want do to morphodynamics run:

     ```
     # define MORPHODYN
     ```

     This should also activate automatically MORPHODYN_MUSTANG_byHYDRO in the cppdefs_dev.

     Plus, you will have to put l_morphocoupl=.true. in paraMUSTANG*.txt

   - Sand transported in suspension in 3D (no CPP key needed) : that might be very cost effective for regional scale modelling (i.e. if your CROCO time step is large compared to the time step neeeded to guarantee the stability of the explicit settling scheme).
   - Sand transported in suspension in pseudo 2D

     ```
     # define key_sand2D
     # define MUSTANG_CORFLUX
     ```

Warning: If you want to add a source of sand (e.g. rivers) with the pseudo-2D scheme, it has not been tested yet. Most probably your discharge will only be a fraction of what you wanted. You will need to either adjust the concentration or to modify step3D_t.F in the following section to sum up the water column fluxes in the bottom layer:

```
!-----------------------------------------------------------
! Apply point sources for river runoff simulations
!-----------------------------------------------------------
```

With # define MUSTANG_CORFLUX the correction will not be done at the open boundaries, creating a gradient there. If this is important to you, you will have to modify sed_MUSTANG.F90 in the following section, similarly to what was done with MARS3D:

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!! interpolation of corflux,corfluy at mesh edges  and                                   !!
!!!!!! treatment of corflux,corfluy at grid corners and exchange between MPI processors  !!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

5. CROCO/MUSTANG CPP keys

   - Read wave files:

     ```
     # define WAVE_OFFLINE
     ```

     Activates the reading of wave data (this is an existing CROCO CPP option). If combined with #define MUSTANG, it reads significant wave height, wave period, wave direction and bottom orbital velocity. Then the wave-induced bottom shear stress is computed in sed_MUSTANG_CROCO.F90. Note that the significant wave height (or wave amplitude) has to be given as for now but is not used to compute the bed shear stress.

     Header of an example wave file:

     ```
     dimensions:
     wwv_time = UNLIMITED ; // (2586 currently)
     eta_rho = 623 ;
     xi_rho = 821 ;
     variables:
     double wwv_time(wwv_time) ;
     double hs(wwv_time, eta_rho, xi_rho) ;
             hs:_FillValue = -32767. ;
     double t01(wwv_time, eta_rho, xi_rho) ;
             t01:_FillValue = -32767. ;
     double dir(wwv_time, eta_rho, xi_rho) ;
             dir:_FillValue = -32767. ;
     double ubr(wwv_time, eta_rho, xi_rho) ;
             ubr:_FillValue = -32767. ;
     ```

   - Read netcdf files for solid discharge in river:

     ```
     #  define PSOURCE_NCFILE
     #  define PSOURCE_NCFILE_TS
     ```

     It reads the concentration values in get_psource_ts.F

     Header of an example source file:

     ```
     dimensions:
     qbar_time = 7676 ;
     n_qbar = 6 ;
     runoffname_StrLen = 30 ;
     temp_src_time = 8037 ;
     salt_src_time = 8037 ;
     MUD1_src_time = 7676 ;
     variables:
     double qbar_time(qbar_time) ;
             qbar_time:long_name = "runoff time" ;
             qbar_time:units = "days" ;
     ```

(continues on next page)

(continued from previous page)

```
        qbar_time:cycle_length = 0 ;
        qbar_time:long_units = "days since 1900-01-01" ;
double Qbar(n_qbar, qbar_time) ;
        Qbar:long_name = "runoff discharge" ;
        Qbar:units = "m3.s-1" ;
char runoff_name(n_qbar, runoffname_StrLen) ;
double temp_src_time(temp_src_time) ;
        temp_src_time:cycle_length = 0 ;
        temp_src_time:long_units = "days since 1900-01-01" ;
double salt_src_time(salt_src_time) ;
        salt_src_time:cycle_length = 0 ;
        salt_src_time:long_units = "days since 1900-01-01" ;
double temp_src(n_qbar, temp_src_time) ;
        temp_src:long_name = "runoff temperature" ;
        temp_src:units = "Degrees Celcius" ;
double salt_src(n_qbar, salt_src_time) ;
        salt_src:long_name = "runoff salinity" ;
        salt_src:units = "psu" ;
double MUD1_src_time(MUD1_src_time) ;
        MUD1_src_time:long_name = "runoff time" ;
        MUD1_src_time:units = "days" ;
        MUD1_src_time:long_units = "days since 1900-01-01" ;
double MUD1_src(n_qbar, MUD1_src_time) ;
```

6. Initial conditions for the sediment cover

   There are mainly 2 options:

   - Uniform sediment cover

     In paraMUSTANG*.txt:

     ```
     l_unised = .true.       ! boolean set to true for a uniform␣
     ↪bottom initialization
     fileinised =  './Init.nc' ! File for initialisation (if l_
     ↪unised is False)
     hseduni = 0.03          ! initial uniform sediment␣
     ↪thickness(m)
     cseduni= 1500.0         ! initial sediment concentration
     csed_mud_ini = 550.0     ! mud concentration into initial␣
     ↪sediment (if =0. ==> csed_mud_ini=cfreshmud)
     ksmiuni = 1             ! lower grid cell indices in the␣
     ↪sediment
     ksmauni = 10            ! upper grid cell indices in the␣
     ↪sediment
     ```

     And then, the fraction of each sediment variable in the seafloor is defined with *cini_sed_n()* in parasubsance_MUSTANG.txt

   - Read the sediment cover from a netcdf file or restart from a RESTART file

     In paraMUSTANG*.txt:

     ```
     l_repsed=.true.        ! boolean set to .true. if␣
     ↪sedimentary variables are initialized from a previous run
     filrepsed='./repsed.nc' ! file from which the model is␣
     ↪initialized for the continuation of a previous run
     ```

     The netcdf file needs to have the concentration values under the names *NAME_sed*, with NAME corresponding to the names defined in the SUBSTANCE input files. The number of vertical levels (ksmi, ksma) and the layer thickness (DZS) also need to be defined. The file structure is similar to the RESTART netcdf file, and filerepsed can be used to restart from a CROCO RESTART file.

Header of an example sediment cover file:

```
dimensions:
ni = 821 ;
nj = 623 ;
time = UNLIMITED ; // (1 currently)
level = 10 ;
variables:
double latitude(nj, ni) ;
double longitude(nj, ni) ;
double time(time) ;
double level(level) ;
double ksmi(time, nj, ni) ;
double ksma(time, nj, ni) ;
double DZS(time, level, nj, ni) ;
double temp_sed(time, level, nj, ni) ;
double salt_sed(time, level, nj, ni) ;
double GRAV_sed(time, level, nj, ni) ;
double SAND_sed(time, level, nj, ni) ;
double MUD1_sed(time, level, nj, ni) ;
```

Alternatively there is a 3rd option possible. If l_repsed=.false. and l_unised=.false., you can specify the filename of your sediment cover dataset (fileinised), but then it is up to you to write yourself the piece of code to read it in initMUSTANG.F90 in the subroutine MUSTANG_sedinit.

How to prescribe the concentration for the initialisation :

- Uniform sediment cover. If you use a uniform sediment cover, the initial fraction of each sediment class is read in parasubastance_MUSTANG.txt. Then the concentration of each sediment class is a fraction of *cseduni* defined in paraMUSTANG.txt (i.e. *cv_sed(iv)=cini_sed_n(iv) x cseduni*). However, since you prescribe *cseduni*, it is not necessarily similar to what the model total concentration should be for the same sediment mixture, unless you used the same porosity model as in MUSTANG to compute *cseduni*.

  With MUSTANG V2, after initialisation, the sediment concentration is adjusted in every layers to match the model porosity law. Hence the initial mass is not preserved, but the bed height and the sediment class fractions are.

  With MUSTANG V1, by default the sediment concentration is not adjusted. In this case, what will happen is that the first time erosion happens, the very first deposit could have a very different porosity to the initial state, and induce an abrupt bed height change. You can select l_init_hsed=.true. to bypass this issue. While adjusting the sediment concentration, it will also adjust the sediment height to conserve the initial mass.

  Note: With MUSTANG V2 we recommend using l_init_hsed=.false. since the subroutine associated with this boolean uses the porosity model of V1.

- RESTART. If you use l_repsed=.true., l_init_hsed is not even read. In V1, the sediment concentrations that you specify will not be overwritten. It means that you have to start with concentrations that follow the porosity law of the model. In V2, concentrations are overwritten in all layers after computing the porosity for the sediment mixture. In this cas you can specify concentration that are just a fraction of an arbitrary constant total sediment concentration.

Warning: In version 1, you can impose no sediment in a grid cell by imposing ksmi=1 and ksma=0. This could be useful to define reefs for instance. In Version 2 you need at least one sediment layer everywhere. The first layer is never eroded, but is needed to manage the small sediment mass that can be left in the layer just above.

To avoid potential issue when computing concentrations for very thin layers, thin layers are merged with underlying layers. Therefore, when initializing sediment concentrations make sure to have at least one layer everywhere.

CHAPTER

TWENTYSIX

TRAINING 2019: DATARMOR SPECIFIC

26.1 Getting the good environment

Warning: This is specific to DATARMOR cluster used for this training; if you are working on you own computer, follow the **System Requirements and Downloading the code** tutorials to download the code, and set-up your environment

An environment script has been created for this training on DATARMOR. It will load the necessary modules and set some useful paths and environment variables. Copy this croco_env.csh script and source it. *If you already have a .cshrc or .tcshrc or .bashrc environment script, please copy it to .chsrc.bck to avoid overdefinitions and use only croco_env.csh during the training period.*

```
cd $HOME
cp /home/datawork-croco/datarmor-only/TRAININGS/TRAINING_2019/croco_env.* .
source croco_env.csh
```

Now the `$CROCO_DIR` environment variable is defined and you will find useful material for this training in this directory.

26.2 Creating your work architecture

Let's work on your WORKDIR to avoid disk space issues.

```
cd $work
mkdir TRAINING_2019
cd TRAINING_2019
mkdir croco
mkdir CONFIGS

cp -r $CROCO_DIR/SOURCE_CODES/CROCO/croco_git/croco croco/.
cp -r $CROCO_DIR/SOURCE_CODES/CROCO/croco_git/croco_tools croco/.
```

If you have followed this architecture, the following environment variables have also been placed to facilitate navigation:

- `$croco` point to your croco sources: `$work/TRAINING_2019/croco/croco`
- `$tools` point to your croco sources: `$work/TRAINING_2019/croco/croco_tools`
- `$confs` point to your croco sources: `$work/TRAINING_2019/CONFIGS`

Investigate by your own the various directories.

Warning: do not modify any of the files contained in your source directories `$croco` and `$tools` to keep your source files clean; modifications should be perfomed in your configuration directories (as we will see later)

26.3 DATA FILES

Datasets for preparing surface and boundary conditions from climatological dataset can be downloaded on CROCO website. For this training you will find them in `$CROCO_DIR/DATA/DATASETS_CROCOTOOLS` ; otherwise see the Download tutorial.

You can also find the following global atmospheric reanalysis in `$CROCO_DIR/DATA/METEOROLOGICAL_FORCINGS/`:

- ERAI
- CFSR

And the following ocean reanalysis in `$CROCO_DIR/DATA/3D_OCEAN_FORCING`:

- SODA
- ECCO2

26.4 BASIN configuration for XIOS tutorial

```
cp -R /home/datawork-croco/datarmor-only/CONFIGS/TUTO20/BASIN_NO_XIOS/*
↪$confs/BASIN
cd $confs/BASIN
```

Path for XIOS sources:

:: XIOS_ROOT_DIR=/home/datawork-croco/datarmor-only/SOURCE_CODES/XIOS/XIOS-2.5

26.5 SOURCES for coupling tutorial

For DATARMOR training, OASIS has already been compiled, so you can just copy the sources and compiled files

```
mkdir -p $work/TRAINING_2019/oasis
cp -r $CROCO_DIR/SOURCE_CODES/OASIS/OASIS3-MCT_3.0_branch_compiled $work/
↪TRAINING_2019/oasis/OASIS3-MCT_3.0_branch
```

The configure file for compiling OASIS on DATARMOR, named `make.datarmor` can be found here

```
$CROCO_DIR/make.datarmor
```

For DATARMOR training, WRF has been compiled, and you can just copy the source and compiled files:

```
mkdir -p $work/TRAINING_2019/wrf
cp -r $CROCO_DIR/SOURCE_CODES/WRF/WRFV3.7.1_compiled $work/TRAINING_2019/wrf/WRFV3.
↪7.1
```

A job for compilation is also provided:: $CROCO_DIR/job_compile_wrf.pbs

For DATARMOR training, WPS has been compiled, and you can just copy the source and compiled files:

```
cp -r $CROCO_DIR/SOURCE_CODES/WRF/WPSV3.7.1 $work/TRAINING_2019/wrf/.
```

For DATARMOR training, these data are avaiable in `$CROCO_DIR/SOURCE_CODES/WRF/geog`.

For DATARMOR training, CFSR data for WRF are available in `$CROCO_DIR/DATA/METEOROLOGICAL_FORCINGS/CFSR/GLOBAL/NATIVE_format`

For DATARMOR training, WW3 has been compiled, and you can just copy the source and compiled files:

```
mkdir -p $work/TRAINING_2019/ww3
cp -r $CROCO_DIR/SOURCE_CODES/WW3/github/WW3_compiled/* $work/TRAINING_2019/ww3/.
```

For DATARMOR training, TOY model files are provided here:

```
cp $CROCO_DIR/SOURCE_CODES/TOY/toy_compiled/toy_model $confs/Run_BENGUELA_LR_
↪cpl/.

cp $CROCO_DIR/DATA/BENGUELA_CPL/toy_files/* $confs/Run_BENGUELA_LR_cpl/.
```

You should now have the following new files in your configuration directory:

- toy_model
- grid_wav.nc
- TOYNAMELIST.nam
- toy_wav.nc

An example of fulfilled namcouple is also provided in `$CROCO_DIR/DATA/BENGUELA_CPL/oasis_files`

Note: Documentation on PBS use on DATARMOR can be found here: https://w3z.ifremer.fr/intraric/Mon-IntraRIC/Calcul-et-donnees-scientifiques/Datarmor-Calcul-et-Donnees/Datarmor-calcul-et-programmes

CHAPTER

TWENTYSEVEN

IFREMER SPECIFIC

This tutorial is written in the Framework of the supercomputer (DATARMOR) located at Ifremer. It's also a guide for those who are working with MARS3D model and who want to make their configurations with CROCO

27.1 *Croco training in the framework of datarmor*

27.1.1 *First step :install*

Getting the good environment

Warning: This is specific to DATARMOR cluster used for this training; if you are working on you own computer, follow the **System Requirements and Downloading the code** tutorials to download the code, and set-up your environment

An environment script has been created for this training on DATARMOR. It will load the necessary modules and set some useful paths and environment variables. Copy this croco_env.csh script and source it. *If you already have a .cshrc or .tcshrc or .bashrc environment script, please copy it to .chsrc.bck to avoid overdefinitions and use only croco_env.csh during the training period.*

```
cd $HOME
cp /home/datawork-croco/datarmor-only/TRAININGS/TRAINING_2021/croco_env.* .
source croco_env.csh
```

Now the `$CROCO_DIR` environment variable is defined and you will find useful material for this training in this directory.

Creating your work architecture

Let's work on your WORKDIR to avoid disk space issues.

```
cd $work
mkdir TRAINING_2021
cd TRAINING_2021
mkdir croco
mkdir CONFIGS

cp -r $CROCO_DIR/../../SOURCE_CODES/CROCO/croco_git/croco_master/croco croco/.
cp -r $CROCO_DIR/../../SOURCE_CODES/CROCO/croco_git/croco_tools croco/.
```

If you have followed this architecture, the following environment variables have also been placed to facilitate navigation:

- `$croco` point to your croco sources: `$work/TRAINING_2021/croco/croco`

- `$tools` point to your croco sources: `$work/TRAINING_2021/croco/croco_tools`
- `$confs` point to your croco sources: `$work/TRAINING_2021/CONFIGS`

Warning: do not modify any of the files contained in your source directories `$croco` and `$tools` to keep your source files clean; modifications should be perfomed in your configuration directories (as we will see later)

1. Investigate by your own the various directory below ./croco

```
AGRIF : AGRIF refinement library
CVTK : to check mpi reproductibility
OCEAN : sources code themselves
PISCES : biogeochemical code
Run/TEST_CASES : the crocos.in for the various test_cases
XIOS : input-ouput server that can be coupled to croco
etc ...
```

27.1.2 *Second step: launch a test case*

BASIN

27.1.3 *Third step: set up your own test case*

Set up you own test case

27.1.4 *REALISTIC CONFIGURATION*

Example of coastal configuration

The VILAINE case is an example of a realistic coastal configuration taking into account :

- Tidal circulation
- Wet/dry areas
- River outflows
- Sediment dynamic with MUSTANG

The configuration is included in CROCO as a reference coastal case (see `cppdefs.h`)

1. **Set the environment**

```
source ~/croco_env.sh
```

2. **Create a configuration directory:**

```
mkdir $confs/VILAINE
```

3. **Copy the input files for compilation from croco sources:**

```
cd $confs/VILAINE
cp $croco/OCEAN/cppdefs.h .
cp $croco/OCEAN/param.h .
cp $croco/OCEAN/jobcomp .
```

4. **Edit `cppdefs.h` for using BASIN case**

```
# define COASTAL

# undef REGIONAL
```

You can also explore the CPP options selected for VILAINE case.

- which physical parametrizations ?
- which advection schemes ?

You can check the VILAINE settings in `param.h`:

- Dimension of the grid ?
- Number of vertical levels ?

5. Edit the compilation script `jobcomp`:

   see *BASIN*

6. **Get the inputs files for the run**

```
cp /home/datawork-croco/public/ftp/CONFIGS_EXAMPLES/VILAINE/croco.in .
cp -r /home/datawork-croco/public/ftp/CONFIGS_EXAMPLES/VILAINE/CROCO_FILES␣
↪.
```

   Take a look of the input files in CROCO_FILES and check if it's filled out correctly in `croco.in` file

7. **Get the namelist for MUSTANG module**

```
cp -r /home/datawork-croco/public/ftp/CONFIGS_EXAMPLES/VILAINE/MUSTANG_
↪NAMELIST .
```

8. Compile the model in MPI with 28 cpus

   - Edit the `param.h` file to choose the number of cpus
   - **Check if MPI is activated for the VILAINE case in `cppdefs.h`**

```
# define MPI
```

   - **Get the compile batch script and compile**

```
cp $CROCO_DIR/batch_comp_datarmor .
qsub batch_comp_datarmor
```

   - **Get the run script to submit your job on Datarmor**

```
cp $CROCO_DIR/job_croco_mpi.pbs .
qsub job_croco_mpi.pbs
```

9. Assign a new fill value to land mask cells

   - copy scalars.h

```
cp $croco/OCEAN/scalars.h .
```

   - edit the file and replace spval

```
spval=999.
```

   - add CPP key FILLVAL in your `cppdefs.h`

```
define FILLVAL
```

- add this key in `cppdefs.h` to not add bathymetry on wet dry cells

```
define ZETA_DRY_IO
```

Build a configuration from scratch

Preparation of forcing files

Mesh building with BMGTOOLS

1. Get BMGTOOLS here

```
mkdir BMG
cp /home/datawork-mars/TOOLS/BATHY/BMGTOOLS/bmg-linux64b-rev1489.tar.gz .
tar -xzvf bmg-linux64b-rev1489.tar.gz
```

2. Open the **Create module** in a terminal

```
cd create_bmg-5.0.0
./CreateBMG.sh
```

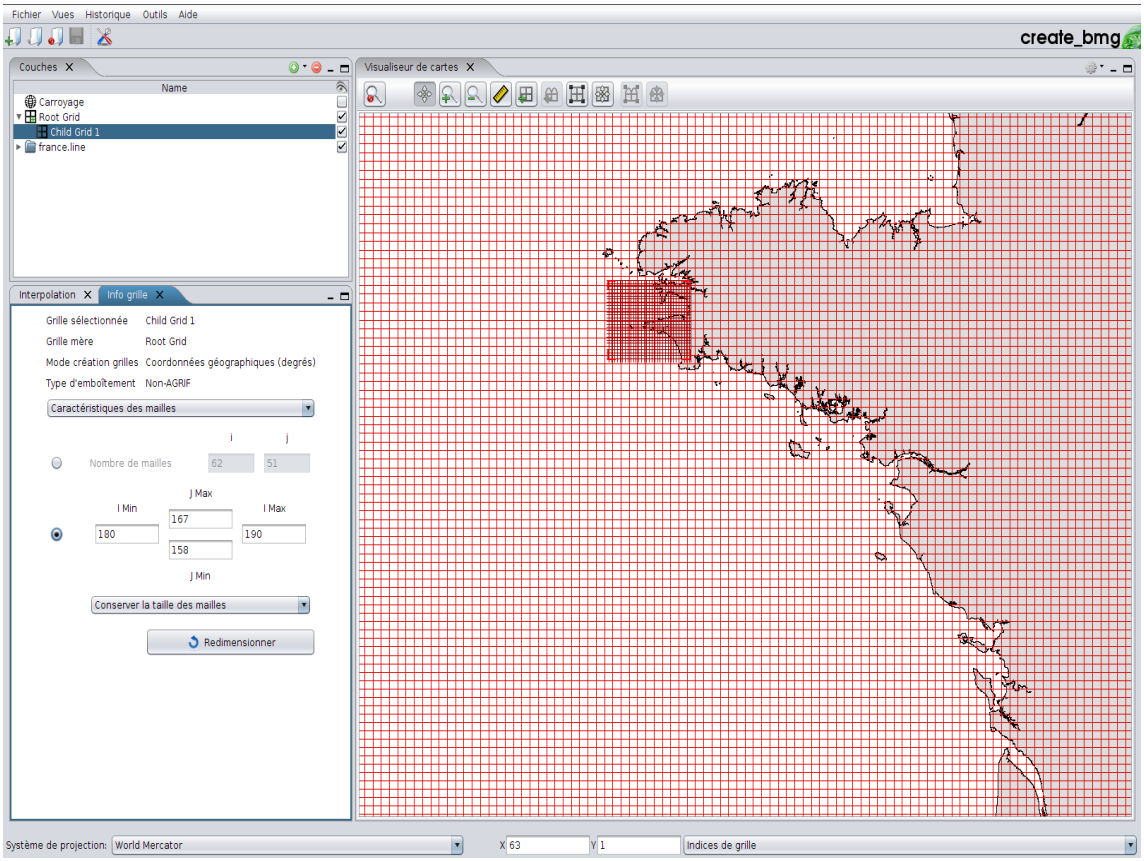
> **Warning:** If a memory is requested put 4GO

3. Get the appropriate coastline to build your configuration here

```
/home/datawork-croco/datarmor-only/DATA/COASTLINE/BMGTOOLS_FORMAT/france.line
/home/datawork-croco/datarmor-only/DATA/COASTLINE/BMGTOOLS_FORMAT/europa.
↪closed.line
/home/datawork-croco/datarmor-only/DATA/COASTLINE/BMGTOOLS_FORMAT/med_sea.line
```

4. Create a new project (Top left button)
5. Create a grid with the following features (Button on the top right bar) and follow the instructions
   - Click and drag on the map to define approximatively your domain
   - **In the popup window you can define :**
     - The limits of your area of interest
     - If you want to choose your grid resolution by meters choose the option **Grid defined by curvilinear mesh size**
6. Save your project and check in the directory that you have a file head.TEST with the features of your grid

```
TEST0  65.0000000  40.0000000  15.0000000 -20.0000000   0.0500000   0.0833333 ␣
↪      5662.40   5563.84 421 501 167 158 190 180 TEST
```



7. Interpolation of the bathymetry on the grid

   - Get the fortran executable, the associated namelist and the batch

```
/home/datawork-mars/TOOLS/BATHY/INTERP/interp_bathy/INTERP_BATHY.exe
/home/datawork-mars/TOOLS/BATHY/INTERP/interp_bathy/namelist
/home/datawork-mars/TOOLS/BATHY/INTERP/interp_bathy/batch_interp
```

   - Build a text file which list the MNT files you want to use and pickup from here

```
cat catalog.dat :
/home/datawork-croco/datarmor-only/DATA/MNT_HOMONIM/MNT_ATL100m_HOMONIM_
↪WGS84_NM.nc
```

   - Edit the namelist

```
Choose the grid2grid mode
&flags
  l_interp_soundings2grid=.false.
  l_interp_grid2grid=.true.
  l_smooth=.false.
  l_connect=.false. /

&interp_soundings2grid
  coastfile='/home/datawork-croco/datarmor-only/DATA/TDC/france.line'

&interp_grid2grid
  data_catalog='catalog.cat'
  l_bathy_bmg=.true.
  l_closed_line=.true.
  landvalue=-999
  grid_file='RootGrid.nc'
  nivmoypath=''
```

(continues on next page)

(continued from previous page)

```
  l_bathy_threshold=.true.
  bathy_threshold=2.0
  mask_method='HXHY' /
```

- Launch the executable

```
qsub batch_interp
```

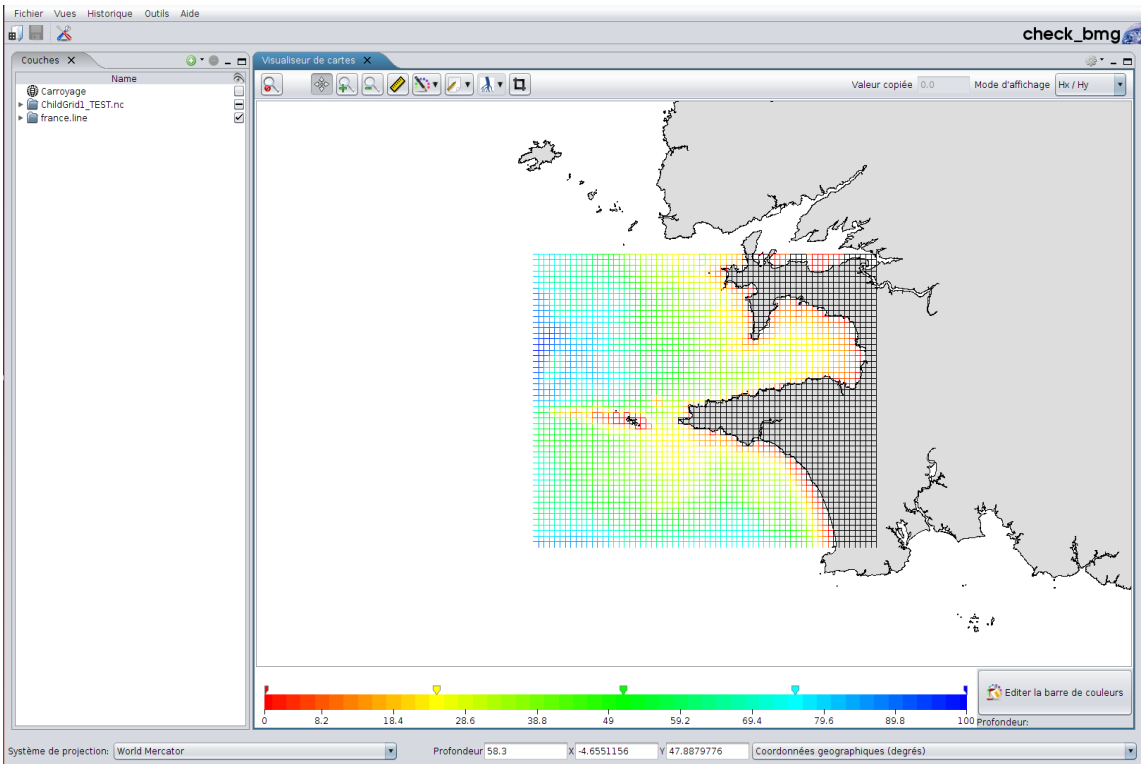
8. Open in another terminal the **Check BMG** module to view and edit (if needed) your bathymetry

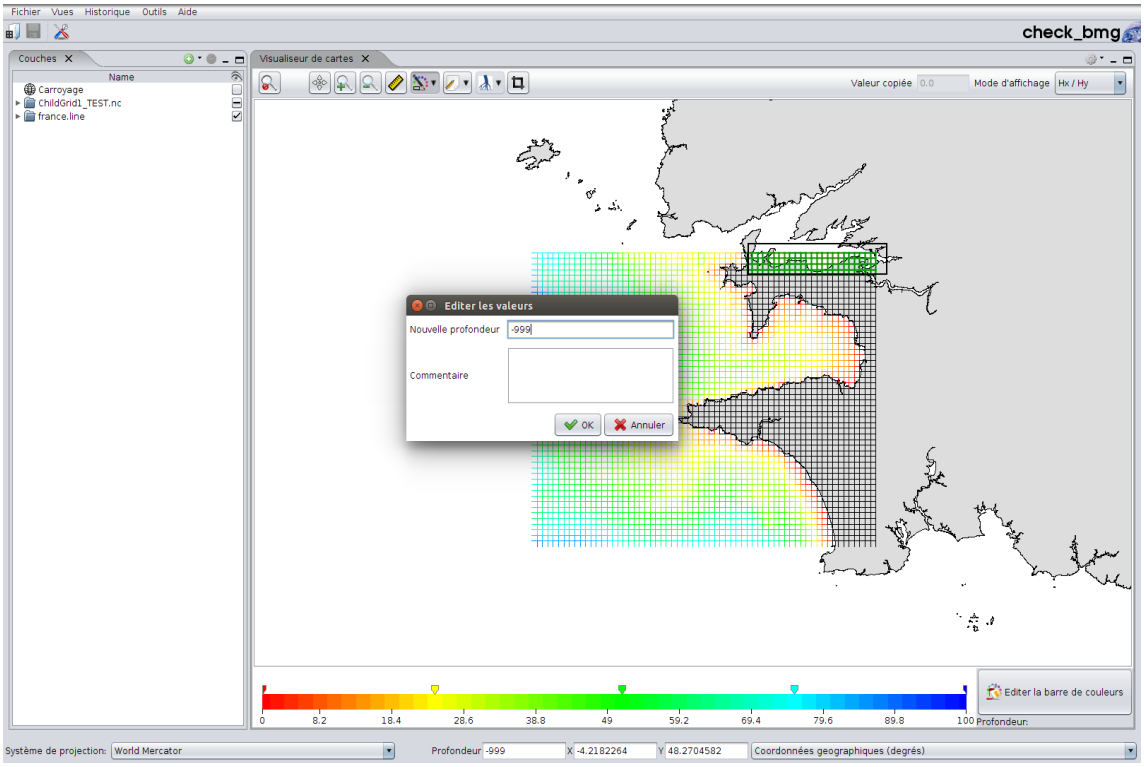
   - Open

```
cd check_bmg-5.0.0
./CheckBMG.sh
```

   - Load your grid file (*RootGrid.nc*) and your coastline file
   - You can edit your grid with the button in the top right pannel with different ways
     - Single (one mesh)
     - Sequential (several meshes in sequential)
     - Polygon (group of meshes within a polygon)
     - Rectangle (groupe meshes within a rectangle)

     You juste have to select the nodes and edit the bathymetry values





Warning: dont forget to save your project to take into acount your modifications

9. Convert the bathymetry and the grid to CROCO framework

   Preprocessing of files are based on a set of python scripts.

   On Datarmor you can get a python with vacumm

```
module load vacumm/3.4.0
```

   - Get the python script from CROCO directory

```
cp -r /home/datawork-croco/datarmor-only/TRAININGS/TRAINING_2021/
↪MARS2CROCO/BATHY .
cd BATHY
```

   - Edit the python script **convert_bathymars2croco.py** and set user parameters
   - Launch the script

```
python convert_bathymars2croco.py bathy_file.nc
==> You get croco_grd.nc
```

   - Check if your bathy seems ok

```
ncview croco_grd.nc
```

Build tidal atlas on CROCO grid

To get tide on your OBC, you need an atlas with harmonic constituents on your model grid **croco_grd.nc** * First you also need the following script

```
cp /home/datawork-croco/datarmor-only/TRAININGS/TRAINING_2021/MARS2CROCO/
↪TIDES/convert_fes2croco.py .
cp /home/datawork-croco/datarmor-only/TRAININGS/TRAINING_2021/MARS2CROCO/
↪TIDES/tides.txt .
cp /home/datawork-croco/datarmor-only/TRAININGS/TRAINING_2021/batch_
↪python .
```

- Edit the python script to set the list of constituents you want in your Atlas
- The script need the **croco_grd.nc** file so you need to link it the directory

```
ln -s ../BATHY/croco_grd.nc .
```

- Run the script

```
qsub batch_python
```

3D Initial and Boundary conditions

This part deals with generation of OBC and IC for your grid, from an Ocean General Circulation Model (exemple :MERCATOR, HYCOM ..)

1. First you have to get the numerical solution which covers your grid and your period of simulation

```
/home/datawork-croco/datarmor-only/DATA/MERCATOR_SOLUTION
```

2. The second step is to interpolate this file on your grid. We use a fortran programm for this :
   - Get the following directory

```
cp -r /home/datawork-croco/datarmor-only/TRAININGS/TRAINING_2021/EXTRACT_
↪CROCO .
cd EXTRACT
```

   - edit the namelist

```
vi IN/namelist
```

   - Change the mode (IC or OBC at once)

```
&extractmode
 l_extract_obc=.false.        ! compute Open Boundaries Conditions (set to␣
↪true)
 l_extract_ic=.false. /       ! compute Initial Conditions         (set to␣
↪true)
```

   - Change the name of the input file with the MERCATOR file

```
&namcoarse
 file_coarse = 'MERCATOR_PSY2V4.nc'   ! name of file containing data to be␣
↪interpolated
```

   - In this section set the name of init and obc file, input bathy file and activate which obc you want to extract

```
&namfine
 head_fine = 'head.useless'        ! head.CONF file (a line of the one␣
↪used in MARS)
 file_ic='init.nc'          ! name of ic  output file
 file_fine_w = 'obc_west.nc'       ! name of west  obc output file
 file_fine_e = 'obc_east.nc'       ! name of east  obc output file
 file_fine_s = 'obc_south.nc'      ! name of south obc output file
 file_fine_n = 'obc_north.nc'      ! name of north obc output file
 l_obc_west  = .false.                  ! Interpolate west  obc ?
 l_obc_east  = .false.                ! Interpolate east  obc ?
 l_obc_south = .false.                ! Interpolate south obc ?
 l_obc_north = .false.                ! Interpolate north obc ?
 obc_width = 2                       ! width of obc domain, must the same␣
↪than in MARS
 file_bathy_fine = 'bathy_rang1_2500_final.nc'
```

- **Adpat the parameters for interpolation :**
  - Set l_interpxyz to .false. ==> it enables 2D interpolation with SCRIP and vertical interpolation with splines
  - If your OGCM model is in SIGMA coordinates set your own intermediate Z vertical profile (used for vertical interpolation sigma to sigma)

```
Z=(0:immersion1:dh1_ref,immersion1:immersion2:dh2_ref,
↪immersion2:immersion3:dh3_ref)
```

```
&param_interp
 l_interpxyz = .false.     !
 rapdist = 6.0             ! only if l_interpxyz=.true. must be
↪<= 6
 radius  = 20.0e+3         ! only if l_interpxyz=.true.
 aspect_ratio = 10         !  only if l_interpxyz=.true.
 dh1_ref=1.0               ! only if OGCM in Sigma (dz between 0␣
↪and immersion1)
 immersion1 =40.0          ! only if OGCM in Sigma (first␣
↪immersion below 0 in z profil)
 dh2_ref= 2.0              ! only if OGCM in Sigma (dz between␣
↪immersion1 and immersion2)
 immersion2 =60.0          ! only if OGCM in Sigma (second␣
↪immersion in z profil)
 dh3_ref=5.0               ! only if OGCM in Sigma (dz between␣
↪immersion2 and immersion3)
 immersion3=200.0          ! only if OGCM in Sigma (last␣
↪immersion in z profil : must be >= MAX(H0) !!!)
 nextrap = 10              ! 2D spatial extrapolation iteration
 l_correct_rho=.false.          ! correct vertical density␣
↪instabilities
 l_complete_prof_first=.false.   ! extrapolate Z profils before␣
↪doing interpolation
 l_interp_conserv=.true.  /     ! perform conservative vericale␣
↪interpolation instead of splines
```

  - Launch the executable in batch mode

```
qsub batch_extract
```

Build a new configuration with CROCO

Open a terminal and login to Datarmor

```
ssh -X login@datarmor
```

Environment and source code

1. Setup environment
   - Source this file to set some environment variables

```
cd $HOME
cp /home/datawork-croco/datarmor-only/TRAININGS/TRAINING_2021/croco_env.* .
source croco_env.csh
```

   - Build a "CROCO" directory on your $DATAWORK

```
cd $work
mkdir TRAINING_2021
cd TRAINING_2021
mkdir croco
mkdir CONFIGS
```

2. Get the source code

```
cp -r $CROCO_DIR/../../SOURCE_CODES/CROCO/croco_git/croco_master/croco croco/.
cp -r $CROCO_DIR/../../SOURCE_CODES/CROCO/croco_git/croco_tools croco/.
```

3. Create a new config
   - build a directory for your new configuration and get the following scripts from the source code directory

```
cd $confs
mkdir my_config
cd my_config
mkdir CROCO_FILES
cp $croco/OCEAN/param.h .
cp $croco/OCEAN/cppdefs.h .
cp $croco/OCEAN/jobcomp .
cp $CROCO_DIR/job* .
```

Edit your configuration parameters files

1. Setup **param.h**
   - First section : define your domain dimensions (get xi_rho and eta_rho from **croco_grd.nc**)

```
LLm0 = xi_rho-2 ; MMm0=eta_rho-2


#elif defined REGIONAL
# elif defined SEINE
     parameter (LLm0=410,   MMm0=180,   N=20)    ! SEINE
```

   - Second section : define your MPI decomposition

```
Choose your decomposition so number_procs=NP_XI*NP_ETA

#ifdef MPI
  integer NP_XI, NP_ETA, NNODES
  parameter (NP_XI=14, NP_ETA=6,  NNODES=NP_XI*NP_ETA)
```

- If you are using WET_DRY set the critical depth

```
#ifdef WET_DRY
      real D_wetdry             ! Critical Depth for Drying cells
# else
      parameter (D_wetdry=0.4)
```

- Third section :: Number of harmonic components

```
#else
  parameter (Ntides=114)
```

- Fourth section :: Number of river

```
#if defined PSOURCE || defined PSOURCE_NCFILE
      integer Msrc               ! Number of point sources
      parameter (Msrc=9)        ! ====== == ===== =======
#endif
```

2. Edit **cppdef.h** : Simple config with only tides at boundaries
   - Activate *REGIONAL* case

```
#define REGIONAL        /* REGIONAL Applications */
```

   - Set configuration name (same as **param.h**)

```
/* Configuration Name */
 # define  MEDI5KM
```

   - Set MPI parallelisation

```
# define  MPI
```

   - Activate TIDES and define Open boundary conditions according to your domain

```
/* Open Boundary Conditions */
# define  TIDES
# undef  OBC_EAST
# define  OBC_WEST
# define  OBC_NORTH
# undef   OBC_SOUTH
```

   - In **preselected options** only change these ones :
     - vertical mixing

```
/* Vertical Mixing */
# define GLS_MIXING
```

     - Analytical surface fluxes

```
/* Surface Forcing */
# undef  BULK_FLUX
/* Suppression des termes atmospheriques */
# define ANA_SSFLUX   /* analytical salinity flux */
```

(continues on next page)

(continued from previous page)

```
# define ANA_STFLUX   /* analytical Latent and Sensible flux */
# define ANA_SMFLUX   /* surface momentum flux = wind */
# define ANA_SRFLUX   /* surface short surface radiative */
# define ANA_SST     /* climatological surface temperature */
# define ANA_SSS      /* climatological surface salinity */
# undef ANA_TCLIMA   /* climatological surface for others tracers */
```

– Lateral Forcing

```
/* Lateral Forcing */
# undef CLIMATOLOGY
# define ANA_INITIAL
# define ANA_BRY
# define FRC_BRY
# ifdef FRC_BRY
#  define Z_FRC_BRY
#  define M2_FRC_BRY
#  undef M3_FRC_BRY
#  undef T_FRC_BRY
# endif
```

– Bottom Forcing

```
/* Bottom Forcing */
# define ANA_BSFLUX
# define ANA_BTFLUX
```

– Desactivate source

```
/* Point Sources - Rivers */
# undef PSOURCE
# undef PSOURCE_NCFILE
# ifdef PSOURCE_NCFILE
#   define PSOURCE_NCFILE_TS
# endif
```

3. Compile the model in batch mode

   • Edit your job comp and set the source code path

   ```
   SOURCE=$croco/OCEAN
   ```

   • Launch the compilation

   ```
   qsub job_comp_datarmor.pbs
   ```

Note: you should get a **croco** executable file

4. Edit the config input file **croco.in**

   • Time stepping

   ```
   time_stepping: NTIMES   dt[sec]  NDTFAST  NINFO
                  594000      40      10      1
   ```

   – **NTIMES** : Number of global time step (dt)

   you can use this script to get NTIMES given your start and end date

   ```
   /home/datawork-croco/datarmor-only/FORMATION/PREPROC/calc_steps.py
   Duration of your simulation == NTIMES*dt
   ```

  - **dt** : baroclinic time step (depend on your mesh grid size)
  - **NDTFAST** : number of fast time step in one baroclinic time step

- Sigma distribution

```
S-coord: THETA_S,   THETA_B,    Hc (m)
          0.0d0      0.0d0      2000.0d0
```

- Origin date (only) for Netcdf

```
start_date:
 01-01-1900 00:00:00
```

- Set the path to your inputs files which should be in a **CROCO_FILES** directory

```
grid:  filename
                CROCO_FILES/croco_grd.nc
forcing: filename
                CROCO_FILES/croco_frc_manga16.nc
bulk_forcing: filename
                CROCO_FILES/bidon.nc
climatology: filename
                CROCO_FILES/croco_clm.nc
boundary: filename
                CROCO_FILES/croco_bry.nc
initial: NRREC  filename
          -1
                CROCO_FILES/croco_ini.nc
restart:          NRST, NRPFRST / filename
                9000     -2
                CROCO_FILES/croco_rst.nc
history: LDEFHIS, NWRT, NRPFHIS / filename
           T       90       0
                CROCO_FILES/croco_his.nc
averages: NTSAVG, NAVG, NRPFAVG / filename
            1     2140     0
                 CROCO_FILES/croco_avg.nc
```

- Choose which variables you want to save in your output (T/F to activate/desactivate)

```
primary_history_fields: zeta UBAR VBAR  U  V   wrtT(1:NT)
                   T    F   F   T  T    30*T
auxiliary_history_fields:   rho Omega  W  Akv  Akt  Aks Visc3d Diff3d  HBL␣
↪HBBL Bostr Wstr Ustr Vstr Shfl Swfl rsw rlw lat sen  HEL
                            F   F     F   F    F    F    F     F       F ␣
↪ F    F     F    F    F    F    F   10*F
gls_history_fields:   TKE  GLS  Lscale
                       F     F    F


primary_averages: zeta UBAR VBAR  U  V   wrtT(1:NT)
                   F    F    F    F  F   30*T
auxiliary_averages: rho Omega  W  Akv  Akt  Aks  Visc3d Diff3d HBL HBBL␣
↪Bostr Wstr Ustr Vstr Shfl Swfl rsw rlw lat sen HEL
                    F   F     F   F    F    F     F     F       F   F    F␣
↪    F   F    F     F    F   10*F
gls_averages:   TKE  GLS  Lscale
                 F     F    F
```

- Set lateral viscosity (ONLY if UV_VIS2 or UV_VIS4 cpp key are enabled)

```
lateral_visc:   VISC2,    VISC4    [m^2/sec for all]
                6.34       0.
```

- Set lateral diffusivity (ONLY if UV_DIFF2 or UV_DIFF4 cpp key are enabled)

```
tracer_diff2: TNU2(1:NT)           [m^2/sec for all]
            30*1.d-2

tracer_diff4: TNU4(1:NT)           [m^4/sec for all]
            30*0.d11
```

- Set bottom drag

```
bottom_drag:     RDRG [m/s],  RDRG2,  Zob [m],  Cdb_min, Cdb_max
                0.0d-4       5.d-3    3.5d-3     1.d-4    1.d-1
```

  - Barotropic mode :: RDRG superseded by RDRG2
  - Baroclinc mode :: RDRG superseded by RDRG2 superseded by Z0B

5. Edit **batch_mpt** to set the right number of nodes and walltime (1node=28 procs)

```
#PBS -q mpi_3
#PBS -l mem=8gb
#PBS -l walltime=10:00:00
#PBS -N CROCO_SEINE
```

6. Launch the model

```
qsub batch_mpt
```

7. Visualize
   - First use ncview

```
module load ncview
ncview CROCO_FILES/croc_his.nc
```

Custom you configuration

Add a source for a river discharge

- In **cppdefs.h** you should activate
  - *PSOURCE* : activate tracer for sources
  - PSOURCE_NCFILE : if you want to use a chronological discharge (dont use it for now)

```
/* Point Sources - Rivers */
# define PSOURCE
# undef PSOURCE_NCFILE
# ifdef PSOURCE_NCFILE
#   define PSOURCE_NCFILE_TS
# endif
```

- In **param.h** set the number of source points

```
#if defined PSOURCE || defined PSOURCE_NCFILE
    integer Msrc                ! Number of point sources
    parameter (Msrc=5)         ! ====== == ===== =======
#endif
```

- Compile your model

- Edit **croco.in** file

  First line is for the number of sources then there should be one line by source

```
psource:   Nsrc  Isrc  Jsrc  Dsrc  Qbar [m3/s]    Lsrc        Tsrc
           1
                  310  23     0    -900.            T T      0. 0.
```

  - **Isrc,Jsrc** : Coordinates of point sources
  - **Dsrc** : Direction of outflow (0 along u, 1 along v)
  - **Qbar** : Average discharge in m3/s (positive to the North/East, negative to the South/West)
  - **Lsrc** : Logical for associate tracers to the source (here Temp,Sal)
  - **Tsrc** : Tracer value (here Temp,Sal)

Add a real Atmospheric forcing

- **In cppdefs.h you should activate**
  - *ONLINE* : Use online interpolation (spatial and temporal) from an meteo model on different grid
  - *AROME* : data are formatted in MeteoFrance framework
  - *BULK_FLUX* : Compute bulk fluxes
  - *BULK_FAIRALL* : use FAIRALL formulation for bulk
  - *BULK_SMFLUX* : compute surface momentum flux (from wind stress)
  - *READ_PATM* : Read atmospherical pressure in atm file and use it in the code for bulk and surface pressure gradient

```
# define  BULK_FLUX
# ifdef BULK_FLUX
#  define BULK_FAIRALL
#  undef BULK_LW
#  undef BULK_EP
#  define BULK_SMFLUX
#  ifdef BULK_SMFLUX
#   define BULK_SM_UPDATE
#  endif
#  undef  SST_SKIN
#  undef  ANA_DIURNAL_SW
#  define  ONLINE
#  define AROME
#  define READ_PATM
#  undef  ERA_ECMWF
#  undef  RELATIVE_WIND
# else
#  undef QCORRECTION
#  undef SFLX_CORR
#  undef ANA_DIURNAL_SW
# endif
```

  - Dont forget to remove analytical bulk fluxes

```
/* Suppression des termes atmospheriques */
 # define ANA_SSFLUX   /* surface salinity */
 # define ANA_STFLUX   /* surface temperature */
 # undef ANA_SMFLUX   /* surface momentum flux = wind */
 # undef ANA_SRFLUX   /* surface short surface radiative */
```

(continues on next page)

(continued from previous page)

```
# define ANA_SST
# define ANA_SSS
```

- Recompile the model
- Go to your configuration directory and make a link to this file

  cd /home1/datawork/login/CROCO/config cd CROCO_FILES ln -s /home/datawork-croco/datarmor-only/DATA/METEOROLOGICAL_FORCINGS/ARPEGE-HR_2017_final.nc .
- Now edit the **croco.in** file (see bottom of file)

  Set begin year, end year and mont, number of records per day in your dataset and the path of the file

```
online:    byear  bmonth recordsperday byearend bmonthend / data path
           2017   1      24             2017     12
           CROCO_FILES/ARPEGE-HR_2017_final.nc
```

Add 3D IC and OBC

- First you need to get your IC and OBC files see *3D Initial and Boundary conditions*
- Edit **cppdefs.h** to activate BRY conditions

  undefine analytical Init and boundary conditions activate BRY for Tracers (T_FRC_BRY)

```
# undef ANA_INITIAL
# undef ANA_BRY
# define FRC_BRY
# ifdef FRC_BRY
#  define Z_FRC_BRY
#  define M2_FRC_BRY
#  undef M3_FRC_BRY
#  undef T_FRC_BRY
# endif
```

- Compile the model
- Copy you croco_ic.nc and croco_bry.nc files in the CROCO_FILES diretory
- Edit **croco.in** file

  Set the path to your IC/OBC files

```
boundary: filename
            CROCO_FILES/croco_bry.nc
 initial: NRREC  filename
     -1
            CROCO_FILES/croco_ini.nc
```

Note: When you start from an init file the start date of your simulation is the date of the file

27.1.5 *FERRET FACILITY*

Ferret : a practical tool for fast visualisation on datarmor

this step is dedicated to basic git usage to manage properly your source code and (potentially) interact with croco's developers the croco's repository is so far hosted at Inria (https://gitlab.inria.fr)

1. Load the appropriate module

```
module avail
module load ferret/7.1__64b
```

   Launch it by typing

```
ferret
```

2. Load your netcdf dataset (yes? is the usual prompt)

```
yes? use data.nc
```

   or

```
yes? use "/home6/datawork/login/Simulation/data.nc"
```

3. Visualise the data structure

```
yes? show data
```

   you will get a list of all the variables contained in the fill loaded and their dimensions :

   - i : designate the x dimension
   - j : designate the y dimension
   - k : designate the vertical dimension
   - l : designate the temporal dimension

4. List the numerical values of a section of a given variable :

   From now on let's consider the variable **temp** which gets four dimensions (time + x,y,z).

```
yes? list /i=10/j=10/k=40 temp
```

   This will list all the numerical data at the level 40, i=10, j=10 of the variable **temp**.

5. Plot a one dimensionnal feature by fixing n-1 of the variable dimension number (n).

```
yes? plot /i=10/j=10/k=40 temp
```

   this will plot the time series of the variable **temp**.

```
yes? plot /i=10/j=10/l=4 temp
```

   this will plot the vertical profile of the variable temp at time l=4.

```
yes? plot /i=10/j=10/l=4 temp
yes? plot /i=10/j=10/l=40/ove temp
```

   the same as the previous but with superimposition of two profile at two different instants (l=10 and l=40)

6. Plot a two dimensionnal features by fixing n-2 of the variable dimension number (n).

```
yes? plot /i=10/j=10 temp
```

   This will plot a Hovmuller diagram (time vs z) of the variable temp.

```
yes? plot /k=40/j=10 temp
```

In case, k=40 designate the surface layer, this will plot a hovemuller diagram along all longitudes vs time.

```
yes? plot /k=40/j=10/lev=(10.,20.,1.) temp
```

The same as the previous one but setting a color bar that extends from 10 to 20 with bins of 1.

```
yes? plot /k=40/j=10/lev=(0)(10.,20.,1.)(30) temp
```

The same as the previous one but extending the first and last color class respectively down to 0 and up to 30.

27.1.6 *GIT FACILITY*

manage coherently your configurations/developments with git

this step is dedicated to basic git usage to manage properly your source code and (potentially) interact with croco's developers the croco's repository is so far hosted at Inria (https://gitlab.inria.fr)

1. Request an access to croco's gitlab

```
go to URL   https://gitlab.inria.fr/croco-ocean/croco
click on the upright corner **register** tab
then on the right side of the page on the highlight register
confirm your registration with the mail you received
go back https://gitlab.inria.fr
log in
search croco project
select the croco projet and then ask for an access
```

2. Once you get the access create your own local repository

```
mkdir /homeX/datahome/login/croco/
cd croco
git init
git clone git@gitlab.inria.fr:croco-ocean/croco.git
```

3. List of all the available branches

```
git branch -v -a
```

4. Create your own local branch (tutu) from a given remote branch (toto)

```
git checkout -b tutu remotes/origin/toto
```

5. Get the status of the local repository

```
git status
```

6. Update of the branch named "toto" with the remote branch

```
git pull origin dyneco_rec
```

7. Toto

```
git remote show origin
```

8. Log

```
git log
```

27.1.7 *XIOS FACILITY*

XIOS step by step

1. Change your jobcomp

```
if [[ $HOSTNAME == *"datarmor"* ]]; then
XIOS_ROOT_DIR=/home1/datawork/mcaillau/CROCO/XIOS
```

2. Get the XML files and the routine send_xios_diags:

```
cp /home/datawork-croco/datarmor-only/FORMATION/SRC/croco/XIOS/*.xml .
cp /home/datawork-croco/datarmor-only/FORMATION/SRC/croco/XIOS/*.xml_full .
cp /home/datawork-croco/datarmor-only/FORMATION/SRC/croco/XIOS/send_xios_diags.
↪F .
```

BIBLIOGRAPHY

[YU2003] Yu, J., & Slinn, D. N. (2003). Effects of wave-current interaction on rip currents. Journal of Geophysical Research: Oceans, 108(C3).

[THORNTON1983] Thornton, E.B. & R.T. Guza, 1983: Transformation of wave height distribution, J. Geophys. Res. 88, 5925-5938.

[UCHYIAMA2009] Uchiyama, Y., McWilliams, J. C., & Restrepo, J. M. (2009). Wave-current interaction in nearshore shear instability analyzed with a vortex force formalism. Journal of Geophysical Research: Oceans, 114(C6).

[WEIR2011] Weir, B., Uchiyama, Y., Lane, E. M., Restrepo, J. M., & McWilliams, J. C. (2011). A vortex force analysis of the interaction of rip currents and surface gravity waves. Journal of Geophysical Research: Oceans, 116(C5).