Croco Documentation

Release 2.1.0

S. Jullien, M. Caillaud, R. Benshila, L. Bordois,
G. Cambon, F. Dufois, F. Dumas, J. Gula,
M. Le Corre, S. Le Gac, S. Le Gentil,
F. Lemarié, P. Marchesiello, G. Morvan,
J. Pianezze, L. Renault, A.L. Schaefer,
M. Schreiber, S. Theetten and S. Valat,

Mar 21, 2025

CONTENTS

1 Model Documentation **3**
1.1 Governing Equations . . . 3
1.1.1 Primitive Equations . . . 3
1.1.2 Quasi-Hydrostatic Equations . . . 6
1.1.3 Wave-averaged Equations . . . 6
1.1.4 Non-Hydrostatic, Non-Boussinesq Equations . . . 10
1.2 Model variables . . . 11
1.2.1 Domain variables (*grid.h*) . . . 11
1.2.2 Barotropic variables (*ocean2d.h*) . . . 12
1.2.3 Tri-dimensionnal variables (*ocean3d.h*) . . . 12
1.2.4 Surface forcing (forces.h) . . . 13
1.3 Grid and Coordinates . . . 14
1.3.1 Vertical Grid parameters . . . 15
1.3.2 Grid staggering . . . 17
1.3.3 Wetting-Drying . . . 17
1.4 Numerics . . . 17
1.4.1 Overview . . . 17
1.4.2 Time Stepping . . . 18
1.4.3 Advection Schemes . . . 22
1.4.4 Pressure gradient . . . 28
1.4.5 Equation of State . . . 28
1.4.6 Wetting and Drying . . . 28
1.4.7 Non-Boussinesq Solver . . . 29
1.5 Parametrizations . . . 29
1.5.1 Vertical mixing parametrizations . . . 29
1.5.2 Horizontal diffusion . . . 35
1.5.3 Bottom friction . . . 37
1.6 Parallelisation . . . 37
1.6.1 Parallel strategy overview . . . 38
1.6.2 Variable placement for staggered grids . . . 41
1.6.3 Loops and indexes for staggered grids . . . 41
1.6.4 Halo layer exchanges . . . 42
1.6.5 Dealing with outputs . . . 42
1.6.6 Run with GPU . . . 43
1.7 Atmospheric Surface Boundary Layer . . . 44
1.8 Open boundaries conditions . . . 48
1.8.1 OBC . . . 48
1.8.2 Sponge Layer . . . 49
1.8.3 Nudging layers . . . 49
1.8.4 Lateral forcing . . . 50
1.9 Rivers . . . 51
1.10 Tides . . . 52
1.11 Wavemaker . . . 54
1.11.1 Formulation . . . 54

1.11.2 Input parameters . . . 54
1.11.3 JONSWAP spectrum . . . 55
1.12 Nesting Capabilities . . . 56
1.13 Other modules : sediment models, flow-obstruction models, biology models . . . 57
1.13.1 Bottom Boundary Layer model . . . 57
1.13.2 Sediment models . . . 59
1.13.3 OBSTRUCTIONS module : flow in presence of various obstructions . . . 115
1.13.4 Biogeochemical models . . . 127
1.13.5 Lagrangian floats . . . 128
1.14 Coupling CROCO with other models . . . 128
1.14.1 OASIS philosophy . . . 129
1.14.2 Detailed OASIS implementation . . . 132
1.14.3 Coupled variables . . . 137
1.14.4 Grids . . . 144
1.15 I/O and Online Diagnostics . . . 145
1.15.1 General ouptut options . . . 145
1.15.2 Advanced diagnostics options . . . 146
1.16 Testing CROCO . . . 150
1.16.1 Test Script . . . 150
1.16.2 Review of test cases . . . 161
1.17 Appendices . . . 236
1.17.1 cppdefs.h . . . 236
1.17.2 croco.in . . . 245
1.17.3 Comparison of ROMS and CROCO versions . . . 251

2 Tutorials **255**
2.1 System requirements . . . 255
2.1.1 Disk space . . . 255
2.1.2 Compilers and Libraries . . . 255
2.1.3 Environment variables . . . 255
2.2 Download . . . 256
2.2.1 Downloading CROCO . . . 256
2.2.2 Getting other codes (coupling) . . . 256
2.3 Contents & Architecture . . . 258
2.3.1 Architecture . . . 258
2.3.2 Contents . . . 258
2.4 Summary of essential steps . . . 264
2.5 Test Cases . . . 265
2.5.1 BASIN . . . 265
2.5.2 Set up you own test case . . . 267
2.6 Regional: Preparing your configuration . . . 270
2.7 Regional: Preprocessing (Matlab) . . . 271
2.7.1 Contents of the `croco_tools` . . . 272
2.7.2 Philosophy of the `croco_tools` . . . 272
2.7.3 Climatological pre-processing . . . 273
2.7.4 Interannual pre-processing . . . 276
2.8 Compiling . . . 280
2.8.1 cppdefs.h . . . 280
2.8.2 param.h . . . 283
2.8.3 Compilation using jobcomp . . . 284
2.8.4 Compilation options . . . 285
2.8.5 Tips in case of errors during compilation . . . 286
2.9 Running the model . . . 286
2.9.1 Edit croco.in . . . 287
2.9.2 Run the model . . . 290
2.9.3 Tips in case of BLOW UP or ERROR . . . 291
2.10 Increasing the resolution: BENGUELA_VHR . . . 291
2.11 Running with interannual forcing . . . 292

2.11.1 Run after classical interannual pre-processing . . . 292
2.11.2 Alternative method: online interpolation of atmospheric bulk forcing . . . 294
2.12 Running forecasts . . . 295
2.12.1 Strategy of Forecast_tools . . . 296
2.12.2 Set forecast parameters . . . 296
2.12.3 Compiling . . . 298
2.12.4 Running the script . . . 298
2.13 Nesting Tutorial . . . 299
2.14 Adding Rivers . . . 302
2.14.1 Constant flow and concentration . . . 302
2.14.2 Variable flow read in a netCDF file and **constant concentration** . . . 303
2.14.3 Variable flow and **variable concentration** from a netCDF file . . . 305
2.14.4 Using a nest . . . 306
2.15 Adding tides . . . 306
2.15.1 Pre-processing (Matlab) . . . 307
2.15.2 Compiling . . . 307
2.15.3 Running . . . 308
2.16 Visualization (Matlab) . . . 308
2.17 Python tools for CROCO . . . 310
2.18 NBQ Tutorial . . . 311
2.18.1 Some important points about Large-Eddy Simulations (LES) . . . 311
2.18.2 KH_INST Test Case . . . 313
2.18.3 Set up your own NBQ configuration . . . 315
2.18.4 NBQ OPTIONS . . . 316
2.18.5 Appendix : some words on CROCO-NBQ kernel . . . 316
2.19 Coupling tutorial . . . 317
2.19.1 Summary of steps for coupling . . . 317
2.19.2 Compiling in coupled mode . . . 318
2.19.3 Simple CROCO-TOY coupled example . . . 326
2.19.4 Advanced coupling tutorial . . . 331
2.20 Littoral tutorial . . . 364
2.21 Realistic coastal configuration . . . 368
2.22 XIOS . . . 368
2.23 Tips . . . 371
2.23.1 Tips in case of errors during compilation . . . 371
2.23.2 TIPS for errors at runtime . . . 372
2.23.3 Analytical forcing . . . 372
2.23.4 MUSTANG tips . . . 373

Bibliography **379**

CROCO is an oceanic modeling system built upon ROMS_AGRIF and maintained by IRD, INRIA, CNRS, IFREMER and SHOM, French institutes working on environmental sciences and applied mathematics. An important objective for CROCO is to resolve very fine scales (especially in the coastal area), and their interactions with larger scales. It includes new capabilities such as a non-hydrostatic solver, ocean-wave-atmosphere coupling, evolving sediment dynamics and marine biogeochemistry, new high-order numerical schemes for advection and mixing, and a dedicated I/O server (XIOS). A toolbox for pre- and post-processing, CROCO_TOOLS, accompanies the source code. CROCO will keep evolving and integrating new capabilities in the following years.

CHAPTER

ONE

MODEL DOCUMENTATION

1.1 Governing Equations

Related CPP options:

SOLVE3D	Solve 3D primitive equations
UV_COR	Activate Coriolis terms
UV_ADV	Activate advection terms
NBQ	Activate non-boussinesq option
CROCO_QH	Activate quasi-hydrostatique option
MRL_WCI	Activate wave-current interactions

Preselected options:

```
# define SOLVE3D
# define UV_COR
# define UV_ADV
# undef  NBQ
# undef  CROCO_QH
# undef  MRL_WCI
```

Presentation

By default (#undef NBQ), CROCO solves the primitive equations as in ROMS, from which it inherited the robustness and efficiency of its time-splitting implementation [Shchepetkin and McWilliams, 2005, Debreu *et al.*, 2012] and the NBQ option proposes an extension for nonhydrostatic applications. In CROCO's time-splitting algorithm, the "slow mode" is similar to ROMS internal (baroclinic) mode described in Shchepetkin and McWilliams [2005], whereas, the "fast mode" can include, 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]. In this case, the slow internal mode is also augmented by a prognostic equation of vertical velocity, replacing the hydrostatic equation. Another option (CROCO_QH) extends the PE equations to form the quasi-hydrostatic equations, relaxing the hypothesis of weak horizontal Coriolis force [Marshall *et al.*, 1997], thus adding a nonhydrostatic pressure component that is solved diagnostically. Then another option (MRL_WCI) treats the wave-averaged equations [McWilliams *et al.*, 2004] with wave-current interaction terms that are both conservative and non-conservative (needing parametrizations).

1.1.1 Primitive Equations

At resolutions larger than 1 km (more marginally above 100 m), The ocean is a fluid that can be described as a good approximation by the primitive equations (PE). The PE equations are simplifications from the Navier-Stokes equations made from scale considerations, along with a nonlinear equation of state, which couples the two active tracers (temperature and salinity):

- Hydrostatic hypothesis: the vertical momentum equation is reduced to a balance between the vertical pressure gradient and the buoyancy force (non-hydrostatic processes such as convection must be parametrized)
- Boussinesq hypothesis: density variations are neglected except in their contribution to the buoyancy force

- Incompressibility hypothesis (stemming from the former): the three-dimensional divergence of the velocity vector is assumed to be zero.
- Spherical earth approximation: the geopotential surfaces are assumed to be spheres so that gravity (local vertical) is parallel to the earth's radius
- Thin-shell approximation: the ocean depth is neglected compared to the earth's radius
- Turbulent closure hypothesis: the turbulent fluxes (which represent the effect of small-scale processes on the large scale) are expressed in terms of large-scale features

By default (#undef NBQ), CROCO solves the primitive equations. However, it also has the ability to relax the first 3 hypotheses (#define NBQ). When SOLVE3D is not activated, CROCO can be used as a classical shallow water model.

1.1.1.1 Equations in Cartesian coordinates

- The momentum balance in zonal x and meridional y directions, written in terms of grid-scale (resolved) and subgrid-scale velocity components:

$$\frac{\partial u}{\partial t} + \vec{\nabla}.(\vec{\mathbf{v}}u) - fv = -\frac{\partial \phi}{\partial x} + \mathcal{F}_u + \mathcal{D}_u$$
$$\frac{\partial v}{\partial t} + \vec{\nabla}.(\vec{\mathbf{v}}v) + fu = -\frac{\partial \phi}{\partial y} + \mathcal{F}_v + \mathcal{D}_v$$
$$Turbulentclosureschemesareappliedtoparametrizedsubgrid - scaleverticalfluxes.$$

- The time evolution of a scalar concentration field, $C(x, y, z, t)$ (e.g., salinity, temperature, or nutrients), is governed by the advective-diffusive equation :

$$\frac{\partial C}{\partial t} + \vec{\nabla}.(\vec{\mathbf{v}}C) = \mathcal{F}_C + \mathcal{D}_C$$

- The equation of state is given by :

$$\rho = \rho(T, S, P)$$

- In the Boussinesq approximation, density variations are neglected in the momentum equations except in their contribution to the buoyancy force in the vertical momentum equation. Under the hydrostatic approximation, it is further assumed that the vertical pressure gradient balances the buoyancy force :

$$\frac{\partial \phi}{\partial z} = -\frac{\rho g}{\rho_0}$$

- The final equation expresses the continuity equation. For an incompressible fluid (Boussinesq approximation):

$$\vec{\nabla}.\vec{\mathbf{v}} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0$$

The variables used are :

$\mathcal{D}_u, \mathcal{D}_v, \mathcal{D}_C$: diffusive terms

$\mathcal{F}_u, \mathcal{F}_v, \mathcal{F}_C$: forcing terms

$f(x, y)$: Traditional Coriolis parameter $2\Omega sin\phi$

g : acceleration of gravity

$\phi(x, y, z, t)$: dynamic pressure $\phi = P/\rho_0$, with P the total pressure

$\rho_0 + \rho(x, y, z, t)$: total in situ density

u, v, w : the (x,y,z) components of vector velocity $\vec{\mathbf{v}}$

1.1.1.2 Equations in terrain-following coordinates

We first introduce a generalized stretched vertical coordinate system (s), which sets the variable bottom flat at $z = -h(x, y)$. s spans the range from -1 (bottom) to 0 (surface), and the transformation rules are:

$$\left(\frac{\partial}{\partial x}\right)_z = \left(\frac{\partial}{\partial x}\right)_s - \left(\frac{1}{H_z}\right)\left(\frac{\partial z}{\partial x}\right)_s \frac{\partial}{\partial s}$$
$$\left(\frac{\partial}{\partial y}\right)_z = \left(\frac{\partial}{\partial y}\right)_s - \left(\frac{1}{H_z}\right)\left(\frac{\partial z}{\partial y}\right)_s \frac{\partial}{\partial s}$$
$$\frac{\partial}{\partial z} = \left(\frac{\partial s}{\partial z}\right)\frac{\partial}{\partial s} = \frac{1}{H_z}\frac{\partial}{\partial s}$$
$$\text{where } H_z \equiv \frac{\partial z}{\partial s}$$

The vertical velocity in s coordinate is:

$$\Omega(x, y, s, t) = \frac{1}{H_z}\left[w - (1+s)\frac{\partial \zeta}{\partial t} - u\frac{\partial z}{\partial x} - v\frac{\partial z}{\partial y}\right]$$
$$w = \frac{\partial z}{\partial t} + u\frac{\partial z}{\partial x} + v\frac{\partial z}{\partial y} + \Omega H_z$$

$\Omega = 0$ at both surface and bottom.

Next, the requirement for a laterally variable grid resolution can also be met for suitably smooth domains by introducing an appropriate orthogonal coordinate transformation in the horizontal. Let the new coordinates be $\xi(x, y)$ and $\eta(x, y)$ where the relationship of horizontal arc length to the differential distance is given by:

$$(ds)_\xi = \left(\frac{1}{m}\right) d\xi$$
$$(ds)_\eta = \left(\frac{1}{n}\right) d\eta$$

Here $m(\xi, \eta)$ and $n(\xi, \eta)$ are the scale factors which relate the differential distances $(\Delta\xi, \Delta\eta)$ to the actual (physical) arc lengths.

$$\frac{\partial}{\partial t}\left(\frac{H_z u}{mn}\right) + \frac{\partial}{\partial \xi}\left(\frac{H_z u^2}{n}\right) + \frac{\partial}{\partial \eta}\left(\frac{H_z u v}{m}\right) + \frac{\partial}{\partial s}\left(\frac{H_z u \Omega}{mn}\right)$$
$$- \left\{\left(\frac{f}{mn}\right) + v\frac{\partial}{\partial \xi}\left(\frac{1}{n}\right) - u\frac{\partial}{\partial \eta}\left(\frac{1}{m}\right)\right\} H_z v =$$
$$-\left(\frac{H_z}{n}\right)\left(\frac{\partial \phi}{\partial \xi} + \frac{g\rho}{\rho_o}\frac{\partial z}{\partial \xi} + g\frac{\partial \zeta}{\partial \xi}\right) + \frac{H_z}{mn}\left(\mathcal{F}_u + \mathcal{D}_u\right)$$
$$\frac{\partial}{\partial t}\left(\frac{H_z v}{mn}\right) + \frac{\partial}{\partial \xi}\left(\frac{H_z u v}{n}\right) + \frac{\partial}{\partial \eta}\left(\frac{H_z v^2}{m}\right) + \frac{\partial}{\partial s}\left(\frac{H_z v \Omega}{mn}\right)$$
$$+ \left\{\left(\frac{f}{mn}\right) + v\frac{\partial}{\partial \xi}\left(\frac{1}{n}\right) - u\frac{\partial}{\partial \eta}\left(\frac{1}{m}\right)\right\} H_z u =$$
$$-\left(\frac{H_z}{m}\right)\left(\frac{\partial \phi}{\partial \eta} + \frac{g\rho}{\rho_o}\frac{\partial z}{\partial \eta} + g\frac{\partial \zeta}{\partial \eta}\right) + \frac{H_z}{mn}\left(\mathcal{F}_v + \mathcal{D}_v\right)$$
$$\frac{\partial}{\partial t}\left(\frac{H_z T}{mn}\right) + \frac{\partial}{\partial \xi}\left(\frac{H_z u T}{n}\right) + \frac{\partial}{\partial \eta}\left(\frac{H_z v T}{m}\right) + \frac{\partial}{\partial s}\left(\frac{H_z \Omega T}{mn}\right) = \frac{H_z}{mn}\left(\mathcal{F}_T + \mathcal{D}_T\right)$$
$$\frac{\partial}{\partial t}\left(\frac{H_z S}{mn}\right) + \frac{\partial}{\partial \xi}\left(\frac{H_z u S}{n}\right) + \frac{\partial}{\partial \eta}\left(\frac{H_z v S}{m}\right) + \frac{\partial}{\partial s}\left(\frac{H_z \Omega S}{mn}\right) = \frac{H_z}{mn}\left(\mathcal{F}_S + \mathcal{D}_S\right)$$
$$\rho = \rho(T, S, P)$$
$$\frac{\partial \phi}{\partial s} = -\left(\frac{g H_z \rho}{\rho_o}\right)$$
$$\frac{\partial}{\partial t}\left(\frac{H_z}{mn}\right) + \frac{\partial}{\partial \xi}\left(\frac{H_z u}{n}\right) + \frac{\partial}{\partial \eta}\left(\frac{H_z v}{m}\right) + \frac{\partial}{\partial s}\left(\frac{H_z \Omega}{mn}\right) = 0$$

1.1.2 Quasi-Hydrostatic Equations

In oceanography, traditional approximation (TA) takes the Coriolis force only partially into account by neglecting the components proportional to the cosine of latitude: $\tilde{f} = 2\Omega cos\phi$ (see Gerkema *et al.* [2008], for a review). The justification for the TA is in the hypothesis that the depth of the oceans is very thin compared to the radius of the Earth. The vertical motions must then be much weaker than the horizontal ones, rendering the non-tradiional (NT) Coriolis terms (with $\tilde{f}$) insignificant compared to the traditional terms (with f) and rendering the pressure field nearly hydrostatic. Similarly, strong vertical stratification in density, which suppresses vertical motions, also diminishes the role of NT terms. However, this argument becomes weak near the equator ($\tilde{f} >> f$), or in motions with a strong vertical component (e.g., convection).

Note also that the QH momentum equations are shown to be more dynamically consistent than PE hydrostatic equations and that they correctly imply conservation laws for energy, angular momentum, and potential vorticity

1.1.2.1 Equations in Cartesian coordinate

- The momentum balance in x and y directions is extended to include $\tilde{f}$ terms (zonal u component):

$$\frac{\partial u}{\partial t} + \vec{\nabla}.(\vec{\mathbf{v}}u) - fv + \tilde{f}w = -\frac{\partial \phi}{\partial x} + \mathcal{F}_u + \mathcal{D}_u$$
$$\frac{\partial v}{\partial t} + \vec{\nabla}.(\vec{\mathbf{v}}v) + fu = -\frac{\partial \phi}{\partial y} + \mathcal{F}_v + \mathcal{D}_v$$

- Under the QH approximation, the quasi-hydrostatic balance is used for the vertical momentum equation, where the zonal flow partially balances the pressure gradient :

$$\frac{\partial \phi}{\partial z} = -\frac{\rho g}{\rho_0} + \tilde{f}u$$

In practice, the non-traditional term $\tilde{f}u$ is introduced as a correction to density (in the density computation subroutine rho_eos).

The variables used are :

$\mathcal{D}_u, \mathcal{D}_v$: diffusive terms

$\mathcal{F}_u, \mathcal{F}_v$: forcing terms

$f(x,y)$: Traditional Coriolis parameter $2\Omega sin\phi$

$\tilde{f}(x,y)$: Non-traditional Coriolis parameter $2\Omega cos\phi$

g : acceleration of gravity

$\phi(x,y,z,t)$: dynamic pressure $\phi = P/\rho_0$, with P the total pressure

$\rho_0 + \rho(x,y,z,t)$: total in situ density

u, v, w : the (x,y,z) components of vector velocity $\vec{\mathbf{v}}$

1.1.3 Wave-averaged Equations

MRL_WCI	Activate wave-current interactions
MRL_CEW	Activate current effect on waves (2-way interaction)
ANA_WWAVE	Analytical (constant) wave parameters (Hs,Tp,Dir)
WAVE_OFFLINE	Activate wave forcing from offline model/data
WKB_WWAVE	Activate CROCO's monochromatic (WKB) model
OW_COUPLING	Activate coupling with spectral wave model (WW3)
WAVE_FRICTION	Activate bottom friction for WKB model and WAVE_STREAMING
WAVE_STREAMING	Activate bottom streaming (needs WAVE_FRICTION)
STOKES_DRIFT	Activate Stokes drift

Preselected options:

```
# define STOKES_DRIFT
```

A vortex-force formalism for the interaction of surface gravity waves and currents is implemented in CROCO [Marchesiello *et al.*, 2015, Uchiyama *et al.*, 2010]. Eulerian wave-averaged current equations for mass, momentum, and tracers are included based on an asymptotic theory by McWilliams *et al.* [2004] plus non-conservative wave effects due to wave breaking, associated surface roller waves, bottom streaming, and wave-enhanced vertical mixing and bottom drag especially for coastal and nearshore applications. The wave information is provided by either a spectrum-peak WKB wave-refraction model that includes the effect of currents on waves, or, alternatively, a spectrum-resolving wave model (e.g., WAVEWATCH3) can be used. In nearshore applications, the currents' cross-shore and vertical structure is shaped by the wave effects of near-surface breaker acceleration, vertical component of vortex force, and wave-enhanced pressure force and bottom drag.

1.1.3.1 Equations in Cartesian coordinates

In the Eulerian wave-averaged current equations, terms for the wave effect on currents (WEC) are added to the primitive equations. Three new variables are defined:

$$\begin{aligned}\xi^c &= \xi + \hat{\xi}\\ \phi^c &= \phi + \hat{\phi}\\ \vec{\mathbf{v}}_L &= \vec{\mathbf{v}} + \vec{\mathbf{v}}_S\end{aligned}$$

where ξ^c is a composite sea level, ϕ^c absorbs the Bernoulli head $\hat{\phi}$, $\vec{\mathbf{v}_L}$ is the wave-averaged Lagrangian velocity, sum of Eulerian velocity and Stokes drift $\vec{\mathbf{v}_S}$. The 3D Stokes velocity is non-divergent and defined for a monochromatic wave field (amplitude A, wavenumber vector $\vec{\mathbf{k}} = (k_x, k_y)$, and frequency σ) by:

$$\begin{aligned}u_S &= \frac{A^2\sigma}{2\sinh^2(kD)}\cosh(2k(z+h))\,k_x\\ v_S &= \frac{A^2\sigma}{2\sinh^2(kD)}\cosh(2k(z+h))\,k_y\\ w_S &= -\int_{-h}^{z}\left(\frac{\partial u_S}{\partial x} + \frac{\partial v_S}{\partial y}\right)\,dz'\end{aligned}$$

Where $D = h + \xi^c$. The quasi-static sea level and Bernouilli head are:

$$\begin{aligned}\hat{\xi} &= -\frac{A^2 k}{2\sinh(2kD)}\\ \hat{\phi} &= \frac{A^2\sigma}{4k\sinh^2(kD)}\int_{-h}^{z}\frac{\partial^2\vec{\mathbf{k}}.\vec{\mathbf{v}}}{\partial z'^2}\sinh(2k(z-z'))\;dz'\end{aligned}$$

The primitive equations become (after re-organizing advection and vortex force terms):

$$\begin{aligned}\frac{\partial u}{\partial t} + \vec{\nabla}.(\vec{\mathbf{v}}_L u) - f v_L &= -\frac{\partial\phi^c}{\partial x} + \left(u_S\frac{\partial u}{\partial x} + v_S\frac{\partial v}{\partial x}\right) + \mathcal{F}_u + \mathcal{D}_u + \mathcal{F}^{\mathcal{W}}{}_u\\ \frac{\partial v}{\partial t} + \vec{\nabla}.(\vec{\mathbf{v}}_L v) + f u_L &= -\frac{\partial\phi^c}{\partial y} + \left(u_S\frac{\partial u}{\partial y} + v_S\frac{\partial v}{\partial y}\right) + \mathcal{F}_v + \mathcal{D}_v + \mathcal{F}^{\mathcal{W}}{}_v\\ \frac{\partial\phi^c}{\partial z} + \frac{\rho g}{\rho_0} &= \vec{\mathbf{v}}_S.\frac{\partial\vec{\mathbf{v}}}{\partial z}\\ \frac{\partial C}{\partial t} + \vec{\nabla}.(\vec{\mathbf{v}}_L C) &= \mathcal{F}_C + \mathcal{D}_C + \mathcal{F}^{\mathcal{W}}{}_C\\ \vec{\nabla}.\vec{\mathbf{v}}_L &= 0\\ \rho &= \rho(T,S,P)\end{aligned}$$

The variables used are :

$\mathcal{D}_u, \mathcal{D}_v, \mathcal{D}_C$: diffusive terms (including wave-enhaced bottom drag and mixing)

$\mathcal{F}_u, \mathcal{F}_v, \mathcal{F}_C$: forcing terms

$\mathcal{F}^{\mathcal{W}}{}_u, \mathcal{F}^{\mathcal{W}}{}_v, \mathcal{F}^{\mathcal{W}}{}_C$: wave forcing terms (bottom streaming, breaking acceleration)

$f(x,y)$: Traditional Coriolis parameter $2\Omega sin\phi$

g : acceleration of gravity

$\phi(x,y,z,t)$: dynamic pressure $\phi = P/\rho_0$, with P the total pressure

$\rho_0 + \rho(x,y,z,t)$: total in situ density

u, v, w : the (x,y,z) components of vector velocity $\vec{\mathbf{v}}$

1.1.3.2 Embedded wave model

WKB_WWAVE	Activate WKB wave model
WAVE_ROLLER	Activate wave rollers
WAVE_FRICTION	Activate bottom friction
WKB_ADD_DIFF	Activate additional diffusion to wave number field
MRL_CEW	Active current effect on waves
WKB_KZ_FILTER	Activate space filter on ubar, vbar, zeta for CEW
WKB_TIME_FILTER	Activate time filter on ubar, vbar, zeta for CEW
WAVE_RAMP	Activate wave ramp
ANA_BRY_WKB	Read boundary data from croco.in
WKB_OBC_WEST	Offshore wave forcing at the western boundary
WKB_OBC_EAST	Offshore wave forcing at the eastern boundary

Preselected options:

```
# ifdef MRL_CEW
#  undef  WKB_KZ_FILTER
#  undef  WKB_TIME_FILTER
# endif
# define WKB_ADD_DIFF
# if defined SHOREFACE || defined SANDBAR || (defined RIP && !defined BISCA)
#  define ANA_BRY_WKB
# endif
```

A WKB wave model for monochromatic waves is embedded in CROCO following Uchiyama *et al.* [2010]. It is based on the conservation of wave action $\mathcal{A} = E/\sigma$ and wavenumber $\mathbf{k}$ – wave crest conservation – and is particularly suitable for nearshore beach applications, allowing refraction from bathymetry and currents (but no diffraction or reflection), with parametrizations for wave breaking and bottom drag:

$$\frac{\partial \mathcal{A}}{\partial t} + \vec{\nabla}.\mathcal{A}\vec{\mathbf{c}}_g = -\frac{\epsilon^w}{\sigma}$$

$$\frac{\partial \vec{\mathbf{k}}}{\partial t} + \vec{\mathbf{c}}_g.\nabla\vec{\mathbf{k}} = -\vec{\mathbf{k}}.\nabla\vec{\mathbf{V}} - \frac{k\sigma}{\sinh 2kD}\nabla D$$

$\vec{\mathbf{V}}$ is the depth-averaged velocity vector and σ is the intrinsic frequency defined by the linear dispersion relation $\sigma^2 = gk\tanh kD$. Current effects on waves are noticeable in the groupe velocity c_g which gets two components: the doppler shift due to currents on waves and the groupe velocity of the primary carrier waves :

$$\vec{\mathbf{c}}_g = \vec{\mathbf{V}} + \frac{\sigma}{2k^2}\left(1 + \frac{2kD}{\sinh 2kD}\right)\vec{\mathbf{k}}$$

The currents may need filtering before entering the wave model equations because the current field should evolve slowly with respect to waves in the asymptotic regime described by McWilliams *et al.* [2004]. By default, this filtering is turned off (WKB_KZ_FILTER, WKB_TIME_FILTER).

ϵ^w is the depth-integrated rate of wave energy dissipation due to depth-induced breaking ϵ^b (including white capping) and bottom friction ϵ^{wd}, both of which must be parameterized (in WKB, WW3 or CROCO if defined WAVE_OFFLINE):

$$\epsilon^w = \epsilon^b + \epsilon^{wd}$$

1.1.3.3 Breaking acceleration and bottom streaming

A formulation for ϵ^b is needed in both the wave model (dissipation term) and the circulation model (acceleration term). In the wave-averaged momentum equations of the circulation model, the breaking acceleration enters as a body force through $\mathcal{F}^{\mathcal{W}}$:

$$\vec{\mathbf{F^b}} = \frac{\epsilon^b}{\rho\sigma}\vec{\mathbf{k}}\, f_b(z)$$

where $f_b(z)$ is a normalized vertical distribution function representing vertical penetration of momentum associated with breaking waves from the surface. The penetration depth is controlled by a vertical length-scale taken as H_{rms}.

The wave model can also include a roller model with dissipation ϵ^r. In this case:

$$\vec{\mathbf{F^b}} = \frac{(1-\alpha_r)\epsilon^b + \epsilon^r}{\rho\sigma}\vec{\mathbf{k}}\, f_b(z)$$

The idea is that some fraction α_r of wave energy is converted into rollers that propagate toward the shoreline before dissipating, while the remaining fraction $1-\alpha_r$ causes local dissipation (hence current acceleration). It can be useful for correcting ϵ^b with some flexibility to depict different breaking wave and beach forms (e.g., spilling or plunging breakers, barred or plane beaches), although the parameter B_b can also be used for that. See Uchiyama *et al.* [2010] for the roller equation and ϵ^r formulation.

Wave-enhanced bottom dissipation enters in the momentum equations through a combined wave-current drag formulation (see parametrizations) and bottom streaming. The latter is due to dissipation of wave energy in the wave boundary layer that causes the instantaneous, oscillatory wave bottom orbital velocities to be slightly in phase from quadrature; this causes a wave stress (bottom streaming) in the wave bottom boundary layer along the direction of wave propagation [Longuet-Higgins, 1953]. The effect of bottom streaming in momentum balance is accounted for by using the wave dissipation due to bottom friction with an upward decaying vertical distribution:

$$\vec{\mathbf{F^{st}}} = \frac{\epsilon^{wd}}{\rho\sigma}\vec{\mathbf{k}}\, f_{st}(z)$$

where $f_{st}(z)$ is a vertical distribution function.

1.1.3.4 Formulation of wave energy dissipation

WAVE_SFC_BREAK	Activate surface breaking acceleration
WAVE_BREAK_CT93	Activate Church and Thornton [1993] breaking acceleration (default)
WAVE_BREAK_TG86	Activate Thornton and Guza [1983], Thornton and Guza [1986]

Preselected options:

```
# define WAVE_BREAK_CT93
# undef  WAVE_BREAK_TG86
# undef  WAVE_SFC_BREAK
```

While a few formulations for ϵ^b are implemented in CROCO, the one by Church and Thornton [1993] is generally successful for nearshore beach applications:

$$\epsilon^b = \frac{3}{16}\sqrt{\pi}\rho g B_b^3 \frac{H_{rms}^3}{D}\left\{1+\tanh\left[8\left(\frac{H_{rms}}{\gamma_b D}-1\right)\right]\right\}\left\{1-\left[1+\left(\frac{H_{rms}}{\gamma_b D}\right)^2\right]^{-2.5}\right\}$$

where B_b and γ_b are empirical parameters related to wave breaking. γ_b represents the wave height-to-depth ratio for which all waves are assumed to be breaking and B_b is the fraction of foam on the face, accounting for the type

of breaker. H_{rms} is the RMS wave height. For the DUCK94 experiment, Uchiyama *et al.* [2010] suggest $\gamma_b = 0.4$ and $B_b = 0.8$, while for Biscarrosse Beach, Marchesiello *et al.* [2015] use $\gamma_b = 0.3$ and $B_b = 1.3$ from calibration with video cameras.

For ϵ^{wd}, the dissipation caused by bottom viscous drag on the primary waves, we use a parameterization for the realistic regime of a turbulent wave boundary layer, consistent with the WKB spectrum-peak wave modeling:

$$\epsilon^{wd} = \frac{1}{2\sqrt{\pi}} \rho f_w u_{orb}^3$$

where u_{orb} is the wave orbital velocity magnitude and f_w is a wave friction factor, function of roughness length z_0:

$$u_{orb} = \frac{\sigma H_{rms}}{2 \sinh kD}$$
$$f_w = 1.39 \left(\frac{\sigma z_0}{u_{orb}} \right)^{0.52}$$

1.1.4 Non-Hydrostatic, Non-Boussinesq Equations

The full set of Navier-Stokes equations for a free-surface ocean is explicitly integrated in the non-hydrostatic, non-Boussinesq version of CROCO (#define NBQ). In this approach, acoustic waves are solved explicitly to avoid Boussinesq-degeneracy, which inevitably leads to a 3D Poisson system in non-hydrostatic Boussinesq methods – detrimental to computational costs and challenging to implement within a split-explicit barotropic/baroclinic model.

NBQ equations include the momentum and continuity equations, the surface kinematic relation (for free surface), temperature, salinity – or other tracer C – and the equation of state, which reads in Cartesian coordinates:

$$\begin{aligned}
\frac{\partial \rho u}{\partial t} + \vec{\nabla}.(\rho \vec{\mathbf{v}} u) - \rho f v + \rho \tilde{f} w &= -\frac{\partial P}{\partial x} + \lambda \frac{\partial \vec{\nabla}.\vec{\mathbf{v}}}{\partial x} + \mathcal{F}_u + \mathcal{D}_u \\
\frac{\partial \rho v}{\partial t} + \vec{\nabla}.(\rho \vec{\mathbf{v}} v) + \rho f u &= -\frac{\partial P}{\partial y} + \lambda \frac{\partial \vec{\nabla}.\vec{\mathbf{v}}}{\partial y} + \mathcal{F}_v + \mathcal{D}_v \\
\frac{\partial \rho w}{\partial t} + \vec{\nabla}.(\rho \vec{\mathbf{v}} w) - \rho \tilde{f} u &= -\frac{\partial P}{\partial z} - \rho g + \lambda \frac{\partial (\vec{\nabla}.\vec{\mathbf{v}})}{\partial z} + \mathcal{F}_w + \mathcal{D}_w \\
\frac{\partial \rho}{\partial t} &= -\vec{\nabla}.(\rho \vec{\mathbf{v}}) \\
\frac{\partial \xi}{\partial t} &= w_f|_{z=\xi} - \vec{\mathbf{v}}|_{z=\xi}.\vec{\nabla}\xi \\
\frac{\partial \rho C}{\partial t} &= -\vec{\nabla}.(\rho \vec{\mathbf{v}} C) + \mathcal{F}_C + \mathcal{D}_C
\end{aligned}$$

λ is the second (bulk) viscosity associated with compressibility (it can be used to damp acoustic waves).

A relation between ρ and P is now required. To that end, and as part of a time-splitting approach, density is decomposed into slow and fast components based on a first-order decomposition concerning total pressure. In the following, s and f subscripts refer to slow and fast-mode components, respectively:

$$\rho = \rho_s(T,S,P) + \overbrace{\left.\frac{\partial \rho}{\partial P}\right|_{T,S} \delta P}^{\rho_f = c_s^{-2} P_f} + O(\delta P^2)$$

$$P = \underbrace{P_{atm} + \int_z^\xi (\rho_s - \rho_0) g \, dz'}_{SLOW} + \underbrace{\rho_0 g (\xi - z) + \overbrace{\delta P}^{P_f}}_{FAST}$$

c_s is the speed of sound and $\delta P = P_f$ is the nonhydrostatic pressure.

The Navier-Stokes equations are then integrated with two different time steps within the time-splitting approach. The slow mode is identical to ROMS, whereas the fast mode (in the NBQ equations) is 3D and the fast time step

includes the integration of the compressible terms of the momentum and continuity equations. In vector form:

$$
\begin{aligned}
\frac{\partial \rho \vec{\mathbf{v}}}{\partial t} &= \underbrace{-\vec{\nabla}.(\rho\vec{\mathbf{v}}\otimes\vec{\mathbf{v}}) - 2\rho\vec{\Omega}\times\vec{\mathbf{v}} - \vec{\nabla}(\int_z^{\xi_f}(\rho_s-\rho_0)g\,dz') + \vec{\mathcal{F}}_{\vec{\mathbf{v}}} + \vec{\mathcal{D}}_{\vec{\mathbf{v}}}}_{SLOW} \\
&\quad \underbrace{-\rho_0 g\vec{\nabla}\xi_f - \vec{\nabla}P_f + \rho\vec{\mathbf{g}} + \lambda\vec{\nabla}(\vec{\nabla}.\vec{\mathbf{v}})}_{FAST} \\
\frac{\partial \rho_f}{\partial t} &= -\frac{\partial \rho_s}{\partial t} - \vec{\nabla}.(\rho\vec{\mathbf{v}}) \\
P_f &= c_s^2\,\rho_f \\
\frac{\partial \xi_f}{\partial t} &= w_f|_{z=\xi} - \vec{\mathbf{v}}_f|_{z=\xi}.\vec{\nabla}\xi_f \\
\frac{\partial \rho C_s}{\partial t} &= -\vec{\nabla}.(\rho\vec{\mathbf{v}}C_s) + \mathcal{F}_C + \mathcal{D}_C \\
\rho_s &= \rho(T_s, S_s, \xi_f) \\
\rho &= \rho_s + \rho_f
\end{aligned}
$$

The momentum is integrated both in slow and fast modes but the right-hand-side of the equation is split in two parts: a slow part, made of slowly varying terms (advection, Coriolis force, baroclinic pressure force and viscous dissipation), and a fast part, made of fast-varying terms (the surface-induced and compressible pressure force, the weight and the dissipation associated with bulk-viscosity). This momentum equation is numerically integrated twice, once with a large time-step keeping the fast part constant, and once with a smaller time-step keeping the slow part constant. This is much more computationally efficient than integrating the whole set of equations at the same fast time step. More details can be found in Auclair *et al.* [2018].

Note that the solved acoustic waves can become pseudo-acoustic if their phase speed c_s is artificially slowed down (it is a model input). In this case, high-frequency processes associated with bulk compressibility may be unphysical, but a coherent solution for slow non-hydrostatic dynamics is preserved, while the CFL constraint is relaxed.

1.2 Model variables

Model variables are defined in .h Fortran 77 files :

1.2.1 Domain variables (*grid.h*)

grid.h : Environmental two-dimensional arrays associated with curvilinear horizontal coordinate system

h : Model topography (bottom depth [m] at RHO-points.)
dh : Topograhy increment in case of moving bathymetry
f : Coriolis parameter [1/s].
fomn : Compound term, f/[pm*pn] at RHO points.

angler : Angle [radians] between XI-axis and the direction to the EAST at RHO-points.

latr : Latitude (degrees_north) at RHO-, U-, and V-points.
latu
latv
lonr : Longitude (degrees_east) at RHO-, U-, and V-points.
lonu
lonv

xp : XI-coordinates [m] at PSI-points.

xr : XI-coordinates [m] at RHO-points.
yp : ETA-coordinates [m] at PSI-points.
yr : ETA-coordinates [m] at RHO-points.

pm : Coordinate transformation metric "m" [1/meters] associated with the differential distances in XI.
pn : Coordinate transformation metric "n" [1/meters]associated with the differential distances in ETA.
om_u : Grid spacing [meters] in the XI -direction at U-points.
om_v : Grid spacing [meters] in the XI -direction at V-points.
on_u : Grid spacing [meters] in the ETA-direction at U-points.
on_v : Grid spacing [meters] in the ETA-direction at V-points.

dmde : ETA-derivative of inverse metric factor "m", d(1/M)/d(ETA).
dndx : XI-derivative of inverse metric factor "n", d(1/N)/d(XI).

pmon_p : Compound term, pm/pn at PSI-points.
pmon_r : Compound term, pm/pn at RHO-points.
pmon_u : Compound term, pm/pn at U-points.
pnom_p : Compound term, pn/pm at PSI-points.
pnom_r : Compound term, pn/pm at RHO-points.
pnom_v : Compound term, pn/pm at V-points.

rmask : Land-sea masking arrays at RHO-,U-,V- and PSI-points (rmask,umask,vmask) = (0=Land, 1=Sea)
umask
vmask
pmask : pmask=(0=Land, 1=Sea, 1-gamma2 =boundary).

reducu : reduction coefficient along x-axis for rivers sections
reducv : reduction coefficient along y-axis for rivers sections

1.2.2 Barotropic variables (*ocean2d.h*)

ocean2d.h : 2D dynamical variables for fast mode

zeta,rzeta : Free surface elevation [m] and its time tendency;
ubar,rubar : Vertically integrated 2D velocity components in
vbar,rvbar : XI- and ETA-directions and their time tendencies;

1.2.3 Tri-dimensionnal variables (*ocean3d.h*)

ocean3d.h : 3D tracers dynamical variables for baroclinic mode

u,v : 3D velocity components in XI- and ETA-directions
t : tracer array (temperature, salinity, passive tracers, sediment)
Hz : level thickness
z_r : depth at rho point
z_w : depth at w point

Huon : transport a U point
Hvon : transport at V point
We, Wi : vertical velocity (explicit, implicit)
rho : density anomaly
rho1 : potential density at 1 atm

1.2.4 Surface forcing (forces.h)

forces.h :

Surface momemtum flux (wind stress) :

sustr : XI- and ETA-components of kinematic surface momentum flux
svstr : wind stresses) defined at horizontal U- and V-points.dimensioned as [m^2/s^2].

Bottom mometum flux :

bustr : XI- and ETA-components of kinematic bottom momentum flux
bvstr : (drag) defined at horizontal U- and V-points [m^2/s^2].!

Surface tracers fluxes :

stflx : Kinematic surface fluxes of tracer type variables at horizontal RHO-points. Physical dimensions [degC m/s] - temperature; [PSU m/s] - salinity.
dqdt : Kinematic surface net heat flux sensitivity to SST [m/s].
sst : Current sea surface temperature [degree Celsius].
dqdtg : Two-time-level grided data for net surface heat flux
sstg : sensitivity to SST grided data [Watts/m^2/Celsius] and sea surface temperature [degree Celsius].
dqdtp : Two-time-level point data for net surface heat flux
sstp : sensitivity to SST grided data [Watts/m^2/Celsius] and sea surface temperature [degree Celsius].
tsst : Time of sea surface temperature data.
sss : Current sea surface salinity [PSU].
tair : surface air temperature at 2m [degree Celsius].
wsp : wind speed at 10m [degree Celsius].
rhum : surface air relative humidity 2m [fraction]
prate : surface precipitation rate [cm day-1]
radlw : net terrestrial longwave radiation [Watts meter-2]
radsw : net solar shortwave radiation [Watts meter-2]
patm2d : atmospheric pressure above mean seal level
paref : reference pressure to compute inverse barometer effect
srflx : Kinematic surface shortwave solar radiation flux [degC m/s] at horizontal RHO-points

Wind induced waves everything is defined at rho-point :

wfrq : wind-induced wave frequency [rad/s]
uorb : xi-component of wave-induced bed orbital velocity [m/s]
vorb : eta-component of wave-induced bed orbital velocity [m/s]

wdrx : cosine of wave direction [non dimension]
wdre : sine of wave direction [non dimension]
whrm : (RMS) wave height (twice the wave amplitude) [m]
wepb : breaking dissipation rate (epsilon_b term) [m3/s3]
wepd : frictional dissipation rate (epsilon_d term) [m3/s3]
wepr :roller dissipation rate (epsilon_r term) [m3/s3]
wbst : frictional dissipation stress (e_d k/sigma) [m2/s2]

Wave averaged quantities :

brk2dx : xi-direciton 2D breaking dissipation (rho)
brk2de : eta-direction 2D breaking dissipation (rho)
frc2dx : xi-direciton 2D frictional dissipation (rho)
frc2de : eta-direction 2D frictional dissipation (rho)
ust2d : xi-direciton Stokes transport (u-point)
vst2d : eta-direciton Stokes transport (v-point)
sup : quasi-static wave set-up (rho-point)
calP : pressure correction term (rho-point)
Kapsrf : Bernoulli head terrm at the surface (rho-point)
brk3dx : xi-direciton 3D breaking dissipation (rho)
brk3de : eta-direction 3D breaking dissipation (rho)
ust : xi-direciton 3D Stokes drift velocity (u-point)
vst : eta-direciton 3D Stokes drift velocity (v-point)
wst : vertical 3D Stokes drift velocity (rho-point)
Kappa : 3D Bernoulli head term (rho-point)
kvf : vertical vortex force term (K term, 3D, rho-point)
Akb : breaking-wave-induced additional diffusivity (w-point)
Akw : wave-induced additional diffusivity (rho-point)
E_pre : previous time-step value for Akw estimation (rho)
frc3dx : xi-direciton 3D frictional dissipation (rho)
frc3de : eta-direction 3D frictional dissipation (rho)

1.3 Grid and Coordinates

Related CPP options:

CURVGRID	Activate curvilinear coordinate transformation
SPHERICAL	Activate longitude/latitude grid positioning
MASKING	Activate land masking
WET_DRY	Activate wetting-Drying scheme
NEW_S_COORD	Choose new vertical S-coordinates

Preselected options:

```
# define CURVGRID
# define SPHERICAL
# define MASKING
# undef WET_DRY
# undef NEW_S_COORD
```

1.3.1 Vertical Grid parameters

Two vertical transformations are available for the generalized vertical terrain-following vertical system : By default, we have :

$$z(x,y,\sigma,t) = z_0(x,y,\sigma) + \zeta(x,y,t)\left[1 + \frac{z_0(x,y,\sigma)}{h(x,y)}\right] \tag{1.1}$$

$$z_0(x,y,\sigma) = h_c\sigma + [h(x,y) - h_c]\, Cs(\sigma) \tag{1.2}$$

When activating the cpp key `NEW_S_COORD`, we have:

$$z(x,y,\sigma,t) = \zeta(x,y,\sigma) + [\zeta(x,y,t) + h(x,y)]\, z_0(x,y,\sigma) \tag{1.3}$$

$$z_0(x,y,\sigma) = \frac{h_c\sigma + h(x,y)Cs(\sigma)}{h_c + h(x,y)} \tag{1.4}$$

with :

- $z_0(x,y,\sigma)$ a nonlinear vertical transformation
- $\zeta(x,y,\sigma)$ the free-surface
- $h(x,y)$ the ocean bottom
- σ a fractional vertical stretching coordinate, $-1 \leq \sigma \leq 0$
- h_c a positive thickness controlling the stretching
- $Cs(\sigma)$ a nondimensional, monotonic, vertical stretching, $-1 \leq (C\sigma) \leq 0$

Vertical grid stretching is controlled by the following parameters, which have to be set similarly in `croco.in`, and `crocotools_param.m`:

theta_s	Vertical S-coordinate surface stretching parameter. When building the climatology and initial CROCO files, we have to define the vertical grid. Warning! The different vertical grid parameters should be identical in this crocotools_param.m and in croco.in. This is a serious cause of bug.
theta_b	Vertical S-coordinate bottom stretching parameter.
hc	Vertical S-coordinate Hc parameter. It gives approximately the transition depth between the horizontal surfacelevels and the bottom terrain following levels. (Note it should be inferior to hmin in case of Vtransform =1).

Then we have, with N the number of vertical levels:

- with the old transformation :

$$Cs(\sigma) = (1-\theta_b)\frac{sinh(\theta_s\,\sigma)}{sinh(\theta_s)} + \theta_b\left[\frac{0.5\,tanh\left((\sigma+0.5)\,\theta_s\right)}{tanh(0.5\,\theta_s)} - 0.5\right]$$

- with NEW_S_COORD defined :

$$sc = \frac{\sigma - N}{N} \tag{1.5}$$

$$csf = \frac{1. - cosh(\theta_s\,sc)}{cosh(\theta_s) - 1.} \quad \text{if } \theta_b > 0, \quad csf = -sc^2 \quad \text{otherwise} \tag{1.6}$$

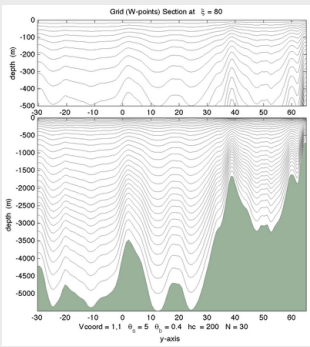
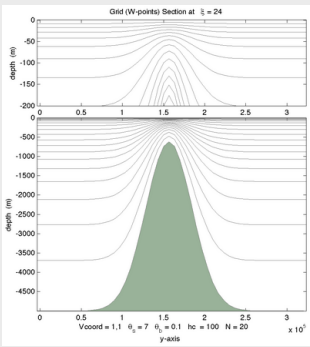
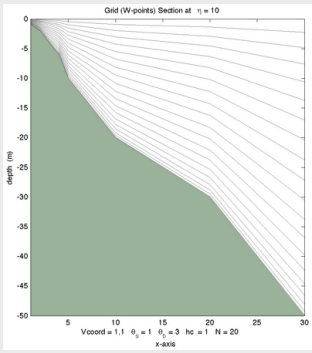
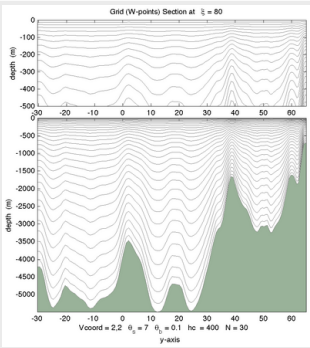
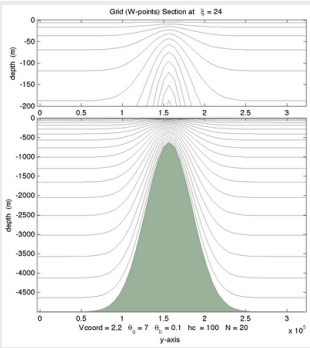
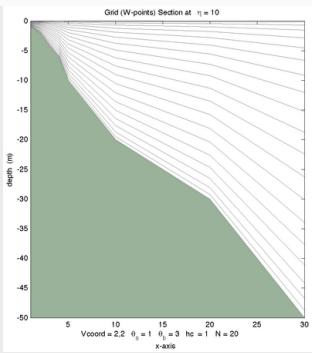
$$Cs(\sigma) = \frac{e^{\theta_b csf} - 1.}{1. - e^{-\theta_b}} \quad \text{if } \theta_s > 0, \quad Cs(\sigma) = csf \quad \text{otherwise} \tag{1.7}$$

Other parameters have to be set to prepare the grid file in `crocotools_param.m`:

vtransform	S-coordinate type (1: old- ; 2: new- coordinates). It is associated to #NEW_S_COORD cpp-keys in CROCO source code.
hmin	Minimum depth in meters. The model depth is cut at this level to prevent, for example, the occurrence of model grid cells without water. This does not influence the masking routines. At lower resolution, hmin should be quite large (for example, 150m for dl=1/2). Otherwise, since topography smoothing is based on, the bottom slopes can be totally eroded.
hmax_coast	Maximum depth under the mask. It prevents selected isobaths (here 500 m) from going under the mask. If this is the case, this could be a source of problems for western boundary currents (for example).
hmax	Maximum depth
rtarget	This variable controls the maximum value of the -parameter that measures the slope of the sigma layers [Beckmann and Haidvogel, 1993] : To prevent horizontal pressure gradient errors, well in terrain-following coordinate models [Haney, 1991], realistic topography requires some smoothing. Empirical results have shown that reliable model results are obtained if it does not exceed 0.2.
n_filter_deep_topo	Number of passes of a Hanning filter to prevent the occurrence of noise and isolated seamounts on deep regions.
n_filter_final	Number of passes of a Hanning filter at the end of the smoothing process to be sure that no noise is present in the topography.

The effects of theta_s, theta_b, hc, and N can be tested using the Matlab script : `croco_tools/Preprocessing_tools/test_vgrid.m`

Below are some examples of different vertical choices (Courtesy of ROMS-RUTGERS team) :

		
Vtransform=1, $\theta_S = 7.0, \theta_B = 0.1$	Vtransform=1, $\theta_S = 7.0, \theta_B = 1.0$	Vtransform=1, $\theta_S = 7.0, \theta_B = 3.0$
		
Vtransform =1, $\theta_S = 7.0, \theta_B = 1.0$	Vtransform=2, $\theta_S = 7.0, \theta_B = 1.0$	Vtransform=2, $\theta_S = 7.0, \theta_B = 3.0$

1.3.2 Grid staggering

The discretization is based on a *staggered grid* where not all variables are stored at the same grid points.

The free-surface (zeta), density (rho), and active/passive tracers (t) are located at the center of the cell whereas the horizontal velocity (u and v) are located at the edges of the cell.

```
   ----  v(i,j+1)  ----
  |                    |
  |                    |
u(i,j)   rho(i,j)   u(i+1,j)
  |                    |
  |                    |
   ----   v(i,j)   ----
```

More information about this and the array indices when using MPI parallelisation is given in *staggered grids*.

1.3.3 Wetting-Drying

The Wetting-Drying scheme is derived from John Warner's code (Rutgers ROMS) and adapted to the time stepping scheme of CROCO. The main idea is to cancel the outgoing momentum flux (not the incoming) from a grid cell if its total depth is below a threshold value (critical depth Dcrit between 5 and 20 cm according to local slope; Dcrit min and max adjustable in param.h). This scheme is tested in the Thacker case producing oscillations in a rotating bowl for which an analytical solution is known.

1.4 Numerics

1.4.1 Overview

CROCO solves the primitive equations in an Earth-centered rotating environment. It is discretized in coastline- and terrain-following curvilinear coordinates using high-order numerical methods. It is a split-explicit, free-surface ocean model, where short time steps are used to advance the surface elevation and barotropic momentum, with a much larger time step used for temperature, salinity, and baroclinic momentum.

The complete time stepping algorithm is described in Shchepetkin and McWilliams [2005]; see also Soufflet *et al.* [2016]. The model has a 2-way time-averaging procedure for the barotropic mode, which satisfies the 3D continuity equation. The specially designed 3rd order predictor-corrector time step algorithm allows a substantial increase in the permissible time-step size.

Combined with the 3rd order time-stepping, a 3rd- or 5th-order, upstream-biased horizontal advection scheme (alternatively WENO or TVD for monotonicity preservation) allows the generation of steep gradients, enhancing the effective resolution of the solution for a given grid size [Soufflet *et al.*, 2016, Shchepetkin and McWilliams, 1998, Ménesguen *et al.*, 2018, Borges *et al.*, 2008]. Because of the implicit diffusion in upstream advection schemes, explicit lateral viscosity is not needed in CROCO for damping numerical dispersion errors.

For vertical advection, SPLINE or WENO5 schemes are proposed (besides lower-order schemes). For SPLINES (default), an option for an adaptive, Courant-number-dependent implicit scheme is propose that has the advantage to render vertical advection unconditionally stable while maintaining good accuracy in locations with small Courant numbers [Shchepetkin, 2015]. This is also available for tracers.

Tracers are treated similarly to momentum. A 3rd- or 5th-order upstream-biased horizontal advection scheme is implemented, but in regional configurations the diffusion part of this scheme is rotated along isopycnal surfaces to avoid spurious diapycnal mixing and loss of water masses [Marchesiello and Estrade, 2009, Lemarié *et al.*, 2012]. For regional/coastal applications, a highly accurate pressure gradient scheme [Shchepetkin and McWilliams, 2003] limits the other type of errors (besides spurious diacpynal mixing) frequently associated with terrain-following coordinate models.

If a lateral boundary faces the open ocean, an active, implicit, upstream biased, radiation condition connects the model solution to the surroundings [Marchesiello *et al.*, 2001]. It comes with sponge layers for a better transition between interior and boundary solutions (explicit Laplacian diffusion and/or newtonian damping)

For nearshore problems, where waves becomes the dominant forcing of circulation, a vortex-force formalism for the interaction of surface gravity waves and currents is implemented in CROCO [Uchiyama *et al.*, 2010].

CROCO can be used either as a Boussinesq/hystrostatic model, or a non-hydrostatic/non-Boussinesq model (NBQ; Auclair *et al.* [2018]). The NBQ solver is relevant in problems from a few tens of meters to LES or DNS resolutions. It comes with shock-capturing advection schemes (WENO5, TVD) and fully 3D turbulent closure schemes (GLS, Smagorinsky).

CROCO includes a variety of additional features, e.g., 1D turbulent closure schemes (KPP, GLS) for surface and benthic boundary layers and interior mixing; wetting and drying; sediment and biological models; AGRIF interface for 2-way nesting; OASIS coupler for ocean-waves-atmosphere coupling…

1.4.2 Time Stepping

CROCO is discretized in time using a third-order predictor-corrector scheme (referred to as LFAM3) for tracers and baroclinic momentum. It is a split-explicit, free-surface ocean model, where short time steps are used to advance the surface elevation and barotropic momentum, with a much larger time step used for tracers, and baroclinic momentum. The model has a 2-way time-averaging procedure for the barotropic mode, which satisfies the 3D continuity equation. The specially designed 3rd order predictor-corrector time step algorithm is described in Shchepetkin and McWilliams [2005] and is summarized in this subsection.

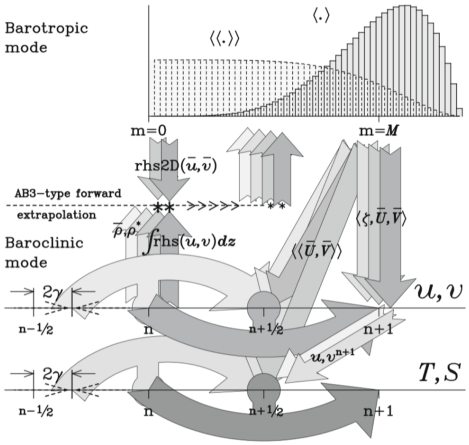


Fig. 1: Fig: schematic view of the Croco predictor-corrector hydrostatic kernel

General structure of the time-stepping:

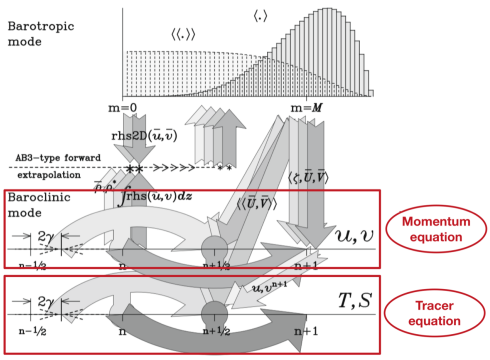
```
call prestep3D_thread()    ! Predictor step for 3D momentum and tracers
call step2d_thread()       ! Barotropic mode
call step3D_uv_thread()   ! Corrector step for momentum
call step3D_t_thread()    ! Corrector step for tracers
```

1.4.2.1 3D momentum and tracers

Predictor-corrector approach : **Leapfrog (LF) predictor with 3rd-order Adams-Moulton (AM) interpolation (LFAM3 timestepping)**. This scheme is used to integrate 3D advection, the pressure gradient term, the continuity equation and the Coriolis term which are all contained in the RHS operator (the time tendencies).

For a given quantity q with $q_t = \mathrm{RHS}(q)$ we can write

$$\left\{\begin{array}{rcll} q^{n+1,\star} & = & q^{n-1} + 2\Delta t\ \mathrm{RHS}\left\{q^n\right\} & \text{(LF)} \\ q^{n+\frac{1}{2}} & = & \dfrac{5}{12}\ q^{n+1,\star} + \dfrac{2}{3}\ q^n - \dfrac{1}{12}\ q^{n-1} & \text{(AM3)} \\ q^{n+1} & = & q^n + \Delta t\ \mathrm{RHS}\left\{q^{n+\frac{1}{2}}\right\} & \text{(corrector)} \end{array}\right.$$



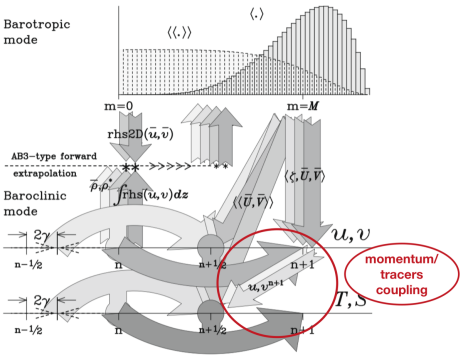
which can be rewritten in a compact way as used in the Croco code :

$$
\begin{aligned}
q^{n+\frac{1}{2}} &= \left(\frac{1}{2}-\gamma\right) q^{n-1} + \left(\frac{1}{2}+\gamma\right) q^{n} + (1-\gamma)\Delta t \text{ RHS}\left\{q^{n}\right\} \\
q^{n+1} &= q^{n} + \Delta t \text{ RHS}\left\{q^{n+\frac{1}{2}}\right\}
\end{aligned}
$$

with $\gamma = \frac{1}{6}$.

Physical parameterizations for vertical mixing, rotated diffusion and viscous/diffusion terms are computed once per time-step using an Euler step.

1.4.2.2 Tracers-momentum coupling



The numerical integration of internal waves can be studied using the following subsystem of equations

$$
\left\{
\begin{aligned}
\partial_z w + \partial_x u &= 0 \\
\partial_z p + \rho g &= 0 \\
\partial_t u + \frac{1}{\rho_0}\partial_x p &= 0 \\
\partial_t \rho + \partial_z (w\rho) &= 0
\end{aligned}
\right.
$$

Predictor step:

$$
\begin{aligned}
\partial_x p^n = g\,\partial_x\left(\int_z^0 \rho^n dz\right) \quad &\rightarrow \quad u^{n+\frac{1}{2}} = \left(\frac{1}{2}-\gamma\right) u^{n-1} + \left(\frac{1}{2}+\gamma\right) u^{n} + (1-\gamma)\frac{\Delta t}{\rho_0}\left(\partial_x p^n\right) \\
w^n = -\int_{-H}^{z} \partial_x u^n dz' \quad &\rightarrow \quad \rho^{n+\frac{1}{2}} = \left(\frac{1}{2}-\gamma\right) \rho^{n-1} + \left(\frac{1}{2}+\gamma\right) \rho^{n} + (1-\gamma)\Delta t\,\partial_z(w^n\rho^n)
\end{aligned}
$$

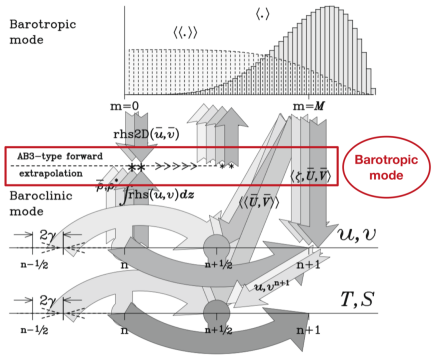
Corrector step:

$$
\begin{aligned}
\partial_x p^{n+\frac{1}{2}} = g\,\partial_x\left(\int_z^0 \rho^{n+\frac{1}{2}} dz\right) \quad &\rightarrow \quad u^{n+1} = u^n + \frac{\Delta t}{\rho_0}\left(\partial_x p^{n+\frac{1}{2}}\right) \\
w^{n+\frac{1}{2}} = -\int_{-H}^{z} \partial_x \left\{\frac{3u^{n+\frac{1}{2}}}{4} + \frac{u^n + u^{n+1}}{8}\right\} dz' \quad &\rightarrow \quad \rho^{n+1} = \rho^n + \Delta t\,\partial_z(w^{n+\frac{1}{2}}\rho^{n+\frac{1}{2}})
\end{aligned}
$$

Consequences:

- 3D-momentum integrated before the tracers in the corrector
- 2 evaluations of the pressure gradient per time-step
- 3 evaluations of the continuity equation per time-step

1.4.2.3 Barotropic mode



Generalized forward-backward (predictor-corrector)

1. AB3-type extrapolation

$$
\begin{aligned}
D^{m+\frac{1}{2}} &= H + \left(\frac{3}{2}+\beta\right)\zeta^{m} - \left(\frac{1}{2}+2\beta\right)\zeta^{m-1} + \beta\zeta^{m-2} \\
\overline{u}^{m+\frac{1}{2}} &= \left(\frac{3}{2}+\beta\right)\overline{u}^{m} - \left(\frac{1}{2}+2\beta\right)\overline{u}^{m-1} + \beta\overline{u}^{m-2}
\end{aligned}
$$

2. Integration of ζ

$$\zeta^{m+1} = \zeta^{m} - \Delta\tau\ \partial_x(D^{m+\frac{1}{2}}\overline{u}^{m+\frac{1}{2}})$$

3. AM4 interpolation

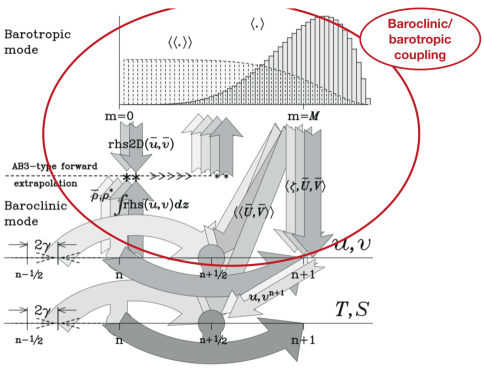
$$\zeta^{\star} = \left(\frac{1}{2}+\gamma+2\varepsilon\right)\zeta^{m+1} + \left(\frac{1}{2}-2\gamma-3\varepsilon\right)\zeta^{m} + \gamma\zeta^{m-1} + \varepsilon\zeta^{m-2}$$

4. Integration of $\overline{u}$

$$\overline{u}^{m+1} = \frac{1}{D^{m+1}}\left[D^{m}\overline{u}^{m} + \Delta\tau\ \mathrm{RHS2D}(D^{m+\frac{1}{2}}, \overline{u}^{m+\frac{1}{2}}, \zeta^{\star})\right]$$

where the parameter values are $(\beta,\gamma,\varepsilon) = (0.281105, 0.088, 0.013)$ except when the filter_none option is activated (see below).

1.4.2.4 Baroclinic-barotropic coupling

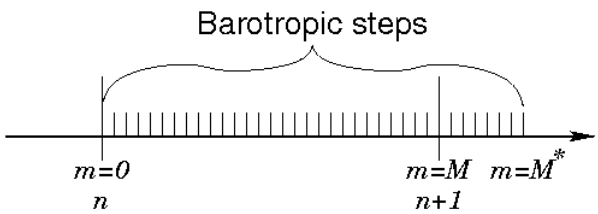


Slow forcing term of the barotrope by the barocline is extrapolated

$$\mathcal{F}_{3D}^{n+\frac{1}{2}} = \left\{ \int \mathrm{rhs}(u,v)dz - \mathrm{rhs2D}(\bar{u},\bar{v}) \right\}^{n+\frac{1}{2}} = \mathrm{Extrap}(\mathcal{F}_{3D}^{n}, \mathcal{F}_{3D}^{n-1}, \mathcal{F}_{3D}^{n-2})$$

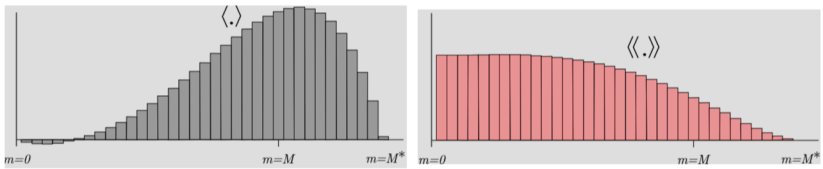
1.4.2.4.1 M2_FILTER_POWER option

Barotropic integration from n to $n + M^* \Delta\tau$ ($M^* \leq 1.5M$)



Because of predictor-corrector integration two barotropic filters are needed

- $\langle \zeta \rangle^{n+1} \rightarrow$ update of the vertical grid
- $\langle U \rangle^{n+1} \rightarrow$ correction of baroclinic velocities at time $n+1$
- $\langle\langle U \rangle\rangle^{n+\frac{1}{2}} \rightarrow$ correction of baroclinic velocities at time $n + \frac{1}{2}$



1.4.2.4.2 M2_FILTER_NONE option

Motivation: averaging filters can lead to excessive dissipation in the barotropic mode

Objective: put the minimum amount of dissipation to stabilize the splitting

Diffusion is introduced within the barotropic time-stepping rather than averaging filters by adapting the parameters in the generalized forward-backward scheme

$$(\beta, \gamma, \varepsilon) = (0.281105, 0.08344500 - 0.51358400\alpha_{\mathrm{d}}, 0.00976186 - 0.13451357\alpha_{\mathrm{d}})$$

with $\alpha_{\mathrm{d}} \approx 0.5$.

Remarks:

- This option may require to increase $\mathrm{NDTFAST} = \Delta t_{3\mathrm{D}}/\Delta t_{2\mathrm{D}}$ because the stability constraint of the modified generalized forward-backward scheme is less than the one of the original generalized forward-backward scheme.
- The filter_none approach is systematically more efficient than averaging filters

1.4.2.5 Stability constraints

- **Barotropic mode** (note that considering an Arakawa C-grid divides the theoretical stability limit by a factor of 2)

$$\Delta t \sqrt{gH\left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}\right)} \leq 0.89$$

- **3D advection**

$$\alpha^x_{\rm adv} + \alpha^y_{\rm adv} + \beta\alpha^z_{\rm adv} \leq \alpha^\star_{\rm horiz}$$

where $\alpha^x_{\rm adv}$, $\alpha^y_{\rm adv}$, and $\alpha^z_{\rm adv}$ are the Courant numbers in each direction and $\beta = \alpha^\star_{\rm horiz}/\alpha^\star_{\rm vert}$ a coefficient arising from the fact that different advection schemes with different stability criteria may be used in the horizontal and vertical directions. Typical CFL values for $\alpha^\star_{\rm horiz}$ and $\alpha^\star_{\rm vert}$ with Croco time-stepping algorithm are

Advection scheme	Max Courant number ($\alpha^\star$)
C2	1.587
UP3	0.871
SPLINES	0.916
C4	1.15
UP5	0.89
C6	1.00

- **Internal waves**

$$\Delta t c_1 \sqrt{\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}} \leq 0.843686$$

where c_1 the phase speed associated with the first (fastest) baroclinic mode.

- **Coriolis**

$$f\Delta t \leq 1.58$$

1.4.3 Advection Schemes

1.4.3.1 Lateral Momentum Advection

Related CPP options:

UV_HADV_UP3	Activate 3rd-order upstream biased advection scheme
UV_HADV_UP5	Activate 5th-order upstream biased advection scheme
UV_HADV_C2	Activate 2nd-order centred advection scheme (should be used with explicit momentum mixing)
UV_HADV_C4	Activate 4th-order centred advection scheme (should be used with explicit momentum mixing)
UV_HADV_C6	Activate 6th-order centred advection scheme (should be used with explicit momentum mixing)
UV_HADV_WENO5	Activate WENO 5th-order advection scheme
UV_HADV_TVD	Activate Total Variation Diminushing scheme

Preselected options:

```
# define UV_HADV_UP3
# undef  UV_HADV_UP5
# undef  UV_HADV_C2
# undef  UV_HADV_C4
# undef  UV_HADV_C6
# undef  UV_HADV_WENO5
# undef  UV_HADV_TVD
```

These options are set in set_global_definitions.h as the default UV_HADV_UP3 is the only one recommended for standard users.

1.4.3.2 Lateral Tracer advection

Related CPP options:

TS_HADV_UP3	3rd-order upstream biased advection scheme
TS_HADV_RSUP3	Split and rotated 3rd-order upstream biased advection scheme
TS_HADV_UP5	5th-order upstream biased advection scheme
TS_HADV_RSUP5	Split and rotated 5th-order upstream biased advection scheme with reduced dispersion/diffustion
TS_HADV_C4	4th-order centred advection scheme
TS_HADV_C6	Activate 6th-order centred advection scheme
TS_HADV_WENO5	5th-order WENOZ quasi-monotonic advection scheme for all tracers
BIO_HADV_WENO5	5th-order WENOZ quasi-monotone advection scheme for passive tracers (including biology and sediment tracers)

Preselected options:

```
# undef  TS_HADV_UP3
# define TS_HADV_RSUP3
# undef  TS_HADV_UP5
# undef  TS_HADV_RSUP5
# undef  TS_HADV_C4
# undef  TS_HADV_C6
# undef  TS_HADV_WENO5
# if defined PASSIVE_TRACER || defined BIOLOGY || defined SEDIMENT
#  define BIO_HADV_WENO5
# endif
```

TS_HADV_RSUP3 is recommended for realistic applications with variable bottom topography as it strongly reduces diapycnal mixing. It splits the UP3 scheme into 4th-order centered advection and rotated bilaplacian diffusion with grid-dependent diffusivity. It calls for CPP options in set_global_definitions.h for the explicit treatment of bilaplacian diffusion (see below). TS_HADV_RSUP3 is expensive in terms of computational cost and requires more than 30 sigma levels to perform properly. Therefore, for small domains dominated by open boundary fluxes, TS_HADV_UP5 may present a cheaper alternative and good compromise. TS_HADV_RSUP5 is still experimental but allows a decrease in numerical diffusivity compared to TS_HADV_RSUP3 by using 6th order rather than 4th-order centered advection (it resembles in spirit a split-rotated UP5 scheme but the use of bilaplacian rather than trilaplacian diffusion keeps it 3rd order). TS_HADV_C4 has no implicit diffusion and is thus accompanied by rotated Smagorinsky diffusion defined in set_global_definitions.h; it is not recommended for usual applications. For RSUP family, by default the diffusive part is oriented along geopotential.

1.4.3.3 Vertical Momentum advection

Related CPP options:

UV_VADV_SPLINES	4th-order compact advection scheme
UV_VADV_C2	2nd-order centered advection scheme
UV_VADV_WENO5	5th-order WENOZ quasi-monotone advection scheme
UV_VADV_TVD	Total Variation Diminushing (TVD) scheme

Preselected options:

```
#ifdef UV_VADV_SPLINES
#elif defined UV_VADV_WENO5
#elif defined UV_VADV_C2
#elif defined UV_VADV_TVD
#else
# define UV_VADV_SPLINES
# undef  UV_VADV_WENO5
# undef  UV_VADV_C2
# undef  UV_VADV_TVD
#endif
```

1.4.3.4 Vertical Tracer advection

Related CPP options:

TS_VADV_SPLINES	4th-order compact advection scheme
TS_VADV_AKIMA	4th-order centered advection scheme with harmonic averaging
TS_VADV_C2	2nd-order centered advection scheme
TS_VADV_WENO5	5th-order WENOZ quasi-monotone advection scheme

Preselected options:

```
#ifdef TS_VADV_SPLINES
#elif defined TS_VADV_AKIMA
#elif defined TS_VADV_WENO5
#elif defined TS_VADV_C2
#else
# undef  TS_VADV_SPLINES
# define TS_VADV_AKIMA
# undef  TS_VADV_WENO5
# undef  TS_VADV_C2
#endif
```

1.4.3.5 Adaptively implicit vertical advection

Related CPP options:

VADV_ADAPT_IMP	Activate adapative, Courant number dependent implicit advection scheme
VADV_ADAPT_PRED	Adaptive treatment at both predictor and corrector steps

Preselected options:

```
#ifdef VADV_ADAPT_IMP
# undef  VADV_ADAPT_PRED
```

(continues on next page)

(continued from previous page)

```
# define UV_VADV_SPLINES
# undef  UV_VADV_C2
# undef  UV_VADV_WENO5
# undef  UV_VADV_TVD
#endif
#ifdef VADV_ADAPT_IMP
# define  TS_VADV_SPLINES
# undef   TS_VADV_AKIMA
# undef   TS_VADV_WENO5
# undef   TS_VADV_C2
#endif
```

1.4.3.6 Numerical details on advection schemes

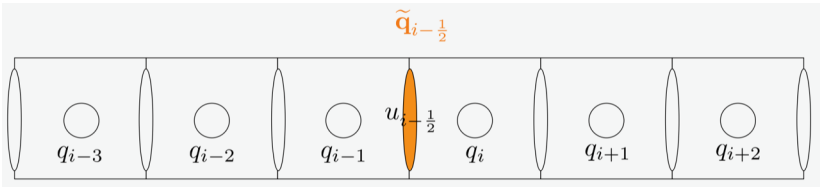


Fig. 2: Fig: variable location on an Arakawa C-grid. Tracer values are cell centered while velocities are defined on interfaces.

$$\partial_x(uq)|_{x=x_i} = \frac{1}{\Delta x_i}\left\{u_{i+\frac{1}{2}}\widetilde{q}_{i+\frac{1}{2}} - u_{i-\frac{1}{2}}\widetilde{q}_{i-\frac{1}{2}}\right\}$$

1.4.3.6.1 Linear advection schemes

$$\widetilde{q}^{\mathrm{C2}}_{i-\frac{1}{2}} = \frac{q_i + q_{i-1}}{2} \tag{1.8}$$

$$\widetilde{q}^{\mathrm{C4}}_{i-\frac{1}{2}} = \left(\frac{7}{6}\right)\widetilde{q}^{\mathrm{C2}}_{i-\frac{1}{2}} - \left(\frac{1}{12}\right)(q_{i+1} + q_{i-2}) \tag{1.9}$$

$$\widetilde{q}^{\mathrm{UP3}}_{i-\frac{1}{2}} = \widetilde{q}^{\mathrm{C4}}_{i-\frac{1}{2}} + \mathrm{sign}\left(\frac{1}{12}, u_{i-\frac{1}{2}}\right)(q_{i+1} - 3q_i + 3q_{i-1} - q_{i-2}) \tag{1.10}$$

$$\widetilde{q}^{\mathrm{C6}}_{i-\frac{1}{2}} = \left(\frac{8}{5}\right)\widetilde{q}^{\mathrm{C4}}_{i-\frac{1}{2}} - \left(\frac{19}{60}\right)\widetilde{q}^{\mathrm{C2}}_{i-\frac{1}{2}} + \left(\frac{1}{60}\right)(q_{i+2} + q_{i-3}) \tag{1.11}$$

$$\widetilde{q}^{\mathrm{UP5}}_{i-\frac{1}{2}} = \widetilde{q}^{\mathrm{C6}}_{i-\frac{1}{2}} - \mathrm{sign}\left(\frac{1}{60}, u_{i-\frac{1}{2}}\right)(q_{i+2} - 5q_{i+1} + 10q_i - 10q_{i-1} + 5q_{i-2} - q_{i-3}) \tag{1.12}$$

1.4.3.6.2 Split upwind schemes

Because odd-ordered advection schemes can be formulated as the sum of the next higher-order (centered) advection scheme with a dissipation term it is possible to split the purely centered and dissipative parts of UP3 and UP5 schemes. In this case the centered part is treated within the predictor-corrector framework while the flow-dependent dissipative part is treated with a one-step Euler scheme. Such splitting has two advantages:

1. It allows better stability for SUP3 and SUP5 schemes comapared to UP3 and UP5 schemes.
2. Isolating the dissipative part allows to rotate it in the neutral direction to reduce spurious diapycnal mixing (RSUP3 scheme).

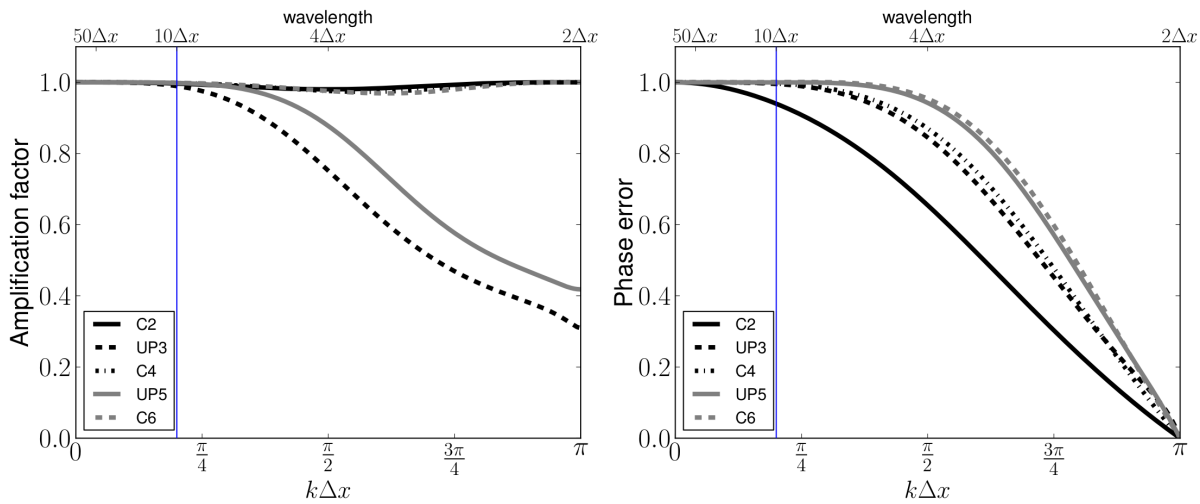


Fig. 3: Fig: amplification errors (left) and phase errors (right) for linear advection of order 2 to 6.

1.4.3.6.3 Splines reconstruction and Akima 4th-order schemes

Similar to a 4th-order compact scheme, the interfacial values for the splines reconstruction scheme are obtained as a solution of a tridiagonal problem

$$\mathrm{Hz}_{k+1}\widetilde{q}_{k-\frac{1}{2}} + 2(\mathrm{Hz}_k + \mathrm{Hz}_{k+1})\widetilde{q}_{k+\frac{1}{2}} + \mathrm{Hz}_k\widetilde{q}_{k+\frac{3}{2}} = 3(\mathrm{Hz}_k\overline{q}_{k+1} + \mathrm{Hz}_{k+1}\overline{q}_k)$$

where $\overline{q}_k$ values should be understood in a finite-volume sense (i.e. as an average over a control volume).

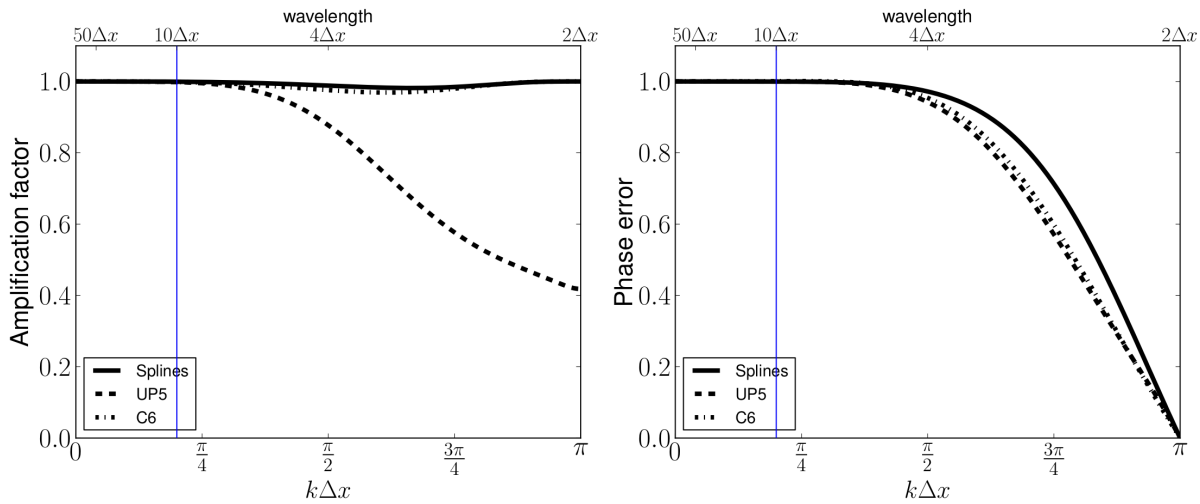


Fig. 4: Fig: amplification errors (left) and phase errors (right) for linear advection of order 5 and 6 and for Splines reconstruction.

The AKIMA scheme corresponds to a 4th-order accurate scheme where an harmonic averaging of the slopes is used instead of the algebraic average used for a standard C4 scheme

$$\widetilde{q}_{k+\frac{1}{2}} = \frac{q_{k+1}+q_k}{2} - \frac{\overline{\delta q}_{k+1} - \overline{\delta q}_k}{6} \qquad \overline{\delta q}_k = \begin{cases} 2\dfrac{\delta q_{k+\frac{1}{2}}\delta q_{k-\frac{1}{2}}}{\delta q_{k+\frac{1}{2}} + \delta q_{k-\frac{1}{2}}}, & \text{if } \delta q_{k+\frac{1}{2}}\delta q_{k-\frac{1}{2}} > 0 \\ 0, & \text{otherwise} \end{cases}$$

1.4.3.6.4 Adaptively implicit vertical advection

Idea: the vertical velocity Ω is split between an explicit and implicit contribution depending on the local Courant number

$$\Omega = \mathbf{\Omega^{(e)}} + \mathbf{\Omega^{(i)}}, \qquad \mathbf{\Omega^{(e)}} = \frac{\Omega}{f(\alpha^z_{\rm adv}, \alpha_{\rm max})}, \quad f(\alpha^z_{\rm adv}, \alpha_{\rm max}) = \left\{ \begin{array}{ll} 1, & \alpha^z_{\rm adv} \leq \alpha_{\rm max} \\ \alpha/\alpha_{\rm max}, & \alpha^z_{\rm adv} > \alpha_{\rm max} \end{array} \right.$$

- $\Omega^{(e)}$ is integrated with an explicit scheme with CFL $\alpha_{\rm max}$.
- $\Omega^{(i)}$ is integrated with an implicit upwind Euler scheme.
- $f(\alpha^z_{\rm adv}, \alpha_{\rm max})$ is a function responsible for the splitting of Ω between an explicit and an implicit part.

This approach has the advantage to render vertical advection unconditionally stable and to maintain good accuracy in locations with small Courant numbers. The current implementation is based on the SPLINES scheme for the explicit part.

1.4.3.6.5 Total variation bounded scheme (WENO5)

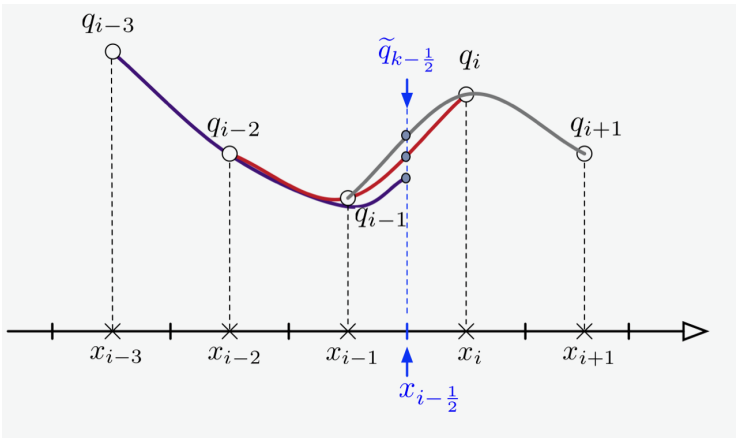


Fig. 5: Fig: different stencils used to evaluate the interfacial value $\widetilde{q}_{k+\frac{1}{2}}$ with WENO5 scheme

Nonlinear weighting between 3 evaluations of interfacial values based on 3 different stencils

$$\widetilde{q}_{k-\frac{1}{2}} = w_0 \widetilde{q}^{(0)}_{k-\frac{1}{2}} + w_1 \widetilde{q}^{(1)}_{k-\frac{1}{2}} + w_2 \widetilde{q}^{(2)}_{k-\frac{1}{2}}$$

where the weights are subject to the following constraints:

1. Convexity $\sum_{j=0}^{2} w_j = 1$.
2. ENO property (Essentially non oscillatory).
3. 5th-order if $q(x)$ is smooth.

The resulting scheme is not monotonicity-preserving but instead it is Total Variation Bounded (TVB).

1.4.3.6.6 Total variation diminishing scheme

1.4.3.6.7 Upwinding of nonlinear terms

In CROCO the nonlinear advection terms are formulated as in Lilly (1965) :

$$\partial_t(\mathrm{Hz}u) \quad + \quad \partial_x\left((\mathrm{Hz}\ u)u\right) + \partial_y\left((\mathrm{Hz}\ v)\ u\right) + \ \ldots \tag{1.13}$$

$$\partial_t(\mathrm{Hz}v) \quad + \quad \partial_x\left((\mathrm{Hz}\ u)v\right) + \partial_y\left((\mathrm{Hz}\ v)\ v\right) + \ \ldots \tag{1.14}$$

which are discretised with third order accuracy as

$$\left(\widetilde{(\mathrm{Hz}\ u)u}\right)_{i,j} \quad = \quad \widetilde{(\mathrm{Hz}\ u)}^{\rm C4}_{i,j}\ \widetilde{u}^{\rm UP3}_{i,j} \tag{1.15}$$

$$\left(\widetilde{(\mathrm{Hz}\ v)u}\right)_{i+\frac{1}{2},j+\frac{1}{2}} \quad = \quad \widetilde{(\mathrm{Hz}\ v)}^{\rm C4}_{i+\frac{1}{2},j+\frac{1}{2}}\ \widetilde{u}^{\rm UP3}_{i+\frac{1}{2},j+\frac{1}{2}} \tag{1.16}$$

where the direction for upwinding is selected considering

$$u^{\mathrm{upw}}_{i,j} = u_{i+\frac{1}{2},j} + u_{i-\frac{1}{2},j}, \qquad v^{\mathrm{upw}}_{i+\frac{1}{2},j+\frac{1}{2}} = (\mathrm{Hz}\ v)_{i,j+\frac{1}{2}} + (\mathrm{Hz}\ v)_{i+1,j+\frac{1}{2}}$$

1.4.4 Pressure gradient

This section is still under redaction. Meanwhile, please refer to Shchepetkin and McWilliams [2003].

1.4.5 Equation of State

Related CPP options:

SALINITY	Activate salinity as an active tracer
NONLIN_EOS	Activate nonlinear equation of state
SPLIT_EOS	Activate the split of the nonlinear equation of state in adiabatic and compressible parts for reduction of pressure gradient errors

Preselected options:

```
# define SALINITY
# define NONLIN_EOS
# define SPLIT_EOS
```

The density is obtained from temperature and salinity (if SALINITY defined) via a choice of linear $\rho(T)$ or non-linear $\rho(T, S, P)$ equation of state (EOS) described in Shchepetkin and McWilliams [2003]. The nonlinear EOS corresponds to the UNESCO formulation as derived by Jackett and Mcdougall [1995] that computes in situ density as a function of potential temperature, salinity and pressure.

To reduce errors of pressure-gradient scheme associated with nonlinearity of compressibility effects, Shchepetkin and McWilliams [2003] introduced a Taylor expansion of this EOS that splits it into an adiabatic and a linearized compressible part (SPLIT_EOS):

$$\rho = \rho_0 + \rho_1(T, S) + q_1(T, S)\ |z|$$

where $\rho_1(T, S)$ is the sea-water density perturbation at the standard pressure of 1 Atm (sea surface), q_1 is the compressibility coefficient, and $|z|$ is absolute depth, i.e. the distance from free-surface to the point at which density is computed. This splitting of the EOS into two separate contributions allows for the representation of spatial derivatives of density as the sum of adiabatic derivatives and the compressible part. This makes it straightforward to remove pressure effects so as to reduce pressure gradient errors, compute neutral directions, enforce stable stratification, compute Brunt-Väisäla frequency etc.

The Brunt-Väisäla frequency N (at horizontal ρ and vertical w points) is defined by:

$$N^2 = -\frac{g}{\rho_0}\frac{\partial \rho_\theta}{\partial z}$$

where ρ_θ is potential density, .i.e., the density that a parcel would acquire if adiabatically brought to depth z_w.

1.4.6 Wetting and Drying

The processes of wetting and drying have important physical and biological impacts on shallow water systems. Flooding and dewatering effects on coastal mud flats and beaches occur on various time scales ranging from storm surge, periodic rise and fall of the tide, to infragravity wave motions. To correctly simulate these physical processes with a numerical model requires the capability of the computational cells to become flooded and dewatered. Warner *et al.* [2013] proposed a method for wetting and drying based on an approach consistent with a cell-face blocking

algorithm. The method allows water to always flow into any cell, but prevents outflow from a cell when the total depth in that cell is less than a user defined critical value. See Warner *et al.* [2013] for details.

The Wetting-Drying scheme is derived from John Warner's code (Rutgers ROMS) and adapted to the time stepping scheme of CROCO. The main idea is to cancel the outgoing momentum flux (not the incoming) from a grid cell if its total depth is below a threshold value (critical depth Dcrit between 5 and 20 cm according to local slope; Dcrit min and max adjustable in param.h). This scheme is tested in the Thacker case producing oscillations in a rotating bowl for which an analytical solution is known.

1.4.7 Non-Boussinesq Solver

CROCO can be used in a Boussinesq hystrostatic mode, or a non-hydrostatic, non-boussinesq mode (NBQ). The Non-Hydrostatic approach is based on the relaxation of the Boussinesq approximation instead of solving a Poisson system. It replaces the barotropic mode solver by a fully 3D fast mode solver, resolving all waves down to acoustic waves. The barotropic mode is part of the fast mode in this case. Depending on the physical problem, the sound speed can be decreased to the maximum wave velocity one wants to solve. The NH solver can be used in problems from a few tens of meters to LES or DNS resolutions. It comes with monotonicity preserving advection schemes (WENO5, TVD) and fully 3D turbulent closure schemes.

Related CPP options (for users):

NBQ	Activates Non-hysrostatic, non-Boussinesq solver

1.5 Parametrizations

1.5.1 Vertical mixing parametrizations

CROCO contains a variety of methods for setting the vertical viscous and diffusive coefficients. The choices range from simply choosing fixed values to the KPP and the generic lengthscale (GLS) turbulence closure schemes. See Large [1998] for a review of surface ocean mixing schemes. Many schemes have a background molecular value which is used when the turbulent processes are assumed to be small (such as in the interior).

Related CPP options:

ANA_VMIX	Analytical definition
BVF_MIXING	Brunt-Vaisaleafrequency based
LMD_MIXING	K-profile parametrisation
GLS_MIXING	Generic lengthscale parametrisation

Preselected options:

```
NONE : default is no mixing scheme
```

1.5.1.1 Analytical definition

Related CPP options:

ANA_VMIX	Analytical definition

Preselected options:

```
NONE
```

A profile for mixing coeefficient $K_{m,s}(z)$ can be set in ana_vmix routine for variables Akv (viscosity) and Akt (diffusivity), which is called at each time step. In this case, background coeeficients read in croco.in can be used.

1.5.1.2 BVF mixing

Related CPP options:

BVF_MIXING	Brunt-Vaisala frequency based

Preselected options:

```
NONE
```

It computes diffusivity using a Brunt-Vaisala frequency based vertical mixing scheme. Viscosity is set to its background. In static unstable regime, diffusivity is enhanced.

- If $N^2(z) < 0$:

$$K_{m,s}(z) = 0.1 \text{ m}^2 \text{ s}^{-1}$$

- If $N^2(z) > 0$:

$$K_{m,s}(z) = 10^{-7}/\sqrt{N^2(z)}, \qquad K_{m,s}^{\min} \leq K_{m,s}(z) \leq K_{m,s}^{\max}$$

Default bounds are quite restrictive :

$$K_{m,s}^{\min} = 3 \times 10^{-5} \text{ m}^2 \text{ s}^{-1}, \qquad K_{m,s}^{\max} = 4 \times 10^{-4} \text{ m}^2 \text{ s}^{-1}$$

1.5.1.3 K-profile parametrization

Large *et al.* [1994]

Related CPP options:

KPP-related options :

LMD_MIXING	K-profile parametrisation
LMD_SKPP	Activate surface boundary layer KPP mixing
LMD_SKPP2005	Activate surface boundary layer KPP mixing (2005 version)
LMD_BKPP	Activate bottom boundary layer KPP mixing
LMD_BKPP2005	Activate bottom boundary layer KPP mixing (2005 version)
LMD_RIMIX	Activate shear instability interior mixing
LMD_CONVEC	Activate convection interior mixing
LMD_DDMIX	Activate double diffusion interior mixing
LMD_NONLOCAL	Activate nonlocal transport for SKPP
LMD_LANGMUIR	Activate Langmuir turbulence mixing

Preselected options:

```
# ifdef LMD_MIXING
#  define LMD_SKPP
#  define LMD_BKPP
#  define LMD_RIMIX
#  define LMD_CONVEC
#  undef  LMD_DDMIX
#  define LMD_NONLOCAL
#  undef  LMD_LANGMUIR
# endif
```

#if defined LMD_SKPP # define LMD_SKPP2005 #endif #ifdef LMD_BKPP # undef LMD_BKPP2005 #endif

Surface boundary layer

- LMD_SKPP [Large *et al.*, 1994]
  - Step 1 : Compute boundary layer depth $h_{bl}(z_r \to z_N)$

$$\mathrm{Ri}_b(z) = \frac{g(z_r - z)\,(\rho(z) - \rho_r)\,/\rho_0}{|\mathbf{u}(z) - (\mathbf{u}_h)_r|^2 + V_t^2(z)}, \qquad \mathrm{Ri}_b(-h_{bl}) = \mathrm{Ri}_{cr}$$

  - Step 2 : In the stable case math::*(B_f > 0) : h_{bl} = min(h_{bl}, h_{ek}, h_{mo})*

$$h_{ek} = 0.7 u_\star / f, \qquad h_{mo} = u_\star^3/(\kappa B_f).$$

  - Step 3 : Compute turbulent viscosity and diffusivity

$$K_{m,s}(z) = w_{m,s}\; h_{bl}\; G(z/h_{bl}), \qquad w_{m,s} = \kappa\; u_\star\; \psi_{m,s}(z B_f/u_\star^3)$$

Choice of the critical Richardson number Ri_{cr} : $\mathrm{Ri}_{cr} \in [0.15, 0.45]$

- LMD_SKPP2005 [Shchepetkin and McWilliams, 2005]
  - Criteria for h_{bl} : integral layer where production of turbulence by shear balances dissipation by the stratification

$$\mathrm{Cr}(z) = \int_z^\zeta \mathcal{K}(z') \left\{ |\partial_{z'} \mathbf{u}_h|^2 - \frac{N^2}{\mathrm{Ri}_{cr}} - C_{Ek}\; f^2 \right\} dz' + \frac{V_t^2(z)}{(\zeta - z)}, \; \mathrm{Cr}(-h_{bl}) = 0$$

  - Consistent with the original KPP

$$\mathrm{Cr}(-h_{bl}) = 0 \Rightarrow \frac{(\zeta - z) \int_z^\zeta \mathcal{K}(z') N^2(z') dz'}{(\zeta - z) \int_z^\zeta \mathcal{K}(z') \left\{ |\partial_z \mathbf{u}_h|^2 - C_{Ek}\; f^2 \right\} dz' + V_t^2(z)} = \mathrm{Ri}_{cr}$$

Advantages :
-> consistent with Ekman problem
-> tends to give deeper boundary layers : $(\zeta - z) \int_z^\zeta |\partial_{z'} \mathbf{u}_h|^2\, dz' \geq |\mathbf{u}_h(z) - \mathbf{u}_h(\zeta)|^2$.

- cpp key LMD_LANGMUIR [McWilliams and Sullivan, 2000]

Following the work of McWilliams and Sullivan [2000], we introduce in KPP an enhancement factor E to the turbulent velocity scale as a function of the turbulent Langmuir number $La_t = \sqrt{u_\star / u_{Stokes}}$, but this function is taken as in Van Roekel *et al.* [2012] which gives good results in Li *et al.* [2016] – still assuming that Stokes drift is aligned with the surface wind stress:

$$w_{m,s} = \frac{\kappa\; u_\star}{\phi_{m,s}}\; E, \qquad E = \sqrt{1 + 0.104 La_t^{-2} + 0.034 La_t^{-4}}$$

Interior scheme

$$K_{m,s}(z) = K_{m,s}^{\mathrm{sh}}(z) + K_{m,s}^{\mathrm{iw}}(z) + K_{m,s}^{\mathrm{dd}}(z)$$

- cpp key LMD_RIMIX, RI_(H-V)SMOOTH [Large *et al.*, 1994]

$$\mathrm{Ri}_g = N^2 / \left[(\partial_z u)^2 + (\partial_z v)^2 \right]$$

$$K_{m,s}^{\mathrm{sh}}(z) = \begin{cases} K_{0,c} & \mathrm{Ri}_g < 0 \quad \leftarrow [\mathrm{LMD\_CONVEC}] \\ K_0 \left[1 - (\frac{\mathrm{Ri}_g}{\mathrm{Ri}_0})^3 \right] & 0 < \mathrm{Ri}_g < \mathrm{Ri}_0 \\ 0 & \mathrm{Ri}_0 < \mathrm{Ri}_g \end{cases}$$

$$K_0 = 5 \times 10^{-3}\ \mathrm{m}^2\ \mathrm{s}^{-1}, \mathrm{Ri}_0 = 0.7$$

- cpp key LMD_NUW_GARGETT (Gargett & Holloway)

$$K_m^{\rm iw}(z) = \frac{10^{-6}}{\sqrt{\max(N^2(z), 10^{-7})}}, \qquad K_s^{\rm iw}(z) = \frac{10^{-7}}{\sqrt{\max(N^2(z), 10^{-7})}}$$

- cpp key LMD_DDMIX (cf Large *et al.* [1994], eqns (31))

Bottom boundary layer

- cpp key LMD_BOTEK : Bottom Ekman layer

$$\begin{aligned}
h_{\rm Ek} &= \min\left\{\frac{0.3u_{\star,b}}{|f|}, h\right\} \\
\sigma_{k+\frac{1}{2}} &= (z_{k+\frac{1}{2}} - h)/h_{\rm Ek} \\
K^{\rm Ek}_{k+\frac{1}{2}} &= \max\left\{4\,\kappa\,u_{\star,b}\,h_{\rm Ek}\,\sigma(1-\sigma), K_{\rm min}\right\} \\
{\rm AKv}_{k+\frac{1}{2}} &= {\rm AKv}_{k+\frac{1}{2}} + K^{\rm Ek}_{k+\frac{1}{2}} \\
{\rm AKt}_{k+\frac{1}{2}} &= {\rm AKt}_{k+\frac{1}{2}} + K^{\rm Ek}_{k+\frac{1}{2}}
\end{aligned}$$

- cpp key LMD_BKPP (Bottom KPP 1994)

Same rationale than surface KPP but this time we search for the critical value ${\rm Ri}_{\rm cr}$ (≈ 0.3) starting from the bottom

$$h_{\rm bbl} = \min\left(h_{\rm bbl}, \frac{0.7u_{\star,b}}{|f|}\right) K_{m,s}(z) = \kappa\,u_{\star,b}\,h_{bbl}\,G(\sigma), \qquad \sigma = \frac{(z-h)}{h_{bbl}}$$

1.5.1.4 Generic length scale

GLS-related options :

GLS_MIXING	Activate Generic Length Scale scheme, default is k-epsilon (see below)
GLS_KOMEGA	Activate K-OMEGA (OMEGA=frequency of TKE dissipation) originating from Kolmogorov [1942]
GLS_KEPSILON	Activate K-EPSILON (EPSILON=TKE dissipation) as in Jones and Launder [1972]
GLS_GEN	Activate generic model of Umlauf and Burchard [2003]
CANUTO_A	Option for CANUTO A stability function (default, see below)
GibLau_78	Option for Gibson and Launder [1978] stability function
MelYam_82	Option for Mellor and Yamada [1982] stability function
KanCla_94	Option for Kantha and Clayson [1994] stability function
Luyten_96	Option for Luyten [1996] stability function
CANUTO_B	Option for CANUTO B stability function
Cheng_02	Option for Cheng *et al.* [2002] stability function

Preselected options for GLS:

```
#ifdef GLS_MIXING
# if   defined GLS_KOMEGA
# elif defined GLS_KEPSILON
# elif defined GLS_GEN
# else
#  define GLS_KEPSILON
# endif
# if   defined CANUTO_A
# elif defined GibLau_78
# elif defined MelYam_82
# elif defined KanCla_94
```

(continues on next page)

(continued from previous page)

```
# elif defined Luyten_96
# elif defined CANUTO_B
# elif defined Cheng_02
# else
#  define CANUTO_A
# endif
#endif
```

The objective of this section is to describe the current implementation of a Generic Length Scale (GLS) turbulence scheme in CROCO that computes the turbulent viscosity K_m and diffusivity K_s. First of all, as usually done in most implementations, the assumption of an horizontally homogeneous flow is made. Following Umlauf and Burchard [2003], the equations satisfied by the two prognostic variables k (the kinetic energy) and ψ (the generic length scale) are

$$\begin{aligned} \partial_t k &= \partial_z(K_k \partial_z k) + P + B - \varepsilon, & K_k = K_m/\mathrm{Sc}_k \\ \partial_t \psi &= \partial_z(K_\psi \partial_z \psi) + \psi k^{-1}\left(\beta_1 P + \beta_3^{\pm} B - \beta_2 \varepsilon\right), & K_\psi = K_m/\mathrm{Sc}_\psi \end{aligned}$$

where the β_j (j=1,3) are constants to be defined, P represents the TKE production by vertical shear $P = K_m\left[(\partial_z u)^2 + (\partial_z v)^2\right]$ and B the TKE destruction by stratification $B = -K_s N^2$ (with N^2 the local Brunt-Vaisala frequency). The dissipation rate ε is related to the generic length scale ψ following

$$\varepsilon = (c_\mu^0)^{3+p/n} k^{3/2+m/n} \psi^{-1/n}, \qquad \psi = (c_\mu^0)^p k^m l^n, \qquad l = (c_\mu^0)^3 k^{3/2} \varepsilon^{-1}$$

with l a mixing length and c_μ^0 a constant (whose value is between 0.526 and 0.555) to be defined. Depending on the parameter values for the triplet (m, n, p) the GLS scheme will either correspond to a $k-\varepsilon$, a $k-\omega$ or the so-called generic [Umlauf and Burchard, 2003] turbulence scheme (to simplify the code and because this scheme do not generally outperform other schemes, the possibility to use the so-called k-kl scheme is not implemented in Croco). Since the equations for e and ψ bear lots of similarities, to avoid excessive code duplication, a unique equation is solved for a quantity $\mathcal{T}_i$ encompassing k (when $i = i_{\mathrm{tke}}$) and ψ (when $i = i_{\mathrm{gls}}$, $i_{\mathrm{gls}} = i_{\mathrm{tke}} + 1$) such that

$$\partial_t \mathcal{T}_i = \partial_z(K_{\mathcal{T}_i} \partial_z \mathcal{T}_i) + (c_i^1 P + c_i^{3,\pm} B - c_i^2 \varepsilon), \qquad K_{\mathcal{T}_i} = K_m/\mathrm{Sc}_{\mathcal{T}_i}$$

where

$$\mathrm{Sc}_{\mathcal{T}_{i_{\mathrm{tke}}}} = \mathrm{Sc}_k, \qquad \mathrm{Sc}_{\mathcal{T}_{i_{\mathrm{gls}}}} = \mathrm{Sc}_\psi$$

and

$$\begin{aligned} c_i^1 &= (i_{\mathrm{gls}} - i) + (i - i_{\mathrm{tke}})\beta_1 e^{-1}\psi \\ c_i^2 &= (i_{\mathrm{gls}} - i) + (i - i_{\mathrm{tke}})\beta_2 e^{-1}\psi \\ c_i^{3,\pm} &= (i_{\mathrm{gls}} - i) + (i - i_{\mathrm{tke}})\beta_3^{\pm} e^{-1}\psi \end{aligned}$$

In practice this explains why in the code the two prognostic quantities k and ψ are stored in a single array $\mathrm{trb(i,j,k,ntime,ngls)}$ avec $\mathrm{ngls} = 2$, $i_{\mathrm{tke}} = 1$ and $i_{\mathrm{gls}} = 2$. Once the quantities k and ψ (hence ε) are known, the turbulent viscosity/diffusivity are given by

$$K_m = c_\mu \left(\frac{k^2}{\varepsilon}\right) = \frac{c_\mu}{(c_\mu^0)^3}(l\sqrt{k}), \qquad K_s = c_\mu^{'} \left(\frac{k^2}{\varepsilon}\right) = \frac{c_\mu^{'}}{(c_\mu^0)^3}(l\sqrt{k}).$$

where c_μ and $c_\mu^{'}$ are determined through so-called stability functions (see below).

Choice of parameter values and stability functions

A particular GLS occurence is defined by the following parameters :

- The exponents (m, n, p) in the definition of ε
- The Schmidt numbers Sc_k and Sc_ψ

- The coefficients β_j (j=1,3)
- The constant c_μ^0
- The stability functions which are generally function of

$$\alpha_M = \left(\frac{k}{\varepsilon}\right)^2 \left[(\partial_z u)^2 + (\partial_z v)^2\right], \qquad \alpha_N = \left(\frac{k}{\varepsilon}\right)^2 N^2$$

Where (m, n, p), Sc_k, Sc_ψ, β_j (j=1,3) are tied to a particular choice of GLS scheme (see table below) while c_μ^0, c_μ and $c_\mu^{'}$ are tied to a particular choice of stability function. The formulation of numerous stability functions can be reconciled when written using the generic form

$$\begin{aligned} c_\mu &= \frac{n_0 + n_1\alpha_N + n_2\alpha_M}{d_0 + d_1\alpha_N + d_2\alpha_M + d_3\alpha_N\alpha_M + d_4\alpha_N^2 + d_5\alpha_M^2} \\ c_\mu^{'} &= \frac{n_0^{'} + n_1^{'}\alpha_N + n_2^{'}\alpha_M}{d_0 + d_1\alpha_N + d_2\alpha_M + d_3\alpha_N\alpha_M + d_4\alpha_N^2 + d_5\alpha_M^2} \end{aligned}$$

where a given choice of stability function will define the parameter values for n_i, d_j, and $n_k^{'}$. In Croco, 7 options are available, these are referred to CANUTO-A, CANUTO-B, Gibson and Launder [1978], Mellor and Yamada [1982], Kantha and Clayson [1994], Luyten [1996], Cheng *et al.* [2002].

Table 1: Table: parameter values corresponding to each particular GLS model

GLS model	m	n	p	β_1	β_2	β_3^-	β_3^+	Sc_e	Sc_ψ
$k-\omega$	0.5	-1	-1	0.555	0.833	-0.6	1	0.5	0.5
$k-\varepsilon$	1.5	-1	3	1.44	1.92	-0.4	1	1	0.7692
Gen	1	-0.67	0	1	1.22	0.05	1	1.25	0.9345

The quantities α_N and α_M in the formulation of c_μ and $c_\mu^{'}$ must satisfy some constraints to guarantee the regularity of numerical solutions. In CROCO, the following steps are done:

1. Apply the Galperin *et al.* [1988] limitation i.e. $l \leq l_{\mathrm{lim}} = \beta_{\mathrm{galp}}\sqrt{2k/N^2}$ on ψ with $\beta_{\mathrm{galp}} = 0.53$. The first step is to use this mixing length l_{lim} to compute $\psi_{\mathrm{min}} = (c_\mu^0)^p k^m (l_{\mathrm{lim}})^n$ and to correct ψ to satisfy the constraint

$$\psi = \max\left(\psi, \psi_{\mathrm{min}}\right)$$

   here the max function is used since the exponent n is negative whatever the GLS scheme.

2. Compute the dissipation rate $\varepsilon = (c_\mu^0)^{3+p/n} k^{3/2+m/n} \psi^{-1/n}$ and correct it

$$\varepsilon = \max\left(\varepsilon, \varepsilon_{\mathrm{min}}\right), \qquad \varepsilon_{\mathrm{min}} = 10^{-12}\ \mathrm{m}^2\ \mathrm{s}^{-3}$$

3. Compute α_N and α_M, and apply "stability and realisability" constraints following Umlauf and Burchard [2003] (their Sec. 4). A first constraint applies on α_N to ensure that $-\partial_{\alpha_N}(c_\mu^{'}/\alpha_N) > 0$ to prevent the occurence of oscillations in $c_\mu^{'}$. This translates into the following limiter

$$\alpha_N^{\mathrm{min}} = \frac{-(d_1 + n_0^{'}) + \sqrt{(d_1 + n_0^{'})^2 - 4d_0(d_4 + n_1^{'})}}{2(d_4 + n_1^{'})}, \qquad \alpha_N = \min\left(\max(0.73\alpha_N^{\mathrm{min}}), 10^{10}\right)$$

   where the coefficient 0.73 is used to ensure the so-called realisability and has been empirically computed thanks to Table 3 in Umlauf and Burchard [2003] in order to satisfy their constraint (48). Then an upper limit is applied on α_M to ensure that $\partial_{\alpha_M}(c_\mu\sqrt{\alpha_M}) \geq 0$ which is also a prerequisite for stability reasons

$$\alpha_M^{\mathrm{max}} = \frac{d_0 n_0 + (d_0 n_1 + d_1 n_0)\alpha_N + (d_1 n_1 + d_4 n_0)\alpha_N^2 + d_4 n_1 \alpha_N^3}{d_2 n_0 + (d_2 n_1 + d_3 n_0)\alpha_N + (d_3 n_1)\alpha_N^2}, \qquad \alpha_M = \min\left(\alpha_M, \alpha_M^{\mathrm{max}}\right)$$

   Once those quantities are computed, stability functions are evaluated as well as the turbulent viscosity/diffusivity.

Surface and bottom boundary conditions

In current version of Croco, both k and ψ are formulated with Neumann boundary conditions at the top and at the bottom. However the nature of those boundary conditions also requires the determination of bottom and surface values for k and ψ.

- For turbulent kinetic energy, the "diagnostic" surface and bottom values are given by

$$k_{\rm sfc} = (u_\star^s/c_\mu^0)^2, \qquad k_{\rm bot} = (u_\star^b/c_\mu^0)^2$$

  and simple homogeneous Neumann boundary conditions are applied

$$K_k \partial_z k|_{\rm sfc} = 0, \qquad K_k \partial_z k|_{\rm bot} = 0$$

  In practice, due to the placement of k and ψ on the computational grid, the Neumann boundary condition is not applied strictly at the surface (resp. at the bottom) but at $z = z_N$ (resp. $z = z_1$) whereas the surface (resp. bottom) is located at $z = z_{N+1/2}$ (resp. $z = z_{1/2}$) with N the number of vertical levels (i.e. the number of cells in the vertical).

- For the generic length scale, a roughness is defined as

$$z_{0,s} = \max\left\{10^{-2}\ \mathrm{m}, \frac{C_{\rm ch}}{g}(u_\star^s)^2\right\}, \qquad C_{\rm ch} = 1400$$

  at the surface and

$$z_{0,b} = \max\left\{10^{-4}\ \mathrm{m}, \mathrm{Zob}\right\}$$

  at the bottom with Zob a user defined roughness length (usually $\mathrm{Zob} = 10^{-2}$ m).

Again, the boundary conditions are applied at the center of the shallowest and deepest grid cells and not at their interfaces which means that the relevant length scales are

$$L_{\rm sfc} = \kappa\left(\frac{\Delta z_N}{2} + z_{0,s}\right), \qquad L_{\rm bot} = \kappa\left(\frac{\Delta z_1}{2} + z_{0,b}\right)$$

with κ the von Karman constant. Moreover TKE values are interpolated at $z = z_N$ and $z = z_1$

$$\widetilde{k}_{\rm sfc} = \frac{1}{2}\left(k_{\rm sfc} + k_{\rm N-1/2}\right), \qquad \widetilde{k}_{\rm bot} = \frac{1}{2}\left(k_{\rm bot} + k_{3/2}\right)$$

where $k_{\rm sfc}$ and $k_{\rm bot}$ are the diagnostic values given above.

The "diagnostic" surface and bottom values for ψ are thus given by

$$\psi_{\rm sfc} = (c_\mu^0)^p (L_{\rm sfc})^n (\widetilde{k}_{\rm sfc})^m, \qquad \psi_{\rm bot} = (c_\mu^0)^p (L_{\rm bot})^n (\widetilde{k}_{\rm bot})^m$$

Then the surface and bottom flux are defined as

$$\mathcal{F}_\psi^{\rm sfc} = K_\psi \partial_z \psi|_{\rm sfc} = -n(c_\mu^0)^{p+1}\frac{\kappa}{\mathrm{Sc}_\psi}(\widetilde{k}_{\rm sfc})^{m+1/2}(L_{\rm sfc})^n$$

$$\mathcal{F}_\psi^{\rm bot} = K_\psi \partial_z \psi|_{\rm bot} = -n(c_\mu^0)^{p+1}\frac{\kappa}{\mathrm{Sc}_\psi}(\widetilde{k}_{\rm bot})^{m+1/2}(L_{\rm bot})^n$$

which correspond to the Neumann boundary conditions applied in the code.

1.5.2 Horizontal diffusion

1.5.2.1 Lateral Momentum Mixing

Related CPP options:

UV_MIX_GEO	Activate mixing on geopotential (constant depth) surfaces
UV_MIX_S	Activate mixing on iso-sigma (constant sigma) surfaces
UV_VIS2	Activate Laplacian horizontal mixing of momentum
UV_VIS4	Activate Bilaplacian horizontal mixing of momentum
UV_VIS_SMAGO	Activate Smagorinsky parametrization of turbulent viscosity (only with UV_VIS2)
UV_VIS_SMAGO3D	Activate 3D Smagorinsky parametrization of turbulent viscosity

Preslected options:

```
# ifdef UV_VIS2
#  define UV_MIX_S
#  define UV_VIS_SMAGO
# endif

#ifdef UV_VIS_SMAGO
# define VIS_COEF_3D
#endif

# ifdef UV_MIX_S
# elif defined UV_MIX_GEO
# else
# define UV_MIX_S /* Default*/
# endif# undef  UV_HADV_TVD
```

Explicit lateral momentum mixing may be only useful when implicit dissipation in UV_HADV_UP3 is not large enough to account for subgrid-scale turbulence resulting from large shear currents (for example in the case of western boundary currents). In this case, Smagorinsky parametrization is recommended (define UV_VIS2 below).

1.5.2.2 Lateral Tracer Mixing

Related CPP options:

TS_MIX_ISO	Activate mixing along isopycnal (isoneutral) surfaces
TS_MIX_GEO	Activate mixing along geopotential surfaces
TS_MIX_S	Activate mixing along iso-sigma surfaces
TS_DIF2	Activate Laplacian horizontal mixing of tracer
TS_DIF4	Activate Bilaplacian horizontal mixing of tracer
TS_MIX_IMP	Activate stabilizing correction of rotated diffusion (used with TS_MIX_ISO and TS_MIX_GEO)

Preslected options:

```
#ifdef TS_HADV_RSUP3 /* Rotated-Split 3rd-order scheme is: */
# define TS_HADV_C4  /* 4th-order centered advection */
# define TS_DIF4 /* + Hyperdiffusion  */
```

(continues on next page)

(continued from previous page)

```
# define TS_MIX_GEO /* rotated along geopotential surfaces */
# define TS_MIX_IMP /* with Semi-Implicit Time-Stepping */
# define DIF_COEF_3D
#endif
```

These options are preselected in set_global_definitions.h for compliance with Advection options.

1.5.3 Bottom friction

Related CPP options:

LIMIT_BSTRESS	Bottom stress limitation forstability
BSTRESS_FAST	Bottom stress computed in step3d_fast
BBL	Bottom boundary layer parametrization

Specification in croco.in:

```
bottom_drag:     RDRG [m/s],  RDRG2,  Zob [m],  Cdb_min, Cdb_max
                    3.0d-04      0.d-3    0.d-3     1.d-4    1.d-1
```

- General form for 3D equations (cf get_vbc.F) :
  - If $z_{0,b} \neq 0$ $\rightarrow$ quadratic friction with log-layer ($C_{d,\min} \leq C_d \leq C_{d,\max}$)

$$\boldsymbol{\tau}_b = C_d \|\mathbf{u}_{k=1}\| \mathbf{u}_{k=1}, \qquad C_d = \left(\frac{\kappa}{\ln\left([z_1 - H]/z_{0,b}\right)} \right)^2$$

  - If $r_{\mathrm{drg2}} > 0$ $\rightarrow$ quadratic friction with C_d = constant}

$$\boldsymbol{\tau}_b = r_{\mathrm{drg2}} \|\mathbf{u}_{k=1}\| \mathbf{u}_{k=1},$$

  - Otherwise $\rightarrow$ linear friction

$$\boldsymbol{\tau}_b = r_{\mathrm{drg}} \mathbf{u}_{k=1},$$

- In the barotropic mode (cf step2D.F) :

$$\boldsymbol{\tau}_b^{2\mathrm{d}} = (r_{\mathrm{drg}} + r_{\mathrm{drg2}} \|\bar{\mathbf{u}}\|) \bar{\mathbf{u}}$$

to be continued here for BSTRESS_FAST and BBL ...

BBL parametrization is detailed in the *Sediment and Biology models* section of the Doc.

1.6 Parallelisation

CROCO has been designed to be optimized on both shared and distributed memory parallel computer architectures. Parallelization is done by two dimensional sub-domains partitioning. Multiple sub-domains can be assigned to each processor in order to optimize the use of processor cache memory. This allow super-linear scaling when performance growth even faster than the number of CPUs.

Related CPP options:

OPENMP	Activate OpenMP parallelization protocol
MPI	Activate MPI parallelization protocol
OPENACC	Activate GPU computation protocol
MPI_NOLAND	No computation on land only CPUs (needs preprocessing)
AUTO_TILING	Compute the best decomposition for OpenMP
PARALLEL_FILES	Output one file per CPU
NC4_PAR	Use NetCDF4 capabilities
XIOS	Dedicated CPU for output (needs XIOS installed)

Preselected options:

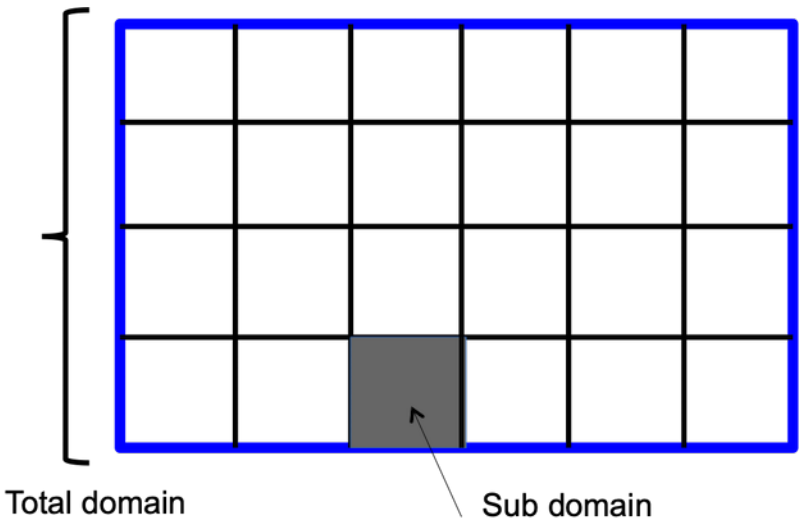
```
# undef MPI
# undef OPENMP
# undef MPI_NOLAND
# undef AUTOTILING
# undef PARALLEL_FILES
# undef NC4_PAR
# undef XIOS
```

1.6.1 Parallel strategy overview

Two kinds of parallelism are currently supported by CROCO: MPI (distributed memory) or OpenMP (shared memory).

CROCO doesn't currently support MPI+OpenMP hybrid parallelisation: use of cpp keys MPI or OPENMP is exclusive.

1.6.1.1 OpenMP (`#define OPENMP`)



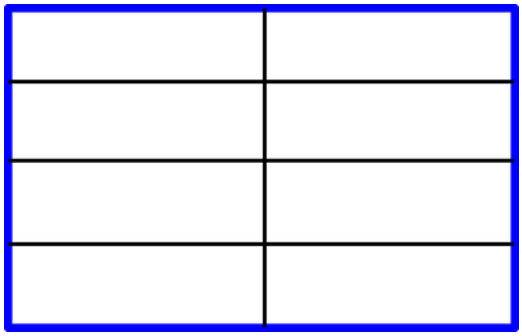
Variables in param.h:

- `NPP`: a number of threads
- `NSUB_X`: number of tiles in XI direction
- `NSUB_E`: number of threads in ETA direction

`NSUB_X` x `NSUB_E` has to be a multiple of `NPP`. Most of the time, we set `NPP=NSUB_X*NSUB_E`

Example 1:

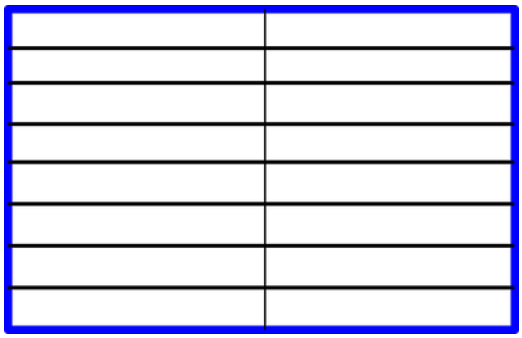
One node with 8 cores: NPP=8, NSUB_X=2, NSUB_E=4



Each thread computes **one** sub-domain.

Example 2:

Still one node with 8 cores: NPP=8, NSUB_X=2, NSUB_E=8



Each thread computes **two** sub-domains.

Code structure

- OpenMP is **NOT** implemented at loop level
- but uses a domain decomposition (similar to MPI) with parallel region
- use of *First touch initialisation* so working arrays are attached to the same thread
- working arrays have the size of the sub-domain only

```
C$OMP PARALLEL
    Call step3D_t_thread()
C$OMP END PARALLEL
```

Fig. 6: Example of a parallel region

```
    Do tile=my_first,my_last   ! Loop on the tiles computed
                               ! by the current thread
      Call compute_1(tile)   ! No Synchronisation needed
      Call compute_2(tile)   ! by these procedures
    Enddo
C$OMP BARRIER ! synchronisation

    Do tile=my_first,my_last   ! Loop on the tiles computed
                               ! by the current thread
      Call compute_3(tile)   !  No Synchronisation needed
      Call compute_4(tile)   !
    Enddo
C$OMP BARRIER ! synchronisation
```

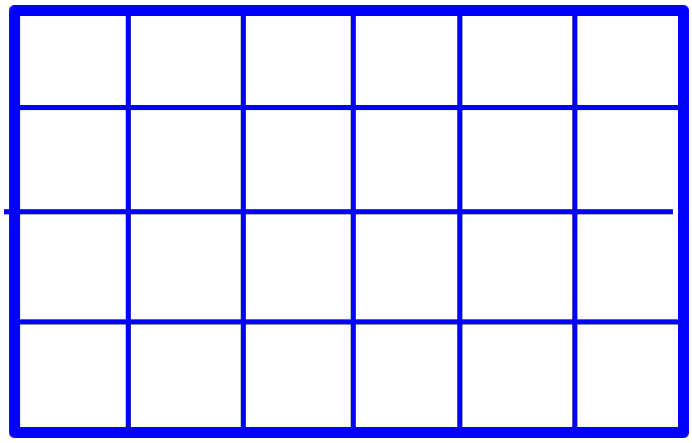
Fig. 7: Inside a parallel region

Here Compute_1 and Compute2 can't write on the same index of a global array.

1.6.1.2 MPI (#define MPI)

Variables in param.h:

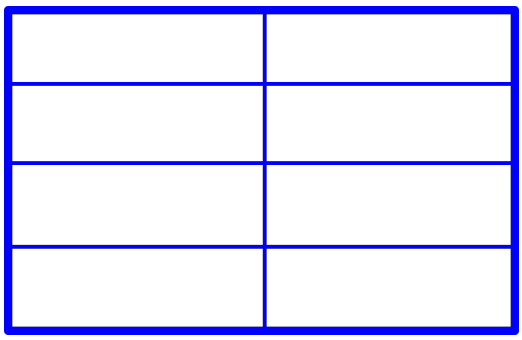
- `NP_XI`: decompostion in XI direction
- `NP_ETA`: decomposition in ETA direction
- `NNODES`: number of cores (=``NP_XI*NP_ETA``, except with `MPI_NOLAND`)
- `NPP = 1`
- `NSUB_X` and `NSUB_ETA`, number of sub-tiles (almost always =1)



Example 1:

8 cores:

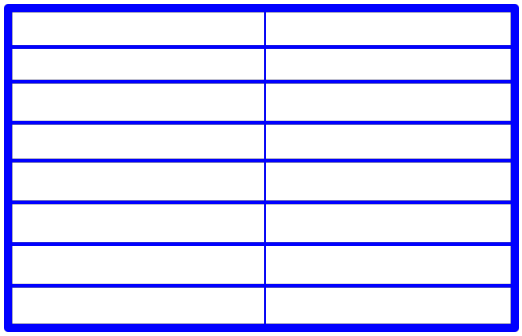
- NP_XI=2, NP_ETA=4, NNODES=8
- NPP=1, NSUB_X=1, NSUB_E=1



Example 2:

8 cores:

- `NP_XI=2`, `NP_ETA=4`, `NNODES=8`
- `NPP=1`, `NSUB_X=1`, `NSUB_ETA=2`



1.6.2 Variable placement for staggered grids

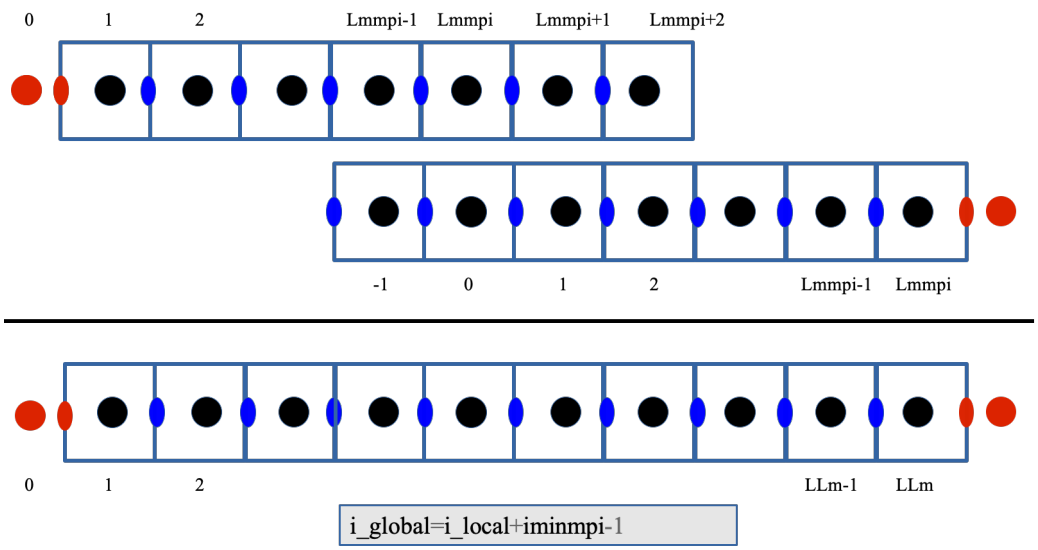
We are working on a staggered grid.

For the barotropic solver, there are 3 different grid points where the variables are stored:

- `RHO`: This variable is "centered" at each grid cell.
- `U`: This variable is at the left/right edge of the grid cell.
- `V`: This variable is at the top/bottom edge of the grid cell

1.6.3 Loops and indexes for staggered grids

1.6.3.1 Parallel/sequential correspondence:

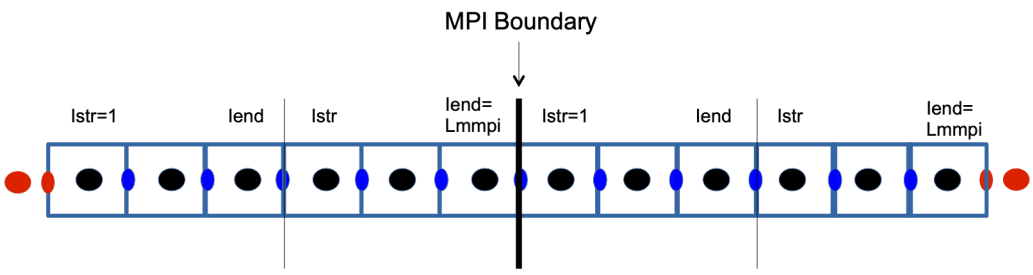


The top image depicts a decomposition of the domain into 2 sub-domains.

The bottom image shows the indices for the total (sequential) domain.

1.6.3.2 Decomposition:

Example : 2 MPI domains, with 2 sub-domains for each MPI rank (with or without OpenMP)



`Istr`, `Iend` are the limits of the sub-domains (without overlap). They are calculated at the beginning of the subroutine by including

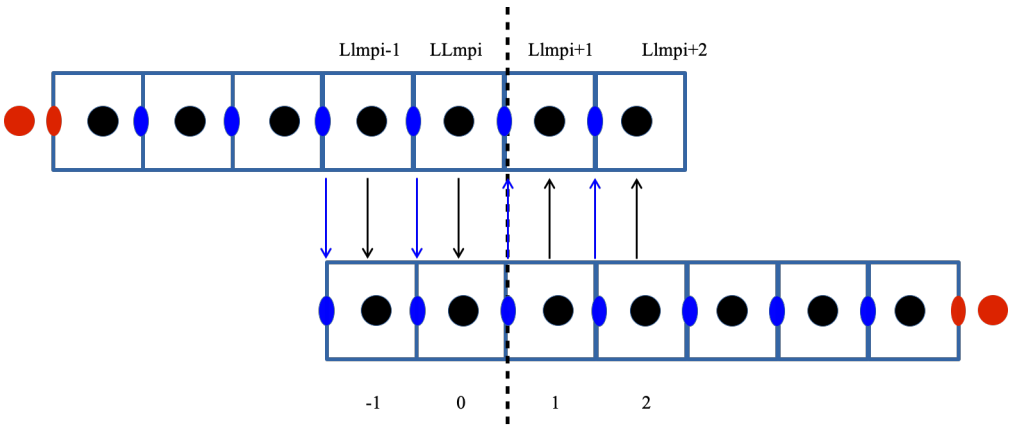
```
      Subroutine compute_1(tile)
      Integer tile, trd
      #include « private_scratch.h » ! Private work arrays
                                   ! A2d, A3d
      #include « compute_tile_bounds.h » Compute Istr, Iend

      trd=0
C$      trd=omp_get_thread_num()
      Call compute_1_tile(Istr,Iend,Jstr,Jend,A2d(1,1,trd),A3d(1,1,trd))
      Return
      end
```

Computation of `Istr`, `Iend` and use of working arrays.

1.6.4 Halo layer exchanges

CROCO makes use 2 or 3 ghost cells depending on the chosen numerical schemes.



In the example above (2 ghosts cells), for correct exchanges, after computation:

- η has to be valid on `(1:Iend)`
- u has to be valid on `(1:Iend)` except on the left domain `(2:Iend)`

`IstrU` is the lower limit of validity of the `U` points.

Computation of auxiliary indexes is done by including a file:

```
subroutine step2D_FB_tile (Istr, Iend, Jstr, Jend, zeta_new, ...
...
#include "compute_auxiliary_bounds.h"   ! Compute IstrU, IstrR
```

1.6.5 Dealing with outputs

By default, with MPI activated input and output files are treated in a pseudo-sequential way, and one NetCDFfile corresponds to the whole domain. This has drawbacks when using a large number of computational cores, since each core is writing its part of the domain sequentially, the time dedicated to outputs increase with the number of cores. Three alternatives are implemented within CROCO.

Splited files (#define PARALLEL_FILES)

In this case, each core is writing its part only of the domain in separated files (one per MPI domain). This writing is performed concurrently. One other advantage is to avoid the creation of huge output files. The domain related output files can be recombined using ncjoin utility (in fortran) compiled in the same time than CROCO. Note that in this case, input files have to be splited as well, using partit utility.

Parallel NetCDF(#define NC4PAR)

This option requires NetcDF4 verion, installed with parallel capabilities. All cores are writing concurrently but in the same time.

IO server (#define XIOS)

XIOS is an external IO server interfaced with CROCO. Informations about use and installation can be found there https://forge.ipsl.jussieu.fr/ioserver. In this case, output variables are defined in .xml files. See also the Diagnostics chapter.

1.6.6 Run with GPU

- OpenACC directive-based approach is the most appropriate method to develop the GPU version of croco. Low development cost, a single program repository can be shared by CPU and GPU. Similar to OpenMP style which is familiar to many users. An openacc compiler compatible is necessary : only nvfortran, and old pgi have been tested.
- To run croco on GPUs, the key OPENACC as to be set in the cppdefs.h file. With MPI, all the mpi process are dispatch over the GPUs. Effiency is better with one or two mpi by GPU card.

```
#define OPENACC
```

1.6.6.1 General implementation

Implementation is done by inserting basic OpenACC directives :

- Kernels or parallel directives : Basically, inserted into the outermost of nested loops
- Loop independent/seq directives : Inserted according to the loop algorithms

No quantitative configurations for GPU threads are specified. Gang or vector clauses are not applied, leaving it to the compiler's decision.

Exemple from `pre_step3.F`

```
!$acc kernels if(compute_on_device) default(present)
!$acc end kernels
!$acc parallel loop if(compute_on_device) default(present)
```

1.6.6.2 Data directives

Remove redundant data transfer between CPU and GPU. There is one main copy data to device, on copy-back before output. The file copy_to_devices.h do this job.

Python script `common2device.py` generate the file `copy_to_devices.h` and have to be rerun when `*.h` files are modified.

1.6.6.3 3D loop tunning with preprocessing by compilation

For GPU efficiency some 3D loops have to be reordered or variable expand. This is done by en additional preprocesing (in python). And it's added in jobcomp compilation procedure. Two kind of transformations, first one transforms 2D variables internal to loops in 3D variables. Second restructures loop with z dependencies.

Example 1:

```
#   if defined OPENACC
        DOEXTEND(k,1,N,FX,FE,WORK)
#   endif
... FX(i,j) will become FX_3D(i,j,k)  (FE_3D, WORK_3D )
#   if defined OPENACC
        ENDDOEXTEND
#   endif
```

Example 2:

```
DOLOOP2D(Istr,Iend,Jstr,Jend)
    do k=1,N
      do i=Istr,Iend
         DC(i,k)=1./Hz_half(i,j,k)
      enddo
    enddo
    ...
ENDDOLOOP2D
```

Become in CPU version:

```
!$acc kernels if(compute_on_device) default(present)
!$acc loop private( DC, FC, CF)
    do j=Jstr,Jend
    do k=1,N
      do i=Istr,Iend
         DC(i,k)=1.D0/Hz_half(i,j,k)
      enddo
    enddo
    do i=Istr,Iend
       DC(i,0)=cdt*pn(i,j)*pm(i,j)
    enddo
    ...
    enddo
```

And become in GPU version :

```
!$acc kernels if(compute_on_device) default(present)
  DO j=Jstr,Jend
!$acc loop private(DC1D,CF1D,FC1D) vector
    DO i=Istr,Iend
    do k=1,N
         DC1D(k)=1.D0/Hz_half(i,j,k)
    enddo
       DC1D(0)=cdt*pn(i,j)*pm(i,j)
    ENDDO
    ...
  ENDDO
```

1.7 Atmospheric Surface Boundary Layer

There are two ways to force CROCO with atmospheric fields. The first is the use of classical bulk parameterization (BULK_FLUX cpp key). Four have been updated and implemented in CROCO (COARE3.0, ECUMEV0, ECUMEV6 and WASP). The second is the use of a simplified atmospheric model (ABL1D cpp key, Lemarié *et al.* [2021]) on top of the bulk parameterization.

Note

- ABL1D forcing strategy is still under evaluation.
- In both cases, the use of a bulk parameterization (BULK_FLUX) is mandatory.

Below you'll find the CPP keys for these two forcing capabilities and their descriptions :

BULK_FLUX's related CPP options:

CPP options	Description
BULK_FLUX	Activate bulk formulation for surface turbulent fluxes (by default, COARE3.0 parametrization is used)
BULK_ECUMEV0	Use ECUMEv0 bulk formulation instead of COARE3.0 formulation
BULK_ECUMEV6	Use ECUMEv6 bulk formulation instead of COARE3.0 formulation
BULK_WASP	Use WASP bulk formulation instead of COARE3.0 formulation
BULK_GUSTINESS	Add in gustiness effect on the surface wind module. Can be used for both bulk parametrizations.
BULK_LW	Add in long-wave radiation feedback from model SST
SFLUX_CFB	Activate current feedback on … [Renault *et al.*, 2020]
CFB_STRESS	… surface stress (used by default when SFLUX_CFB is defined)
CFB_WIND_TRA	… surface tracers (used by default when SFLUX_CFB is defined)
SST_SKIN	Activate skin sst computation [Zeng and Beljaars, 2005]
ONLINE	Read native files and perform online interpolation on CROCO grid (default cubic interpolation)
QCORRECTION	Activate heat flux correction around model SST (if BULK_FLUX is undefined)
SFLX_CORR	Activate freshwater flux correction around model SSS (if BULK_FLUX is undefined)
ANA_DIURNAL_SW	Activate analytical diurnal modulation of short wave radiations (only appropriate if there is no diurnal cycle in data)
RAIN_FLUX	Add E-P flux into the mass budget (added in omega)

By default COARE3.0 parametrization is used with GUSTINESS effects. To change bulk parametrization, one has to define one of the following cpp keys (not additional) :

- define BULK_ECUMEV0 to use ECUME_v0 parametrization
- define BULK_ECUMEV6 to use ECUME_v6 parametrization

- define BULK_WASP to use WASP parametrization

> **Warning**
>
> It is possible to add GUSTINESS effects for all parametrizations by defining BULK_GUSTINESS cpp key

ABL1D's related CPP options:

CPP options	Description
ABL1D	Activate ABL1d model and BULK_FLUX parametrization
ANA_ABL_LSDATA	Impose large scale forcing data for analytical simulations
ANA_ABL_VGRID	Compute vertical grid for ABL1d online for analytical simulations
STRESS_AT_RHO_POINTS	Compute stress at rho points
ABL_NUDGING	Active nudging for …
ABL_NUDGING_DYN	… dynamic (two components of the horizontal wind speed)
ABL_NUDGING_TRA	… tracers (air temperature and specific humidity)
ABL_DYN_RESTORE_EQ	Active Equator dynamic restoration

ONLINE related CPP options:

ONLINE option is an alternative to pre-processing of surface forcing data for both forcing strategy (ABL1D / BULK_FLUX), that can be useful for long-term and/or high resolution simulations, especially if handling multiple configurations. ONLINE option calls for CUBIC_INTERP in `set_global_definitions.h`.

CPP options	Description
ECMWF	Use ECMWF atm fluxes
AROME	Use METEO FRANCE fluxes
READ_PATM	Read atmospheric pressure instead of using default reference pressure and take into account the atmospherical pressure gradient in the equations
OBC_PATM	In the case of READ_PATM, inverse barometer effect to the open boundaries if the atmospherical pressure is read in the meteo file.

ONLINE variables description

The following table describe what is expected in online forcing file.

variables Long name	units	option given cppkey AROME	ERA_ECMWF
Temperature above ground	Kelvin	t2m	T2M
specific humidity	Kg/Kg	rh	Q
precipitation rate	Kg.m-2.s-1	rain	TP
Upward Net short wave	W.m-2	swhf	SSR
Downward long wave	W.m-2	lwhf	STRD
U component of wind	m/s	u10m	U10M
V component of wind	m/s	v10m	V10M
Pressure at sea level	Pa	pmer	msl
time vector	days	time	time

Warning

For *AROME* time units must be in days since 1900-01-01

Note

Long wave flux is either downward or net given your inputs in atmospherical modeles :

- First case : You have a downward flux only. So you have to add *BULK_LW* option to compute the net flux.
- Second case : You already have a net long flux. You must remove the *BULK_LW* to not take twice the feedback into account.

Preselected options (cppdefs.h):

```
# undef BULK_FLUX
# ifdef BULK_FLUX
#  undef  BULK_ECUMEV0
#  undef  BULK_ECUMEV6
#  undef  BULK_WASP
#  define BULK_GUSTINESS
#  define BULK_LW
#  undef  SST_SKIN
#  undef  ANA_DIURNAL_SW
#  undef  ONLINE
#  ifdef ONLINE
#   undef  AROME
#   undef  ERA_ECMWF
#  endif
#  undef READ_PATM
#  ifdef READ_PATM
#   define OBC_PATM
#  endif
# else
#  define QCORRECTION
#  define SFLX_CORR
#  undef  SFLX_CORR_COEF
#  define ANA_DIURNAL_SW
# endif
# undef  SFLUX_CFB
# undef  SEA_ICE_NOFLUX
```

Preselected options (cppdefs_dev.h):

```
#ifdef BULK_FLUX
# ifdef ONLINE
#  define CUBIC_INTERP
# endif
# ifdef BULK_ECUMEV0
#  define BULK_GUSTINESS
# elif defined BULK_ECUMEV6
#  define BULK_GUSTINESS
# elif defined BULK_WASP
#  define BULK_GUSTINESS
# endif
#endif
```

```
#ifdef SFLUX_CFB
# ifdef BULK_FLUX
#  define CFB_STRESS
#  define CFB_WIND_TRA
# else
#  undef  CFB_STRESS
#  undef  CFB_WIND_TRA
# endif
#endif
```

1.8 Open boundaries conditions

If a lateral boundary faces the open ocean, robust open boundary conditions (OBCs) are needed [Marchesiello *et al.*, 2001]. Forcing of tracer and baroclinic flow is applied via an adaptive radiation condition, which helps perturbations to leave the domain with only a small effect on the interior solution. The same method can be applied to the depth-averaged flow, but (for tidal forcing in particular) we generally prefer the incoming characteristic of the shallow water system as in Flather-type conditions [Marchesiello *et al.*, 2001, Blayo and Debreu, 2005]. This allows long-wave data to be forced in, while those generated inside the domain can leave it, which also guarantees the quasi-conservation of mass and energy across the open boundary. A Sponge layer is added near the open boundaries to limit small-scale effects and ease the transition between the interior solution and the boundary data. The boundary data can be applied only at the boundary (see BRY strategy below) or in a nudging layer (CLIMATOLOGY strategy).

This set of OBCs are given as default if no other choice is made. They have performed well in most applications, from the deep ocean to the coastal areas. For details, refer to Marchesiello *et al.* [2001] and Blayo and Debreu [2005].

1.8.1 OBC

Related CPP options:

OBC_EAST	Open eastern boundary
OBC_WEST	Open western boundary
OBC_SOUTH	Open southern boundary
OBC_NORTH	Open northern boundary

Related CPP options:

OBC_M2SPECIFIED	Activate specified OBCs for barotropic velocities
OBC_M2CHARACT	Activate OBCs from characteristic methods for barotropic velocities (default)
OBC_M2ORLANSKI	Activate radiative OBCs for barotropic velocities
OBC_VOLCONS	Enforce mass conservation at open boundaries (with OBC_M2ORLANSKI)
OBC_M3SPECIFIED	Activate specified OBCs for baroclinic velocities
OBC_M3ORLANSKI	Activate radiative OBCs for baroclinic velocities (default)
OBC_TSPECIFIED	Activate specified OBCs for tracers
OBC_TORLANSKI	Activate radiative OBCs for tracers (default)
OBC_TUPWIND	Activate upwind OBCs for tracers

For non-tidal forcing, the combination of OBC_M2ORLANSKI and OBC_VOLCONS often provides the best performances in terms of transparency of barotropic flow at the open boundaries. However, OBC_M2CHARACT is near as good and also provides the best conditions for tidal forcing. It is therefore set as default in cppdefs_dev.h.

Preselected options (in cppdefs_dev.h but set your own choice in cppdefs.h if needed):

```
# undef  OBC_M2SPECIFIED
# define OBC_M2CHARACT
# undef  OBC_M2ORLANSKI
# ifdef OBC_M2ORLANSKI
#  define OBC_VOLCONS
# endif
# define OBC_M3ORLANSKI
# define OBC_TORLANSKI
# undef  OBC_M3SPECIFIED
# undef  OBC_TSPECIFIED
```

1.8.2 Sponge Layer

SPONGE is preselected in cppdefs.h and calls for SPONGE_GRID in cppdefs_dev.h. SPONGE_GRID selects the sponge layer extension (10 points with cosine shape function) and viscosity and diffusivity values according to the horizontal resolution (limited by the CFL stability conditions).

Related CPP options:

SPONGE	Activate areas of enhanced viscosity and diffusivity near lateral open boundaries.
SPONGE_GRID	Automatic setting of the sponge width and value
SPONGE_DIF2	Sponge on tracers (default)
SPONGE_VIS2	Sponge on momentum (default)
SPONGE_SED	Sponge on sediment (default)

1.8.3 Nudging layers

The nudging layer has the same extension as the sponge layer. In nudging layers, tracer and momentum fields are nudged towards climatological values at a time scale Tau_out (possibly different for momentum and tracers) that is given in namelist croco.in

Related CPP options:

ZNUDGING	Activate nudging layer for sea level
M2NUDGING	Activate nudging layer for barotropic velocities
M3NUDGING	Activate nudging layer for baroclinic velocities
TNUDGING	Activate nudging layer for tracers
ROBUST_DIAG	Activate nudging over the whole domain

1.8.4 Lateral forcing

1.8.4.1 CLIMATOLOGY strategy

Related CPP options:

CLIMATOLOGY	Activate processing of 2D/3D data (climatological or simulation/reanalysis) used as forcing at the open boundary points + nudging layers
ZCLIMATOLOGY	Activate processing of sea level
M2CLIMATOLOGY	Activate processing of barotropic velocities
M3CLIMATOLOGY	Activate processing of baroclinic velocities
TCLIMATOLOGY	Activate processing of tracers

1.8.4.2 BRY strategy

FRC_BRY is useful for inter-annual forcing on high-resolution domains. FRC_BRY is compatible with CLIMATOLOGY that can still be used for nudging layers.

Related CPP options:

FRC_BRY	Activate processing of 1D/2D data used as forcing at open boundary points strictly
Z_FRC_BRY	Activate open boundary forcing for sea level
M2_FRC_BRY	Activate open boundary forcing for barotropic velocities
M3_FRC_BRY	Activate open boundary forcing for baroclinic velocities
T_FRC_BRY	Activate open boundary forcing for tracers

Preselected options (cppdefs.h):

```
# define CLIMATOLOGY
# ifdef CLIMATOLOGY
#  define ZCLIMATOLOGY
#  define M2CLIMATOLOGY
#  define M3CLIMATOLOGY
#  define TCLIMATOLOGY
#  define ZNUDGING
#  define M2NUDGING
#  define M3NUDGING
#  define TNUDGING
#  undef ROBUST_DIAG
# endif
# undef FRC_BRY
# ifdef FRC_BRY
#  define Z_FRC_BRY
#  define M2_FRC_BRY
#  define M3_FRC_BRY
#  define T_FRC_BRY
# endif
```

1.9 Rivers

Related CPP options:

PSOURCE	Activate point sources (rivers)
ANA_PSOURCE	use analytical vertical profiles for point sources (set in set_global_definitions.h)
PSOURCE_NCFILE	Read variable river transports in netcdf file
PSOURCE_NCFILE_TS	Read variable river concentration in netcdf file
PSOURCE_MASS	Apply point source as volume vertical influx at rho point instead of a flux through u,v interfaces

ANA_PSOURCE gives the vertical distribution of point source outflow. The default shape is an exponential vertical distribution. The vertical shape can be customized in subroutine *ana_psource* in *analytical.F*

An example of runoff file with **PSOURCE_NCFILE** is given below

```
netcdf croco_runoff {
dimensions:
qbar_time = 28193 ;
n_qbar = 9 ;
runoffname_StrLen = 30 ;
two = 2 ;
temp_src_time = 10690 ;
salt_src_time = 10690 ;
variables:
double qbar_time(qbar_time) ;
  qbar_time:long_name = "runoff time" ;
  qbar_time:units = "days" ;
  qbar_time:cycle_length = 0. ;
  qbar_time:long_units = "days since 1900-01-01" ;
char runoff_name(n_qbar, runoffname_StrLen) ;
  runoff_name:long_name = "runoff name" ;
double runoff_position(n_qbar, two) ;
  runoff_position:long_name = "position of the runoff (by line) in the CROCO grid" ;
double runoff_direction(n_qbar, two) ;
  runoff_direction:long_name = "direction/sense of the runoff (by line) in the CROCO␣
↪grid" ;
double Qbar(n_qbar, qbar_time) ;
  Qbar:long_name = "runoff discharge" ;
  Qbar:units = "m3.s-1" ;
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, temp_src_time) ;
```

(continues on next page)

(continued from previous page)

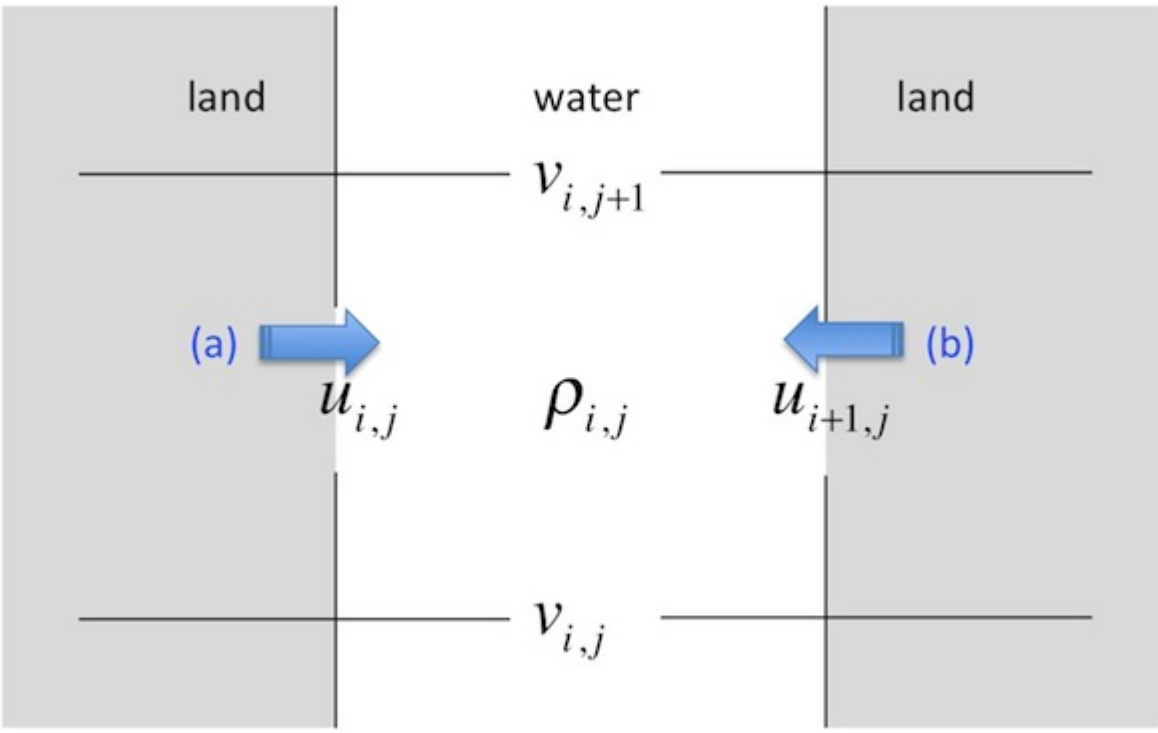
```
salt_src:long_name = "runoff salinity" ;
salt_src:units = "psu" ;
            }
```

When using **PSOURCE**, Isrc and Jsrc refer to the i,j index of the u-face or v-face the flow crosses - NOT the i,j index of the rho cell it flows into. The i,j values must follow ROMS Fortran numbering convention for the appropriate u-point or v-point on the ROMS staggered grid.

This numbering convention is shown in the figure below (Courtesy of ROMS-RUTGERS team) for flow crossing a u-face into a cell from either the left or the right. This makes it more obvious why the index of the u-face must be specified, because to give the i,j indices of the receiving rho-cell would be ambiguous.

The u-face or v-face should be a land/sea mask boundary (i.e. a coastline). If the cell face is placed wholly in the land you get nothing because there is no wet cell for the flow to enter. If the face is in the middle of open water you have a situation where the flow at that cell face computed by the advection algorithm is 'REPLACED, not augmented, by the source.

It is very easy to misconfigure source/sink locations so caution and careful checking is required.



When using **PSOURCE_MASS**, Isrc and Jsrc refer to the i,j index of the rho cell it flows into. In this case there is no need to be along land/sea boundary

> **Note**
>
> **PSOURCE_MASS** is not available with openmp parallelization

1.10 Tides

Related CPP options:

TIDES	Activate tidal forcing at open boundaries
SSH_TIDES	Process and use tidal sea level data
UV_TIDES	Process and use tidal current data
OBC_REDUCED_PHYSICS	Compute tidal velcocity from tidal elevation in case of tidal current is not available
TIDERAMP	Apply ramping on tidal forcing (1 day) at initialization Warning! This should be undefined if restarting the model
TIDES_MAS	Harmonic composition to build tide elevation from SHOM (Simon method) to take into account nodal correction Harmonic composition for currents use the classic method if UV_TIDES is activated.

Preselected options:

```
# ifdef TIDES
#  define SSH_TIDES
#  define UV_TIDES
#  ifndef UV_TIDES
#   define OBC_REDUCED_PHYSICS
#  endif
#  define TIDERAMP
# endif
```

Coastal case options with TIDES_MAS:

```
# ifdef TIDES
#  define SSH_TIDES
#  define UV_TIDES
#  ifndef UV_TIDES
#   define OBC_REDUCED_PHYSICS
#  endif
#  define TIDES_MAS
#  define TIDERAMP
# endif
```

Important

With **TIDES_MAS** the user only need an atlas with tidal harmonics (elevation and current) interpolated on the Croco grid. The harmonics dont need to take nodal correction in their amplitude and phase like in classic method.

Important

TIDES_MAS requires the **USE_CALENDAR** cppkey, because it needs a precise time reference and the origin date from the harmonic's atlas must be 1900-01-01.

1.11 Wavemaker

WAVE_MAKER is useful for wave-resolving simulations in the nearshore zone. In wave-resolving nearshore problem, a spectrum of linear waves can be forced at the offshore boundary via an analytical wavemaker (parameters set in croco.in and used in wave_maker.h as part of the ana_bry subroutine). The spectrum is based by default on a JONSWAP statistical distribution and has frequency and directional spreading.

A single-sum wavemaker (i.e. each component of the wave has a distinct frequency and direction) is implemented in CROCO to alleviate the problem of coherent wave interference arising from double-summation methods (Treillou *et al.* [2024]). A single-sum wavemaker solves this problem and is also much more cost-effective (and does not require high-frequency resolution).

1.11.1 Formulation

The offshore wavemaker is used with OBC_WEST or OBC_EAST, providing BRY array values for free surface elevation η, cross-shore and alongshore velocities (u, v), according to the following equations:

$$\eta_{bc}(y,t) = \sum_{i}^{N\times M} a_i \cos(k_{y,i}y - \omega_i t - \phi_i)$$

$$u_{bc}(x,y,t) = \eta_{bc}(y,t)\omega_p \cos(\theta_m)\frac{\cosh(k_p(z+h))}{\sinh(k_p h)}$$

$$v_{bc}(x,y,t) = \eta_{bc}(y,t)\omega_p \sin(\theta_m)\frac{\cosh(k_p(z+h))}{\sinh(k_p h)}$$

with :

- $a_i = a_m\sqrt{\frac{D(\theta_i)S(\omega_i)d\omega}{\sum_i D(\theta_i)S(\omega_i)d\omega}}$: amplitude given by the statistical distribution $S(\omega_i,\theta_i) = S(\omega_i) \times D(\theta_i)$, the product of a JONSWAP frequency spectrum $S(\omega_i)$ and a Gaussian-type directional spectrum $D(\theta_i)$. i is the index of spectral distribution in frequency and direction; N and M are the number of frequencies and directions; and $d\omega$ is frequency resolution.
- $a_m = \sqrt{H_s/8}$ is the rms wave amplitude (related to significant wave height H_s).
- $\theta_i, \theta_m, \sigma_\theta$: i-th wave direction, mean wave direction, and directional spread around the mean.
- $k_{y,i} = k_i \sin(\theta_i)$: alongshore wavenumber (k_i being the linear theory wavenumber given by $\omega_i^2 = gk_i\tanh(k_i h)$ with h the mean water depth)
- ω_p, k_p : peak frequency and wavenumber,
- ϕ_i uniformly distributed random phase.

1.11.2 Input parameters

Input parameters for the wavemaker are set in **croco.in** under keyword: wave_maker (e.g., *Flash Rips* test case):

```
wave_maker:  amp [m], prd [s], ang [deg], ang spread [deg], freq spread␣
↪(gamma)
             0.24     13.       0.         30.              3.3
```

amp: wave amplitude $a_m = \sqrt{H_s/8}$

prd: peak period $T_p = 2\pi/\omega_p$

ang: mean wave angle θ_m

`ang spread`: directional spread σ_θ

`freq spread`: peak enhancement factor γ of JONSWAP spectrum

> **Note**
>
> The directional spread is essential for generating short-crested waves (and flash rips). The mean crest length should vary as $\lambda_p/(2\sin\sigma_\theta)$, with $\lambda_p = 2\pi/k_p$ the peak wavelength.

> **Note**
>
> if WAVE_MAKER_SPECTRUM is not defined (see below), the wavemaker produces a monochromatic wave.

Related CPP options:

WAVE_MAKER	Activate Wavemaker for wave resolving nonhydrostatic case
WAVE_MAKER_SPECTRUM	Activate Wave spectrum with frequency spread (default: JONSWAP)
WAVE_MAKER_DSPREAD	Add directional spread in wave spectrum
WAVE_MAKER_OBLIQUE	Specify if Non-zero mean angle (compute v component)

Preselected options:

```
#ifdef WAVE_MAKER
# if defined WAVE_MAKER_JONSWAP || defined WAVE_MAKER_GAUSSIAN
#  define WAVE_MAKER_SPECTRUM
# endif
# ifdef WAVE_MAKER_SPECTRUM
#  ifdef WAVE_MAKER_JONSWAP
#  elif defined WAVE_MAKER_GAUSSIAN
#  else
#   define WAVE_MAKER_JONSWAP /* default */
#  endif
# endif
# if defined WAVE_MAKER_DSPREAD && defined NS_PERIODIC
#  define WAVE_MAKER_DSPREAD_PER /* correct direction for periodicity */
# endif
# ifndef WAVE_MAKER_SPECTRUM
#  define STOKES_WAVES
# endif
#endif
```

1.11.3 JONSWAP spectrum

As previously said, the spectrum distribution is the product of a JONSWAP frequency spectrum $S(\omega)$ and a Gaussian-type directional spectrum $D(\theta) : S(\omega,\theta) = S(\omega) \times D(\theta)$.

The JONSWAP spectrum is formulated in CROCO as follows:

$$S(\omega) = H_s^2 \beta_J \omega_p^4 \omega^{-5} \exp[-1.25\omega^{-4}\omega_p^4]\gamma^r$$

function of the significant wave height H_s, peak wave frequency ω_p, peak enhancement factor γ, and:

$$r = \exp\left[-\frac{1}{2}\left(\frac{\omega-\omega_p}{\sigma_\omega \omega_p}\right)^2\right]$$

$$\sigma_\omega = 0.07, \text{ if } \omega \leq \omega_p$$
$$0.09, \text{ if } \omega > \omega_p$$

$$\beta_J = \frac{0.06238(1.094 - 0.01915\log\gamma)}{0.23 + 0.0336\gamma - 0.185(1.9+\gamma)^{-1}}$$

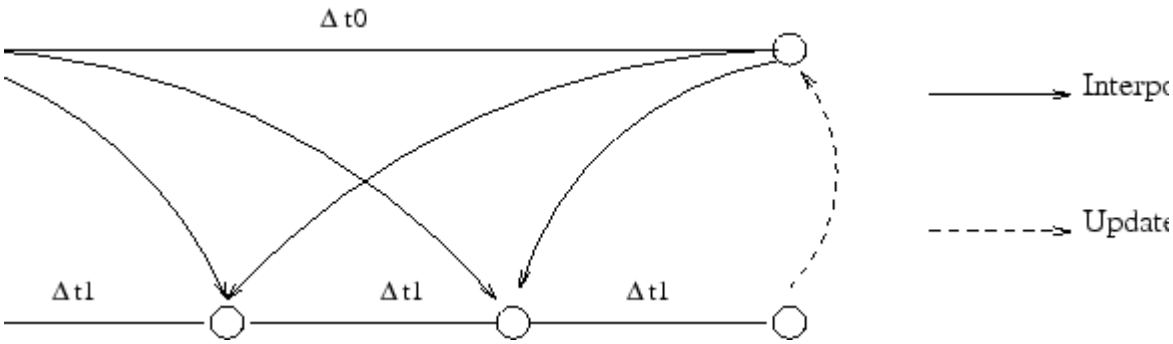
The directional spectrum around the mean direction θ_m, with directional spread σ_θ, is:

$$D(\theta) = \exp\left[-\left(\frac{\theta-\theta_m}{1.5\sigma_\theta}\right)^2\right]$$

with $\int_{\theta_{min}}^{\theta_{max}} D(\theta)d\theta = 1$ to ensure that the directional spread (σ_θ) around the mean wave angle does not affect wave energy.

1.12 Nesting Capabilities

To address the challenge of bridging the gap between near-shore and offshore dynamics, a nesting capability has been added to CROCO and tested for the California Upwelling System [Debreu *et al.*, 2012, Penven *et al.*, 2006]. The method chosen for embedded griding takes advantage of the AGRIF (Adaptive Grid Refinement in Fortran) package [Debreu *et al.*, 2012, Blayo and Debreu, 1999, Debreu *et al.*, 2008]. AGRIF is a Fortran 95 package for the inclusion of adaptive mesh refinement features within a finite difference numerical model. One of the major advantages of AGRIF in static-grid embedding is the ability to manage an arbitrary number of fixed grids and an arbitrary number of embedding levels.



A recursive integration procedure manages the time evolution for the child grids during the time step of the parent grids (Fig. 2). In order to preserve the CFL criterion, for a typical coefficient of refinement (say, a factor of 3 for a 5 km resolution grid embedded in a 15 km grid), for each parent time step the child must be advanced using a time step divided by the coefficient of refinement as many time as necessary to reach the time of the parent (Fig. 2). For simple 2-level embedding, the procedure is as follows:

1. Advance the parent grid by one parent time step.
2. Interpolate the relevant parent variables in space and time to get the boundary conditions for the child grid.
3. Advance the child grid by as much child time steps as necessary to reach the new parent model time.
4. Update point by point the parent model by averaging the more accurate values of the child model (in case of 2-way nesting).

The recursive approach used in AGRIF allows the specification of any number of nesting levels. Additional CPP options are related to AGRIF, they are in set_global_definitions.h and set_obc_definitions.h files. These are default options intended for nesting developers and should not be edit by standard users.

For a better understanding of ROMS nesting capabilties using AGRIF, check the published articles on CROCO/ROMS nesting implementation and also the AGRIF project homepage:

1. CROCO/ROMS 1 way nesting : Penven *et al.* [2006]
2. CROCO/ROMS 2 way nesting: Debreu *et al.* [2012]
3. AGRIF homepage : http://www-ljk.imag.fr/MOISE/AGRIF/

Related CPP options:

AGRIF	Activate nesting capabilities (1-WAY by default)
AGRIF_2WAY	Activate 2-WAY nesting (update parent solution by child solution)

1.13 Other modules : sediment models, flow-obstruction models, biology models

1.13.1 Bottom Boundary Layer model

Related CPP options:

BBL	Activate bottom boundary layer parametrization
ANA_WWAVE	Set analytical (constant) wave forcing (hs,Tp,Dir).
ANA_BSEDIM	Set analytical bed parameters (if SEDIMENT is undefined)
Z0_BL	Compute bedload roughness for ripple predictor and sediment purposes
Z0_RIP	Determine bedform roughness ripple height and ripple length for sandy bed
Z0_BIO	Determine (biogenic) bedform roughness ripple height and ripple length for silty beds

Preselected options:

```
#ifdef BBL
# ifdef OW_COUPLING
# elif defined WAVE_OFFLINE
# elif defined WKB_WWAVE
# else
#  define ANA_WWAVE
# endif
# ifdef SEDIMENT
#  undef  ANA_BSEDIM
# else
#  define ANA_BSEDIM
# endif
# ifdef SEDIMENT
#  define Z0_BL
# else
#  undef  Z0_BL
# endif
# ifdef Z0_BL
#  define Z0_RIP
# endif
# undef  Z0_BIO
#endif
```

DESCRIPTION

Reynolds stresses, production and dissipation of turbulent kinetic energy, and gradients in velocity and suspended-sediment concentrations vary over short vertical distances, especially near the bed, and can be difficult to resolve with the vertical grid spacing used in regional-scale applications. CROCO provides algorithms to parameterize some of these subgrid-scale processes in the water column and in the bottom boundary layer (BBL). Treatment of the BBL is important for the circulation model solution because it determines the stress exerted on the flow by the

bottom, which enters the Reynolds-averaged Navier-Stokes equations as a boundary conditions for momentum in the x and y directions:

$$K_m \frac{\partial u}{\partial s} = \tau_{bx}$$
$$K_m \frac{\partial v}{\partial s} = \tau_{by}$$

Determination of the BBL is even more important for the sediment-transport formulations because bottom stress determines the transport rate for bedload and the resuspension rate for suspended sediment.

CROCO implements either of two methods for representing BBL processes: (1) simple drag-coefficient expressions or (2) more complex formulations that represent the interactions of wave and currents over a moveable bed. The drag-coefficient methods implement formulae for linear bottom friction, quadratic bottom friction, or a logarithmic profile. The other, more complex wave-current BBL model is described by Blaas *et al.* [2007] with an example of its use on the Southern California continental shelf. The method uses efficient wave-current BBL computations developed by Soulsby [1995] in combination with sediment and bedform roughness estimates of Grant and Madsen [1982], Nielsen [1986] and Li and Amos [2001].

Linear/quadratic drag

The linear and/or quadratic drag-coefficient methods depend only on velocity components u and v in the bottom grid cell and constant, spatially-uniform coefficients γ_1 and γ_2 specified as input:

$$\tau_{bx} = \left(\gamma_1 + \gamma_2 \sqrt{u^2 + v^2}\right) u$$
$$\tau_{by} = \left(\gamma_1 + \gamma_2 \sqrt{u^2 + v^2}\right) v$$

where γ_1 is the linear drag coefficient and γ_2 is the quadratic drag coefficient. The user can choose between linear or quadratic drag by setting one of these coefficients to zero. The bottom stresses computed from these formulae depend on the elevation of u and v (computed at the vertical mid-elevation of the bottom computational cell). Therefore, in this s-coordinate model, the same drag coefficient will be imposed throughout the domain even though the vertical location of the velocity is different.

Logarithmic drag (with roughness length z_0)

To prevent this problem, the quadratic drag γ_2 can be computed assuming that flow in the BBL has the classic vertical logarithmic profile defined by a shear velocity u_* and bottom roughness length z_0 (m) as:

$$|u| = \frac{u_*}{\kappa} \ln\left(\frac{z}{z_0}\right)$$

where $|u| = \sqrt{u^2 + v^2}$, friction velocity $u_* = \sqrt{\tau_b}$, z is the elevation above the bottom (vertical mid-elevation point of the bottom cell), $\kappa = 0.41$ is von Kármán's constant. z_0 is an empirical parameter. It can be constant (default) or spatially varying. Kinematic stresses are calculated as`

$$\tau_{bx} = \frac{\kappa^2}{\ln^2\left(z/z_0\right)} \sqrt{u^2 + v^2} u$$
$$\tau_{by} = \frac{\kappa^2}{\ln^2\left(z/z_0\right)} \sqrt{u^2 + v^2} v$$

The advantage of this approach is that the velocity and the vertical elevation of that velocity are used in the equation. Because the vertical elevation of the velocity in the bottom computational cell will vary spatially and temporally, the inclusion of the elevation provides a more consistent formulation.

Combined wave-current drag (BBL)

To provide a more physically relevant value of z_0, especially when considering waves and mobile sediments, a more complex formulation is available (BBL).

The short (order 10-s) oscillatory shear of wave-induced motions in a thin (a few cm) wave-boundary layer produces turbulence and generates large instantaneous shear stresses. The turbulence enhances momentum transfer, effectively increasing the bottom-flow coupling and the frictional drag exerted on the wave-averaged flow. The large instantaneous shear stresses often dominate sediment resuspension and enhance bedload transport. Sediment

transport can remold the bed into ripples and other bedforms, which present roughness elements to the flow. Bed-load transport can also induce drag on the flow, because momentum is transferred to particles as they are removed from the bed and accelerated by the flow. Resuspended sediments can cause sediment-induced stratification and, at high concentrations, change the effective viscosity of the fluid.

The BBL parameterization implemented in CROCO requires inputs of velocities u and v at reference elevation z, representative wave-orbital velocity amplitude u_b, wave period T, and wave propagation direction θ (degrees, clockwise from north). The wave parameters may be the output of a wave model such as WKB or WW3 or simpler calculations based on specified surface wave parameters. Additionally the BBL models require bottom sediment characteristics (median grain diameter D_{50}, mean sediment density ρ_s, and representative settling velocity w_s); these are constant (ANA_BSEDIM) or based on the composition of the uppermost active layer of the bed sediment during the previous time step if the sediment model is used.

The wave-averaged, combined wave–current bottom stress is expressed as function of τ_w and τ_c (i.e., the stress due to waves in the absence of currents and due to currents in the absence of waves, respectively) according to Soulsby [1995]:

$$\bar{\tau}_{wc} = \tau_c \left(1 + 1.2 \left(\frac{\tau_w}{\tau_w + \tau_c} \right)^{3.2} \right)$$

The maximum wave–current shear stress within a wave cycle is obtained by adding $\bar{\tau}_{wc}$ and τ_w (with ϕ the angle between current and waves):

$$\tau_{wc} = \left((\bar{\tau}_{wc} + \tau_w \cos\phi)^2 + (\tau_w \sin\phi)^2 \right)^{1/2}$$

The stresses τ_c and τ_w are determined using:

$$\tau_c = \frac{\kappa^2}{\ln^2 (z/z_0)} |u|^2$$
$$\tau_w = 0.5 \rho f_w u_b^2$$

u_b, the bottom orbital velocity, is determined from the significant wave height H_s and peak frequency ω_p using the Airy wave theory:

$$u_b = \omega_p \frac{H_s}{2 \sinh kh}$$

with h the local depth and k the local wave number from the dispersion relation. The wave-friction factor f_w is, according to Soulsby [1995]:

$$f_w = 1.39 (u_b / \omega_p z_0)^{-0.52}$$

The wave–current interaction in the BBL is taken into account only if $u_b > 1$ cm/s; otherwise, current-only conditions apply.

Shear stress for sediment resuspension and roughness length due to bed form

To determine the shear stress relevant for sediment resuspension and the roughness length due to bed forms, we follow the concept of Li and Amos [2001] briefly summarized here. First, the maximum wave–current skin friction τ_s is computed from the equations above, using the Nikuradse roughness $z_0 = D_{50}/12$.

A bed-load layer develops as soon as the maximum wave–current skin friction τ_s exceeds the critical stress τ_{cr}. This layer affects the stress effective for ripple formation and sediment resuspension. Subsequently, for sandy locations, ripple height and length are computed, leading to a space- and time-dependent ripple roughness length $z_0 = z_{rip}$, which is used to compute the drag on the flow (instead of a constant value when BBL is not activated). This drag provides boundary conditions to the momentum and turbulence equations (KPP or GLS).

1.13.2 Sediment models

There are two sediment models in CROCO: the USGS model derived from the UCLA/USGS ROMS community, and MUSTANG derived from the Ifremer SIAM/MARS community.

1.13.2.1 USGS Sediment Model

This USGS sediment model is derived from the UCLA/USGS ROMS community. See Blaas *et al.* [2007], Warner *et al.* [2008] and Shafiei [2021] for details.

Regarding the time and space resolution considered, the explicit solution generally refers to quantities averaged over wave periods, although the implementation of a nonhydrostatic solver in CROCO opens the way to a wave-resolved approach. One of the crucial ingredients in the sediment transport model is a reliable representation of wave-averaged (or wave-resolved) hydrodynamics and turbulence.

In the wave-averaged approach, the wave boundary layer is not resolved explicitly, but the lower part of the velocity and sediment concentration profile in the current boundary layer is important for the calculation of the sediment transport rates. Similarly, an accurate assessment of the bottom boundary shear stress (including wave effects) is required since it determines the initiation of grain motion and settling and resuspension of suspended load (see BBL). Thus, the sediment concentration and current velocity profiles in the unresolved part of the near-bottom layer have to be parameterized. Characterization of the sediments (mainly density and grain size, making general assumptions about shape and cohesiveness) is done either as a time-dependent prescribed function at the point sources or at the sea bed as an initial (soon space-dependent) condition. Sediment concentration may be considered as passive with respect to the flow density or as active if concentration values require such (the latter is not implemented yet).

1.13.2.1.1 Sediment bed

The sediment bed is represented by three-dimensional arrays with a fixed number of layers beneath each horizontal model cell. Each cell of each layer in the bed is initialized with a thickness, sediment-class distribution, porosity, and age. The mass of each sediment class in each cell can be determined from these values and the grain density. The bed framework also includes two-dimensional arrays that describe the evolving properties of the seabed, including bulk properties of the surface layer (active layer thickness, mean grain diameter, mean density, mean settling velocity, mean critical stress for erosion) and descriptions of the subgrid scale morphology (ripple height and wavelength). These properties are used to estimate bed roughness in the BBL formulations and feed into the bottom stress calculations. The bottom stresses are then used by the sediment routines to determine resuspension and transport, providing a feedback from the sediment dynamics to the hydrodynamics.

The bed layers are modified at each time step to account for erosion and deposition and track stratigraphy. At the beginning of each time step, an active layer thickness z_a is calculated [Harris and Wiberg, 1997]. z_a is the minimum thickness of the top bed layer. If the top layer is thicker than z_a, no action is required. If the top layer is less than z_a, then the top layer thickness is increased by entraining sediment mass from deeper layers until the top layer thickness equals z_a. If sediment from deeper than the second layer is mixed into the top layer, the bottom layer is split to enforce a constant number of layers and conservation of sediment mass. Each sediment class can be transported by suspended-load and/or bedload (below). Suspended-load mass is exchanged vertically between the water column and the top bed layer. Mass of each sediment class available for transport is limited to the mass available in the active layer. Bedload mass is exchanged horizontally within the top layer of the bed. Mass of each sediment class available for transport is limited to the mass available in the top layer. Suspended-sediment that is deposited, or bedload that is transported into a computational cell, is added to the top bed layer. If continuous deposition results in a top layer thicker than a user-defined threshold, a new layer is provided to begin accumulation of depositing mass. The bottom two layers are then combined to conserve the number of layers. After erosion and deposition have been calculated, the active-layer thickness is recalculated and bed layers readjusted to accommodate it. This step mixes away any very thin layer (less than the active layer thickness) of newly deposited material. Finally the surficial sediment characteristics, such as D50, ripple geometry, etc., are updated and made available to the bottom stress calculations.

1.13.2.1.2 Suspended-sediment transport

The concentration of sediment suspended in the water column is transported, like other conservative tracers (e.g., temperature and salinity) by solving the advection–diffusion equation with a source/sink term for vertical settling and erosion:

$$\underbrace{\frac{\partial C}{\partial t}}_{RATE} = - \underbrace{\vec{\nabla}.\vec{\mathbf{v}}C}_{ADVECTION} + \underbrace{\mathcal{D}_C}_{MIXING} - \underbrace{\frac{\partial w_s C}{\partial z}}_{SETTLING} + \underbrace{\frac{E}{\delta z_b}\Big|_{z=z_b}}_{EROSION}$$

C is the Reynolds-averaged, wave-averaged (unless used in wave-resolving mode) sediment concentration of a particular size class; $\vec{\mathbf{v}}$ is the flow velocity (it is the Lagrangian velocity $\vec{\mathbf{v}_L}$ in wave-averaged equations, comprising the Stokes drift $\vec{\mathbf{v}_S}$).

For each size class, the source or sink term represents the net of upward flux of eroded material E and downward settling, i.e., the deposition flux. w_s is the settling velocity, dependent on sediment grain size, but independent of flow conditions and concentrations. It is an input parameter of the model (WSED in sediment.in; see below). Settling is computed via a semi-Lagrangian advective flux algorithm, which is unconditionally stable [Durran, 2010]. It uses a piece-wise parabolic vertical reconstruction of the suspended sediment for high-order interpolation, with WENO constraints to avoid oscillations. E is the erosion flux at the sea floor and is only applied to the first grid level of height z_b and cell size δz_b. The erosion flux for each class is given by:

$$E = E_0(1-p)\,\phi\left(\frac{\tau_s}{\tau_c} - 1\right) \text{ for } \tau_s > \tau_c$$

E_0 is an empirical erosion rate (ERATE parameter in sediment.in; see below); p is the sediment porosity; ϕ is the volumetric fraction of sediment of the class considered; τ_c is the critical shear stress; and τ_s is the shear stress magnitude on the grains (skin stress due to wave-induced bed orbital velocities and mean bottom currents; see BBL). The critical shear stress is the threshold for the initiation of sediment motion.

Zero-flux boundary conditions are imposed at the surface and bottom in the vertical diffusion equation. Lateral open boundaries are treated as other tracers according to Marchesiello *et al.* [2001]. A quasi-monotonic 5th-order advection scheme (WENO5-Z, Borges *et al.* [2008]) can be used for horizontal and vertical advection of all tracers, including sediments.

1.13.2.1.3 Bedload transport

The bedload flux q_b, which is considered unresolved by the model can be calculated using different bedload models implemented in CROCO. The formulation by Meyer-Peter Muller [Meyer-Peter and Müller, 1948] is suited to rivers or continental shelf problems, where nonlinear wave effects are small. For nearshore applications, where wave nonlinearity is important, the bedload transport formulation proposed by van der A et al. (2013) is implemented as in Shafiei [2021] following Kalra *et al.* [2019] with some modifications.

Each formulation depends on the characteristics of individual sediment classes, including median size d_{50}, grain density ρ_s, specific density in water $s = \rho/\rho_s$, and critical shear stress τ_c. Non-dimensional transport rates Φ are calculated for each sediment class and converted to dimensional bedload transport rates q_b using:

$$q_b = \Phi\sqrt{(s-1)gd_{50}^3\rho_s}$$

These are horizontal vector quantities with directions that correspond to the combined bed-stress vectors. Details on the computation of Φ differs in the Meyer-Peter Müeller or van der A formulations.

Slope effect: bedload fluxes are corrected to account for the avalanche process, i.e., the gravitational flow of sand occuring when the bottom slope exceeds the critical slope angle:

$$q_{b,slope} = q_b\left(\frac{0.65}{(0.65 - \tan\beta)\cos\beta}\right),$$

This correction considers the effect of the bed slope $\beta = \tan^{-1}(dz_b/dx)$. The value 0.65 is derived from the consideration of an angle of repose of 33°.

Bedload numerics: bedload fluxes are computed at grid-cell centers and are limited by the availability of each sediment class in the top layer. Fluxes are then interpolated on cell faces using an upwind approach, either 1rst-order (e.g., Lesser *et al.* [2004]) or 5th order, or even a WENO5 interpolation to avoid oscillations. Flux differences are then used to determine changes of sediment mass in the bed at each grid cell.

1.13.2.1.3.1 Meyer-Peter Müller : Transport by currents

Meyer-Peter Müller [Meyer-Peter and Müller, 1948] formulation :

$$\Phi = max\left[8(\theta_s - \theta_c)^{1.5}, 0\right]$$

where Φ is the magnitude of the non-dimensional transport rate for each sediment class, θ_s is the non-dimensional Shields parameter for skin stress:

$$\theta_s = \frac{\tau_s}{(s-1)gd_{50}}$$

θ_c is the critical Shields parameter, and τ_s the magnitude of total skin-friction component of bottom stress computed from:

$$\tau_s = \sqrt{\tau_{sx}^2 + \tau_{sy}^2}$$

where τ_{sx} and τ_{sy} are the skin-friction components of bed stress, from currents alone or the maximum wave-current combined stress, in the x and y directions. These are computed at cell faces (u and v locations) and then interpolated to cell centers (ρ points). The bedload transport vectors are partitioned into x and y components based on the magnitude of the bed shear stress as:

$$q_{bx} = q_b \frac{\tau_{sx}}{\tau_s}$$
$$q_{by} = q_b \frac{\tau_{sy}}{\tau_s}$$

1.13.2.1.3.2 van der A (2013): Transport by nonlinear waves

The SANTOSS bedload model is based on the half-wave cycle concept proposed by Dibajnia and Watanabe [1993] that captures asymmetric transport by non-linear waves and the effect of phase lag between mobilization and transport. CROCO contains an adapted version by Shafiei [2021], based on the implementation of Kalra *et al.* [2019]. In our formulation, the effect of wave-averaged currents is removed by default (assuming that the transport by currents is effectively performed by the suspended load model) and it thus only retains the nonlinear effects of waves. In the brief presentation below, we retain the current terms for completeness, but focus on wave effects.

The method to obtain bedload transport under asymmetric waves can be divided into three major steps (van der A, 2013) [Kalra *et al.*, 2019]. In the first step, the asymmetric waveform based on the Ursell number is evaluated using wave statistics. The Shields parameter for each half cycle of the wave form is computed in the second step. Finally, a phase lag is estimated from the velocity and sediment concentrations that determine the amount of bedload transported in the half cycle following mobilization. The non-dimensional bedload transport rate Φ is thus given by:

$$\Phi = \frac{1}{T}\left[\frac{\theta_c}{\left|\theta_c\right|^{1/2}} T_c \left(\Omega_{cc} + \frac{T_c}{2T_{cu}}\Omega_{tc}\right) + \frac{\theta_t}{\left|\theta_t\right|^{1/2}} T_t \left(\Omega_{tt} + \frac{T_t}{2T_{tu}}\Omega_{ct}\right)\right],$$

where T, T_c, T_t, T_{cu} and T_{tu} are the wave period, duration of wave crest half cycle, duration of wave trough half cycle, duration of accelerating flow within the crest half cycle and duration of accelerating flow within the trough half cycle respectively, θ_c and θ_t represent the Shields numbers associated with the wave crest and trough half cycles. The sand load transported during the crest period is the combination of Ω_{cc} (mobilized during the crest period) and Ω_{tc} (mobilized during the trough period). Similarly, Ω_{tt} and Ω_{ct} are the sand load transported during the trough period (mobilized during the trough and crest periods respectively).

The sand load transported during each half-cycle is conventionally modeled according to a power law of Shields number:

$$\Omega_i = \max\left(11\left(\left|\theta_i\right| - \theta_{cr}\right)^{1.2}, 0\right),$$

where θ_{cr} is the critical Shields number and, hereafter, the subscript "i" is either "c" for crest or "t" for trough half cycles. To determine Ω_{ct} and Ω_{tc}, i.e., the portion of the bedload remaining in suspension to be transported in the next half cycle, a phase lag parameter is evaluated.

Let's assume a two-dimensional (x,z) cross-shore problem for simplicity. The Shields number for the peak or trough ($\theta_i = \theta_t$ or θ_c) is calculated according to:

$$\theta_i = \frac{\frac{1}{2} f_{w\delta i} |u_{i,r}| u_{i,r}}{(s-1)gd_{50}}.$$

$u_{i,r}$ is the representative cross-shore combined wave-current velocity at trough or crest half cycles calculated as:

$$u_{i,r} = \frac{\hat{u}_i}{\sqrt{2}} + |u_\delta|,$$

where $\hat{u}_i$ is the peak crest or trough orbital velocities, u_δ is the steady current velocity at the top of the wave boundary layer. $f_{w\delta i}$ is the linear wave-current friction factor at crest or trough calculated by Ribberink [1998]:

$$f_{w\delta i} = \frac{\hat{u}}{u_\delta + \hat{u}} f_{wi} + \frac{u_\delta}{u_\delta + \hat{u}} f_\delta$$

where $\hat{u}$ is the representative orbital velocity amplitude for the whole flow cycle (given by $\hat{u} = \sqrt{2} u_{orb}$). f_δ is the current-related friction factor dependent on a current-related roughness $k_{s\delta}$ and f_{wi} is the wave friction factor, calculated separately for the crest and trough half-cycles and depends on a wave-related roughness k_{sw}. If the representative orbital excursion amplitude $\hat{a} = \hat{u}T/2\pi$ is large enough (i.e., greater than 1.587 k_{sw}):

$$f_{wi} = 0.00251\, e^{5.21\left[\left(\frac{2T_{iu}}{T_i}\right)^{2.6} \frac{\hat{a}}{k_{sw}}\right]^{-0.19}},$$

otherwise, $f_{wi} = 0.3$.

If $u_\delta = 0$ ($u_{i,r} = \hat{u}_i/\sqrt{2}$ and $f_{w\delta i} = f_{wi}$), the effect of currents is completely removed from the bedload transport calculation. This choice represents our default to avoid double counting the transport by wave-averaged currents.

Finally, following Kalra *et al.* [2019], after calculating Φ, we apply to q_b a bedload factor f_{bld} (bedload_coeff in CROCO). This factor allows us to adjust the relative contribution to sediment transport of wave-induced bedload compared to the suspended load transported by mean currents. In this way, we have a better control of the antagonistic mechanisms that govern onshore and offshore transports respectively.

1.13.2.1.4 Morphology

The bed evolution (variation in time of z_b, the height of the bed), is calculated from the divergence of sediment fluxes (Exner equation), which results from the difference between erosion and sedimentation of suspended sediments. In wave-averaged equations, where residual wave effects need to be parametrized as bedload fluxes q_b, the bed evolution also arises from the divergence of these fluxes.

$$\frac{\partial z_b}{\partial t} = -\frac{f_{mor}}{1-p}\left(\frac{\partial q_b}{\partial x} - w_s \frac{\partial C}{\partial z} + E\right).$$

This equation accounts for a morphological acceleration factor f_{mor} (morph_fac in CROCO). A value of 1 has no effect, and values greater than 1 accelerate the bed response. The concept of morphological acceleration is based on the fact that morphodynamic changes are slower than hydrodynamic ones [van Rijn, 1993]. In this case, the bed evolution can be accelerated without affecting the hydro-morphological solution. The increased rate of morphological change can be useful for simulating evolution over long time periods. Strategies for morphological updating are described by Roelvink [2006] and implemented in CROCO following Warner *et al.* [2008]. In our implementation, bedload fluxes, erosion, and deposition rates are multiplied by f_{mor}, while the magnitude of sediment concentrations in the water column is not modified – just the exchange rate to and from the bed. For both bedload and suspended load, sediment is limited in availability, based on the true amount of sediment mass (not multiplied by the scale factor).

For dynamical consistency, the vertical velocity is modified (in omega.F) by the rate of change of vertical grid levels dz/dt, adjusting to the moving sea floor and free surface (grid "breathing" component; Shchepetkin and McWilliams [2005]). This method is mass conserving and retains tracer constancy preservation.

1.13.2.1.5 Sediment Density

Witt CPP SED_DENS, effects of suspended sediment on the density field are included with terms for the weight of each sediment class in the equation of state for seawater density as:

$$\rho = \rho_{water} + \sum_{m=1}^{Nsed} \frac{C_m}{\rho_{s,m}} \left(\rho_{s,m} - \rho_{water}\right)$$

This enables the model to simulate processes where sediment density influences hydrodynamics, such as density stratification and gravitationally driven flows.

Related CPP options:

SUSPLOAD	Activate suspended load transport
BEDLOAD	Activate bedload transport
MORPHODYN	Activate morphodynamics
BEDLOAD_VANDERA	van der A formulation for bedload (van der A et al., 2013)
BEDLOAD_MPM	Meyer-Peter-Muller formulation for bedload [Meyer-Peter and Müller, 1948]
SLOPE_LESSER	Lesser formulation for avalanching [Lesser *et al.*, 2004]
SLOPE_NEMETH	Nemeth formulation for avalanching (Nemeth et al, 2006)
BEDLOAD_UP1	Bedload flux interpolation: upwind 1rst order
BEDLOAD_UP5	Bedload flux interpolation: upwind 5th order
BEDLOAD_WENO5	Bedload flux interpolation: WENO 5th order
ANA_SEDIMENT	Set analytical sediment size, initial ripple and bed parameters
ANA_BPFLUX	Set kinematic bottom flux of sediment tracer (if different from 0)
SPONGE_SED	Gradually reduce erosion/deposition near open boundaries
SED_DENS	Activate the effect of suspended sediment on the density field

Preselected options:

```
#ifdef SEDIMENT
# undef  MUSTANG
# define ANA_SEDIMENT
# define SPONGE_SED
# define Z0_BL
# define Z0_RIP
# ifdef BEDLOAD
#  ifdef BEDLOAD_VANDERA       /* default BEDLOAD scheme  */
#  elif defined BEDLOAD_MPM
#  elif defined BEDLOAD_WULIN
#  elif defined BEDLOAD_MARIEU
#  else
#   if (defined WAVE_OFFLINE || defined WKB_WWAVE ||\
        defined ANA_WWAVE    || defined OW_COUPLING)
#    define BEDLOAD_VANDERA
#   else
#    define BEDLOAD_MPM
#   endif
#  endif
#  ifdef BEDLOAD_UP1           /* default INTERPOLATION  */
#  elif defined BEDLOAD_UP5
#  elif defined BEDLOAD_WENO5
#  else
#   define BEDLOAD_UP1
#  endif
#  ifdef SLOPE_LESSER          /* default SLOPE scheme */
#  elif defined SLOPE_NEMETH
#  elif defined SLOPE_KIRWAN
#  else
#   define SLOPE_LESSER
#  endif
# endif /* BEDLOAD */
 #endif /* SEDIMENT */
```

Parameters in sediment.in

```
1  Stitle (a80)
CROCO - Sediment - Test

2  Sd(1-NST), CSED, SRHO,  WSED,  ERATE,  TAU_CE, TAU_CD, BED_FRAC(1:NLAY)
     0.125    9.9   2650.  9.4    25.0e-5  0.05    0.14    0.4  0.4
     0.050    0.0   2650.  1.6     4.0e-5  0.01    0.14    0.6  0.6

3 BTHK(1:NLAY)
      1.    10.

4 BPOR(1:NLAY)
      0.41  0.42

5 Hrip
      0.03

6 Lrip
      0.14

7 bedload_coeff
      0.

8 morph_fac
     10.

9  transC
     0.03

10  transN
     0.2

11  tcr_min
     0.03

12  tcr_max
     5.5

13  tcr_slp
     0.3

14  tcr_off
     1.

15  tcr_tim
     28800.

16  L_ADS  L_ASH  L_COLLFRAG  L_TESTCASE
    F       T         F            F

17  F_DP0      F_NF    F_DMAX     F_NB_FRAG    F_ALPHA   F_BETA  F_ATER   F_ERO_FAC F_
↪ERO_NBFRAG  F_COLLFRAGPARAM  F_CLIM    F_ERO_IV
   0.000004    2.     0.0015        2.          0.35     0.15    0.         0.       ␣
↪2.             0.01          0.001        1

18  MUD_FRAC_EQ [1:NMUD]
   0.10  0.20  0.40  0.20  0.10  0.0  0.0  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
```

(continues on next page)

(continued from previous page)

```
19  MUD_T_DFLOC
   200.

99 END of sediment input data
```

GLOSSARY

CARD 1: String with a maximum of eighty characters.

- **Stitle** : Sediment case title.

CARD 2: Sediment grain parameters & initial values (NST lines)

- **Sd** : Diameter of grain size class [mm].
- **CSED** : Initial concentration (spatially uniform) [kg/m3].
- **SRHO** : Density of sediment material of size class [kg/m3]. Quartz: SRHO=2650 kg/m^3
- **WSED** : Settling velocity of size class [mm/s].

  Typically [Soulsby, 1997]:

  WSED = 10^3 $(visc\,(\sqrt{10.36^2 + 1.049D^3}\ - 10.36)\ /\ D_{50}$ [mm/s]

  with

  - $D = D_{50}\ (g\ (\mathrm{SRHO}/\rho_0 - 1)\ /\ (visc^2)\ )^{0.33333}$
  - $D_{50} = 10^{-3} Sd$ [m]
  - $visc = 1.3\ 10^{-3}/\rho_0$ [m2/s]
- **ERATE** : Erosion rate of size class [kg/m2/s].

  Typically:

  ERATE $= 10^{-3}\ \gamma_0$ WSED SRHO $[kg/m^2/s]$

  with $\gamma_0 = 10^{-3} - 10^{-5}$ [Smith and McLean, 1977]
- **TAU_CE** : Critical shear stress for sediment motion [N/m2] (initiation of bedload for coarses, suspension for fines).

  Typically :

  $\mathrm{TAU_C E} = 6.4\ 10^{-7}\ \rho_0\ \mathrm{WSED}^2$ [N/m2]
- **TAU_CD** : Critical shear stress for deposition of cohesive sediments [N/m2]
- **BED_FRAC** : Volume fraction of each size class in each bed layer (NLAY columns)

  [0<BED_FRAC<1]

CARD 3: Sediment bed thickness, 1st field is top layer ('delt_a')

- **BTHK** : Initial thicknesses of bed layers [m] Bthk(1) active layer thickness, fixed in simulation unless SUM(Bthk(:))<Bthk(1)

CARD 4: Sediment bed porosity

- **BPOR** : Initial porosity of bed layers [m] used in ana_sediment ifdef ANA_SEDIMENT (not in init.nc)

CARD 5: Bottom ripple height

- **Hrip** : Initial ripple height [m] used in ana_sediment ifdef ANA_SEDIMENT (not in init.nc)

CARD 6: Bottom ripple length

- **Lrip** : Initial ripple length [m] used in ana_sediment ifdef ANA_SEDIMENT (not in init.nc)

CARD 7: Bedload coefficient

- **bedload_coeff** : factor limiting the magnitude of bedload flux 0<bedload_coef<1

CARD 8: Morphological acceleration factor

- **morph_fac** : factor accelerating bed evolution morph_fac>=1

CARD 9 :

- **transC** : Cohesive transition- Under that value of total mud fraction entire bed behaves as a non-cohesive bed

---

CARD 10 :

- **transN** : Noncohesive transition- Over that value of total mud fraction entire bed behaves as a cohesive bed

---

CARD 11 :

- **tcr_min** : Minimum shear for erosion

CARD 12 :

- **tcr_max** : Maximum shear for erosion

CARD 13 :

- **tcr_slp** : Tau_crit profile slope

CARD 14 :

- **tcr_off** : Tau_crit profile offset

CARD 15 :

- **tcr_tim** : Tau_crit consolidation rate

CARD 16 : booleans for flocculation

- **L_ADS** : Boolean set to .true. if differential settling aggregation
- **L_ASH** : Boolean set to .true. if shear aggregation

- **L_COLLFRAG** : Boolean set to .true. if collision-induced fragmentation enable
- **L_TESTCASE** : If .TRUE. sets G(t) to values from Verney *et al.* [2011] lab experiment

---

CARD 17 : flocculation Sediment Parameters

- **F_DP0** : Primary particle size (m), typically 4e-6 m
- **F_NF** : Floc fractal dimension, typically ranging from 1.6 to 2.6
- **F_DMAX** : Maximum diameter (m)

1.13.2.2 MUSTANG Sediment model

1.13.2.2.1 MUSTANG presentation

MUSTANG (**MU**d and **S**and **T**ran **ANS**port modelli **NG**) is a sediment module. Comparing to a pure hydrodynamical simulation, it adds variables relatives to sediment behavior such as sediment concentration in water and sediment bed evolution. This module has been developped by **Le Hir et al**, coupled with the hydrodynamic model **SiAM**. Then, it has continued to evolve, coupled with the parallelized **MARS3D** hydrodynamic model. It is now renamed **MUSTANG** (MUd and Sand TranSport modelliNG) and is available in the **CROCO** community model since release 1.2.

The module provides for the modeling of two types of sediment transport:

- **Bedload**: sediment particles are set in motion from a certain critical tension. They move by staying close to the bottom
- **Suspension**: sediment particles are suspended in the water column. They are transported in the water column as a passive tracer with a settling velocity through the advection-diffusion equation and eventually settle to the bottom.

MUSTANG module deals with various sediments classified into 3 types: GRAVEL, SAND and MUD. Gravels are transported only in bedload (no suspension). Sands can be transported by both types. Muds are transported only in suspension (no bedload).

From a sedimentary bottom consisting of a set of sedimentary facies, each of which is defined by a proportion of each of the classes, hydrosedimentary modeling consists of changing the proportions of each class of the sedimentary bottom as well as the thickness of the layers which constitute it, and therefore the bathymetry. It is then possible to inject the new bathymetry thus obtained into the hydrodynamic module (morphodynamic coupling).

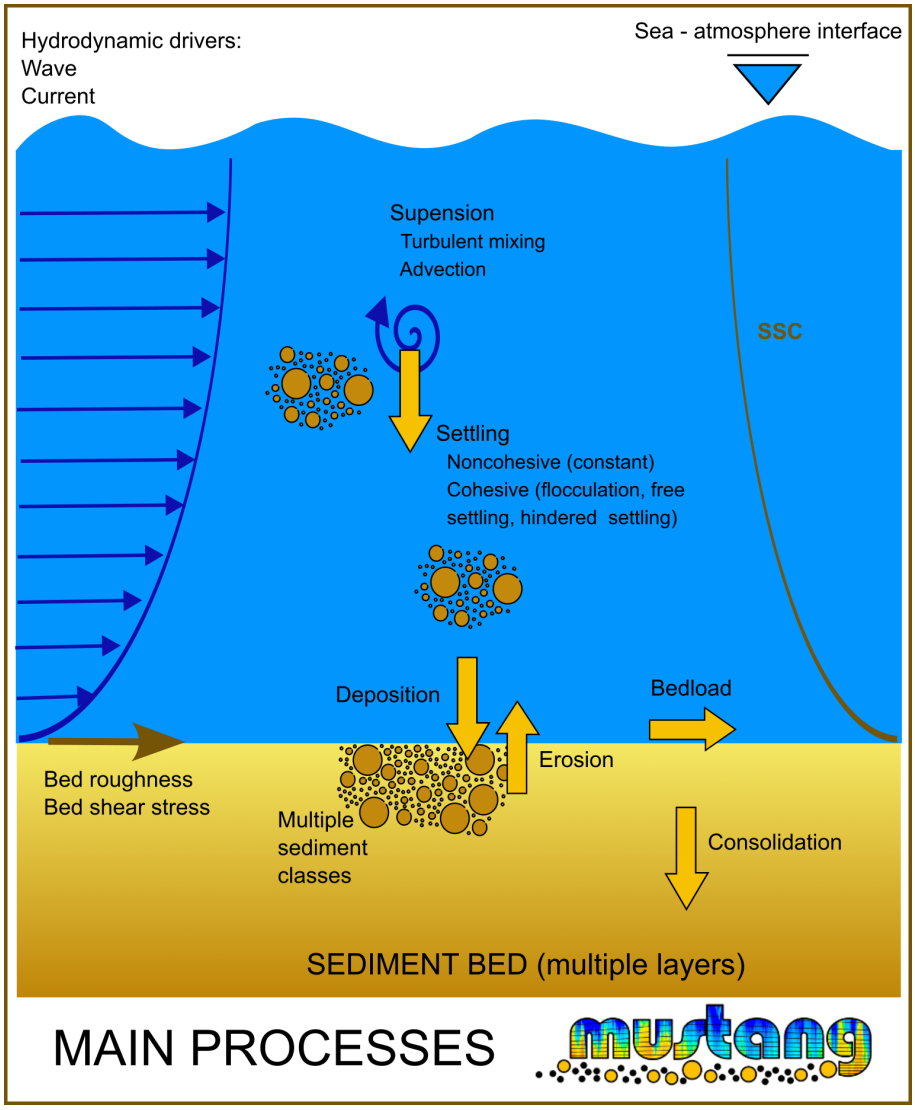
MUSTANG computes these evolutions by taking into account several processes such as settling, flocculation, erosion fluxes, deposit fluxes, consolidation (porosity) in sediment bed, bedload transport for non-cohesive sediment, morphodynamic…

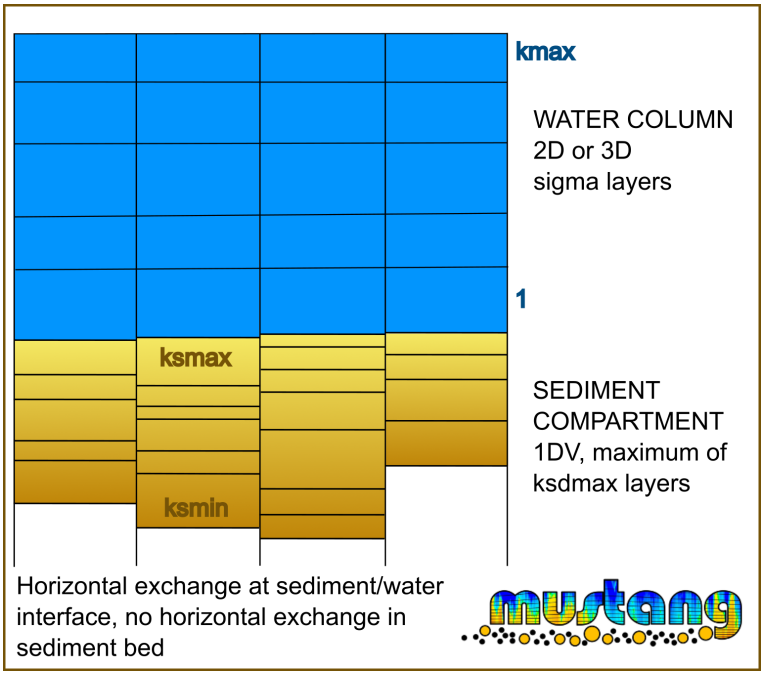
There are 2 main options for this module:

- One equivalent to the previous module "mixsed" [Le Hir *et al.*, 2011] - **default**
- One developped by Mengual *et al.* [2021], which includes bedload processes [Rivier *et al.*, 2017] - **activated by cppkey #key_MUSTANG_V2**

MUSTANG sediment compartment is 1DV in the sediment part, all exchanges between horizontal meshes, such as bedload, lateral erosion or sliding effect, are done at the interface between sediment and water.

MUSTANG consider 3 types of sediment but all features are not available for all type of sediment:


Hydrodynamic drivers:
Wave
Current
Sea - atmosphere interface
Supension
Turbulent mixing
Advection
SSC
Settling
Noncohesive (constant)
Cohesive (flocculation, free settling, hindered settling)
Deposition
Bedload
Erosion
Bed roughness
Bed shear stress
Multiple sediment classes
Consolidation
SEDIMENT BED (multiple layers)
MAIN PROCESSES
mustang


kmax
WATER COLUMN
2D or 3D
sigma layers
1
ksmax
SEDIMENT
COMPARTMENT
1DV, maximum of
ksdmax layers
ksmin
Horizontal exchange at sediment/water interface, no horizontal exchange in sediment bed
mustang

Type		Suspended load	
	Bedload transport (if #key_MUSTANG_V2 & #key_MUSTANG_bedloac		Flocculation (if #key_MUSTANG_flocmod)
MUD	NO	YES	YES
SAND	YES	YES	NO
GRAVEL	YES	NO	NO

Other substances can be defined via the susbstance module and can interact with sediment depending on their characteristics.

This documentation presents:

- in MUSTANG *user guide*, input and output files format are detailed and available cppkeys are listed with a description of the effect of each key and a list of compatibility/incompatibility between keys
- in MUSTANG *technical documentation*, a description of the insertion of MUSTANG calls in the CROCO temporal loop and a description of the modelisation of the main processes

1.13.2.2.2 MUSTANG user guide

MUSTANG module can not be used alone, it needed an hydrodynamic model and the substance module.

In this documentation, the hydrodynamic model is **CROCO**. Hydrodynamic parametrization will not be detailed here, please refer to hydrodynamic module documentation.

To use MUSTANG in CROCO you will need to:

- define appropriate cppkeys in **cppdefs.h**. To activate MUSTANG within CROCO environment, the **MUSTANG** and **SUBSTANCE** cppkeys must be defined. To activate certains processes other cppkeys must be defined (see *available cppkeys*)
- define MUSTANG and SUBSTANCE files in CROCO input file **croco.in**, these file contains MUSTANG parameters and SUBSTANCE characteristics (see *MUSTANG namelist* and *SUBSTANCE namelist*)
- define appropriate dimensions in **param.h** file
- depending on the options you choose via your cppkeys and input file (MUSTANG namelist and SUBSTANCE namelist), you will have to define other input file such as *initialization file*, *wave file*, *source with solid discharge file*

Then you can compile and run CROCO as any other CROCO run.

> **Note**
>
> MUSTANG is controlled through both CPP keys and input files. For some processes it is needed to activate the options through a CPP key, and also through a flag (true or false) in the input files

1.13.2.2.2.1 Input file: croco.in

The name and location of the MUSTANG and SUBSTANCE input files are defined in CROCO input file **croco.in** as follow:

```
sediments_mustang: input file
               MUSTANG_NAMELIST/parasubstance.txt
               MUSTANG_NAMELIST/paraMUSTANG.txt
```

In this example, the files are located in MUSTANG_NAMELIST directory but they could be placed in any directory. Therefore, example and test cases namelists could be found in MUSTANG_NAMELIST directory in MUSTANG source directory.

In particular, in MUSTANG_NAMELIST directory (relative path from the current directory where croco executable is), a file **paraMUSTANG_default.txt** contains all default values used, if the user does not specify a parameter in the namelist file, the default value in MUSTANG_NAMELIST/paraMUSTANG_default.txt is used.

There is no default file for susbstance but a full example is given in MUSTANG_NAMELIST directory in MUSTANG source directory.

1.13.2.2.2.2 Input file: param.h

Before the code compilation, the following dimensions must be defined in the **param.h** file to use SUBSTANCE and MUSTANG:

- ksdmin and ksdmax: integers corresponding to the layers of sediment (sediment variables are allocated with ksdim:ksdmax dimension)
- ntrc_subs: number of substance corresponding to a tracer (advected)
- ntfix: number of fixed substance (not advected)
- ntrc_substot: total number of substance (= ntrc_subs + ntfix)

If these dimensions are modified, the code must be compiled again.

1.13.2.2.2.3 Input file: Substance namelist

Substance module allow to define 8 different types of substance with different behavior. Sediments are defined as MUD, SAND or GRAVEL.

Substance input file contains at least 5 namelists and at most 10 namelists depending on the cppkeys MUSTANG, key_benthic and SUBSTANCE_SUBMASSBALANCE:

- *nmlnbvar*: number of each type of substance to be defined (other than T (temperature) & S (salinity))
- *nmlpartnc*: characterization of Non Constitutive Particulate subtances
- *nmlpartsorb*: characterization of particulate susbtances sorbed on an other particule
- *nmlvardiss*: characterization of dissolved susbtances
- *nmlvarfix*:characterization of fixed susbtances (not advected)
- *nmlgravels* (if MUSTANG only): characterization of GRAVEL substances
- *nmlsands* (if MUSTANG only): characterization of SAND substances
- *nnmlmuds* (if MUSTANG only): characterization of MUD substances
- *nmlvarbent* (if key_benthic only): characterization of benthic substances
- *nmlsubmassbalance* (if SUBSTANCE_SUBMASSBALANCE only): parameters for submassbalance computation (see also: *Details on submassbalance computation*)

Each namelist is described bellow with the description of each parameter.

&nmlnbvar namelist:

- **nv_dis**: number of dissolved susbtances
- **nv_ncp**: number of Non Constitutive Particulate subtances, like for example trace metals, bacteria, nitrogen, organic matter.
- **nv_fix**: number of fixed susbtances (not advected)
- **nv_bent**: number of benthic susbtances
- **nv_grav**: number of Constitutive susbtances type GRAVELS (only if cppkey MUSTANG)

- **nv_sand**: number of Constitutive susbtances type SAND (only if cppkey MUSTANG)
- **nv_mud**: number of Constitutive susbtances type MUD (only if cppkey MUSTANG)
- **nv_sorb**: number of particulate susbtances sorbed on an other particule

> **Note**
>
> If MUSTANG cpp key is not defined, **nv_grav**, **nv_sand** and **nv_mud** are not read and **must not be present**

&nmlgravels namelist: each parameter is a vector of length nv_grav

- **name_var_n()**: name of variable
- **long_name_var_n()**: long name of variable
- **standard_name_var_n()**: standard name of variable
- **unit_var_n()**: string, unit of concentration of variable
- **flx_atm_n()**: uniform atmospherical deposition (unit/m2/s)
- **cv_rain_n()**: concentration in rainwater (kg/m3 of water)
- **cini_wat_n()**: initial concentration in water column (kg/m3)
- **cini_air_n()**: initial concentration in air
- **l_out_subs_n()**: saving in output file if TRUE
- **init_cv_name_n()**: name of substance read from initial condition file
- **obc_cv_name_n()**: name of substance read from obc file
- **cini_sed_n()**: initial fraction in the seafloor (only if cppkey MUSTANG)
- **tocd_n()**: critical stress of deposition (N/m2) (only if cppkey MUSTANG)
- **ros_n()**: density of particle (kg/m3) (only if cppkey MUSTANG)
- **l_bedload_n()**: allow bedload transport if true (only if cppkey MUSTANG and key_MUSTANG_V2 and key_MUSTANG_bedload)
- **diam_n()**: diameter of particles (m)

> **Note**
>
> If nv_grav = 0 this namelist is not read.

&nmlsands namelist: each parameter is a vector of length nv_sand

- **name_var_n()**: name of variable
- **long_name_var_n()**: long name of variable
- **standard_name_var_n()**: standard name of variable
- **unit_var_n()**: string, unit of concentration of variable
- **flx_atm_n()**: uniform atmospherical deposition (unit/m2/s)
- **cv_rain_n()**: concentration in rainwater (kg/m3 of water)
- **cini_wat_n()**: initial concentration in water column (kg/m3)
- **cini_air_n()**: initial concentration in air
- **l_out_subs_n()**: saving in output file if TRUE
- **init_cv_name_n()**: name of substance read from initial condition file

- **obc_cv_name_n()**: name of substance read from obc file
- **cini_sed_n()**: initial fraction in the seafloor (only if cppkey MUSTANG)
- **tocd_n()**: critical stress of deposition (N/m2) (only if cppkey MUSTANG)
- **ros_n()**: density of particle (kg/m3) (only if cppkey MUSTANG)
- **l_bedload_n()**: allow bedload transport if true (only if cppkey MUSTANG and key_MUSTANG_V2 and key_MUSTANG_bedload)
- **diam_n()**: diameter of particles (m)
- **l_sand2D()**: treat sand variable as 2D variable if true (used only if key_sand2D, see *Treatment of high settling velocities: SAND variables*)
- **l_outsandrouse()**: if true, use a reconstitution of a ROUSE profil for output in water column (used only if key_sand2D and l_sand2D is TRUE for this variable, see *Treatment of high settling velocities: SAND variables*)

Note

If nv_sand = 0 this namelist is not read.

Warning

If you have several sands in your simulation: start with the coarser sands and continue more and more finely

&nmlmuds namelist: each parameter is a vector of length nv_mud

- **name_var_n()**: name of variable
- **long_name_var_n()**: long name of variable
- **standard_name_var_n()**: standard name of variable
- **unit_var_n()**: string, unit of concentration of variable
- **flx_atm_n()**: uniform atmospherical deposition (unit/m2/s)
- **cv_rain_n()**: concentration in rainwater (kg/m3 of water)
- **cini_wat_n()**: initial concentration in water column (kg/m3)
- **cini_air_n()**: initial concentration in air
- **l_out_subs_n()**: saving in output file if TRUE
- **init_cv_name_n()**: name of substance read from initial condition file
- **obc_cv_name_n()**: name of substance read from obc file
- **cini_sed_n()**: initial fraction in the seafloor (only if cppkey MUSTANG)
- **tocd_n()**: critical stress of deposition (N/m2) (only if cppkey MUSTANG)
- **ros_n()**: density of particle (kg/m3) (only if cppkey MUSTANG)
- **cobc_wat_n()**: boundaries uniform and constant concentration (kg/m3)
- **ws_free_opt_n()**: integer, choice of free settling formulation: 0 constant, 1 Van Leussen, 2 Winterwerp, 3 Wolanski (see *Settling velocity*)
- **ws_free_min_n()**: minimum setling velocity (m/s) (see *Settling velocity*)
- **ws_free_max_n()**: maximum setling velocity (m/s) (see *Settling velocity*)
- **ws_free_para_n(1:4,num substance)**: 4 additional parameters (see *Settling velocity*)

- **ws_hind_opt_n()**: choice of hindered settling formulation: 0 no hindered settling, 1 Scott, 2 Winterwerp, 3 Wolanski (see *Settling velocity*)
- **ws_hind_para_n(1:2,num substance)**: 2 additional parameters (see *Settling velocity*)
- **diam_n()**: diameter of particles (m) (used only if #key_mustang_flocmod is activated see *FLOCMOD*)

> **Note**
>
> If nv_mud = 0 this namelist is not read.

&nmlpartnc namelist: each parameter is a vector of length nv_ncp

- **name_var_n()**: name of variable
- **long_name_var_n()**: long name of variable
- **standard_name_var_n()**: standard name of variable
- **unit_var_n()**: string, unit of concentration of variable
- **flx_atm_n()**: uniform atmospherical deposition (unit/m2/s)
- **cv_rain_n()**: concentration in rainwater (kg/m3 of water)
- **cini_wat_n()**: initial concentration in water column (kg/m3)
- **cini_air_n()**: initial concentration in air
- **l_out_subs_n()**: saving in output file if TRUE
- **init_cv_name_n()**: name of substance read from initial condition file
- **obc_cv_name_n()**: name of substance read from obc file
- **cini_sed_n()**: initial concentration in sediment (quantity/kg of dry sediment) (only if cppkey MUSTANG)
- **tocd_n()**: critical stress of deposition (N/m2) (only if cppkey MUSTANG)
- **ros_n()**: density of particle (kg/m3) (only if cppkey MUSTANG)
- **cobc_wat_n()**: boundaries uniform and constant concentration (kg/m3)
- **ws_free_opt_n()**: integer, choice of free settling formulation: 0 constant, 1 Van Leussen, 2 Winterwerp, 3 Wolanski (see *Settling velocity*)
- **ws_free_min_n()**: minimum setling velocity (m/s) (see *Settling velocity*)
- **ws_free_max_n()**: maximum setling velocity (m/s) (see *Settling velocity*)
- **ws_free_para_n(1:4,num substance)**: 4 additional parameters (see *Settling velocity*)
- **ws_hind_opt_n()**: choice of hindered settling formulation: 0 no hindered settling, 1 Scott, 2 Winterwerp, 3 Wolanski (see *Settling velocity*)
- **ws_hind_para_n(1:2,num substance)**: 2 additional parameters (see *Settling velocity*)

> **Note**
>
> If nv_ncp = 0 this namelist is not read.
>
> If MUSTANG cpp key is not defined, **cini_sed_n**, **tocd_n**, **ros_n**, **ws_free_opt_n**, **ws_free_para_n**, **ws_hind_opt_n**, **ws_hind_para_n** are not read and must not be present

&nmlpartsorb namelist: each parameter is a vector of length nv_sorb

- **name_var_n()**: name of variable
- **long_name_var_n()**: long name of variable

- **standard_name_var_n()**: standard name of variable
- **unit_var_n()**: string, unit of concentration of variable
- **flx_atm_n()**: uniform atmospherical deposition (unit/m2/s)
- **cv_rain_n()**: concentration in rainwater (kg/m3 of water)
- **cini_wat_n()**: initial concentration in water column (kg/m3)
- **cini_air_n()**: initial concentration in air
- **l_out_subs_n()**: saving in output file if TRUE
- **init_cv_name_n()**: name of substance read from initial condition file
- **obc_cv_name_n()**: name of substance read from obc file
- **cini_sed_n()**: initial concentration in sediment (quantity/kg of dry sediment) (only if cppkey MUSTANG)
- **cobc_wat_n()**: boundaries uniform and constant concentration (kg/m3)
- **name_varpc_assoc_n()**: name of associated particulate substance on which this substance is sorbed

> **Note**
>
> If nv_sorb = 0 this namelist is not read.
>
> If MUSTANG cpp key is not defined, **cini_sed_n** is not read and must not be present

&nmlvardiss namelist: each parameter is a vector of length nv_dis

- **name_var_n()**: name of variable
- **long_name_var_n()**: long name of variable
- **standard_name_var_n()**: standard name of variable
- **unit_var_n()**: string, unit of concentration of variable
- **flx_atm_n()**: uniform atmospherical deposition (unit/m2/s)
- **cv_rain_n()**: concentration in rainwater (kg/m3 of water)
- **cini_wat_n()**: initial concentration in water column (kg/m3)
- **cini_air_n()**: initial concentration in air
- **l_out_subs_n()**: saving in output file if TRUE
- **init_cv_name_n()**: name of substance read from initial condition file
- **obc_cv_name_n()**: name of substance read from obc file
- **cini_sed_n()**: initial concentration in sediment (quantity/kg of dry sediment) (only if cppkey MUSTANG)
- **cobc_wat_n()**: boundaries uniform and constant concentration (kg/m3)

> **Note**
>
> If nv_dis = 0 this namelist is not read.
>
> If MUSTANG cpp key is not defined, **cini_sed_n** is not read and must not be present

&nmlvarfix namelist: each parameter is a vector of length nv_fix

- **name_var_fix()**: name of variable
- **long_name_var_fix()**: long name of variable
- **standard_name_var_fix()**: standard name of variable

- **unit_var_fix()**: string, unit of concentration of variable
- **cini_wat_fix()**: initial concentration in water column (kg/m3)
- **l_out_subs_fix()**: saving in output file if TRUE
- **init_cv_name_fix()**: name of substance read from initial condition file

> **Note**
>
> If nv_fix = 0 this namelist is not read.

&nmlvarbent namelist: each parameter is a vector of length nv_bent

- **name_var_bent()**: name of variable
- **long_name_var_bent()**: long name of variable
- **standard_name_var_bent()**: standard name of variable
- **unit_var_bent()**: string, unit of concentration of variable
- **cini_bent()**: initial concentration
- **l_out_subs_bent()**: saving in output file if TRUE

> **Note**
>
> If nv_bent = 0 or cppkey key_benthic is not defined, this namelist is not read.

&nmlsubmassbalance namelist: (see also: *Details on submassbalance computation*)

- **submassbalance_l**: activate submassblance computation if TRUE and if cppkeys SUBSTANCE_SUBMASSBALANCE is defined
- **submassbalance_nb_border**: number of polygons and lines
- **submassbalance_input_file**: path of the input file (see format information here: *submassbalance input file format*) (if '' or 'all_domain', only one budget zone is taking into account (all the domain) and any fluxes threw open borders)
- **submassbalance_output_file**: path of the output file (in netcdf, more information here: *submassbalance output file format*)
- **submassbalance_dtout**: = output frequency in hours (h)
- **submassbalance_date_start**: = starting date for budget/fluxes computing, exemple '1999/01/01 00:00:00'

> **Note**
>
> If cppkey SUBSTANCE_SUBMASSBALANCE is not defined, this namelist is not read.

1.13.2.2.2.4 Input file: Mustang namelist

> **Note**
>
> If the user does not specify a parameter in the namelist file, the default value in MUSTANG_NAMELIST/paraMUSTANG_default.txt is used

This default input files incorporates all the parameters that are needed for both V1 or V2 MUSTANG versions. Therefore, not all parameters are used, depending on the set of CPP keys or booleans included in the MUSTANG input file itself.

All the parameters in the paraMUSTANG_default.txt are read first, before reading the user-defined input file defined in croco.in. The parameters defined in the user-defined namelist file will overwrite those defined in the default file. The user-defined input file can either be a full copy of the default input file, or only define the parameters that matter the most for a specific configuration. In the later case, even if the parameters are not mentionned, the namelist group section needs to be present in the file even if empty, e.g.:

```
&namsedim_deposition
/
```

Mustang input file contains at least 10 namelists and at most 16 namelists depending on the cpp-keys key_MUSTANG_V2, key_MUSTANG_debug, key_MUSTANG_flocmod key_MUSTANG_lateralerosion, key_noTSdiss_insed:

- *namsedim_init*: relative to sediment initialization
- *namsedim_layer*: relative to sediment layers characterization and active layer
- *namsedim_bottomstress*: relative to bottom shear stress
- *namsedim_deposition*: relative to sediment deposition
- *namsedim_erosion*: relative to sediment erosion
- *namsedim_poro*: relative to porosity
- *namsedim_bedload*: relative to sediment bedload (only if key_MUSTANG_V2)
- *namsedim_lateral_erosion*: relative to lateral sediment erosion (only if key_MUSTANG_lateralerosion)
- *namsedim_consolidation*: relative to sediment consolidation
- *namsedim_diffusion*: relative to dissolved diffusion in sediment
- *namsedim_bioturb*: relative to bioturbation in sediment
- *namsedim_morpho*: relative to morphodynamic
- *namtempsed*: relative to temperature estimation in sediment (only if !defined key_noTSdiss_insed)
- *namsedoutput*: parameters used for output results in the file sediment
- *namsedim_debug*: output for debug (only if key_MUSTANG_debug and key_MUSTANG_V2)
- *namflocmod*: parameters using for FLOCMOD module (only if key_MUSTANG_flocmod)
- *namdredging*: relative to dredging feature (see also: *Details on dredging effect*)

Each namelist is described bellow with the description of each parameter.

&namsedim_init:

- **date_start_dyninsed**: starting date for dynamic processes in sediment
- **date_start_morpho**: starting date for morphodynamic processes
- **l_unised**: set to .true. for a uniform bottom initialization, set to .false. to read initialisation from filerepsed file
- **filrepsed**: file path from which the model is initialized for the continuation of a previous run or a non-uniform initialisation used if l_unised = .false.
- **hseduni**: initial uniform sediment thickness (m)
- **cseduni**: initial sediment concentration (kg/m3)

- **l_unised_adjust_hsed**: set to .true. if we want to adjust the sediment thickness in order to be coherent with sediment parameters (calculation of a new hseduni based on cseduni, cvolmax values, and csed_ini of each sediment)
- **csed_mud_ini**: mud concentration into initial sediment (kg/m3) (if = 0. ==> csed_mud_ini = cfreshmud)
- **ksmiuni**: lower grid cell index in the sediment
- **ksmauni**: upper grid cell index in the sediment
- **sini_sed**: initial interstitial water uniform salinity (in the sediment) (PSU)
- **tini_sed**: initial interstitial water uniform temperature (in the sediment) (Celsius degree)
- **poro_mud_ini**: if key_MUSTANG_V2 only, initial porosity of mud fraction
- **l_initsed_vardiss**: set to .true. if initialization of dissolved variables, temperature and salinity in sediment (will be done with concentrations in water at bottom (k=1))

&namsedim_layer:

- **l_dzsminuni**: set to .false. if dzsmin vary with sediment bed composition, else dzsmin = dzsminuni (used if key_MUSTANG_V2 only)
- **dzsminuni**: minimum sediment layer thickness (m) (used if key_MUSTANG_V2 only)
- **dzsmin**: minimum sediment layer thickness (m)
- **dzsmax_bottom**: maximum thickness of bottom layers which result from the fusion when ksdmax is exceeded (m)
- **l_dzsmaxuni**: if set to .true. dzsmax = dzsmaxuni , if set to .false. then linearly computed in MUSTANG_sedinit from dzsmaxuni to dzsmaxuni/100 depending on water depth
- **dzsmaxuni**: uniform maximum thickness for the superficial sediment layer (m), must be >0
- **nlayer_surf_sed**: number of layers below the sediment surface that can not be melted (max thickness = dzsmax)
- **k1HW97**: ref value k1HW97 = 0.07, parameter to compute active layer thickness [Harris and Wiberg, 1997] (key_MUSTANG_V2 only)
- **k2HW97**: ref value k2HW97 = 6.0, parameter to compute active layer thickness [Harris and Wiberg, 1997] (key_MUSTANG_V2 only)
- **fusion_para_activlayer**: criterion cohesiveness for fusion in active layer (key_MUSTANG_V2 only):
  - 0: no fusion,
  - = 1: frmudcr1,
  - > 1: between frmudcr1 & frmudcr2

&namsedim_bottomstress, this part combine *bottom stress* and *roughness* parameters:

- **l_z0seduni**: if true, z0seduni is used; if false z0sed is computed from sediment diameter
- **z0seduni**: uniform bed roughness (m)
- **z0sedmud**: mud (i.e.minimum) bed roughness (m) (used only if l_unised is false)
- **z0sedbedrock**: bed roughness for bedrock (no sediment) (m) (used only if l_unised is false)
- **l_fricwave**: if true the wave related friction factor is computed from wave orbital velocity and period; if false then fricwav namelist value is used (see *wave skin friction*)
- **fricwav**: default value is 0.06, wave related friction factor (used for bottom shear stress computation if l_fricwave is false)
- **l_z0hydro_coupl_init**: if true the evaluation of z0 hydro depends on sediment composition at the beginning of the simulation
- **l_z0hydro_coupl**: if true the evaluation of z0 hydro depends on sediment composition along the run

- **coef_z0_coupl**: if l_z0hydro_coupl is true, parameter to compute z0hydro in the first centimeter z0hydro = coef_z0_coupl * sand diameter
- **z0_hydro_mud**: if l_z0hydro_coupl is true, z0hydro if pure mud (m)
- **z0_hydro_bed**: if l_z0hydro_coupl is true, z0hydro if no sediment (m)

&namsedim_deposition:

- **cfreshmud**: prescribed fresh deposit concentration in kg/m3 (must be around 100 if consolidation or higher (300-500 if no consolidation)
- **csedmin**: concentration of the upper layer under which there is fusion with the underlying sediment cell (in kg/m3)
- **cmudcr**: critical relative concentration of the surface layer above which no mixing is allowed with the underlying sediment (in kg/m3)
- **aref_sand**: reference height above sediment in meter. Used for computing of sand deposit for sand extrapolation on water column and correct sand transport, value by default = 0.02 correspond to Van Rijn experiments. **DO NOT CHANGED IF NOT EXPERT**. (see *Treatment of high settling velocities: SAND variables*)
- **cvolmaxsort**: maximum volumic concentration of sorted sand
- **cvolmaxmel**: maximum volumic concentration of mixed sediments
- **slopefac**: slope effect multiplicative on sliding part of deposit (only if key_MUSTANG_slipdeposit see *sliding fluxes*)

&namsedim_erosion:

- **activlayer**: active layer thickness (m)
- **frmudcr2**: critical mud fraction under which the behaviour is purely sandy
- **coef_frmudcr1**: such that critical mud fraction under which sandy behaviour (frmudcr1=min(coef_frmudcr1*d50 sand,frmudcr2))
- **x1toce_mud**: mud erosion parameter: toce = x1_toce_mud*(relative mud concentration)**x2_toce_mud
- **x2toce_mud**: mud erosion parameter: toce = x1_toce_mud*(relative mud concentration)**x2_toce_mud
- **E0_sand_option**: choice of formulation for E0_sand evaluation:
  - 0: E0_sand = E0_sand_Cst read in this namelist
  - 1: E0_sand evaluated with Van Rijn [1984] formulation
  - 2: E0_sand evaluated with erodimetry (min(0.27,1000*d50-0.01)*toce**n_eros_sand)
  - 3: E0_sand evaluated with Wu and Lin [2014] formulation
- **E0_sand_Cst**: constant erosion flux for sand (used if E0_sand_option= 0)
- **E0_sand_para**: coefficient used to modulate erosion flux for sand (=1 if no correction)
- **n_eros_sand**: parameter for erosion flux for sand (E0_sand*(tenfo/toce-1.)**n_eros_sand). WARNING: choose parameters compatible with E0_sand_option (example: n_eros_sand=1.6 for E0_sand_option=1)
- **E0_mud**: parameters for erosion flux for pure mud
- **E0_mud_para_indep**: parameter to correct E0_mud in case of erosion class by class in non cohesive regime (key_MUSTANG_V2 only)
- **n_eros_mud**: E0_mud*(tenfo/toce-1.)**n_eros_mud
- **ero_option**: choice of erosion formulation for mixing sand-mud. **These formulations are debatable and must be considered carefully by the user. Other laws are possible and could be programmed.**
  - 0: pure mud behavior (for all particles and whatever the mixture)
  - 1: linear interpolation between sand and mud behavior, depend on proportions of the mixture
  - 2: formulation derived from that of J. Vareilles (2013)

  - 3: formulations proposed by Mengual *et al.* [2017] with exponential coefficients depend on proportions of the mixture

- **l_xexp_ero_cst**: set to .true. if xexp_ero estimated from empirical formulation, depending on frmudcr1 (key_MUSTANG_V2 only)
- **xexp_ero**: used only if ero_option=3: adjustment on exponential variation (more brutal when xexp_ero high)
- **tau_cri_option**: ichoice of critical stress formulation
  - 0: Shields
  - 1: Wu and Lin [2014]
- **tau_cri_mud_option_eroindep**: choice of mud critical stress formulation
  - 0: x1toce_mud*cmudr**x2toce_mud
  - 1: toce_meansan if somsan>eps (else->case0)
  - 2: minval(toce_sand*cvsed/cvsed+eps) if >0 (else->case0)
  - 3: min(case 0; toce(isand2)) (key_MUSTANG_V2 only)
- **l_eroindep_noncoh**:
  - set to .true. in order to activate independant erosion for the different sediment classes sands and muds
  - set to .false. to have the mixture mud/sand eroded as in version V1 (key_MUSTANG_V2 only)
- **l_eroindep_mud**:
  - set to .true. if mud erosion independant for sands erosion
  - set to .false. if mud erosion proportionnal to total sand erosion (key_MUSTANG_V2 only)
- **l_peph_suspension**: set to .true. if hindering / exposure processes in critical shear stress estimate for suspension (key_MUSTANG_V2 only)

&namsedim_poro:

- **poro_option**: choice of porosity formulation
  - 1: Wu and Li [2017] (incompatible with consolidation))
  - 2: mix ideal coarse/fine packing
- **poro_min**: minimum porosity below which consolidation is stopped
- **Awooster**: parameter of the formulation of Wooster *et al.* [2008] for estimating porosity associated to the non-cohesive sediment see Cui *et al.* [1996] ; ref value = 0.42
- **Bwooster**: parameter of the formulation of Wooster *et al.* [2008] for estimating porosity associated to the non-cohesive sediment see Cui *et al.* [1996] ; ref value = -0,458
- **Bmax_wu**: maximum portion of the coarse sediment class participating in filling; ref value = 0.65

&namsedim_bedload: (key_MUSTANG_V2 only)

- **l_peph_bedload**: set to .true. if hindering / exposure processes in critical shear stress estimate for bedload
- **l_slope_effect_bedload**: set to .true. if accounting for slope effects in bedload fluxes (Lesser formulation)
- **alphabs**: coefficient for slope effects (default coefficients Lesser *et al.* [2004], alphabs = 1.)
- **alphabn**: coefficient for slope effects (default coefficients Lesser *et al.* [2004], default alphabn is 1.5 but can be higher, until 5-10 (Gerald Herling experience))
- **hmin_bedload**: no bedload in u/v directions if h0+ssh <= hmin_bedload in neighbouring cells
- **l_fsusp**: limitation erosion fluxes of non-coh sediment in case of simultaneous bedload transport, according to Wu & Lin formulations. Set to .true. if erosion flux is fitted to total transport should be set to .false. if E0_sand_option=3 (Wu & Lin)

&namsedim_lateral_erosion: (see *Lateral erosion*)

- **coef_erolat**: slope effect multiplicative factor
- **coef_tauskin_lat**: parameter to evaluate the lateral stress as a function of the average tangential velocity on the vertical
- **l_erolat_wet_cell**: set to .true in order to take into account wet cells lateral erosion
- **htncrit_eros**: critical water height so as to prevent erosion under a given threshold (the threshold value is different for flooding or ebbing, cf. Hibma's PhD, 2004, page 78)

&namsedim_consolidation:

- **l_consolid**: set to .true. if sediment consolidation is accounted for
- **dt_consolid**: time step for consolidation processes in sediment (will use in fact the min between dt_consolid, dt_diffused if l_diffused, dt_bioturb if l_bioturb)
- **subdt_consol**: sub time step for consolidation processes in sediment (< or = min(dt_consolid, ..))(will use in fact the min between subdt_consolid, subdt_bioturb if l_bioturb)
- **csegreg**: DO NOT CHANGE VALUE if not expert, default 250.0
- **csandseg**: DO NOT CHANGE VALUE if not expert, default 1250.0
- **xperm1**: parameter to compute permeability permeability=xperm1*d50*d50*voidratio**xperm2
- **xperm2**: parameter to compute permeability permeability=xperm1*d50*d50*voidratio**xperm2
- **xsigma1**: parameter used in Merckelbach and Kranenburg [2004] formulation. DO NOT CHANGE VALUE if not expert, default 6.0e+05
- **xsigma2**: real, parameter used in Merckelbach and Kranenburg [2004] formulation. DO NOT CHANGE VALUE if not expert, default 6

&namsedim_diffusion:

- **l_diffused**: set to .true. if taking into account dissolved diffusion in sediment and at the water/sediment interface
- **dt_diffused**: time step for diffusion processes in sediment (will use in fact the min between dt_diffused, dt_consolid if l_consolid, dt_bioturb if l_bioturb)
- **choice_flxdiss_diffsed**: choice for expression of dissolved fluxes at sediment-water interface
  - 1: Fick law: gradient between Cv_wat at dz(1)/2
  - 2: Fick law: gradient between Cv_wat at distance epdifi
- **xdifs1**: diffusion coefficients within the sediment
- **xdifsi1**: diffusion coefficients at the water sediment interface
- **epdifi**: diffusion thickness in the water at the sediment-water interface
- **fexcs**: factor of eccentricity of concentrations in vertical fluxes evaluation (.5 a 1) (numerical scheme for dissolved diffusion/advection(by consol) in sediment)

&namsedim_bioturb:

- **l_bioturb**: set to .true. if taking into account particulate bioturbation (diffusive mixing) in sediment
- **l_biodiffs**: set to .true. if taking into account dissolved bioturbation diffusion in sediment
- **dt_bioturb**: time step for bioturbation processes in sediment (will use in fact the min between dt_bioturb, dt_consolid if l_consolid, dt_diffused if l_diffused)
- **subdt_bioturb**: sub time step for bioturbation processes in sediment (< or = min(dt_bioturb, ..)) (will use in fact the min between subdt_bioturb, subdt_consolid if l_consolid)
- **xbioturbmax_part**: max particular bioturbation coefficient by bioturbation Db (in surface)
- **xbioturbk_part**: coef (slope) for part. bioturbation coefficient between max Db at sediment surface and 0 at bottom

- **dbiotu0_part**: max depth beneath the sediment surface below which there is no bioturbation
- **dbiotum_part**: sediment thickness where the part-bioturbation coefficient Db is constant (max)
- **xbioturbmax_diss**: max diffusion coeffient by biodiffusion Db (in surface)
- **xbioturbk_diss**: coef (slope) for biodiffusion coefficient between max Db at sediment surface and 0 at bottom
- **dbiotu0_diss**: max depth beneath the sediment surface below which there is no bioturbation
- **dbiotum_diss**: sediment thickness where the diffsolved-bioturbation coefficient Db is constant (max)
- **frmud_db_min**: mud fraction limit (min) below which there is no Biodiffusion
- **frmud_db_max**: mud fraction limit (max)above which the biodiffusion coefficient Db is maximum (muddy sediment)

&namsedim_morpho:

- **l_morphocoupl**: set to .true if coupling module morphodynamic, see *Morphodynamic*
- **MF**: morphological factor: multiplication factor for morphological evolutions, equivalent to a "time acceleration" (morphological evolutions over a MF*T duration are assumed to be equal to MF * the morphological evolutions over T).
- **dt_morpho**: time step for morphodynamic (s)
- **l_MF_dhsed**:
  - set to .true. if morphodynamic applied with sediment height variation amplification
  - set to .false. if morphodynamic is applied with erosion/deposit fluxes amplification

&namtempsed: (only if !defined key_noTSdiss_insed)

- **mu_tempsed1**: parameters used to estimate thermic diffusitiy function of mud fraction
- **mu_tempsed2**: parameters used to estimate thermic diffusitiy function of mud fraction
- **mu_tempsed3**: parameters used to estimate thermic diffusitiy function of mud fraction
- **epsedmin_tempsed**: sediment thickness limits for estimation heat loss at bottom, if hsed < epsedmin_tempsed: heat loss at sediment bottom = heat flux a sediment surface
- **epsedmax_tempsed**: sediment thickness limits for estimation heat loss at bottom, if hsed > epsedmax_tempsed: heat loss at sediment bottom = 0.

&namsedoutput:

- **l_outsed_nb_lay_sed**: boolean, set to true to output the number of sediment layers (upper sediment layer index named ksma)
- **l_outsed_hsed**: boolean, set to true to output sediment height (m)
- **l_outsed_dzs**: boolean, set to true to output sediment layer thickness (m)
- **l_outsed_temp_sed**: boolean, set to true to output temperature in sediment layers
- **l_outsed_salt_sed**: boolean, set to true to output salinity in sediment layers
- **l_outsed_cv_sed**: boolean, set to true to output concentration of sediment class (kg/m3). Note that if in substance namelist l_out_subs_n boolean of the class is set to false, the corresponding variable is not outputed
- **l_outsed_ws**: boolean, set to true to output settling welocity of sediment class (m/s). Note that it is not outputed for gravels, sands andif l_out_subs_n boolean of the class is set to false in substance namelist
- **l_outsed_z0sed**: boolean, set to true to output skin roughness length (m)
- **l_outsed_z0hydro**: boolean, set to true to output hydrodynamic roughness length (m)
- **l_outsed_tauskin**: boolean, set to true to output total bottom shear stress (N/m2)

- **l_outsed_tauskin_cw**: boolean, set to true to output current bottom shear stress and wave bottom shear stress (N/m2)
- **l_outsed_poro**: boolean, set to true to output porosity of sediment layers
- **l_outsed_toce**: boolean, set to true to output critical shear stress for each sediment class. Note that if in substance namelist l_out_subs_n boolean of the class is set to false, the corresponding variable is not outputed
- **l_outsed_flx_s2w_w2s**: boolean, set to true to output erosion fluxes and deposition fluxes (kg/m-2)
- **l_outsed_frmudsup**: boolean, set to true to output mud fraction in the interface sediment/water layer
- **l_outsed_dzs_ksmax**: boolean, set to true to output the interface sediment/water layer thickness (m)
- **l_outsed_pephm_fcor**: boolean, set to true to output the hindering exposure factor on critical shear stress for each class
- **l_outsed_theoric_active_layer**: boolean, set to true to output theoretical active layer thickness according Harris and Wiberg 1997
- **l_outsed_ero_details**: boolean, set to true to output time elapsed in the cohesive/non-cohesive erosion regime and part of erosion iterations in each regime
- **l_outsed_bedload**: boolean, set to true to output beadload components on x and y axis (kg/m/s) and divergence of bedload fluxes
- **l_outsed_fsusp**: boolean, set to true to output the fraction of transport in suspension
- **l_outsed_consolidation**: boolean, set to true to output consolidation variables
- **choice_nivsed_out**: choice of saving output
  - 1: all the layers (ksdmin to ksdmax) are saved (k=1: bottom layer) (nk_nivsed_out, ep_nivsed_out, epmax_nivsed_out are not used)
  - 2: only save the nk_nivsed_out surficial layers (k=1: layer most bottom)
  - 3: each layers from sediment surface are saved till the thickness epmax_nivsed_out (which must be non zero and > dzsmax (k=1: bottom layer)) This option is not recommended if l_dzsmaxuni=.False.
  - 4: 1 to 5 layers of constant thickness are saved; thickness are selected with ep_nivsed_out and concentrations are interpolated to describe the sediment thickness (k=1: surface layer) the thickness of the bottom layer (nk_nivsed_out+1) will vary depending on the total thickness of sediment in the cell
- **nk_nivsed_out**: number of saved sediment layers
  - unused if choice_nivsed_out = 1
  - <ksdmax if choice_nivsed_out = 2,
  - unused if choice_nivsed_out = 3
  - <6 if choice_nivsed_out = 4,
- **ep_nivsed_out()**: 5 values of sediment layer thickness (mm), beginning with surface layer (used if choice_nivsed_out=4)
- **epmax_nivsed_out**: maximum thickness (mm) for output each layers of sediment (used if choice_nivsed_out=3). Below the layer which bottom level exceed this thickness, an addition layer is an integrative layer till bottom

&namsedim_debug:

- **l_debug_effdep**: set to .true. if print some informations for debugging MUSTANG deposition
- **l_debug_erosion**: set to .true. if print informations for debugging in erosion routines
- **date_start_debug**: string, starting date for write debugging informations
- **lon_debug**: define mesh location where we print these informations
- **lat_debug**: define mesh location where we print these informations

- **i_MUSTANG_debug**: indexes of the mesh where we print these informations (only if lon_debug and lat_debug = 0.)
- **j_MUSTANG_debug**: indexes of the mesh where we print these informations (only if lon_debug and lat_debug = 0.)

&namflocmod:

- **l_ADS**: set to .true. if aggregation by differential settling
- **l_ASH**: set to .true. if aggregation by shear
- **l_COLLFRAG**: set to .true. if fragmentation by collision
- **f_dp0**: primary particle size (default 4.e-6 m)
- **f_nf**: fractal dimension (default 2.0, usual range from 1.6 to 2.8)
- **f_nb_frag**: nb of fragments of equal size by shear fragmentation (default 2.0 as binary fragmentation)
- **f_alpha**: flocculation efficiency parameter (default 0.15)
- **f_beta**: floc break up parameter (default 0.1)
- **f_ater**: ternary fragmentation factor: proportion of flocs fragmented as half the size of the initial binary fragments (0.0 if full binary fragmentation, 0.5 if ternary fragmentation)
- **f_ero_frac**: floc erosion (% of floc mass eroded) (default 0.05)
- **f_ero_nbfrag**: nb of fragments produced by erosion (default 2.0)
- **f_ero_iv**: fragment class (mud variable index corresponding to the eroded particle size - typically 1)
- **f_mneg_param**: negative mass after flocculation/fragmentation allowed before redistribution (default 0.001 g/l)
- **f_dmin_frag**: minimum diameter for fragmentation (default 10e-6 microns)
- **f_cfcst**: fraction of mass lost by flocs if fragmentation by collision .. (default: =3._rsh/16._rsh)
- **f_fp**: relative depth of inter particle penetration (default =0.1) [McAnally, 1999]
- **f_fy**: floc yield strength (default= 1.0e-10) [Winterwerp *et al.*, 2002]
- **f_collfragparam**: fraction of shear aggregation leading to fragmentation by collision (default 0.0, must be less than 1.0)

&namdredging: Namelist relative to dredging effect (see also: *Details on dredging effect*)

- **dredging_location_file**: input netcdf file path containing :
  - dredging areas location and name
  - dumping area location and name

  Note that in case of overlapping dredging areas, the deepest one is chosen during the simulation
- **dredging_settings_file**: parameters input file path containing association between dredging area and dumping area and level limit for dredging
- **dredging_out_file**: output netcdf file path containing for every sediment class and every dredging area the cummulation mass dredged
- **dredging_dumping_layer**: water layer where the mud and sand are dumped (gravel are directly in deposit). Must be an integer between 1 and the number of layer in water.
- **dredging_dt**: dredging time step (in seconds)
- **dredging_dt_out**: dredging output time step (in seconds, must be greater than dredging_dt)

1.13.2.2.2.5 Input file: Initialization of the sediment cover

If the initialization of sediment bed is not uniform (spatially and vertically), the sediment cover is read from a netcdf file (see *Initialization* for more details).

This netcdf file path is given in *MUSTANG namelist &namsedim_init* as follow:

```
l_repsed=.true.
filrepsed='./repsed.nc'
```

The netcdf file needs to have the concentration values under the names *NAME_sed*, with NAME corresponding to the names defined in the *SUBSTANCE namelist*. 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 a CROCO RESTART file can be used here.

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) ;
```

If the dredging feature is used, this file must contains another variable calle dredg_hsed_init containing the initial sediment height to be able to retrieve bed rock position even in morphostatic simulation.

In this case, the header of a sediment cover file is:

```
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) ;
     double dredg_hsed_init(time, nj, ni) ;
```

1.13.2.2.2.6 Input file: Wave

If cpp keys **WAVE_OFFLINE** and **MUSTANG** are activated, a netcdf file must be given in CROCO input file **croco.in** as follow:

```
wave_offline:   filename
      wave.nc
```

Significant wave height, wave period, wave direction and bottom orbital velocity are read in this netcdf file. 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.

The netcdf file should have a temporal window at leats covering the simulation period. Temporal interpolation are made between each time step in the file. **If your configuration contains wetting-drying effect, be careful with values on land, do not put negative values**. We advice you to put 0. for hs, dir and ubr and 10 for t01 (to avoid division by zero).

Header of an example wave file:

```
dimensions:
    wwv_time = UNLIMITED ; // (25 currently)
    eta_rho = 132 ;
    xi_rho = 182 ;
variables:
    double dir(wwv_time, eta_rho, xi_rho) ;
        dir:_FillValue = 0s ;
    double hs(wwv_time, eta_rho, xi_rho) ;
        hs:_FillValue = 0s ;
    double t01(wwv_time, eta_rho, xi_rho) ;
        t01:_FillValue = 10s ;
    double ubr(wwv_time, eta_rho, xi_rho) ;
        ubr:_FillValue = 0s ;
    double uubr(wwv_time, eta_rho, xi_rho) ;
        uubr:_FillValue = 0s ;
    double vubr(wwv_time, eta_rho, xi_rho) ;
        vubr:_FillValue = 0s ;
    double wwv_time(wwv_time) ;
```

1.13.2.2.2.7 Input file: Solid discharge in rivers

If cpp keys **PSOURCE_NCFILE** and **PSOURCE_NCFILE_TS** are activated, a netcdf file must be given in CROCO input file **croco.in** as follow:

```
psource_ncfile:   Nsrc  Isrc  Jsrc  Dsrc qbardir  Lsrc  Tsrc   runoff file name
        psource.nc
    2
                     167  56  0 -1   30*T   20.0  15.0   Loire
                     91  99  0 -1   30*T   20.0  15.0   Vilaine_arzal
```

This file is in netcdf format. It reads the concentration values (T,S and sediment) in get_psource_ts.F.

The name of sediment variable must match the name chosen in *SUBSTANCE namelist*, in the example below: MUD1.

Header of an example source file:

```
dimensions:
    qbar_time = 7676 ;
    n_qbar = 6 ;
    runoffname_StrLen = 30 ;
```

(continues on next page)

(continued from previous page)

```
    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" ;
        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) ;
```

1.13.2.2.2.8 Input file: submassbalance definition of borders and budget areas

The name of this file is set in *nmlsubmassbalance*.

This feature permits to compute mass fluxes and/or budget during the simulation on areas (closed domain) or passing though boundaries (open domain). (see also:

Details on submassbalance computation)

Boundaries are defined by several contiguous segments S-N and/or W-E with any number of segments and any direction (S-N or N-S, WE or E-W).

An example of submassbalance input file is :

```
*****
1
test
F
2
10 100 45 N
100 45 60 E
*****
2
test2
T
4
50 60 50 S
```

(continues on next page)

(continued from previous page)

```
60 50 80 E
60 50 80 N
50 80 50 W
*****
```

Each segment is separate by a line (here containing `*****`).

Then the file contains :

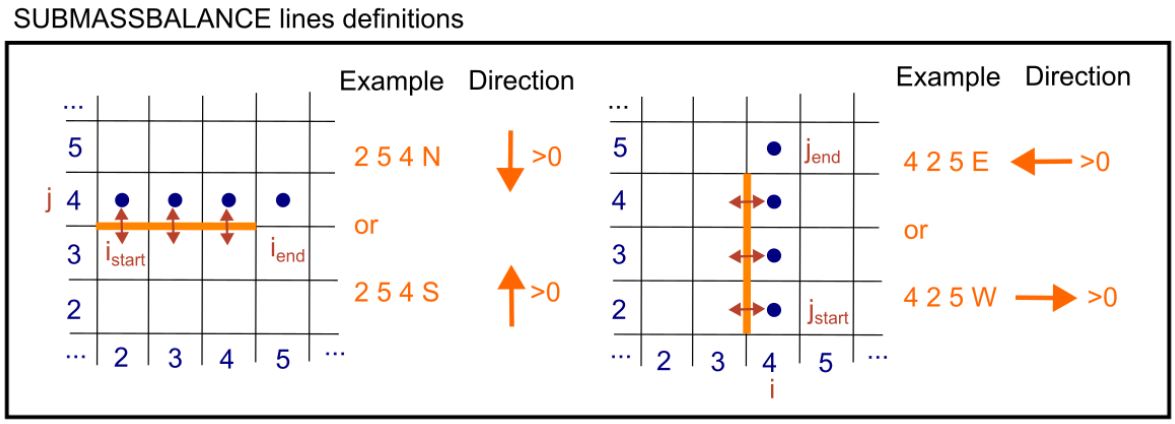
- the number of the border
- the name of the border
- a boolean set to True if the border is a closed polygon
- the number of segment of the border
- the list of segment border (number of lines correspond to the number of segment of the border)

```
*****
1                   ! number of the border
test                ! name of the border
F                   ! boolean set to true for closed domain
2                   ! number of segment of the border
10 100 45 N         ! segment description
100 45 60 E         ! segment description
*****
...
```

Each segment is characterized by 3 indices and a letter N, S, W or E to determine whether this border is a north, south, west or east segment of the sub-domain.

For north and south segments, the 3 indices are i_start i_end j.

For west and east segments, the 3 indices are i j_start j_end.

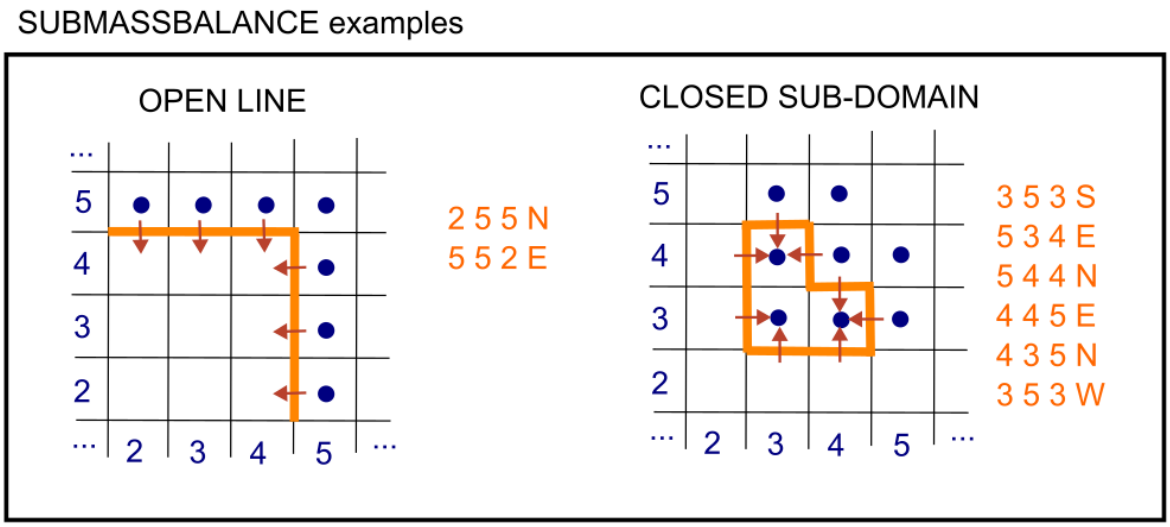


Open boundaries are assigned FIRST and closed boundaries next.

For an open boundary, E,W,S,N give the direction of flow.

For a closed sub-domain: "East" segments are borders placed at the east of the closed area. "South" segments are borders placed at the south of the area etc… Budgets are computed only inside sub-domains surrounded by closed boundaries.

You can verify your domains masks in the output file see *mustang_submassbalance_outputfile*.



1.13.2.2.2.9 Output options

Outputs for sediment variables are written directly by CROCO or XIOS.

Two options are available :

- Classic Netcdf outputs
- XIOS output

1.13.2.2.2.10 Classic Netcdf CROCO output

Classic Netcdf CROCO output can be used to have MUSTANG variables.

The selection is done by the *namsedoutput* namelist.

Note that if l_out_subs_n boolean of the class is set to false in *SUBSTANCE namelist* the corresponding variables are not output.

1.13.2.2.2.11 XIOS Outputs

XIOS (#define XIOS) can be configured to output the same variables as the classic netcdf output.

Here is an example of a .xml field file for a case with four sediments classes named: GRAV, SAND, MUD and MUD2 for the preselected output booleans in *namsedoutput* namelist. The id of each field has to be coherent with the given name of sediment.

```
<field id="GRAV" long_name="Sediment concentration in the water column" unit="g/l"␣
↪grid_ref="rho_3D" />
<field id="SAND" long_name="Sediment concentration in the water column" unit="g/l"␣
↪grid_ref="rho_3D"/>
<field id="MUD" long_name="Sediment concentration in the water column" unit="g/l"␣
↪grid_ref="rho_3D" />
<field id="MUD2" long_name="Sediment concentration in the water column" unit="g/l"␣
↪grid_ref="rho_3D" />
<field id="GRAV_sed" long_name="Sediment concentration in the seabed" unit="g/l" grid_
↪ref="b_3D"/>
<field id="SAND_sed" long_name="Sediment concentration in the seabed" unit="g/l" grid_
↪ref="b_3D"/>
<field id="MUD_sed" long_name="Sediment concentration in the seabed" unit="g/l" grid_
↪ref="b_3D"/>
<field id="MUD2_sed" long_name="Sediment concentration in the seabed" unit="g/l" grid_
```

(continues on next page)

(continued from previous page)

```
↪ref="b_3D"/>
<field id="temp_sed" long_name="Temperature in the seabed" unit="Celsius" grid_ref="b_
↪3D"/>
<field id="salt_sed" long_name="Salinity in the seabed" unit="PSU" grid_ref="b_3D"/>
<field id="dzs" long_name="Sediment layer thickness" unit="m" grid_ref="b_3D"/>
<field id="z0sed" long_name="Skin roughness length" unit="m"/>
<field id="z0hydro" long_name="Hydrodynamic roughness length" unit="m"/>
<field id="tauskin" long_name="Total bottom shear stress" unit="N meter-2"/>
<field id="tauskin_c" long_name="Current-induced bottom shear stress" unit="N meter-2
↪"/>
<field id="tauskin_w" long_name="Wave-induced bottom shear stress" unit="N meter-2"/>
<field id="hsed" long_name="Bottom thickness" unit="m"/>
<field id="ksma" long_name="Upper sediment layer index" unit="no unit"/>
```

Other fields are available depending on *namsedoutput* namelist and activation of cppkeys #key_MUSTANG_V2, #key_MUSTANG_bedload :

- settling velocities for sediment of type MUD if l_outsed_ws is set to True :

```
<field id="MUD_ws" long_name="Settling velocity" unit="m/s" grid_ref="rho_3D" />
```

- critical shear stress if l_outsed_toce is set to True :

```
<field id="MUD_toce" long_name="critical shear stress" unit="N/m2"/>
```

- erosion and deposition fluxes if l_outsed_flx_s2w_w2s is set to True :

```
<field id="MUD_flx_s2w" long_name="erosion flux" unit="kg.m-2"/>
<field id="MUD_flx_w2s" long_name="deposition flux" unit="kg.m-2"/>
<field id="flx_s2w_noncoh" long_name="erosion flux of non-cohesive sediments␣
↪(sum: isand1 to isand2)" unit="kg.m-2"/>
<field id="flx_w2s_noncoh" long_name="deposition flux of non-cohesive sediments␣
↪(sum: isand1 to isand2)" unit="kg.m-2"/>
<field id="flx_s2w_coh" long_name="erosion flux of cohesive sediments (sum:␣
↪imud1 to imud2)" unit="kg.m-2"/>
<field id="flx_w2s_coh" long_name="deposition flux of cohesive sediments (sum:␣
↪imud1 to imud2)" unit="kg.m-2"/>
```

- mud fraction in the upper sediment layer if l_outsed_frmudsup is set to True :

```
<field id="frmudsup" long_name="mud fraction in the ksmax layer" unit="no unit"/>
```

- layer thickness in the upper sediment layer if l_outsed_dzs_ksmax is set to True :

```
<field id="dzs_ksmax" long_name="layer thickness at sediment surface" unit="m"/>
```

- if #key_MUSTANG_V2:
  - if l_outsed_pephm_fcor is set to True :

    ```
    <field id="SAND_pephm_fcor" long_name="Hindering exposure factor on toce"␣
    ↪unit="no unit"/>
    ```

  - if l_outsed_theoric_active_layer is set to True :

    ```
    <field id="theoric_active_layer" long_name="Theoretical active layer␣
    ↪thickness Harris and Wiberg 1997" unit="m"/>
    ```

  - if l_outsed_ero_details is set to True :

```
<field id="tero_noncoh" long_name="time elapsed in the non-cohesive erosion␣
↪regime" unit="h"/>
<field id="tero_coh" long_name="time elapsed in the cohesive erosion regime"␣
↪unit="h"/>
<field id="pct_iter_noncoh" long_name="part of erosion iterations in the non-
↪cohesive regime" unit="percent"/>
<field id="pct_iter_coh" long_name="part of erosion iterations in the␣
↪cohesive regime" unit="percent"/>
<field id="niter_ero" long_name="number of iterations in sed_erosion during␣
↪time step" unit="no unit"/>
```

- if #key_MUSTANG_V2 and #key_MUSTANG_bedload and l_outsed_bedload :

```
<field id="SAND_flx_bx" long_name="Bedload flux along x-axis" unit="kg/m/s"/>
<field id="SAND_flx_by" long_name="Bedload flux along y-axis" unit="kg/m/s"/>
<field id="SAND_bil_bedload" long_name="Divergence of bedload flux" unit="kg.m-2
↪"/>
<field id="SAND_fsusp" long_name="fraction of transport in suspension" unit="no␣
↪unit"/>
<field id="flx_bx_int" long_name="Total bedload flux along x-axis" unit="kg/m/s"/
↪>
<field id="flx_by_int" long_name="Total bedload flux along y-axis" unit="kg/m/s"/
↪>
<field id="bil_bedload_int" long_name="Divergence of total bedload flux" unit=
↪"kg.m-2"/>
```

- if l_outsed_consolidation and l_dyn_insed :

```
<field id="loadograv" long_name="excess of interstitial water pressure in the␣
↪middle of the layer" unit="-" grid_ref="b_3D"/>
<field id="sigmadjge" long_name="sigma unseparated (without the share of water)"␣
↪unit="-" grid_ref="b_3D"/>
<field id="sigmapsg" long_name="effective stress (transmitted from grain to␣
↪grain)" unit="-"grid_ref="b_3D"/>
<field id="stateconsol" long_name="state of consolidation indicator" unit="-"␣
↪grid_ref="b_3D"/>
<field id="permeab" long_name="permeability" unit="-" grid_ref="b_3D"/>
<field id="hinder" long_name="shackling sand / gravel between 0 and 1'" unit="-"␣
↪grid_ref="b_3D"/>
<field id="sed_rate" long_name="advection speed of mud particles" unit="-" grid_
↪ref="b_3D"/>
<field id="dtsdzs" long_name="dtsdzs" unit="-" grid_ref="b_3D"/>
```

1.13.2.2.2.12 Available CPP keys

The compulsory CPP keys to use MUSTANG in CROCO:

- **MUSTANG**: activate module MUSTANG
- **SUBSTANCE**: activate module SUBSTANCE
- **SALINITY**: needed for SUBSTANCE
- **TEMPERATURE**: needed for SUBSTANCE
- **USE_CALENDAR**: needed for MUSTANG, issue with MUSTANG coupling timing otherwise
- **key_noTSdiss_insed**: temperature, salinity and others dissolved variables are not computed in sediment. They have constant values and no fluxes of dissolved variables between water and sediment are computed.

- **key_nofluxwat_IWS**: no exchange water fluxes between water and sediment (recommended if key_noTSdiss_insed).

The optional CPP keys, to choose processes or version:

- **key_MUSTANG_V2**: to use MUSTANG in V2, without this key, the version V1 is used
- **MORPHODYN**: to activate morphodynamic (**l_morphocoupl** must also be set to .true, see *Morphodynamic*)
- **SED_DENS**: to activate the effect of suspended sediment on the density, see *Effect on density*
- **key_sand2D**: to treat SAND in suspension as 2D variable, see *Treatment of high settling velocities: SAND variables*
- **MUSTANG_CORFLUX**: to correct SAND horizontal fluxes, see *Treatment of high settling velocities: SAND variables*
- **WAVE_OFFLINE**: to use wave in bed shear stress computation, see *wave skin friction*
- **key_MUSTANG_flocmod**: to activate module floculation (*FLOCMOD*)
- **SUBSTANCE_SUBMASSBALANCE**: to activate submassbalance computing (*SUBMASSBALANCE*)
- **key_tauskin_c_upwind**: Upwind scheme for current-induced bottom shear stress, see *Shear stress*
- **key_tauskin_c_center**: Compute bottom shear stress with u^* directly at (rho) location (center of the cell), see *Shear stress*
- **key_tauskin_c_ubar**: Shear stress computed form depth-averaged velocity, see *Shear stress*
- **PSOURCE_NCFILE** and **PSOURCE_NCFILE_TS**: to read solid discharge in river from netcdf files
- **key_MUSTANG_slipdeposit**: see *Sliding fluxes*
- **key_MUSTANG_lateralerosion**: see *Lateral erosion*
- **key_MUSTANG_splitlayersurf** : cutting of surface sediment layers to have regular and precise discretization at surface
- **key_MUSTANG_bedload**: only with key_MUSTANG_V2, bedload processus included
- **key_MUSTANG_debug**: only with key_MUSTANG_V2, choice print information during erosion or deposit process. **Does not work in MPI** print a lot of variable during run for debugging choice of coordinates of the point to be checked

1.13.2.2.3 MUSTANG technical documentation

1.13.2.2.3.1 Overall architecture of the module

To compute sediment behavior, MUSTANG execute the steps :

- *Initialization* of sediment variables from input files
- *Temporal loop* with a sequence of forcing update, erosion phase, exchange between sediment and water, deposition phase, morphodynamic update and call to output feature

MUSTANG module is integrated into CROCO code. It is composed of elements added to the existing code in a MUSTANG directory. The interface between the sedimentary module and the hydrodynamic code is done via :

- plug_MUSTANG_CROCO.F90 which makes the 4 main subroutines available to the rest of the code :
  - mustang_init_main: initialization
  - mustang_update_main: update of forcing and erosion phase
  - mustang_deposition_main: deposition phase
  - mustang_morpho_main: to apply morphodynamic
- modification of CROCO files :

- Initialization in main.F
- Main calls in step.F
- Treatment of settling and sediment/water exchanges in step3d_t.F
- Input features in read_inp.F
- Output features in: XIOS/send_xios_diags.F, OCEAN/wrt_his.F, OCEAN/wrt_rst.F, OCEAN/wrt_sta.F, OCEAN/nc_sta.h, OCEAN/ncscrum.h, OCEAN/nf_read.h, OCEAN/fillvalue.F, step_sta.F, sta.h
- Wave forcing in OCEAN directory in files: forces.h, get_vbc.F, get_wwave.F, init_arrays.F, init_scalars.F
- Test cases in OCEAN/ana_grid.F and OCEAN/ana_initial.F
- Compilation and dimension in OCEAN directory in files: Makefile, jobcomp, param.h, cppdefs.h, cppdefs_dev.h

Roughly, all sediment calculations are done in the MUSTANG fortran files, except for the advection part which is intimately linked to step3d_t.F. Modifications in CROCO files are mostly for I/O features or test cases analytical forcing.

1.13.2.2.3.2 Initialization

The initialization must be done for water column variables and sediment bed variables.

In the water column, the initialization is done from initial file if provided (see hydrodynamic code). If there is no initial file, the water concentrations are initialized from uniform value in *SUBSTANCE namelist*.

To initialize the sediment cover, two options are available :

- Uniform sediment cover

  In paraMUSTANG*.txt:

  ```
  l_unised = .true.  ! boolean set to true for a uniform bottom initialization
  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 (format of a RESTART file, see *input file for sediment cover*)

Warning

- The restarts are not *perfect* restarts. To do a perfect restart, you will need to save the erosion and deposition fluxes in a restart file, as was done in MARS3D (cf. subroutine sed_outsaverestart in sed_MUSTANG_CROCO.F90). This has not been implemented yet.

1.13.2.2.3.3 Temporal loop

In the temporal loop of CROCO model, the main calls to MUSTANG routines are in step3D_t_thread.

```
call prestep3D_thread()
call step2d_thread()
call step3D_uv_thread()
```

(continues on next page)

(continued from previous page)

```
call step3D_t_thread()
----> call mustang_update_main()
----> call step3d_t
----------------> #  include "t3dmix_tridiagonal_settling.h"
----> CALL mustang_deposition_main
----> CALL mustang_morpho_main
```

The erosion and deposition phases are sequenced at each time step:

- erosive phase is treated before the call to step3d_t (treatment of vertical advection). This phase contains:
  - calculation of the *roughness* and *bottom shear stress*,
  - calculation of the *erosion fluxes* for each class
  - evolution of the sedimentary bed from erosion: *erosion layer managment*
  - calculation of the tendency to deposition *deposit fluxes*.
- exchanges from the water column point of view are processed in step3d_t via "t3dmix_tridiagonal_settling.h" and compute also the effective deposit fluxes
- deposit step is processed after the call to step3d_t. This phase includes
  - calculation of the deposit for each class,
  - evolution of the sedimentary bed: *deposition layer managment*
- *morphodynamic* coupling.

The calculations are carried out cell by cell by considering most of the sedimentary variables at the center of the cell. The majority of the calculations are therefore carried out in 1DV. Certain calculations must however be carried out taking into account the adjacent meshes:

- calculate skin stress has current are computed on the mesh edges
- calculate the slope of the bottom and the coefficients necessary to take into account its effect on transport by bedload
- calculate the correction of horizontal sand fluxes
- calculate the fluxes entering a cell induced by bedload and suspension transport

TODO: add a scheme to explain the two main phases: erosion/deposit

1.13.2.2.3.4 Roughness length

Mustang use **grain roughness length** to evaluate moving conditions of particles.

Mustang can also compute a **form roughness length** to account of ripple effect on flow and transmit it to the hydrodynamic code (here CROCO)

The **grain roughness length** could be :

- constant in time and uniform in space with **l_z0seduni** = .TRUE., in this case, the skin roughness length is equal to z0seduni namelist value (see *namelist namsedim_bottomstress*)
- variable in time and space with **l_z0seduni** = .FALSE., in this case, the skin roughness length is computed at each time step from sediment bed composition in each cell (i,j) :
  - if bathymetry is not defined in the cell: z0sed = z0sedmud (see *z0sedmud in namsedim_bottomstress*) (to avoid division by zero in skinstress evaluation when neighbour cells are used)
  - if bathymetry is defined in the cell:
    - if sediment is not present, then z0sed = z0sedbedrock (see *z0sedbedrock in namsedim_bottomstress*)
    - if sediment is present,

- if there is only mud in the superficial layer z0sed = z0seduni
- if there is sand or gravel, Nikuradse formulation is used (z0sed = diam/12) with diam corresponding to the ponderate sum of gravel and sand diameter

  $diam = \sum_{n=igrav1}^{isand2} cv_{sed}(n, kmax)/c_{sedtot}(n, kmax) * diameter(n)$

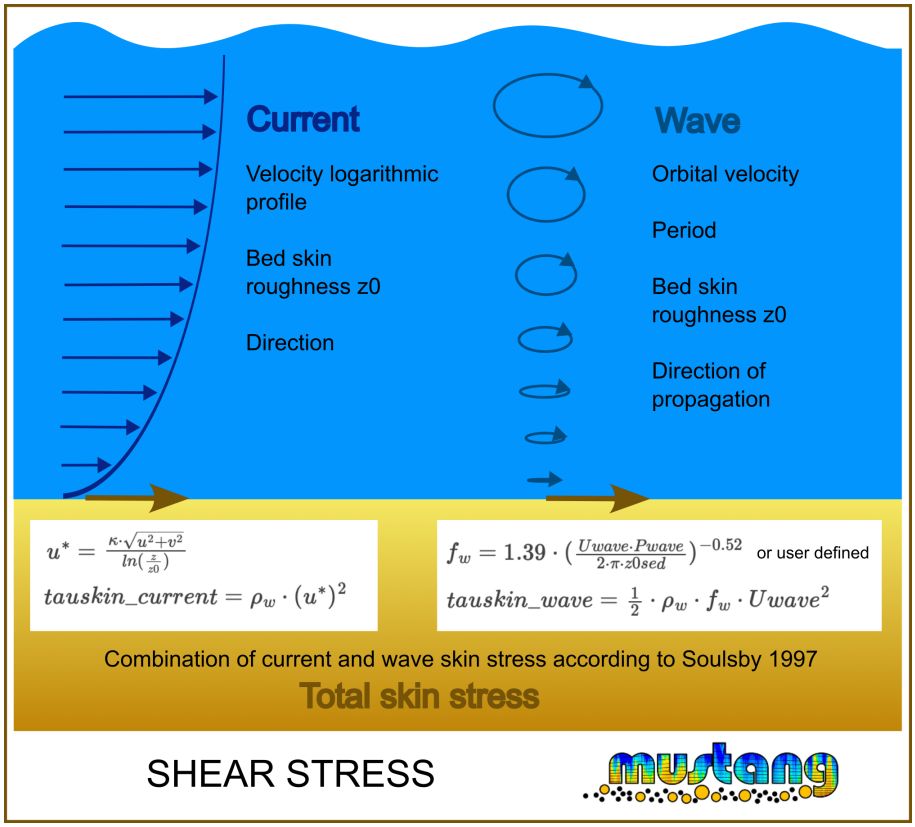
The **form roughness length** is computed and sent to CROCO only if l_z0hydro_coupl (at each time step) or l_z0hydro_coupl_init (just at initialization) :

- if there is no sediment, z0hydro = z0_hydro_bed (see *z0_hydro_bed in namsedim_bottomstress*)
- if there is sediment :
  - if there is more than 30% of mud in the first centimeter of sediment, z0hydro = z0_hydro_mud (see *z0_hydro_mud in namsedim_bottomstress*)
  - else the ponderate diameter of sand is used and z0hydro = coef_z0_coupl * ponderate diameter of sand (see *coef_z0_coupl in namsedim_bottomstress*). Note that gravel are not taking into account here

1.13.2.2.3.5 Shear stress

The shear stress skin friction is computed following steps :

- compute current skin friction
- if cpp keys WAVE_OFFLINE is activated, compute wave skin friction
- if cpp keys WAVE_OFFLINE is activated, compute combination between current and wave skin friction



Current skin friction

Current skin friction is computed from the friction velocity using a logarithmic profile. The friction velocity u^* is computed from roughness length z0 and current component (bottom (u,v) or barotropic (ubar,vbar) using a logarithmic profile :

$u^* = \frac{\kappa \cdot \sqrt{u^2+v^2}}{ln(\frac{z}{z0})}$ with z, height of the bottom cell.

or

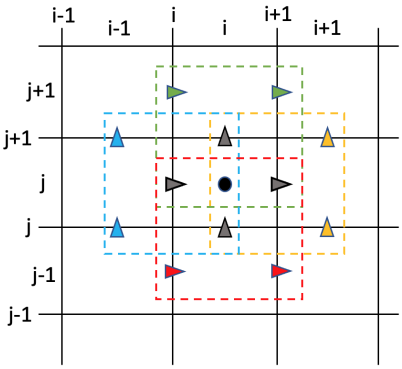
$u^* = \frac{\kappa \cdot \sqrt{ubar^2 + vbar^2}}{ln(\frac{z}{e \cdot z0})}$ with z, the water height.

Current skin friction is compute using : $tauskin\_current = \rho_w \cdot (u^*)^2$

The following option are available via cpp keys :

- **default**: compute u^* components at (u,v) locations first and then at the center of the cell. This option use 12 points of current components (see figure below).
- **key_tauskin_c_center**: compute u^* directly at (rho) location (center of the cell) using immediate u,v components. This option use 4 points of current components (see figure below).
- **key_tauskin_c_ubar**: compute u^* using ubar,vbar value instead of bottom layer u,v values.
- **key_tauskin_c_upwind**: depending on current direction, use only component upwind (not combinable with **key_tauskin_c_center**). This option use 8 to 12 points of current components (see figure below).
- **BBL**: computation is done via BBL module of CROCO using constant roughness (from 160 microns diameter). **This option is not recommended** with MUSTANG due to constant roughness.

default : 12 u,v points to compute in cell i,j



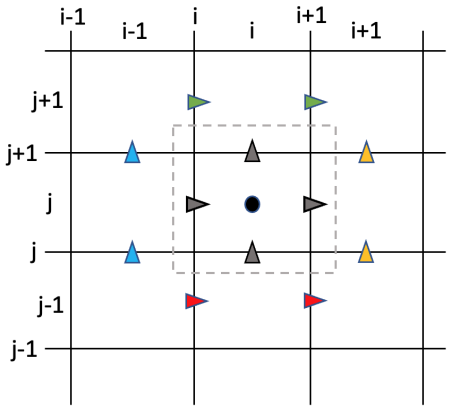
$$\text{ustar2\_u(i,j)} = \left[\frac{\kappa}{\log(z/z_0)}\right]^2 \sqrt{\text{u(i,j)}^2 + \left[\frac{\text{v(i,j)+v(i-1,j)+v(i,j+1)+v(i-1,j+1)}}{4}\right]^2}\ \text{u(i,j)}$$

$$\text{ustar2\_v(i,j)} = \left[\frac{\kappa}{\log(z/z_0)}\right]^2 \sqrt{\text{v(i,j)}^2 + \left[\frac{\text{u(i,j)+u(i+1,j)+u(i,j-1)+u(i+1,j-1)}}{4}\right]^2}\ \text{v(i,j)}$$

tauskin_x(i,j) = ρ (ustar2_u(i, j) + ustar2_u(i+1, j)) / 2
tauskin_y(i,j) = ρ (ustar2_v(i, j) + ustar2_v(i, j+1)) / 2

$$\text{tauskin\_c(i,j)} = \sqrt{(\text{tauskin\_x(i,j)})^2 + (\text{tauskin\_x(i,j)})^2}$$

key_tauskin_c_center : 4 u,v points to compute in cell i,j



$$\text{Urho(i,j)} = \left[\frac{u(i,j)+u(i+1,j)}{2}\right]$$

$$\text{Vrho(i,j)} = \left[\frac{v(i,j)+v(i,j+1)}{2}\right]$$

$$\text{speed(i,j)} = \sqrt{Urho(i,j)^2 + Vrho(i,j)^2}$$

$$\text{tauskin\_c(i,j)} = \rho \left[\frac{\kappa}{\log(z/z_0)}\right]^2 speed^2$$

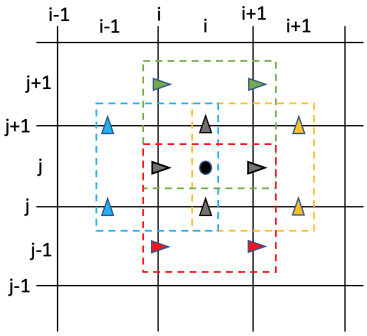
Wave skin friction

Wave skin friction is computed using Soulsby (1997) formula.

$tauskin\_wave = \frac{1}{2} \cdot \rho_w \cdot f_w \cdot Uwave^2$

With

key_tauskin_c_upwind : 8 to 12 u,v points to compute in cell i,j depending on current direction



$$\text{ustar2\_u(i,j)} = \left[\frac{\kappa}{\log(z/z_0)}\right]^2 \sqrt{\text{u(i,j)}^2 + \left[\frac{\text{v(i,j)+v(i-1,j)+v(i,j+1)+v(i-1,j+1)}}{4}\right]^2}\ \text{u(i,j)}$$

$$\text{ustar2\_v(i,j)} = \left[\frac{\kappa}{\log(z/z_0)}\right]^2 \sqrt{\text{v(i,j)}^2 + \left[\frac{\text{u(i,j)+u(i+1,j)+u(i,j-1)+u(i+1,j-1)}}{4}\right]^2}\ \text{v(i,j)}$$

If u(i,j) >0 & u(i+1,j)>0 : tauskin_x(i,j) = ρ ustar2_u(i, j)
else if u(i,j) <0 & u(i+1,j)<0 : tauskin_x(i,j) = ρ ustar2_u(i+1, j)
else : tauskin_x(i,j) = ρ (ustar2_u(i, j) + ustar2_u(i+1, j)) / 2

If v(i,j) >0 & v(i,j+1)>0 : tauskin_y(i,j) = ρ ustar2_v(i, j)
else if v(i,j) <0 & v(i,j+1)<0 : tauskin_y(i,j) = ρ ustar2_v(i, j+1)
else : tauskin_y(i,j) =ρ (ustar2_v(i, j) + ustar2_v(i, j+1)) / 2

$$\text{tauskin\_c(i,j)} = \sqrt{(\text{tauskin\_x(i,j)})^2 + (\text{tauskin\_x(i,j)})^2}$$

- f_w equal to **fricwav** *namelist namsedim_bottomstress* value or computed from *WAVE_OFFLINE file* variables ($Uwave$ and $Pwave$) and roughness if **l_fricwave** *namelist namsedim_bottomstress* value is True using $f_w = 1.39 \cdot (\frac{Uwave \cdot Pwave}{2 \cdot \pi \cdot z0sed})^{-0.52}$
- $Uwave$ and $Pwave$ reading from *WAVE_OFFLINE file*

Combinaison of current and wave stresses

Combinaison of current and wave stresses is done from Soulsby [1997] (Dynamics of marine sands - eq.69 and 70 page 92), $tauskin$ equal to tau_max

$$tau\_mean = tauskin\_current \cdot \left(1 + 1.2 \cdot \left(\frac{tauskin\_wave}{tauskin\_current + tauskin\_wave}\right)^{3.2}\right)$$

$$tau\_max = \sqrt{(tau\_mean + tauskin\_wave \cdot |cos(phi)|)^2 + (tauskin\_wave \cdot |sin(phi)|)^2}$$

With phi angle between current and wave.

> **Note**
>
> abs() is used on cos() and sin() because of alternative direction of tau_w in the vector addition of tau_c and tau_w (see Soulsby [1997] (Dynamics of marine sands - figure 16 page 89))

1.13.2.2.3.6 Settling

The settling process is taken into account during the advection-diffusion scheme in the water column and in the exchange from water to sediment via the deposit fluxes.

Settling velocity modelling strategy

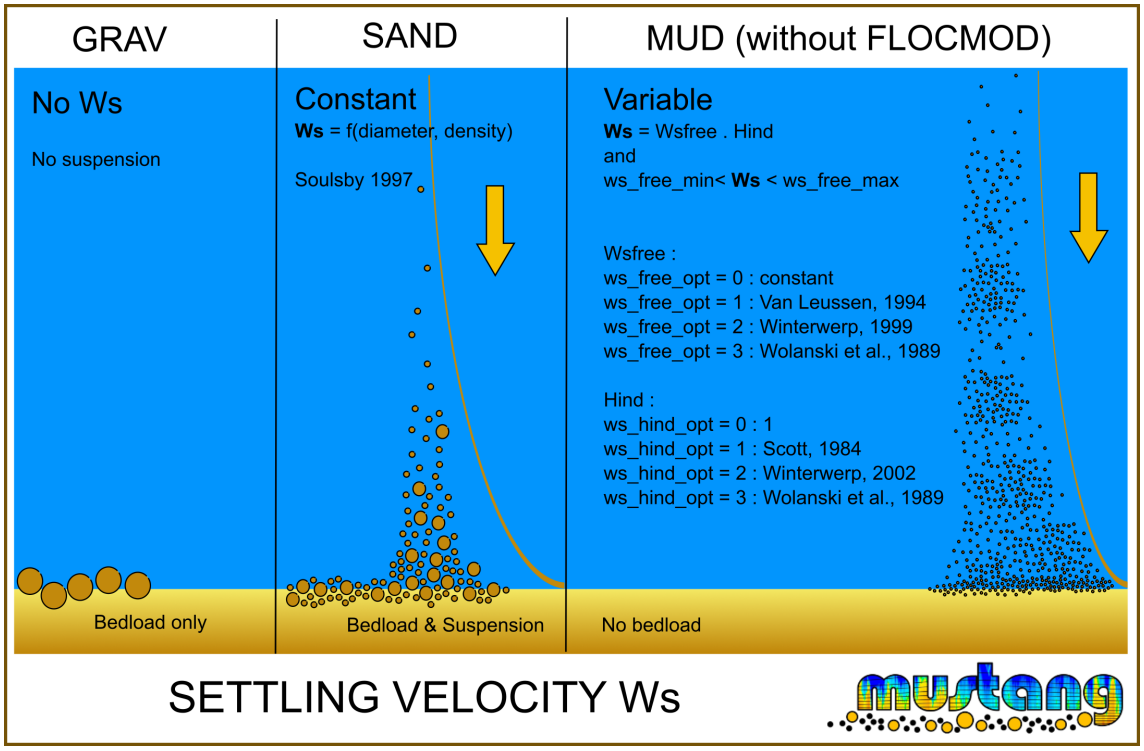
The settling velocity is the main variable of the settling process and is moddelled differently for each substance type :

- GRAV: gravels are not transported in suspension, they have no settling velocity because they are not in the water column
- SAND: sands have a constant settling velocity directly compute from Soulsby [1997] using their diameters defined in *SUBSTANCE namelist &nmlsands* (diam_n).

  $Ws = 10^{-6} \cdot \frac{(107.33 + 1.049 \cdot Dstar^3)^{0.5} - 10.36}{D}$

  With $Dstar = D \cdot 10^4 \cdot (g \cdot (\frac{\rho_s}{\rho_w} - 1))^{\frac{1}{3}}$ the dimensionless diameter of sediment and D diameter of sediment class, ρ_w water density and ρ_s sediment density.

  The resulting settling velocity could be high.



- MUD and Non Constitutive Particulate subtances: the settling velocity can vary in time and space depending on the parameters chosen by user in *SUBSTANCE namelist &nmlmuds*: ws_free_opt_n(), ws_free_min_n(), ws_free_max_n(), ws_free_para_n(1:4,num substance), ws_hind_opt_n(), ws_hind_para_n(1:2,num substance)) or by cpp key for flocculation module (see *Flocculation*)

  If flocculation module is not used, settling velocity is the result of the choice on ws_free_opt_n and ws_hind_opt_n values in *SUBSTANCE namelist &nmlmuds* and is computed for every cell "i,j" and layer "k" in the water domain.

  $Ws = max(ws\_free\_min\_n\ ;\ min(ws\_free\_max\_n\ ;\ Ws_{free} \cdot Hind))$

  See appropriate chapters for details on *free settling velocity* and *hindered settling parameter*.

- Sorbed substances: the settling velocity of sorbed substance is the same as the particulate susbtance to which it is associated

- Dissolved and fixed substances: no settling velocity

Free settling velocity

Ws_{free} the free settling velocity is computed from :

- if ws_free_opt_n = 0: constant value, $Ws_{free} = ws\_free\_min\_n$
- if ws_free_opt_n = 1: formulation of van Leussen [1994], $Ws_{free} = kC^m \cdot \frac{1+aG}{1+bG^2}$

  With :

  - $k = ws\_free\_para\_n(1)$ (= 0.0005 in the reference)
  - $m = ws\_free\_para\_n(2)$ (= 1.2 in the reference)
  - $a = ws\_free\_para\_n(3)$ (= 0.3 in the reference)
  - $b = ws\_free\_para\_n(4)$ (= 0.09 in the reference)
  - C = Sum of concentration of MUD substances in the layer
  - $G = \sqrt{\frac{turbulence\ dissipation\ rate}{vertical\ viscosity\ coefficient}}$ = turbulence energy

- if ws_free_opt_n = 2: formulation of Winterwerp [1999], $Ws_{free} = \frac{1}{18} \cdot \frac{(\rho_s - \rho_w)g}{\rho_w \nu} \cdot Dp^{3-nf} \cdot De^{nf-1}$

  With :
  - $De = max(Dp + \frac{ka \cdot C}{kb \cdot \sqrt{G}}\ ;\ \sqrt{\frac{\nu}{G}})$
  - $Dp = ws\_free\_para\_n(1)$ = Primary Particle Diameter, (= 4.10-6 in the reference)
  - $ka = ws\_free\_para\_n(2)$ = aggregation factor, (= 14.6 in the reference)
  - $kb = ws\_free\_para\_n(3)$ = breakup factor, (= 30000 in the reference)
  - $nf = ws\_free\_para\_n(4)$ = fractal dimension, (= 2 in the reference)
  - C = Sum of concentration of MUD substances in the layer
  - $G = \sqrt{\frac{turbulence\ dissipation\ rate}{vertical\ viscosity\ coefficient}}$ = turbulence energy
  - $\nu = 0.00000102\ m^2/s$ = water kinematic viscosity
  - g = gravity
  - ρ_s = sediment density
  - ρ_w = water density
- if ws_free_opt_n = 3: formulation of Wolanski *et al.* [1989], $Ws_{free} = kC^m$

  With :
  - $k = ws\_free\_para\_n(1)$ (= 0.01 in the reference)
  - $m = ws\_free\_para\_n(2)$ (= 2.1 in the reference)
  - C = Sum of concentration of MUD substances in the layer

Hindered settling parameter

$Hind$ the hindered settling parameter is computed from :

- if ws_hind_opt_n = 0: no hindered effect, $Hind = 1$
- if ws_hind_opt_n = 1: formulation of Scott, 1984, $Hind = (1 - \phi)^m$

  With :
  - $\phi = min(1\ ;\ \frac{C}{cgel})$
  - C = Sum of concentration of MUD substances in the layer
  - $cgel = ws\_hind\_para\_n(1)$ (= 40 in the reference)
  - $m = ws\_hind\_para\_n(2)$ (= 4.5 in the reference)
- if ws_hind_opt_n = 2: formulation of Winterwerp *et al.* [2002], $Hind = (1 - \phi_v)^m \cdot \frac{(1-\phi)}{(1+2.5\phi_v)}$

  With :
  - $\phi = \frac{C}{\rho_s}$
  - If ws_free_opt_n is not 2 then $\phi_v = \frac{C}{cgel}$
  - If ws_free_opt_n is 2 then $\phi_v = \phi \cdot (\frac{De}{Dp})^{3-nf}$
    - $De = max(Dp + \frac{ka \cdot C}{kb \cdot \sqrt{G}}\ ;\ \sqrt{\frac{\nu}{G}})$
    - $Dp = ws\_free\_para\_n(1)$ = Primary Particle Diameter, (= 4.10-6 in the reference)
    - $nf = ws\_free\_para\_n(4)$ = fractal dimension, (= 2 in the reference)
  - $cgel = ws\_hind\_para\_n(1)$ (= 40 in the reference)
  - $m = ws\_hind\_para\_n(2)$ (= 1 in the reference)
  - C = Sum of concentration of MUD substances in the layer

  - ρ_s = sediment density

- if ws_hind_opt_n = 3: formulation of Wolanski *et al.* [1989], this formulation has to be combined with ws_free_opt_n = 3.

  If ws_free_opt_n is not 3 then no hindered effect $Hind = 1$

  If ws_free_opt_n is 3 then $Hind = \frac{1}{(C^2+b_w^2)^{m_w}}$

  With :

  - $b_w = ws\_hind\_para\_n(1)$ (= 2 in the reference)
  - $m_w = ws\_hind\_para\_n(2)$ (= 1.46 in the reference)
  - C = Sum of concentration of MUD substances in the layer

Treatment of high settling velocities: SAND variables

For high settling velocities, the major part of the sediment are concentrated in a thin layer near bottom. The thickness of the bottom layer in water could be unadapted to represent correctly the concentration at bottom. This could lead to an underestimation of deposit fluxes and horizontal transport fluxes.

Furthermore, to avoid numerical instabilities, vertical transport is computed using sub time steps. The number of sub time steps depends on the settling velocity of each substance. For high settling velocities, this could be time consuming.

In MUSTANG, these issues are considered for SAND variables only. It is considered that MUD have too low settling velocities and GRAVEL are not transported in suspension.

That is why three features have been developped for SAND variables in MUSTANG to treat these issues :

- vertical deposit fluxes are corrected considering a Rouse profile in the bottom layer in water. The concentration is extrapolated at a given reference height (parameter **aref_sand** of *MUSTANG namelist*) to obtain a correct deposit flow.
- horizontal advection fluxes could, as an option, be corrected considering a Rouse profile of concentration and the current logarithmic gradient near bottom. This option is activated by the cpp key **MUSTANG_CORFLUX**
- SAND sediments could, as an option, be considered as 2D variables with the cpp key **key_sand2D**. When this ccp key **key_sand2D** is activated, a boolean **l_sand2D** in *SUBSTANCE namelist &nmlsands* specify for each SAND substance if it has to be treated as a 2D variable and not 3D. This option makes it possible to treat the fall of SAND-type sediments by considering that the transport in suspension only takes place in the bottom layer (2D sand transport in a single layer). This makes it possible to skip the vertical transport and mixing for this susbtance and save calculation time. As an option (**l_outsandrouse** in *SUBSTANCE namelist &nmlsands*), the Rouse profile is reconstructed in the outputs.

The Rouse profile used in those three features involves the ratio Ws/u* to calculate the Rouse number (Z in the formulation below). The formulations used are those proposed by Julie Vareilles (activity report of the post-doctorate Consequences of Climate Change on Ecogeomorphology of Estuaries, March 2013)

Rouse profile:

$C(z) = C(a) \cdot (\frac{h-z}{z} \cdot \frac{a}{z-a})^Z$

With:

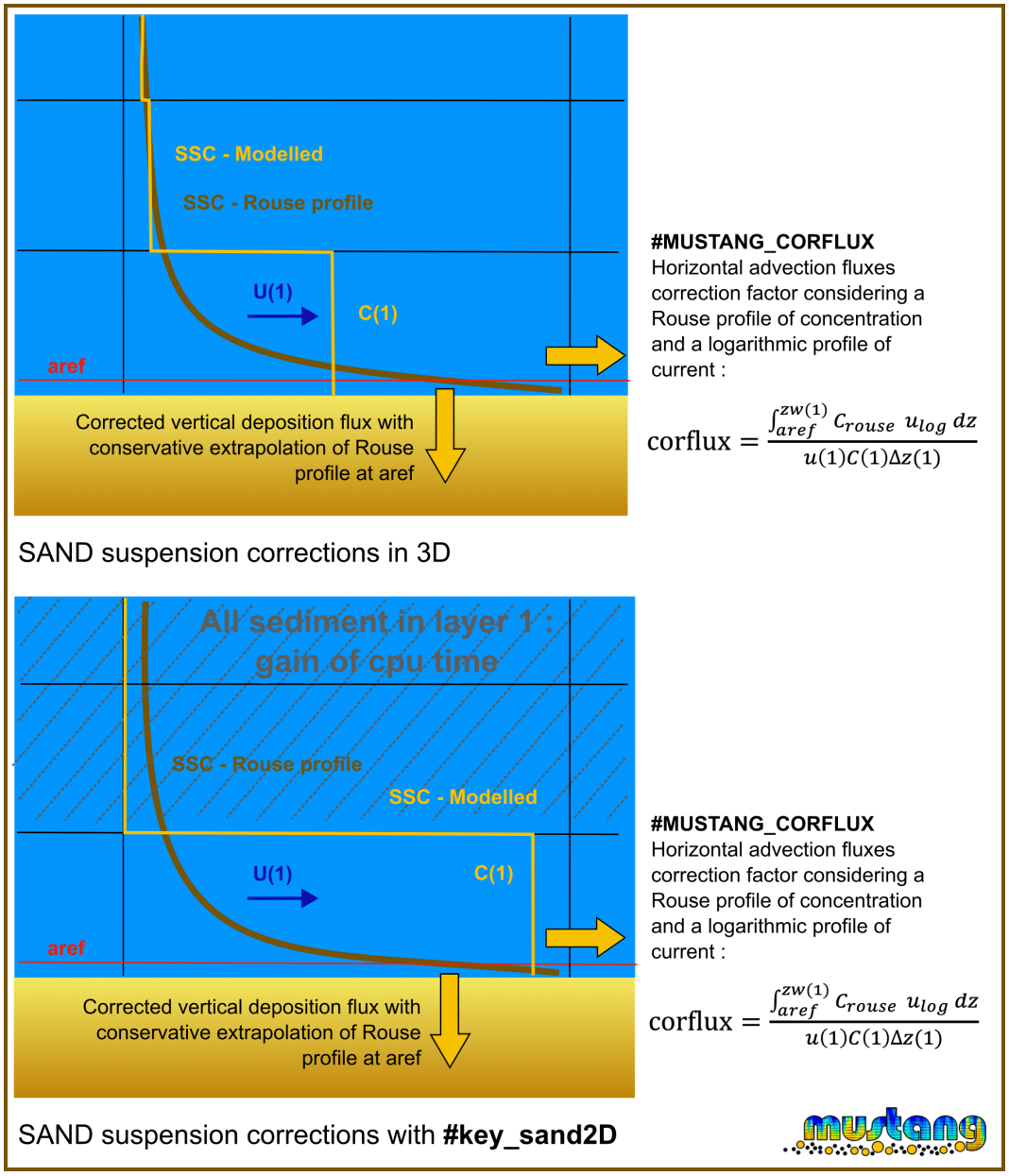
- a: reference height
- h: water height
- Z: Rouse number

For $\frac{W_s}{u_*} < 0.1$: $\beta = 1, Z = \frac{W_s}{\kappa \cdot u_*}$

For $0.1 <= \frac{W_s}{u_*} < 0.75$: $\beta = 1 + 2(\frac{W_s}{u_*})^2, Z = \frac{W_s}{\beta \cdot \kappa \cdot u_*}$

For $0.75 <= \frac{W_s}{u_*} < 1.34$: $Z = 0.35\frac{W_s}{u_*} + 0.727$

For $1.34 <= \frac{W_s}{u_*}$: $Z = 1.2$



1.13.2.2.3.7 Flocculation with FLOCMOD

1.13.2.2.3.8 Introduction

Suspended particulate matter (SPM) dynamics in estuaries and coastal seas is driven by flocculation processes. These processes modify floc sizes and floc densities, and hence their settling velocity. This parameter is crucial when modelling SPM transport in coastal systems. It can be set as a constant value, or evaluated though an empirical function (such as Van Leussen relationship, see *settling velocity chapter*) to mimic flocculation dynamics with processes "at equilibrium".

FLOCMOD is a 0D size-class-based module developed by Ifremer to simulate the explicitly flocculation processes (See Verney *et al.* [2011]). It based on the population equation system originally proposed by Smoluchowski [1917]. As a size class-based model, this means that the floc size distribution is represented by a discrete number of sediment classes of increasing sizes. This module simulates the effect of turbulence and SPM concentration and flocculation processes, and uses the fractal approach to represent main floc properties (floc density, floc mass, floc settling velocity).

Future developments are scheduled to include seasonal variability in organic matter content and hence modulate flocculation processes, and floc resistance to breakup for instance.

1.13.2.2.3.9 Module description: FLOCMOD in CROCO

FLOCMOD is available in CROCO using the SEDIMENT (USGS) module or the MUSTANG module. Using MUSTANG, **FLOCMOD** is activated with the cppkey **#key_mustang_flocmod**. The MUSTANG specific cpp-keys must also be activated.

Floc size classes are defined as mud types in *SUBSTANCE namelist parasubstance_MUSTANG.txt*. The floc diameter in the namelist correspond to the floc size of the given class. Update the number of sediment classes in param.h (ntrc_sub) accordingly. You may also update obc and input files and sediment initial distribution file if you want to prescribe a specific floc size distribution from rivers or open boundary conditions. Floc sizes are discrete classes, which means that floc created by aggregation or fragmentation is redistributed along the two nearest classes (in floc mass) using a mass-conservative interpolation scheme based on inverse distance.

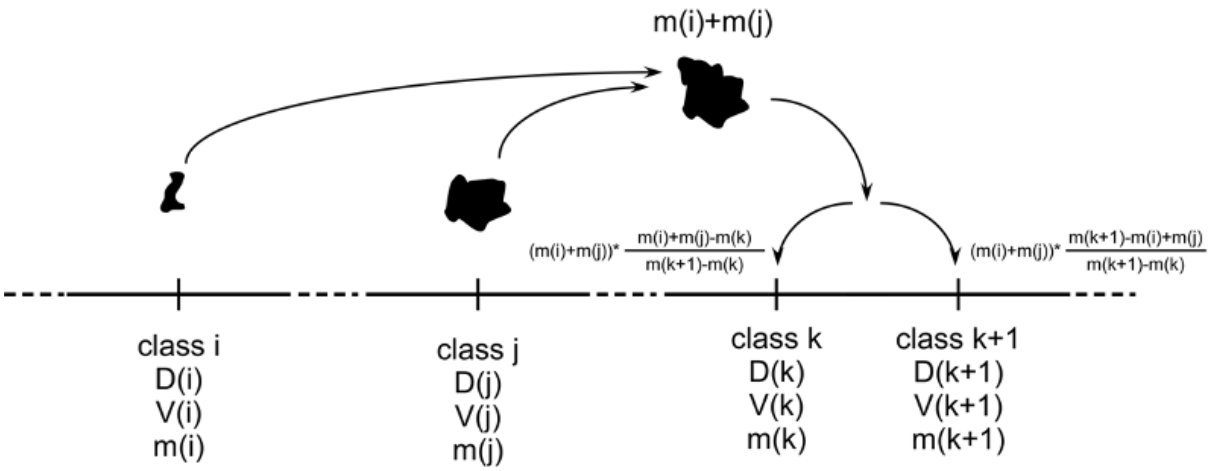


Fig. 8: Flocmod redistribution on discrete classes

FLOCMOD parameters are all defined in *MUSTANG namelist* within the flocmod namelist *namflocmod*. The different parameters are listed and detailed below when describing flocculation processes.

To save computation time, probability kernels are mainly pre-processed before the computation loop, except the fragmentation by collision kernel. Be careful, if **l_COLLFRAG** is activated, computation time can be very significantly impacted.

1.13.2.2.3.10 FLOCMOD processes

Flocculation processes are actually competition between aggregation and breakup mechanisms, driven by turbulence, SPM concentration, and potentially salinity and organic matter content. The last two control parameter are not yet included in FLOCMOD explicitly. Aggregation is controlled by turbulent shear and differential settling, while fragmentation is exclusively controlled by shear, with different (concurrently usable) options: shear fragmentation, shear erosion, collision-induced fragmentation. All floc interactions are limited to "two-body" interactions. The generic equation given below define all processes involved in FLOCMOD, both in term of gain (G) or loss (L) per class. The main model variable is the number concentration of flocs Nk in a given class k. ASH, ADS and fragmentation by collision can be activated using boolean l_ASH, l_ADS and l_COLLFRAG respectively from the namelist.

1.13.2.2.3.11 Floc fractal approach

Flocs observed in nature are characterized by a wide range of size and densities, based on mineral intrinsic features, the organic matter content in SPM, and based on hydrological and hydrodynamics factors. In general, small flocs have high excess densities (O(100-1000 kg/m3)), and large flocs low densities (O(10-100kg/m3)). In FLOCMOD, we use the fractal approach to represent floc characteristics [Kranenburg, 1994]: floc size D, floc mass M, floc excess density $\Delta\rho$ $(\rho - \rho_w)$. Flocs are formed from N primary particles with a unique size D_p (f_dp0 in the

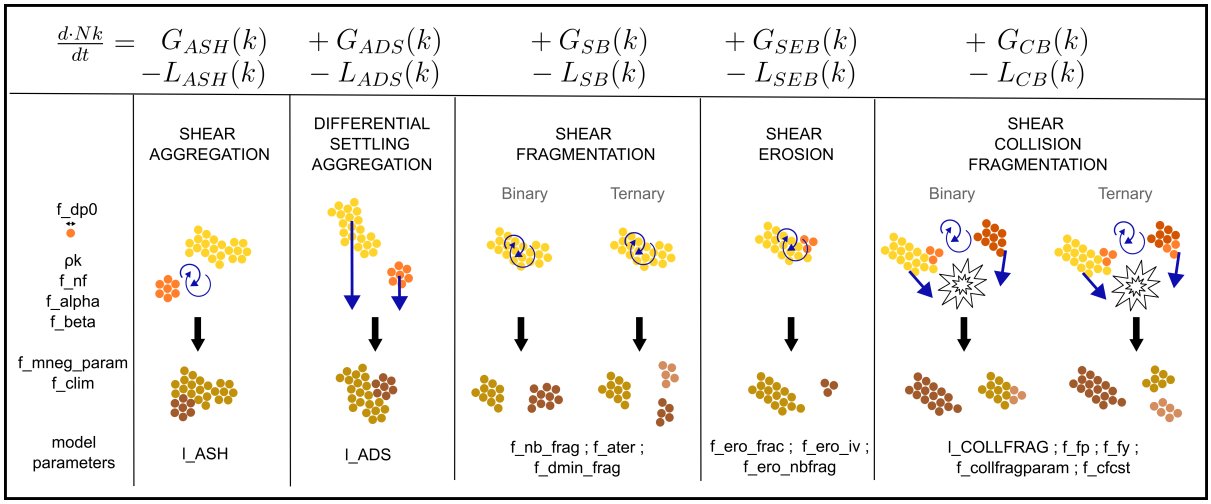


Fig. 9: Flocmod processes

namelist) and density ρ_p (ros from the *SUBSTANCE namelist parasubstance_MUSTANG.txt*). The floc structure (~compacity) is controlled by the fractal dimension n_f (f_nf) such as:

$N = (\frac{D}{D_p})^n_f$

$M = \rho_p \cdot \frac{\pi}{6} \cdot {D_p}^3 \cdot N$

$\Delta\rho = (\rho_p - \rho_w) \cdot (\frac{D}{D_p})^{3-n_f}$

1.13.2.2.3.12 Shear rate G

The shear rate G is an hydrodynamic parameter. If GLS_MIXING cppkey is used in CROCO, G can be directly calculated from ϵ such as:

$G = \sqrt{\frac{\epsilon}{\nu}}$

Otherwise, an ϵ vertical profile is calculated from Nezu and Nakagawa and the bottom friction velocity u^* (from MUSTANG) such as:

$\epsilon = \frac{u^{*3}}{\kappa h} \frac{h-z}{z}$

Where h is the water depth and z the distance from bottom.

1.13.2.2.3.13 Shear aggregation (G_{ASH} ; L_{ASH})

These terms correspond to the gain or loss of class k particles when i) two particles in movement (by shear) collide and ii) the collision is efficient, i.e. the newly formed bound between the two particles can withstand the shear induced by the collision. The two-body collision probability function $A_{SH}(i,j)$ is a function of the shear rate G and particle diameters D_i and D_j [McAnally and Mehta, 2000, McAnally and Mehta, 2002].

$G_{ASH}(k) = \frac{1}{2} \sum_{M_i+M_j=M_k} \alpha_{ij} A_{SH}(i,j) n_i n_j$

$L_{ASH}(k) = \sum_{i=1}^{nc} \alpha_{ik} A_{SH}(i,k) n_i n_k$

$A_{SH}(i,j) = \frac{G}{6}(D_i + D_j)^3$

α_{ij} is the collision efficiency representing particle cohesiveness, i.e. the physico-chemical forces and the sticking properties of organic matter. In the actual version of FLOCMOD, $\alpha_{ij} = \alpha$ is a constant parameter (between 0 and 1) set in the FLOCMOD namelist.

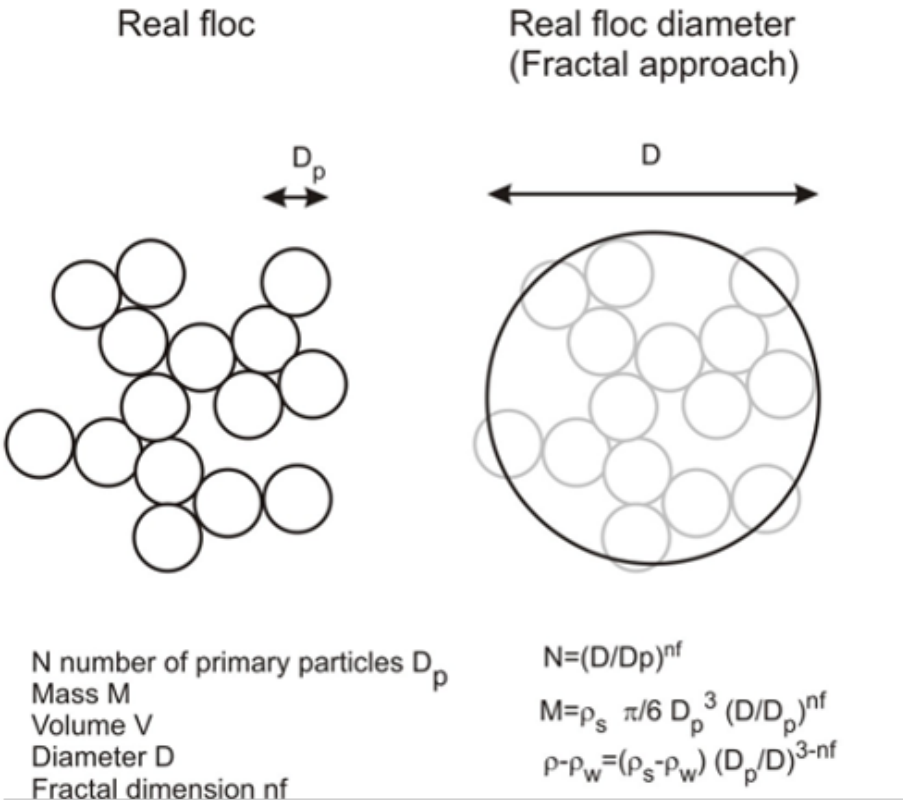


Fig. 10: Flocmod fractal approach

1.13.2.2.3.14 Differential settling aggregation (G_{ADS} ; L_{ADS})

A_{DS} represents collisions and aggregation that can occur when 2 flocs with different settling velocities can interact during settling. Kernels are similar to G_{ASH} and L_{ASH}, except that the collision probability A_{SH} is substituted with A_{DS} such as:

$A_{DS}(i,j) = \frac{\pi}{4}(D_i + D_j)^2 \mid W_{s,i} - W_{s,j} \mid$

Where $W_{s,i}$ is the floc settling velocity of class i, calculated from Stokes :

$W_{s,i} = \frac{g}{18\mu}\Delta\rho_i {D_i}^2$

1.13.2.2.3.15 Shear fragmentation (G_{SB} ; L_{SB})

Shear fragmentation represents the action of shear (turbulent) forces on flocs, leading to breakup.

$G_{SB}(k) = \sum_{i=k+1}^{nc} FDSB_{ki} B_i n_i$

$L_{SB}(k) = B_k n_k$

$B_i = \beta_i G^{\beta_2} D_i (\frac{D_i - D_p}{D_p})^{\beta_3}$

β_i is the fragmentation rate, indirectly related to yield strength and floc resistance to breakup. In the actual version of FLOCMOD, β_i is a constant set in the FLOCMOD namelist. Following Winterwerp *et al.* [2002], *beta_2 = 3/2* and *beta_3 = 3 - n_f*.

FDSB is the size distribution function of fragmented flocs by shear. Two modes are available: binary or ternary breakup.

- Binary distribution: fragmentation of a floc with mass m_i in two identical flocs of mass $m_i/2$.

$$FDSB_{ij} = \begin{cases} 2 & \text{if } m_j = \frac{m_i}{2} \\ 0 & \text{otherwise} \end{cases}$$

- Ternary distribution: fragmentation of a floc with mass m_i in one floc of mass $m_i/2$ and 2 flocs of mass $m_i/4$.

$$FDSB_{ij} = \begin{cases} 1 & \text{if } m_j = \frac{m_i}{2} \\ 2 & \text{if } m_j = \frac{m_i}{4} \\ 0 & \text{otherwise} \end{cases}$$

In FLOCMOD, binary fragmentation is activated if f_nb_frag = 2 and f_ater = 0, and ternary fragmentation if f_nb_frag = 2 and f_ater = 0.5; Shear breakup can be only applied from a given floc size to limit fragmentation for small flocs. This can be set in FLOCMOD by changing f_dmin_frag.

1.13.2.2.3.16 Erosion fragmentation

Shear fragmentation by floc erosion is an additional option to represent floc breakup. In this case, flocs are not broken by 2 or 4 but a small fraction of the floc mass is eroded from the parent floc. This mode transfers a part of the shear fragmentation probability to floc erosion using f_ero_frac. This means that (1-f_ero_frac) contributes to binary/ternary fragmentation and f_ero_frac to fragmentation by erosion.

G_{SEB} and L_{SEB} are calculated identically to G_{SB} and L_{SB}, except that the fragmentation distribution changes according to the floc erosion parameter (FDSEB instead of FDSB):

$$FDSEB_{ij} = \begin{cases} 1 & \text{if } m_j = m_i - f\_ero\_nbfrag \cdot m_i \\ 2 & \text{if } m_j = m_i \\ 0 & \text{otherwise} \end{cases}$$

1.13.2.2.3.17 Collision fragmentation

Collision can not only contribute to form bigger flocs, but instead if turbulence is strong, collision can overpass floc strength and then contribute to break particles by mechanical failure. Hence, when activating this option, part of the collision probability ($A_{SH}(i,j)$) can induce fragmentation.

$G_{CB}(k) = \sum_{i=1}^{nc} \sum_{j=i}^{nc} FDCB_{ij} \cdot A_{SH}(i,j) \cdot n_i \cdot n_j$

$L_{CB}(k) = \sum_{i=1}^{nc} FDCB_{ik} \cdot A_{SH}(i,k) \cdot n_i \cdot n_k$

$FDCB_{ij}$ is the distribution of flocs after collision, and is function of the floc strength $\tau_{y,i}$ and the collision-induced shear stress between floc i and j: $\tau_{collij,i}$.

$\tau_{collij,i} = \frac{8}{\pi} \frac{(G\frac{D_i+D_j}{2})^2}{F_p \cdot {D_i}^2 (D_i+D_j)} \frac{M_i \cdot M_j}{M_i+M_j}$

$\tau_{y,i} = F_y \frac{\Delta\rho_i}{\rho_w}^{\frac{2}{3-nf}}$

Fp and Fy can be user-defined in the FLOCMOD namelist as f_fp and f_fy respectively.

For two-body interactions (i and j), two types of failures are likely to happen and control the expression of $FDBC_{ij}$ [McAnally, 1999]:

- Collision fragmentation #1 : $\tau_{y,i} > \tau_{collij,i}$ and $\tau_{collij,i} > \tau_{y,j}$: the collision-induced shear stress exceeds the shear strength of the weakest aggregate only, consequently during the collision, the j floc breaks into two fragments ($F_{j,1}$ and $F_{j,2}$) such as $MF_{j,1} = (1 - cfcst) \cdot Mj$ and $MF_{j,2} = cfcst \cdot Mj$. $F_{j,1}$ is a free fragment while $F_{j,2}$ is bound with the i floc. cfcst is the inter-penetration depth, fixed as 3/16 based on McAnally [1999]. This parameter is found in the namelist as f_cfcst.
- Collision fragmentation #2: $\tau_{y,i} < \tau_{collij,i}$ and $\tau_{collij,i} > \tau_{y,j}$: the shear stress is larger than shear strength of both i and j flocs. Both flocs break into two fragments ($F_{i,1}$; $F_{i,2}$) and ($F_{j,1}$; $F_{j,2}$) such as $[m_F i, 1; m_F j, 1] = (1 - cfcst)[M_j; M_j]$ and $[M_F i, 2; m_F j, 2] = cfcst[M_j; M_j]$. Two particles are formed from the two parent flocs $F_{i,1}$ and $F_{j,1}$. A third flocs is formed from the two fragments ($F_{i,2}$ and $F_{j,2}$) that bound during collision.

The fraction of collision contributing to floc breakup is controlled through the parameter f_collfragparam.

1.13.2.2.3.18 FLOCMOD execution

FLOCMOD can be non-conservative for high shear rates and/or high SPM concentration. To prevent for instabilities, FLOCMOD includes a sub-time step algorithm. After mass exchange due to flocculation, FLOCMOD checks if the new floc size distribution is fully positive or null. If true, execution continues. Otherwise, the time step is divided by two and a new floc size distribution is recalculated. This time-step adaptation is applied as long as floc size distribution contains negative mass. Flocculation is then repeated until reaching a cumulated time step corresponding to the CROCO time step.

It is possible to be slightly permissive and allow a small negative mass concentration (f_mneg_param). In this case, the class characterized by negative mass is set to 0 and the "corresponding added mass" is proportionally removed from the positive classes to be mass conservative. This can help to limit time step adaptation, and hence reduce computation costs.

It is also possible to disconnect FLOCMOD when SPM concentration is very low. In FLOCMOD, this concentration threshold (in g/l) is defined in f_clim and set by default to 0.001 g/l.

1.13.2.2.3.19 Example

Test cases are provided see *FLOCMOD cases* for more informations.

1.13.2.2.3.20 Erosion process

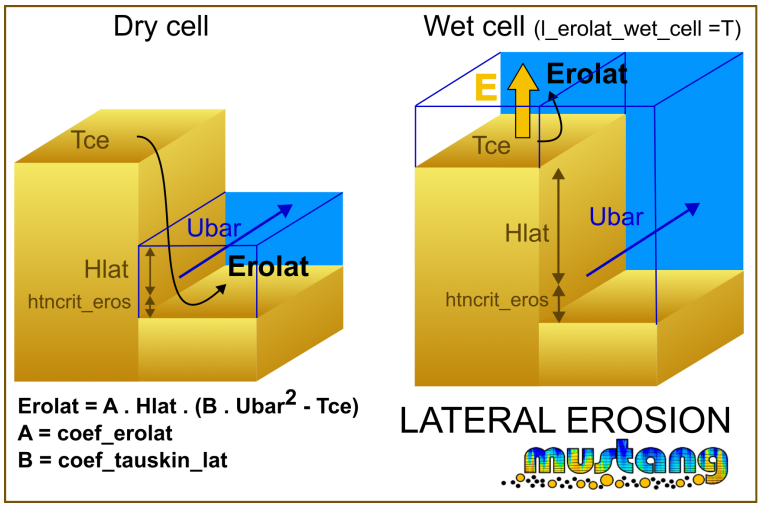
> **Note**
>
> Patience, work in progress, meanwhile see: https://mars3d.ifremer.fr/docs/doc_MUSTANG/doc.MUSTANG.erosion.html

Erosion fluxes

Lateral erosion

key_MUSTANG_lateralerosion

> **Note**
>
> Patience, work in progress



Erosion, layer managment

1.13.2.2.3.21 Deposit process

> **Note**
>
> Patience, work in progress, meanwhile see: https://mars3d.ifremer.fr/docs/doc_MUSTANG/doc.MUSTANG.deposit.html

Deposit processes are treated implicitly in water transport equations. First, deposition flux trends are evaluated for each variable before advection computations.

Then after advection resolution, effective deposit is computed with new concentrations in water (estimated after transport and settling) and sediment layers are updated (see *Deposition layer managment*)

Deposit fluxes

TODO: add formulation used for deposit fluxes

Sliding fluxes

A sliding process of the fluid mud is implemented. Only MUD sediments are concerned by this feature. The modelling strategy consists in :

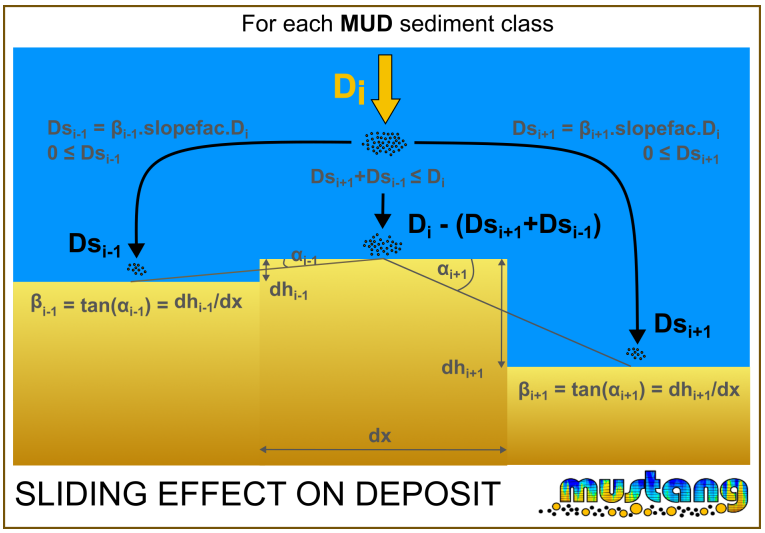
- compute the part of mud which slides if the slope is steep
- deposit this part towards lower neighboring cells according to the slope

To activate this behavior, the cppkey **#key_MUSTANG_slipdeposit** must be define and the **slopefac** value in MUSTANG namelist *&namsedim_deposition* must be greater than 0.

The part of mud which slides on each cell limit is the product of the slope by the deposit fluxe for each mud class in the cell and by the factor slopefac.

No sediment slides if the slope is not positive.

The sum of sliding sediment over the four cell's limits could not be greater than the deposit flux computed in the cell.



Deposit layer managment

> **Note**
>
> Patience, work in progress

1.13.2.2.3.22 Consolidation

The mixed-sediment consolidation model, detailed in [Grasso *et al.*, 2015], is based on [Toorman, 1996] unifying theory for sedimentation and consolidation of several classes of sediment.

Following Merckelbach's derivation of Gibson equation, and using as state variable the mass concentration of each sediment class Ci, the mass conservation equation during consolidation can be written as:

$\frac{\partial C_i}{\partial t} = + \frac{\partial}{\partial z}[\frac{k}{\rho_w} C_i \triangle(load)]$ (Equation 2)

with

$\triangle(load) = C \frac{\rho_s - \rho_w}{\rho_s} + \frac{1}{g}\frac{\partial \sigma'}{\partial z}$

where :

- C is the sediment total mass concentration, assuming the same grain density ρ_s for all sediment classes *i*,
- *k* is the permeability (m/s),
- ρ_w is the water density,
- *g* the gravity
- σ' the effective stress.

In order to account for segregation due to polydispersity during sedimentation, the sand settling velocity was chosen as the maximum between the sedimentation rate in Eq.2 and the hindered settling velocity Ws_{si} hindered of the sand class *si* considered.

The mud fraction, however, is only driven by the sedimentation rate in Eq. 2, so that finally the following equation 3 is solved:

$\frac{\partial C_i}{\partial t} = + \frac{\partial}{\partial z}[C_i MAX(\frac{k}{\rho_w} C_i \triangle(load), Ws_{si,hindered}]$ (Equation 3)

We used a segregation formulation based on the relative mud concentration (C_{relmud}):

$C_{relmud} = \frac{C_{mud}}{1-\varphi_{sand}} = \varphi_{relmud}\rho_s$ (Equation 4)

with :

- C_{mud} the mass concentration of mud (clay and silt)
- φ_{sand} the volumetric concentration of sand (grain diameter > 63 µm), to express the hindered settling of sand class *si* as:

  $Ws_{si,hindered} = Ws_{si}[1 - \frac{C_{relmud}}{C_{relmud_{crit}}}]^p$ (Equation 5)

  where :

  - Ws_{si} is the non-hindered settling velocity estimated by Souslby's (1997) formulation and the power *p* is defined as 4.65 according to Richardson and Zaki's (1954) observations.
  - $C_{relmud_{crit}}$ is an empirical parameter calibrated in order that the sand settling becomes hindered by fine (muddy) particles when their relative concentration get close to a threshold value.

The resolution of Eq.5 requires the specification of two constitutive relationships for the permeability and the effective stress, respectively (e.g. [Alexis *et al.*, 1992]; [Toorman, 1999]).

The permeability constitutive relationship is computed in coupling two formulations.

The first is related to the void ratio e (e.g. [Bartholomeeusen *et al.*, 2002]; [Le Hir *et al.*, 2011]), which reads:

$k_e = k_1 e^{k_2}$ (Equation 6)

and the second is related to the relative volume fraction of fine particles relmud (see Eq.5), based on the fractal theory presented by [Merckelbach and Kranenburg, 2004] and [Merckelbach and Kranenburg, 2004], expressed as:

$k_\varphi = K_k \cdot \varphi_{relmud}{}^{-n}$ (Equation 7)

with

- $n = \frac{2}{3-n_f}$
- n_f is the fractal number that characterizes the distribution of solids in the sediment.

Similarly, this fractal theory enabled to compute the effective stress as:

$\sigma' = K_d \cdot \varphi_{relmud}{}^n$ (Equation 8)

where k_1, k_2, K_k, K_d, n are empirical parameters.

Note

In the code, values from [Grasso *et al.*, 2015] are:

- k_1 = 2e-9
- k_2 = 3.7

And in *namsedim_consolidation*, default parameters from [Grasso *et al.*, 2015] are :

- K_k corresponding to xperm1 = 4e-12
- K_d corresponding to xsigma1 = 6e+05
- n corresponding to xsigma2 = 6 (and corresponding to xperm2 = -6)

1.13.2.2.3.23 Diffusion within sediment and at interface

Cpp keys involved: #key_noTSdiss_insed #key_nofluxwat_IWS

Warning

NOT TESTED YET IN CROCO Documentation to come after. Meanwhile: https://mars3d.ifremer.fr/docs/doc_MUSTANG/doc.MUSTANG.diffu.html

1.13.2.2.3.24 Bioturbation

Warning

NOT TESTED YET IN CROCO Documentation to come after. Meanwhile: https://mars3d.ifremer.fr/docs/doc_MUSTANG/doc.MUSTANG.bioturb.html

1.13.2.2.3.25 Suspended sediment concentration effect on density

Witt cpp key SED_DENS, effects of suspended sediment on the density field are included with terms for the weight of each sediment class in the equation of state for seawater density as:

$$\rho = \rho_w + \sum_{i=1}^{nvpc} \frac{C_i}{\rho_{s,i}} \left(\rho_{s,i} - \rho_w\right)$$

This enables the model to simulate processes where sediment density influences hydrodynamics, such as density stratification and gravitationally driven flows.

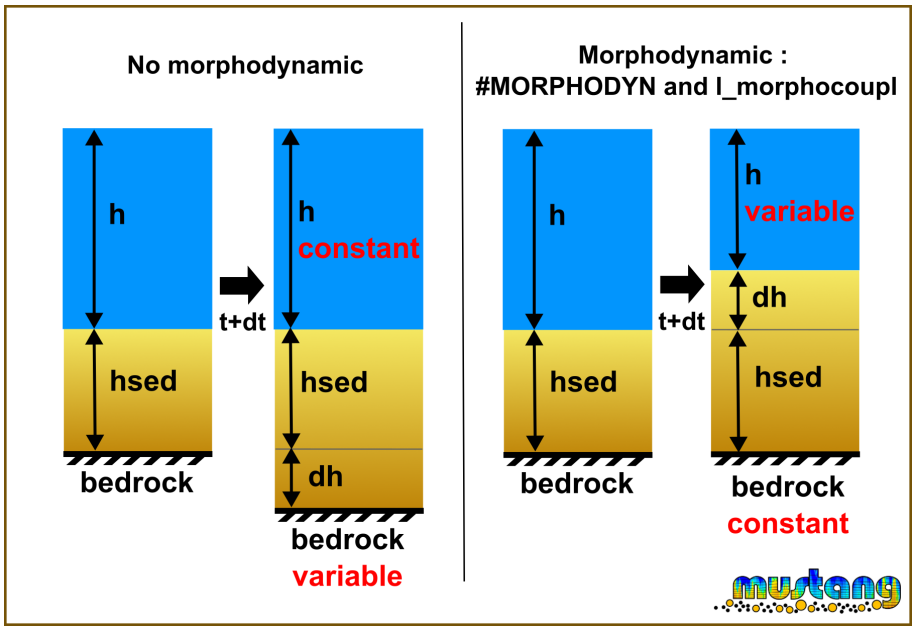
Note

If key_sand2D is used, sand variable treated as 2D variable are excluded of the sum and do not modify density

1.13.2.2.3.26 Morphodynamic

To activate morphodynamic means that the bathymetry used in the hydrodynamic model will evoluate with time. The following figure shows the difference between a non-morphodynamic (ie morphostatic) simulation and a morphodynamic simulation.

The user needs to activate cpp key #MORPHODYN and to set to true the boolean l_morphocoupl (see *&namsedim_morpho*) to run in morphodynamic mode.



The user can also accelerate morphologic evolution by using MF parameter (see *&namsedim_morpho*). In this case, two option are available to accelerate the changes :

- MF could amplified directly sediment height variation (l_MF_dhsed = T) on water height. In this case, sediment height and the bed layers compositions are not modified. Bedrock location is modified.
- MF could amplified erosion/deposition fluxes (l_MF_dhsed = F). In this case, depending on sediment classes in the simulation, the bed layer composition could be different from the case without MF or with (l_MF_dhsed = T) as coarse sediment settled before small one. Sediment height and bed layers compositions are modified but bedrock location is maintained.

1.13.2.2.4 Specific features

1.13.2.2.4.1 Available online diagnosis: SUBMASSBALANCE

This functionnality allows you to compute for each substance :

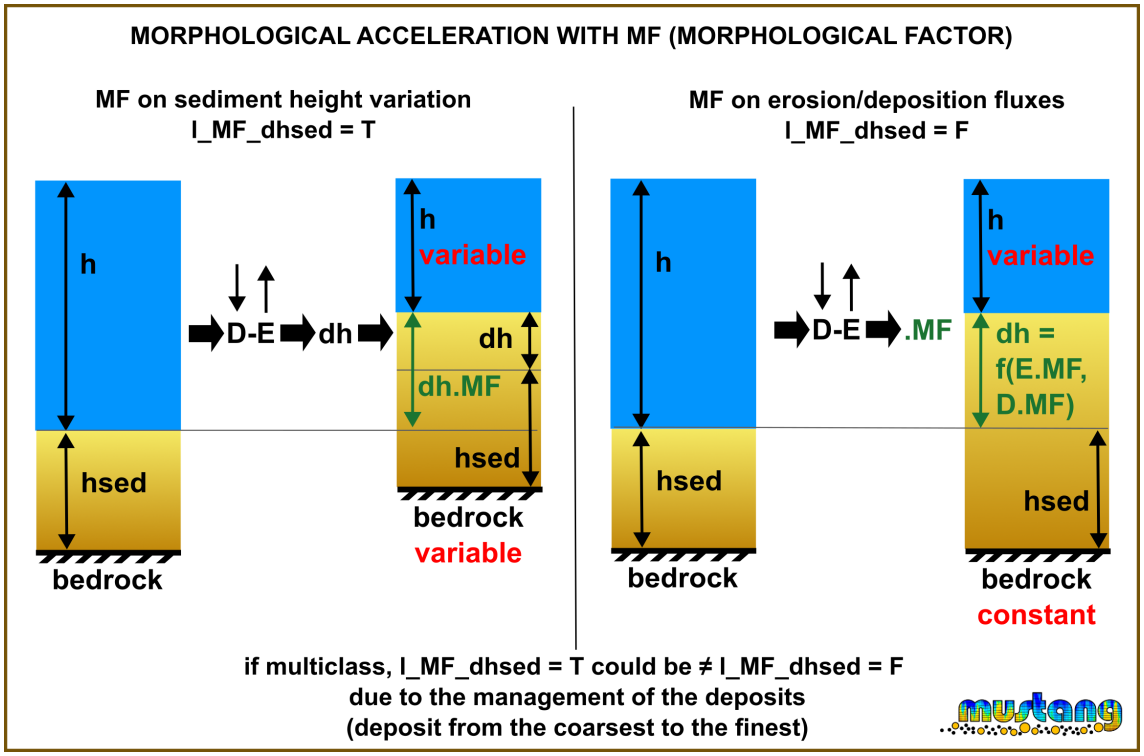
- fluxes through boundaries
- budgets (stocks and fluxes) in sub-domains

By default, one domain is considered, containing all the computational grid. User can also define one or several subdomain and boundaries in a specific file (see *submassbalance input file*).

To use this functionnality: set submassbalance_l=.true. in *SUBSTANCE namelist* and activate cppkey SUBSTANCE_SUBMASSBALANCE.

Two type of borders can be defined: closedsub-domains or open boundaries.

- For closed sub-domain (a budget sub-domain is only valid if the boundary is closed), this functionnality computes:
  - net cumulated fluxes through water (since the start date of massbalance computation) in and out of a given sub-domain,



  - net cumulated input fluxes into the sub-domain due to rivers discharges,
  - total budget of substance in the sub-domain (total budget must be constant if conservative substance),
  - stocks in the water column and in the sediment within the sub-domain (if MUSTANG is activated),
  - net cumulated input fluxes into the sub-domain due to bedload transport (if MUSTANG and bedload is activated).
- For open boundaries, this functionnality computes:
  - net cumulated fluxes through boundaries (since the start date of massbalance computation), sign depending on the definition of each segment of the boundary.

Boundaries and sub-domains are defined for all water column (from the surface water to the bottom).

Submassbalance results are writen in a separate netcdf file (path defined in *nmlsubmassbalance: parameter submassbalance_output_file*)

Each border can be retrieve by its name in border_name variable and each substance can be retrieve by its name in tracer_name variable. Units correspond to the unit of the substance, for example kg for sediment substance.

"border_*" variables correspond to opened borders results and mask.

"budget_*" variables correspond to closed domains results and mask.

Mask variables allow the user to check the border definition :

- border_mask_N and budget_mask_N: equals 1 if the mesh is a South boundary ; -1 if the mesh is a North boundary and 0 otherwise
- border_mask_E and budget_mask_E: equals 1 if the mesh is a West boundary ; -1 if the mesh is an East boundary and 0 otherwise
- budget_mask: equals 1 in the sub-domain and 0 out of the sub-domain

Header of an example submassbalance output file:

```
dimensions:
    xi_rho = 821 ;
```

(continues on next page)

(continued from previous page)

```
    eta_rho = 623 ;
    lchain = 200 ;
    time = UNLIMITED ; // (8748 currently)
    border = 89 ;
    budget = 3 ;
    tracer = 3 ;
variables:
    double xi_rho(xi_rho) ;
        xi_rho:units = "index x axis" ;
    double eta_rho(eta_rho) ;
        eta_rho:units = "index y axis" ;
    double time(time) ;
        time:units = "seconds since 1900-01-01" ;
    double lon_rho(eta_rho, xi_rho) ;
        lon_rho:long_name = "longitude of RHO-points" ;
        lon_rho:units = "degree_east" ;
    double lat_rho(eta_rho, xi_rho) ;
        lat_rho:long_name = "latitude of RHO-points" ;
        lat_rho:units = "degree_north" ;
    double border(border) ;
    char border_name(border, lchain) ;
    int border_mask_N(eta_rho, xi_rho, border) ;
    int border_mask_E(eta_rho, xi_rho, border) ;
    double budget(budget) ;
    char budget_name(budget, lchain) ;
    int budget_mask_N(eta_rho, xi_rho, budget) ;
    int budget_mask_E(eta_rho, xi_rho, budget) ;
    int budget_mask(eta_rho, xi_rho, budget) ;
    double tracer(tracer) ;
    char tracer_name(tracer, lchain) ;
    double border_flux(time, tracer, border) ;
        border_flux:description = "FLux through line" ;
    double budget_total(time, tracer, budget) ;
        budget_total:description = "Global budget (should be constant if conservative␣
↪var)" ;
    double budget_stwat(time, tracer, budget) ;
        budget_stwat:description = "Stock in water" ;
    double budget_stsed(time, tracer, budget) ;
        budget_stsed:description = "Stock in sediment" ;
    double budget_flux_ws(time, tracer, budget) ;
        budget_flux_ws:description = "FLux at interface water-sediment (> if from sed␣
↪to wat)" ;
    double budget_flux_obc(time, tracer, budget) ;
        budget_flux_obc:description = "FLux from zone boundaries (>if in)" ;
    double budget_flux_source(time, tracer, budget) ;
        budget_flux_source:description = "FLux from rivers" ;
```

1.13.2.2.4.2 Dredging effect

This feature has been developed to account for the impact of dredging in estuaries, channels, and harbors, where these anthropogenic activities cannot be overlooked.

During the simulation, sediment are removed from user defined areas in the domain and eventually discharged elsewhere in the domain.

To enable this feature, the user must complete the *dredging_location_file* and *dredging_settings_file* with at least one dredging zone. (See *namelist for dredging* , *dredging setting file* and *dredging netcdf location file*)

An example of a namelist is given here :

```
&namdredging
   dredging_location_file = 'dredging_areas.nc'
   dredging_settings_file = 'dredging_settings.txt'
   dredging_out_file = 'dredging_out.nc'
   dredging_dumping_layer = 1
   dredging_dt = 3600.
   dredging_dt_out = 3600.   /
```

The feature operates using designated dredging and dumping zones defined by *dredging_location_file* and *dredging_settings_file*. During the simulation, at user-defined time intervals (*dredging_dt*), MUSTANG assesses the depth of dredging areas. If the depth in certain cells is too shallow based on user defined criteria for that area, sediment is removed until the criteria is achieved. The extracted sediment is then either discharged in the designated dumping zone or excluded from the computation. MUDS and SANDS dredged sediments are released into the water column at a user-defined sigma level (*dredging_dumping_layer*), while GRAVELS are deposited directly. The sediments are evenly distributed across the dumping area. The dredging and the dumping occur instantly.

A dredging zone is characterized by:

- a name
- a location provided in a netcdf file
- a dredging level in meter with a reference to the mean sea level (and positive towards the bottom)
- a dumping zone where the dredged sediment is discharged. This is an option, the dredged sediment could also just be remove from the model domain.

A dumping zone is characterized by:

- a name
- a location provided in a netcdf file

This dredging feature can be first explored with the ESTUARY test case, see *ESTUARY test case*

1.13.2.2.4.3 Details of a dredging setting file

The dredging setting file define the depth of the dredging in each area and the associated dumping area.

It is a text file with a first line of header not read by MUSTANG. The file must contains one line by dredging area and the name must correspond to the name used in the *dredging netcdf location file*.

In the example below, 3 zones are dredged. Zone_1 and zone_3 dredged sediments are dumped in a zone called dump_1. Zone_2 dredged sediments are dumped outside of the model area.

The level of dredging is given in m/mean sea level.

```
#Dredging_area depth(m/mean sea level) Dumping_area
zone_1     20.        dump_1
zone_2     16.        none
zone_3     25.        dump_1
```

Tip

To dump outside of the model area, put *none* in the dumping area column

Note

Criteria to dredge

Sediment layers are dredged while h - (hsed - hsed_init) < dredg_depth

With:

- h: model bathymetry at initial state (m/mean sea level)
- hsed: sediment height (m)
- hsed_init: sediment_height at initial state (m). This is used to follow bedrock level even in morphostatic simulation.
- dredg_depth: depth defined in the *dredging_setting_file*

If the dredg_depth aimed is in the middle of a sediment layer, the layer is partially dredged.

1.13.2.2.4.4 Details of a dredging netcdf location file

For each dredging or dumping name specified in the settings file, a corresponding variable of the same dimensions as the model must be present in the NetCDF file, containing:

- 0 = outside the area
- 1 = inside the area

The total surface of each dumping zone is computed by summing the surface areas of all cells within the zone. These total surfaces are then used to evenly distribute the dredged sediments.

An example of a netcdf file header is given below:

```
dimensions:
  eta_rho = 92 ;
  xi_rho = 202 ;
variables:
  double zone_1(eta_rho, xi_rho) ;
  double zone_2(eta_rho, xi_rho) ;
  double zone_3(eta_rho, xi_rho) ;
  double dump_1(eta_rho, xi_rho) ;
```

Note

Overlapping dredging areas

In case of overlapping dredging areas, the dredging depths defined in the dredging setting file are used to keep the deepest dredging zone in the corresponding cells.

1.13.2.2.4.5 Dredging output file

For each dredging zone and each sediment class, MUSTANG compute the cummulative dredged mass over the simulation.

The output file specified in *dredging_out_file* namelist parameter is a netcdf file with a variable cummulative_mass of dimention the number of sediment class and the number of dredging areas.

An example of a netcdf file header is given below:

```
dimensions:
  lchain = 200 ;
  time = UNLIMITED ; // (188 currently)
  area = 3 ;
  class = 2 ;
variables:
```

(continues on next page)

(continued from previous page)

```
double time(time) ; time:units = "seconds since 1900-01-01" ;
double area(area) ;
char area_name(area, lchain) ;
double class(class) ;
char class_name(class, lchain) ;
double cummulative_mass(time, class, area)
```

1.13.2.2.5 FAQ and known issues

1.13.2.2.5.1 Coarse sediment in MUSTANG

In MUSTANG V1 (ie, when #key_MUSTANG_V2 is not defined), GRAVEL can not go in suspension, they will not move. Another way to deal with GRAVEL in V1 is to declare them as SAND. They will not travel far anyway in suspension, but at least they will impact the sediment dynamics. **If their diameter is greater than 2 mm, they will not impact the mean critical bed shear stress** (i.e. common critical bed shear stress for all sediment classes in V1).

Nevertheless, MUSTANG V2 is recommended to deal with coarse sediment.

1.13.2.2.5.2 Not yet implemented features

Bioturbation, diffusion within sediment and at interface are code but have not been tested yet.

1.13.2.2.5.3 MUSTANG V1 vs MUSTANG V2

Using the cppkey #key_MUSTANG_V2 is another way of dealing with erosion and deposition processes. A review of its implementation will be done in the next MUSTANG version to clarify it.

The main differencies are:

- porosity computation can be parametrize in V2 while in V1 it is constant
- erosion as a mixture whatever the cohesive fraction in V1 versus erosion by class under conditions in V2 (which allow to add bedload in the process)

1.13.3 OBSTRUCTIONS module : flow in presence of various obstructions

1.13.3.1 Introduction

In coastal environments, hydrodynamics is often modified by obstructions (natural or anthropogenic) such as seagrass meadows, oyster and mussels farming, salt-marsh and estuarine vegetation.

The OBSTRUCTIONS module have been implemented to take account for these modifications of hydrodynamics induced by differents kinds of obstructions. This is a generic module adapted to various types of natural and anthropogenic obstructions that can be found in coastal ecosystems (i.e. rigid/flexible, submerged/emergent, upward/downward/3D).

The module as been designed to need a minimal, optimized, number of empirical calibration parameters. Multiple obstruction types can be defined in the same grid cell.

The influence of obstruction elements on three-dimensional flow is taken into account through:

- the loss of momentum due to the drag exerted on obstruction elements
- the balance between turbulence production and dissipation introduced within the k-ϵ turbulence closure scheme

The OBSTRUCTIONS module is coupled with the hydrodynamic CROCO model.

This module was primarily designed to describe the three-dimensional hydrodynamic effects of flexible seagrass Zostera noltei on flow Kombiadou *et al.* [2014]. In its updated present state (Ganthy *et al.* [2024]), the module allows the simulation of various types of obstructions. The numerical scheme has also been modified to allow

multiple obstructions in the same grid cell. The computation procedure for flexible obstructions height has been improved.

Currently three generic obstructions types can be taken into account :

- **Upward obstructions (UP)**: obstructions starting from the seabed and erected toward the water surface (e.g. vegetation, mussel post…). These obstructions can be rigid or flexible.
- **Downward obstructions (DO)**: obstruction starting from the water surface and hanging toward the bottom (e.g. mussel/oyster lines). As for the UP type, these obstructions can be rigid or flexible.
- **Three dimensional obstruction (3D)**: specific case of upward type, describes obstructions which are in mid-water (no structures near the bed nor the surface). This specific type has been typically dedicated to represent oyster bags.

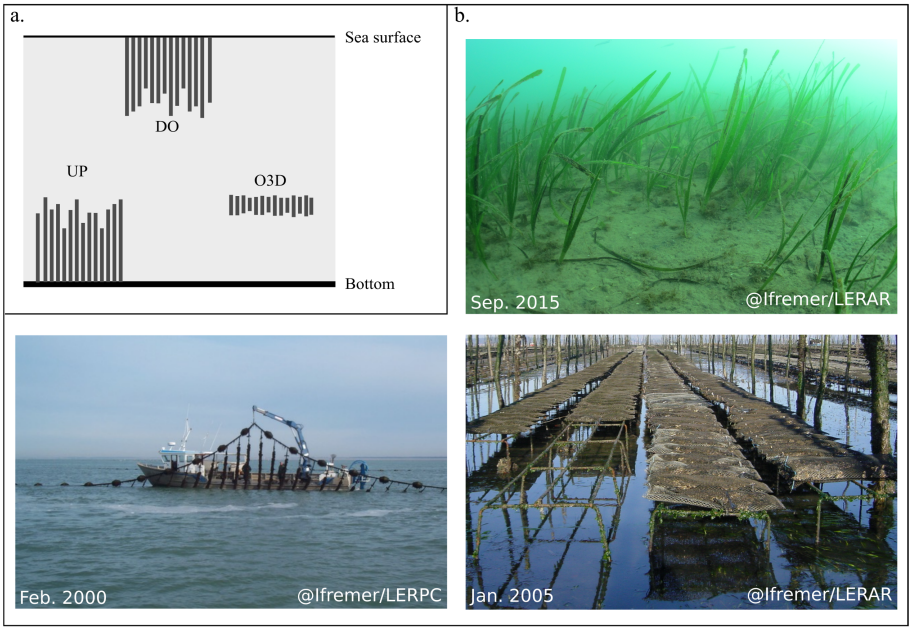


Fig. 11: Description of 3 types of obstructions

Each obstruction element can be described in two ways depending on its geometry:

- **cylinder-like** (e.g reeds or mussel rops)
- **parallelepiped-like** (e.g. seagrass leaves)

For upward or downward obstruction, obstruction can be **flexible**. This means the obstruction elements will interact with ambient current flow while bending, leading to changes in obstruction height, element density and horizontal cross-sectional area depending on their bending angle.

The main formulations (cylinder case) of the module are :

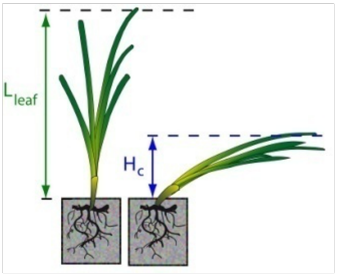
- for the drag part :

$$F_u(z) = -\frac{1}{2} \cdot C_d \cdot \rho \cdot d_0(z) \cdot n(z) \cdot u(z) \cdot \sqrt{u(z)^2 + v(z)^2} \cdot f_z(z) \cdot f_{xy}(z)$$
$$F_v(z) = -\frac{1}{2} \cdot C_d \cdot \rho \cdot d_0(z) \cdot n(z) \cdot v(z) \cdot \sqrt{u(z)^2 + v(z)^2} \cdot f_z(z) \cdot f_{xy}(z)$$

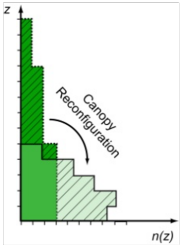
- for the turbulence part :

$$(\frac{\partial k}{\partial t})_{obstruction} = \frac{1}{1 - A(z)} \cdot \frac{\partial}{\partial z} \left\{ (1 - A(z)) \cdot \frac{\nu + \nu_t}{\sigma_k} \cdot \frac{\partial k}{\partial z} \right\} + T(z)$$
$$(\frac{\partial \epsilon}{\partial t})_{obstruction} = \frac{1}{1 - A(z)} \cdot \frac{\partial}{\partial z} \left\{ (1 - A(z)) \cdot \frac{\nu + \nu_t}{\sigma_\epsilon} \cdot \frac{\partial \epsilon}{\partial z} \right\} + T(z) \cdot \tau_\epsilon^{-1}$$

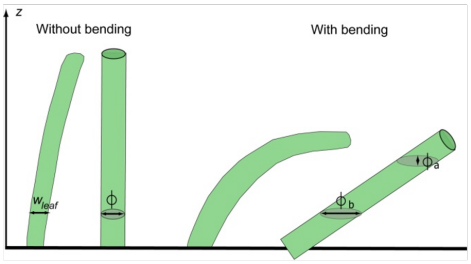
Correction of the 3 variables :



Obstruction height



Obstruction density



Obstruction apparent diameter

Fig. 12: Description of the effect of flexibility on obstructions

With :

- C_d : the drag coefficient
- d_0 : the obsruction diameter
- n : the obsruction density
- f_z : the fraction of layer effectively occupied by obstructions
- f_{xy} : the fraction of grid cell effectively occupied by obstructions
- A : the horizontal cross-sectional obstruction area per unit area
- T : a function of F_u, F_v, u, v, ρ
- τ_ϵ : a function of A, n, k, ϵ, T

This module have been developped by Florian Ganthy coupled with MARS model since 2011 (Ganthy *et al.* [2024]). It has been implemented in CROCO in 2023-2024 with an 1DV module (all computing are done within one (i,j) grid cell).

At each time step, the module execute the steps:

- Reading forcing height,density, width and thickness of obstruction (if time varying)
- Preparing hydrodynamic variables from CROCO :
  - current component u,v at the center of the cell
  - thicknesses of layers at the center of the cell
  - height of the center of each layer at the center of the cell
- Computes obstructions height and bending angle (if flexible)
- Computes vertical distribution of obstructions densities
- Computes the fraction of sigma layer occupied by obstructions
- Computes obstructions width and thickness according to bending angle (if flexible)
- Computes obstructions correction term for coverage (fragmentation) within one single grid cell
- Computes obstructions projected horizontal and vertical area
- Computes obstructions bottom roughness (for obstruction represented as macro-roughness instead of turbulent approach)
- Computes obstructions parameters used after by CROCO (drag and turbulence)
  - the sink terms for 3D friction force component at the center of the cell : F_u, F_u
  - the source terms : T, τ_ϵ for 3D turbulent dissipation at the center of the cell
  - the 3D obstruction vertical area for all obstructions A

Note

OBSTRUCTIONS module can only be used if #SOLVE3D and #GLS_KEPSILON are activated as the turbulence closure scheme in the equations is k-ϵ

1.13.3.2 Inputs files

To use the OBSTRUCTIONS module in CROCO you will need to :

- activate module OBSTRUCTIONS within CROCO environment by defining the **#OBSTRUCTION** cppkey.
- define path to the module main parameter file in CROCO input file **croco.in** as follow. In this example, the file is located in TEST_CASES directory but it could be placed in any directory.

```
obstruction: input file
            TEST_CASES/obstruction_seagrass_para.txt
```

Then you can compile and run CROCO as any other CROCO run.

1.13.3.2.1 Obstruction main parameter file

This file is the main file of obstruction module. It's where the number of obstruction and the path to specific parameters files are defined. It's also where the output of obstruction variables is defined.

The file is structured in 3 namelists :

- *obst_main* : number of obstruction and parameter for unconfined conditions
- *obst_input* : paths to each obstruction variable file and to position file
- *obst_output* : booleans to choose which variable will be outputed

Each namelist is described bellow with the description of each parameter.

&obst_main namelist:

- **obst_nbvar** : number of obstructions
- **obst_c_paramhuv** : coefficient for unconfined conditions. For flexible obstructions, when model from Abdelrhman [2007] is not used but exponential formula is used (see parameter *r_l_obst_param_height*), obstruction's height is compute using partial depth averaged velocity corresponding to unconfined canopy. This partial depth is computed from bottom (or surface depending on obstruction type) with formula: obst_c_paramhuv * height of obstruction at precedent time step. As default value, one could take obst_c_paramhuv=10.

&obst_input namelist:

- **obst_fn_position** : file name for input obstruction variables positions, see *position file*
- **obst_fn_var** : list of paths to the parameter file of each obstruction variable. For example with obst_nbvar = 2 : 'file1.txt','file2.txt'. See *variable specific file*

&obst_output namelist:

- **l_obstout_pos** : write obstructions position within output file, one 2D variable per obstruction
- **l_obstout_height_f** : write obstructions forcing height within output file, one 2D variable per obstruction
- **l_obstout_height_e** : write obstructions effective height within output file, one 2D variable per obstruction
- **l_obstout_dens_f** : write obstructions forcing density within output file, one 2D variable per obstruction
- **l_obstout_dens_e** : write obstructions effective density within output file, one 3D variable per obstruction
- **l_obstout_width_f** : write obstructions forcing width, one 2D variable per obstruction
- **l_obstout_width_e** : write obstructions effective width, one 3D variable per obstruction
- **l_obstout_thick_f** : write obstructions forcing thick, one 2D variable per obstruction
- **l_obstout_thick_e** : write obstructions effective thick, one 3D variable per obstruction
- **l_obstout_theta** : write obstructions bending angle, one 3D variable per obstruction
- **l_obstout_frac_xy** : write obstructions fragmentation correction factor, one 2D variable per obstruction
- **l_obstout_frac_z** : write obstructions fraction of sigma layer occupied, one 3D variable per obstruction
- **l_obstout_fuzvz** : write obstructions resistance force 3D, two 3D variable (vector)
- **l_obstout_a2d** : write 2D obstructions horizontal area, one 2D variable per obstruction and 3 additionnal 2D variables to distinguish "No_turb" variables, "Turb" variable and "All" variables
- **l_obstout_a3d** : write 3D obstructions horizontal area, one 3D variable per obstruction and 3 additionnal 3D variables to distinguish "No_turb" variables, "Turb" variable and "All" variables

- **l_obstout_s2d** : write 2D obstructions vertical area, one 2D variable per obstruction and 3 additionnal 2D variables to distinguish "No_turb" variables, "Turb" variable and "All" variables
- **l_obstout_s3d** : write 3D obstructions vertical area, one 3D variable per obstruction and 3 additionnal 3D variables to distinguish "No_turb" variables, "Turb" variable and "All" variables
- **l_obstout_drag** : write obstructions drag coefficient, one 3D variable per obstruction
- **l_obstout_tau** : write obstructions turbulent stress, 3D variable

If you have several type of obstructions, the number of variables could become huge. Be careful to output only the needed variables.

"Turb"("No_turb") variables correspond to the contribution of obstructions with *r_l_obst_noturb* == False (True), see *variable specific file*.

> **Note**
>
> Variables dens_e, width_e, thick_e, theta, frac_xy, frac_z, a2d, s2d, s3d and drag are not allocated if not wanted in output. If you do not need them, put False in the corresponding boolean to reduce space disk and memory needs of your simulation

1.13.3.2.2 Obstruction variable specific characteristics file

This file contains the characteristics of one type of obstruction structured in 7 namelists :

- *obst_var_main* : main variable parameters
- *obst_var_option* : variable behaviour's options
- *obst_var_init* : parameters relative to initialization
- *obst_var_flexibility* : parameters relative to the flexible obctructions
- *obst_var_roughdrag* : parameters relative to the roughness length and drag computing
- *obst_var_fracxy* : parameters relative to small-scale patchiness correction
- *obst_var_bstress* : parameters relative to the bottom shear stress

Each namelist is described bellow with the description of each parameter.

&obst_var_main namelist:

- **r_obst_varname** : Name (identifier) of the variable, for example "Seagrass"
- **r_obst_type** : choice between "UP", "DO", "3D" :
  - "UP" : if variable start from the bed
  - "DO" : if variable hang from sea-surface (down)
  - "3D" : if variable is full 3D (based on vertical density variation, see *r_l_obst_filedistri* and *variable vertical distribution file*)
- **r_l_obst_cylinder** : boolean, True if variable representation is a cylinder (if False : parallelepiped)

&obst_var_option namelist:

- **r_l_obst_flexible** : boolean, True if variable is flexible (if False : rigid)
- **r_l_obst_noturb** : boolean, True, if variable should be represented as macro-roughness (instead of turbulent approach) (only for obstruction of type "UP")
- **r_l_obst_filetimeserie** : boolean, True to use a time-series (not spatial) of obstructions characteristics. NOT AVAILABLE with r_l_obst_init_spatial = .TRUE.
- **r_obst_fn_timeserie** : Netcdf file containing temporal obstructions characteristics (if r_l_obst_filetimeserie = .TRUE.), For information about format, see *variable temporal file*

- **r_l_obst_filedistri** : To use a file describing the vertical distribution of obstruction density
- **r_obst_fn_distrib** : File containing the vertical distribution of obstruction density (if r_l_obst_filedistri = .TRUE.), see *variable vertical distribution file*

&obst_var_init namelist:

- **r_l_obst_init_spatial** : boolean, True to use spatially variable file (not temporal) of obstructions characteristics. NOT AVAILABLE with r_l_obst_filetimeserie = .TRUE.
- **r_obst_fn_initspatial** : Netcdf file containing spatial obstructions characteristics (if r_l_obst_init_spatial = .TRUE.) (variables : height,width,thick and dens) For information about format see : *initialization spatial file*
- **r_obst_i_height** : Initial height (unbent, eg. leaf-length for segrasses) of obstructions, used if not spatial initialisation (r_l_obst_init_spatial = .FALSE.)
- **r_obst_i_width** : Initial width (or diameter for cylindric obstructions) of obstructions (perpendicular to flow), used if not spatial initialisation (r_l_obst_init_spatial = .FALSE.)
- **r_obst_i_thick** : Initial thick (or diameter for cylindric obstructions) of obstructions (along the flow), used if not spatial initialisation (r_l_obst_init_spatial = .FALSE.).
- **r_obst_i_dens** : Initial density of obstructions (maximum density if using a *vertical distribution file*), used if not spatial initialisation (r_l_obst_init_spatial = .FALSE.)

> **Note**
>
> Width (r_obst_i_width) should equal thickness (r_obst_i_thick) for cylindric obstructions

If the obstruction is flexible, the obstruction posture is computed at each time step depending on current (u,v) with an update of obstruction bending depending on the folowing 3 options :

- Abdelrhman [2007] procedure,
- exponential decrease, using r_obst_c_height_x0 and r_obst_c_height_x1,
- proportional (no influence of current), using only r_obst_c_height_x0

To choose an option and the corresponding parameters, fill the &obst_var_flexibility namelist:

- **r_l_obst_abdelposture** : boolean, True to use Abdelrhman [2007] procedure to compute bending
- **r_obst_c_abdel_nmax** : Number of segments for Abdelrhman [2007] procedure
- **r_obst_c_rho** : Volumic mass of obstructions for Abdelrhman [2007] procedure
- **r_obst_c_lift** : Lift coefficient for Abdelrhman [2007] procedure
- **r_obst_c_shelter** : Sheltering coefficient A for Abdelrhman [2007] procedure
- **r_l_obst_param_height** : boolean, True to use exponential decrease formulation to compute bending (height = r_obst_c_height_x0 * height * EXP(r_obst_c_height_x1*uv), with uv the partial depth averaged velocity corresponding to unconfined canopy)
- **r_obst_c_height_x0** : First parameter for empirical formulation
- **r_obst_c_height_x1** : Second parameter for empirical formulation

&obst_var_roughdrag namelist:

- **r_l_obst_drag_cste** : To use a constant drag coefficient (r_obst_c_drag) for C_d (see *equations*) (if false, drag varies depending on the bending angle)
- **r_obst_c_drag** : Drag coefficient C_d (maximum value if not constant) for obstructions elements
- **r_obst_c_lz** : Coefficient for turbulent dissipation time-scale between obstructions elements (used to compute τ_ϵ in *equations*)

- **r_l_obst_abdelrough_cste** : To use a constant coefficient during Abdelrhman [2003] procedure used to compute obstruction macro-roughness
- **r_obst_c_crough_x0** : First coefficient for drag coefficient during Abdelrhman [2003] procedure
- **r_obst_c_crough_x1** : Second coefficient for drag coefficient during Abdelrhman [2003] procedure

If obstructions do not fill completly the cell surface, a patchiness correction can be applied using the fraction of cell occupied by obstructions (given in *position file*) and several parametrization given in &obst_var_fracxy namelist:

- **r_l_obst_fracxy** : boolean, True to take account for patchiness correction (if false, no correction is applied)
- **r_obst_fracxy_type** : if r_l_obst_fracxy is True, choose the kind of correction method :
  - 0 : patchiness correction is equal to the fraction of cell occupied by obstructions (given in *position file*)
  - 1 : patchiness correction is equal to an exponential of the fraction of cell occupied by obstructions with one coefficient (r_obst_c_fracxy_k0)
  - 2 : patchiness correction is equal to an exponential of the fraction of cell occupied by obstructions with several coefficients (r_obst_c_fracxy_k0, r_obst_c_fracxy_k1 and r_obst_c_fracxy_l)
  - 3 : patchiness correction is equal to the product of the fraction of cell occupied by obstructions and r_obst_c_fracxy_k0
- **r_obst_c_fracxy_k0** : Coefficient for the corrections type 1, 2 and 3
- **r_obst_c_fracxy_k1** : First parameter for correction of the exponential coefficient (type 2)
- **r_obst_c_fracxy_l** : Second parameter for correction of the exponential coefficient (type 2)

If a sediment model is used (here available with MUSTANG), the OBSTRUCTIONS module can be used to modify the roughness length used to compute the bottom shear stress. The corresponding parameters are in &obst_var_bstress namelist:

- **r_l_obst_z0bstress** : To activate the impact of obstruction on roughness length used to compute the bottom shear stress (only for UP type)
- **r_obst_z0bstress_option** : Option to compute the obstruction induced roughness length:
  - 0 : constant z0 (r_obst_c_z0bstress)
  - 1 : parameterization
- **r_obst_c_z0bstress** : Constant (uncorrected value of roughness length)
- **r_obst_c_z0bstress_x0** : First parameter for rouhgness length computation (in 3D)
- **r_obst_c_z0bstress_x1** : Second parameter for rouhgness length computation (in 3D)

1.13.3.2.3 Obstruction variable specific vertical distribution file

To specified 3D obstruction or a variation of obstruction density on its height, a text file can be used to specified the fraction (in %, between 0 and 100) of density to apply at a fraction of height (in %, between 0 and 100) .

Example for a density equal to 100% of the specified density through 0 to 50% of specified height and 50% above :

```
name
nb_hnorm
4
Hnorm    nnorm
0.       100
50.      100
50.0001  50
100.100  50
END OF FILE
```

The number of vertical discretization is given on line number 3. The readed lines begin at line number 5.

Example to specified a 3D obsctruction :

```
Table
nb_hnorm
4
Hnorm    nnorm
0.0000   0
87.5     0
87.5001  100.
100.100  100.
END OF FILE
```

1.13.3.2.4 Obstruction position file

This file is a netcdf file containing the fraction of cell occupied by obstructions (value from 0 to 1) on the model grid.

The dimension are the same as the grid file (eta_rho, xi_rho). The name of the variable must be pos_<obstruction name> (with obstruction name given in *obst_var_main*)

Example with an obstruction named Obstruct :

```
netcdf obstruction_seagrass_position {
dimensions:
    eta_rho = 7 ;
    time = UNLIMITED ; // (1 currently)
    xi_rho = 38 ;
variables:
    float pos_Obstruct(time, eta_rho, xi_rho) ;
        pos_Obstruct:_FillValue = NaN ;
    double time(time) ;
        time:long_name = "time since initialization" ;
        time:units = "seconds since 2019/01/01 00:00:00" ;
        time:field = "time, scalar, series" ;
        time:standard_name = "time" ;
        time:axis = "T" ;
}
```

Only the first time index is read.

1.13.3.2.5 Obstruction initialization spatial file

This file is a netcdf file containing the variables : height, density, width and thickness on the model grid.

The dimension are the same as the grid file (eta_rho, xi_rho). The name of the variable must be :

- height_f_<obstruction name>
- dens_f_<obstruction name>
- width_f_<obstruction name>
- thick_f_<obstruction name>

(with obstruction name given in *obst_var_main*)

Example with an obstruction named Obstruct :

```
ncdump -h ../ktest/SEAGRASS_initspatial/spatial.nc
netcdf spatial {
```

(continues on next page)

(continued from previous page)

```
dimensions:
    xi_rho = 38 ;
    eta_rho = 7 ;
    time = UNLIMITED ; // (1 currently)
variables:
    double time(time) ;
        time:long_name = "time since initialization" ;
        time:units = "seconds since 2019/01/01 00:00:00" ;
        time:field = "time, scalar, series" ;
        time:standard_name = "time" ;
        time:axis = "T" ;
    float height_f_Obstruct(time, eta_rho, xi_rho) ;
        height_f_Obstruct:long_name = "Obstruction forcing height for Obstruct" ;
        height_f_Obstruct:units = "m" ;
        height_f_Obstruct:field = "" ;
        height_f_Obstruct:coordinates = "time lat_rho lon_rho" ;
    float dens_f_Obstruct(time, eta_rho, xi_rho) ;
        dens_f_Obstruct:long_name = "Obstruction forcing density for Obstruct" ;
        dens_f_Obstruct:units = "m-2" ;
        dens_f_Obstruct:field = "" ;
        dens_f_Obstruct:coordinates = "time lat_rho lon_rho" ;
    float width_f_Obstruct(time, eta_rho, xi_rho) ;
        width_f_Obstruct:long_name = "Obstruction forcing width for Obstruct" ;
        width_f_Obstruct:units = "m" ;
        width_f_Obstruct:field = "" ;
        width_f_Obstruct:coordinates = "time lat_rho lon_rho" ;
    float thick_f_Obstruct(time, eta_rho, xi_rho) ;
        thick_f_Obstruct:long_name = "Obstruction forcing thickness for Obstruct" ;
        thick_f_Obstruct:units = "m" ;
        thick_f_Obstruct:field = "" ;
        thick_f_Obstruct:coordinates = "time lat_rho lon_rho" ;
}
```

Only the first time index is read.

1.13.3.2.6 Obstruction temporal file

This file is a netcdf file containing the variables : height, density, width and thickness.

The name of the variable must be :

- height_f_<obstruction name>
- dens_f_<obstruction name>
- width_f_<obstruction name>
- thick_f_<obstruction name>

(with obstruction name given in *obst_var_main*)

The only axis heare is time. The height, density, width and thickness values are applied where the fraction of cell occupied by obstructions is greater than 0 (given in *position file*).

Example with an obstruction named Obstruct :

```
netcdf timeserie2 {
dimensions:
    time = UNLIMITED ; // (741 currently)
variables:
```

(continues on next page)

(continued from previous page)

```
    float dens_f_Obstruct(time) ;
        dens_f_Obstruct:long_name = "Obstruction forcing density for Obstruct" ;
        dens_f_Obstruct:units = "m-2" ;
        dens_f_Obstruct:field = "" ;
        dens_f_Obstruct:coordinates = "time lat_rho lon_rho" ;
        dens_f_Obstruct:cell_methods = "eta_rho, xi_rho: mean" ;
    float height_f_Obstruct(time) ;
        height_f_Obstruct:long_name = "Obstruction forcing height for Obstruct" ;
        height_f_Obstruct:units = "m" ;
        height_f_Obstruct:field = "" ;
        height_f_Obstruct:coordinates = "time lat_rho lon_rho" ;
        height_f_Obstruct:cell_methods = "eta_rho, xi_rho: mean" ;
    float thick_f_Obstruct(time) ;
        thick_f_Obstruct:long_name = "Obstruction forcing thickness for Obstruct" ;
        thick_f_Obstruct:units = "m" ;
        thick_f_Obstruct:field = "" ;
        thick_f_Obstruct:coordinates = "time lat_rho lon_rho" ;
        thick_f_Obstruct:cell_methods = "eta_rho, xi_rho: mean" ;
    double time(time) ;
        time:long_name = "time since initialization" ;
        time:units = "seconds since 2019/01/01 00:00:00" ;
        time:field = "time, scalar, series" ;
        time:standard_name = "time" ;
        time:axis = "T" ;
    float width_f_Obstruct(time) ;
        width_f_Obstruct:long_name = "Obstruction forcing width for Obstruct" ;
        width_f_Obstruct:units = "m" ;
        width_f_Obstruct:field = "" ;
        width_f_Obstruct:coordinates = "time lat_rho lon_rho" ;
        width_f_Obstruct:cell_methods = "eta_rho, xi_rho: mean" ;
}
```

1.13.3.3 Outputs

1.13.3.3.1 Using CROCO output file

To select whether a variable is written to the output file, the boolean in namelist *&obst_output* has to be filled in.

> **Note**
>
> Variables dens_e, width_e, thick_e, theta, frac_xy, frac_z, a2d, s2d, s3d and drag are not allocated if not wanted in output. If you do not need them, put False in the corresponding boolean to reduce space disk and memory needs of your simulation

1.13.3.3.2 Using XIOS

XIOS can be used to output the same variables as in CROCO output file.

Example of a .xml field file for a case with one obstruction called "Obstruct". The id of each field has to be coherent with the given name of obstruction.

```
<field_group id="rho" grid_ref="rho_2D">
    <field id="pos_Obstruct"
    long_name="Obstruction occupation rate for Obstruct"
    unit="-" grid_ref="rho_2D" />
    <field id="height_f_Obstruct"
```

(continues on next page)

(continued from previous page)

```
    long_name="Obstruction forcing height for Obstruct"
    unit="m" grid_ref="rho_2D" />
    <field id="height_e_Obstruct"
    long_name="Obstruction effective height for Obstruct"
    unit="m" grid_ref="rho_2D" />
    <field id="dens_f_Obstruct"
    long_name="Obstruction forcing density for Obstruct"
    unit="m-2" grid_ref="rho_2D" />
    <field id="dens_e_Obstruct"
    long_name="Obstruction effective density for Obstruct"
    unit="m-2" grid_ref="rho_3D" />
    <field id="width_f_Obstruct"
    long_name="Obstruction forcing width for Obstruct"
    unit="m" grid_ref="rho_2D" />
    <field id="width_e_Obstruct"
    long_name="Obstruction effective width for Obstruct"
    unit="m" grid_ref="rho_3D" />
    <field id="thick_f_Obstruct"
    long_name="Obstruction forcing thickness for Obstruct"
    unit="m" grid_ref="rho_2D" />
    <field id="thick_e_Obstruct"
    long_name="Obstruction effective thickness for Obstruct"
    unit="m" grid_ref="rho_3D" />
    <field id="theta_Obstruct"
    long_name="Obstruction bending angle for Obstruct"
    unit="deg" grid_ref="rho_3D" />
    <field id="frac_xy_Obstruct"
    long_name="Obstruction fragmentation correction factor for Obstruct"
    unit="-" grid_ref="rho_2D" />
    <field id="frac_z_Obstruct"
    long_name="Obstruction sigma fraction for Obstruct"
    unit="-" grid_ref="rho_2D" />
    <field id="cd3d_Obstruct"
    long_name="Obstruction drag coefficient for Obstruct"
    unit="-" grid_ref="rho_3D" />
    <field id="a2d_Obstruct"
    long_name="2D Obstruction horizontal area for Obstruct"
    unit="-" grid_ref="rho_2D" />
    <field id="a3d_Obstruct"
    long_name="3D Obstruction horizontal area for Obstruct"
    unit="-" grid_ref="rho_3D" />
    <field id="s2d_Obstruct"
    long_name="2D Obstruction vertical area for Obstruct"
    unit="-" grid_ref="rho_2D" />
    <field id="s3d_Obstruct"
    long_name="3D Obstruction vertical area for Obstruct"
    unit="-" grid_ref="rho_3D" />

    <field id="a2d_NoTurb"
    long_name="2D Obstruction horizontal area for NoTurb variables"
    unit="-" grid_ref="rho_2D" />
    <field id="a2d_Turb"
    long_name="2D Obstruction horizontal area for Turb variables"
    unit="-" grid_ref="rho_2D" />
    <field id="a2d_All"
    long_name="2D Obstruction horizontal area for All variables"
```

(continues on next page)

(continued from previous page)

```
    unit="-" grid_ref="rho_2D" />
    <field id="a3d_NoTurb"
    long_name="3D Obstruction horizontal area for NoTurb variables"
    unit="-" grid_ref="rho_3D" />
    <field id="a3d_Turb"
    long_name="3D Obstruction horizontal area for Turb variables"
    unit="-" grid_ref="rho_3D" />
    <field id="a3d_All"
    long_name="3D Obstruction horizontal area for All variables"
    unit="-" grid_ref="rho_3D" />
    <field id="s2d_NoTurb"
    long_name="2D Obstruction vertical area for NoTurb variables"
    unit="-" grid_ref="rho_2D" />
    <field id="s2d_Turb"
    long_name="2D Obstruction vertical area for Turb variables"
    unit="-" grid_ref="rho_2D" />
    <field id="s2d_All"
    long_name="2D Obstruction vertical area for All variables"
    unit="-" grid_ref="rho_2D" />
    <field id="s3d_NoTurb"
    long_name="3D Obstruction vertical area for NoTurb variables"
    unit="-" grid_ref="rho_3D" />
    <field id="s3d_Turb"
    long_name="3D Obstruction vertical area for Turb variables"
    unit="-" grid_ref="rho_3D" />
    <field id="s3d_All"
    long_name="3D Obstruction vertical area for All variables"
    unit="-" grid_ref="rho_3D" />

    <field id="fuzvz_uz"
    long_name="Obstruction 3D friction force FUZ"
    unit="N.m-2" grid_ref="rho_3D" />
    <field id="fuzvz_vz"
    long_name="Obstruction 3D friction force FVZ"
    unit="N.m-2" grid_ref="rho_3D" />
    <field id="tau3d"
    long_name="Obstruction 3D turbulent dissipation"
    unit="N.m-2" grid_ref="rho_3D" />

</field_group>
```

1.13.3.4 Example

A test case is provided with cppkey #SEAGRASS. See *SEAGRASS* for more informations.

1.13.4 Biogeochemical models

CROCO comes with series of biogeochemical (BGC) models of increasing complexity, from relatively simple 5- or 7-component NPZD [Gruber *et al.*, 2006, Gruber *et al.*, 2011] and N2P2Z2D2 BioEBUS model [Gutknecht *et al.*, 2013] that proved well suited to upwelling regions to 24-component PISCES [Aumont *et al.*, 2005].

BioEBUS is a nitrogen-based model (Fig. 1) derived from a N2P2Z2D2 evolution of ROMS NPZD model [Gruber *et al.*, 2006, Gruber *et al.*, 2011] and accounting for the main planktonic communities in upwelling ecosystems associated oxygen minimum zones (OMZs). It is validated in Gutknecht *et al.* [2013] using available satellite and in situ data in the northern part of the Benguela upwelling system. In this model, phytoplankton and zooplankton are split into small (PS and ZS: flagellates and ciliates, respectively) and large (PL and ZL: diatoms and copepods, respectively) organisms. Detritus are also separated into small and large particulate compartments (DS and DL). A

semi-labile dissolved organic nitrogen (DON) compartment was added since DON can be an important reservoir of OM and can potentially play an important role in supplying nitrogen or carbon from the coastal region to the open ocean [Huret *et al.*, 2005]. The pool of dissolved inorganic nitrogen is split into nitrate (NO3-), nitrite (NO2-) and ammonium (NH4+) species to have a detailed description of the microbial loop: ammonification/nitrification processes under oxic conditions, and denitrification/anammox processes under suboxic conditions [Yakushev *et al.*, 2007]. These processes are directly oxygen dependent, so an oxygen (O2) equation was also introduced in BioEBUS with the source term (photosynthesis), sink terms (zooplankton respiration, bacteria re-mineralisation) and sea–air O2 fluxes following Coba De La Peña *et al.* [2010] and Yakushev *et al.* [2007]. To complete this nitrogen-based model, nitrous oxide (N2O) was introduced using the parameterization of Suntharalingam *et al.* [2000], Suntharalingam *et al.* [2012]. It allows determining the N2O production under oxygenated conditions and at low-oxygen levels, mimicking the N2O production from nitrification and denitrification processes. The SMS terms of BioEBUS and parameter values are described in detail in Gutknecht *et al.* [2013].

PISCES was developed for NEMO (the French ocean climate model). It was implemented in CROCO for its supposed suitability for a wide range of oceanic regimes. PISCES currently has five modeled limiting nutrients for phytoplankton growth: Nitrate and Ammonium, Phosphate, Silicate and Iron. Phosphate and nitrate+ammonium are linked by constant Redfield ratios but the nitrogen pool undergoes nitrogen fixation and denitrification. Four living compartments are represented: two phytoplankton size-classes/groups corresponding to nanophytoplankton and diatoms, and two zooplankton size classes which are micro-zooplankton and mesozooplankton. For phytoplankton, prognostic variables are total biomass, the iron, chlorophyll and silicon contents. This means that the Fe/C, Chl/C and Si/C ratios of both phytoplankton groups are fully predicted by the model. For zooplankton, only the total biomass is modeled. For all species, the C/N/P/O2 ratios are supposed constant and are not allowed to vary. The Redfield ratio O/C/N/P is set to 172/122/16/1. In addition, the Fe/C ratio of both zooplankton groups is kept constant. No silicified zooplankton is assumed. The bacterial pool is not yet explicitly modeled. There are three non-living compartments: semi-labile dissolved organic matter, small and big sinking particles. The iron, silicon and calcite pools of the particles are explicitly modeled and their ratios are allowed to vary. The sinking speed of the particles is not altered by their content in calcite and biogenic silicate ("The ballast effect"). The latter particles are assumed to sink at the same speed as big organic matter particles. All the non-living compartments experience aggregation due to turbulence and differential settling. In addition to the ecosystem model, PISCES also simulates dissolved inorganic carbon, total alkalinity and dissolved oxygen. The latter tracer is also used to define the regions where oxic or anoxic remineralization takes place. See Aumont *et al.* [2005] in the documentation section for details.

Related CPP options:

PISCES	Activate 24-component PISCES biogeochemical model
BIO_NChlPZD	Activate 5-component NPZD type model
BIO_N2PZD2	Activate 7-component NPZD type model
BIO_BioEBUS	Activate 12-component NPZD type model

Preselected options:

```
# ifdef BIOLOGY
# undef PISCES
# define BIO_NChlPZD
# undef BIO_N2ChlPZD2
# undef BIO_BioEBUS
# endif
```

1.13.5 Lagrangian floats

1.14 Coupling CROCO with other models

CROCO is coupled to atmospheric and wave models through the OASIS-MCT (Ocean-Atmosphere-Sea-Ice-Soil, Model Coupling Toolkit) coupler developed by CERFACS (Toulouse, France). This coupler allows the atmospheric, oceanic, and wave models to run at the s ame time in **parallel**, it **exchanges** variables, and performs **grid interpolations** and **time transformations** if requested. OASIS is not an executable file, but a set of libraries pro-

viding functions which are called in the models themselves. The variables exchanged by the coupler, as well as the grid interpolations are specified through a namelist file (called `namcouple`).

CROCO can therefore be coupled to any code in which OASIS-MCT is implemented. Non-exhaustively, here are some models including OASIS-MCT, that can be coupled to CROCO:

- WRF (Weather Research and Forecast model developed at NCAR, Boulder, USA)
- Meso-NH (Mesoscale Non-Hydrostatic model developed at Laboratoire d'Aérologie, Toulouse, France)
- WW3 (WaveWatch III model developed at NCEP, USA, and Ifremer, France)
- ...

Those model are not provided for download with CROCO and need to be installed separately, as well as OASIS-MCT library.

A description of the OASIS-MCT features, its implementation in CROCO, WW3 and WRF codes, and the coupled variables that can be exchanged are given in the following.

Datailed step by step coupled tutorial is also available in the **Tutorials** section.

1.14.1 OASIS philosophy

1.14.1.1 OASIS libraries

OASIS-MCT libraries are:

- **psmile** for coupling
- **mct** (Argonne National Laboratory) for parallel exchanges
- **scrip** (Los Alamos National Laboratory) for interpolations

Functions provided by the OASIS-MCT framework are:

> **Note**
>
> `oasis_` / `prism_` are new / old names for backward compatibility, both useable

- Initialization and creation of a local communicator for internal parallel computation in each model:
  - oasis_init_comp / prism_init_comp_proto
  - oasis_get_localcomm / prism_get_localcomm_proto
- Grid data definition for exchanges and interpolations:
  - oasis_write_grid
  - oasis_write_corner
  - oasis_write_area
  - oasis_write_mask
  - oasis_terminate_grids_writing
  - Partition and exchanged variables definition:
    - oasis_def_partition / prism_def_partition_proto
    - oasis_def_var / prism_def_var_proto
    - oasis_enddef / prism_enddef_proto
  - Exchange of coupling fields:
    - oasis_get / prism_get_proto
    - oasis_put / prism_put_proto

- Finalization:
  - oasis_terminate / prism_terminate_proto

These OASIS3-MCT intrinsic functions are called in each model involved in the coupling. **Initialization** phase, **Definition** phase, and **Finalization** phase are called only once in each simulation while **Exchange** phase is called every time step. The effective exchanges are done only at specified times, defined by the coupling frequency, although the **Exchange** phase is called every model time step. The coupling frequency is controlled through the OASIS3-MCT namcouple.

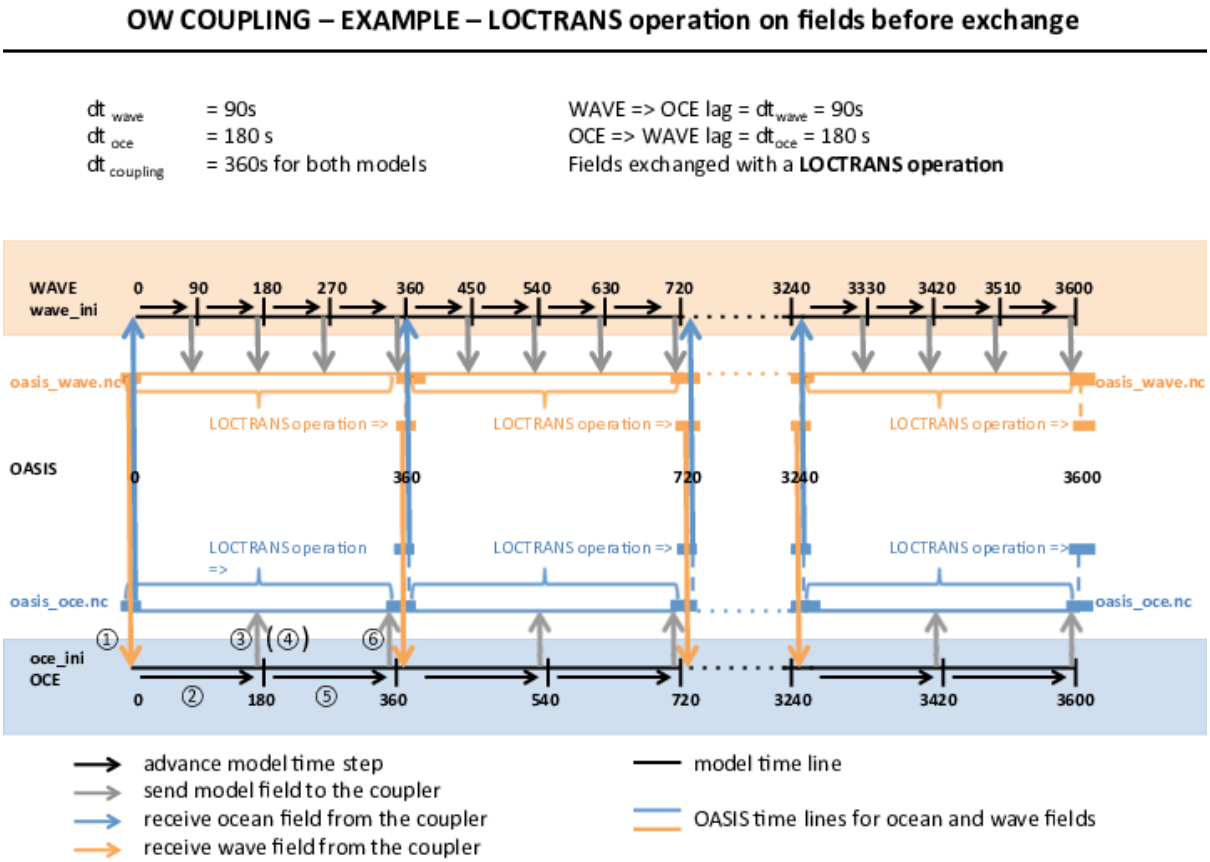
1.14.1.2 Coupling sequence

The frequency of exchanges between two models is defined by the **coupling time step**.

The **coupling time step** must be a multiple of the models time steps. An example of coupling sequence is pictured in the following Figure. In this example, the coupling time step is defined at 360s for both models. The wave model time step is 90s, so it will exchange every 4 time steps. The ocean model time step is 180s, so it will exchange every 2 time steps.

Another coupling parameter defined in the namcouple is the **lag**. It is used by the OASIS coupler to synchronize the `send` and `receive` functions. The lag must be defined for each model at the same value than its own time step. For instance:

- WAVE to OCEAN `lag = dt wave = 90`
- OCEAN to WAVE `lag = dt ocean = 180`



Therefore, `receive` and `send` functions have to be set at the same time in the model codes. OASIS will send the fields at the appropriate time thanks to the lag defined in the namcouple.

The coupling sequence in each model is:

initialization	`oasis_time = 0`
reception of coupled fields	`rcv(oasis_time)`
model time stepping	computation `t -> t+dt`
sending of coupled fields	`snd(oasis_time)`
increment of coupling time	`oasis_time = oasis_time + dt`

OASIS will exchange fields (get/put) if the time corresponds to a coupling time step, *e.g.* if:

- `oasis_time` corresponds to a coupling time step for get
- `oasis_time + lag` corresponds to a coupling time step for put

IN THE MODEL	IN OASIS
receive (date)	get(date)
send(date)	put(date+lag)

OASIS is also able to store fields from a model if a time transformation is requested in the namcouple (keyword `LOCTRANS + type of transformation`, see next section). OASIS will store the fields until a coupling time step is reached, then it will apply the time transformation, interpolate spatially the field as specified in the namcouple, and exchange the field with the other model.

1.14.1.3 Restart files

As reception of coupled fields is called before model computation, you need to create **restart files** for the coupler containing initial or restart fields for the first time step.

These **restart files** are for OASIS, and therefore need to have variable names corresponding to OASIS `namcouple` coupled fields. The initial files for OASIS are named `oasis_oce.nc` and `oasis_wave.nc` in the example pirctured in the above Figure. `oce_ini` and `wave_ini` are not related to OASIS, they are usual initialization or restart files from your oceanic and wave model; *e.g.* in CROCO, `oce_ini` is `croco_ini.nc`, and in WW3, `wave_ini` is `restart.ww3`).

Summary of the restart files:

- `oasis_oce.nc`, `oasis_wave.nc`: restart files for OASIS, you need to create them at the beginning of the run, OASIS will overwrite them at the end of the run, and they will be available for next restart
- `oce_ini`, `wave_ini`: correspond to `croco_ini.nc`, `restart.ww3`. These are your ocean and wave model initial or restart files

Practical example of the coupling sequence pictured in the above Figure:

```
oasis_time = 0
#1 => get field from oasis_wave.nc
rcv(0) => in oasis: get(0)
#2 => timestepping
t = 0+dt = 0+180 = 180
#3 => 180 is not a coupling time step, do nothing
snd(0) => in oasis: put(0+lag) = put(0+180) = put(180)
oasis_time = oasis_time+dt = 0+180 = 180
#4 => 180 is not a coupling time step, do nothing
rcv(180) => in oasis: get(180)
#5 => timestepping
t = 180+dt = 180+180 = 360
#6 => 360 is a coupling time step, put field
snd(180) => in oasis: put(180+lag) = put(180+180) = put(360)
```

1.14.1.4 Interpolations

The OASIS3-MCT coupler can process time transformations and 2D spatial interpolations of the exchanged fields. The 2D spatial interpolation, requested if models have different grids, is performed by the **scrip** library using `SCRIPR` keyword in the `namcouple`. Available interpolation types are:

`BILINEAR`	interpolation based on a local bilinear approximation
`BICUBIC`	interpolation based on a local bicubic approximation
`CONSERV`	1st or 2nd order conservative remapping
`DISTWGT`	distance weighted nearest-neighbour interpolation (N neighbours)
`GAUSWGT`	N nearest-neighbour interpolation weighted by their distance and a gaussian function

See OASIS manual for detailed informations.

Time transformations can also be performed by OASIS using `LOCTRANS` keywork in the `namcouple`. Available transformations are:

`INSTANT`	no time transformation, the instantaneous field is transfered
`ACCUMUL`	the field accumulated over the previous coupling period is exchanged
`AVERAGE`	the field averaged over the previous coupling period is transfered
`T_MIN`	the minimum value of the field for each source grid point over the previous coupling period is transfered
`T_MAX`	the maximum value of the field for each source grid point over the previous coupling period is transfered

1.14.2 Detailed OASIS implementation

1.14.2.1 In CROCO

The following routines are specifically built for coupling with OASIS and contain calls to OASIS intrinsic functions:

- `cpl_prism_init.F`: Manage the initialization phase of OASIS3-MCT : local MPI communicator
- `cpl_prism_define.F`: Manage the definition phase of OASIS3-MCT: domain partition, name of exchanged fields as read in the namcouple
- `cpl_prism_grid.F`: Manage the definition of grids for the coupler
- `cpl_prism_put.F`: Manage the sending of arrays from CROCO to the OASIS3-MCT coupler
- `cpl_prism_getvar.F`: Manage the generic reception from OASIS3-MCT.
- `cpl_prism_get.F`: Manage the specificity of each received variable: C-grid position, and field unit transformations

These routines are called in the the code in:

- `main.F`: Initialization, and finalization phases
- `get_initial.F`: Definition phase
- `zoom.F`: Initialization phase for AGRIF nested simulations
- `step.F`: Exchanges (sending and reception) of coupling variables

Other CROCO routines have also been slightly modified to introduce coupling:

- `testkeys.F`: To enable automatic linking to OASIS3-MCT libraries during compilation with jobcomp
- `cppdefs.h`: Definition of the `OA_COUPLING` and `OW_COUPLING` cpp-keys, and the other related and requested cpp-keys, as `MPI`
- `set_global_definitions.h`: Definition of cpp-keys in case of coupling (`undef OPENMP`, `define MPI`, `define MPI_COMM_WORLD ocean_grid_comm`: MPI_COMM_WORLD generic MPI communicator is redefined as the local MPI communicator ocean_grid_comm, `undef BULK_FLUX`: no bulk OA parametrization)
- `mpi_roms.h`: Newly added to define variables related to OASIS3-MCT operations. It manage the MPI communicator, using either the generic MPI_COMM_WORLD, either the local MPI communicator created by OASIS3-MCT
- `read_inp.F`: Not reading atmospheric forcing files (`croco_frc.nc` and/or `croco_blk.nc`) in OA coupled mode

A schematic picture of the calls in CROCO is (with # name.F indicating the routine we enter in):

```
# main.F
if !defined AGRIF
call cpl_prism_init
else
call Agrif_MPI_Init
endif
...
call read_inp
...
call_get_initial
       # get_initial.F
       ...
       call cpl_prism_define
               # cpl_prism_define.F
               call prism_def_partition_proto
               call cpl_prism_grid
               call prism_def_var_proto
               call prism_enddef_proto
       oasis_time=0
# main.F
...
DO 1:NT
call step
       # step.F
       if ( (iif==-1).and.(oasis_time>=0).and.(nbstep3d<ntimes) ) then
               call cpl_prism_get(oasis_time)
                       # cpl_prism_get.F
                       call cpl_prism_getvar
       endif
       call prestep3d
               call get_vbc
               ...
```

(continues on next page)

(continued from previous page)

```
        call step2d
        ...
        call step3d_uv
        call step3d_t
        iif = -1
        nbstep3d = nbstep3d + 1
        if (iif==-1) then
                if (oasis_time>=0.and.(nbstep3d<ntimes)) then
                        call cpl_prism_put (oasis_time)
                        oasis_time = oasis_time + dt
                endif
        endif
# main.F
END DO
...
call prism_terminate_proto
...
```

1.14.2.2 In WW3

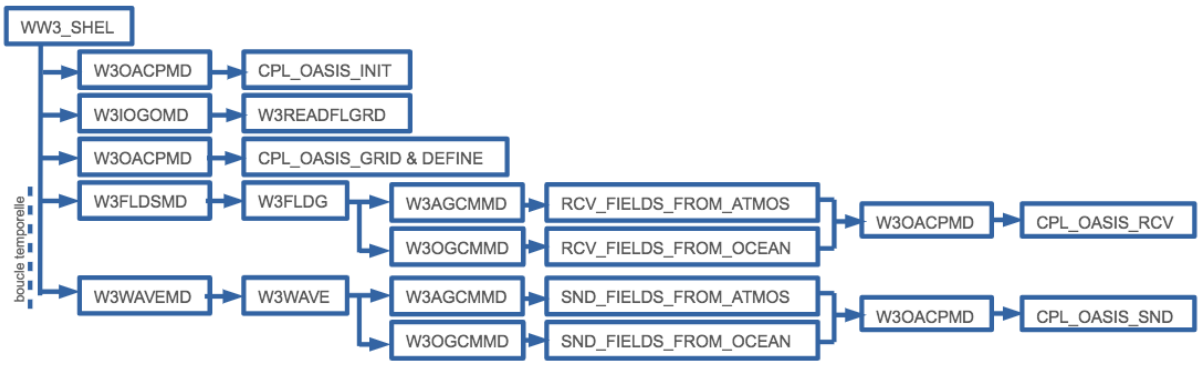
The following routines have been specifically built for coupling with OASIS:

- `w3oacpmd.ftn`: main coupling module with calls to oasis intrinsic functions
- `w3agcmmd.ftn`: module for coupling with an atmospheric model
- `w3ogcmmd.ftn`: module for coupling with an ocean model

The following routines have been modified for coupling with OASIS:

- `w3fldsmd.ftn`: routine that manage input fields, and therefore received fields from the coupler
- `w3wdatmd.ftn`: routine that manage data structure for wave model, and therefore time for coupling
- `w3wavemd.ftn`: actual wave model, here is located the sending of coupled variables
- `ww3_shel.ftn`: main routine managing the wave model, definition/initialisation/partition phases are located here

A schematic picture of the calls in WW3 is given here:



1.14.2.3 In WRF

The routines specifically built for coupling are:

- `module_cpl_oasis3.F`
- `module_cpl.F`

Implementation of coupling with the ocean implies modifications in the following routines:

- phys/module_bl_mynn.F
- phys/module_bl_ysu.F
- phys/module_pbl_driver.F
- phys/module_surface_driver.F
- phys/module_sf_sfclay.F
- phys/module_sf_sfclayrev.F

Implementation of coupling with waves implies modifications in the following routines:

- Regristry/Registry.EM_COMMON: CHA_COEF added
- dyn/module_first_rk_step_part1.F: CHA_COEF=grid%cha_coef declaration added
- frame/module_cpl.F: rcv CHA_COEF added
- phys/module_sf_sfclay.F and ..._sfclayrev.F: introduction of wave coupled case: isftcflx=5 as follows:

```
! SJ: change charnock coefficient as a function of waves, and hence roughness
! length
  IF ( ISFTCFLX.EQ.5 ) THEN
    ZNT(I)=CHA_COEF(I)*UST(I)*UST(I)/G+0.11*1.5E-5/UST(I)
  ENDIF
```

- phys/module_surface_driver.F: CHA_COEF added in calls to sfclay and sfclayrev and "CALL cpl_rcv" for CHA_COEF

Schematic picture of WRF architecture and calls to the coupling dependencies:

```
# main/wrf.F
CALL wrf_init

   # main/module_wrf_top.F
   CALL wrf_dm_initialize

      # frame/module_dm.F
      CALL cpl_init( mpi_comm_here )
      CALL cpl_abort( 'wrf_abort', 'look for abort message in rsl* files' )

   CALL cpl_defdomain( head_grid )

# main/wrf.F
CALL wrf_run

   # main/module_wrf_top.F
   CALL integrate ( head_grid )

      # frame/module_integrate.F
      CALL cpl_defdomain( new_nest )
      CALL solve_interface ( grid_ptr )

         # share/solve_interface.F
         CALL solve_em ( grid , config_flags ... )

            # dyn_em/solve_em.F
            curr_secs2  # time for the coupler
            CALL cpl_store_input( grid, config_flags )
            CALL cpl_settime( curr_secs2 )
```

(continues on next page)

(continued from previous page)

```
            CALL first_rk_step_part1

               # dyn_em/module_first_rk_step_part1.F
               CALL surface_driver( ... )

                  # phys/module_surface_driver.F
                  CALL cpl_rcv( id, ... )
                  u_phytmp(i,kts,j)=u_phytmp(i,kts,j)-uoce(i,j)
                  v_phytmp(i,kts,j)=v_phytmp(i,kts,j)-voce(i,j)

                  CALL SFCLAY( ... cha_coef ...)
                     # phys/module_sf_sfclay.F
                     CALL SFCLAY1D
                     IF ( ISFTCFLX.EQ.5 ) THEN
                       ZNT(I)=CHA_COEF(I)*UST(I)*UST(I)/G+0.11*1.5E-5/UST(I)
                     ENDIF

                  CALL SFCLAYREV( ... cha_coef ...)
                     # phys/module\_sf\_sfclayrev.F
                     CALL SFCLAYREV1D
                     IF ( ISFTCFLX.EQ.5 ) THEN
                       ZNT(I)=CHA_COEF(I)*UST(I)*UST(I)/G+0.11*1.5E-5/UST(I)
                     ENDIF

               # dyn_em/module_first_rk_step_part1.F
               CALL pbl_driver( ... )

                  # phys/module_pbl_driver.F
                  CALL ysu( ... uoce,voce, ... )
                     # module_bl_ysu.F
                     call ysu2d ( ... uox,vox, ...)
                     wspd1(i) = sqrt( (ux(i,1)-uox(i))*(ux(i,1)-uox(i))
                                    + (vx(i,1)-vox(i))*(vx(i,1)-vox(i)) )+1.e-9
                     f1(i,1) = ux(i,1)+uox(i)*ust(i)**2*g/del(i,1)*dt2/wspd1(i)
                     f2(i,1) = vx(i,1)+vox(i)*ust(i)**2*g/del(i,1)*dt2/wspd1(i)

                  CALL mynn_bl_driver( ... uoce,voce, ... )
                     # module_bl_mynn.F
                     d(1)=u(k)+dtz(k)*uoce*ust**2/wspd
                     d(1)=v(k)+dtz(k)*voce*ust**2/wspd

            # dyn_em/solve_em.F
            CALL first_rk_step_part2

      # frame/module_integrate.F
      CALL cpl_snd( grid_ptr )

   # Check where this routine is called...
   # frame/module_io_quilt.F  # for IO server (used with namelist variable: nio_tasks_
↪per_group
   CALL cpl_set_dm_communicator( mpi_comm_local )
   CALL cpl_finalize()

# main/wrf.F
CALL wrf_finalize
    #main/module_wrf_top.F
```

(continues on next page)

(continued from previous page)

```
    CALL cpl_finalize()
```

1.14.3 Coupled variables

1.14.3.1 Coupling with an atmospheric model

When coupling CROCO to an atmospheric model, to have a consistent interface, you should use momentum and heat fluxes computed from the atmospheric model bulk formula.

No surface forcing file is required (only boundary forcing, and eventually tidal forcing).

The following cpp-keys have to be set:

```
# define OA_COUPLING
# define MPI

# undef  BULK_FLUX
# undef  SMFLUX_CFB
```

Note

`SMFLUX_CFB` is a cpp-key to use a wind stress relative to the current in forced mode. In coupled mode, as current is sent to the atmosphere, the wind stress from the atmospheric model account for such a current feedback.

Fields sent by CROCO Name (units)	name and eventual oper. in the model	OASIS name
SST (K)	t(:,:,N,nnew,itemp) + 273.15	CROCO_SST
U-component of current (m/s)	u (at rho points): 0.5*(u(1:Lmmpi,1:Mmmpi,N,nnew) +u(2:Lmmpi+1,1:Mmmpi,N,nnew))	CROCO_UOCE
V-component of current (m/s)	v (at rho points): 0.5*(v(1:Lmmpi,1:Mmmpi,N,nnew) +v(1:Lmmpi,2:Mmmpi+1,N,nnew))	CROCO_VOCE
Eastward component of current (m/s)	u (at rho points) rotated eastwards (useful for rotated grids) (0.5 * (u(1:Lmmpi ,1:Mmmpi,N,nnew) + u(2:Lmmpi+1,1:Mmmpi,N,nnew))) * cos(angler(1:Lmmpi ,1:Mmmpi)) - (0.5 * (v(1:Lmmpi,1:Mmmpi ,N,nnew) + v(1:Lmmpi,2:Mmmpi+1,N,nnew))) * sin(angler(1:Lmmpi ,1:Mmmpi)) +u(2:Lmmpi+1,1:Mmmpi,N,nnew))	CROCO_EOCE
Northward component of current (m/s)	v (at rho points) rotated northward (useful for rotated grids) (0.5 * (u(1:Lmmpi ,1:Mmmpi,N,nnew) + u(2:Lmmpi+1,1:Mmmpi,N,nnew))) * sin(angler(1:Lmmpi ,1:Mmmpi)) + (0.5 * (v(1:Lmmpi,1:Mmmpi ,N,nnew) + v(1:Lmmpi,2:Mmmpi+1,N,nnew))) * cos(angler(1:Lmmpi ,1:Mmmpi))	CROCO_NOCE

Fields received by CROCO Name (units)	name and eventual oper. in the model	OASIS name
U component of wind stress (N/m2)	sustr (at u point): 0.5*(FIELD(io-1,jo)+FIELD(io,jo))/rho0 if eastward, it is first rotated: FIELD = etau * cos(angler) + ntau * sin(angler)	CROCO_UTAW or CROCO_ETAW
V component of wind stress (N/m2)	svstr (at v point): 0.5*(FIELD(io,jo-1)+FIELD(io,jo))/rho if northward, it is first rotated: FIELD = ntau * cos(angler) - etau * sin(angler)	CROCO_VTAW or CROCO_NTAW
Wind stress module (N/m2)	smstr = FIELD / rho0	CROCO_TAUM
Surface net solar flux (W/m2)	srflx = FIELD / (rho0*Cp)	CROCO_SRFL
Surface net non-solar flux (W/m2)	stflx(:,:,itemp) = FIELD / (rho0*Cp)	CROCO_STFL
Evaporation-Precipitation (kg/m2/s)	stflx(:,:,isalt) = FIELD / 1000	CROCO_EVPR
Surface atmospheric pressure (Pa)	patm2d = FIELD	CROCO_PSFC

Fields received by WRF Name (units)	name (in the model)	OASIS name
SST (K)	SST	WRF_d01_EXT_d01_SST
U component of current (m/s)	UOCE	WRF_d01_EXT_d01_UOCE
V component of current (m/s)	VOCE	WRF_d01_EXT_d01_VOCE
Eastward component of current (m/s)	EOCE	WRF_d01_EXT_d01_EOCE
Northward component of current (m/s)	NOCE	WRF_d01_EXT_d01_NOCE

Fields sent by WRF Name (units)	name (in the model)	OASIS name
Surface net solar flux (W/m2)	GSW	WRF_d01_EXT_d01_SURF_NET_SOLAR
Surface net non-solar flux (W/m2)	GLW-STBOLT*EMISS*SST**4-LH-HFX	WRF_d01_EXT_d01_SURF_NET_NON-SOLAR
Evaporation-precipitation (kg/m2/s)	QFX-(RAINCV+RAINNCV)/DT	WRF_d01_EXT_d01_EVAP-PRECIP
Surface atmospheric pressure (Pa)	PSFC	WRF_d01_EXT_d01_PSFC
Wind stress module (N/m2)	taut = rho * ust**2	WRF_d01_EXT_d01_TAUMOD
U component of wind stress (N/m2)	taui = taut * u_uo / wspd	WRF_d01_EXT_d01_TAUX
V component of wind stress (N/m2)	tauj = taut * v_uo / wspd	WRF_d01_EXT_d01_TAUY
Eastward comp. of wind stress(N/m2)	cosa * taui - sina * tauj	WRF_d01_EXT_d01_TAUE
Northward comp. of wind stress(N/m2)	cosa * tauj + sina * taui	WRF_d01_EXT_d01_TAUN

Note

If you decide to couple CROCO with multiple WRF domains, variables coming from WRF will be defined by adding _EXT*. Here * corresponds to which domains the variable is coming (1=Parent, 2=Nest 1 ,...).

1.14.3.2 Coupling with a wave model

When coupling CROCO to a wave model, the wave-current interactions have to be set on. At the moment, only mean wave parameters are exchanged, their contribution to ocean dynamics is computed into the wave-current interaction routine in CROCO.

The following cpp-keys have to be set:

```
# define OW_COUPLING
# define MPI

# define MRL_WCI
```

Note

You also have to be careful to the choice of the momentum flux. For better consistency, here we suggest to account for the momentum flux seen by the wave model, and thus set:

```
# undef  BULK_FLUX
# define WAVE_SMFLUX
```

Fields sent by CROCO Name (units)	name and eventual oper. in the model	OASIS name
SSH (m)	zeta	CROCO_SSH
U-component of current (m/s)	u (at rho points): 0.5*(u(1:Lmmpi,1:Mmmpi,N,nnew) +u(2:Lmmpi+1,1:Mmmpi,N,nnew))	CROCO_UOCE
V-component of current (m/s)	v (at rho points): 0.5*(v(1:Lmmpi,1:Mmmpi,N,nnew) +v(1:Lmmpi,2:Mmmpi+1,N,nnew))	CROCO_VOCE
Eastward component of current (m/s)	u (at rho points) rotated to east (useful for rotated grids) (0.5 * (u(1:Lmmpi ,1:Mmmpi,N,nnew) + u(2:Lmmpi+1,1:Mmmpi,N,nnew))) * cos(angler(1:Lmmpi ,1:Mmmpi)) - (0.5 * (v(1:Lmmpi,1:Mmmpi ,N,nnew) + v(1:Lmmpi,2:Mmmpi+1,N,nnew))) * sin(angler(1:Lmmpi ,1:Mmmpi)) +u(2:Lmmpi+1,1:Mmmpi,N,nnew))	CROCO_EOCE
Northward component of current (m/s)	v (at rho points) rotated to north (useful for rotated grids) (0.5 * (u(1:Lmmpi ,1:Mmmpi,N,nnew) + u(2:Lmmpi+1,1:Mmmpi,N,nnew))) * sin(angler(1:Lmmpi ,1:Mmmpi)) + (0.5 * (v(1:Lmmpi,1:Mmmpi ,N,nnew) + v(1:Lmmpi,2:Mmmpi+1,N,nnew))) * cos(angler(1:Lmmpi ,1:Mmmpi))	CROCO_NOCE

Fields received by CROCO Name (units)	name and eventual oper. in the model	OASIS name
Significant wave height (m)	whrm = FIELD * 0.70710678	CROCO_HS
Mean wave period (s) -> frequency	wfrq = 2*pi / FIELD	CROCO_T0M1
Mean wave direction -> wavenumbers	wdrx = cos(FIELD - angler) wdre = sin(FIELD - angler)	CROCO_DIR
U component of wave stress (m2/s2)	twox (at u point): 0.5*(FIELD(io-1,jo)+FIELD(io,jo)) if eastward, it is first rotated: FIELD = etwo * cos(angler) + ntwo * sin(angler)	CROCO_UTWO or CROCO_ETWO
V component of wave stress (m2/s2)	twoy (at v point): 0.5*(FIELD(io,jo-1)+FIELD(io,jo)) if northward, it is first rotated: FIELD = ntwo * cos(angler) - etwo * sin(angler)	CROCO_VTWO or CROCO_NTWO
U comp. of wind-to-wave stress (m2/s2)	tawx (at u point): 0.5*(FIELD(io-1,jo)+FIELD(io,jo)) if eastward, it is first rotated: FIELD = etaw * cos(angler) + ntaw * sin(angler)	CROCO_UTAW or CROCO_ETAW
V comp. of wind-to-wave stress (m2/s2)	tawy (at v point): 0.5*(FIELD(io,jo-1)+FIELD(io,jo)) if northward, it is first rotated: FIELD = ntaw * cos(angler) - etaw * sin(angler)	CROCO_VTAW or CROCO_NTAW

Other optional fields enventually sent, if not, they are analytically computed in the MRL_WCI routine		
Bernoulli head pressure (N/m)	bhd	CROCO_BHD
Wave-to-ocean TKE flux (W/m2)	foc	CROCO_FOC
Mean wavelength (m)	wlm	CROCO_LM
Wave orbital bottom velocity (m/s)	ubr = sqrt(ubrx**2+ubry**2)	CROCO_UBRX and CROCO_UBRY
Stokes drift surface velocity (m/s)	ust_ext = sqrt(ustx_ext**2+usty_ext**2)	CROCO_USSX and CROCO_USSY

Fields received by WW3		
Name (units)	name (in the model)	OASIS name
SSH = water level (m)	LEV	WW3__SSH
Zonal current (m/s)	CUR	WW3_OSSU
Meridional current (m/s)	CUR	WW3_OSSV

Fields sent by WW3		
Name (units)	name (in the model)	OASIS name
Mean wave period (s)	T0M1	WW3_T0M1
Significant wave height (m)	HS	WW3__OHS
Mean wave direction	THM	WW3__DIR
Zonal wave stress (N/m2)	TWOX	WW3_TWOX
Meridional wave stress (N/m2)	TWOY	WW3_TWOY
Zonal wind stress (N/m2)	TAWX	WW3_TAWX
Meridional wind stress(N/m2) TAWY WW3_TAWY		
Other fields possibly sent, but not used in coupling with CROCO at the moment		
Bernoulli head pressure (N/m)	BHD	WW3__BHD
Bottom orbital velocity (m/s)	UBR	WW3__UBR
Wave-to-ocean TKE flux (W/m2)	FOC	WW3__FOC
Mean wavelength (m)	LM	WW3___LM
Wave peak frequency (/s)	FP	WW3___FP

1.14.3.3 Coupling atmosphere and wave models

Fields received by WW3		
Name (units)	name (in the model)	OASIS name
Zonal wind (m/s)	WND	WW3__U10
Meridional wind (m/s)	WND	WW3__V10

Fields sent by WW3		
Name (units)	name (in the model)	OASIS name
Significant wave height (m)	HS	WW3__AHS
Charnock coefficient	ACHA	WW3_ACHA

Fields sent by WRF		
Name (units)	name (in the model)	OASIS name
Zonal wind at first level((m/s)	u_uo	WRF_d01_EXT_d01_WND_E_01
Meridional wind at first level (m/s)	v_vo	WRF_d01_EXT_d01_WND_N_01

Fields received by WRF		
Name (units)	name (in the model)	OASIS name
Charnock coefficient	CHA_COEF	WRF_d01_EXT_d01_CHA_COEF

1.14.3.4 Note on momentum flux when coupling 3 models

As the wave model has a quite complex parameterization of wave generation by winds, which is in subtle balance with the wave dissipation, the wind stress for the wave model is computed by its own parameterization. Therefore, to ensure energetic consistency of the momentum flux when coupling 3 models, we prescribe the wind stress in CROCO as:

```
sustr = sustr_from_atm_model - tawx + twox
svstr = svstr_from_atm_model - tawy + twoy

# where taw is stress from atm to waves
# and   two is stress from waves to ocean
```

1.14.3.5 Note on coupling with AGRIF

You may decide to coupled CROCO while using AGRIF. To do so, the variables sent by the parent domain (0) and the child domains (1,2,...) must be separated. Thus the variables sent, in case of using AGRIF, take the radical defined above (CROCO_VAR) to which we add _0 (for parent) or _1 (for first child). This gives, for example for variable SST, `CROCO_SST_0` or `CROCO_SST_1` for parent and child respectively.

For the variables received by CROCO, we will use its ability to handle CPLMASK. Each of the domains (parent or children) will be assigned a coupling mask named coupling_mask0.nc (parent), coupling_mask1.nc (child 1), each coupling mask being relative to its grid. The CROCO domain that receives a variable will be identified by its mask (CPLMASK*), which will be added to the previous radical. This will give CROCO_VAR_CPLMASK0 for the parent or CROCO_VAR_CPLMASK1 for child 1.

This makes it easy to define the received variables in a case where one decides to couple CROCO-AGRIF with several WRF domains. In this case the variables will have the nomenclature `CROCO_VAR_CPLMASK0` for the parent CROCO to which we add _EXT1 for the first domain of coupling_mask0.nc. By continuity `_EXT2` will correspond to domain 2 of coupling_mask0.nc. Then the variables received by CROCO, in a case of CROCO-AGRIF/WRF-nest simulation, will follow the format `CROCO_VAR_CPLMASK*_EXT*`.

1.14.4 Grids

1.14.4.1 OASIS grid files

OASIS manage grids and interpolations by using dedicated grid files:

- `grids.nc`
- `masks.nc`
- `areas.nc` (requested only for some of the interpolation types)

These files can be automatically created by OASIS functions called in each model, or can be created by the user in advance if specificities are requested. Some facilities are provided in `croco_tools/Coupling_tools` to create such grids.

If `grids.nc`, `masks.nc`, `areas.nc` exist in the working directory, they won't be overwritten by OASIS functions. So, be sure to have the good files or remove them before running the coupled model.

1.14.4.2 Multiple model grids (nesting case)

Multiple nested grids in the different models can be used in coupled mode.

The variables are therefore exchanged from/to the different grids. To do so, each coupled variable is identified in the coupler with its grid number:

- For CROCO the last character of the OASIS variable name defines the domain
  - 0 being the parent domain
  - 1 the first child domain, etc.
- For WRF the domains are defined by d01, d02, etc, and the target domain (CROCO for instance), by EXT_d01, EXT_d02, etc.

For example if you are coupling 2 CROCO domains to one atmospheric domain, you will specify 2 types of exchanges in the `namcouple`:

```
# exchange between CROCO parent domain to WRF domain
SRMSSTV0 WRF_d01_EXT_d01_SST

# exchange between CROCO child domain to WRF domain
SRMSSTV1 WRF_d01_EXT_d02_SST
```

If you are coupling 2 WRF domains to one CROCO domain:

```
# exchange between WRF d01 domain to CROCO domain
WRF_d01_EXT_d01_TAUMOD RRMTAUM0

# exchange between WRF d02 domain to CROCO domain
WRF_d02_EXT_d01_TAUMOD RRMTAUM0
```

Related CPP options:

OW_COUPLING	Activate Ocean-Wave coupling
OA_COUPLING	Activate Ocean-Atmosphere coupling
OA_MCT	Use OASIS-MCT for coupling

Preselected options:

```
#undef OA_COUPLING
#undef OW_COUPLING
```

1.15 I/O and Online Diagnostics

1.15.1 General ouptut options

AVERAGES	Process and output time-averaged data
AVERAGES_K	Process and output time-averaged vertical mixing
XIOS	Use XIOS IO server (only version >= 2 is supported)

XIOS is an external library for output (developed at IPSL) providing for flexibility and design to improve performances for HPC : see http://forge.ipsl.jussieu.fr/ioserver

Preselected options:

```
# define AVERAGES
# define AVERAGES_K
# undef XIOS
```

1.15.2 Advanced diagnostics options

DIAGNOSTICS_TS	Store and output budget terms of the tracer equations
DIAGNOSTICS_TS_A	Choose advection rather than transport formulation for tracer budgets
DIAGNOSTICS_TS_M	Integrate tracer budgets over the mixing layer depth (hbl, defined as the turbulent layer from the mixing param)
DIAGNOSTICS_TS_M	Integrate tracer budgets over the mixed-layer depth (MLD, defined with criterion in density, temperature or Brunt-Vaisala frequency)
DIAGNOSTICS_TSVA	Store and output budget terms of the tracer variance equations (instead of tracer)
DIAGNOSTICS_UV	Store and output budget terms of the momentum equations.
DIAGNOSTICS_BARC	Isolate contribution from barotropic/baroclinic coupling (included in the vertical mixing term otherwise) for momentum, barotropic vorticity and kinetic energy budgets
DIAGNOSTICS_VRT	Store and output budget terms of the barotropic vorticity equation
DIAGNOSTICS_EK	Store and output budget terms of the kinetic energy equation (vertically integrated)
DIAGNOSTICS_EDDY	Store and output time-averaged quadratic quantities u^2, v^2, u*v, u*w, v*w, u*b, v*b, w*b, u*sustr, v*svstr, u*bustr, v*bvstr, zeta^2
DIAGNOSTICS_PV	Store and output non conservative term in the momentum equations and diabatic term in the tracer equations. if DIAGNOSTICS_DISS is also defined, terms are multiplied by momentum and thermal/saline expension coefficients to be used to estimate kinetic and potential energy dissipation.

Preselected options:

```
# undef DIAGNOSTICS_TS        Store and output budget terms of the tracer equations
# undef DIAGNOSTICS_TS_ADV Choose advection rather than transport formulation for␣
↪tracer budgets
# undef DIAGNOSTICS_TS_MLD Integrate tracer budgets over the mixed-layer depth
# undef DIAGNOSTICS_TSVAR  output budgets of tracer variance instead of tracer
# undef DIAGNOSTICS_UV        Store and output budget terms of the momentum equations
# undef DIAGNOSTICS_BARO   Isolate contribution from barotropic/baroclinic coupling
# undef DIAGNOSTICS_VRT    Store and output budget terms of the barotropic vorticity␣
↪equation
# undef DIAGNOSTICS_EK     Store and output budget terms of the kinetic energy␣
↪equation (vertically integrated)
# undef DIAGNOSTICS_PV     Store and output non conservative / diabatic terms
# undef DIAGNOSTICS_EDDY   Store and output time-averaged quadratic quantities
```

1.15.2.1 Energy budget terms

The different budgets and their computation are detailled in https://www.jgula.fr/Croco/diagnostics_croco.pdf

1.15.2.2 Tracer budget terms

DIAGNOSTICS_TS computes the tracer budgets in 3D.

DIAGNOSTICS_TS_MLD computes the tracer budgets:

- over the mixing layer (*hbl*, defined as the turbulent layer from the vertical mixing parametrization)
- over the mixed layer (defined as the homogeneous surface layer) if **DIAGNOSTICS_TS_MLD_CRIT** is defined in addition.

> **Note**
>
> To integrate tracer budgets over the mixed layer depth, the CPP key **LMD_SKPP** needs to be activated.

Tracer budgets will be stored in **croco_dia.nc** (instantaneous) and/or **croco_dia_avg.nc** (average) output files.

1.15.2.2.1 Mixing layer calculation

The turbulent mixing layer (hbl) is defined depending on the mixing parameterization (see *Vertical mixing parametrizations*).

1.15.2.2.2 Mixed layer calculation

The mixed layer depth (MLD) is defined as the slice of the surface ocean where temperature, salinity and density are homogeneous. In reality, as the mixed layer is not perfectly homogeneous, a threshold value is applied to determine when temperature/density is no longer considered homogeneous.

> **Warning**
>
> Temperature budget term is computed according to 4 criteria. But for the other tracers the only criteria is on density !

4 criteria are available to compute the mixed layer, and the associated temperature budgets:

- one density criterion (CRT1), which default value is set to 0.03 kg/m^3
- two temperature critera (CRT2 and CRT3), which default values are respectively 0.2°C and 0.5°C
- one Brunt-Vaisala frequency criterion (CRT4) based on the delta density threshold of CRT1.

These default values are taken from [de Boyer Montegut C. *et al.*, 2004].

> **Note**
>
> These values depend on the study area and are valid for deep waters.

1.15.2.2.2.1 Density criterion

The MLD is evaluated as the depth where the density is equal to the density at a reference level plus the chosen threshold ([de Boyer Montegut C. *et al.*, 2004]). The reference level is defined as the level with the depth closest to the reference depth defined in *diag_mld_depth_ref*, a user-modifiable parameter (and set to 10m by default).

The threshold value and the reference depth are user-modifiable parameters.

Starting from the surface, the mixed layer level corresponds to the last level that satisfies the following condition:

$$D(x,y,k,t) \leq D(x,y,k_{ref},t) + D_{threshold} \tag{1.17}$$

$$\tag{1.18}$$

with :

- $D(x,y,k,t)$ the density

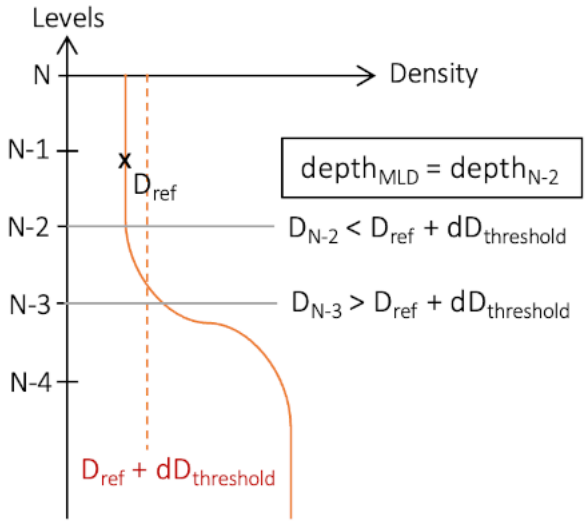


Fig. 13: Fig: schematic view of the calculation of the mixed layer depth based on the density criterion.

1.15.2.2.2.2 Temperature criterion

The MLD is evaluated as the depth where the temperature is equal to the temperature at a reference level minus the chosen threshold ([de Boyer Montegut C. *et al.*, 2004]). The reference level is defined as the level with the depth closest to the reference depth defined in *diag_mld_depth_ref*, a user-modifiable parameter (and set to 10m by default).

The threshold value and the reference depth are user-modifiable parameters.

Starting from the surface, the mixed layer level corresponds to the last level that satisfies the following condition:

$$T(x,y,k,t) \geq T(x,y,k_{ref},t) - T_{threshold} \tag{1.19}$$

$$\tag{1.20}$$

with :

- $T(x,y,k,t)$ the temperature

1.15.2.2.2.3 Brunt-Vaisala frequency criterion

The Brunt-Vaisala frequency is defined as the frequency at which a water parcel would oscillate vertically if slightly displaced from its equilibrium position. If the frequency squared is greater than 0, then a restoring force is applied, oscillation occurs and the fluid is stratified. The corresponding criterion is calculated in relation to a density delta equal to the user-defined threshold. The Brunt-Vaisala frequency is calculated and integrated over depth levels. The layer is no longer considered mixed when the bvf is greater than the criterion.

Starting from the surface, the mixed layer level corresponds to the last level that satisfies the following condition:

$$\sum_{k=\text{indxdepth}}^{1} \max(\text{bvf}(i,j,k,t),0) \cdot \Delta z_r \quad < zN_c^2 \tag{1.21}$$

$$with\ zN_c^2 = \frac{g * D_{threshold}}{rho0} \tag{1.22}$$

with :

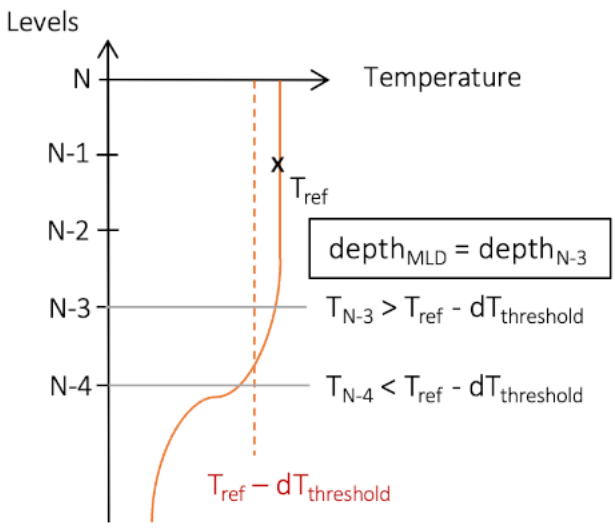


Fig. 14: Fig: schematic view of the calculation of the mixed layer depth based on the temperature criterion.

- $indxdepth$ the index of the depth closest to the reference depth chosen by the user
- $bvf(x, y, k, t)$ the Brunt-Vaisala frequency
- z_r depth at rho-point
- $\ddagger\mathcal{N}^{\in}{}_{c}$ Brunt-Vaisala frequency criteria based on the density threshold

1.15.2.2.3 Parametrization tips

The parameters are defined and can be modified in the **croco.in** file (or **croco_inter.in**):

```
diag_mld_crit_threshold: CRT1 (Dens [kg/m3]), CRT2 (Temp [Celsius]), CRT3 (Temp␣
↪[Celsius])
                                3.d-2                 2.d-1                   5.d-1

diag_mld_depth_ref: [m]
                  10
```

The user can choose which variables to include in the output files by flagging the tracers with T (True) or F (False).

Temperature and salinity are examples of tracers. The temperature tracer has been multiplied to output for each criterion.

Here's the order of the tracers when the CPP-keys **TEMPERATURE** and **SALINITY** are enabled (which is the default): temp_CRT1, salinity, temp_CRT2, temp_CRT3, temp_CRT4

Here's an example to output the temperature budget diagnostics for the 4 MLD criteria, but deactivate the salinity budget diagnotics:

```
diag3D_history_fields:    diag_tracers3D(1:NT)
                  T F T T T 50*F

diag2D_history_fields:    diag_tracers2D(1:NT)
                  T F T T T 50*F

diag3D_average_fields:    diag_tracers3D_avg(1:NT)
                  T F T T T 50*F

diag2D_average_fields:    diag_tracers2D_avg(1:NT)
                  T F T T T 50*F
```

> **Note**
>
> The depth variable's metadata (*temp_CRTx_depth_mld*) is used as a reminder of the user's parametrization choices and thus provides the threshold value and the reference depth used for the calculations.

1.16 Testing CROCO

CROCO comes with dozen of test cases and benchmarking scripts.

1.16.1 Test Script

> **Note**
>
> This section describes how to use the script for testing the CROCO code on different configurations and hosts.

1.16.1.1 Introduction

This script allows executing and evaluating the performance of the CROCO code on various setups:

- **Execution modes**: Sequential, MPI, OpenMP, debug or standard optimizations
- **Hosts**: Runs on different architectures
- **Tests performed**:
  - Checking parallel reproducibility
  - Checking restartability
  - Non-regression testing by comparing with a reference version
  - Performance analysis (scalability, execution time)
  - Visualization of physical results

It uses the concepts of :

- **cases** : The cases to run (BASIN, SOLITON…) and some case size changing the grid size (eg. BASIN-LARGE, BASIN-HUGE).
- **variants** : The parallelization mode (openmp, mpi, gpu) with specification on the resources to use because CROCO needs to be rebuilt for every case du to some currently static definitions in the CROCO code on the use of those resources.
- **hosts** : The specific adaptation to make for the host you are running on.

All the options are passed through *json* files wholly configurable.

The benchmarking script is used to compare CROCO on various tests configurations including the ones listed at the beginning of cppdefs.h file and in the *review of test cases*.

The benchmarking script is in BENCH folder : `BENCH/bench-croco.py`. A lot of options are available and detailed in this documentation.

1.16.1.2 Requirements

As it will build CROCO for you, you will need to have the basic CROCO dependencies available (see *CROCO system requirements*). Including MPI, OPENMP or OPENACC if you want to use it.

Additionaly, you will have to install python packages listed in `requirements.txt`.

If you already have an activated venv

```
pip install -r requirements.txt
```

Otherwise you first needs to create a `virtualenv` :

```
# only the first time
python3 -m venv venv
# for every terminal used to run the bench script
source venv/bin/activate
# only once
pip install -r requirement.txt
```

1.16.1.3 Usage

1.16.1.3.1 Running the tests

To execute the script with a predefined set of configurations:

```
# simplest way
./bench-croco.py

# use your own config file
./bench-croco.py -c config-mine.jsonc
```

It will run the *cases* and *variants* combination defined in *default_selection* of the config file.

You can also choose by hand some of those :

```
# just a single
./bench-croco.py --case BASIN --variants sequential

# or many
./bench-croco.py --case BASIN,BASIN-LARGE --variants sequential,openmp-4,mpi-4
```

1.16.1.3.2 Script options

The testing script comes with a lot of options, you can list them all with the following command:

```
./bench-croco.py -h
```

More details on each capability are given in the following sections.

1.16.1.3.2.1 Use a specific configuration file

The default configuration file is `BENCH/config.jsonc`. It contains all the available cases, variants and host. If you want to run a new test case configuration, you will have to add it in this file.

A specific configuration file can be provided.

Example :

```
# use your own config file
./bench-croco.py -C config-mine.jsonc
# or equivalent
./bench-croco.py --config config-mine.jsonc
```

Details on the configuration file are in *configuration file section*.

Tip

Configuration file can be splitted in subdirectory mode by merging several subfiles.

In the main *config.jsonc* file, the inclusion is described by the *imports* section.

See also *imports section*.

1.16.1.3.2.2 Select cases, variants, host

The default cases run are BASIN and BASIN-LARGE. You can select the case you want to run.

Example :

```
# use a specific case
./bench-croco.py -c CANYON
# or equivalent
./bench-croco.py --cases CANYON
```

The default variants run are @cpu which is a *meta_variant* defined in the configuration file. You can select a specific variant you want to run.

Example for the variant mpi-2 (MPI using 2 cores) :

```
# use a specific variant
./bench-croco.py -v mpi-2
# or equivalent
./bench-croco.py --variants mpi-2
```

If you want to adjust options to your host, you can specify a host. If you want to add your specific host, you can add it in the configuration file or in a new file in the hosts directory.

Example to use "bigfoot" host options:

```
# use your own config file
./bench-croco.py --host bigfoot
```

Tip

Wildcards

You can also use wildcards for *case* and *variant* to select them. For example below to run all BASIN* cases and all OpenMP variants :

```
./bench-croco.py --cases 'BASIN*' --variants 'openmp-*'
```

Tip

Meta variants

You will notice in the *config.jsonc* file the definition of *meta_variants* which allow you to define sets of variants which you might use often for your run.

For example :

```
"meta_variants": {
"@debug": ["sequential", "mpi-2", "mpi-4", "openmp-2", "openmp-4"],}
```

Then you can just use them easily:

```
./bench-croco.py --cases BASIN --variants @debug
```

And you can remove some of the elements by using '-' :

```
./bench-croco.py --cases BASIN --variants @debug,-openmp-4
```

Tip

Meta case

If you want to simply run all available cases and all variants, you can just:

```
./bench-croco.py --cases @all --variants @all
# you can remove some of the elements by using '-' :
./bench-croco.py --cases @all,-BASIN-LARGE --variants @all
# you can also use wildcard in the expr for removing :
./bench-croco.py --cases '@all,-BASIN-*' --variants '@all,-openmp-*'
```

You can also defined your own meta-case in your specific configuration file.

1.16.1.3.2.3 Select the modes to run

The script run in steps. The available steps are :

- build : code compilation for each case and variant
- run : run simulation for each case and variant
- check : perform comparison with a reference
- plotperf : performance plot to show runtime of each variant
- plotphy : physical plot using python script specified in case config
- plotraw : plot raw map of variables at start/middle/end of simulation.
- animraw : same plot as mesh with an animation over the simulation.

The default steps are build,run,check,plotperf.

You can specify the steps you want to run.

Example:

```
./bench-croco.py --cases BASIN -s build,run
#or
./bench-croco.py --cases BASIN --steps build,run
```

1.16.1.3.2.4 Build options

The first step run by the script is the build one.

For each case and each variant, CROCO will be compiled in the corresponding *rundir*.

The script use jobcomp script to build croco executable. Example:

```
./bench-croco.py --cases BASIN -s build
```

If you rebuild a case you have already built, the script won't rebuild it. To force a rebuild, you can add `-r` or `--rebuild` in the command line. Example:

```
./bench-croco.py --cases BASIN -s build --rebuild
```

By default, the script use `-j 4` option to compile in parallel. You can adjust this behavior with the option `-j` or `--jobs` in the command line.

Example:

```
./bench-croco.py --cases BASIN -s build --jobs 1
```

Warning

Note that with AGRIF activated, code won't compile in parallel as -j option is set to 1 in jobcomp.

1.16.1.3.2.5 Run options

By default, **each run is done only one time**. To do performance tests, you can perform each run several time by specifying the number of times you need in the command line using `-R` or `--runs` options.

Example:

```
./bench-croco.py --cases BASIN -R 5
```

If you run tests on an host without sufficient resources to run a specific variant, this one can be automatically skipped by adding the option `-a` or `--auto-skip` in the command line.

Example:

```
./bench-croco.py --cases @all --variants @all -r --auto-skip
```

Some tests requires external datasets (mostly large Netcdf files). This is the case for the predefined configurations REGIONAL and COASTAL. You can download the needed datasets to run test in the Download section: https://www.croco-ocean.org/download/.

Once you have the files on the host you are running, you can specify its path in the command line using `--data-root-path`. In this case, specified path should contain cases specific directories (croco_files_path in *case config file*).

Example:

```
./bench-croco.py --cases BENGUELA_LR* --data-root-path /my/data/path
```

Note

The option `--data-root-path` is not required for all analytical cases which does not need large external datasets.

1.16.1.3.2.6 Check options

When all cases and variants are done, you can perform the check step.

For each case, this step will compare netcdf file resulting of each variant with the file resulting of the reference variant (by default sequential).

You can specify another variant than sequential using `--compare-to` option.

Example:

```
./bench-croco.py -v sequential-ifort,mpi-ifort-4 --compare-to sequential-ifort
```

By default, comparison is made over netcdf results file. A more precise option is available through the option `--cvtk`. This enable usage of CVTK_DEBUG cppkey for comparison with a reference and track difference directly in the code.

For some reason, the reference sequential run can cost a lot to produce if the case is large. In order not to redo it all the time, you can use two approach :

The simpler one consist in making the run a first time to get the reference data generated then to run without *sequential*:

Example:

```
# first time
./bench-croco.py --cases @all --variants @all
# others
./bench-croco.py --cases '@all,-sequential' --variants @all
```

The other approach consist in producing a reference directory which can be zipped and uploaded somewhere. This can be done using `--build-ref` and then `--use-ref` options.

Example:

```
# first time
./bench-croco.py --cases @all --variants @all --build-ref ./croco-refs-2025-02
# others
./bench-croco.py --cases @all --variants @all --use-ref ./croco-refs-2025-02
```

1.16.1.3.2.7 Performance options

In order to perform correct performance measurement you can ask *bench-croco.py* to repeat several times the run to get some error bars.

By default it makes only a single run per case/variant.

You can set it to 5 by :

```
./bench-croco.py --case BASIN --variants @all --runs 5
```

By default, when plotting performance results, the scripts will search for all results corresponding to the case in the result directory. You can avoid loading previous results by using `-N` or `--no-previous`.

1.16.1.3.2.8 Working and results directories options

The script will build CROCO for each *case/variant* in the workdir (*./rundir* by default). You can specify another directory using `-w` or `--workdir` option.

The results are stored in *./results* (by default). You can specify another directory using `--results` option.

The script will keep the results of each run by *{hostname}-{date}* but you can also ask the script to add a given name if you are making different tries *{title}-{hostname}-{date}* .

```
# while you are developping your optimizations (trash dir)
./bench-croco.py --title dev
# Produce clean final perf ready for paper
./bench-croco.py --title final-perf
```

You can also remove date from the folder name by using `--no_date_in_result_dir` option.

```
./bench-croco.py --title mytest --no_date_in_result_dir`
```

1.16.1.3.2.9 Reporting

If you want to run *bench-croco.py* for validation and get a summary report on what is valid or not, you can simply add ``–report``to the command line :

```
./bench-croco.py --case @all --variants @all -r --report
```

You will get an output like :

```
------------------------------ REPORT SUMMARY ---------------------------------
CASE-VARIANT                      Build     Run           Check    Plotphy
SOLITON-sequential                None      True          True     None
SOLITON-openmp-2                  None      True          True     None
SOLITON-LARGE-sequential          None      True          True     None
SOLITON-LARGE-openmp-2            None      True          True     None
```

Plus the results stored in *status_report.json*.

> **Note**
>
> The option `--report` automatically add the option `--continue` which allows the script to continue instead of stopping on first error.
>
> You can therefore used `--continue` without `--report` option.

1.16.1.3.2.10 Html navigation

You can ask *bench-croco.py* to generate a html treeview report by adding `--html`.

You can also ask for a general html report of all the results directory with the option `--globalhtml`. In this case bench steps are not run and the scripts will only browse all the previous runs in the results directory.

1.16.1.3.2.11 Debug option

You can add `--debug` option to your command line. This will add debug compilation option defined in the *host config file*.

To avoid long unnecessary simulations, this will also limit the number of time steps to **6** and modify the CROCO output to write every timesteps.

This option is helpfull for CROCO developpers.

1.16.1.3.2.12 Restart option

You can add `--restart` option to your command line. This will run the reference variant once and after all the variant in restarted mode using the restart file generatad by the reference variant.

The check steps will then verify the restartability of the case.

To avoid long unnecessary simulations, this will also limit the number of time steps to **6**.

1.16.1.4 Configuration files structures

The configuration file `config.jsonc` contains all the available cases, variants and hosts known by *bench-croco.py* script.

If you want to add a new case, a new variant or a new host, you must add it in the configuration.

1.16.1.4.1 Imports

You will notice at the head of the configuration file the *imports* section which allow to split the config file in multiple sub files and merge them at loading time.

```
//////////////////////////// IMPORTS ////////////////////////////////////////
"imports": {
  "hosts": ["config.d/hosts/*"],
  "cases": ["config.d/cases/*"],
  "variants": ["config.d/variants/*"],
},
```

This is usefull for exemple for the *hosts* and *cases* definitions as it permits a user to define its own hosts without requirement to patch the files tracked by git and to keep its definition for himself until he finally decide to commit it.

1.16.1.4.2 Case

For each case configuration, a specific file is added in the config.d/cases/ directory. You can browse this directory to explore how cases are configured.

> **Tip**
>
> For one configuration, you can put several cases to change cppkeys or domain definition.
>
> For example, for REGIONAL configuration, several cases are defined :
>
> - BENGUELA_LR_TIDES_BULK
> - BENGUELA_LR_AGRIF_1WAY
> - ...

Each case can contains the following keys :

- title: string, the title you will have in plot
- case: string, the cppkeys activated for the case
- splitting: dict, used with mpi or openmp variants to distribute resources along xi/eta axis. Note that if a splitting is not available for the number of processors you ask, the variant will be automatically skipped
- patches: dict, see *Case Configuration patching*
- check_outputs: list of strings, list of files used for check step (example : ["basin_his.nc"])
- croco_files_path: string[**optionnal**], path toward external dataset in the given *data_root_path*
- plotphy : dict, [**optionnal**], used with plotphy step to produce a physical plot, must contains plotphy_script (string, name of the python script to use) and plotphy_script_input (string, name of the file tu use in script arg)
- input_file: string[**optionnal**], default is TEST_CASES/croco.in.<capitalised_case>
- unsupported: list of strings[**optionnal**], list of unsupported option or variant (example : ["restart","openmp"]).
- cppkeys: dict[**optionnal**], to list cppkeys to activate or deactivate for this case.

1.16.1.4.3 Variant

Each variant can contains the following keys :

- configure: variant options to add to the configure command line
- tuning_familly: "gnu", "intel" or "nvfortran"
- environ : if a specific environment variable is needed
- command_prefix : if a prefix is needed

- unpack : to avoid to repeat for each threads number variables can be unpacked at runtime (See also *Variables*)
- requires : to defined the number of cpu/gpu needed

Sequential, mpi-, *openmp-* and openacc variants are already defined, in config.d/variants directory.

1.16.1.4.4 Host

Each host configuration can contains the following keys :

- resources: to list the cpu and gpu available on the host
- tuning: to list options [extra, standard, debug, mpi_option] for gnu, intel and nvfortran
- environ[**optionnal**]: if you need specific variable
- hwloc_bind[**optionnal**]: if you need options for openmp or mpi

A default configuraion is given in config.jsonc in host @default.

1.16.1.4.5 Case Configuration patching

You will notice in the cases configuration files (*config.d/cases/**) that there is a patching part which explain how to modify the CROCO configuration files to get the case setted up.

As an example, the patching to increase the domain size can be done by :

```
"patches": {
// Define a file to patch
"param_override.h": {
    // Patch mode
    "mode": "replace",
    // What to replace (expected to be present)
    "what":     "      parameter (LLm0=60,   MMm0=50,   N=10)",
    // By what to replace
    "by":       "      parameter (LLm0=200,  MMm0=200,  N=50)",
    // Description to be printed in the terminal when applied
    "descr":   "Set mesh size to (200 x 200 x 50)"
},
},
```

Tip

It can also contains an array of patch elements if needs to change several things in the same file.

Example :

```
"patches": {
    // Change the number of steps and the output frequency
    "TEST_CASES/croco.in.Basin": [{
        "mode": "replace",
        "what":    "                3240     9600     65      1",
        "by":      "                   6     9600     65      1",
        "descr":   "Set stepping (steps=6, dt=9600, fast=65)"
    },{
        "mode": "replace",
        "what":    "            T     90       0 ",
        "by":      "            T      1       0 ",
        "descr":   "Set dtout every steps"
    }]
},
```

Available mode of patching are :

- replace
- insert-before
- insert-after
- insert-after-before
- insert-at-begin
- insert-at-end

Each patch must be filled with a dictionnary called *rule* with the following keys :

- "mode" (str, required): One of listed modes above.
- "what" (str, required): The line to match (after normalization) for certain operations.
- "by" (str): Line used in 'replace' mode.
- "insert" (str or list): Line(s) to insert in 'insert-before', 'insert-after', 'insert-after-before' modes.
- "after" (str, required for 'insert-after-before'): The line after which to insert.
- "before" (str, required for 'insert-after-before'): The line before which to insert.
- "descr" (str, optional): Description of the rule for logging

1.16.1.4.6 Variables

You might notice in the configuration file the use of some variables to be replaced at parsing time on the form of : *{tuning.gnu}* or *{task}*.

It avoids duplicating too many entries in the file.

You will mostly find them at two places :

- On the cases to unpack king of template not to fully duplicate all cases for OpenMP and MPI.
- To inject some values from *host* into the *jobcomp-configure* call.
- To inject the mpi extra options in *command_prefix*.

1.16.1.5 Not implemented yet

The script can not perform XIOS and coupled cases using OASIS yet.

Theses features are planned to be add in the future.

1.16.1.6 Frequently Asked Questions (FAQ)

Below are some frequently asked questions about the CROCO test script.

1.16.1.6.1 Can I use intel compiler ?

Yes you can, you will only have to check the tuning option for intel are given in host file and that the variant you used have the "tuning_familly": "intel". And of course intel compiler must be available in the host you used.

1.16.1.6.2 Does –cvtk option available with –debug and –restart ?

Yes you can run tests with all these options at the same time. Therefore, if the run failed due to non reproducibility or non restartability with `--cvtk` option, the `--continue` or `--report` options will not work and the script will exit at the first fail.

1.16.1.6.3 How can I run a simple analytical case ?

One of the most simple use of the script is :

```
./bench-croco.py -c BASIN -v sequential
```

This command is running the BASIN case in sequential. If you encounter errors, check if the default host is corresponding to your machine. If not, you can configure your own host. See also : *host*

For a list of available analytical cases, you can search in config.d/cases the JSONC files for each configurations.

You can also search in the *review of test cases*.

1.16.1.6.4 How can I run a realist case ?

Code is provide with realist configurations defined in files realist.jsonc and coastal.jsonc. To run them, you will need external datasets available at the url https://data-croco.ifremer.fr/ForCI-INRIA.tar.gz.

Once you have download the archive, extract it and put in `--data-root-path` the path toward the directory

```
./bench-croco.py -c BENGUELA_LR*,VILAINE -v mpi-4 --data-root-path /my/data/path
```

See also : *data_root_path*

1.16.1.6.5 Can I define a new test based on an existing configuration ?

Yes.

If your configuration fit with one already in cppdefs.h file, you will need to add your sub-test in the jsonc file corresponding to the configuration (example basin.jsonc or canyon.jsonc). Then you will have to complete the *case config file*.

For example, you can choose wich cppkeys you want to activate or deactivate

```
"cppkeys": {
    "FRC_BRY": true,
    "BULK_FLUX": true,
    "GLS_MIXING": true,
    "LMD_MIXING": false,
},
```

You can also specify another croco.in file if the default one for this configuration does not fit your needs.

```
"input_file": "TEST_CASES/croco.in.My_awesome_case",
```

If you need external datasets, do not forget to put the path in

```
"croco_files_path": "MY_PATH",
```

1.16.1.6.6 Can I define a new test NOT based on an existing configuration ?

Yes.

If your test is a whole new configuration, you will have to :

- update cppdefs.h file to add the configuration
- add corresponding input file in TEST_CASES directory
- add corresponding *case config file* in config.d/cases.

1.16.1.6.7 How can I get help ?

For further assistance, refer to the documentation or contact the development team.

1.16.1.7 Conclusion

This script provides full automation of CROCO tests, simplifying performance evaluation and simulation validation.

For any questions, post on the user forum or contact the development team.

1.16.2 Review of test cases

1.16.2.1 Basin

This is a rectangular, flat-bottomed basin with double-gyre wind forcing. It produces a western boundary current flowing into a central Gulf Stream which goes unstable and generates eddies if resolution is increased.

```
# define BASIN
```

CPP options:

```
# undef  OPENMP
# undef  MPI
# define UV_ADV
# define UV_COR
# define UV_VIS2
# define SOLVE3D
# define TS_DIF2
# define BODYFORCE
# define ANA_GRID
# define ANA_INITIAL
# define ANA_SMFLUX
# define ANA_STFLUX
# define ANA_BTFLUX
# define NO_FRCFILE
```

Settings :

Results :

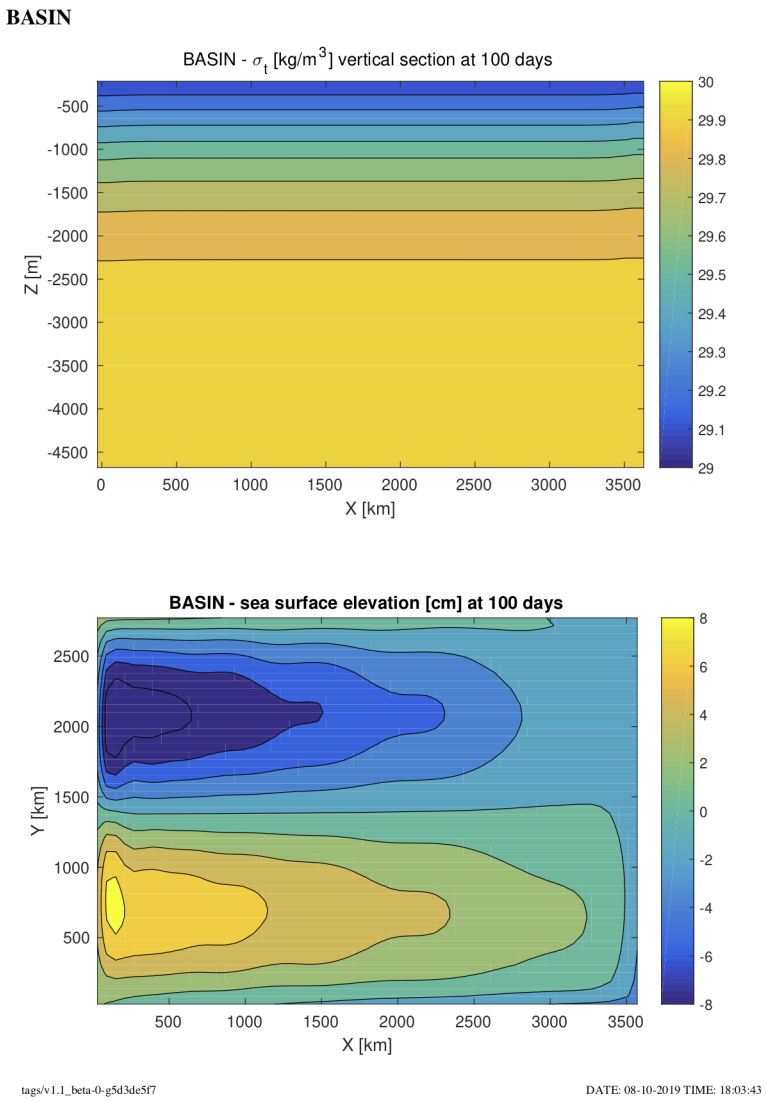


Fig. 15: BASIN results : density (up) and sea surface elevation (down)

1.16.2.2 Canyon

```
# define CANYON
```

CPP options:

```
# undef  OPENMP
# undef  MPI
# define CANYON_STRAT
# define UV_ADV
# define UV_COR
# define SOLVE3D
# define EW_PERIODIC
# define ANA_GRID
# define ANA_INITIAL
# define ANA_SMFLUX
# define ANA_STFLUX
# define ANA_BTFLUX
# define NO_FRCFILE
```

Settings :

Results :

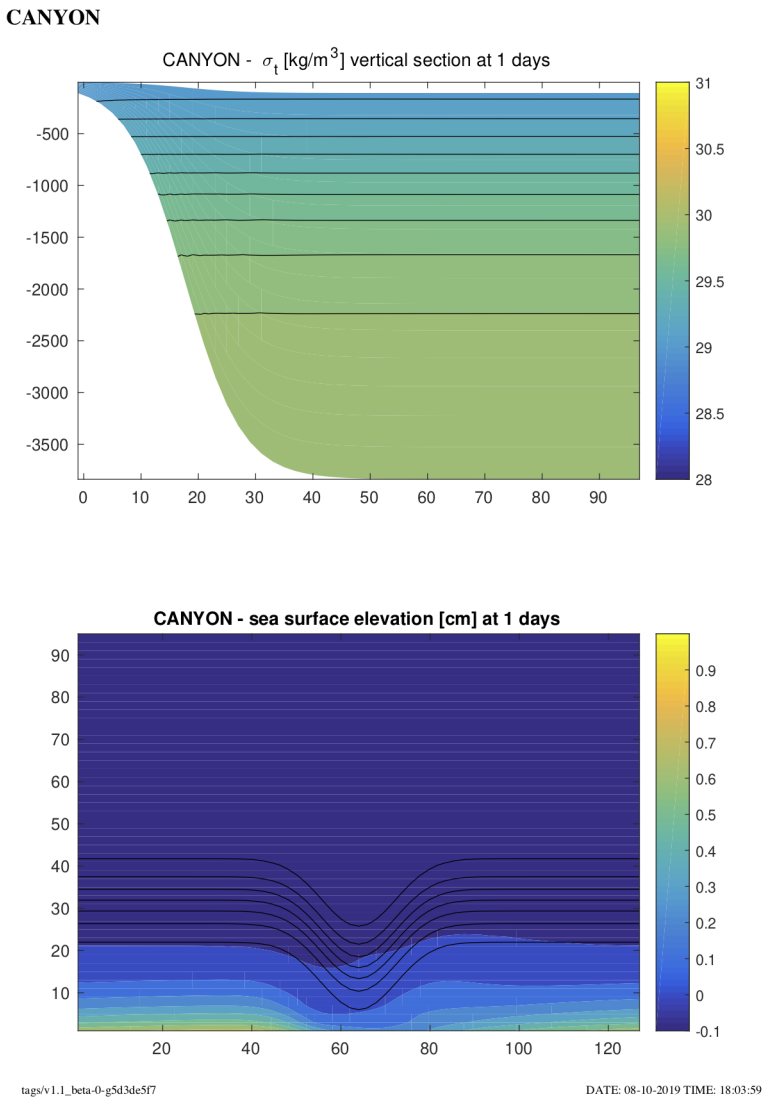


Fig. 16: CANYON results : density (up) and sea surface elevation (down)

1.16.2.3 Equator

Boccaletti *et al.* [2004]

```
# define EQUATOR
```

CPP options:

```
# undef  OPENMP
# undef  MPI
# define UV_ADV
# define UV_COR
# define UV_VIS2
# define SOLVE3D
# define TS_DIF2
# define ANA_GRID
# define ANA_INITIAL
# define ANA_SMFLUX
# define ANA_STFLUX
# define ANA_SRFLUX
# define ANA_SSFLUX
# define ANA_BTFLUX
# define ANA_BSFLUX
# define QCORRECTION
# define ANA_SST
# define LMD_MIXING
# define LMD_RIMIX
# define LMD_CONVEC
# define NO_FRCFILE
```

Settings :

Results :

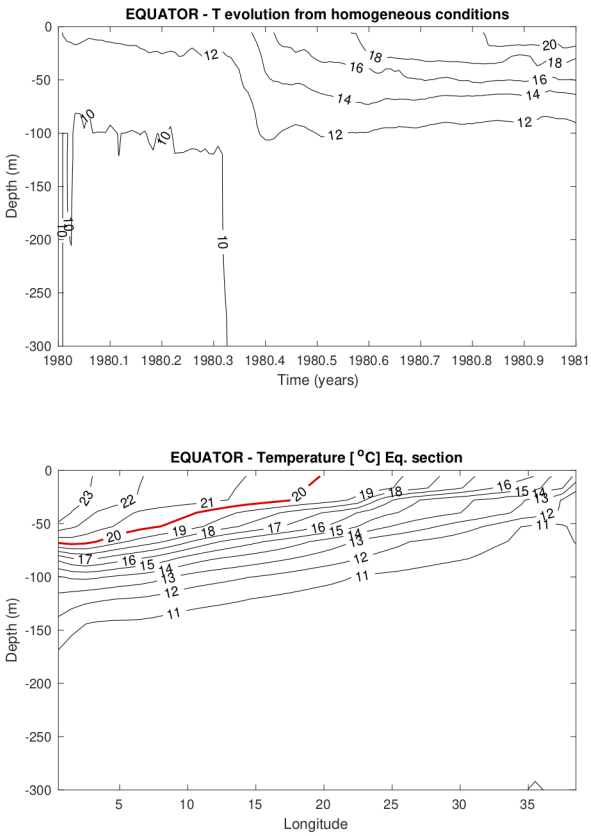


Fig. 17: EQUATOR results : temperature , time evolution (up) vertical section (down)

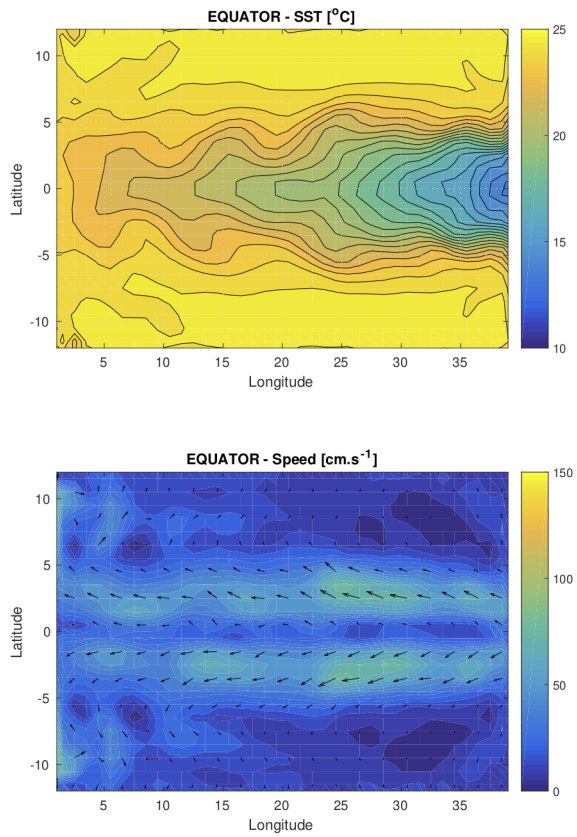


Fig. 18: EQUATOR results : speed (up) and sea surface elevation (down)

1.16.2.4 Inner Shelf

Compare wind driven innershelf dynamics between 2D Ekman theory and CROCO numerical solution [Estrade *et al.*, 2008, Marchesiello and Estrade, 2010].

```
# define INNERSHELF
```

CPP options:

```
# undef  OPENMP
# undef  MPI
# undef  NBQ
# define INNERSHELF_EKMAN
# define INNERSHELF_APG
# define SOLVE3D
# define UV_COR
# define ANA_GRID
# define ANA_INITIAL
# define AVERAGES
# define ANA_SSFLUX
# define ANA_SRFLUX
# define ANA_STFLUX
# define ANA_BSFLUX
# define ANA_BTFLUX
# define ANA_SMFLUX
# define NS_PERIODIC
# define OBC_WEST
# define SPONGE
# ifndef INNERSHELF_EKMAN
#  define UV_ADV
#  define SALINITY
#  define NONLIN_EOS
#  define LMD_MIXING
#  undef  GLS_MIXING
#  ifdef LMD_MIXING
#   define LMD_SKPP
#   define LMD_BKPP
#   define LMD_RIMIX
#   define LMD_CONVEC
#  endif
#  undef WAVE_MAKER_INTERNAL
#  ifdef WAVE_MAKER_INTERNAL
#   define ANA_BRY
#   define Z_FRC_BRY
#   define M2_FRC_BRY
#   define M3_FRC_BRY
#   define T_FRC_BRY
#  endif
# endif
# define NO_FRCFILE
```

Settings :

Results :

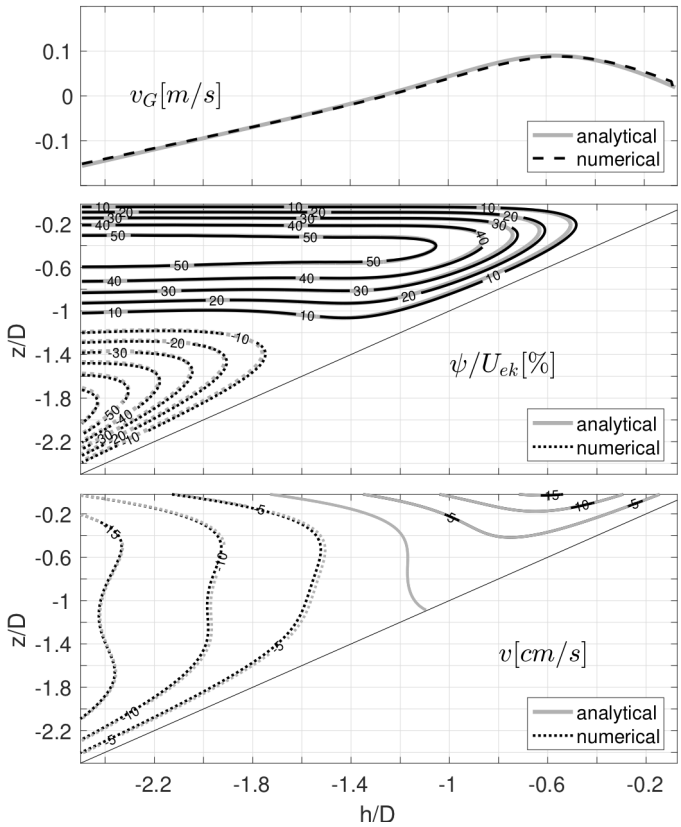


Fig. 19: INNERSHELF results : comparison with analytical solution

1.16.2.5 River Runoff

```
# define RIVER
```

CPP options:

```
# undef  OPENMP
# undef  MPI
# define SOLVE3D
# define UV_ADV
# define UV_COR
# define NONLIN_EOS
# define SALINITY
# define ANA_GRID
# define MASKING
# define ANA_INITIAL
# define ANA_SMFLUX
# define ANA_STFLUX
# define ANA_SSFLUX
# define ANA_SRFLUX
# define ANA_BTFLUX
# define ANA_BSFLUX
# define LMD_MIXING
# define LMD_SKPP
# define LMD_BKPP
# define LMD_RIMIX
# define LMD_CONVEC
# define PSOURCE
# define ANA_PSOURCE
# define NS_PERIODIC
# undef  FLOATS
# ifdef FLOATS
#  define RANDOM_WALK
#  ifdef RANDOM_WALK
#   define DIEL_MIGRATION
#   define RANDOM_VERTICAL
#   define RANDOM_HORIZONTAL
#  endif
# endif
# define NO_FRCFILE
```

Settings :

Results :

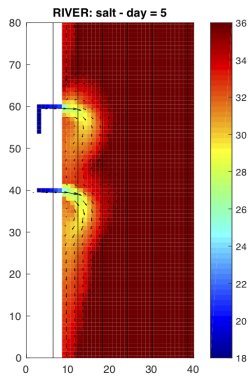


Fig. 20: RIVER results : river plume

1.16.2.6 Gravitational/Overflow

```
# define OVERFLOW
```

CPP options:

```
# undef  OPENMP
# undef  MPI
# define UV_ADV
# define UV_COR
# define UV_VIS2
# define TS_DIF2
# define TS_MIX_GEO
# define SOLVE3D
# define ANA_GRID
# define ANA_INITIAL
# define ANA_SMFLUX
# define ANA_STFLUX
# define ANA_BTFLUX
# define NO_FRCFILE
```

Setting :

Results :

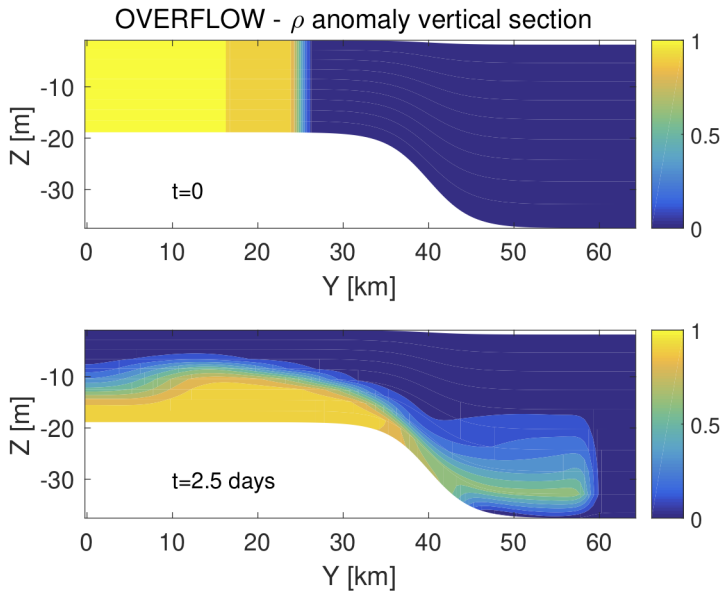


Fig. 21: OVERFLOW results : initial state (up) and density evolution (down)

1.16.2.7 Seamount

```
# define SEAMOUNT
```

CPP options:

```
# undef  OPENMP
# undef  MPI
# define UV_ADV
# define UV_COR
# define SOLVE3D
# define SALINITY
# define NONLIN_EOS
# define ANA_GRID
# define ANA_INITIAL
# define ANA_SMFLUX
# define ANA_STFLUX
# define ANA_SSFLUX
# define ANA_SRFLUX
# define ANA_BTFLUX
# define ANA_BSFLUX
# define NO_FRCFILE
```

Settings :

Results :

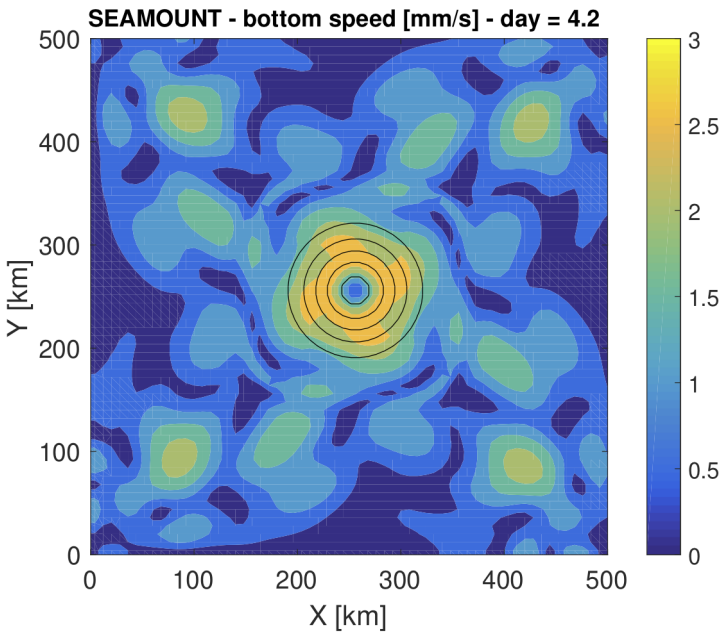


Fig. 22: SEAMOUNT results : bottom speed

1.16.2.8 Shelf front

```
# define SHELFRONT
```

CPP options:

```
# undef  OPENMP
# undef  MPI
# define UV_ADV
# define UV_COR
# define SOLVE3D
# define SALINITY
# define ANA_GRID
# define ANA_INITIAL
# define ANA_SMFLUX
# define ANA_STFLUX
# define ANA_SSFLUX
# define ANA_SRFLUX
# define ANA_BTFLUX
# define ANA_BSFLUX
# define EW_PERIODIC
# define NO_FRCFILE
```

Settings :

Results :

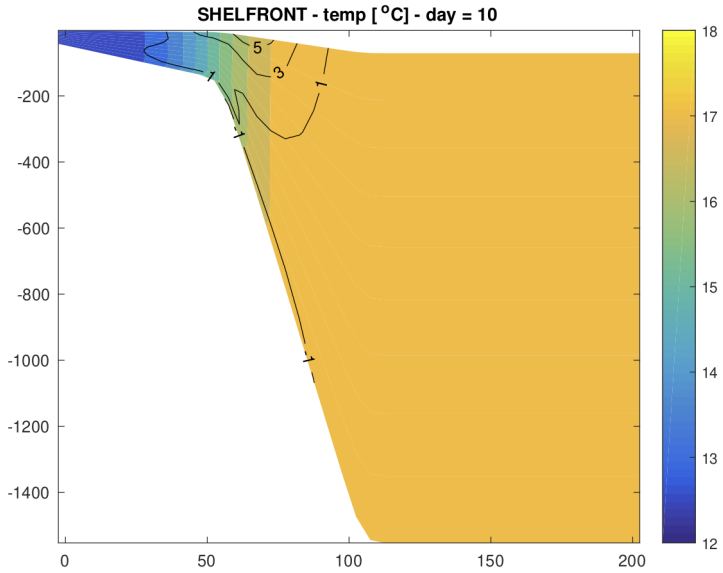


Fig. 23: SHELFRONT results : temerature profile

1.16.2.9 Equatorial Rossby Wave

This test problem considers the propagation of a Rossby soliton on an equatorial beta-plane, for which an asymptotic solution exists to the inviscid, nonlinear shallow water equations. In principle, the soliton should propagate westwards at fixed phase speed, without change of shape. Since the uniform propagation and shape preservation of the soliton are achieved through a delicate balance between linear wave dynamics and nonlinearity, this is a good context in which to look for erroneous wave dispersion and/or numerical damping.

The problem is nondimensionalized with: H = 40 cm, L=295 km, T = 1.71 days and U=L/T=1.981 m/s. Theorical propagation speed is 0.4 (0.395) so that at t=120, the soliton should be back to its initial position after crossing the periodic channel of length 48.

Boyd [1980]

```
# define SOLITON
```

CPP options:

```
# undef  OPENMP
# undef  MPI
# define UV_COR
# define UV_ADV
# define ANA_GRID
# define ANA_INITIAL
# define AVERAGES
# define EW_PERIODIC
# define ANA_SMFLUX
# define NO_FRCFILE
```

Settings :

Results :

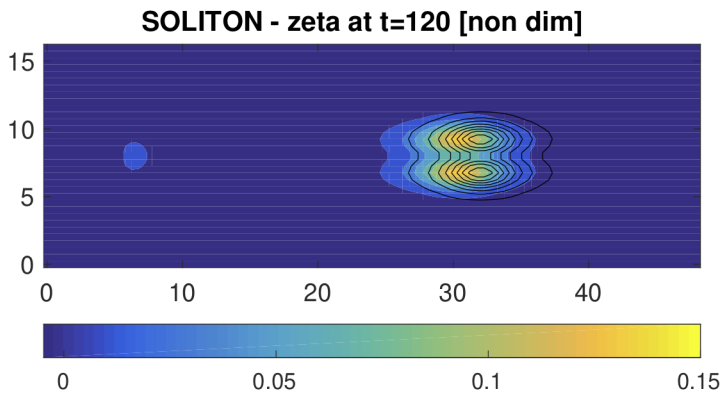


Fig. 24: SOLITON results : sea surface evolution

1.16.2.10 Thacker

Thacker [1981]

```
# define THACKER
```

CPP options:

```
# undef  OPENMP
# undef  MPI
# define THACKER_2DV
# define SOLVE3D
# define UV_COR
# define UV_ADV
# undef  UV_VIS2
# define WET_DRY
# define NEW_S_COORD
# define ANA_GRID
# define ANA_INITIAL
# define ANA_BTFLUX
# define ANA_SMFLUX
# define ANA_SRFLUX
# define ANA_STFLUX
# define NO_FRCFILE
```

Settings :

Results :

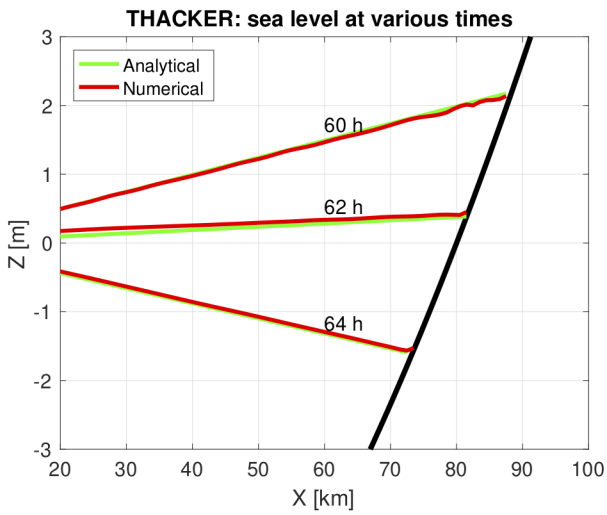


Fig. 25: THACKER results : elevation

1.16.2.11 Upwelling

```
# define UPWELLING
```

CPP options:

```
# undef  OPENMP
# undef  MPI
# define SOLVE3D
# define UV_COR
# define UV_ADV
# define ANA_GRID
# define ANA_INITIAL
# define AVERAGES
# define SALINITY
# define NONLIN_EOS
# define ANA_SSFLUX
# define ANA_SRFLUX
# define ANA_STFLUX
# define ANA_BSFLUX
# define ANA_BTFLUX
# define ANA_SMFLUX
# define LMD_MIXING
# define LMD_SKPP
# define LMD_BKPP
# define LMD_RIMIX
# define LMD_CONVEC
# define EW_PERIODIC
# define NO_FRCFILE
```

Settings :

Results :

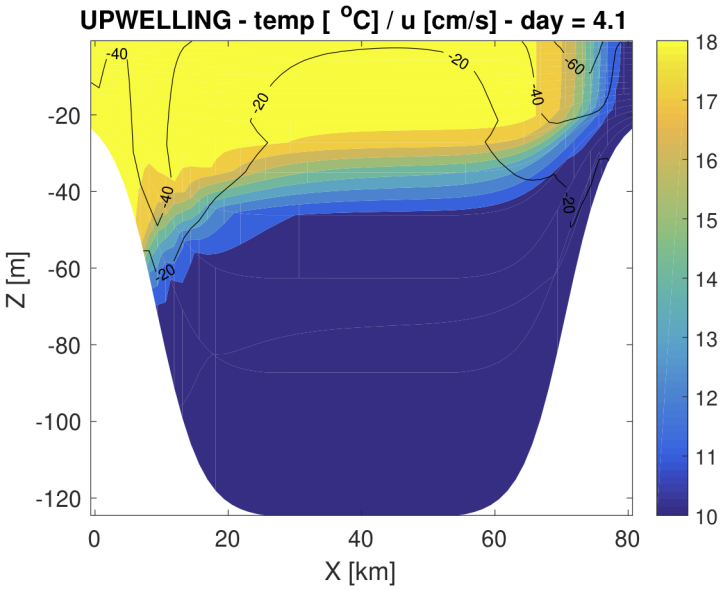


Fig. 26: UPWELLING results : temperature

1.16.2.12 Baroclinic Vortex

Free evolution of a baroclinic vortex (South West drift) that retains part of its initial axisymmetric shape as advective effects compensate for weak-amplitude Rossby-wave dispersion in its wake. 1-way and 2-way nesting were tested with this configuration [Debreu *et al.*, 2012, Penven *et al.*, 2006, McWilliams and Flierl, 1979].

```
# define VORTEX
```

CPP options:

```
# undef  OPENMP
# undef  MPI
# undef  AGRIF
# undef  AGRIF_2WAY
# undef  NBQ
# define SOLVE3D
# define UV_COR
# define UV_ADV
# define ANA_STFLUX
# define ANA_SMFLUX
# define ANA_BSFLUX
# define ANA_BTFLUX
# define ANA_VMIX
# define OBC_EAST
# define OBC_WEST
# define OBC_NORTH
# define OBC_SOUTH
# define SPONGE
# define ZCLIMATOLOGY
# define M2CLIMATOLOGY
# define M3CLIMATOLOGY
# define TCLIMATOLOGY
# define ZNUDGING
# define M2NUDGING
# define M3NUDGING
# define TNUDGING
# define NO_FRCFILE
```

Settings :

Results :

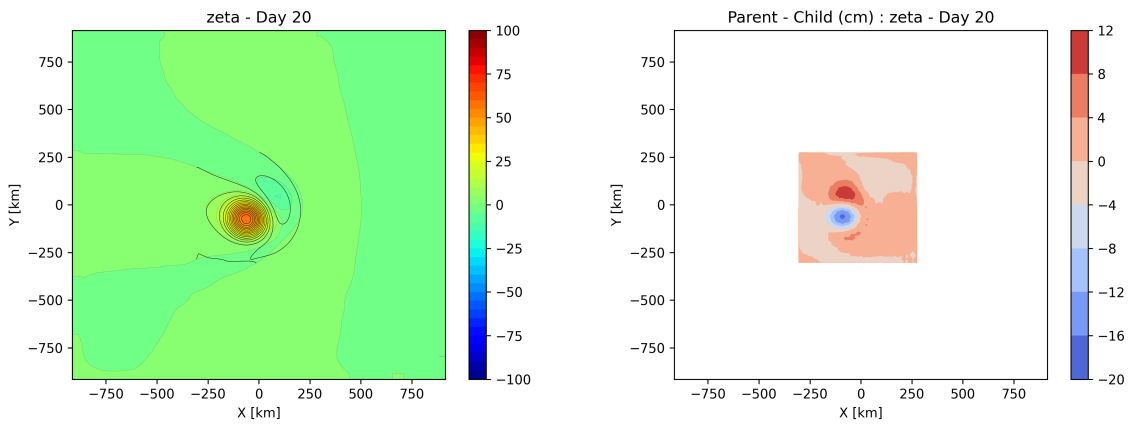


Fig. 27: VORTEX results : parent (grey lines) and child (black line) surface elevation (left) difference between parent and child grid (cm) (right)

1.16.2.13 Internal Tide

Internal Gravity Wave solution over a ridge [Di Lorenzo *et al.*, 2006].

```
# define INTERNAL
```

CPP options:

```
# undef  OPENMP
# undef  MPI
# define SOLVE3D
# define UV_COR
# define UV_ADV
# define BODYTIDE
# define ANA_GRID
# define ANA_INITIAL
# define ANA_BTFLUX
# define ANA_SMFLUX
# define ANA_SRFLUX
# define ANA_STFLUX
# define ANA_VMIX
# define EW_PERIODIC
# define NS_PERIODIC
# undef  INTERNALSHELF
# ifdef INTERNALSHELF
#  undef   EW_PERIODIC
#  define  OBC_EAST
#  define  OBC_WEST
#  define  SPONGE
#  define  ANA_SSH
#  define  ANA_M2CLIMA
#  define  ANA_M3CLIMA
#  define  ANA_TCLIMA
#  define  ZCLIMATOLOGY
#  define  M2CLIMATOLOGY
#  define  M3CLIMATOLOGY
#  define  TCLIMATOLOGY
#  define  M2NUDGING
#  define  M3NUDGING
#  define  TNUDGING
# endif
# define NO_FRCFILE
```

Settings :

Results :

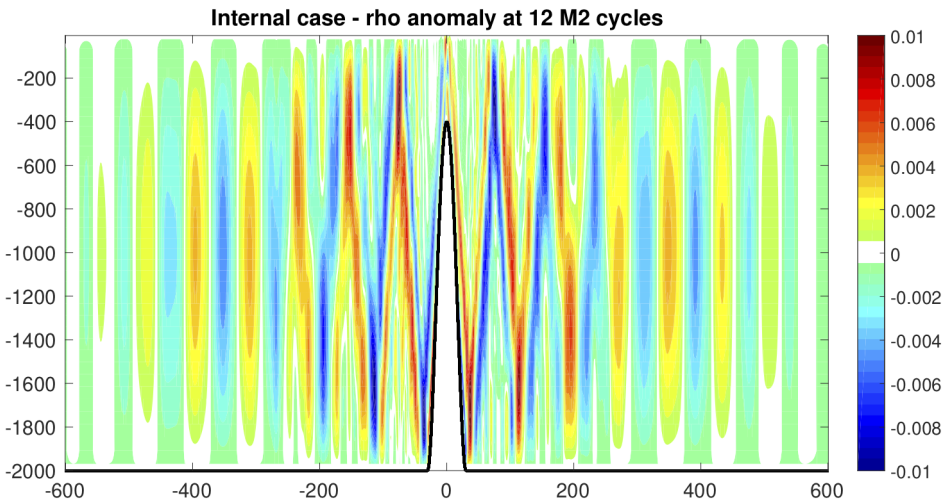


Fig. 28: INTERNAL results : generation of an internal wave

1.16.2.14 Internal Tide (COMODO)

Internal Gravity Wave solution over continental slope and shelf (COMODO test)

Pichon, A., 2007: Tests academiques de maree, Rapport interne n 21 du 19 octobre 2007, Service Hydrographique et Oceanographique de la Marine.

```
# define IGW
```

CPP options:

```
# define EXPERIMENT3
# undef  OPENMP
# undef  MPI
# undef  NBQ
# define NEW_S_COORD
# define TIDES
# define TIDERAMP
# define SSH_TIDES
# define UV_TIDES
# define SOLVE3D
# define UV_ADV
# define UV_COR
# define UV_VIS2
# undef  VADV_ADAPT_IMP
# define SPHERICAL
# define CURVGRID
# define ANA_INITIAL
# define ANA_VMIX
# define ANA_SMFLUX
# define ANA_STFLUX
# define ANA_SRFLUX
# define ANA_SSFLUX
# define ANA_BTFLUX
# define ANA_BSFLUX
# define NS_PERIODIC
# define OBC_EAST
# define OBC_WEST
# undef  SPONGE
# define ANA_SSH
# define ANA_M2CLIMA
# define ANA_M3CLIMA
# define ANA_TCLIMA
# define ZCLIMATOLOGY
# define M2CLIMATOLOGY
# define M3CLIMATOLOGY
# define TCLIMATOLOGY
# define M2NUDGING
# define M3NUDGING
# define TNUDGING
# undef  ONLINE_ANALYSIS
```

Settings :

Results :

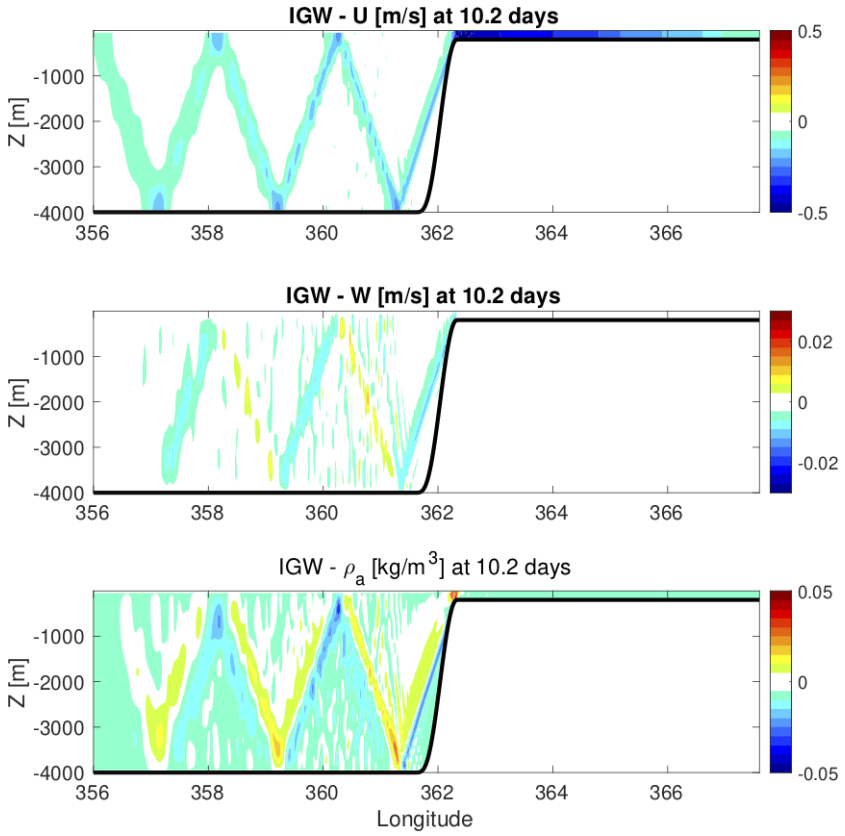


Fig. 29: IGW results : internal gravity waves generation

1.16.2.15 Baroclinic Jet

Effective resolution is limited by the numerical dissipation range, which is a function of the model numerical filters (assuming that dispersive numerical modes are efficiently removed). Soufflet *et al.* [2016] present a Baroclinic jet test case set in a zonally reentrant channel that provides a controllable test of a model capacity at resolving submesoscale dynamics.

A semi-idealized configuration in a periodic channel is set up to generate two dominant mechanisms of upper ocean turbulence: (i) surface density stirring by mesoscale eddies and (ii) fine scale instabilities directly energizing the submesoscale range. The setup consists of a flat reentrant channel of 500 km by 2000 km by 4000 m, centered around 30 deg of latitude on a β-plane (the Coriolis frequency is $1.10^{-4}s^{-1}$ at the center, $\beta = 1.610^{-11}m^{-1}s^{-1}$). Eastern/western boundary conditions are periodic while northern/southern conditions are closed. The initial density field is constructed with interior and surface meridional density gradients and associated geostrophic currents that are linearly unstable to both interior baroclinic and Charney instability modes. A linear stability analysis provides the exponential growth rate of unstable modes as a function of wavenumber. The two most unstable modes are clearly distinct in length scales on either side of the Rossby deformation radius (around 30 km in the center +/- 5 km from south to north). The interior geostrophic instability thus injects energy at mesoscale and Charney instability at submesoscale if resolution allows (2 km). The default resolution is 20 km (40 vertical levels) where only mesoscale instabilities are at work.

```
# define JET
```

CPP options:

```
# define ANA_JET
# undef  MPI
# undef  NBQ
# define SOLVE3D
# define UV_COR
# define UV_ADV
# define UV_VIS2
# ifdef ANA_JET
#  define ANA_GRID
#  define ANA_INITIAL
# endif
# define ANA_STFLUX
# define ANA_SMFLUX
# define ANA_BSFLUX
# define ANA_BTFLUX
# define ANA_VMIX
# define EW_PERIODIC
# define CLIMATOLOGY
# ifdef CLIMATOLOGY
#  define ZCLIMATOLOGY
#  define M2CLIMATOLOGY
#  define M3CLIMATOLOGY
#  define TCLIMATOLOGY
#  define ZNUDGING
#  define M2NUDGING
#  define M3NUDGING
#  define TNUDGING
#  define ROBUST_DIAG
#  define ZONAL_NUDGING
#  ifdef ANA_JET
#   define ANA_SSH
#   define ANA_M2CLIMA
#   define ANA_M3CLIMA
#   define ANA_TCLIMA
#  endif
```

(continues on next page)

(continued from previous page)

```
# endif
# define LMD_MIXING
# ifdef  LMD_MIXING
#  undef  ANA_VMIX
#  define ANA_SRFLUX
#  undef  LMD_KPP
#  define LMD_RIMIX
#  define LMD_CONVEC
# endif
# define NO_FRCFILE
```

Settings :

Results :

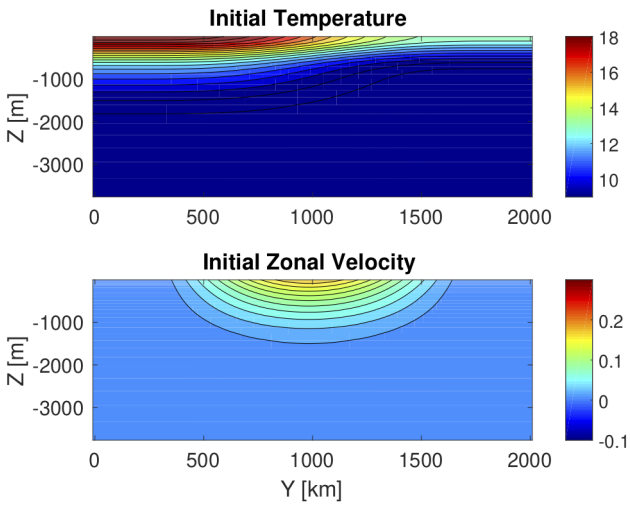


Fig. 30: JET results : initial state

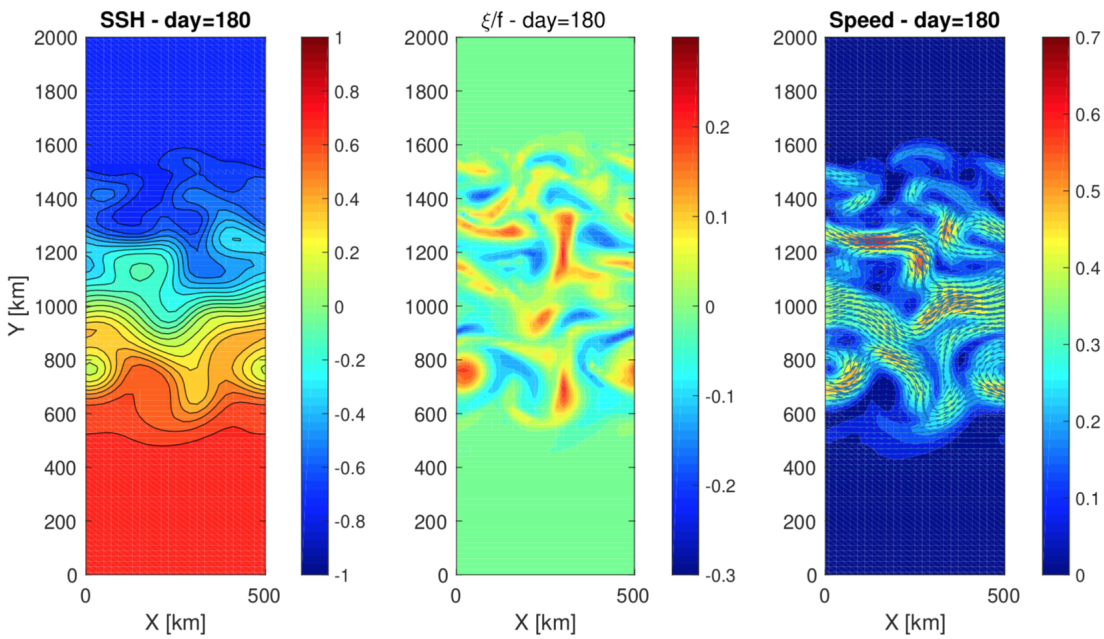


Fig. 31: JET results : results after 180 days

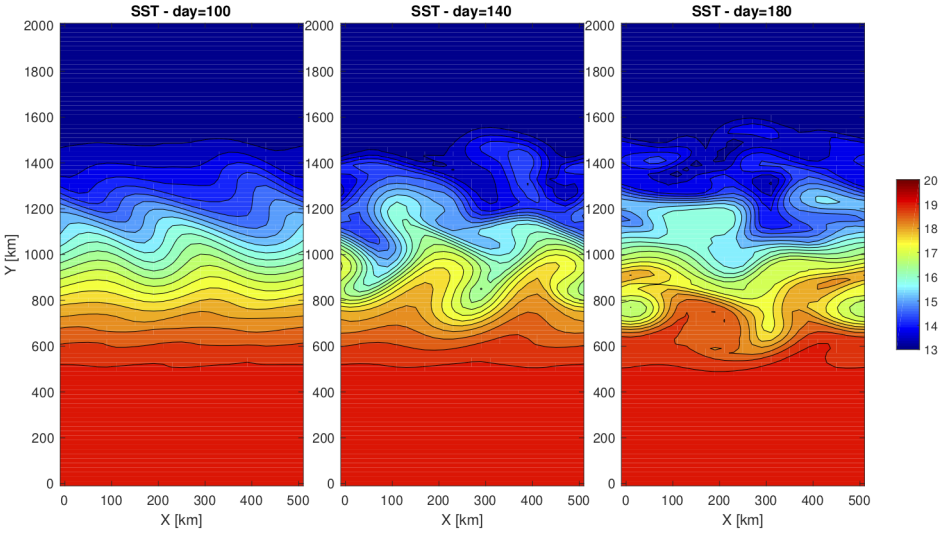


Fig. 32: JET results : vorticity evolutione

1.16.2.16 Plannar Beach

This test case is a littoral flow driven by obliquely incident waves on a plane beach with a uniform slope of 1:80. The model is forced by monochromatic waves computed with the WKB wave model [Uchiyama *et al.*, 2010] propagating offshore waves with 2 m significant wave height, a peak period of 10 s at an angle of 10° off the shore-normal direction. The horizontal extent of the domain is 1180 m in x (cross- shore), 140 m in y (alongshore) with grid spacings of dx = dy = 20 m. The model coordinates have a west-coast orientation, with the offshore open boundary located at x = 0. The resting depth h varies linearly from 12 m offshore, and is discretized with 20 uniform vertical sigma levels. Boundary conditions are alongshore periodicity, weetin-drying conditions at shore and open boundary conditions at the offshore boundary. Rotation is excluded with f = 0. There is no lateral momentum diffusion, stratification, nor surface wind/heat/freshwater fluxes. Breaking acceleration is given by the Church and Thornton [1993] formulation in the WKB model and wave-enhanced vertical mixing is computed by the first-order turbulent closure model, K-Profile Parameterization (KPP).

```
# define SHOREFACE
```

CPP options:

```
# undef  OPENMP
# undef  MPI
# define SOLVE3D
# define UV_ADV
# undef  MASKING
# define WET_DRY
# define NEW_S_COORD
# define ANA_GRID
# define ANA_INITIAL
# define ANA_SMFLUX
# define ANA_STFLUX
# define ANA_SSFLUX
# define ANA_SRFLUX
# define ANA_SST
# define ANA_BTFLUX
# define NS_PERIODIC
# define OBC_WEST
# define SPONGE
# define MRL_WCI
# ifdef MRL_WCI
#  undef  WAVE_OFFLINE
#  ifndef WAVE_OFFLINE
#   define WKB_WWAVE
#   define WKB_OBC_WEST
#   define WAVE_FRICTION
#   undef  WAVE_ROLLER
#   undef  MRL_CEW
#  endif
# endif
# define LMD_MIXING
# define LMD_SKPP
# define LMD_BKPP
# undef  BBL
# undef  SEDIMENT
# ifdef SEDIMENT
#  define TCLIMATOLOGY
#  define TNUDGING
#  define ANA_TCLIMA
# endif
```

Settings :

Results :

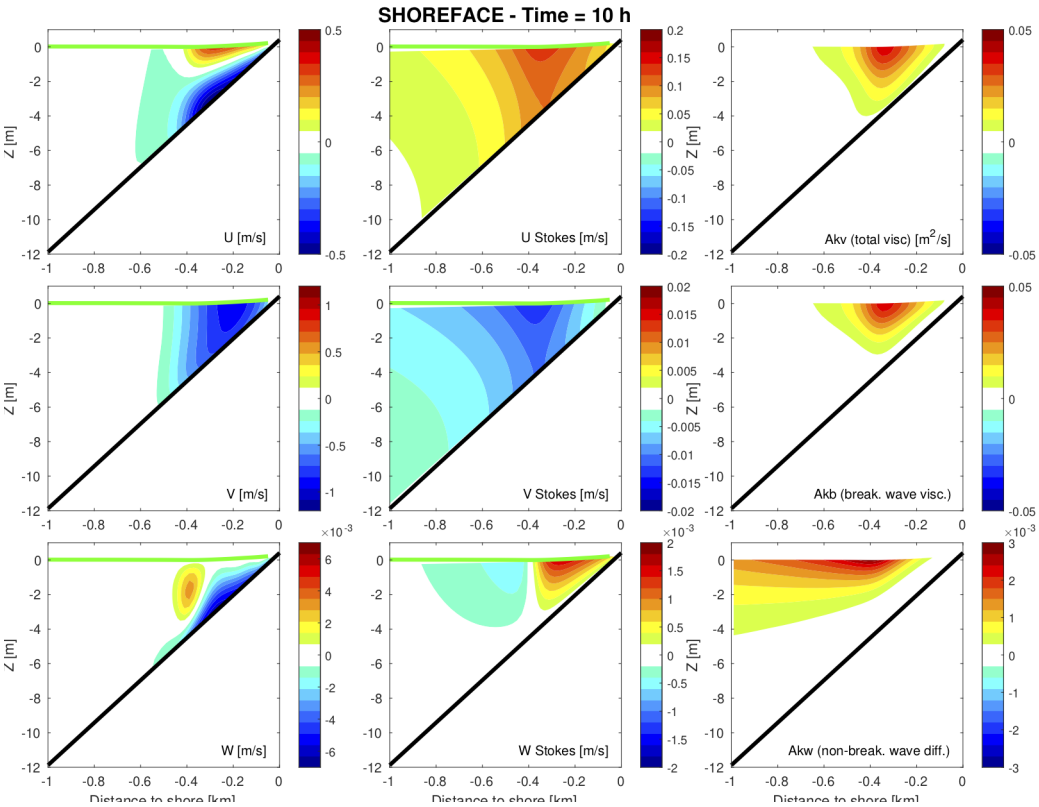


Fig. 33: SHOREFACE results : Eulerian velocities (left), Stokes velocities (center), Vertical mixing (right)

1.16.2.17 Rip Current

Rip currents are strong, seaward flows formed by longshore variation of the wave-induced momentum flux. They are responsible for the recirculation of water accumulated on a beach by a weaker and broader shoreward flow. 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 Weir *et al.* [2011] but in the 3D case. 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.*, 2010]. 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.

Related CPP options:

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  BISCA
# undef  RIP_TOPO_2D
# undef  GRANDPOPO
# ifdef GRANDPOPO
#  define RIP_TOPO_2D
# endif
# undef ANA_TIDES
# undef  OPENMP
# undef  MPI
# define SOLVE3D
# define NEW_S_COORD
# define UV_ADV
# 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
#  define FRC_BRY
#  define ANA_BRY
#  define Z_FRC_BRY
#  define M2_FRC_BRY
#  define M3_FRC_BRY
#  define T_FRC_BRY
```

(continues on next page)

(continued from previous page)

```
#  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_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
# ifdef SEDIMENT
#  define SUSPLOAD
```

(continues on next page)

(continued from previous page)

```
#  define BEDLOAD
#  undef  MORPHODYN
# endif
# undef  DIAGNOSTICS_UV
```

Settings :

Results :

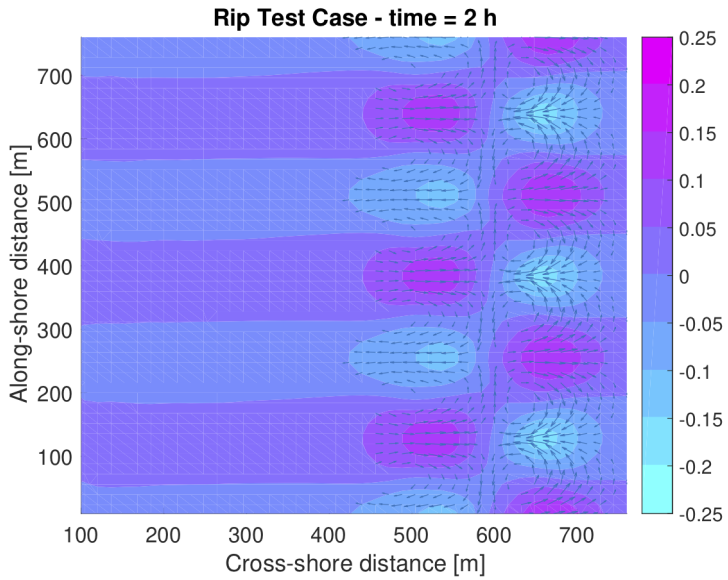


Fig. 34: RIP results : velocity

1.16.2.18 Flash Rips

Rip currents in the littoral zone are essential for exchanges between the continent and the ocean, as they can extend seaward to the continental shelf. There are two main types: channeled (bathymetric) and transient (flash) rip currents. Flash rips are purely wave-induced processes that can occur on uniform bathymetry along the coast. Over the last decade, the process of generating flash rips by short-crested waves, referred to as the Peregrine process, has become the current consensus in the nearshore community. In this case, the generation of flash rips and associated surfzone eddies requires explicit wave resolution rather than wave-averaged equations (RIP test case). The simulation of flash rips with CROCO is described in Marchesiello *et al.* [2021] — along with a new species of smaller rip currents, called mini-rips, arising from vertical shear instability of the undertow.

Settings :

The FLASH_RIP test case presents an idealized simulation of flash rips (and mini-rips) generated by wave directional spread over a longshore-uniform bathymetry. It is described in Treillou et al. (2025), as a simplification of the 2009 Imperial Beach experiment IB09 (Hally-Rosendahl and Feddersen 2016), with waves normal to the coast and zero wind stress.

The size of the computational domain is 300 m in both cross-shore and alongshore directions. The bathymetry is a synthethic representation of Imperial Beach. The model is forced by a JONSWAP wave spectrum with parameters : significant wave height Hs = 0.7 m, peak period Tp = 13 s and peak enhancement factor gamma = 3.3. The mean wave angle is zero and and the directional spread is 30 deg. The northern and southern boundaries are periodic (the wave angle around the mean is corrected to preserve periodicity; see Treillou *et al.* [2024]). The horizontal resolution is 1.5 m (a relatively coarse resolution for wave-resolving simulations) with 8 vertical levels. The timestep is 50 ms with 5 substeps to resolve fast dynamics. The simulation lasts about 15 minutes and has time to spinup and generate a few rip currents and numerous mini-rips.

Input parameters :

Parameters to run FLASH_RIP are in **croco.in.Flashrip** available in the TEST_CASES directory. A basic Matlab plotting code (plot_flashrip.m) is also available in this directory. Input parameters in **croco.in.Flashrip** for the wavemaker (see formulation in: *Wavemaker*) are set under keyword **wave_maker** :

```
wave_maker:  amp [m], prd [s], ang [deg], ang spread [deg], freq spread␣
↪(gamma)
             0.24     13.       0.         30.               3.3
```

`amp`: wave amplitude ($a_m = \sqrt{H_s/8}$)

`prd`: peak period

`ang`: mean wave angle

`ang spread`: directional spread (σ_θ)

`freq spread`: peak enhancement factor of JONSWAP spectrum

> **Note**
>
> The directional spread is essential for generating short-crested waves (and flash rips). The mean crest length should vary as $\lambda_p/(2\sin\sigma_\theta)$, with λ_p the peak wavelength.

> **Note**
>
> if WAVE_MAKER_SPECTRUM is not defined (see below), the wavemaker produces a monochromatic wave.

Related CPP options:

FLASH_RIP	Idealized longshore-uniform Imperial Beach (CA)
WAVE_MAKER & NBQ	Wavemaker for wave resolving nonhydrostatic case
WAVE_MAKER_SPECTRUM	Wave spectrum with frequency spread (default: JONSWAP)
WAVE_MAKER_DSPREAD	Add directional spread in wave spectrum
OBC_SPECIFIED_WEST	Specify wave fields at the offshore boundary (not critical)
NO_TRACER & NO_TEMPERATURE	Save time by avoiding calculation of tracer equations

CPP options:

```
# define FLASH_RIP
```

```
# define MPI
# undef  NC4PAR
# define SOLVE3D
# define NEW_S_COORD
# define UV_ADV
# define NO_TRACER
# define NO_TEMPERATURE
# undef  PASSIVE_TRACER
# undef  ANA_TIDES
# define NBQ
# define NBQ_PRECISE
# define LIMIT_BSTRESS
# 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
# define GLS_KOMEGA
# define NS_PERIODIC
# define OBC_WEST
# define OBC_SPECIFIED_WEST
# define SPONGE
# define FRC_BRY
# define ANA_BRY
# define Z_FRC_BRY
# define M2_FRC_BRY
# define M3_FRC_BRY
# define T_FRC_BRY
# define WET_DRY
# define ANA_GRID
# define ANA_INITIAL
# define ANA_SMFLUX
# define ANA_STFLUX
# define ANA_SSFLUX
# define ANA_SRFLUX
# define ANA_SST
# define ANA_BTFLUX
# define AVERAGES
# define AVERAGES_K
# undef  DIAGNOSTICS_EDDY
# undef  CVTK_DEBUG
```

Results :

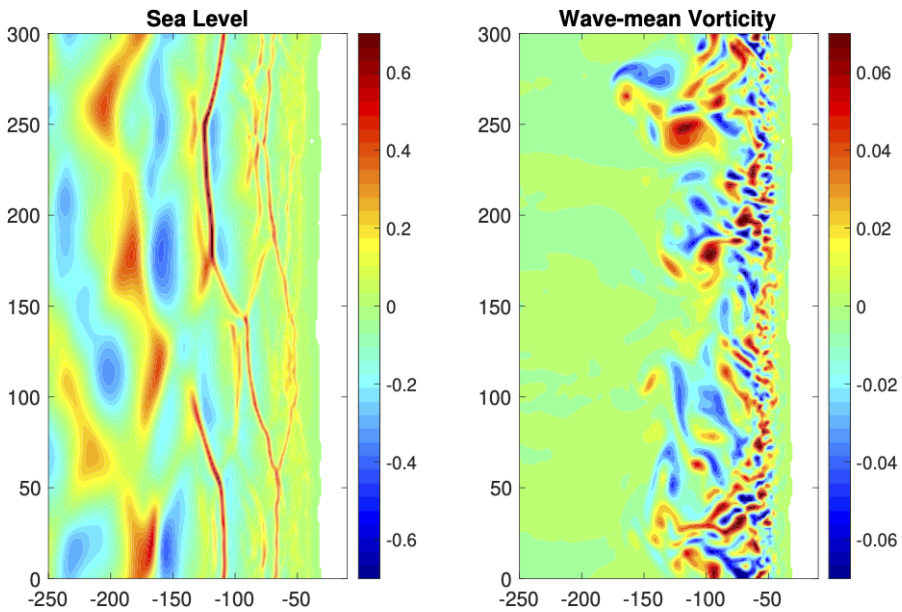


Fig. 35: FLASHRIP results : sea level and wave-mean vorticity

1.16.2.19 Sandbar

This test case is part of an effort to develop a comprehensive 3D nearshore model that predicts onshore and offshore sandbar migrations under storm and post-storm conditions, without the need to modify the model setting parameters. In this test, we attempt to reproduce the results of sandbar migration experiments, the European Large Installation Plan (LIP) experiments, which were carried out at full scale in Delft Hydraulics's Delta Flume (Roelvink and Reniers, 1995). Hydrostatic wave-averaged simulations of LIP-1B (erosion) and LIP-1C (accretion) using CROCO are described in Shafiei et al. (2022), while wave-resolving simulations are in Marchesiello *et al.* [2021] for hydrodynamics and Marchesiello *et al.* [2022] for morphodynamics.

In LIP, three types of experiments were carried out under different types of irregular waves, which subsequently resulted in a stable (1A), erosive (1B), and accretive (1C) beach state. The initial profile is linear in LIP-1A, with a slope of 1:30 and consisting of a median grain size of 0.22 mm. The final profile of LIP-1A was used as the initial profile of LIP-1B and the final profile of LIP-1B as the initial profile of LIP-1C. The wave conditions were a JONSWAP narrow-banded random wave spectrum with a peak enhancement factor of 3.3 and characteristic wave height and period: Hs =1.4m,Tp =5s (LIP-1B) and Hs =0.6m,Tp =8s (LIP-1C).Under this wave forcing, the sandbar developed during LIP-1B, increasing in height and migrating in the offshore direction. Under the accretive conditions of LIP-1C, the bar migration reversed to the onshore direction. For validation of currents and sand concentration, we consider the time 8 hours after initialization in experiment 1B and 7 hours in 1C. The LIP-1B and LIP-1C experiments lasted 18 and 13 hours, respectively. In both cases, the model was run for one hour with a morphological acceleration factor equal to 18 and 13 respectively.

The model can be run using wave-averaged equations in hydrostatic mode or wave-resolving nonhydrostatic equations.

1.16.2.19.1 Wave-averaged solution (default)

Here, wave-averaged equations are used that require parametrization of wave effects on morphodynamics. Bedload nonlinear wave-related transport is parametrized with the SANTOSS formulation, which follows the wave half-cycle concept to account for wave asymmetry and skewness. LIP1b and LIP1c experiments are conducted sequentially, meaning that the final bathymetry of LIP1b is the initial bathymetry of LIP1c. The numerical domain is 200 meters long and 4.1 m deep. The numerical domain is discretized using a uniform grid with horizontal resolution of 1.5 m and the number of vertical layers is 20 throughout the domain (the heights of the cells in the deep region and around the bar are about 21 cm and 5 cm respectively).

For wave forcing, CROCO is fully coupled to built-in ray-theory spectrum-peak wave propagation model. The offshore wave height is forced at the model boundary with values of the experiments. The resulting Dean number $\Omega = H_s/T_pW_s$ clearly differentiates the erosive and accretive conditions. Apart from the forcing conditions, all other wave model parameters are the same for both cases.

For the sediment transport model, the main calibration parameters to be tuned in the suspended load model are: the settling velocity $W_s = 25$ mm/s; the critical bed shear stress $\tau_{CE} = 0.18\,\mathrm{N/m^2}$; and erosion rate $E_0 = 0.001\,\mathrm{kg/m^2/s}$. For bed roughness, the bottom boundary layer model (BBL) uses empirical formulations for sand mobilization based on grain size and wave statistics. For bedload transport, SANTOSS is implemented with only one calibration parameter: the bedload factor, which is set to 0.5.

1.16.2.19.2 Wave-resolved solution (#define NBQ)

In this case, we do not rely on parametrization for the bottom boundary layer or bedload transport, as as skewed-asymmetric waves are resolved explicitly, but we make sure that the wave-boundary layer is resolved, and that the first vertical level is in a sheet flow layer (about 10 times the grain size). This is particularly important for the onshore bar migration phase. Note that in our formulation, the turbulence intensity (calculated with the closure model) affects the sediment resuspension. A numerical wave maker forces the JONSWAP spectrum of linear waves at the offshore boundary (as in the laboratory experiments).

Roelvink, J.A., Reniers, 1995. LIP 11D delta flume experiments : a dataset for profile model validation. WL / Delft Hydraulics.

Shafiei H., J. Chauchat, C. Bonamy, and P. Marchesiello, 2022: Numerical simulation of on-shore/off-shore sandbar migration using wave-cycle concept – application to a 3D wave-averaged oceanic model (CROCO), in preparation for Ocean Modelling.

```
# define SANDBAR
```

CPP options:

```
# define SANDBAR_OFFSHORE /* LIP-1B */
# undef  SANDBAR_ONSHORE  /* LIP-1C */
# undef  OPENMP
# undef  MPI
# undef  NBQ
# define SOLVE3D
# define UV_ADV
# define NEW_S_COORD
# define ANA_GRID
# define ANA_INITIAL
# define ANA_SMFLUX
# define ANA_STFLUX
# define ANA_SSFLUX
# define ANA_SRFLUX
# define ANA_SST
# define ANA_BTFLUX
# define OBC_WEST
# define SPONGE
# define WET_DRY
# ifndef NBQ /* ! NBQ */
#  define MRL_WCI
#  ifdef MRL_WCI
#   define WKB_WWAVE
#   define MRL_CEW
#   define WKB_OBC_WEST
#   define WAVE_ROLLER
#   define WAVE_FRICTION
#   define WAVE_BREAK_TG86
#   define WAVE_BREAK_SWASH
#   define WAVE_STREAMING
#   undef  WAVE_RAMP
#  endif
#  define GLS_MIXING
#  define GLS_KOMEGA
#  undef  LMD_MIXING
#  ifdef LMD_MIXING
#   define LMD_SKPP
#   define LMD_BKPP
#   define LMD_VMIX_SWASH
#  endif
#  define BBL
# else /* NBQ */
#  define MPI
#  define NBQ_PRECISE
#  define WAVE_MAKER
#  define UV_ADV
#  define UV_HADV_WENO5
#  define UV_VADV_WENO5
#  define W_HADV_WENO5
#  define W_VADV_WENO5
#  define GLS_MIXING_3D
#  define GLS_KOMEGA
#  define ANA_BRY
```

(continues on next page)

(continued from previous page)

```
#  define Z_FRC_BRY
#  define M2_FRC_BRY
#  define M3_FRC_BRY
#  define T_FRC_BRY
#  define AVERAGES
#  define AVERAGES_K
#  define DIAGNOSTICS_EDDY
# endif /* NBQ */
# define SEDIMENT
# ifdef SEDIMENT
#  define SUSPLOAD
#  define BEDLOAD
#  define MORPHODYN
#  define TCLIMATOLOGY
#  define TNUDGING
#  define ANA_TCLIMA
# endif
# undef  STATIONS
# ifdef STATIONS
#  define ALL_SIGMA
# endif
# undef  DIAGNOSTICS_TS
# ifdef DIAGNOSTICS_TS
#  define DIAGNOSTICS_TS_ADV
# endif
# define NO_FRCFILE
# undef  CVTK_DEBUG
```

Results :

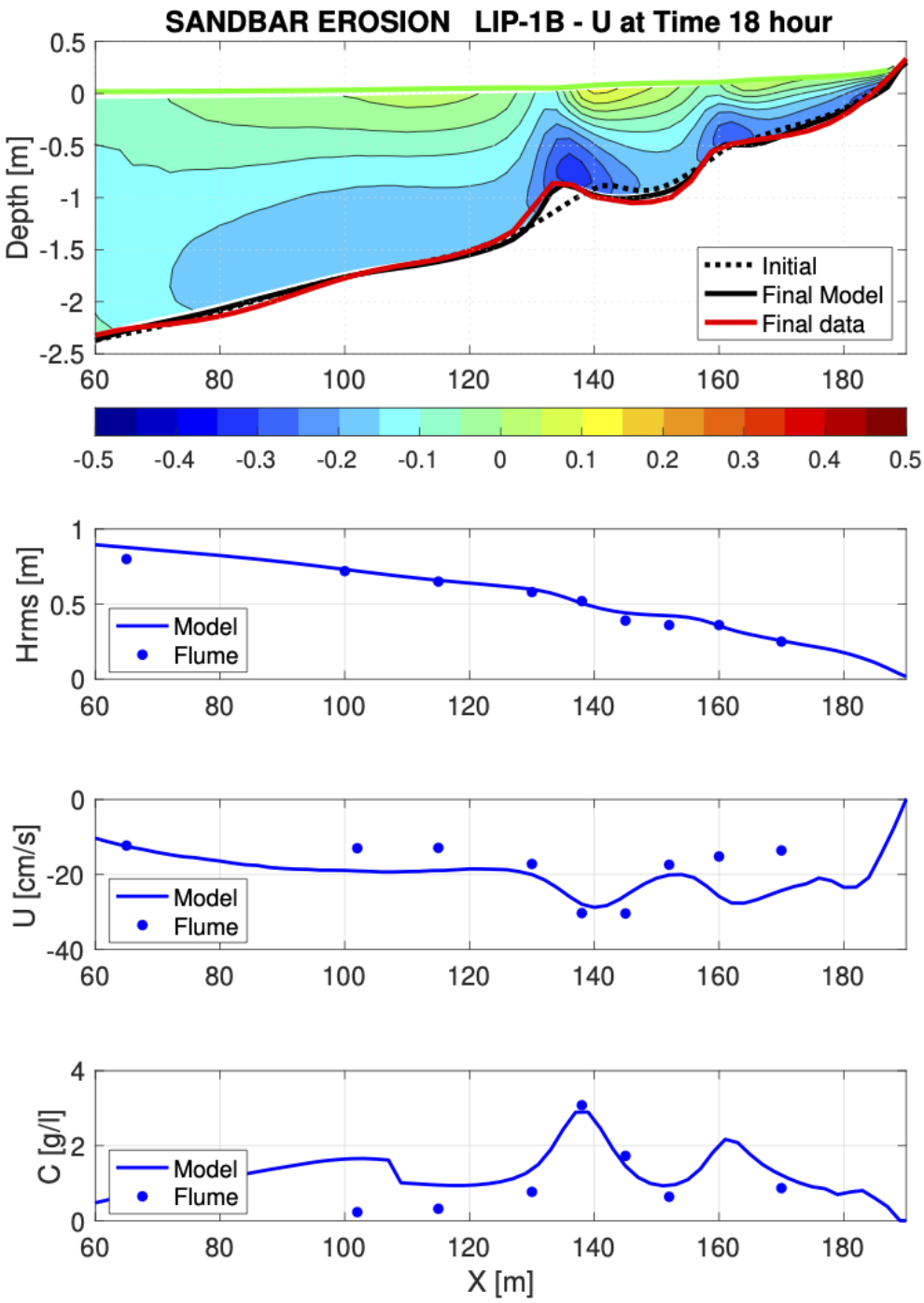


Fig. 36: SANDBAR results : validation of offshore sandbar migration against LIP-1B flume experiment

1.16.2.20 Swash

This test case addresses wave dynamics on a gently sloping laboratory beach (Globex experiment), using a wave-resolving configuration. The simulation is compared in Marchesiello *et al.* [2021] against GLOBEX experiments B3 in 2012 in the Scheldt flume of Deltares (Delft, the Netherlands), and described in Michallet *et al.* [2014] (parameters for B2 and A3 experiments are also available). The flume is 110 m long and contains a solid beach of 1:80 slope with its toe at 16.6 m from the wave maker. Experiments are run with a still water depth of 0.85 m and shoreline at x = 84.6 m. Second-order bichromatic waves (B3) are generated at the offshore boundary, with shore normal direction. The grid spacing is dx=12 cm with 10 vertical levels evenly spaced between the free surface and bottom. Best results are given with dx=1 cm, but the solution is moderately degraded with coarser horizontal resolution (dx=3, 6 and 12 cm), which shows good convergence properties. The model time step is dt = 2 ms. The minimum depth is 1 mm on the shore, the position of which varies with swash oscillation, relying on CROCO's wetting-drying scheme. For bottom drag, the logarithmic law of the wall is used with roughness length $z_0 = 0.0625$ mm.

```
# define SWASH
```

CPP options:

```
# define SWASH_GLOBEX_B3
# undef  SWASH_GLOBEX_B2
# undef  SWASH_GLOBEX_A3
# undef  OPENMP
# undef  MPI
# define SOLVE3D
# define AVERAGES
# define NBQ
# define NBQ_PRECISE
# define WAVE_MAKER
# define UV_ADV
# define UV_HADV_WENO5
# define UV_VADV_WENO5
# define W_HADV_WENO5
# define W_VADV_WENO5
# define GLS_MIXING_3D
# define GLS_KOMEGA
# define NEW_S_COORD
# define ANA_GRID
# define ANA_INITIAL
# define ANA_SMFLUX
# define ANA_STFLUX
# define ANA_SSFLUX
# define ANA_SRFLUX
# define ANA_SST
# define ANA_BTFLUX
# define OBC_WEST
# define OBC_SPECIFIED_WEST
# define FRC_BRY
# define ANA_BRY
# define Z_FRC_BRY
# define M2_FRC_BRY
# define M3_FRC_BRY
# define T_FRC_BRY
# define WET_DRY
# define NO_FRCFILE
```

Settings :

Results :

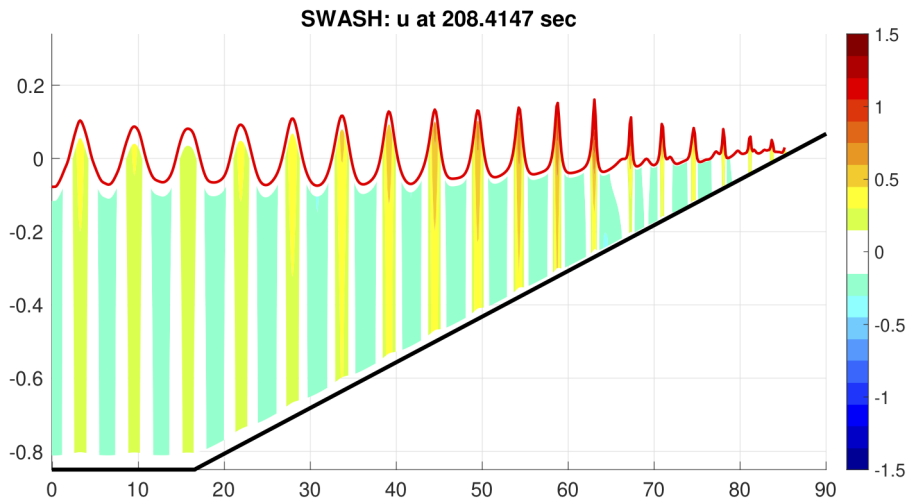


Fig. 37: SWASH results : Velocity and elevation

1.16.2.21 Tank

The non-hydrostatic solver is tested with several analytical solutions and laboratory experiments. The TANK test case simulates a two-dimensional, deepwater standing wave in a rectangular basin with a depth D and length l of 10 m. The oscillation is caused by a sinusoidal free-surface set-up at time=0. The model uses a uniform grid spacing of 0.2 m in the horizontal and vertical directions. From the dispersion relation ($\sigma = 2\pi/T = gk \tanh kD$, with $L = k/2\pi = 2l$ the wave length), the wave period is T = 3.6 s and phase speed is c = 5.6 m/s. With the hydrostatic assumption, the phase speed and frequency are higher (T = 2.0 s and c = 9.9 m/s). The simulations are compared to analytical solutions.

Chen [2003]

```
# define TANK
```

CPP options:

```
# undef  MPI
# define NBQ
# ifdef NBQ
#  define NBQ_PRECISE
# endif
# define SOLVE3D
# undef  UV_ADV
# define NEW_S_COORD
# define ANA_GRID
# define ANA_INITIAL
# define ANA_BTFLUX
# define ANA_SMFLUX
# define ANA_SRFLUX
# define ANA_STFLUX
# define NO_FRCFILE
```

Settings :

Results :

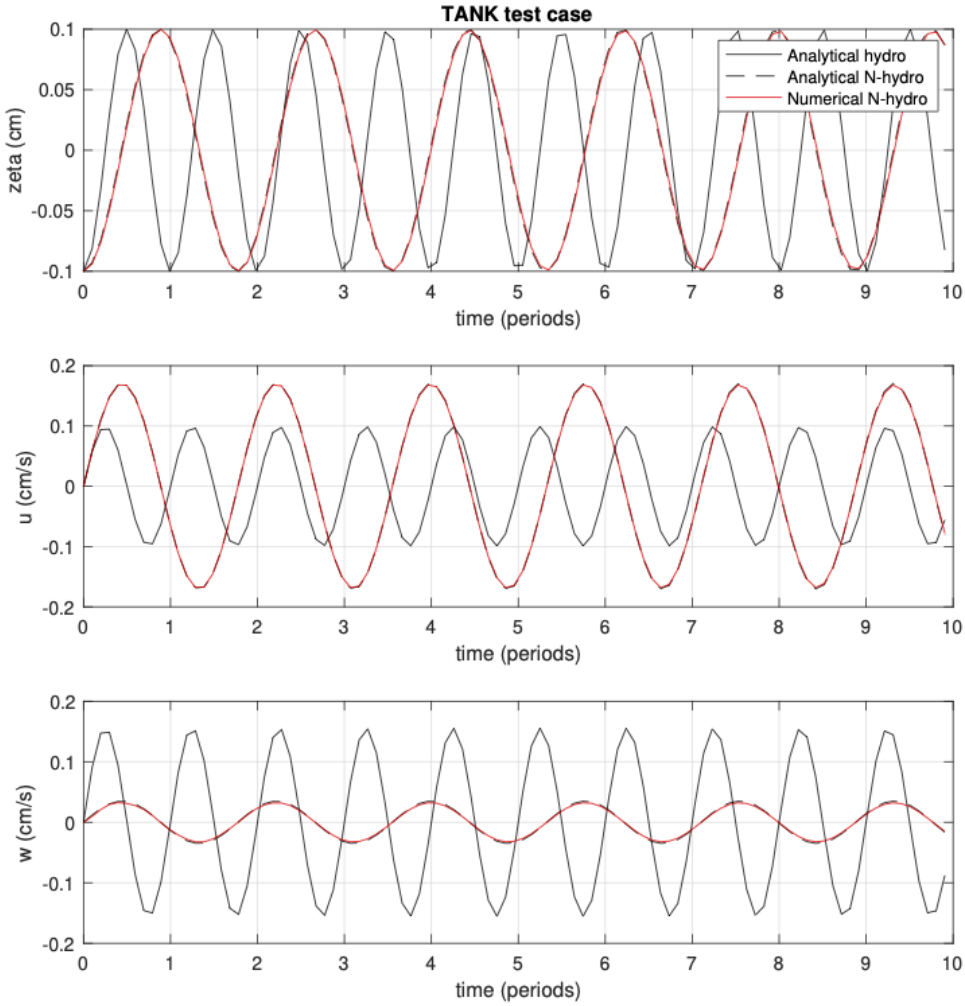


Fig. 38: TANK results : Comparison between analytical, hydrostatique and non-hydrostatique solutions

1.16.2.22 Acoustic wave

```
# define ACOUSTIC
```

CPP options:

```
# undef  MPI
# define NBQ
# ifdef NBQ
#  undef  NBQ_PRECISE
#  define NBQ_PERF
# endif
# undef  UV_VIS2
# define SOLVE3D
# define NEW_S_COORD
# define ANA_GRID
# define ANA_INITIAL
# define ANA_SMFLUX
# define ANA_STFLUX
# define ANA_SRFLUX
# define ANA_BTFLUX
# define NO_FRCFILE
```

1.16.2.23 Gravitational Adjustment

The goal of this test case, also known as Lock-Exchange experiment, is to evaluate different numerical advection schemes on representing the adiabatic process in a dam breaking experiment. At the initial time, a vertical density front separates two density classes. Adjustment occurs in which lighter water moves above heavier water [Shin *et al.*, 2004]. The model experiments are designed to reproduce the lock-exchange problem described in Ilıcak *et al.* [2012]. Analytical solutions to this problem exist and from Bernouilli's equation for an ideal fluid, the front propagates with speed $0.5\sqrt{gH\delta\rho/\rho_0}$. This speed may be slowed down by mixing between the two layers due to numerical diapycnal diffusion.

The setup is a closed, two-dimensional (x,z) domain with a constant depth of H = 20 m and a length of L = 64 km. At t = 0 the two initial densities that represent the two water masses are separated by a vertical barrier. The right and left halves of the domain have densities of 1020 and 1025 kg/m^3 respectively. To investigate the impact of the model resolution and the choice of advection scheme on spurious mixing, the model uses three different horizontal and vertical model grid spacings: coarse (dx=2 km; N=10); medium is default (dx=500 m, N=40); fine (dx=125 m, N=160).

A non-hydrostatique version can be run (#define NBQ) in a smaller domain of 3 m by 30 cm and resolution of 1 cm. In this case, Kelvin-Helmholtz instabilities develop along the front during the gravitational adjustment.

```
# define GRAV_ADJ
```

CPP options:

```
# undef  OPENMP
# undef  MPI
# undef  NBQ
# undef  XIOS
# define SOLVE3D
# define NEW_S_COORD
# define UV_ADV
# define TS_HADV_WENO5
# define TS_VADV_WENO5
# define UV_HADV_WENO5
# define UV_VADV_WENO5
# ifdef NBQ
#  define W_HADV_WENO5
#  define W_VADV_WENO5
# endif
# undef  UV_VIS2
# define ANA_GRID
# define ANA_INITIAL
# define ANA_SMFLUX
# define ANA_STFLUX
# define ANA_BTFLUX
# undef  PASSIVE_TRACER
# define NO_FRCFILE
# undef  CVTK_DEBUG
```

Settings :

Results :

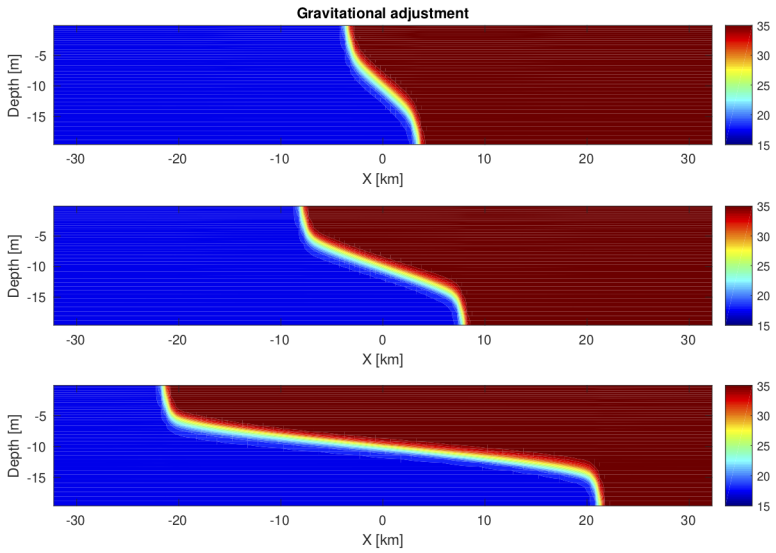


Fig. 39: GRAV_ADJ results : density front evolution for a medium resolution of 500m.

1.16.2.24 Internal Soliton

The non-hydrostatic solver is tested with several analytical solutions and laboratory experiments. The Internal Soliton test case is setup from the experiment of Horn *et al.* [2001]. It illustrates the processes acting on an interfacial basin-scale standing wave known as an internal seiche, neglecting the Earth's rotation. The propagation regimes depends on the ratio of the amplitude of the initial wave to the depth of the thermocline, and the ratio of the depth of the thermocline to the overall depth of the basin. In the present setup with moderate wave amplitude, the degeneration mechanism of the basin-scale internal wave is the generation of solitons by nonlinear steepening. As the wave steepens its horizontal lengthscale decreases until the dispersive terms can no longer be neglected. Eventually, a balance between nonlinear steepening and dispersion leads to the evolution of solitary waves, a process described by the Korteweg–de Vries (KdV) equation for the interfacial displacement η_i:

$$\frac{\partial \eta_i}{\partial t} + c_0 \frac{\partial \eta_i}{\partial x} + \alpha \eta_i \frac{\partial \eta_i}{\partial x} + \beta \frac{\partial^3 \eta_i}{\partial x^3}$$

The evolution of solitons is sensitive to the numerical damping associated with the choice of resolution, advection schemes and diffusion operators (implicit in the advection scheme or explicit).

The simulations can be compared with the laboratory experiments of Horn *et al.* [2001], which were carried out in a tank 6 m long and 29 cm deep. The two-layer fluid is given by a hyperbolic tangent density profile, which is rotated around the center of the basin to initiate the internal seiche at the basin scale. The resolution of the model is 10 cm horizontally and 4 mm vertically.

```
# define ISOLITON
```

CPP options:

```
# undef  MPI
# define NBQ
# undef  XIOS
# define SOLVE3D
# define NEW_S_COORD
# define UV_ADV
# define TS_HADV_WENO5
# define TS_VADV_WENO5
# define UV_HADV_WENO5
# define UV_VADV_WENO5
# define W_HADV_WENO5
# define W_VADV_WENO5
# undef  UV_VIS2
# undef  TS_DIF2
# define ANA_GRID
# define ANA_INITIAL
# define ANA_SMFLUX
# define ANA_STFLUX
# define ANA_BTFLUX
# undef  PASSIVE_TRACER
# define NO_FRCFILE
# undef  CVTK_DEBUG
```

Settings :

Results :

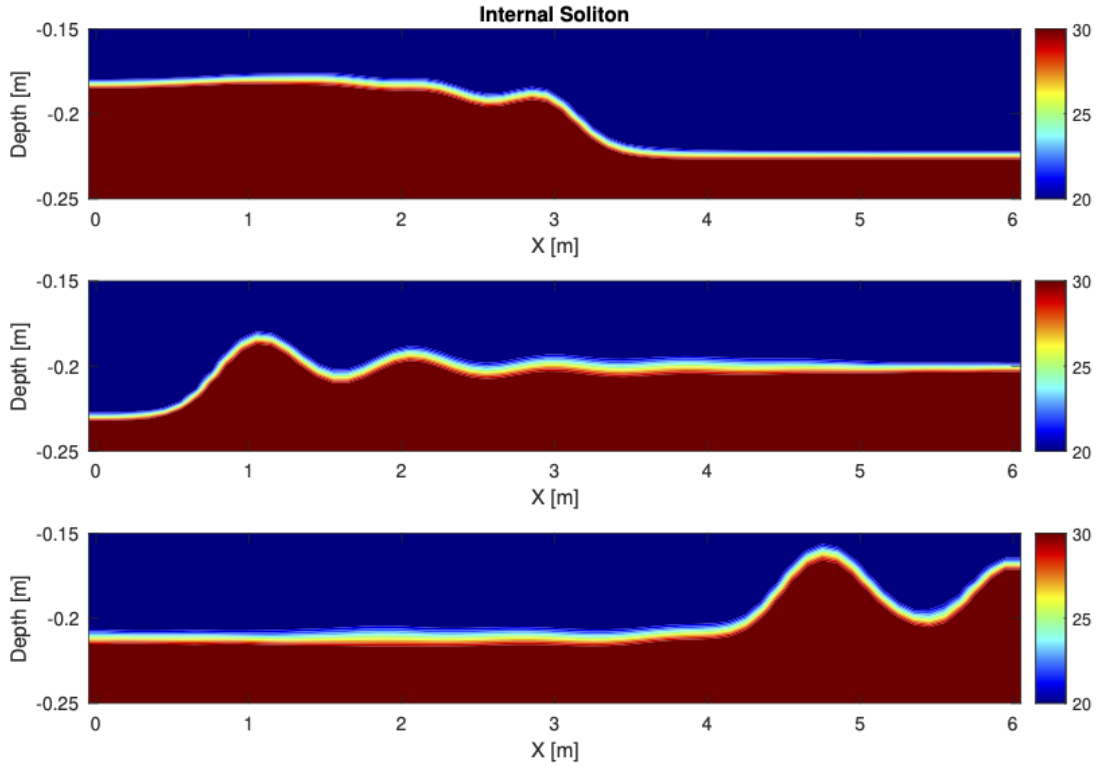


Fig. 40: ISOLITON results : generation of a train of internal solitons from a basin-scale internal seiche

1.16.2.25 Kelvin-Helmoltz Instability

This test case runs a Kelvin-Helmholtz instability between two fluid layers. It is part of experiments conducted with CROCO by Penney *et al.* [2020]. The numerical simulations are performed using the non-hydrostatic, non-Boussinesq version of CROCO. While numerical simulations of KH instabilities are often considered in a periodic domain with rigid lid conditionsfor the upper boundary, the implementation presented here uses a free-surface upper boundary, with periodic lateral boundary conditions in the x- and y-directions. The setup is two-dimensional (default) or three-dimensional, with initial density distribution defined as two constant-density layers separated by a strongly stratified pycnocline, with a weak stable background stratification superimposed. The configuration parameters are chosen so that the necessary criterion for stratified shear instability, Ri < 1/4, is satisfied. U(z), the initial background flow providing the shear, is defined by a hyperbolic tangent profile, with the upper layer moving leftward, and the lower layer rightward. Small amplitude perturbations are required to kickstart the instability.

The existence of a free surface and compressibility adds two dynamical processes (surface and acoustic waves) compared to more traditional studies in incompressible, unbounded or rigid lid flows. With the chosen configurations where the instability develops far from the vertical boundaries, the impact of these additional processes is negligible, but in certain circumstances, surface and acoustic waves may play a role in modifying the turbulent cascade.

The results are sensitive to the resolution (1 m by default) and the choice of advection schemes and diffusion operator (implicit in the advection schemes or explicit).

```
# define KH_INST
```

CPP options:

```
# undef  KH_INSTY
# undef  KH_INST3D
# define MPI
# define NBQ
# define NBQ_PRECISE
# undef  XIOS
# define SOLVE3D
# define NEW_S_COORD
# define UV_ADV
# define TS_HADV_WENO5
# define TS_VADV_WENO5
# define UV_HADV_WENO5
# define UV_VADV_WENO5
# define W_HADV_WENO5
# define W_VADV_WENO5
# undef  SALINITY
# undef  PASSIVE_TRACER
# define ANA_GRID
# define ANA_INITIAL
# define ANA_SMFLUX
# define ANA_STFLUX
# undef  ANA_SRFLUX
# define ANA_BTFLUX
# define ANA_SSFLUX
# define ANA_BSFLUX
# ifndef KH_INSTY
#  define EW_PERIODIC
# else
#  define NS_PERIODIC
# endif
# define NO_FRCFILE
```

Settings :

Results :

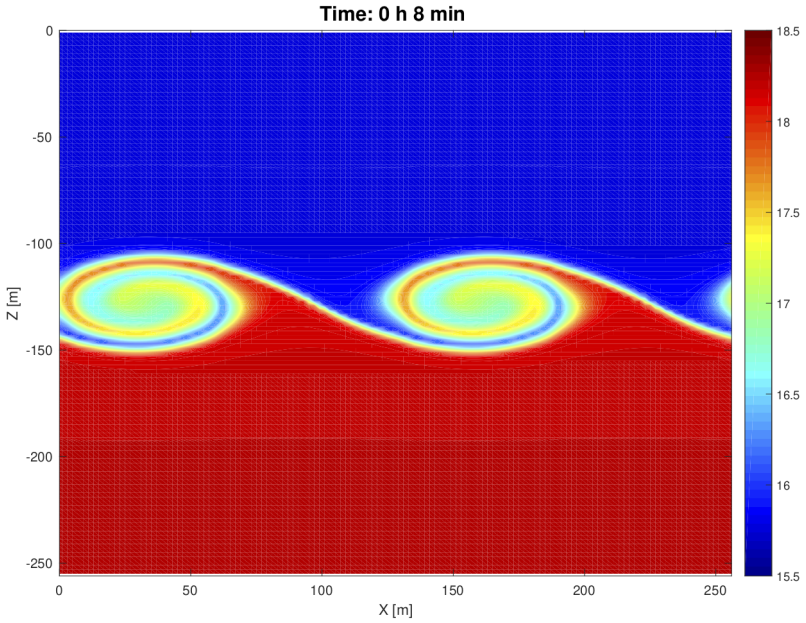


Fig. 41: KH_INST results : instability generation

1.16.2.26 Horizontal tracer advection

Test CROCO horizontal advection schemes for tracers

SOLID_BODY_ROT Example with spatially varying velocity DIAGONAL_ADV Constant advection in the diagonal SOLID_BODY_PER Example with a space and time-varying velocity

```
# define TS_HADV_TEST
```

CPP options:

```
# undef  SOLID_BODY_ROT
# undef  DIAGONAL_ADV
# define SOLID_BODY_PER

# undef  OPENMP
# undef  MPI
# undef  UV_ADV
# define NEW_S_COORD
# undef  UV_COR
# define SOLVE3D
# define M2FILTER_NONE
# define ANA_VMIX
# define ANA_GRID
# define ANA_INITIAL
# define ANA_SMFLUX
# define ANA_SRFLUX
# define ANA_STFLUX
# define ANA_BTFLUX
# define ANA_BSFLUX
# define ANA_SSFLUX
# define NO_FRCFILE
# define SALINITY
# define EW_PERIODIC
# define NS_PERIODIC

# define TS_HADV_UP3
# undef  TS_HADV_C4
# undef  TS_HADV_UP5
# undef  TS_HADV_WENO5
# undef  TS_HADV_C6
```

1.16.2.27 Sediment test cases

1.16.2.27.1 DUNE cases

Migration of a dune composed by different sand classes. Bedload process only

```
# define DUNE
```

CPP options:

```
# undef  OPENMP
# undef  MPI
# define M2FILTER_NONE
# define UV_ADV
# define NEW_S_COORD
# undef  UV_COR
# define SOLVE3D
# define ANA_GRID
# define ANA_INITIAL
# define ANA_SSFLUX
# define ANA_SRFLUX
# define ANA_STFLUX
# define ANA_BSFLUX
# define ANA_BTFLUX
# define ANA_SMFLUX
# define OBC_WEST
# define OBC_EAST
# define ANA_SSH
# define ZCLIMATOLOGY
# define ANA_M2CLIMA
# define M2CLIMATOLOGY
# define GLS_MIXING
# define MORPHODYN
```

=> For Mustang model, just add:

```
# define MUSTANG
# ifdef MUSTANG
#  define key_MUSTANG_V2
#  define key_MUSTANG_bedload
#  define key_tenfon_upwind
# endif
```

=> For USGS sediment model, just add:

```
# define SEDIMENT
# ifdef SEDIMENT
#  undef  SUSPLOAD
#  define BEDLOAD
#  undef  BEDLOAD_WENO5
#  define BEDLOAD_WULIN
#  define TAU_CRIT_WULIN
# endif
```

1.16.2.27.1.1 DUNE case (default)

Dune 2m. Sediment composed of two sand fractions. stratigraphy diagnostics

CPP options to add:

```
# undef  ANA_DUNE     /* Analytical test case (Marieu) */
# undef  DUNE3D       /* 3D example */
```

Results :

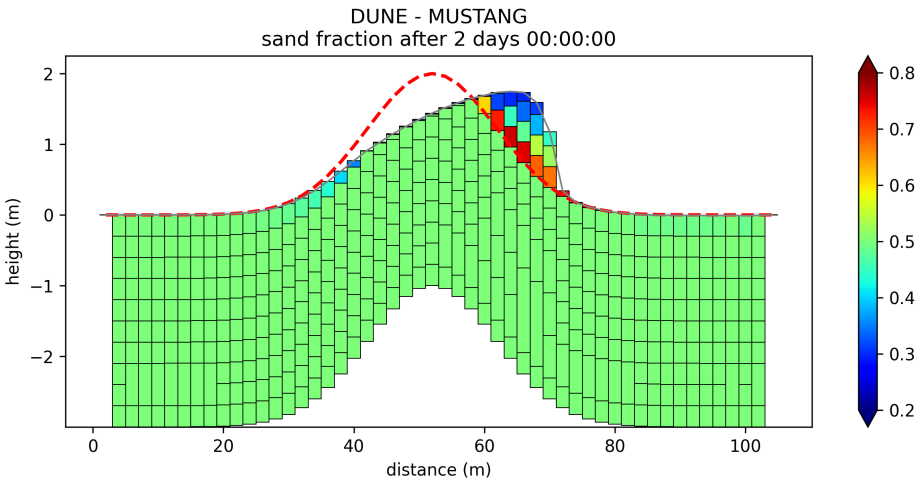


Fig. 42: Fine sand fraction after 10 days in the seabed. The red line indicates the initial position of the dune.

1.16.2.27.1.2 DUNE3D case

Extension of the DUNE case in 3D. Migration of a Sand bump forced by a barotropic constant flow

CPP options to add:

```
# undef  ANA_DUNE     /* Analytical test case (Marieu) */
# define  DUNE3D      /* 3D example */
```

Results :

1.16.2.27.1.3 ANA_DUNE case

Adaptation of the DUNE case. Migration of a sand dune with an analytical bedload formulation that provides an analytical solution for the dune evolution [Long *et al.*, 2008].

CPP options to add:

```
# define ANA_DUNE     /* Analytical test case (Marieu) */
# undef  DUNE3D        /* 3D example */
```

=> For Mustang model, just add:

```
# define MUSTANG
# ifdef MUSTANG
#  define key_MUSTANG_V2
#  define key_MUSTANG_bedload
#  define key_tenfon_upwind
# endif
```

=> For USGS sediment model, just add:

```
# define SEDIMENT
# ifdef SEDIMENT
#  undef  SUSPLOAD
```

(continues on next page)

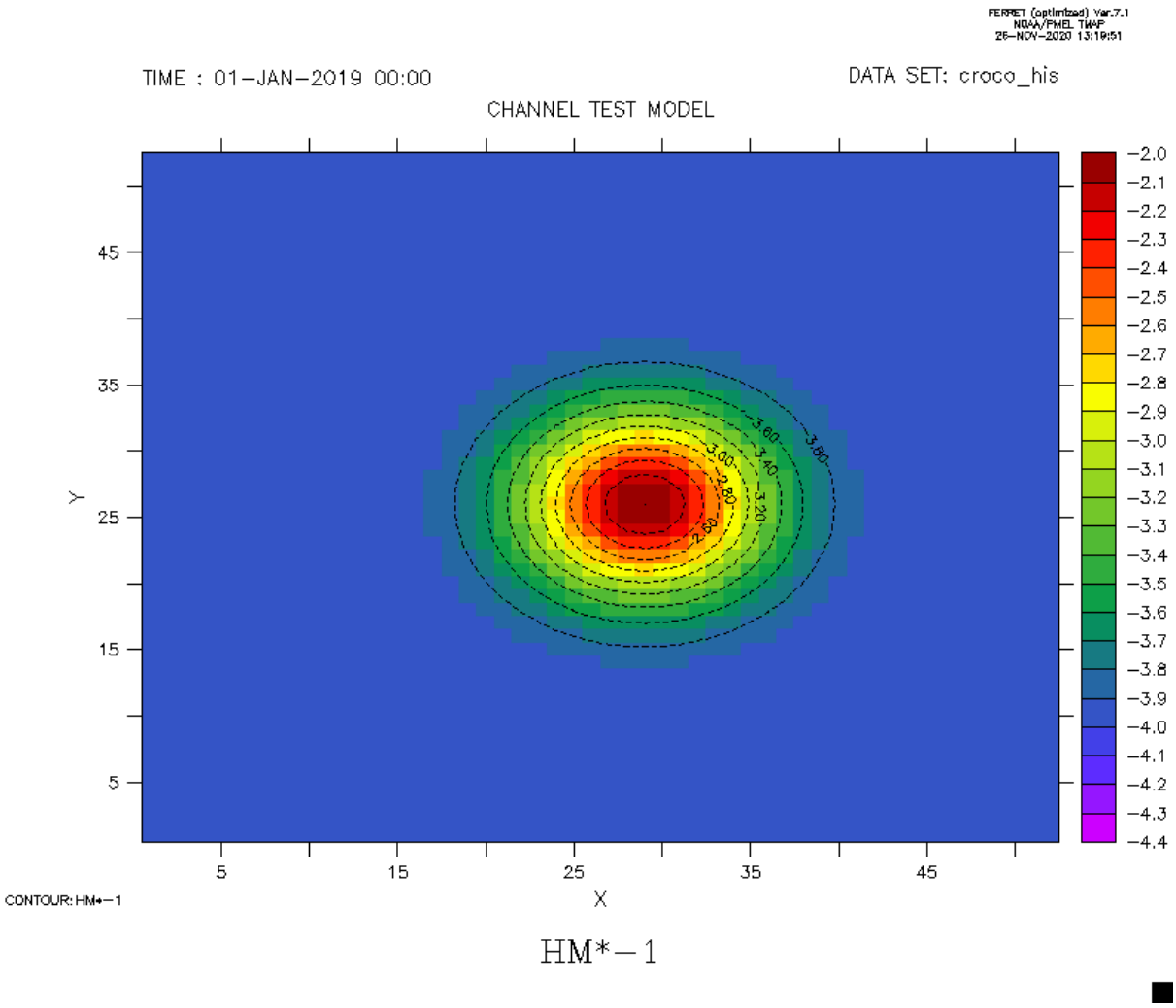


Fig. 43: Sand bump at initialization

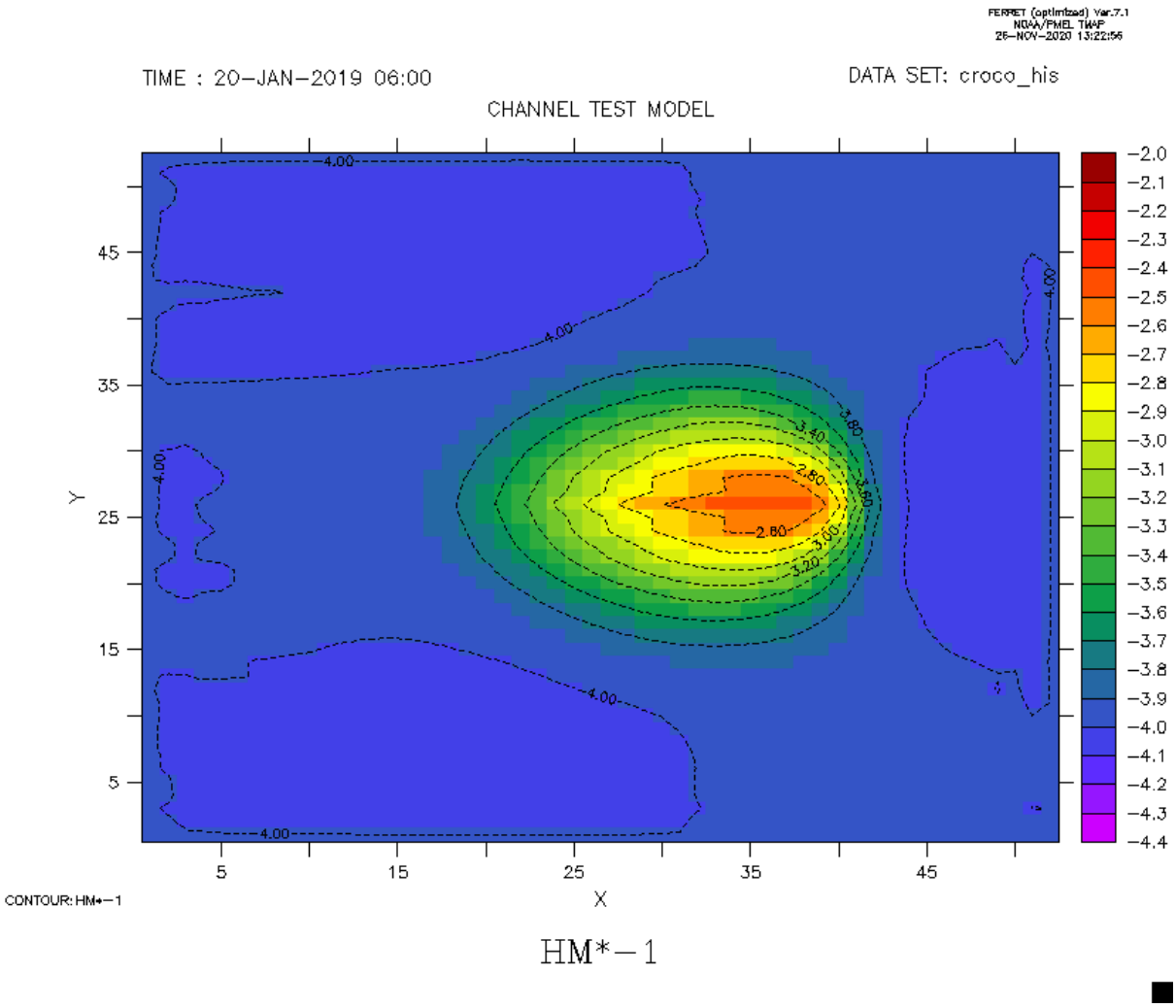


Fig. 44: Sand bump after 20 days

(continued from previous page)

```
#  define BEDLOAD
#  undef  BEDLOAD_WENO5
#  define BEDLOAD_MARIEU
# endif
```

Results :

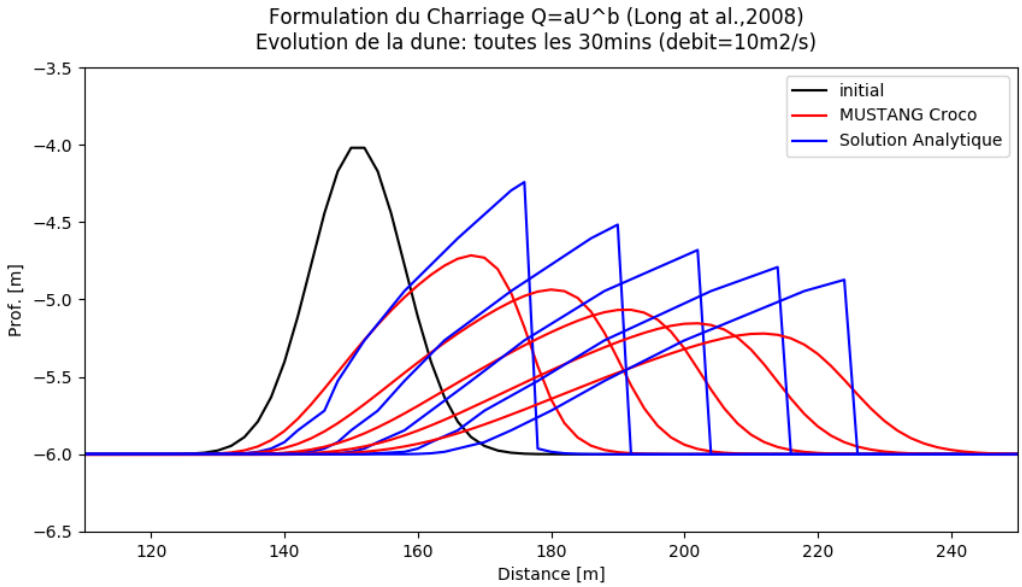


Fig. 45: Comparison between dune propagation (every 30 mins) simulated with CROCO/MUSTANG and computed using the analytical solution

1.16.2.27.2 SED_TOY cases

Single column test case

```
# define SED_TOY
```

CPP options:

```
# undef  OPENMP
# undef  MPI
# define NEW_S_COORD
# define SOLVE3D
# undef  NONLIN_EOS
# define SALINITY
# undef  UV_VIS2
# define ANA_GRID
# define ANA_INITIAL
# define ANA_SMFLUX
# define ANA_SRFLUX
# define ANA_STFLUX
# define ANA_SSFLUX
# define ANA_BTFLUX
# define ANA_BSFLUX
# define EW_PERIODIC
# define NS_PERIODIC
```

1.16.2.27.2.1 SED_TOY/ROUSE case

Testing sediment suspension in a 1DV framework to verify the agreement with Rouse theory

CPP options to add:

```
# define SED_TOY_ROUSE

# define ANA_VMIX
# define BODYFORCE
```

=> For Mustang model, just add:

```
# define MUSTANG
```

=> For USGS sediment model, just add:

```
# define SEDIMENT
# define SUSPLOAD
# define SED_TAU_CD_CONST
```

Results :

1.16.2.27.2.2 SED_TOY/CONSOLID case

This 1DV test case exemplifies the sequence of depth-limited erosion, deposition, and compaction that characterizes the response of mixed and cohesive sediment in the model. From COAWST experiments, Cohesive and mixed sediment in the Regional Ocean Modeling System (ROMS v3.6) implemented in the Coupled Ocean–Atmosphere–Wave–Sediment Transport Modeling System (COAWST r1234) [Sherwood *et al.*, 2018]

2 sand classes and 2 mud classes, cohesive behaviour, up to 38 days :

CPP options to add:

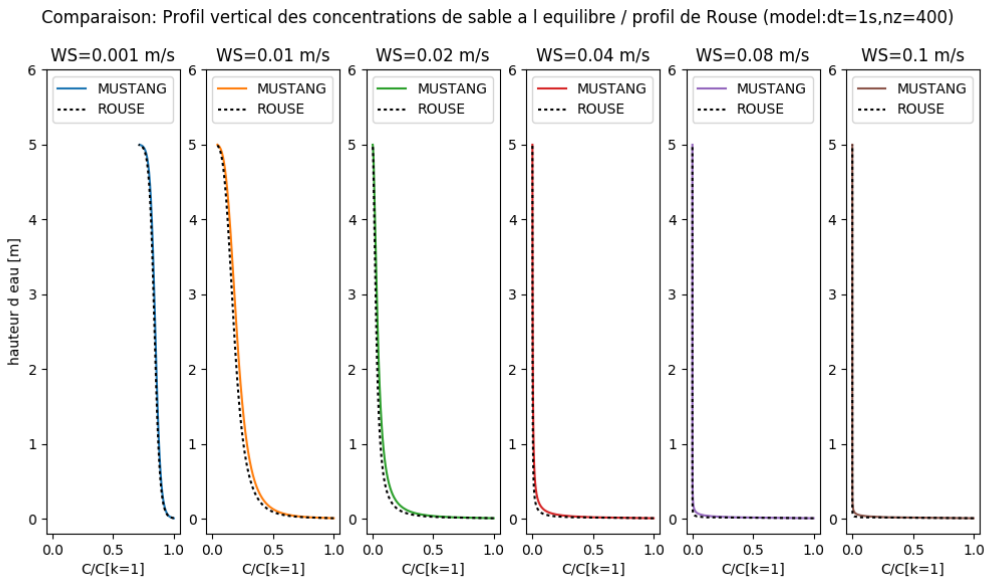


Fig. 46: Comparison between suspended concentration and analytical Rouse profiles for 6 different settling velocities

```
# define  SED_TOY_CONSOLID

# define SEDIMENT
# define SUSPLOAD
# undef  BBL
# define GLS_MIXING
# define GLS_KOMEGA
# define MIXED_BED
# undef  COHESIVE_BED
```

Results :

1.16.2.27.2.3 SED_TOY/RESUSP case

This 1DV test case to demonstrate the evolution of stratigraphy caused by resuspension and subsequent settling of different class of sediment during time-dependent bottom shear stress events. From COAWST experiments, Cohesive and mixed sediment in the Regional Ocean Modeling System (ROMS v3.6) implemented in the Coupled Ocean–Atmosphere–Wave–Sediment Transport Modeling System (COAWST r1234) [Sherwood *et al.*, 2018]

2 sand classes and 2 mud classes, non cohesive behaviour:

CPP options to add:

```
# define  SED_TOY_RESUSP

# define SEDIMENT
# define SUSPLOAD
# undef  BBL
# define GLS_MIXING
# define GLS_KOMEGA
# define MIXED_BED
# undef  COHESIVE_BED
```

Results :

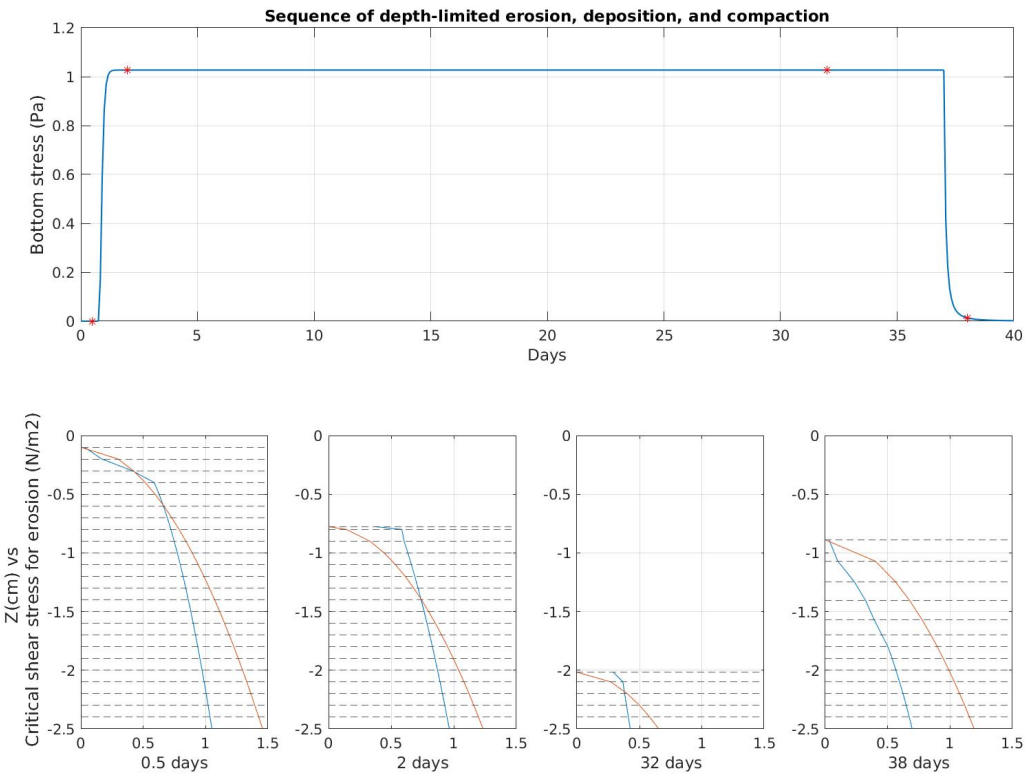


Fig. 47: Evolution of equilibrium bulk critical stress profile for erosion (red solid line) and the instantaneous profile of bulk critical stress for erosion (blue solid line)

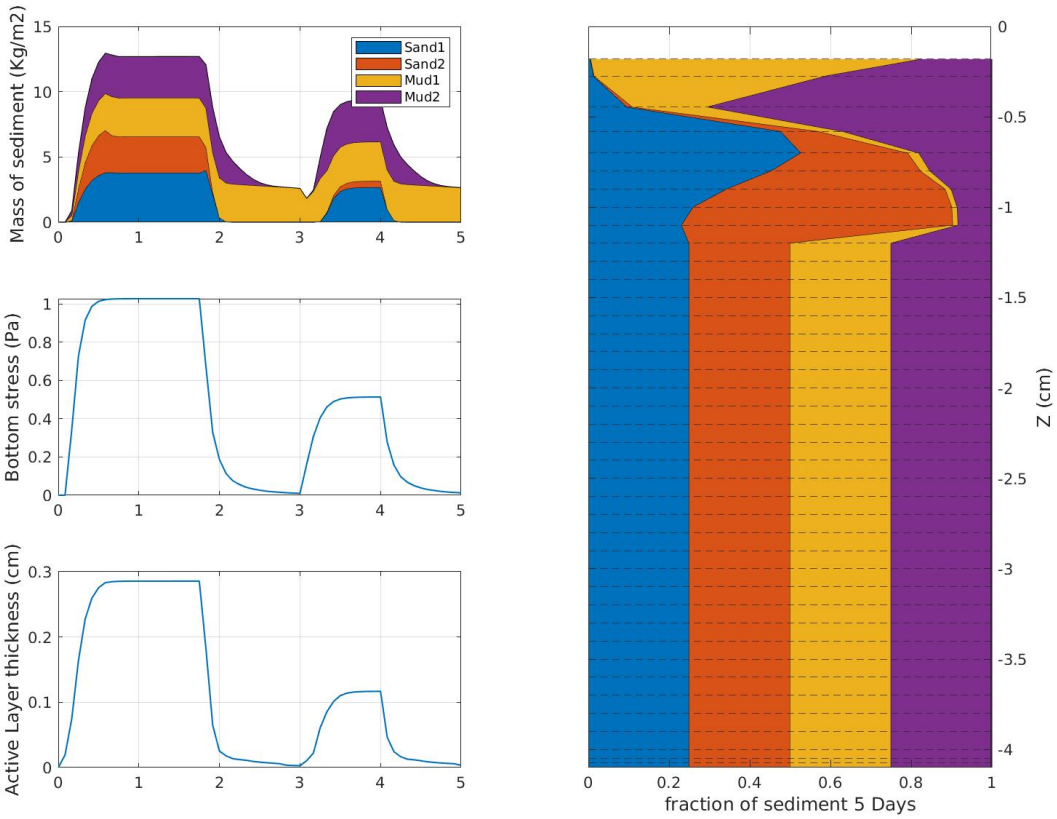


Fig. 48: Double surface stress event and response on stratigraphy 5 days later

1.16.2.27.3 TIDAL_FLAT case

2DV tidal flat with a sediment mixture (mud, fine sand, medium sand) - suspension only

```
# define  TIDAL_FLAT
```

CPP options:

```
# undef  OPENMP
# undef  MPI
# undef  NONLIN_EOS
# define NEW_S_COORD
# define SALINITY
# define UV_ADV
# define TS_HADV_WENO5
# define TS_VADV_WENO5
# define UV_HADV_WENO5
# define UV_VADV_WENO5
# define UV_COR
# define SOLVE3D
# define UV_VIS2
# define GLS_MIXING
# define ANA_INITIAL
# define WET_DRY
# define TS_DIF2
# define SPONGE
# define ANA_GRID
# define ANA_INITIAL
# define ANA_SMFLUX
# define ANA_SRFLUX
# define ANA_STFLUX
# define ANA_SSFLUX
# define ANA_BTFLUX
# define ANA_BSFLUX
# define OBC_WEST
# define FRC_BRY

# define MUSTANG
# ifdef MUSTANG
#  define key_sand2D
#  undef  key_MUSTANG_V2
# endif
```

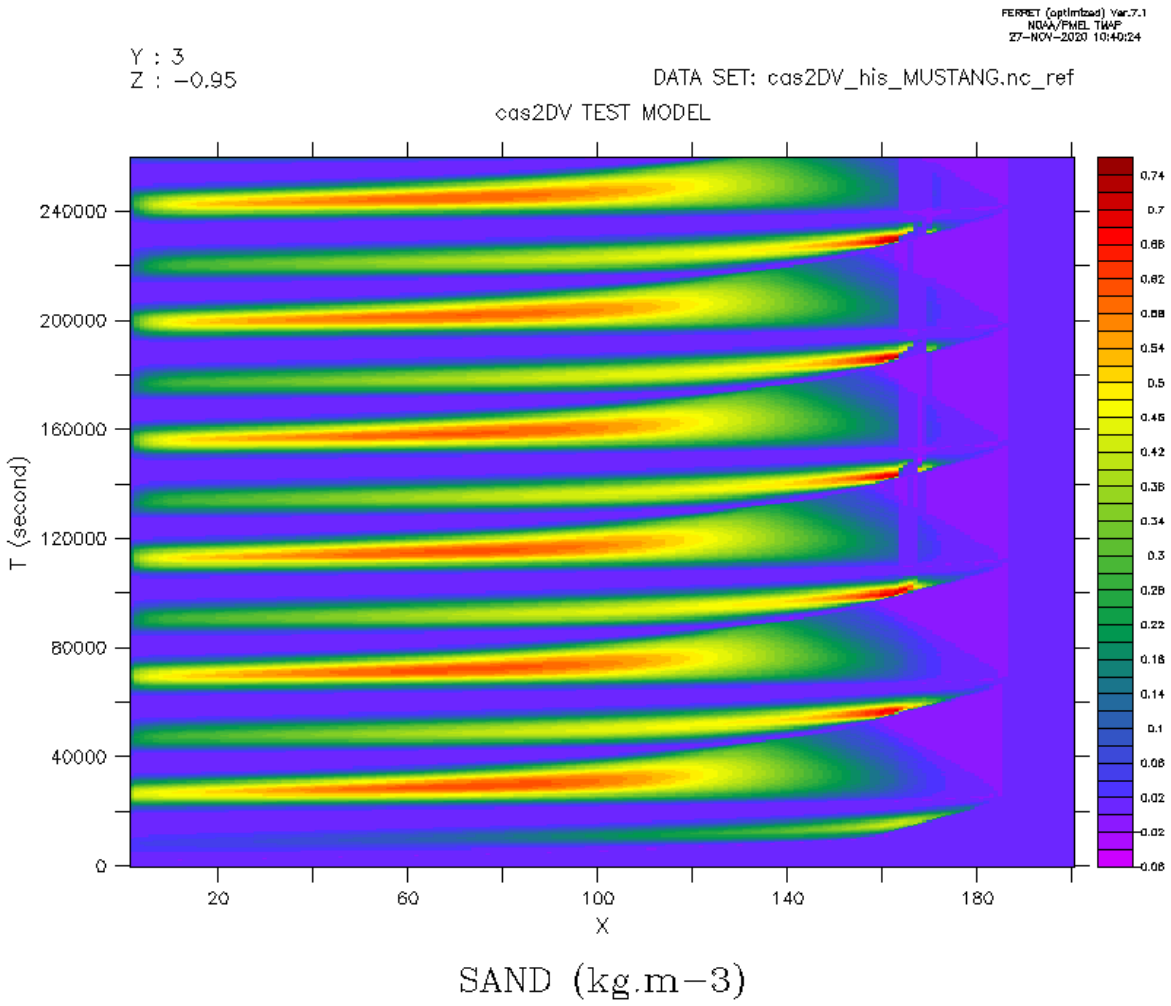
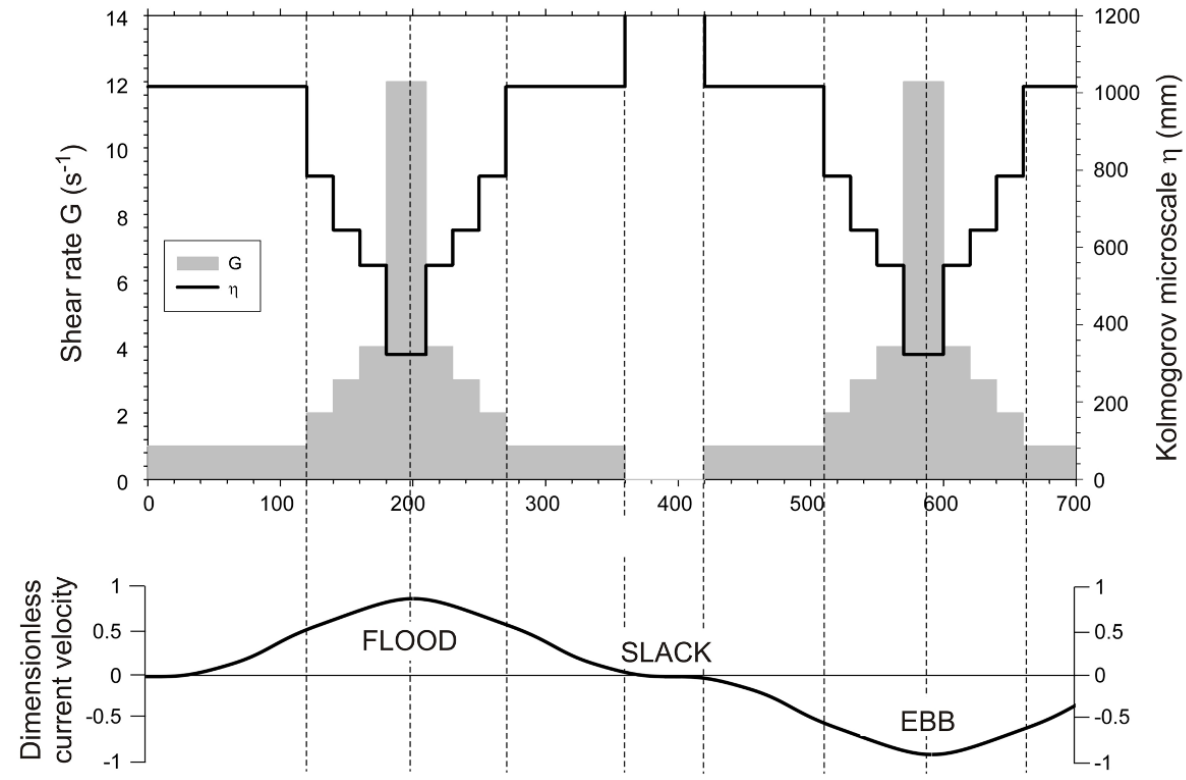


Fig. 49: Bottom mud concentration evolution over several tidal cycles

1.16.2.27.4 FLOCMOD cases

1.16.2.27.4.1 FLOCMOD 0D – comparison with laboratory experiments [#SED_TOY_FLOC_0D]

This test case simulates a laboratory experiment dedicated to flocculation experiments under controlled conditions (see Verney *et al.* [2011]). Natural SPM were mixed in a jar and agitation was tuned to simulate turbulence variations during a tidal cycle. G values ranged from 1 s-1 (around slack periods) to 12 s-1 (during flood/ebb periods). Floc size were monitored using a CCD camera, and PSD were extracted from image processing routines.



This test case can be activated with the cppkey #SED_TOY_FLOC_0D. This test case has a 1DV structure but current is set to 0 (no advection, no diffusion), and settling is not allowed (Ws = 0 m.s-1 for all classes). Shear rate is imposed using experimental values in each vertical grid cell.

The initial concentration is set to 0.093 g.l-1 (experimental value) and the initial distribution is spread over the floc size classes lower than 50 μm.

15 floc classes are used, logarithmically distributed from 4 μm to 1500 μm. Primary particle size is set to 4 μm and nf = 1.9. CROCO time step is set to 2 s.

1.16.2.27.4.2 *Shear aggregation and binary shear fragmentation only*

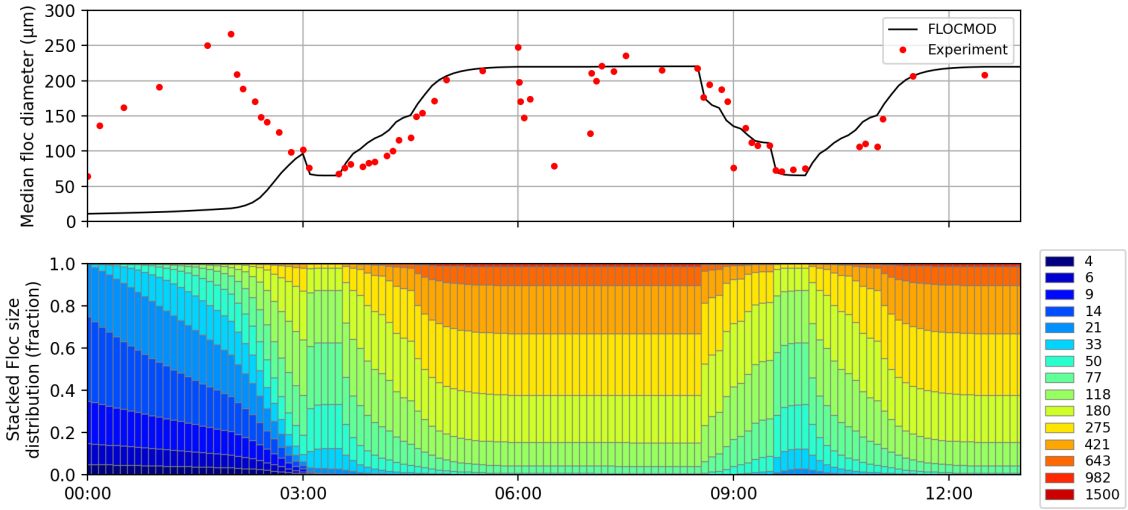
FLOCMOD main parameters are : alpha = 0.43 and beta = 0.1.

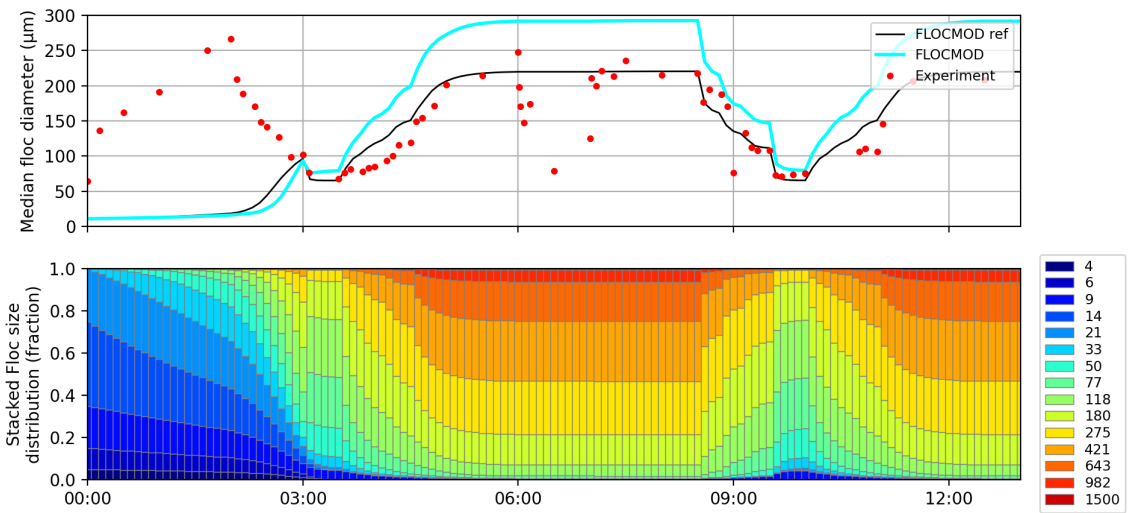
Initial floc size distribution is far from the equilibrium, and FLOCMOD fails to reproduce the first flocculation period. After the first flood period, FLOCMOD and experimental results are in good agreement considering the D50. The settling phase observed in the experiment from 06:00 to 07:00 is not simulated in the 0D model.

1.16.2.27.4.3 *Shear aggregation, binary shear fragmentation and floc erosion*

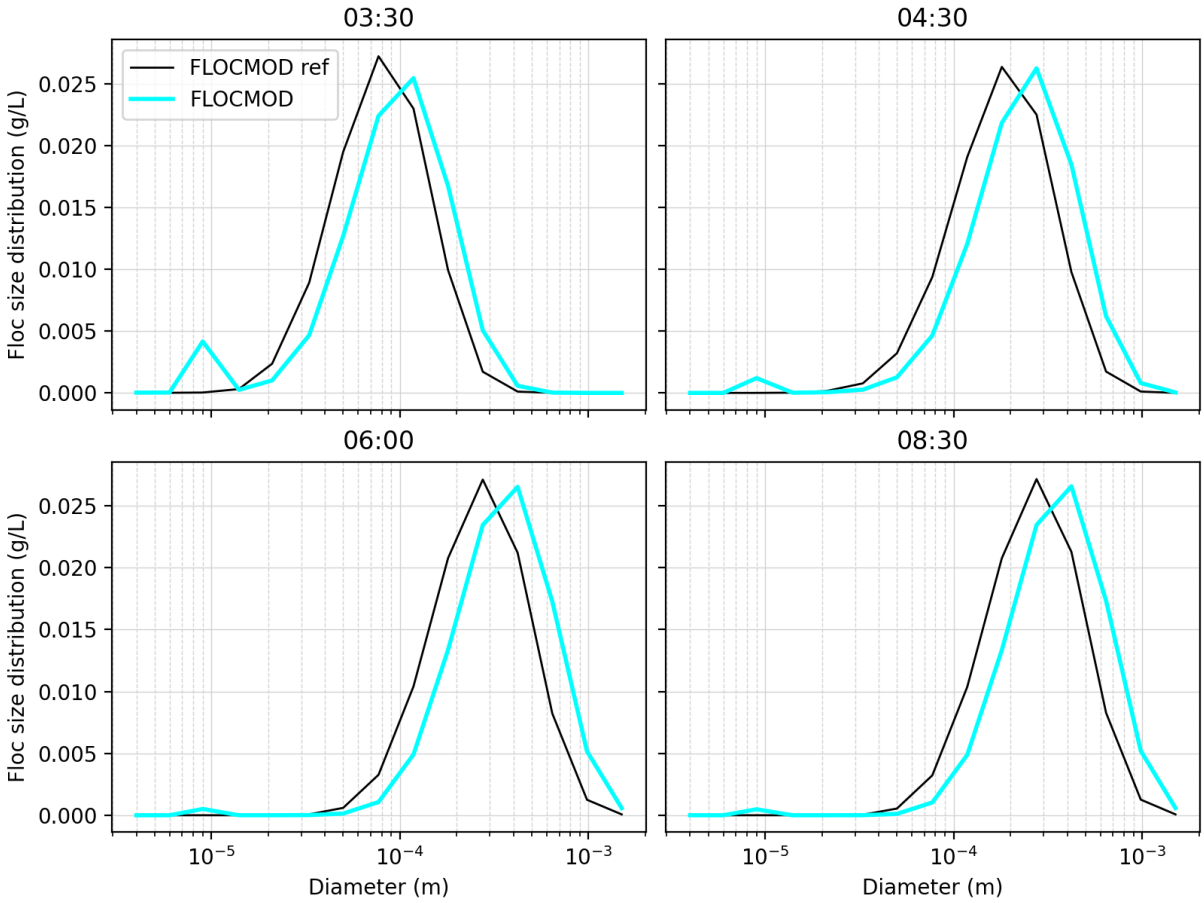
FLOCMOD main parameters are : alpha = 0.35 , beta = 0.1 , f_ero_frac = 0.4 and f_ero_iv = 3.

FLOCMOD reference is the shear aggregation/fragmentation test detailed above.


Median floc diameter (µm)
300
250
200
150
100
50
0
FLOCMOD
Experiment
Stacked Floc size distribution (fraction)
1.0
0.8
0.6
0.4
0.2
0.0
00:00
03:00
06:00
09:00
12:00
4
6
9
14
21
33
50
77
118
180
275
421
643
982
1500


Median floc diameter (µm)
300
250
200
150
100
50
0
FLOCMOD ref
FLOCMOD
Experiment
Stacked Floc size distribution (fraction)
1.0
0.8
0.6
0.4
0.2
0.0
00:00
03:00
06:00
09:00
12:00
4
6
9
14
21
33
50
77
118
180
275
421
643
982
1500

The median floc size dynamics is globally similar to the reference, however flocculation is more intense as part of shear fragmentation is attributed to floc erosion, hence flocs are less fragmented globally. We can also notice that erosion mode maintains a bimodal distribution, with a microfloc population (due to erosion) and macrofloc population varying in time with turbulence.



1.16.2.27.4.4 FLOCMOD 1DV [#SED_TOY_FLOC_1D]

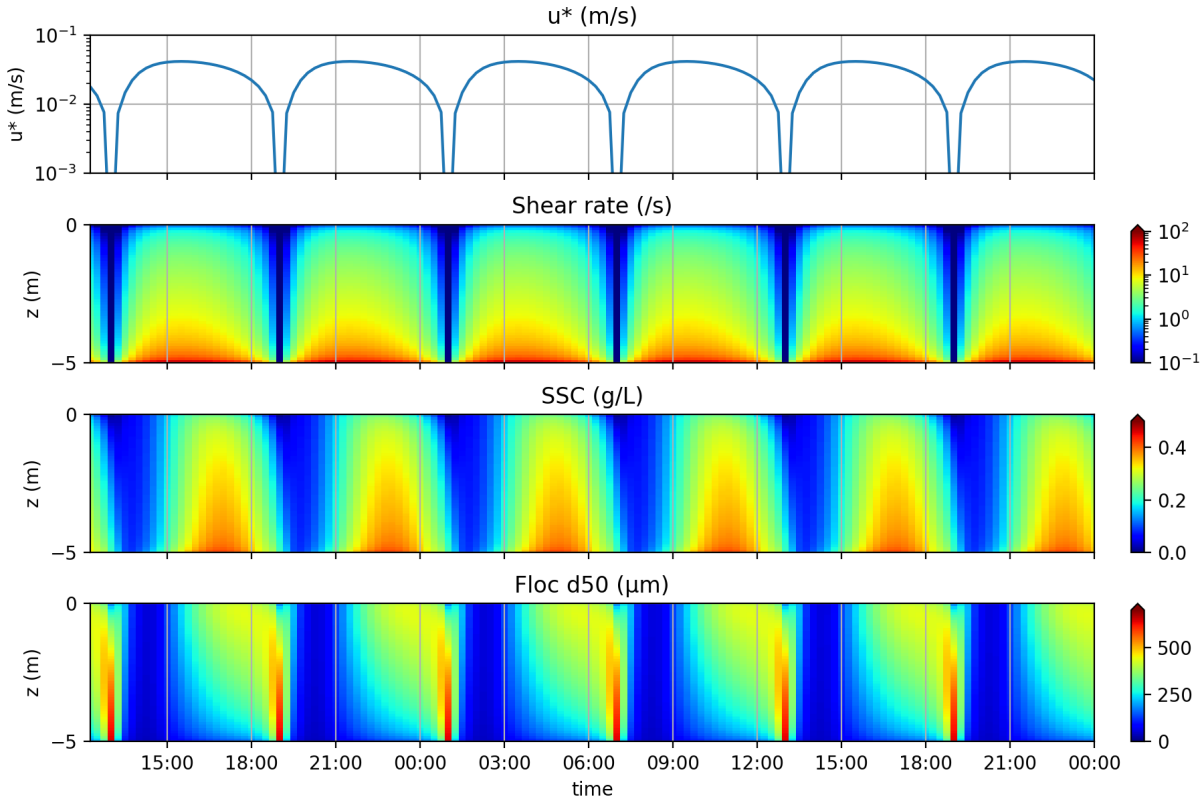
This test case illustrates the vertical flocculation dynamics along a tidal cycle. 15 floc classes are used, logarithmically distributed from 4 μm to 1500 μm. Primary particle size is set to 4 μm and nf = 1.9. CROCO time step is set to 10s.

CROCO main parameters are : h = 5 m, 50 vertical layers. A sinusoidal forcing is applied.

MUSTANG parameters : E0 = 0.0005 kg.m-3 and toce = 0.3N.m-2.

FLOCMOD main parameters are : alpha = 0.4 and beta = 0.2, including shear erosion and binary fragmentation: f_ero_frac = 0.5 and f_ero_iv = 4.

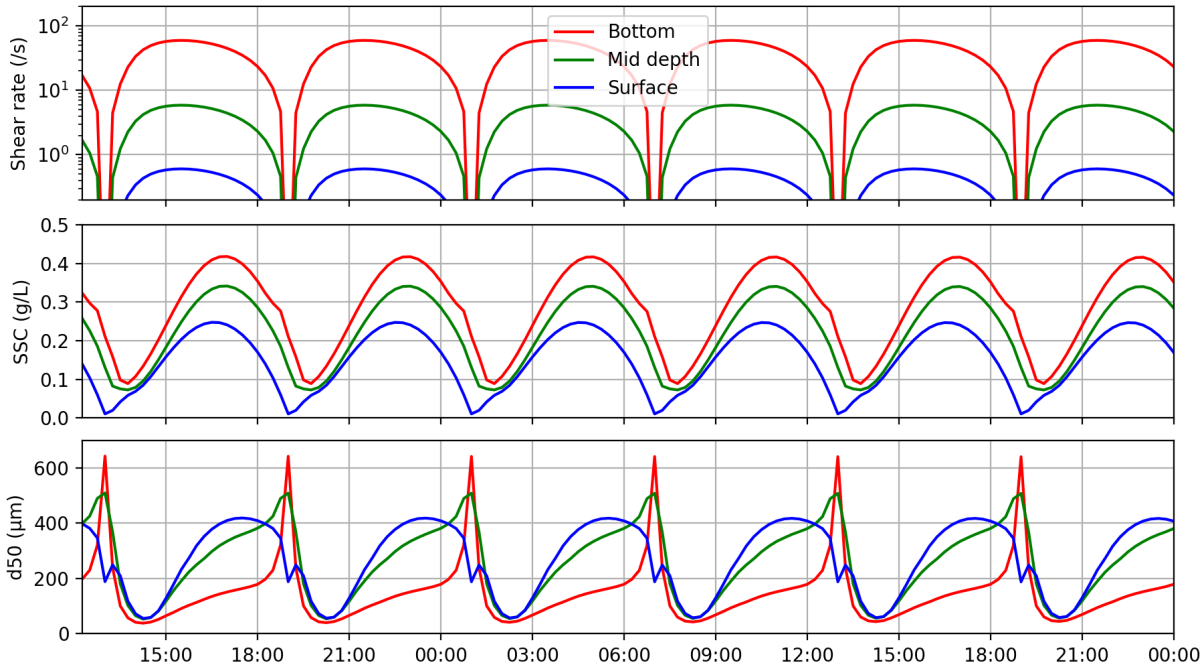
The shear rate varied from O(0.1 s-1) during slack to O(50 s-1) during max flood/ebb currents close to the bed. In the upper part of the water column, the shear rate is lower, and reaches up to O(1 s-1) during maximum current velocities.

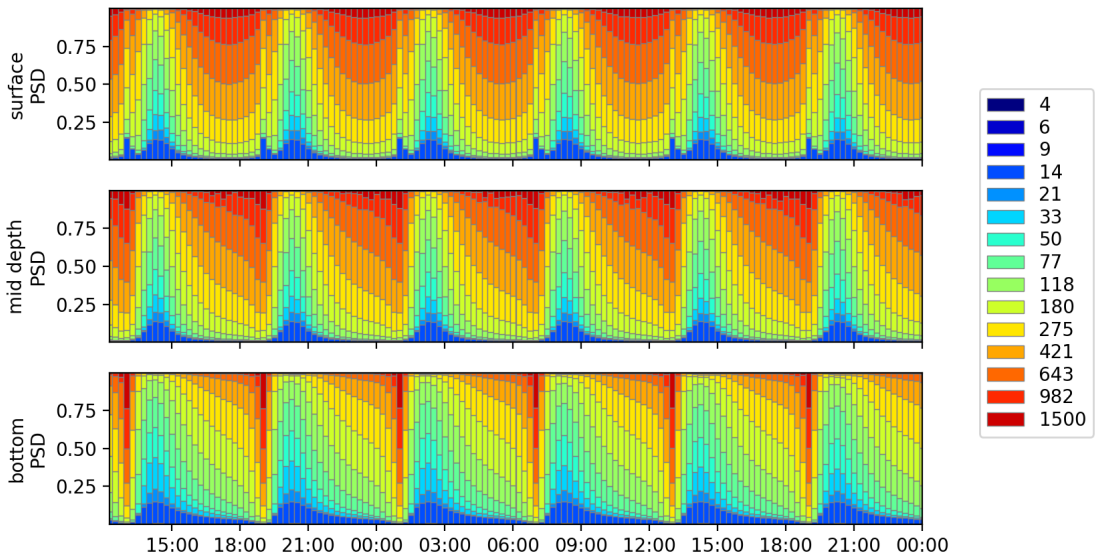


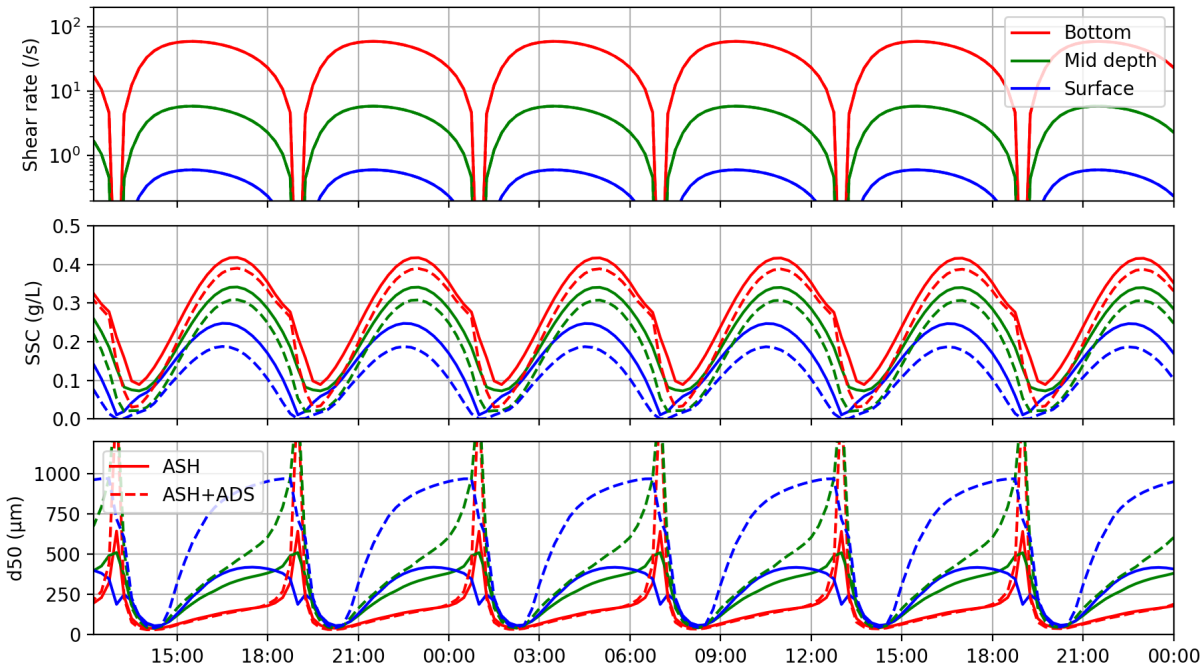
This tidal forcing induces resuspension during high shear stress periods, and SSC reaches up to 0.5 g/L close to the bed. Floc size distribution strongly varies along the tide, with the smallest floc sizes ($50\mu m$) close to the bed during maximum flood/ebb periods and the largest ($500\mu m$) during slack periods. We can note that flocculation starts earlier in the upper part of the water column, due to i) lower shear rate and ii) larger SSC values. Next flocs settle and accumulate close to the bed before settling in the sediment compartment.

1.16.2.27.4.5 *Adding differential settling aggregation*

For this test, the same setup as above is used, and aggregation by differential settling is also activated (L_ADS=.TRUE.). Adding a complementary aggregation term (and a constant fragmentation term) induces a more intense flocculation, especially in the upper half of the water column, where the shear stress is less intense (D50 greater than 1mm). Close to the bed, the floc dynamics is similar to the reference, except during slack period where settling is dominant. As a consequence, large floc sizes imply lower SSC especially in the upper part of the water column.


Bottom
Mid depth
Surface
Shear rate (/s)
SSC (g/L)
d50 (µm)
15:00
18:00
21:00
00:00
03:00
06:00
09:00
12:00
15:00
18:00
21:00
00:00

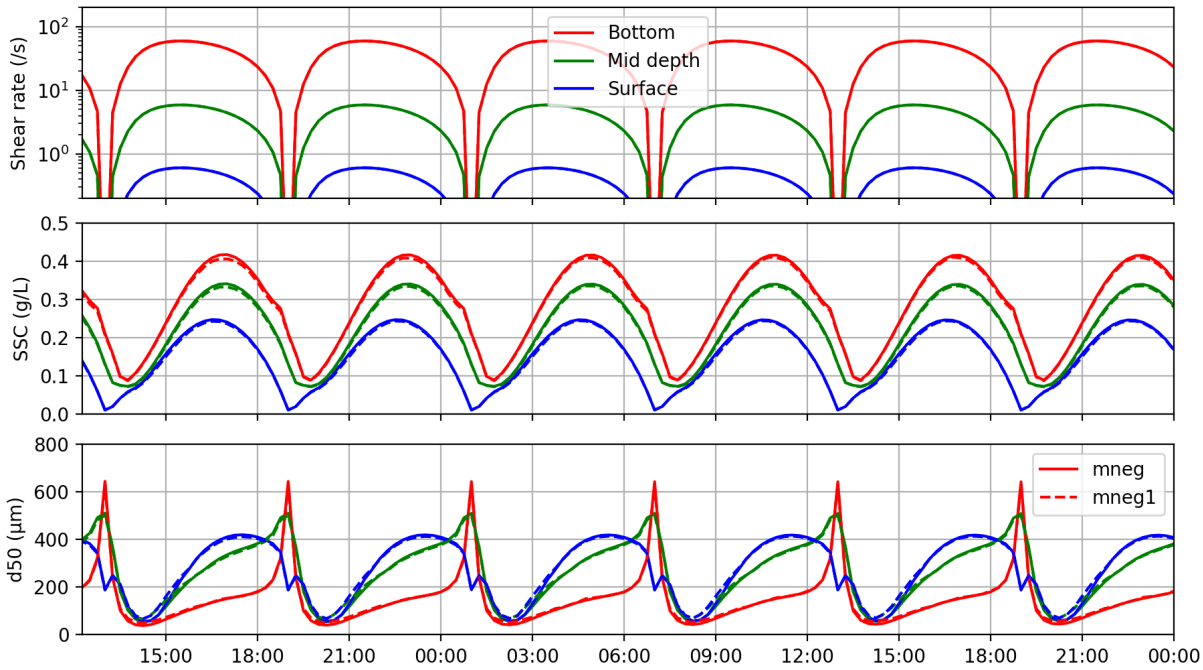

surface PSD
mid depth PSD
bottom PSD
4
6
9
14
21
33
50
77
118
180
275
421
643
982
1500
15:00
18:00
21:00
00:00
03:00
06:00
09:00
12:00
15:00
18:00
21:00
00:00


Shear rate (/s)
10^2
10^1
10^0
Bottom
Mid depth
Surface
SSC (g/L)
0.5
0.4
0.3
0.2
0.1
0.0
d50 (µm)
1000
750
500
250
0
ASH
ASH+ADS
15:00
18:00
21:00
00:00
03:00
06:00
09:00
12:00
15:00
18:00
21:00
00:00

1.16.2.27.4.6 *Adding "low negative mass option" mneg_param = 0.001 g/L.**

This test case is similar to the first 1DV test case (ADS not activated), except that low negative mass is "allowed" to limit the number of sub time steps. This means that when the total negative mass is below mneg_param, negative classes have their masses set to 0 and the remaining positive classes are proportionally lowered to ensure mass conservation.

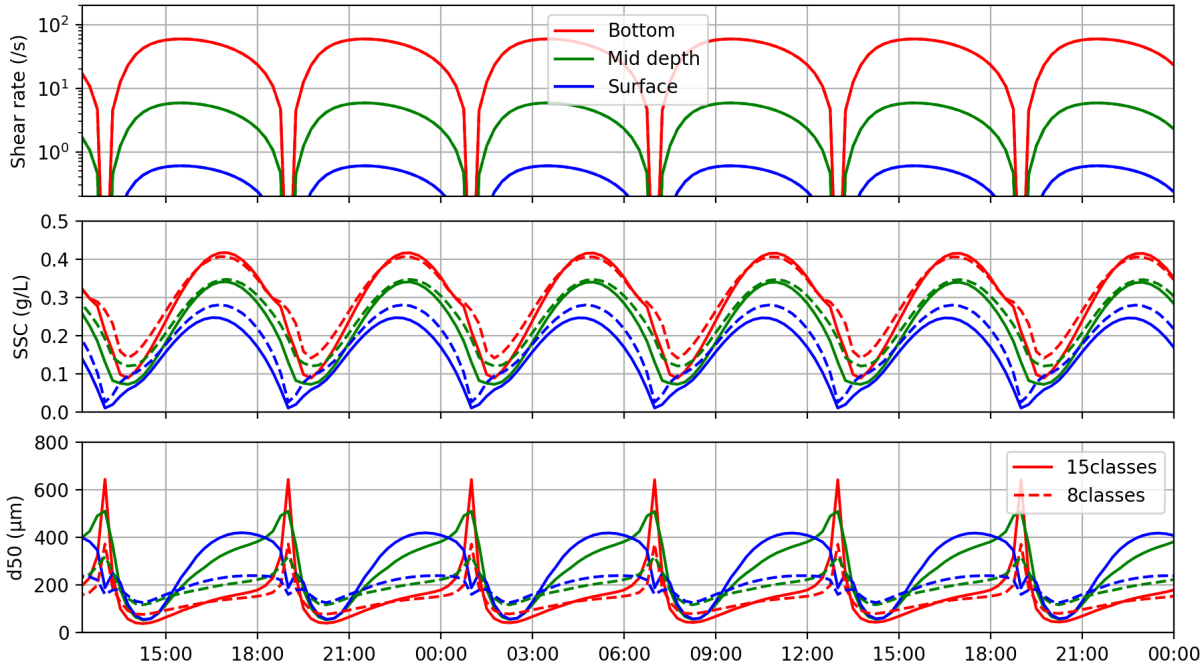
Results are very similar both in term of SSC and floc D50, hence validating the possibility to use this option to improve computation time. For this 1DV configuration, activating this option decreased the computation time of about 30%.



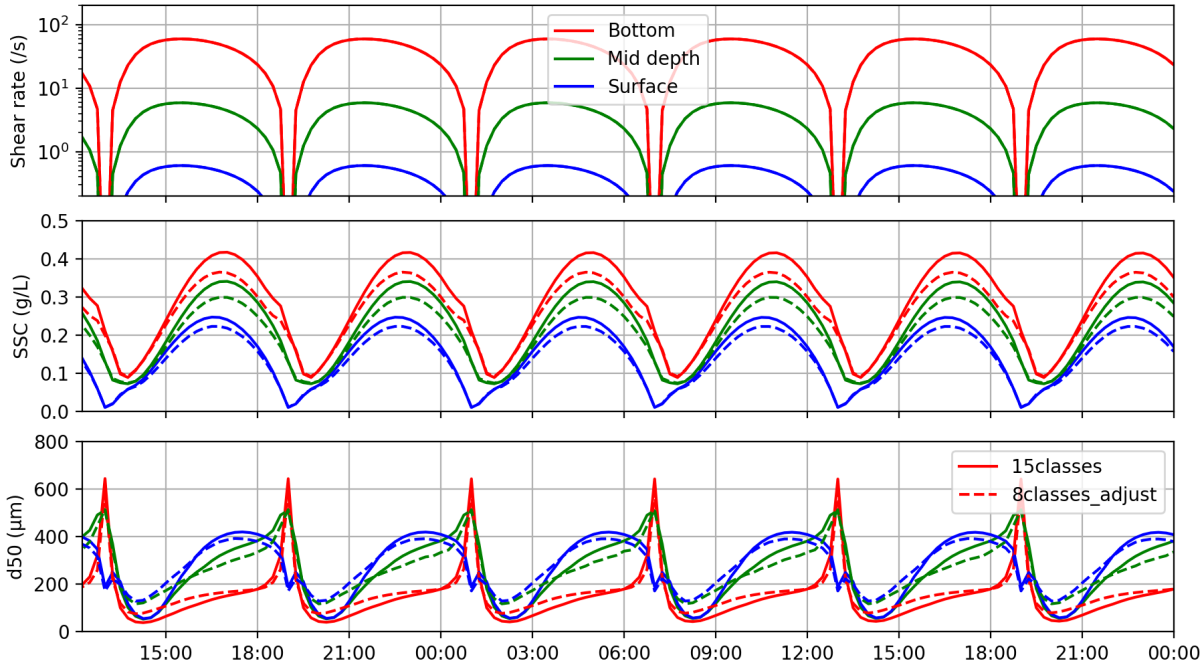
1.16.2.27.4.7 *Passing from 15 classes to 8 classes*

This last test concerns the number of classes to be used. The 15 original sediment classes are : [4; 6; 9; 14; 21; 33; 50; 77; 118; 180; 275; 421; 643; 982; 1500] in μm. We run exactly the same configuration but with 8 classes : [50; 77; 118; 180; 275; 421; 643; 982] in μm.

In this case, flocculation is less intense, which can be explained by a less important numerical diffusion induced by small size classes (aggregating with the largest).



Very similar results compared with our reference (15 classes) can be obtained when tuning alpha (= 0.8) and f_ero_frac (= 0.3).



Switching from 15 to 8 classes is crucial in term of computation time, saving up to 85% of computation time.

1.16.2.27.5 ESTUARY case

The ESTUARY configuration is a 3D test case corresponding to a schematic estuary forced by a schematic tide to the west and a river flow to the east.

Sediments are defined with 2 classes: 1 sand and 1 mud.

The bottom is initialized with a mixture of 60% mud and 40% sand.

This testcase can be used in morphostatic or morphodynamic mode by activating the cppkey MORPHODYN (see *Morphodynamic with MUSTANG* for details).

You can also see the effect of a dredging area (see *MUSTANG namelist*, dredging part) at the mouth of the estuary. (see also *dredging details* for this feature)

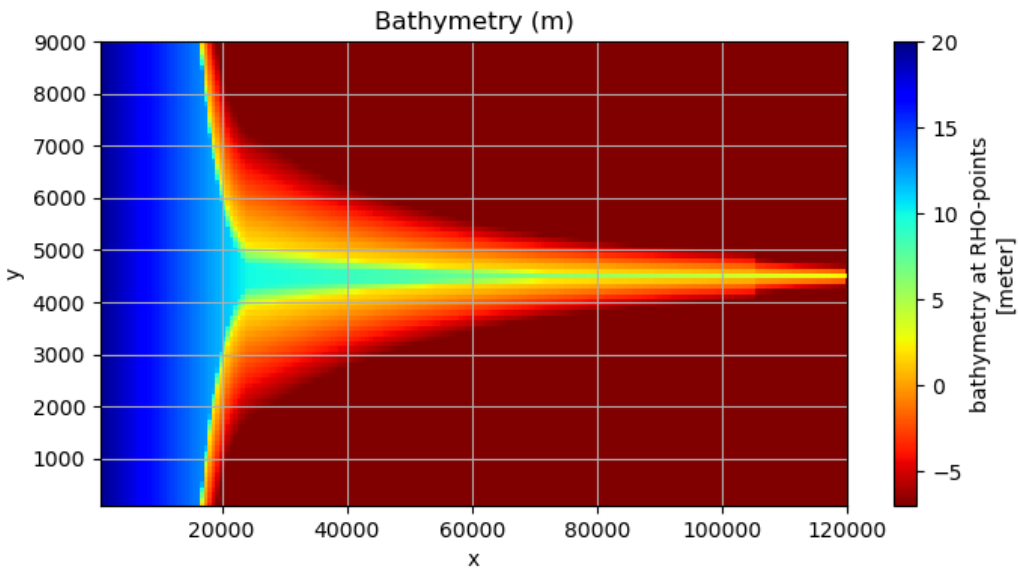


Fig. 50: Schematic bathymetry of the ESTUARY test case

To run this test case, first modify CPP options (`cppdefs.h`) :

```
# define ESTUARY
# undef REGIONAL
```

```
#elif defined ESTUARY
/*
!                        ESTUARY  Example
!                        =======  =======
*/
# undef  OPENMP
# undef  MPI
# undef  NONLIN_EOS
# define NEW_S_COORD
# define SALINITY
# define UV_ADV
# define TS_HADV_WENO5
# define TS_VADV_WENO5
# define UV_HADV_WENO5
# define UV_VADV_WENO5
# define UV_COR
# define SOLVE3D
# define UV_VIS2
```

(continues on next page)

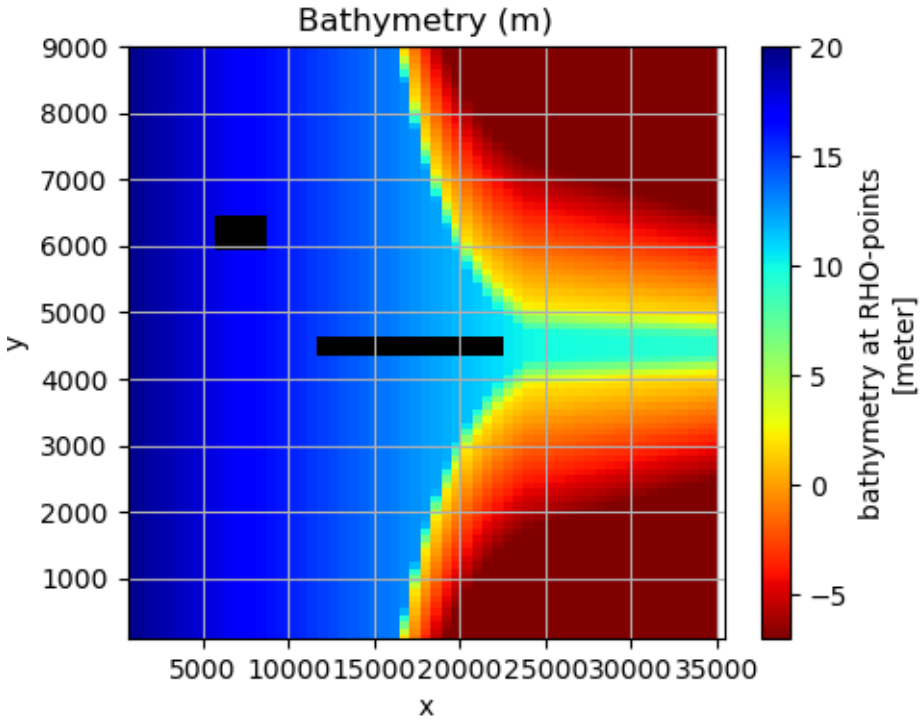


Fig. 51: Location of dredging and dumping areas (in black) in the ESTUARY test case

(continued from previous page)

```
# define GLS_MIXING
# define ANA_INITIAL
# define WET_DRY
# define TS_DIF2
# define SPONGE
# define ANA_GRID
# define ANA_INITIAL
# define ANA_SMFLUX
# define ANA_SRFLUX
# define ANA_STFLUX
# define ANA_SSFLUX
# define ANA_BTFLUX
# define ANA_BSFLUX
# define OBC_WEST
# define FRC_BRY
# ifdef FRC_BRY
#  define ANA_BRY
#  define Z_FRC_BRY
#  define OBC_M2CHARACT
#  define OBC_REDUCED_PHYSICS
#  define M2_FRC_BRY
#  undef  M3_FRC_BRY
#  define T_FRC_BRY
# endif
# undef  SEDIMENT
# define MUSTANG
# ifdef SEDIMENT
#  define SUSPLOAD
```

(continues on next page)

(continued from previous page)

```
#  undef  BEDLOAD
# endif
# ifdef MUSTANG
#  define key_sand2D
#  undef  key_MUSTANG_V2
#  define MUSTANG_CORFLUX
#  define SUBSTANCE_SUBMASSBALANCE
# endif
# define PSOURCE
# define ANA_PSOURCE
# define MASKING
# define NO_FRCFILE
# undef  ZETA_DRY_IO
# undef  RVTK_DEBUG
```

After one month in a MORPHODYN case, you can see the evolution at the mouth of the estuary and the dredging/dumping effect.

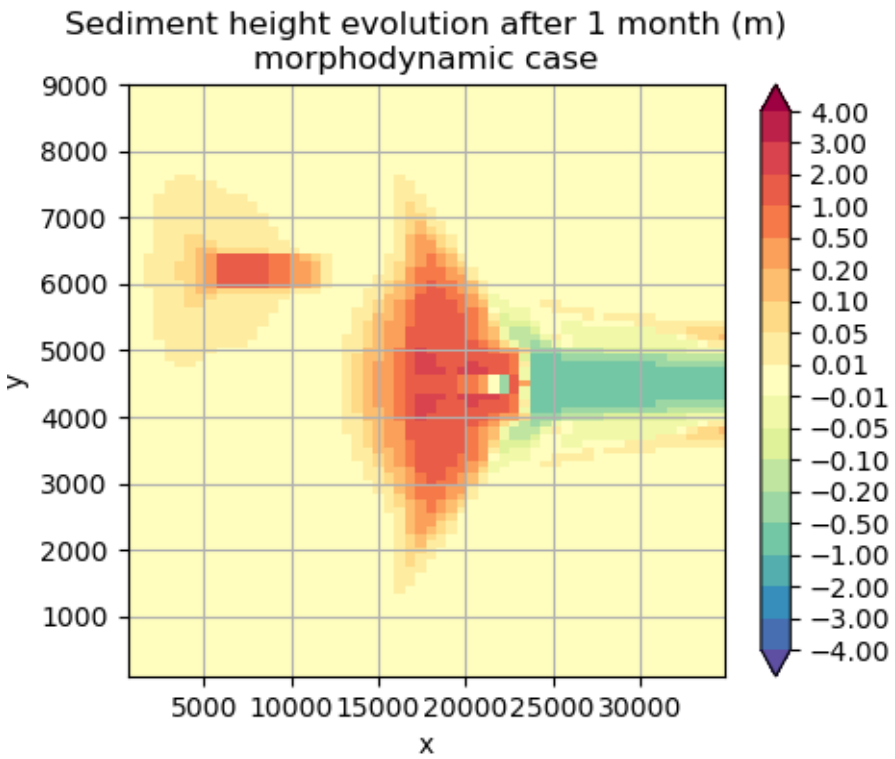


Fig. 52: Evolution of estuary mouth after one month

1.16.2.28 Kilpatrick

The purpose of this ktest is to simulate the response of the Atmospheric Boundary Layer 1d (ABL1d) model to a 2-dimensional (x-z) SST front. This ktest is based on Kilpatrick *et al.* [2014] case study (see also Lemarié *et al.* [2021], Spall [2007], Ayet and Redelsperger [2019]).

Description of the ABL1d model can be found in Lemarié *et al.* [2021]. Same version of the ABL1d model described in this paper has been implemented in CROCO.

Settings :

```
# define KILPATRICK
```

CPP options (`cppdefs.h`) :

```
#elif defined KILPATRICK
/*
!                       KILPATRICK  Example
!                       ==========  =======
*/
# define MPI
# define AVERAGES
# define NONLIN_EOS
# define SOLVE3D
# define ANA_GRID
# define ANA_INITIAL
# define ANA_SMFLUX
# define ANA_STFLUX
# define ANA_BTFLUX
# define NO_FRCFILE
# define ABL1D
# ifdef  ABL1D
#  define BULK_FLUX
#  undef  BULK_ECUMEV0
#  undef  BULK_ECUMEV6
#  define BULK_GUSTINESS
#  define ANA_ABL_LSDATA
#  define ANA_ABL_VGRID
#  define STRESS_AT_RHO_POINTS
#  undef  ABL_NUDGING
#  undef  ABL_NUDGING_DYN
#  undef  ABL_NUDGING_TRA
#  undef  ABL_DYN_RESTORE_EQ
#  undef  SFLUX_CFB
# else
#  undef BULK_FLUX
# endif
```

Warning

It's mandatory to use BULK_FLUX when ABL1D is activated. The file `cppdefs_dev.h` has been modified accordingly :

```
=====================================================================
   ABL1D option
=====================================================================
!
! Using BULK_FLUX is mandatory whenever ABL1D is activated.
!
*/
#ifdef ABL1D
# define BULK_FLUX
#endif
```

Results :

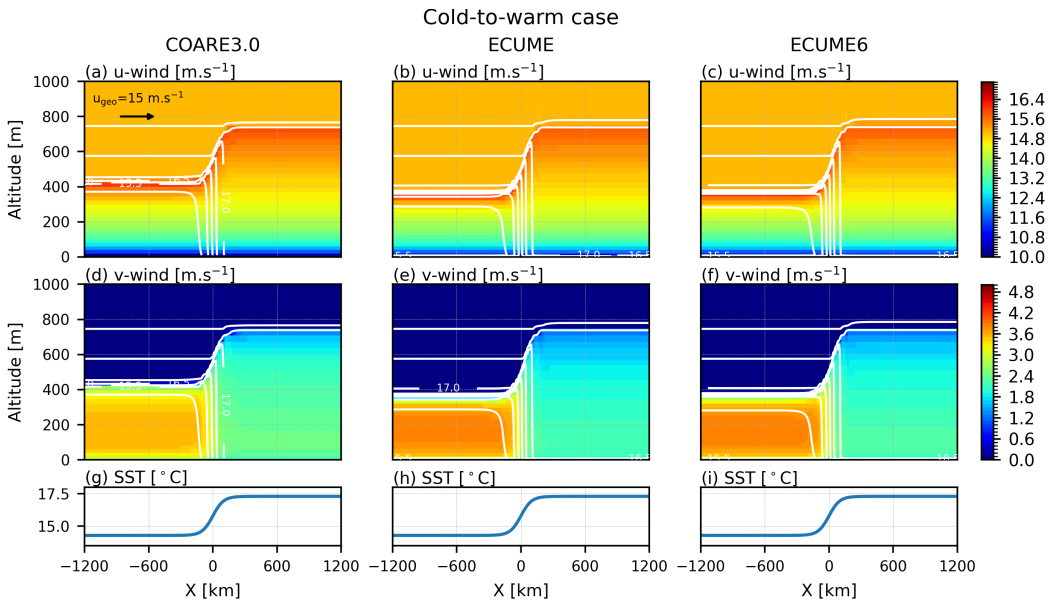


Fig. 53: Cold-to-warm case of Kilpatrick - sensitivity to bulk parametrizations

1.16.2.29 Seagrass

This case aims at reproducing flume experiments of Ganthy *et al.* [2015].

These experiments were conducted in a recirculating straight flume for five contrasted seagrass development stages (T1 to T5) and a case without vegetation (Ts) and four flow regimes (V1 to V4), leading to twenty four velocity profiles measured 0.45 m downward the leading edge of vegetation patch.

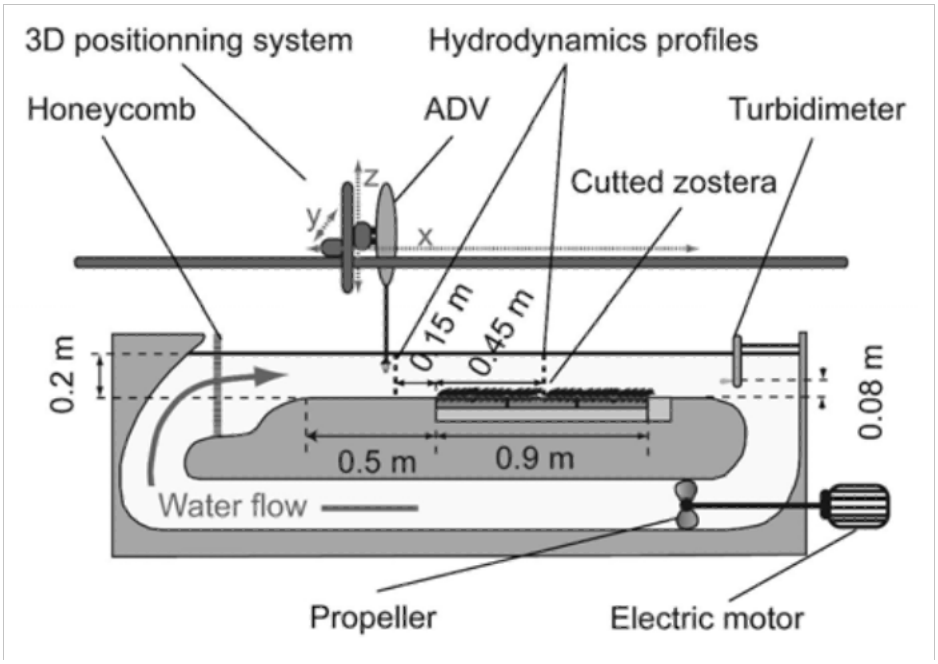


Fig. 54: Description of the experiment modelled with SEAGRASS test case

To run this test case, first modify CPP options (`cppdefs.h`) :

```
# define SEAGRASS
# undef REGIONAL
```

```
#elif defined SEAGRASS
/*
!                       Seagrass example
!                       ================
*/
# define OBSTRUCTION

# undef  OPENMP
# undef  MPI
# define SOLVE3D
# define UV_ADV
# define UV_COR
# define NONLIN_EOS
# define SALINITY
# define ANA_GRID
# define MASKING
# define ANA_INITIAL
# define ANA_SMFLUX
# define ANA_STFLUX
# define ANA_SSFLUX
# define ANA_SRFLUX
# define ANA_BTFLUX
```

(continues on next page)

(continued from previous page)

```
# define ANA_BSFLUX
# define GLS_MIXING
# define PSOURCE
# undef  PSOURCE_MASS
# define ANA_PSOURCE
# define NS_PERIODIC
# undef  FLOATS
# define NO_FRCFILE
# define USE_CALENDAR
# define NEW_S_COORD
# define OBC_EAST
# define FRC_BRY
# ifdef FRC_BRY
#  define ANA_BRY
#  define Z_FRC_BRY
#  define OBC_M2CHARACT
#  define OBC_REDUCED_PHYSICS
#  define M2_FRC_BRY
#  undef  M3_FRC_BRY
#  define T_FRC_BRY
# endif
# undef  CVTK_DEBUG

#endif
```

Settings :

The test case correspond to a 2DV case. The water depth is around 0.20 m discretized with 40 sigmas layers. The spatial resolution is 0.05 m and the domain is 1.8 m long.

For each test (T1 to T5 and Ts) and flow regime (V1 to V4 corresponding to speed from 0.1 to 0.4 m/s), the model was run during 3 minutes. An equilibrium state was reach on average after 1 minute. Results are outputted at 1 Hz over the last 30 seconds of the run and then time-averaged.

All parameters for T4V4 configuration are available in TEST_CASES directory in files :

- croco.in.Seagrass : CROCO parameters, current speed is forced by river input
- obstruction_seagrass_para.txt : OBSTRUCTION module parameters, see *OBSTRUCTION main parameter file* for more details about parameters
- obstruction_seagrass_var.txt : seagrass parameters, see *variable specific file* for more details about parameters
- obstruction_seagrass_position.nc : seagrass position, see *position file format* for more details
- obstruction_seagrass_distri.txt : seagrass vertical distribution, see *vertical distribution file format* for more details

Please refer to Ganthy *et al.* [2015], Ganthy *et al.* [2024] and Ganthy [2011] for details for the others configurations.

Results :

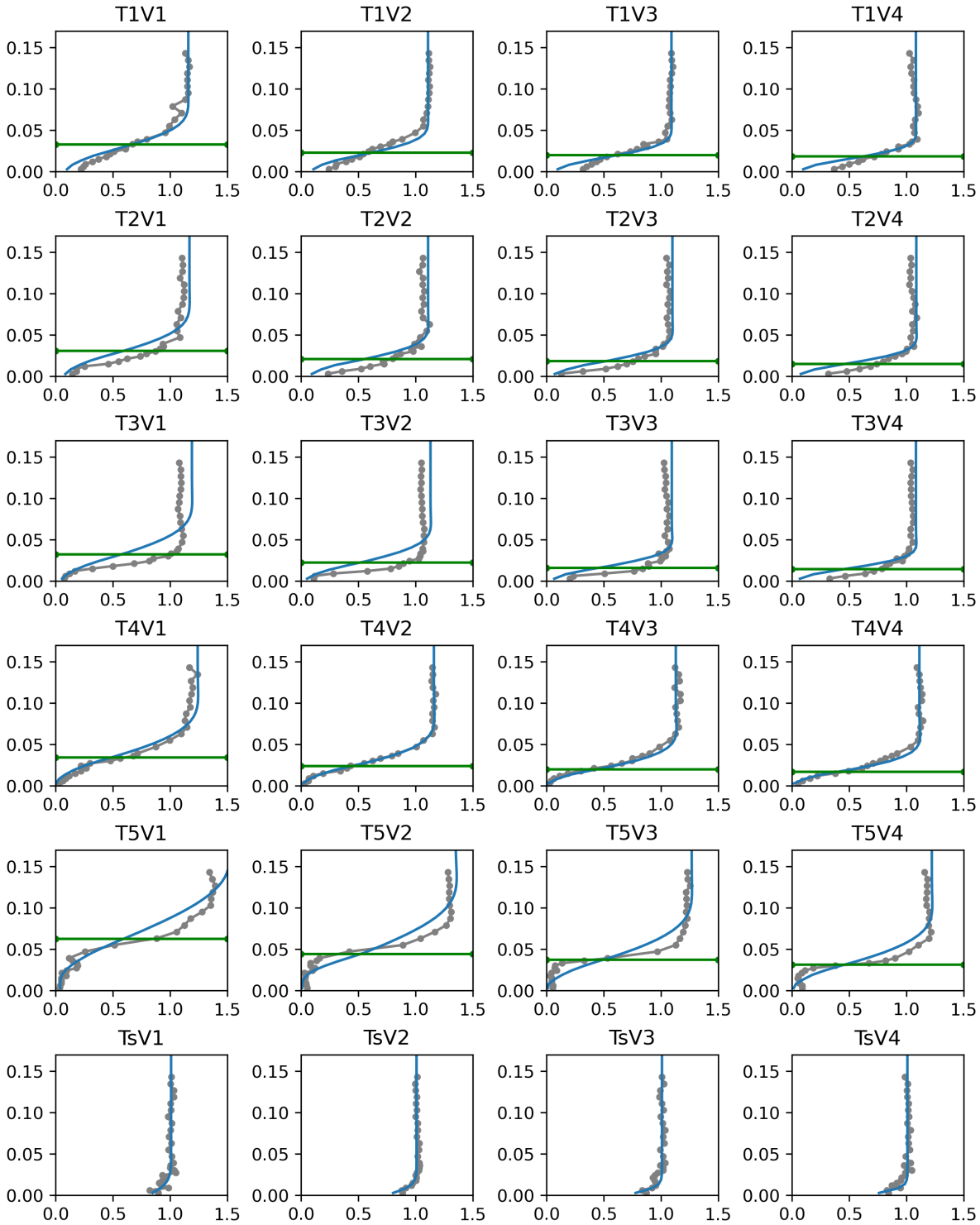


Fig. 55: Normalized velocity profiles obtained at 0.45 m downward the leading edge of vegetation patch for 4 speed values (V1 to V4) for T4 seagrass characteristics (modelled in blue, mesured in grey) and modelled canopy (green).

1.17 Appendices

1.17.1 cppdefs.h

This file defines the CPP keys that are used by the the C-preprocessor when compiling CROCO. The C-preprocessor selects the different parts of the Fortran code which needs to be compiled depending on the defined CPP options. These options are separated in two parts: the basic option keys in `cppdefs.h` and the advanced options keys in `cppdefs_dev.h`.

CPP keys define the case: test case, realistic case, as well as the numerical schemes, parameterizations, and modules used, and the forcing and boundary conditions.

- **Configuration**

CPP options	Description
BASIN	Must be defined for running the Basin example
CANYON_A	Must be defined for running the Canyon_A example
CANYON_B	Must be defined for running the Canyon_B example
EQUATOR	Must be defined for running the Equator example
GRAV_ADJ	Must be defined for running the Gravitational Adjustment example
ACOUSTIC	Must be defined for running the accoustic example
INNERSHELF	Must be defined for running the Inner Shelf example
OVERFLOW	Must be defined for running the Gravitational/Overflow example
SEAMOUNT	Must be defined for running the Seamount example
SHELFRONT	Must be defined for running the Shelf Front example
SOLITON	Must be defined for running the Equatorial Rossby Wave example
UPWELLING	Must be defined for running the Upwelling example
INTERNAL	Must be defined for running the Internal tides example
VORTEX	Must be defined for running the Baroclinic Vortex example
JET	Must be defined for running the Jet example
THACKER	Must be defined for running the Thacker example
TANK	Must be defined for running the Tank example
S2DV	Must be defined for running the S2DV example
RIP	Must be defined for running the Rip current example
SHOREFACE	Must be defined for running the Shoreface example
SWASH	Must be defined for running the Swash example
REGIONAL	Must be defined if running realistic regional simulations

- **Prallelisation**

CPP options	Description
OPENMP	Activate Open-MP parallelization protocol
MPI	Activate MPI parallelization protocol
PARALLEL_FILES	Activate parallel I/O writing
XIOS	Use external server for output
AUTOTILING	Activate subdomains partitionning optimization

- **Nesting**

CPP options	Description
AGRIF	Activate nesting capabilities (1-WAY by default)
AGRIF_2WAY	Activate 2-WAY nesting (update parent solution by child solution)

- **Open Boundary Conditions**

CPP options	Description
OBC_EAST	Open eastern boundary
OBC_WEST	Open western boundary
OBC_SOUTH	Open southern boundary
OBC_NORTH	Open northern boundary

- **Tides**

CPP options	Description
TIDES	Activate tidal forcing at open boundaries
SSH_TIDES	process and use tidal sea level data
UV_TIDES	process and use tidal current data
TIDERAMP	Apply ramping on tidal forcing (1 day) at initialization Warning! This should be undefined if restarting the model

- **Applications**

CPP options	Description
BIOLOGY	Activate biogeochemical modeling
FLOATS	Activate floats
STATIONS	Store high frequency model outputs at stations
PASSIVE_TRACER	Add a passive tracer
BBL	Activate bottom boundary layer parametrization
SEDIMENT	Activate sediment modeling (USGS model)
MUSTANG	Activate sediment modeling (MUSTANG model)

- **Grid Configuration**

CPP options	Description
CURVGRID	Activate curvilinear coordinate transformation
SPHERICAL	Activate longitude/latitude grid positioning
MASKING	Activate land masking
WET_DRY	Activate wetting-Drying scheme
NEW_S_COORD	Choose new vertical S-coordinates

- **Model Dynamics**

CPP options	Description
SOLVE3D	Solve 3D primitive equations
UV_COR	Activate Coriolis terms
UV_ADV	Activate advection terms
NBQ	Activate non-boussinesq option

- **Lateral Momentum Advection**

CPP options	Description
DV_UP3	Activate 3rd-order upstream biased advection scheme
UV_HADV_UP5	Activate 5th-order upstream biased advection scheme
UV_HADV_C2	Activate 2nd-order centred advection scheme (should be used with explicit momentum mixing)
UV_HADV_C4	Activate 4th-order centred advection scheme (should be used with explicit momentum mixing)
UV_HADV_C6	Activate 6th-order centred advection scheme (should be used with explicit momentum mixing)
UV_HADV_WENO5	Activate WENO 5th-order advection scheme
UV_HADV_TVD	Activate Total Variation Diminushing scheme

- **Lateral Momentum Mixing**

CPP options	Description
UV_MIX_GEO	Activate mixing on geopotential (constant depth) surfaces
UV_MIX_S	Activate mixing on iso-sigma (constant sigma) surfaces
UV_VIS2	Activate Laplacian horizontal mixing of momentum
UV_VIS4	Activate Bilaplacian horizontal mixing of momentum
UV_VIS_SMAGO	Activate Smagorinsky parametrization of turbulent viscosity (only with UV_VIS2)
UV_VIS_SMAGO3D	Activate 3D Smagorinsky parametrization of turbulent viscosity

- **Lateral Tracer Advection**

CPP options	Description
TS_HADV_UP3	3rd-order upstream biased advection scheme
TS_HADV_RSUP3	Split and rotated 3rd-order upstream biased advection scheme
TS_HADV_UP5	5th-order upstream biased advection scheme
TS_HADV_RSUP5	Split and rotated 5th-order upstream biased advection scheme with reduced dispersion/diffustion
TS_HADV_C4	4th-order centred advection scheme
TS_HADV_C6	Activate 6th-order centred advection scheme
TS_HADV_WENO5	5th-order WENOZ quasi-monotonic advection scheme for all tracers
BIO_HADV_WENO5	5th-order WENOZ quasi-monotone advection scheme for passive tracers (including biology and sediment tracers)

- **Lateral Tracer Mixing**

CPP options	Description
TS_MIX_ISO	Activate mixing along isopycnal (isoneutral) surfaces
TS_MIX_GEO	Activate mixing along geopotential surfaces
TS_MIX_S	Activate mixing along iso-sigma surfaces
TS_DIF2	Activate Laplacian horizontal mixing of tracer
TS_DIF4	Activate Bilaplacian horizontal mixing of tracer
TS_MIX_IMP	Activate stabilizing correction of rotated diffusion (used with TS_MIX_ISO and TS_MIX_GEO)

- **Nudging**

CPP options	Description
ZNUDGING	Activate nudging layer for zeta.
M2NUDGING	Activate nudging layer for barotropic velocities.
M3NUDGING	Activate nudging layer for baroclinic velocities.
TNUDGING	Activate nudging layer for tracer.
ROBUST_DIAG	Activate strong tracer nudging in the interior for diagnostic simulations

- **Vertical Mixing**

CPP options	Description
BODYFORCE	Apply surface and bottom stresses as body-forces
ANA_VMIX	Activate analytical viscosity/diffusivity coefficients
BVF_MIXING	Activate a simple mixing scheme based on the Brunt-Väisälä frequency
LMD_MIXING	Activate Large/McWilliams/Doney mixing (turbulent closure for interior and planetary boundary layers) with following options

LMD_SKPP	Activate surface boundary layer KPP mixing
LMD_BKPP	Activate bottom boundary layer KPP mixing
LMD_RIMIX	Activate shear instability interior mixing
LMD_CONVEC	Activate convection interior mixing
LMD_DDMIX	Activate double diffusion interior mixing
LMD_NONLOCAL	Activate nonlocal transport for SKPP

GLS_MIXING	Activate Generic Length Scale scheme as implemented by Warner et al. (2005), default is k-kl (see below)
GLS_MIX2017	Activate Generic Length Scale scheme with a sligthly different implementation (under test), default is k-epsilon (see below)
GLS_KKL	Activate K-KL (K=TKE; L=length Scale) as in Mellor and Yamada [1974]
GLS_KOMEGA	Activate K-OMEGA (OMEGA=frequency of TKE dissipation) originating from Kolmogorov [1942]
GLS_KEPSILON	Activate K-EPSILON (EPSILON=TKE dissipation) as in Jones and Launder [1972]
GLS_GEN	Activate generic model of Umlauf and Burchard [2003]

- **Equation of State**

CPP options	Description
SALINITY	Activate salinity as an active tracer
NONLIN_EOS	Activate nonlinear equation of state
SPLIT_EOS	Activate the split of the nonlinear equation of state in adiabatic and compressible parts for reduction of pressure gradient errors

- **Surface Forcing**

CPP options	Description
BULK_FLUX	Activate bulk formulation for surface heat fluxes
BULK_FAIRALL	Choose Fairall formulation (default: COAMPS formulation)
BULK_EP	Add in bulk formulation for fresh water fluxes
BULK_LW	Add in long-wave radiation feedback from model SST
BULK_SMFLUX	Add in bulk formulation for surface momentum fluxes
SST_SKIN	Activate skin sst computation [Zeng and Beljaars, 2005]
ONLINE	Read native files and perform online interpolation on ROMS grid (default cubic interpolation)
QCORRECTION	Activate linearized bulk formulation providing heat flux correction dQdSST for nudging towards model SST
SFLX_CORR	Activate freshwater flux correction around model SSS
ANA_DIURNAL_SW	Activate analytical diurnal modulation of short wave radiations (only appropriate if there is no diurnal cycle in data)
RAIN_FLUX	Add E-P flux into the mass budget (added in omega)

- **Coupling**

CPP options	Description
OW_COUPLING	Activate Ocean-Wave coupling
OA_COUPLING	Activate Ocean-Atmosphere coupling
OA_MCT	Use OASIS-MCT for coupling

- **Sponge Layer**

CPP options	Description
SPONGE	Activate areas of enhanced viscosity and diffusivity near lateral open boundaries.
SPONGE_GRID	Automatic setting of the sponge width and value
SPONGE_DIF2	Sponge on tracers (default)
SPONGE_VIS2	Sponge on momentum (default)
SPONGE_SED	Sponge on sediment (default)

- **Lateral forcing**

CPP options	Description
CLIMATOLOGY	Activate processing of 2D/3D climatological data for nudging layers and open boundary forcing
ZCLIMATOLOGY	Activate processing of sea level
M2CLIMATOLOGY	Activate processing of barotropic velocities
M3CLIMATOLOGY	Activate processing of baroclinic velocities
TCLIMATOLOGY	Activate processing of tracers
ZNUDGING	Activate nudging layer for sea level
M2NUDGING	Activate nudging layer for barotropic velocities
M3NUDGING	Activate nudging layer for baroclinic velocities
TNUDGING	Activate nudging layer for tracers
ROBUST_DIAG	Activate nudging over the whole domain
FRC_BRY	Activate processing of 1D/2D climatological or simulation/reanalysis data at open boundary points
Z_FRC_BRY	Activate open boundary forcing for sea level
M2_FRC_BRY	Activate open boundary forcing for barotropic velocities
M3_FRC_BRY	Activate open boundary forcing for baroclinic velocities
T_FRC_BRY	Activate open boundary forcing for tracers

- **Bottom Forcing**

CPP options	Description
ANA_BSFLUX	Activate analytical bottom salinity flux (generally 0)
ANA_BTFLUX	Activate analytical bottom temperature flux (generally 0)

- **Point Sources - Rivers**

CPP options	Description
PSOURCE	Activate point sources (rivers)
ANA_PSOURCE	use analytical vertical profiles for point sources (set in set_global_definitions.h)
PSOURCE_NCFILE	Read variable river transports in netcdf file
PSOURCE_NCFILE_TS	Read variable river concentration in netcdf file
PSOURCE_MASS	Apply point source as volume vertical influx at rho point instead of a flux through u,v interfaces

- **Open boundary conditions II**

CPP options	Description
OBC_VOLCONS	Activate mass conservation enforcement at open boundaries (with OBC_M2ORLANSKI)
OBC_M2SPECIFIED	Activate specified open boundary conditions for barotropic velocities
OBC_M2ORLANSKI	Activate radiative open boundary conditions for barotropic velocities
OBC_M2FLATHER	Activate Flather open boundary conditions for barotropic velocities
OBC_M2CHARACT	Activate open boundary conditions based on characteristic methods barotropic velocities
OBC_M3SPECIFIED	Activate specified open boundary conditions for baroclinic velocities
OBC_M3ORLANSKI	Activate radiative open boundary conditions for baroclinic velocities
OBC_TSPECIFIED	Activate specified open boundary conditions for tracers
OBC_TORLANSKI	Activate radiative open boundary conditions for tracers
OBC_TUPWIND	Activate upwind open boundary conditions for tracers

- **I/O - Diagnostics**

CPP options	Description
AVERAGES	Process and output time-averaged data
AVERAGES_K	Process and output time-averaged vertical mixing
DIAGNOSTICS_TS	Store and output budget terms of the tracer equations
DIAGNOSTICS_TS_ADV	Choose advection rather than transport formulation for tracer budgets
DIAGNOSTICS_TS_MLD	Integrate tracer budgets over the mixed-layer depth
DIAGNOSTICS_UV	Store and output budget terms of the momentum equations
XIOS	Use XIOS IO server

- **Biogeochemical models**

CPP options	Description
PISCES	Activate 24-component PISCES biogeochemical model
BIO_NChlPZD	Activate 5-component NPZD type model
BIO_N2PZD2	Activate 7-component NPZD type model
BIO_BioEBUS	Activate 12-component NPZD type model

- **Sediment Dynamics**

CPP options	Description
ANA_SEDIMENT	Set analytical sediment ripple and bed parameters
ANA_WWAVE	Analytical (constant) wave (hs,Tp,Dir) values.
SUSPLOAD	Activate suspended load transport
BEDLOAD	Activate bedload transport
MORPHODYN	Activate morphodynamics
ANA_BPFLUX	Set kinematic bottom flux of sediment tracer (if different from 0)
SLOPE_NEMETH	Nemeth formulation for avalanching (Nemeth et al, 2006)
SLOPE_LESSER	Lesser formulation for avalanching [Lesser *et al.*, 2004]
BEDLOAD_SOULSBY	Soulsby formulation for bedload (Soulsby, R.L. and J.S. Damgaard, 2005)
BEDLOAD_MPM	Meyer-Peter-Muller formulation for bedload [Meyer-Peter and Müller, 1948]

- **Bottom Boundary Layer**

CPP options	Description
ANA_WWAVE	Set analytical wave forcing
ANA_BSEDIM	Set analytical bed parameters (if SEDIMENT is undefined)
Z0_BL	Compute bedload roughness for ripple predictor and sediment purposes
Z0_RIP	Determine bedform roughness ripple height and ripple length for sandy beds
Z0_BIO	Determine (biogenic) bedform roughness ripple height and ripple length for silty beds

- **Wave-Current interactions**

CPP options	Description
MRL_WCI	Set wave-current interaction model
ANA_WWAVE	Analytical values for waves
WAVE_OFFLINE	Set-wave offline (read from file) forcing
WKB_WWAVE	Set WKB wave model
OW_COUPLING	Set wave coupling for WWIII (to install separately)
MRL_CEW	Set current feedback on waves
WKB_OBC_WEST	Set East/West/North/South offshore forcing for WKB model
ANA_BRY_WKB	Analytical boundary wave foring (from croco.in)
WAVE_BREAK_CT93	Thornton and Guza [1983] wave breaking (for WKB)
WAVE_BREAK_TG86	Church and Thornton [1993] wave breaking (for WKB)
WAVE_BREAK_TG86A	Another Church and Thornton formulation
WAVE_ROLLER	Set wave roller model
WAVE_STREAMING	Set bottom wave streaming
WAVE_FRICTION	Set bottom friction in WKB model (used for streaming)
WAVE_RAMP	Set wave forcing ramp using wave_ramp parameter

1.17.2 croco.in

KEYWORD	DESCRIPTION
title	Configuration name
time_stepping	NTIMES : Number of time-steps required for the simulation dt : Baroclinic time step [in s] NDTFAST : Number of barotropic time-steps between each baroclinic time step. For 2D configurations, ndtfast should be unity NINFO : Number of time-steps between printing of information to standard output
time_stepping_nbq	NDTNBQ CSOUND_NBQ VISC2_NBQ
S-coord	THETA_S: S-coordinate surface control parameter THETA_B: S-coordinate bottom control parameter Hc(m): Width of surface or bottom boundary layer in which higher vertical resolution is required during stretching
start_date	Run start date (used with `USE_CALENDAR`)
end_date	Run end date (used with `USE_CALENDAR`)
output_time_steps	DT_HIS(H) DT_AVG(H) DT_RST(H)
grid	Grid filename
forcing	Forcing filename

continues on next page

Table 2 – continued from previous page

KEYWORD	DESCRIPTION
bulk_forcing	Bulk forcing filename (used with `BULK_FLUX`)
climatology	Climatology filename (boundary and nudging, used with `CLIMATOLOGY`)
boundary	Boundary filename (used with `FRC_BRY``)
initial	NRREC: Switch to indicate start or re-start from a previous solution. nrrec is the time index of the initial or re-start NetCDF file assigned for initialization. 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. filename: Name of file containing the initial state.
restart	NRST: Number of time-steps between writing of re-start fields NRPFRST 0: write several records every NRST time steps >0: create more than one file (with sequential numbers) and write NRPRST records per file -1: overwrite record every NRST time steps filename: name of restart file
history	LDEFHIS: flag (T/F) for writing history files NWRT: Number of time-steps between writing of history fields NRPFHIS: 0: write several records every NWRT time steps >0: create more than one file (with sequential numbers) and write NRPHIS records per file -1: overwrite record every NWRT time steps filename: Name of history file
averages	NTSAVG: Starting timestep for the accumulation of output time-averaged data. For instance, you might want to average over the last day of a thirty-day run. NAVG: Number of time-steps between writing of averaged fields NRPFAVG: 0: write several records every NAVG time steps >0: create more than one file (with sequential numbers) and write NRPFAVG records per file -1: overwrite record every NAVG time steps filename: Name of average file

continues on next page

Table 2 – continued from previous page

KEYWORD	DESCRIPTION
primary_history_fields	Flags for writing primary variables in history NetCDF file
auxiliary_history_fields	Flags for writing auxiliary variables in history NetCDF file
primary_averages	Flags for writing primary variables in average NetCDF file
auxiliary_averages	Flags for writing auxiliary variables in average NetCDF file
rho0	Mean density used in the Boussinesq equation.
lateral_visc	VISC2: Laplaplacian background viscosity in m2/s (with UV_VIS2 CPP option) VISC4: Bilaplacian background viscosity in m4/s (with UV_VIS4 CPP option)
tracer_diff2	TNU2(1:NT): Laplacian background diffusivity in m2/s for each tracer (with TS_DIF2 CPP option)
tracer_diff4	TNU4(1:NT): Laplacian background diffusivity in m4/s for each tracer (with TS_DIF4 CPP option)
vertical_mixing	Constant vertical viscosity coefficient in m2/s for analytical vertical mixing scheme (with ANA_VMIX CPP option)
bottom_drag	RDRG [m/s]: Drag coefficient for linear bottom stress formulation RDRG2: Drag coefficient for constant quadratic bottom stress formulation Zob [m]: Roughness length for Von-Karman quadratic bottom stress formulation Cdb_min: Minimum value of drag coefficient for Von-Karman quadratic bottom stress formulation Cdb_max: Maximum value of drag coefficient for Von-Karman quadratic bottom stress formulation.
gamma2	Free- or partial- or no-slip wall boundary condition. 1 means free slip conditions are used.
sponge	sponge parameters are only needed if SPONGE_GRID is undefined in set_global_definitions.h; otherwise, these parameters are assigned internally. X_SPONGE [m]: width of sponge layers V_SPONGE [m2/s]: viscosity/diffusivity values in sponge layers. These values follow a cosine profile from zero interior value to V_SPONGE at the boundary.

continues on next page

Table 2 – continued from previous page

KEYWORD	DESCRIPTION
nudg_cof	TauT_in [days]: Short nudging time scale used in radiative open boundary conditions for tracer signal propagating inward the computational domain. This coefficient is used at boundary points and imposes strong nudging towards external data TauT_out [days]: Long nudging time scale used in radiative open boundary conditions for tracer signal propagating outward the computational domain. This coefficient is used at boundary points and imposes mild nudging towards external data. If CLIMATOLOGY is defined, it is also used in nudging layers with gradual decrease (cosine profile) from the open boundary to the inner border of the nudging layer. TauM_in [days]: Same as above, but for momentum equations TauM_out [days]: Same as above, but for momentum equations
diagnostics	ldefdia: flag that activates the storage of the instantaneous tracer budget terms in a diagnostic file nwrtdia: Number of time-steps between writing of diagnostic fields nrpfdia: 0: write several records every NWRTDIA time steps >0: create more than one file (with sequential numbers) and write NRPFDIA records per file -1: overwrite record every NWRTDIA time steps filename: Name of instantaneous tracer diagnostic file

continues on next page

Table 2 – continued from previous page

KEYWORD	DESCRIPTION
diag_avg	ldefdia_avg: flag that activates the storage of averaged tracer budget terms in a diagnostic file ntsdia_avg: Starting timestep for the accumulation of output time-averaged data. For instance, you might want to average over the last day of a thirty-day run. nwrtdia_avg: Number of time-steps between writing of averaged diagnostic fields nprfdia_avg: 0: write several records every NWRTDIA_AVG time steps >0: create more than one file (with sequential numbers) and write NRPFDIA_AVG records per file -1: overwrite record every NWRTDIA_AVG time steps filename: Name of average tracer diagnostic file
diag3D_history_fields	Flag to select which tracer equation (temp, salt, etc ...) to store in diagnostic file. These terms are 3D.
diag2D_history_fields	Flags to select which tracer equation integrated over the mixed layer depth (cf DIAGNOSTICS_TS_MLD) to store in diagnostic file. These terms are 2D.
diag3D_average_fields	Same as diag3D_history _fields but for averaged fields
diag2D_average_fields	Same as diag2D_history _fields but for averaged fields
diagnosticsM	Same format as diagnostics but for momentum equations ldefdiaM: nwrtdiaM: nrpfdiaM: filename:
diagM_avg	Same format as diag_avg but for momentum equations
diagM_history_fields	Flag to select which momentum equation (u,v) to store in diagnostic file. These terms are 3D.
diagM_average _fields	Same as diagM_history _fields but for averaged fields
diagnosticsM_bio	Same format as diagnostics but for biogeochemical budget terms (other than advection/diffusion)
diagbio_avg	Same format as diag_avg but for biogeochemical budget terms (other than advection/diffusion)
diagbioFlux_history_fields	Flag (T or F) to select which biogechemical tracer flux to store in diagnostic file. These terms are 3D. (For NPZD type model)

continues on next page

Table 2 – continued from previous page

KEYWORD	DESCRIPTION
diagbioVSink_history_fields	Flag (T or F) to select which biogechemical tracer sinking flux equation to store in diagnostic file These terms are 3D. This is for NPZD type model(BIO_NChlPZD, BIO_N2ChlPZD2 and BIO_BioEBUS), you need to follow the biogechemical tracers order.
diagbioGasExc_history_fields	Flag (T or F) to select which biogechemical tracer Gas exchange flux equation to store in diagnostic file. These terms are 2D.
diagbioFlux_average_fields	Same as above but averaged
diagbioVSink_average_fields	Same as above but averaged
diagbioGasExc_average_fields	Same as above but averaged
biology	Name of file containing the Iron dust forcing used in the PISCES biogeochemical model
sediments	Input file: sediment parameters input file
sediment_history_fields	Flags for storing sediment fields in history file bed_thick:Thickness of sediment bed layer (m) bed_poros: Porosity of sediment bed layer (no unit) bed_fra(sand,silt): Volume fraction of sand/silt in bed layer (no unit)
bbl_history_fieldsi	Flags for storing bbl fields in history file Abed: Bed wave excursion amplitude (m) Hripple: Bed ripple length (m) Lripple: Bed ripple length (m) Zbnot: Physical hydraulic bottom roughness (m) Zbapp: Apparent hydraulic bottom roughness (m) Bostrw: Wave-induced kinematic bottom stress (m)
floats	Lagrangian floats application. Same format as diagnostics LDEFFLT NFLT NRPFFLT inpname, hisname
floats_fields	Type of fields computed for each lagrangian floats
station_fields	Fixed station application. Same format as diagnostics LDEFSTA NSTA NRPFSTA inpname, hisname

continues on next page

Table 2 – continued from previous page

KEYWORD	DESCRIPTION
psource	Nsrc: point source number Isrc: I point source indice Jsrc: J point source indice Dsrc: Direction of point source flow (u=0,v=1) Qbar [m3/s]: Total transport at point source Lsrc: Logical switch for type of tracer to apply Tsrc: Tracer value
psource_ncfile	Nsrc: point source number Isrc: I point source indice Jsrc: J point source indice Dsrc: Direction of point source flow (u=0,v=1) qbardir: Orientation: South=0 or North=0, East=0 or West=1 Lsrc: Logical switch for type of tracer to apply Tsrc: Tracer value in case of analytical value [#undef PSOURCE_NCFILE_TS] runoff file name: Input netCDF runoff file

1.17.3 Comparison of ROMS and CROCO versions

Models	CROCO	ROMS-UCLA	ROMS-Rutgers / COASWT
Origin	UCLA-IRD-INRIA-IFREMER-SHOM-CNRS	UCLA	UCLA-Rutgers-USGS
Maintenance	IRD-INRIA-IFREMER- SHOM-CNRS	UCLA	Rutgers-USGS
Realm	Europe-World	US West Coast	US East Coast
Introductory year	1999 (AGRIF) 2016 (CROCO)	2002	1998

CODE FEATURES

Models	CROCO	ROMS-UCLA	ROMS-Rutgers / COASWT
Parallelization	MPI or OpenMP (Hybrid branch exists)	Hybrid MPI-OpenMP	MPI or OpenMP
Nesting	Online at barotropic level	Off-line (On-line at baroclinic level and not yet operational)	Off-line
Data assimilation	3DVAR	3DVAR	4DVAR
Wave-current interact.	McWilliams *et al.* [2004]	McWilliams *et al.* [2004]	Mellor [2003] McWilliams *et al.* [2004]
Air-sea coupling	OASIS-MCT	Home made	MCT
Sediment Dynamics	Blaas *et al.* [2007] MUSTANG	Blaas *et al.* [2007]	Blaas *et al.* [2007] Warner *et al.* [2008]
Biogeochemistry	NPZD Gruber *et al.* [2006], PISCES	NPZD Gruber *et al.* [2006]	EcoSim, NEMURO, NPZD Franks, NPZD Powell, Fennel
Sea ice	none	none	Budgell [2005]
Vertical mixing	KPP, GLS	KPP, GLS	KPP, GLS
Wetting-Drying	Warner *et al.* [2013]	none	Warner *et al.* [2013]

TIME STEPPING

Models	CROCO	ROMS-UCLA	ROMS-Rutgers / COASWT
2D momentum	Generalized FB AB3-AM4	Generalized FB AB3-AM4	LF-AM3 with FB feedback
3D momentum	LF-AM3	LF-AM3	AB3
Tracers	LF-AM3 with stabilizing correction for isopycnal hyperdiffusion	LF-AM3 with stabilizing correction for isopycnal hyperdiffusion	LF-TR with explicit geopotential diffusion (no stabilizing correction : strong stability constraint)
Internal waves	LF-AM3 with FB feedback	LF-AM3 with FB feedback	Generalized FB (AB3-TR)
Coupling stage	Predictor	Corrector	

STABILITY CONSTRAINTS (Max Courant number)

Models	CROCO	ROMS-UCLA	ROMS-Rutgers / COASWT
2D	1.78	1.78	1.85
3D advection	1.58	1.58	0.72
Coriolis	1.58	1.58	0.72
Internal waves	1.85	1.85	1.14

CHAPTER

TWO

TUTORIALS

2.1 System requirements

2.1.1 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.

2.1.2 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.

2.1.3 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`

2.2 Download

2.2.1 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
- **Datasets to create the input files:**
  - grid
  - surface atmospheric forcing
  - oceanic boundaries and initialization

2.2.1.1 Source code

CROCO and CROCO_TOOLS stable releases are available in the Download section: https://www.croco-ocean.org/download/

They are available as tarball or can be checked-out using git. Follow instructions in the **Download stable release** section.

2.2.1.2 External datasets

Some external datasets are needed by CROCO_TOOLS. Some of them, like bathymetry, tide atlas, atmospheric, oceanic and biogeochemical climatological datasets are directly available in the **Datasets** section as tarball archives.

CROCO_TOOLS also provide pre-processing scripts for the download and create interannual forcings as:

- CFSR, ERA-interim, ERA5 … for atmospheric forcing
- SODA and MERCATOR for the oceanic boundaries and initialization

2.2.2 Getting other codes (coupling)

- **OASIS coupler**

To use CROCO in coupled mode (coupling with atmosphere and/or waves), **OASIS3-MCT version 3 or later** 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://oasis.cerfacs.fr/en/home/

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

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

We advise to use the latest stable release, ie the tag OASIS3-MCT_5.2 of the branch OASIS3-MCT_5.0

```
git clone --branch OASIS3-MCT_5.0 https://gitlab.com/cerfacs/oasis3-mct.git
cd oasis3-mct
git checkout OASIS3-MCT_5.2
```

- **WW3**

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

Warning

Currently the coupling tools provided in croco are designed to work with the WW3 6.07.1 release plus some additional changes in a few coupled routines in WW3 which are provided in the `croco/SCRIPTS/SCRIPTS_COUPLING/WW3_IN/modified_ftn` directory. See `readme_ww3_version` in `WW3_IN`. More recent versions of WW3 contains these modifications, but as these more recent versions are currently not tagged as "releases", we prefer to stick to the latest official release and just change these few modified routines.

You can clone the WW3 6.07.1 version with:

```
git clone --branch 6.07.1 --single-branch https://github.com/NOAA-EMC/WW3.git
```

And copy the modified routines:

```
cp ~/croco/croco/SCRIPTS/SCRIPTS_COUPLING/WW3_IN/modified_ftn/*.ftn  ~/ww3/model/ftn/.
```

- **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
```

A tag is available for using the up-to-date wrf-croco version.

Note

If using older versions of the wrf-croco fork, you may encounter issues regarding a namelist variable named max_cpldom, which is present in the up-to-date version, but was inexistent in previous version (with older version, you should remove this variable from your namelist.input.base.complete file). We encourage to use the tagged up-to-date version.

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
```

2.3 Contents & Architecture

2.3.1 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:

2.3.2 Contents

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

2.3.2.1 croco

AGRIF	Agrif library for nesting
BENCH	Testing script
CVTK	Regression test library
MPI_NOLAND_preprocessing	Fortran utility to determine the optimal MPI decomposition to supress land computation
MUSTANG	MUSTANG sediment model source files
OBSTRUCTION	OBSTRUCTION module source files
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

2.3.2.2 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, ERA5, ERA-Interim, QuickScat data for atmospheric forcing, SODA and CMEMS/Mercator 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.

2.3.2.2.1 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, 2005] or CMEMS/Mercator [Lellouche *et al.*, 2021]) 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

2.3.2.2.2 UTILITIES

export_fig	A MATLAB toolbox for exporting publication quality figures : https://github.com/altmany/export_fig
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.

2.3.2.2.3 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)
GSHHS	A Global Self-consistent, Hierarchical, High-resolution Geography Database [Wessel and Smith, 1996]. Original data can be found at https://www.soest.hawaii.edu/pwessel/gshhg/
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 and Bopp [2006] Si, O2, NO3, PO4 from WOA2005, DIC and Alkalinity come from Goyet et al.

2.4 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`.

2.5 Test Cases

2.5.1 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) ))
```

(continues on next page)

(continued from previous page)

```
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:
   - 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

2.5.2 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 performed 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 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 (concerning 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 we 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/..
```

(continues on next page)

(continued from previous page)

```
#
# 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)
```

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.

2.6 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
```

(continues on next page)

(continued from previous page)

```
# 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 .

# 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

2.7 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/

2.7.1 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, 2005] or CMEMS/Mercator [Lellouche *et al.*, 2021]) 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

2.7.2 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

2.7.3 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/
```

CARS20	CSIRO Atlas of Regional Seas database. Annual, seasonal and monthly climatology for temperature, salinity, nitrate, phosphate and oxygen
COADS(	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
QuikSC/	QuickSCAT monthly climatology of wind stress
RUNOFI	River discharge monthly climatology in $m.s^{-3}$ for the 925 largest rivers reaching the ocean (from Dai en Trenberth, 2000)
SST_pat	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]
SeaW-ifs	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
WOA200	World Ocean Atlas 2009 global dataset References list: http://www.nodc.noaa.gov/OC5/WOA09/pubwoa09.html
WOAPIS	A global dataset for biogeochemical PISCES data (annual and seasonal climatology). References are : Fe and DOC : Aumont and 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
```

2.7.4 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, 2005] or CMEMS/Mercator [Lellouche *et al.*, 2021]) 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)
%
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 ERA5 and mercator forcing sets

```
% 7 - Parameters for Interannual forcing (SODA, mercator, CFSR, ERA5 ...)
%
%
Download_data = 1;   % Get data from OPENDAP sites
level         = 0;   % AGRIF level; 0 = parent grid
%
% ...
                 %           1/1/1979 - 31/3/2011
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
add_waves    = 0;       % 1: add waves
% ...
%
%-------------------------------------------------
%  Options for make_ERA5
%-------------------------------------------------
%
ERA5_dir     = [FORC_DATA_DIR,'ERA5_',CROCO_config,'/'];        % ERA5 data dir.␣
↪[croco format]
My_ERA5_dir = [FORC_DATA_DIR,'ERA5_native_',CROCO_config,'/']; % ERA5 native␣
↪data downloaded
                                                                % with python␣
↪script

itolap_era5 = 2;                                                % 2 records = 2␣
↪hours
%
% ...
%
%-------------------------------------------
% Options for make_OGCM_SODA or make_OGCM_mercator
%-------------------------------------------
%
OGCM         = 'mercator';        % Select OGCM: SODA or mercator
%
OGCM_dir     = [FORC_DATA_DIR,OGCM,'_',CROCO_config,'/'];  % OGCM data dir.␣
↪[croco format]
```

(continues on next page)

(continued from previous page)

```
%
bry_prefix  = [CROCO_files_dir,'croco_bry_',OGCM,'_'];    % generic boundary␣
↪file name
clm_prefix  = [CROCO_files_dir,'croco_clm_',OGCM,'_'];    % generic climatology␣
↪file name
ini_prefix  = [CROCO_files_dir,'croco_ini_',OGCM,'_'];    % generic initial file␣
↪name
OGCM_prefix = [OGCM,'_'];                                 % generic OGCM file␣
↪name

mercator_type=1;    % 1 -->  1/12 deg Mercator global reanalysis
                    % 2 -->  1/12 deg Mercator global analysis
                    % 3 -->  1/12 deg Mercator global forecast (See Section 8.)
                    % 4 -->  1/24 deg Mercator Mediterranean analysis/forecast␣
↪(See Section 8.)
                    % 5 -->      the same than 4 but with detiding␣
↪postprocessing on current and ssh

%  =============
%  To download CMEMS data: set login/password (http://marine.copernicus.eu)
%  and path to copernicusmarine executable
%  see Oforc_OGCM/Copernicus_Marine_Toolbox_installation.md
%
%  Various sets of data are proposed in the Copernicus web site
%
if strcmp(OGCM,'mercator')
%
pathCMC='/path/to/home/copernicusmarine'; % copernicusmarine client
%
user     = 'XXXX';
password = 'XXXX';
%
```

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 ERA5 interannual atmospheric forcing, script are in `Aforc_ERA5/`. Details are available in `README_ERA5.txt`. The download part used python script

   To build your CMEMS/mercator interannual ocean forcing, the useful script is `Oforc_OGCM/make_OGCM_mercator.m`. The download part used python script

   Warning

   As this pluri-month preprocessing can be longer and uses more CPU ressources, you may need to submit it as a job. A few example scripts (for SODA and CFSR) are provided:

   ```
   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_ERA5_Y????M?.nc
   croco_bry_mercator_Y????M?.nc
   ```

(continues on next page)

(continued from previous page)

```
croco_ini_mercator_Y????M?.nc
```

2.8 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:

2.8.1 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  SINGLE_COLUMN   /* 1DV vertical mixing Example */
#undef  RIVER           /* River run-off Example */
#undef  OVERFLOW        /* Gravitational/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 */
```

(continues on next page)

(continued from previous page)

```
#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  MOVING_BATHY    /* Moving Bathymetry 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 */
#undef  DUNE            /* Dune migration Example */
#undef  SED_TOY         /* 1DV sediment toy Example */
#undef  TIDAL_FLAT      /* 2DV tidal flat Example */
#undef  ESTUARY         /* 3D tidal estuary Example */
#undef  KILPATRICK      /* 2D sst front*/
#undef  SEAGRASS        /* 2DV over seagrass using OBSTRUCTION module*/
```

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` section of the Tutorials). Let's just highlight a few ones:

- 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.

2.8.2 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 subdomains partition in terms of computation time.

> **Note**
>
> MPI tiles should be at least 20x20 points.

2.8.3 Compilation using jobcomp

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

You have to copy the jobcomp script in your working directory.

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
```

To use the jobcomp script, you can first explore the available option through :

```
./jobcomp -h
```

Available option are :

- –src <path> : Specify the CROCO source directory (default: ../croco/OCEAN).
- –fc <compiler> : Specify the Fortran compiler (default: gfortran). Available = gfortran, ifort, ifc, nvfortran, pgfortran
- –mpif90 <compiler> : Specify the MPI Fortran compiler (default: mpif90).
- –fflags <flags> : Specify the compilation flags
  - default: if fc=ifort or ifx : '-O2 -mcmodel=medium -fno-alias -i4 -r8 -fp-model precise' .
  - default: if fc=gfortran : '-O2 -mcmodel=medium -fdefault-real-8 -fdefault-double-8 -std=legacy' .
  - default: if fc=nvfortran or pgfortran : '-g -fast -r8 -i4 -mcmodel=medium -Mbackslash' .
- –jobs <flags> : Number of processes for compilation (default: 1).
- –netcdf-inc <path> : Specify the NetCDF include directory. (default: nf-config –includedir)
- –netcdf-lib <path> : Specify the NetCDF library directory. (default: nf-config –flibs)
- –xios-dir <path> : Specify the XIOS root directory. (default: /root/xios)
- –prism-dir <path> : Specify the OASIS-MCT root directory. (default: ../../../oasis3-mct/compile_oa3-mct)

If you use default options, you can compile with :

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

Or you can for example use ifort instead of gfortran :

```
     ./jobcomp --fc ifort --mpif90 mpiifort > 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
```

2.8.4 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)

2.8.5 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

2.9 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
```

2.9.1 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:

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

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:

```
start_date:
2000-01-01 00:00:00

end_date:
2000-02-01 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
```

(continues on next page)

(continued from previous page)

```
                        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.

2.9.2 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?

2.9.3 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

2.10 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)?

2.11 Running with interannual forcing

2.11.1 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_inter.bash`: Plurimonth run with interannual forcing
- `run_croco.bash`: Plurimonth run with climatological/cycling forcing. It is a simplified version of run_croco_inter.bash which is not described below.

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.

```
#
# Name used for the input files. For example croco_grd.nc
MODEL=croco

# Scratch directory where the model is run
SCRATCHDIR=`pwd`/SCRATCH

# Input directory where the croco_inter.in input file is located
INPUTDIR=`pwd`/CROCO_IN  # prod architecture
#INPUTDIR=`pwd`          # dev architecture

# AGRIF input file which defines the position of child grids
AGRIF_FILE=AGRIF_FixedGrids.in

# Directory where the croco input NetCDF files (croco_grd.nc, ...) are stored
MSSDIR=`pwd`/CROCO_FILES

# Directory where the croco output and restart NetCDF files (croco_his.nc, ...)␣
↪are stored
MSSOUT=$SCRATCHDIR

# CROCO executable
CODFILE=./croco
```

> **Warning**
>
> check `INPUTDIR` depending on the architecture you choosed (prod or dev architecture)

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=ERA5
# Atmospheric surface forcing dataset used for the wind stress (NCEP, QSCAT)
ATMOS_FRC=QSCAT
# Oceanic boundary and initial dataset (SODA, ECCO,...)
OGCM=SODA
# Runoff dataset (Daie and Trenberth,...)
RUNOFF_DAT=DAI
```

Time step settings

```
# Model time step [seconds]
DT=3600
# Number of barotropic time steps within one baroclinic time step [number],␣
↪NDTFAST in croco.in
NFAST=60
```

Agrif nesting settings

```
# Number total of grid levels (1: No child grid)
NLEVEL=1
# AGRIF nesting refinement coefficient
AGRIF_REF=3
```

Dates settings (according to `crocotools_param.m`)

```
# Start and End year
NY_START=2005
NY_END=2005
# Start and End month
NM_START=1
NM_END=3
# Set month format at 1 or 2 digits (for input and output files): "%01d" = 1␣
↪digit/ "%02d" = 2 digit
MTH_FORMAT="%02d"
#  Time Schedule  -  TIME_SCHED=0 --> yearly files
#                    TIME_SCHED=1 --> monthly files
TIME_SCHED=1
```

(continues on next page)

(continued from previous page)

```
# Number of year that are considered to be part of the spin-up (i.e. 365 days␣
↪per year)
NY_SPIN=0
```

Outputs settings

```
# 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
```

Restart settings

```
#  Restart file - RSTFLAG=0 --> No Restart
#                 RSTFLAG=1 --> Restart
RSTFLAG=0
#  Exact restart - EXACT_RST=0 --> Exact restart OFF
#                - EXACT_RST=1 --> Exact restart ON
EXACT_RST=0
```

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`.

2.11.2 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.

2.12 Running forecasts

With the method described by Marchesiello *et al.* [2008], CROCO can be used to downscale global forecasts in order to provide higher resolution forecasting. To do so, It is advised to start from building a regional configuration following the dedicated tutorials and testing it with climatological or interannual forcing after editing `crocotools_param.m`, `croco.in`, `cppdefs.h`, `param.h`. Then, specifically for the forecast system, you will

need to set the forecast parameters in `crocotools_param.m`, define cpp key `ROBUST_DIAG` in cppdefs.h and recompile the model using jobcomp, then edit `croco_forecast.in`. Finally, run the script `run_croco_forecast.bash` that will download global forecast (ocean and atmosphere), create specific forcing files using the Forecast_tools Matlab package, then run croco for the current period.

2.12.1 Strategy of Forecast_tools

CROCO_TOOLS allows running both inter-annual and real-time simulations. In the latter case, we rely on operational global ocean circulation models for the initial and lateral boundary conditions and operational global atmospheric models (introduced through bulk formulations) for surface forcing. In order to limit the volume of data transferred over the Internet, we use OPeNDAP (Matlab) or the Copernicus Marine Service Toolbox (Python) and extract only the necessary subgrid. We use the U.S. NCEP/NOAA data base for atmospheric forcing and the European CMEMS data for oceanic forcing.

In a regional domain with low intrinsic variability where the circulation is directly forced by surface fluxes (away from fronts and eddies), initialization is of lower importance. However, if oceanic intrinsic variability with low predictability is dominant (e.g., mesoscale eddies with monthly time-scale), the model may well provide a statistically reliable image of the eddy field, but out of phase at any given time. An initialization method is therefore needed to bring ocean fields into phase with real time. Here, we do not perform data assimilation, but rely on the global CMEMS forecast products, which assimilates satellite altimetry and in-situ data. Newtonian nudging is used to adjust CMEMS data to CROCO dynamics in a downscaling procedure.

The strategy for a hindcast/forecast cycle is as follows. The simulation starts at t0-hdays (hdays=1, t0 being the present time) and ends at t0+fdays (fdays=3). The model is run using interpolated data from CMEMS for lateral boundary conditions and NCEP for surface forcing. CMEMS data or a CROCO restart file (from a previous forecast) is used for initialization at t0-1, while nudging assimilates CMEMS data in the interior domain (with a nudging time-scale of about 10 days). A Shell script (run_croco_forecast.bash) manages the whole procedure: download and pre-processing, forecast simulations, post-processing (plot_forecast_croco.m), data storage and preparation of the next forecast cycle.

2.12.2 Set forecast parameters

1. Edit `crocotools_param.m`

   In section 7 Make sure that `OGCM=mercator` and set your login/password (http://marine.copernicus.eu) to access CMEMS data through the python Copernicus Marine Service Toolbox in Forecast_tools directory.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% 8 - Parameters for the forecast system
%
%     --> select OGCM name above (mercator ...)
%     --> select Aforc name (GFS)
%     --> don't forget to define in cppdefs.h:
%             - ROBUST_DIAG
%             - CLIMATOLOGY
%             - BULK_FLUX
%             - TIDES if you choose so, but without TIDERAMP
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
FRCST_dir = [FORC_DATA_DIR,'Forecast/'];  % path to local OGCM data directory
%
%
% Number of hindcast / forecast days
%
hdays=1;
fdays=3;
%
% Local time= UTC + timezone
```

(continues on next page)

(continued from previous page)

```
%
timezone = +2;
%
% Add tides
%
add_tides_fcst = 1;       % 1: add tides
%
%  MERCATOR cases (See Section 7):
%  =============
%
if strcmp(OGCM,'mercator')
    if mercator_type==3
        % ========================
        % 3 -> For Global Forecast PSY4
        % ========================
        product_id={'cmems_mod_glo_phy_anfc_0.083deg_P1D-m'  ...
                    'cmems_mod_glo_phy-cur_anfc_0.083deg_P1D-m', ...
                    'cmems_mod_glo_phy-thetao_anfc_0.083deg_P1D-m', ...
                    'cmems_mod_glo_phy-so_anfc_0.083deg_P1D-m'};
    %
    elseif mercator_type==4
        % ========================
        % 4 -> For Mediterannean Forecast PSY4 4.2km
        % ========================
        product_id={'cmems_mod_med_phy-ssh_anfc_4.2km_P1D-m', ...
                    'cmems_mod_med_phy-cur_anfc_4.2km_P1D-m', ...
                    'cmems_mod_med_phy-tem_anfc_4.2km_P1D-m', ...
                    'cmems_mod_med_phy-sal_anfc_4.2km_P1D-m'};
    %
    elseif mercator_type==5
        % ========================
        % 5 -> For Mediterannean Forecast PSY4 4.2km (ssh detided)
        % ========================
        product_id={'cmems_mod_med_phy-ssh_anfc_detided_4.2km_P1D-m', ...
                    'cmems_mod_med_phy-cur_anfc_4.2km_P1D-m', ...
                    'cmems_mod_med_phy-tem_anfc_4.2km_P1D-m', ...
                    'cmems_mod_med_phy-sal_anfc_4.2km_P1D-m'};
    %
    else
    end
end
```

Make sure that `OGCM=mercator` (section 7), choose the hindcast/forecast period `hdays/fdays` (start with default), set your time zone and then your login/password (http://marine.copernicus.eu) to access CMEMS data through motuclient python package in Forecast_tools directory.

> **Warning**
>
> CMEMS url and files may sometimes change name, resulting in system failure until we make the necessary adjustments.

2. Edit `croco_forecast.in` if you want to adjust some default parameters like nudging coefficient to forecast data (by default on the order of 15 days)

```
nudg_cof:    TauT_in, TauT_out, TauM_in, TauM_out  [days for all]
```

(continues on next page)

(continued from previous page)

```
              1.       15.      1.      15.
```

> **Warning**
>
> `croco_forecast.in` contains template variables for NTIMES, DT, NDTFAST, NRST,NWRT and NAVG

2.12.3 Compiling

1. Edit `cppdefs.h` for defining forcing and nudging procedures:

```
# define BULK_FLUX

              /* Lateral Forcing */
# define CLIMATOLOGY
# ifdef CLIMATOLOGY
#  define ZCLIMATOLOGY
#  define M2CLIMATOLOGY
#  define M3CLIMATOLOGY
#  define TCLIMATOLOGY

#  define ZNUDGING
#  define M2NUDGING
#  define M3NUDGING
#  define TNUDGING
#   define ROBUST_DIAG
# endif
```

   `ROBUST_DIAG` sets non-zero nudging coefficients tauT_out for nudging to CMEMS forecast (T,S) data.

2. Re-compile the model:

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

2.12.4 Running the script

1. Copy run_croco_forecast.bash from SCRIPTS/Plurimonths_scripts to local directory.
2. Edit run_croco_forecasts.bash.

   Check in particular the locations of matlab executable and loaddap library (used to extract NCEP atmospheric forecast data)

```
export TOOLSDIR=$HOME/croco/croco_tools/Forecast_tools
export RUNDIR=${PWD}
export MATLAB=/usr/local/bin/matlab
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/loaddap-3.5.2/lib
```

   Check the forecast cycle options (default choice is generally fine but you may want to set RESTART=1 after the first forecast cycle)

```
# Clean results of previous forecasts
#
export CLEAN=0
#
# Get forcing files from DODS SERVER and process them for CROCO
# PRE_PROCESS=1 ==> do the work (0 otherwise)
```

(continues on next page)

(continued from previous page)

```
#
export PRE_PROCESS=1
#
# Restart from previous forecast
#
export RESTART=0
#
# Run hindcast/forecast
#
export RUN=1
#
#
# Make a few plots
export PLOT=1
#
#=====================================================================
# Define time parameters (it will modify croco_forecast.in)
#=====================================================================
#
# Model time step [seconds]
DT=3600
# Number of barotropic time steps within one baroclinic time step [number]
# NDTFAST in croco.in
NFAST=60
# Hindcast depth [days] see crocotools_param (hdays/fdays)
NDAYS_HIND=1
# Forecast depth [days]
NDAYS_FCST=3
# Output frequency [hours]
#   average
ND_AVG=24
#   history (if = -1 set equal to NUMTIMES)
ND_HIS=6
#   restart (if = -1 set equal to NUMTIMES)
ND_RST=24
```

3. Run the script (from Terminal or from Crontab)

```
./run_croco_forecast.bash
```

2.13 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 :

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

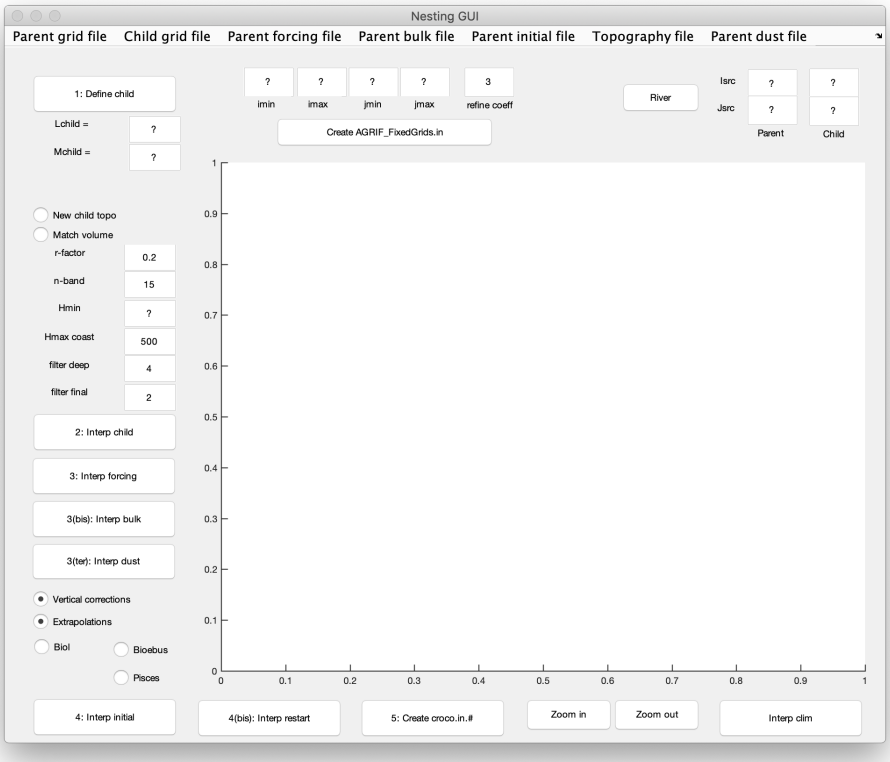


Fig. 1: Entrance window of nestgui

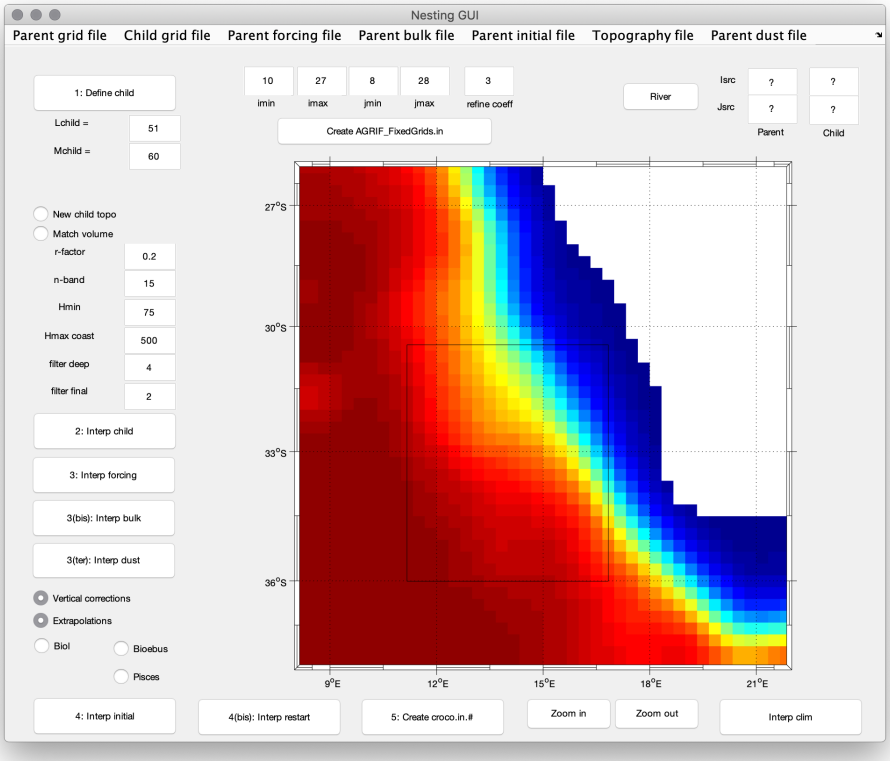


Fig. 2: Main window of nestgui

the position of the child domain. The child domain can be finely tuned using the `imin`, `imax`, `jmin` and `jmax` boxes.

> **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
```

2.14 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**

2.14.1 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
```

2.14.2 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.

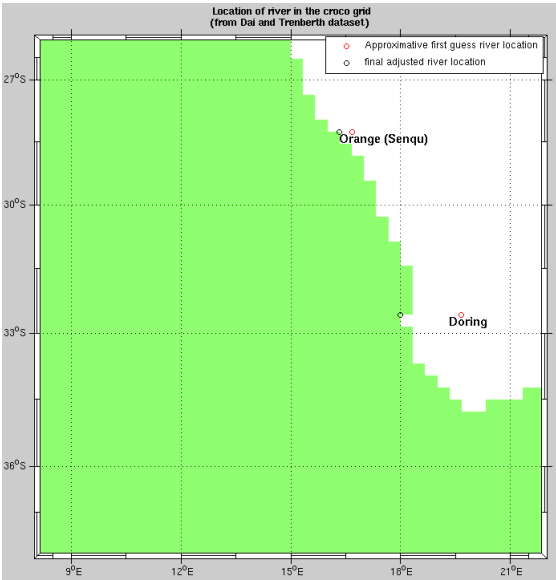


Fig. 3: First and final guess rivers positions

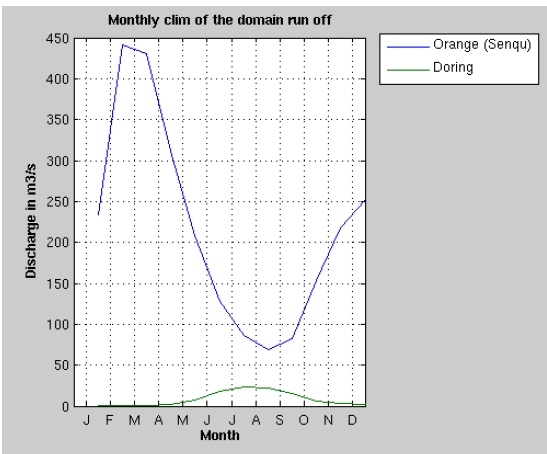


Fig. 4: Rivers flow seasonal cycle

> **Warning**
>
> The order of the sources in the `croco.in` file must match the order of the netcdf file. It's impliticly the case if you generate your runoff file with the croco_tools which give you the lines to add into your croco.in

2.14.3 Variable flow and variable concentration from a netCDF file

To run CROCO with a variable con`entration 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.

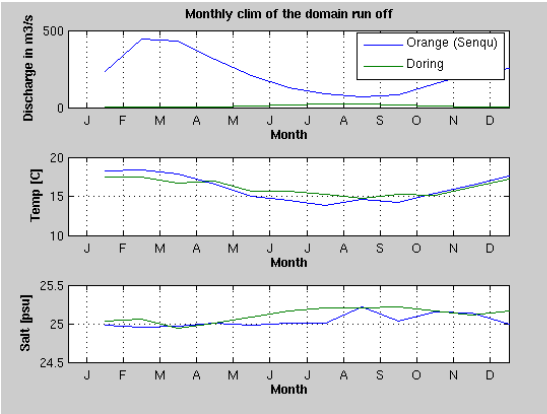


Fig. 5: Rivers tracer concentration seasonal cycle

2.14.4 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
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.

2.15 Adding tides

Using the method described by Flather and Davies [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`)

2.15.1 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

2.15.2 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
```

(continues on next page)

(continued from previous page)

```
#   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
```

2.15.3 Running

Run the model

```
qsub job_croco_mpi.pbs
```

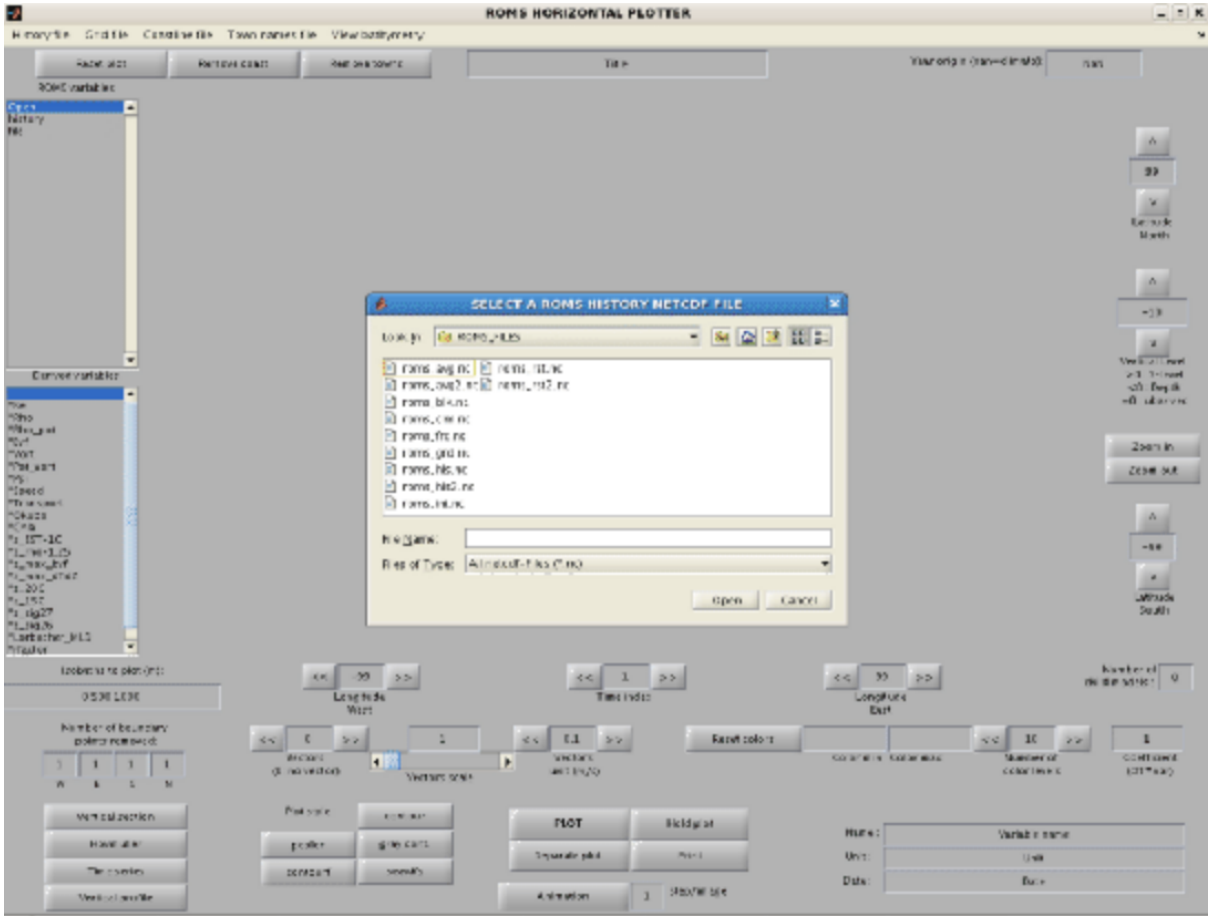
2.16 Visualization (Matlab)

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".

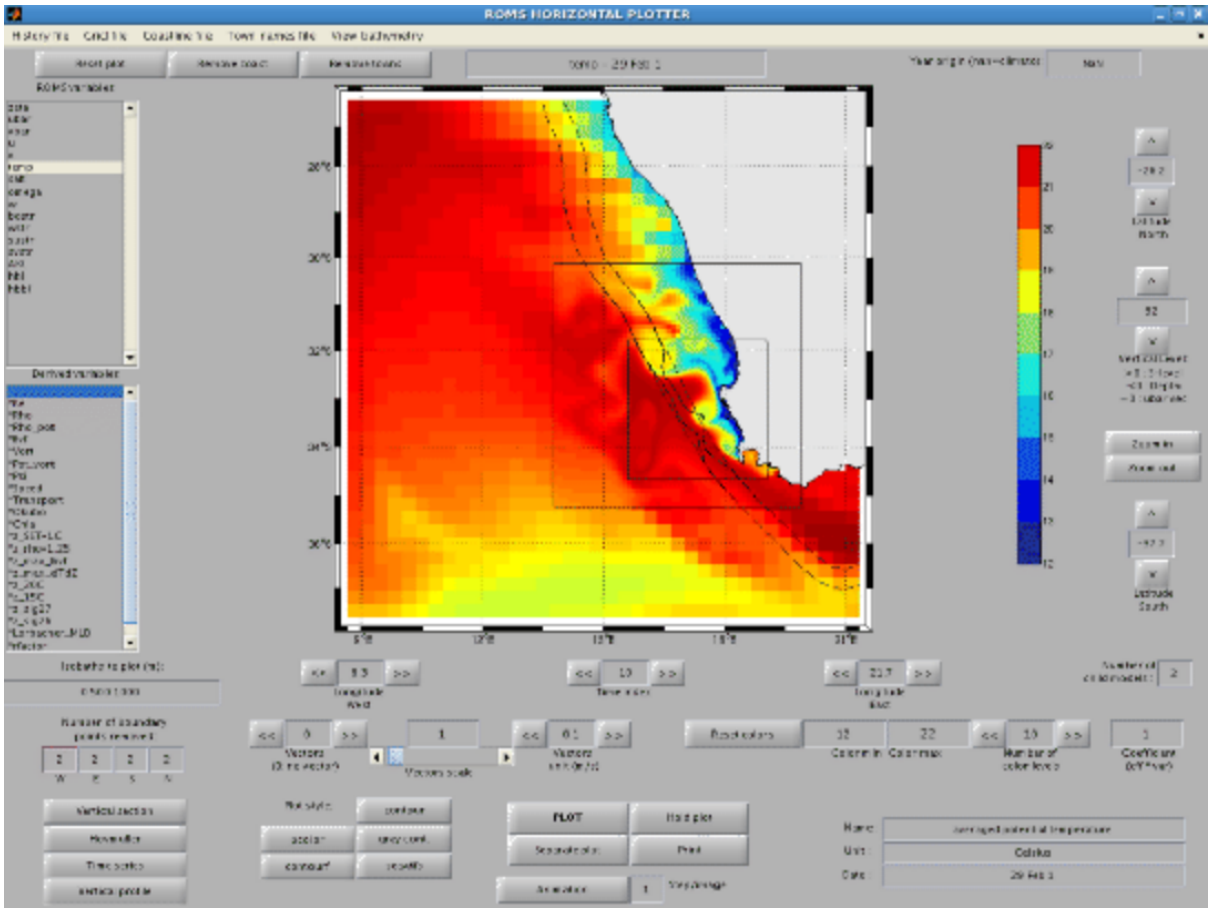


The main window appears, variables can be selected to obtain an image such as Figure below. On the left side, the upper box gives the available CROCO variable names and the lower box presents the variables derived from the CROCO model outputs :

- 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, $\lambda = \rho$
- 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.



2.17 Python tools for CROCO

As an alternative to the matlab pre- and post-processing tools for CROCO (croco_tools) developed over the last 20 years, a python toolbox is under development. All the features present in croco_tools matlab are not available in this version.

These python tools bring together the methods of several users of the CROCO community but do not constitute a definitive toolbox. A new, more complete toolbox is under development by the CROCO team.

The temporary **croco_pytools** toolbox consists of:

- ***prepro***: a set of python routines, interface with fortran to process the grid and forcing files (initialization, open-boundary conditions, tides forcing, rivers forcing)
- ***croco_pyvisu***: a vizualisation GUI
- ***xcroco***: for analysis based on xarray and xgcm

The documentation for this toolbox is available at: https://croco-ocean.gitlabpages.inria.fr/croco_pytools/

2.18 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).

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

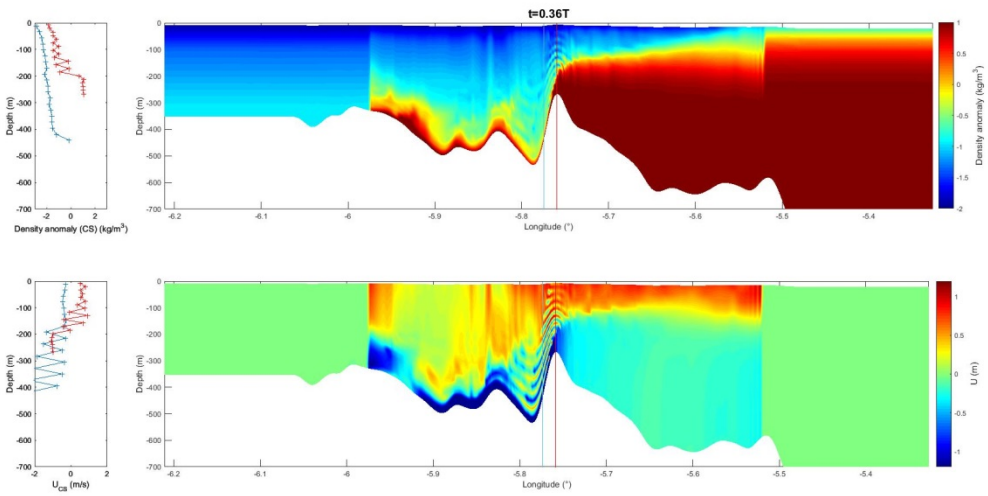
- **Momentum and tracer advection schemes :**

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).

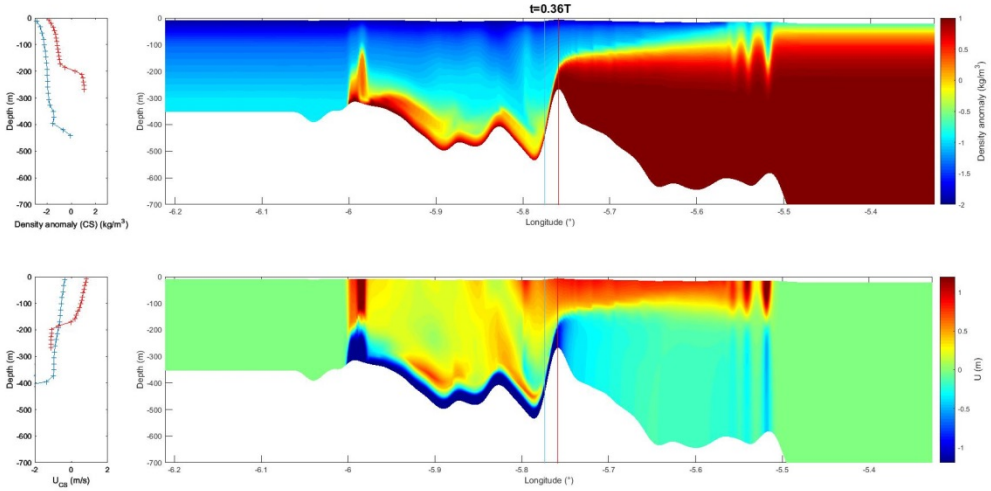
An exemple of numerical artefacs related to the advection schemes: Gibbs phenomenon

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

2.18.2 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`
```

(continues on next page)

(continued from previous page)

```
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)
```

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?

2.18.3 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).

2.18.4 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 non-linearities (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

2.18.5 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*.

Related CPP options (for developers):

NBQ_IMP	The equation of motion for vertical velocity is solved implicitly in the vertical direction.
NBQ_THET	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_P	Prognostic the grid evolution
NBQ_AM4	Classical fourth-order Adams-Moulton (AM4) time-stepping method
NOT_NBQ_	Forward-Backward time-stepping method
NBQ_MASS	Perfect conservation of mass (undef NBQ_MASS : perfect conservation of volume)
NBQ_HZCC	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	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

2.19 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`.

2.19.1 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.

2.19.2 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

2.19.2.1 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`)

2.19.2.2 Compiling CROCO

> **Note**
>
> CROCO needs to be compiled for each configuration and parallel settings. Thus the scripts provided in the `croco/SCRIPTS/SCRIPTS_COUPLING` toolbox provide an option to compile CROCO "online" when the run is launched according to the configuration, coupled options and parallel settings. Here is just a roadmap on how to do it "by hand".

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.

2.19.2.3 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 another model (wave, atmosphere, ocean). 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 use the compilation script:

```
./make_toy_compil.sh
```

If the compilation is successfull you should have the TOY executables `toy_wav toy_atm toy_oce`

2.19.2.4 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 WRF `main` directory 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
```

(continues on next page)

(continued from previous page)

```
# 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 `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 WRF `main` directory 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/.
```

Note

Using the WRF moving nest in coupled mode is possible, but only the parent static model can be coupled through OASIS. Feedback between the static parent domain and the moving nest are used to update fields

computed at high-resolution in the moving nest on one hand, and coupled to the ocean or wave model in the static parent domain on the other hand. To use this functionnality, WRF has to be compiled with the moving nest option, and a dedicated `Registry.EM` is available in `WRF_IN/FOR_MOVING_NEST` to allow the moving nest to receive surface updates from the parent static domain, that is coupled to the ocean or wave model. Copy this `Registry.EM` in your `WRF/Registry` directory before compiling with the moving nest option. Warning, this `Registry.EM` should not be used in "normal" mode (no moving nest).

2.19.2.5 Compiling WPS

Note

Note that you should use the WPS version consistent with your WRF version!

1. Enter WPS directory, and configure your compilation

```
cd ~/wrf/WPS
./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`.

2.19.2.6 Compiling WW3

Warning

Currently the following compilation procedure, and coupling tools provided in croco are designed to work with the WW3 6.07.1 release plus some additional changes in a few coupled routines in WW3 which are provided in the `croco/SCRIPTS/SCRIPTS_COUPLING/WW3_IN` directory. See `readme_ww3_version` in this directory. More recent versions of WW3 contains these modifications, but as these more recent versions are currently not tagged as "releases", we prefer to stick to the latest official release and just add the few modified routines.

1. First copy the modified routines into WW3 6.07.1 directory:

```
cp ~/croco/croco/SCRIPTS/SCRIPTS_COUPLING/WW3_IN/modified_ftn/*.ftn  ~/ww3/model/
↪ftn/.
```

2. 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.

3. 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"
```

4. 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`

5. 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/.
```

6. 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`

2.19.2.7 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

2.19.3 Simple CROCO-TOY coupled example

For this first step towards coupling, we will just use the BENGUELA_LR configuration and add coupling with 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.

2.19.3.1 Get necessary files

First create the configuration, here named `BENGUELA_TOY`:

```
mkdir BENGUELA_TOY
```

Get the useful files for CROCO compilation and settings:

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

Get the TOY model:

```
cp -r ~/croco/croco/SCRIPTS/SCRIPTS_COUPLING/TOY_IN .
```

Get the useful input files:

```
wget "https://data-croco.ifremer.fr/CONFIGS_EXAMPLES/BENGUELA_LR_INPUT_FILES/CROCO_
↪FILES.tar.gz"
tar -zxvf CROCO_FILES.tar.gz

wget "https://data-croco.ifremer.fr/CONFIGS_EXAMPLES/BENGUELA_LR_INPUT_FILES/TOY_
↪FILES.tar.gz
tar -zxvf TOY_FILES.tar.gz
```

Get some useful scripts:

```
cp -r ~/croco/croco/SCRIPTS/SCRIPTS_COUPLING/SCRIPTS_TOOLBOX/OASIS_SCRIPTS .
```

2.19.3.2 Compile

Compile OASIS

For running in coupled mode, you need to have OASIS compiled. For OASIS follow the instructions in the Compilation section of the coupling tutorial. We assume here that you have OASIS compiled in `~/oasis/compile_oasis3-mct`

Compile CROCO

In `cppdefs.h` in the "REGIONAL (realistic) Configurations" section:

```
#define MPI
#define OW_COUPLING
#define MRL_WCI
```

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

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
```

And compile:

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

If the compilation is successfull you should have the CROCO executable `croco`.

Compile the TOY model

A script `make_toy_compil.sh` is provided. Check and eventually edit it (notably the line regarding the environment `source ../myenv_mypath.sh`, that should be adapt to point towards you environment file where the compilers and libraries are defined). You should also check/edit the `Makefile.MACHINE` for your machine.

```
cd TOY_IN
ln -sf Makefile.YOURMACHINE Makefile
./make_toy_compil.sh
```

If the compilation is successfull you should have the TOY executables `toy_wav toy_atm toy_oce`

2.19.3.3 Prepare the configuration files for a 3-day run

Set up CROCO

Edit the `croco.in`:

```
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
wci_history_fields:  SUP UST2D VST2D UST VST WST AKB AKW KVF CALP KAPS
```

(continues on next page)

(continued from previous page)

```
                        20*T
wci_average_fields:  SUP UST2D VST2D UST VST WST AKB AKW KVF CALP KAPS
                        20*T
```

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 need to provide a grid to the toy model, here named `grid_wav.nc`. Here, the toy model will read and exchange variables specified in the `TOYNAMELIST.nam` from an input file, here named `toy_wav.nc`.

Get the chosen `TOYNAMELIST.nam.wav.ow` and rename it:

```
cp TOY_IN/TOYNAMELIST.nam.wav.ow TOYNAMELIST.nam.wav
```

You need to edit all the fields denoted into brackets: `<...>`, as well as the paths towards the input files in `TOYNAMELIST.nam.wav`:

```
&NAM_OASIS NB_TIME_STEPS=12,
            DELTA_T=21600,
            GRID_FILENAME='grid_wav.nc' /

&NAM_FCT_SEND CTYPE_FCT='FILES',
               CNAME_FILE='toy_wav.nc',
               VALUE=1 /

&NAM_RECV_FIELDS NB_RECV_FIELDS=3,
                  CRCVFIELDS(1)='TOY__SSH',
                  CRCVFIELDS(2)='TOY_UOCE',
                  CRCVFIELDS(3)='TOY_VOCE' /

&NAM_SEND_FIELDS NB_SEND_FIELDS=3,
                  CSNDFIELDS(1)='TOY_T0M1',
                  CSNDFIELDS(2)='TOY___HS',
                  CSNDFIELDS(3)='TOY__DIR' /
```

In the current example, the toy model is set to run 12 time steps of 21600s.

Also, prepare the toy input files. To do so you can use the script provided in `OASIS_SCRIPTS/create_oasis_toy_files.sh`:

```
./OASIS_SCRIPTS/create_oasis_toy_files.sh TOY_FILES/ww3_20050101_20050131.nc toy_wav.
↪nc ww3 1,12
```

You should now have `toy_wav.nc` and `grid_wav.nc` files.

2.19.3.4 Prepare OASIS files

Edit OASIS namelist, `namcouple`, which specifies which fields will be coupled, and at which frequency, etc.

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 namcouple
```

In this `namcouple`, **you 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 6: 3 from the ocean to**
  the wave model (SSH, UOCE, VOCE), and 3 from the wave to the ocean model (HS, T0M1, DIR))
- **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
cp TOY_IN/toy_wav toywav
```

- **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:

```
CROCO_SSH 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> -> 21600	the coupling frequency in seconds for each field you will exchange
<ocenx> -> 41	the number of points in xi direction for CROCO (see param.h)
<oceny> -> 42	the number of points in eta direction for CROCO (see param.h)
<wavnx> -> 41	the number of points in x direction for the TOY model (see grid_wav.nc file)
<wavny> -> 42	the number of points in y direction for the TOY model (see grid_wav.nc file)
<ocedt> -> 3600	the CROCO time step
<wavdt> -> 21600	the TOY model time step (see TOYNAMELIST.nam)

Then, 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. Here, 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.

First, 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

```
./OASIS_SCRIPTS/create_oasis_restart_from_calm_conditions.sh grid_wav.nc␣
↪wav.nc toy "TOY_T0M1 TOY___HS TOY__DIR"
```

Then, do the same for the restart file for OASIS for CROCO model:

```
./OASIS_SCRIPTS/create_oasis_restart_from_calm_conditions.sh CROCO_FILES/croco_grd.nc␣
↪oce.nc croco "CROCO_SSH CROCO_EOCE CROCO_NOCE"
```

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

2.19.3.5 Run the models

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

```
mpirun -np 2 toywav : -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
toywav.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???????)
```

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 you should see small differences of the surface currents for example if you do a difference of coupled and non-coupled CROCO outputs).

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

2.19.4 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.

2.19.4.1 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.MACHINE	Example files for OASIS compilation on different MACHINES
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`
cppdefs.h.base	Base cppdefs.h file for CROCO in which coupling options will be replaced by `oce_compile.sh` from `SCRIPTS_TOOLBOX`
param.h.base	Base param.h file for CROCO in which grid size and MPI options will be replaced by `oce_compile.sh` from `SCRIPTS_TOOLBOX`
jobcomp	Script to compile CROCO, with adapted paths and environment variables

In `WRF_IN`:

compile.wrf.*	Jobs to launch `make_WRF_compil` on some MACHINES
make_WRF_compil	Script to compile wrf
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
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_ounp.inp.base	ounp input file in which dates will be replaced
ww3_bounc.inp	boundary input file for running `ww3_bounc`
make_WW3_compil	Script to compile ww3
modified_ftn	Directory with a few ww3 functions modified from v6.07.1

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 (multiple executables launch on HPC clusters)
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

> **Note**
>
> 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

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 supercomputer
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

2.19.4.2 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
```

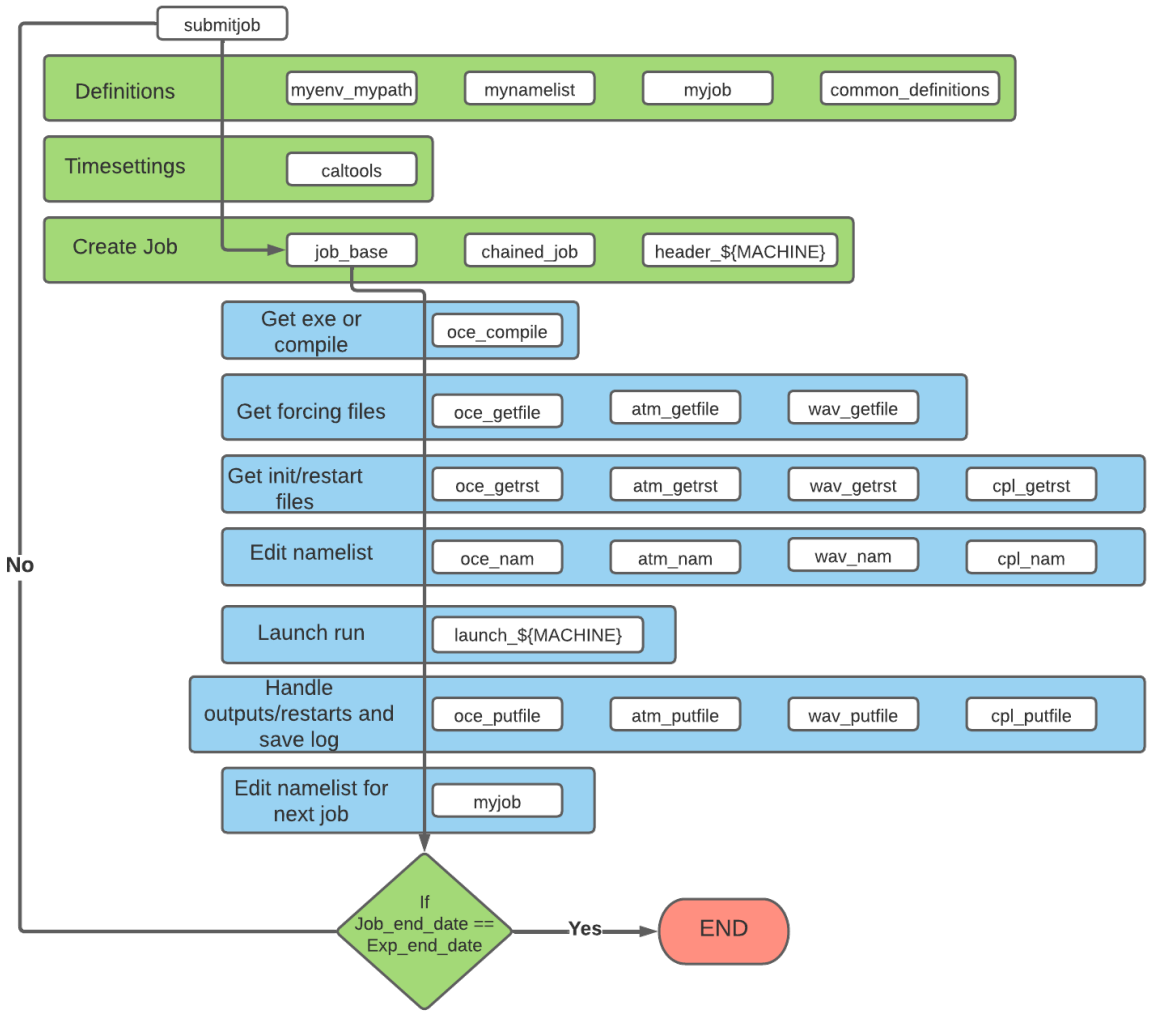
The user will provide:

- 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



2.19.4.3 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
```

(continues on next page)

(continued from previous page)

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

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

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

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.

2.19.4.4 Pre-processing for coupled run

2.19.4.4.1 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.

2.19.4.4.2 WW3 pre-processing

2.19.4.4.2.1 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;
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');
     ```

2.19.4.4.2.2 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.

2.19.4.4.2.3 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`.

2.19.4.4.3 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).

2.19.4.4.3.1 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

2.19.4.4.3.2 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.real
- namelist.input.base.complete

1. Edit all the required lines in configure.namelist.real, and edit all the required paths in run_real.bash
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.real 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
```

2.19.4.4.3.3 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/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.

2.19.4.4.4 OASIS pre-processing

In `SCRIPTS_TOOLBOX/OASIS_SCRIPTS` 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 calm or preexisting model outputs: `create_oasis_restart_from_calm_conditions.sh` `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.

2.19.4.5 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:

2.19.4.5.1 CROCO-TOY (wav or atm)

`myenv_mypath.sh` should already have been filled in before the compilation.

In `TOY_IN`, you must have the executable `toy_wav`. First go to the "Compiling in coupled mode" section otherwise.

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

- In `myjob.sh` , you will have to fill in information about dates, job sequence:

```
# 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
```

- In `mynamelist.sh`, first specify the run type, and the name of the experiment:

```
# Run type (o/a/w, w.Afrc, oa, 2o1a, owa, owa.full...)
#  - Will select the models to use reading letters o/w/a/toywav/toyoce/toyatm
#  - Will select the executables, and some options (see in the following␣
↪sections)
#  - In coupled mode corresponds to the suffix of the OASIS_IN/namcouple.base.
↪$RUNtype to use
export RUNtype=ow.toywav
export MOD=`echo $RUNtype | cut -d . -f 1`

#  Name of the experiment you are about to launch (max 30. CHAR)
export CEXPER=BENGUELA_example_${RUNtype}
```

Then, there is a section indicating where the run will be executed and where the outputs and restarts will be stored:

```
#-----------------------------------------------------------------------------
# RUN_DIR
#-----------------------------------------------------------------------------

export EXEDIR_ROOT="$CWORK/rundir/${CEXPER}_execute"
export OUTPUTDIR_ROOT="$CWORK/rundir/${CEXPER}_outputs"
export RESTDIR_ROOT="$CWORK/rundir/${CEXPER}_restarts"

export  JOBDIR_ROOT=${CHOME}/jobs_${CEXPER}
```

Then, there are sections for the different components.

For the coupler settings:

```
#-------------------------------------------------------------------------------
# CPL
#-------------------------------------------------------------------------------

# Namelist
#---------
# Note: namelist example files are provided in OASIS_IN/
# if you want to use a pre-built weight file for grid interpolations, point to
# e.g. namcouple.base.oa.smtho2a
export namcouplename=namcouple.base.${RUNtype}

# Coupling frequency
#-------------------
export CPL_FREQ=21600

# Restart files for OASIS
#------------------------
# If TRUE: create OASIS restart files from pre-existing atm/oce/wav outputs.
# If FALSE: create OASIS restart files from calm conditions (need to read at␣
↪least the grid for each model)
export CPL_restart="FALSE"
export oce_rst_file="${OCE_FILES_DIR}/croco_grd.nc"
export oce_rst_timeind=-1 # time index (-1 is last) in the file to extract as␣
↪restart
export atm_rst_file="${ATM_FILES_DIR}/wrfinput_dXX_2005_01_01_00" # the domain␣
↪dXX will be automatically replaced
export atm_rst_timeind=-1 # time index (-1 is last) in the file to extract as␣
↪restart
export wav_rst_file="${WAV_FILES_DIR}/ww3.200501.nc"
export wav_rst_timeind=-1 # time index (-1 is last) in the file to extract as␣
↪restart
```

You should check the coupling frequency, the restart flag and path towards model files to use to create oasis restart files.

For CROCO settings, first indicate if you request CROCO compilation:

```
#-------------------------------------------------------------------------------
# OCE
#-------------------------------------------------------------------------------

# Where to find or put the croco execuatble
export OCE_EXE_DIR=${CHOME}/CROCO_IN

# Online Compilation
#-------------------
#!!!!!!! IMPORTANT NOTE !!!!!!!
# If activated, creates croco executable depending on this namelist.
#   - In param.h it modifies the grid size, the number of procs in x and y␣
↪direction with those given in myjob.sh
#   - In cppdefs.h it modifies the following options with informations given␣
↪below
#       MPI, OA_COUPLING, OW_COUPLING, MRL_WCI,
#       XIOS, LOGFILE, MPI_NOLAND,
#       AGRIF, AGRIF_2WAY,
```

(continues on next page)

(continued from previous page)

```
#       BULK_FLUX, ONLINE, AROME, ARPEGE, ERA_ECMWF
#       FRC_BRY, CLIMATOLOGY
#       TIDES, PSOURCE, PSOURCE_NCFILE, PSOURCE_NCFILE_TS
# Other changes of parameterizations, numerical schemes, etc should be made "by␣
↪hand" in CROCO_IN/cppdefs.h.base
#!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
export ONLINE_COMP=1
```

Then, edit model time steps:

```
# Time steps
#-----------
export DT_OCE=3600
export NDTFAST=60
```

Then several options for zooms, wave coupling are provided. They are generally automatically set-up depending on the `RUNtype` defined at the beginning.

Then, edit the options regarding the forcing files:

```
# Forcings
#---------
export ini_ext='ini_SODA' # ini extension file (ini_SODA,...)
export bdy_ext='bry_SODA' # bry extension file (clm_SODA,bry_SODA,...)

# flag for surface forcing should be true except in the case of atm coupling
if [[ $MOD =~ .*a.* ]] ; then
    export surfrc_flag="FALSE"
else
    export surfrc_flag="TRUE"
fi
export interponline=0 # switch (1=on, 0=off) for online surface interpolation.␣
↪Only works with MONTHLY input files!
export frc_ext='blk_ERA5' # surface forcing extension(blk_ERA5, frc_ERA5,...).␣
↪If interponline=1 precise the type (ERA_ECMWF or AROME,  [CFSR by default],␣
↪names as cppkey name in croco)

export tide_flag="FALSE" # the forcing extension must be blk_??? otherwise tide␣
↪forcing overwrites it
export river_flag="FALSE"
```

Finally edit CROCO output settings:

```
# Output settings
#----------------
#!!! WARNING: when XIOS is activated the following values (for the model) are␣
↪not taken into account
export oce_his_sec=86400     # history output interval (in number of second)
export oce_avg_sec=86400     # average output interval (in number of second)
```

Then go down to the TOY model section, and set the toy options, and model files the toy model should use:

```
#-------------------------------------------------------------------------------
# TOY
#-------------------------------------------------------------------------------

# Where to find the toy executable(s)
```

(continues on next page)

(continued from previous page)

```
export TOY_EXE_DIR=${CHOME}/TOY_IN

# Choose for which model you use the toy
# If several separate with spaces
# options are: oce atm wav
#-------------------------------------
export toytype=("wav")

# Forcing files that will be read by the toy
#-----------------------------------------
export toyfile=("$CWORK/TOY_FILES/ww3.201301.nc")
export toytimerange=('2,124')
```

- Now that you have completed the necessary files, you are ready to run your simulation. To do so, simply do

```
./submitjob.sh
```

2.19.4.5.2 CROCO-WRF

In your ${CHOME} repository you should have already filled in `myenv_mypath.sh`.

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

- In `myjob.sh` , you will have to fill in information about dates, job sequence:

```
# 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=14
```

- In `mynamelist.sh`, first specify the run type, and the name of the experiment:

```
# Run type (o/a/w, w.Afrc, oa, 2o1a, owa, owa.full...)
#  - Will select the models to use reading letters o/w/a/toywav/toyoce/toyatm
#  - Will select the executables, and some options (see in the following␣
↪sections)
#  - In coupled mode corresponds to the suffix of the OASIS_IN/namcouple.base.
↪$RUNtype to use
export RUNtype=oa
export MOD=`echo $RUNtype | cut -d . -f 1`

#  Name of the experiment you are about to launch (max 30. CHAR)
export CEXPER=BENGUELA_example_${RUNtype}
```

Then, there is a section indicating where the run will be executed and where the outputs and restarts will be stored:

```
#-------------------------------------------------------------------------------
# RUN_DIR
#-------------------------------------------------------------------------------

export EXEDIR_ROOT="$CWORK/rundir/${CEXPER}_execute"
export OUTPUTDIR_ROOT="$CWORK/rundir/${CEXPER}_outputs"
export RESTDIR_ROOT="$CWORK/rundir/${CEXPER}_restarts"

export  JOBDIR_ROOT=${CHOME}/jobs_${CEXPER}
```

Then, there are sections for the different components.

For the coupler settings:

```
#-------------------------------------------------------------------------------
# CPL
#-------------------------------------------------------------------------------

# Namelist
#---------
# Note: namelist example files are provided in OASIS_IN/
# if you want to use a pre-built weight file for grid interpolations, point to
# e.g. namcouple.base.oa.smtho2a
export namcouplename=namcouple.base.${RUNtype}

# Coupling frequency
#-------------------
export CPL_FREQ=21600

# Restart files for OASIS
#------------------------
# If TRUE: create OASIS restart files from pre-existing atm/oce/wav outputs.
# If FALSE: create OASIS restart files from calm conditions (need to read at␣
↪least the grid for each model)
export CPL_restart="FALSE"
export oce_rst_file="${OCE_FILES_DIR}/croco_grd.nc"
export oce_rst_timeind=-1 # time index (-1 is last) in the file to extract as␣
↪restart
export atm_rst_file="${ATM_FILES_DIR}/wrfinput_dXX_2005_01_01_00" # the domain␣
↪dXX will be automatically replaced
export atm_rst_timeind=-1 # time index (-1 is last) in the file to extract as␣
↪restart
export wav_rst_file="${WAV_FILES_DIR}/ww3.200501.nc"
```

(continues on next page)

(continued from previous page)

```
export wav_rst_timeind=-1 # time index (-1 is last) in the file to extract as␣
↪restart
```

You should check the coupling frequency, the restart flag and path towards model files to use to create oasis restart files.

For CROCO settings, first indicate if you request CROCO compilation:

```
#-----------------------------------------------------------------------------
# OCE
#-----------------------------------------------------------------------------

# Where to find or put the croco execuatble
export OCE_EXE_DIR=${CHOME}/CROCO_IN

# Online Compilation
#------------------
#!!!!!!! IMPORTANT NOTE !!!!!!!
# If activated, creates croco executable depending on this namelist.
#   - In param.h it modifies the grid size, the number of procs in x and y␣
↪direction with those given in myjob.sh
#   - In cppdefs.h it modifies the following options with informations given␣
↪below
#       MPI, OA_COUPLING, OW_COUPLING, MRL_WCI,
#       XIOS, LOGFILE, MPI_NOLAND,
#       AGRIF, AGRIF_2WAY,
#       BULK_FLUX, ONLINE, AROME, ARPEGE, ERA_ECMWF
#       FRC_BRY, CLIMATOLOGY
#       TIDES, PSOURCE, PSOURCE_NCFILE, PSOURCE_NCFILE_TS
# Other changes of parameterizations, numerical schemes, etc should be made "by␣
↪hand" in CROCO_IN/cppdefs.h.base
#!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
export ONLINE_COMP=1
```

Then, edit model time steps:

```
# Time steps
#-----------
export DT_OCE=3600
export NDTFAST=60
```

Then several options for zooms, wave coupling are provided. They are generally automatically set-up depending on the `RUNtype` defined at the beginning.

Then, edit the options regarding the forcing files:

```
# Forcings
#---------
export ini_ext='ini_SODA' # ini extension file (ini_SODA,...)
export bdy_ext='bry_SODA' # bry extension file (clm_SODA,bry_SODA,...)

# flag for surface forcing should be true except in the case of atm coupling
if [[ $MOD =~ .*a.* ]] ; then
    export surfrc_flag="FALSE"
else
    export surfrc_flag="TRUE"
fi
export interponline=0 # switch (1=on, 0=off) for online surface interpolation.␣
```

(continues on next page)

(continued from previous page)

```
↪Only works with MONTHLY input files!
export frc_ext='blk_ERA5' # surface forcing extension(blk_ERA5, frc_ERA5,...).␣
↪If interponline=1 precise the type (ERA_ECMWF or AROME,  [CFSR by default],␣
↪names as cppkey name in croco)

export tide_flag="FALSE" # the forcing extension must be blk_??? otherwise tide␣
↪forcing overwrites it
export river_flag="FALSE"
```

Finally edit CROCO output settings:

```
# Output settings
#----------------
#!!! WARNING: when XIOS is activated the following values (for the model) are␣
↪not taken into account
export oce_his_sec=86400     # history output interval (in number of second)
export oce_avg_sec=86400     # average output interval (in number of second)
```

Then continue with the atmospheric model section:

```
#-------------------------------------------------------------------------------
# ATM
#-------------------------------------------------------------------------------

# Where to find the atm exectuable
if [[ $RUNtype =~ .*a.* && ( $RUNtype =~ .*o.* || $RUNtype =~ .*w.* ) ]] ; then
    export ATM_EXE_DIR=${ATM}/exe_coupled
else
    export ATM_EXE_DIR=${ATM}/exe_uncoupled
fi
```

First the paths for the executable are defined. Those are generally set-up with the `RUNtype` but can be changed if necessary.

Then choose the namelist file, and namelist options: model time step, forcing files informations, Cd formulation (coupling with waves requires isftcflx=5, otherwise choose another Cd formulation, see `WRF_IN/README.namelist`), domains informations.

> **Note**
>
> Other changes in the WRF namelist should be made "by hand" in the `WRF_IN/namelist.input.base.complete`. Only settings into <...> in the namelist are automatically changed by the scripts

```
# Namelist
#---------
#!!!!!!! IMPORTANT NOTE !!!!!!!
# Changes of parameterizations, numerical schemes, etc in atmnamelist
# should be made "by hand" in the WRF_IN/namelist.input.base.complete file
# Only settings into <...> in WRF_IN/namelist.input.base.complete are␣
↪automatically
# filled in by the present mynamelist settings
#!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
export atmnamelist=namelist.input.base.complete

# Time steps
#-----------
```

(continues on next page)

(continued from previous page)

```
export DT_ATM=150

# Boundaries
#-----------
export interval_seconds=21600 # interval ( in sec ) of the lateral input
export auxinput4_interval=360 # interval ( in min ) of bottom input
export nbmetsoil=4
export nbmetlevel=38

# Physics
#--------
 # Cd formulation (default = 0, wave cpl needs = 5)
if [[ $RUNtype =~ .*aw.* || $RUNtype =~ .*owa.* ]] ; then
    export isftcflx=5
else
    export isftcflx=1
fi

# Domains
#--------
export NB_dom=1 # Number of coupled domains
export wrfcpldom='d01' # which WRF domain to couple
export nestfeedback="TRUE" # 1 way (FALSE) or 2 Way (TRUE) nesting
export onlinecplmask="TRUE" # Erase existing CPLMASK and build default mask␣
↪(depending on the nb of atm and oce domains)
```

Then a section when using moving nest is available. It is important to note that to use the moving nest WRF has to be compiled with the moving nest option. In addition, in coupled mode, the moving nest can be used, but only the parent static model can be coupled through OASIS. A dedicated `Registry.EM` is available in `WRF_IN/FOR_MOVING_NEST` to compile WRF with moving nest + coupling so that the moving nest receive surface updates from the parent static domain, that is coupled to the ocean or wave model. The following section is used only if the moving nest is activated `ATM_CASE="MOVING_NEST"`. In other case keep `ATM_CASE="DEFAULT"`, and ignore the rest of the section.

```
# Moving nest
#------------
export ATM_CASE="DEFAULT"  # no moving nest: DEFAULT or with: MOVING_NEST
# if ATM_CASE=DEFAULT, the following is not used
export num_mv_nest=1 # number of moving nests
# if several nest, the following variables need to have the format "1st_nest 2nd_
↪nest"
export ref_coef="3" # refinement coef for nest
export ew_size="283" # nest size in east-west dim ([multiple of ref_coef] + 1)
export ns_size="295" # nest size in north-south dim ([multiple of ref_coef] + 1)
export i_prt_strt="580" # where nest is starting in parent's grid x-dim
export j_prt_strt="59" # where nest is starting in parent's grid y-dim
# Tracking parameters
export vortex_interval=5 # When to update vortex position
export max_vortex_speed=40 # Used to compute the search radius for the new␣
↪vortex center position
export corral_dist=8 # The closest distance between child and parend boundary␣
↪(in parent grid cell)
export track_level=50000 # The pressure level (in Pa) where the vortex is tracked
export time_to_move=0 # The time (in minutes) until the nest is moved (at the␣
↪beginning)
```

Then a section when using nudging (assimilation) is available. If not using nudging, ignore this section.

```
# Nudging (assimilation) options
#-------------------------------
export switch_fdda=0 # To activate fdda nudging
export nudgedom="1" # select which kind of nudging you want (1=grid-nudging,␣
↪2=spectral nudging) for each domain. Example for spectral nudging over parent␣
↪only "2 0"
export nudge_coef="0.0003" # nudge coef. Need to be the same size than nudge
export nudge_interval_m="360" # time interval (in min) between analysis times
export nudge_end_h="144" # time (in hours) to stop nudging after start of␣
↪forecast
```

Finally, set the ouptut settings:

```
# 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
# file for specifying different than default output variables: OPTIONAL, leave␣
↪empty if not used
export atm_iofields='myoutfields.txt'
```

- Now that you have completed the necessary files, you are ready to run your simulation. To do so, simply do

```
./submitjob.sh
```

Note

If using older versions of the wrf-croco fork than the latest tag, you may encounter issues regarding a namelist variable named max_cpldom, which is present in the up-to-date version, but was inexistent in previous version (with older version, you should remove this variable from your namelist.input.base.complete file). We encourage to use the tagged up-to-date version.

2.19.4.5.3 CROCO-WW3

In your ${CHOME} repository you should have already filled in `myenv_mypath.sh`.

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

- In `myjob.sh` , you will have to fill in information about dates, job sequence:

```
# 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
```

(continues on next page)

(continued from previous page)

```
# 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
```

- In `mynamelist.sh`, first specify the run type, and the name of the experiment:

```
# Run type (o/a/w, w.Afrc, oa, 2o1a, owa, owa.full...)
#  - Will select the models to use reading letters o/w/a/toywav/toyoce/toyatm
#  - Will select the executables, and some options (see in the following␣
↪sections)
#  - In coupled mode corresponds to the suffix of the OASIS_IN/namcouple.base.
↪$RUNtype to use
export RUNtype=ow
export MOD=`echo $RUNtype | cut -d . -f 1`

#  Name of the experiment you are about to launch (max 30. CHAR)
export CEXPER=BENGUELA_example_${RUNtype}
```

Then, there is a section indicating where the run will be executed and where the outputs and restarts will be stored:

```
#-----------------------------------------------------------------------------
# RUN_DIR
#-----------------------------------------------------------------------------

export EXEDIR_ROOT="$CWORK/rundir/${CEXPER}_execute"
export OUTPUTDIR_ROOT="$CWORK/rundir/${CEXPER}_outputs"
export RESTDIR_ROOT="$CWORK/rundir/${CEXPER}_restarts"

export  JOBDIR_ROOT=${CHOME}/jobs_${CEXPER}
```

Then, there are sections for the different components.

For the coupler settings:

```
#-----------------------------------------------------------------------------
# CPL
#-----------------------------------------------------------------------------

# Namelist
#---------
# Note: namelist example files are provided in OASIS_IN/
# if you want to use a pre-built weight file for grid interpolations, point to
```

(continues on next page)

(continued from previous page)

```
# e.g. namcouple.base.oa.smtho2a
export namcouplename=namcouple.base.${RUNtype}

# Coupling frequency
#-------------------
export CPL_FREQ=21600

# Restart files for OASIS
#------------------------
# If TRUE: create OASIS restart files from pre-existing atm/oce/wav outputs.
# If FALSE: create OASIS restart files from calm conditions (need to read at␣
↪least the grid for each model)
export CPL_restart="FALSE"
export oce_rst_file="${OCE_FILES_DIR}/croco_grd.nc"
export oce_rst_timeind=-1 # time index (-1 is last) in the file to extract as␣
↪restart
export atm_rst_file="${ATM_FILES_DIR}/wrfinput_dXX_2005_01_01_00" # the domain␣
↪dXX will be automatically replaced
export atm_rst_timeind=-1 # time index (-1 is last) in the file to extract as␣
↪restart
export wav_rst_file="${WAV_FILES_DIR}/ww3.200501.nc"
export wav_rst_timeind=-1 # time index (-1 is last) in the file to extract as␣
↪restart
```

You should check the coupling frequency, the restart flag and path towards model files to use to create oasis restart files.

For CROCO settings, first indicate if you request CROCO compilation:

```
#-----------------------------------------------------------------------------
# OCE
#-----------------------------------------------------------------------------

# Where to find or put the croco execuatble
export OCE_EXE_DIR=${CHOME}/CROCO_IN

# Online Compilation
#-------------------
#!!!!!!! IMPORTANT NOTE !!!!!!!
# If activated, creates croco executable depending on this namelist.
#   - In param.h it modifies the grid size, the number of procs in x and y␣
↪direction with those given in myjob.sh
#   - In cppdefs.h it modifies the following options with informations given␣
↪below
#       MPI, OA_COUPLING, OW_COUPLING, MRL_WCI,
#       XIOS, LOGFILE, MPI_NOLAND,
#       AGRIF, AGRIF_2WAY,
#       BULK_FLUX, ONLINE, AROME, ARPEGE, ERA_ECMWF
#       FRC_BRY, CLIMATOLOGY
#       TIDES, PSOURCE, PSOURCE_NCFILE, PSOURCE_NCFILE_TS
# Other changes of parameterizations, numerical schemes, etc should be made "by␣
↪hand" in CROCO_IN/cppdefs.h.base
#!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
export ONLINE_COMP=1
```

Then, edit model time steps:

```
# Time steps
#-----------
export DT_OCE=3600
export NDTFAST=60
```

Then several options for zooms, wave coupling are provided. They are generally automatically set-up depending on the `RUNtype` defined at the beginning.

Then, edit the options regarding the forcing files:

```
# Forcings
#---------
export ini_ext='ini_SODA' # ini extension file (ini_SODA,...)
export bdy_ext='bry_SODA' # bry extension file (clm_SODA,bry_SODA,...)

# flag for surface forcing should be true except in the case of atm coupling
if [[ $MOD =~ .*a.* ]] ; then
    export surfrc_flag="FALSE"
else
    export surfrc_flag="TRUE"
fi
export interponline=0 # switch (1=on, 0=off) for online surface interpolation.␣
↪Only works with MONTHLY input files!
export frc_ext='blk_ERA5' # surface forcing extension(blk_ERA5, frc_ERA5,...).␣
↪If interponline=1 precise the type (ERA_ECMWF or AROME,  [CFSR by default],␣
↪names as cppkey name in croco)

export tide_flag="FALSE" # the forcing extension must be blk_??? otherwise tide␣
↪forcing overwrites it
export river_flag="FALSE"
```

Finally edit CROCO output settings:

```
# Output settings
#----------------
#!!! WARNING: when XIOS is activated the following values (for the model) are␣
↪not taken into account
export oce_his_sec=86400     # history output interval (in number of second)
export oce_avg_sec=86400     # average output interval (in number of second)
```

Then go to the WAVE model section:

```
#-------------------------------------------------------------------------------
# WAV
#-------------------------------------------------------------------------------

# Where to find the wav executable
if [[ $RUNtype =~ .*owa.* ]] ; then
    export WAV_EXE_DIR=${WAV}/exe_owa
elif [[ $RUNtype =~ .*ow.* ]] ; then
    export WAV_EXE_DIR=${WAV}/exe_ow
elif [[ $RUNtype =~ .*aw.* ]] ; then
    export WAV_EXE_DIR=${WAV}/exe_aw
else
    export WAV_EXE_DIR=${WAV}/exe_frc
fi

# Namelist
```

(continues on next page)

(continued from previous page)

```
#---------
# Chosing the ww3_shel.inp.base.SHELL_EXT (see options in WW3_IN)
if [[ $RUNtype =~ .*toy.* ]] ; then
    export SHELL_EXT=$MOD
else
    export SHELL_EXT=$RUNtype
fi
```

First the paths for the executable are defined, and the ww3_shel namelist to use is defined. Those are generally set-up with the `RUNtupe` but can be changed if necessary.

Then edit the ww3 time steps and grid settings (these will be updated in ww3_grid.inp):

```
# Time steps
#-----------
export DT_WAV=3600      # TMAX = 3*TCFL
export DT_WW_PRO=1200   # TCFL = 0.8 x dx/(g/fmin4pi) with fmin=0.0373 => 3-4 %␣
↪of dx
export DT_WW_REF=1800   # TMAX / 2
export DT_WW_SRC=10     # TSRC = usually 10s  (could be between 5s and 60s)

# Grid size
#----------
export wavnx=41 ; export wavny=42

# Parameter
#----------
export hmin=75; # e.g. minimum water depth in CROCO (will be used to delimit␣
↪coastline in WW3)
```

Then, choose the forcing to use for ww3:

```
# Forcing files
#--------------
# forcin: forcing file(s) PREFIX list (input file are supposed to be in the␣
↪form: PREFIX_Y????M??.nc)
# forcww3: name of ww3_prnc.inp extension, e.g current or wind/era5, see in WW3_
↪IN directory
if [[ $RUNtype =~ .*owa.* ]]; then
    export forcin=()
    export forcww3=()
elif [[ $RUNtype =~ .*Afrc.* || $RUNtype =~ .*ow.* ]] ; then
    export forcin=(ERA5_wind)
    export forcww3=(wind.era5)
elif [[ $RUNtype =~ .*Ofrc.* ]] ; then
    export forcin=(CROCO_current CROCO_level)
    export forcww3=(current level)
elif [[ $RUNtype =~ .*frc.* ]] ; then
    export forcin=(ERA5_wind CROCO_current CROCO_level)
    export forcww3=(wind.era5 current level)
fi
```

As well as the boundary data (can be left empty):

```
# Boundary files
#---------------
# prefix for boundary files (leave empty is none), there are supposed to be in␣
```

(continues on next page)

(continued from previous page)

```
↪WAV_FILES_DIR
export bouncin=
```

Finally, set output settings:

```
# Output settings
#----------------
export wav_int=21600  # output interval (s)
export wav_pnt=0  # point output interval. Put 0 if no point output
export point_output_list=${WAV_FILES_DIR}/my_point_output_test.txt # file where␣
↪to find list of point (format: lon lat name) to output spectrum
export wav_trck=0     # track output interval. Put 0 if no track output
export flagout="TRUE" # Keep (TRUE) or not (FALSE) the ww3 output binary files␣
↪(e.g. out_grd.ww3)
```

- Now that you have completed the necessary files, you are ready to run your simulation. To do so, simply do

```
./submitjob.sh
```

2.19.4.5.4 CROCO-WW3-WRF

In your ${CHOME} repository you should have already filled in `myenv_mypath.sh`.

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

- In `myjob.sh` , you will have to fill in information about dates, job sequence:

```
# 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=14
export NP_WAV=4
```

- In `mynamelist.sh`, first specify the run type, and the name of the experiment:

```
# Run type (o/a/w, w.Afrc, oa, 2o1a, owa, owa.full...)
#  - Will select the models to use reading letters o/w/a/toywav/toyoce/toyatm
#  - Will select the executables, and some options (see in the following␣
↪sections)
#  - In coupled mode corresponds to the suffix of the OASIS_IN/namcouple.base.
↪$RUNtype to use
export RUNtype=owa
export MOD=`echo $RUNtype | cut -d . -f 1`

#  Name of the experiment you are about to launch (max 30. CHAR)
export CEXPER=BENGUELA_example_${RUNtype}
```

Then, there is a section indicating where the run will be executed and where the outputs and restarts will be stored:

```
#-------------------------------------------------------------------------------
# RUN_DIR
#-------------------------------------------------------------------------------

export EXEDIR_ROOT="$CWORK/rundir/${CEXPER}_execute"
export OUTPUTDIR_ROOT="$CWORK/rundir/${CEXPER}_outputs"
export RESTDIR_ROOT="$CWORK/rundir/${CEXPER}_restarts"

export  JOBDIR_ROOT=${CHOME}/jobs_${CEXPER}
```

Then, there are sections for the different components.

For the coupler settings:

```
#-------------------------------------------------------------------------------
# CPL
#-------------------------------------------------------------------------------

# Namelist
#---------
# Note: namelist example files are provided in OASIS_IN/
# if you want to use a pre-built weight file for grid interpolations, point to
# e.g. namcouple.base.oa.smtho2a
export namcouplename=namcouple.base.${RUNtype}

# Coupling frequency
#-------------------
export CPL_FREQ=21600

# Restart files for OASIS
#------------------------
# If TRUE: create OASIS restart files from pre-existing atm/oce/wav outputs.
# If FALSE: create OASIS restart files from calm conditions (need to read at␣
↪least the grid for each model)
export CPL_restart="FALSE"
export oce_rst_file="${OCE_FILES_DIR}/croco_grd.nc"
export oce_rst_timeind=-1 # time index (-1 is last) in the file to extract as␣
↪restart
export atm_rst_file="${ATM_FILES_DIR}/wrfinput_dXX_2005_01_01_00" # the domain␣
↪dXX will be automatically replaced
export atm_rst_timeind=-1 # time index (-1 is last) in the file to extract as␣
↪restart
```

(continues on next page)

(continued from previous page)

```
export wav_rst_file="${WAV_FILES_DIR}/ww3.200501.nc"
export wav_rst_timeind=-1 # time index (-1 is last) in the file to extract as␣
↪restart
```

You should check the coupling frequency, the restart flag and path towards model files to use to create oasis restart files.

For CROCO settings, first indicate if you request CROCO compilation:

```
#-------------------------------------------------------------------------------
# OCE
#-------------------------------------------------------------------------------

# Where to find or put the croco execuatble
export OCE_EXE_DIR=${CHOME}/CROCO_IN

# Online Compilation
#-------------------
#!!!!!!! IMPORTANT NOTE !!!!!!!
# If activated, creates croco executable depending on this namelist.
#   - In param.h it modifies the grid size, the number of procs in x and y␣
↪direction with those given in myjob.sh
#   - In cppdefs.h it modifies the following options with informations given␣
↪below
#       MPI, OA_COUPLING, OW_COUPLING, MRL_WCI,
#       XIOS, LOGFILE, MPI_NOLAND,
#       AGRIF, AGRIF_2WAY,
#       BULK_FLUX, ONLINE, AROME, ARPEGE, ERA_ECMWF
#       FRC_BRY, CLIMATOLOGY
#       TIDES, PSOURCE, PSOURCE_NCFILE, PSOURCE_NCFILE_TS
# Other changes of parameterizations, numerical schemes, etc should be made "by␣
↪hand" in CROCO_IN/cppdefs.h.base
#!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
export ONLINE_COMP=1
```

Then, edit model time steps:

```
# Time steps
#-----------
export DT_OCE=3600
export NDTFAST=60
```

Then several options for zooms, wave coupling are provided. They are generally automatically set-up depending on the `RUNtype` defined at the beginning.

Then, edit the options regarding the forcing files:

```
# Forcings
#---------
export ini_ext='ini_SODA' # ini extension file (ini_SODA,...)
export bdy_ext='bry_SODA' # bry extension file (clm_SODA,bry_SODA,...)

# flag for surface forcing should be true except in the case of atm coupling
if [[ $MOD =~ .*a.* ]] ; then
    export surfrc_flag="FALSE"
else
    export surfrc_flag="TRUE"
fi
```

(continues on next page)

(continued from previous page)

```
export interponline=0 # switch (1=on, 0=off) for online surface interpolation.␣
↪Only works with MONTHLY input files!
export frc_ext='blk_ERA5' # surface forcing extension(blk_ERA5, frc_ERA5,...).␣
↪If interponline=1 precise the type (ERA_ECMWF or AROME,  [CFSR by default],␣
↪names as cppkey name in croco)

export tide_flag="FALSE" # the forcing extension must be blk_??? otherwise tide␣
↪forcing overwrites it
export river_flag="FALSE"
```

Finally edit CROCO output settings:

```
# Output settings
#----------------
#!!! WARNING: when XIOS is activated the following values (for the model) are␣
↪not taken into account
export oce_his_sec=86400     # history output interval (in number of second)
export oce_avg_sec=86400     # average output interval (in number of second)
```

Then continue with the atmospheric model section:

```
#-----------------------------------------------------------------------------
# ATM
#-----------------------------------------------------------------------------

# Where to find the atm exectuable
if [[ $RUNtype =~ .*a.* && ( $RUNtype =~ .*o.* || $RUNtype =~ .*w.* ) ]] ; then
    export ATM_EXE_DIR=${ATM}/exe_coupled
else
    export ATM_EXE_DIR=${ATM}/exe_uncoupled
fi
```

First the paths for the executable are defined. Those are generally set-up with the `RUNtype` but can be changed if necessary.

Then choose the namelist file, and namelist options: model time step, forcing files informations, Cd formulation (coupling with waves requires isftcflx=5), domains informations.

> **Note**
>
> Other changes in the WRF namelist should be made "by hand" in the `WRF_IN/namelist.input.base.complete`. Only settings into <...> in the namelist are automatically changed by the scripts

```
# Namelist
#---------
#!!!!!!! IMPORTANT NOTE !!!!!!!
# Changes of parameterizations, numerical schemes, etc in atmnamelist
# should be made "by hand" in the WRF_IN/namelist.input.base.complete file
# Only settings into <...> in WRF_IN/namelist.input.base.complete are␣
↪automatically
# filled in by the present mynamelist settings
#!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
export atmnamelist=namelist.input.base.complete

# Time steps
#-----------
```

(continues on next page)

(continued from previous page)

```
export DT_ATM=150

# Boundaries
#-----------
export interval_seconds=21600 # interval ( in sec ) of the lateral input
export auxinput4_interval=360 # interval ( in min ) of bottom input
export nbmetsoil=4
export nbmetlevel=38

# Physics
#--------
 # Cd formulation (default = 0, wave cpl needs = 5)
if [[ $RUNtype =~ .*aw.* || $RUNtype =~ .*owa.* ]] ; then
    export isftcflx=5
else
    export isftcflx=1
fi

# Domains
#--------
export NB_dom=1 # Number of coupled domains
export wrfcpldom='d01' # which WRF domain to couple
export nestfeedback="TRUE" # 1 way (FALSE) or 2 Way (TRUE) nesting
export onlinecplmask="TRUE" # Erase existing CPLMASK and build default mask␣
↪(depending on the nb of atm and oce domains)
```

Then a section when using moving nest is available. It is important to note that to use the moving nest WRF has to be compiled with the moving nest option. In addition, in coupled mode, the moving nest can be used, but only the parent static model can be coupled through OASIS. A dedicated `Registry.EM` is available in `WRF_IN` to compile WRF with moving nest + coupling so that the moving nest receive surface updates from the parent static domain, that is coupled to the ocean or wave model. The following section is used only if the moving nest is activated `ATM_CASE="MOVING_NEST"`. In other case keep `ATM_CASE="DEFAULT"`, and ignore the rest of the section.

```
# Moving nest
#------------
export ATM_CASE="DEFAULT"  # no moving nest: DEFAULT or with: MOVING_NEST
# if ATM_CASE=DEFAULT, the following is not used
export num_mv_nest=1 # number of moving nests
# if several nest, the following variables need to have the format "1st_nest 2nd_
↪nest"
export ref_coef="3" # refinement coef for nest
export ew_size="283" # nest size in east-west dim ([multiple of ref_coef] + 1)
export ns_size="295" # nest size in north-south dim ([multiple of ref_coef] + 1)
export i_prt_strt="580" # where nest is starting in parent's grid x-dim
export j_prt_strt="59" # where nest is starting in parent's grid y-dim
# Tracking parameters
export vortex_interval=5 # When to update vortex position
export max_vortex_speed=40 # Used to compute the search radius for the new␣
↪vortex center position
export corral_dist=8 # The closest distance between child and parend boundary␣
↪(in parent grid cell)
export track_level=50000 # The pressure level (in Pa) where the vortex is tracked
export time_to_move=0 # The time (in minutes) until the nest is moved (at the␣
↪beginning)
```

Then a section when using nudging (assimilation) is available. If not using nudging, ignore this section.

```
# Nudging (assimilation) options
#-------------------------------
export switch_fdda=0 # To activate fdda nudging
export nudgedom="1" # select which kind of nudging you want (1=grid-nudging,␣
↪2=spectral nudging) for each domain. Example for spectral nudging over parent␣
↪only "2 0"
export nudge_coef="0.0003" # nudge coef. Need to be the same size than nudge
export nudge_interval_m="360" # time interval (in min) between analysis times
export nudge_end_h="144" # time (in hours) to stop nudging after start of␣
↪forecast
```

Finally, set the ouptut settings:

```
# 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
# file for specifying different than default output variables: OPTIONAL, leave␣
↪empty if not used
export atm_iofields='myoutfields.txt'
```

Then continue with the wave model section:

```
#-------------------------------------------------------------------------------
# WAV
#-------------------------------------------------------------------------------

# Where to find the wav executable
if [[ $RUNtype =~ .*owa.* ]] ; then
    export WAV_EXE_DIR=${WAV}/exe_owa
elif [[ $RUNtype =~ .*ow.* ]] ; then
    export WAV_EXE_DIR=${WAV}/exe_ow
elif [[ $RUNtype =~ .*aw.* ]] ; then
    export WAV_EXE_DIR=${WAV}/exe_aw
else
    export WAV_EXE_DIR=${WAV}/exe_frc
fi

# Namelist
#---------
# Chosing the ww3_shel.inp.base.SHELL_EXT (see options in WW3_IN)
if [[ $RUNtype =~ .*toy.* ]] ; then
    export SHELL_EXT=$MOD
else
    export SHELL_EXT=$RUNtype
fi
```

First the paths for the executable are defined, and the ww3_shel namelist to use is defined. Those are generally set-up with the `RUNtupe` but can be changed if necessary.

Then edit the ww3 time steps and grid settings (these will be updated in ww3_grid.inp):

```
# Time steps
#-----------
```

(continues on next page)

(continued from previous page)

```
export DT_WAV=3600      # TMAX = 3*TCFL
export DT_WW_PRO=1200   # TCFL = 0.8 x dx/(g/fmin4pi) with fmin=0.0373 => 3-4 %␣
↪of dx
export DT_WW_REF=1800   # TMAX / 2
export DT_WW_SRC=10     # TSRC = usually 10s  (could be between 5s and 60s)

# Grid size
#----------
export wavnx=41 ; export wavny=42

# Parameter
#----------
export hmin=75; # e.g. minimum water depth in CROCO (will be used to delimit␣
↪coastline in WW3)
```

Then, choose the forcing to use for ww3:

```
# Forcing files
#--------------
# forcin: forcing file(s) PREFIX list (input file are supposed to be in the␣
↪form: PREFIX_Y????M??.nc)
# forcww3: name of ww3_prnc.inp extension, e.g current or wind/era5, see in WW3_
↪IN directory
if [[ $RUNtype =~ .*owa.* ]]; then
    export forcin=()
    export forcww3=()
elif [[ $RUNtype =~ .*Afrc.* || $RUNtype =~ .*ow.* ]] ; then
    export forcin=(ERA5_wind)
    export forcww3=(wind.era5)
elif [[ $RUNtype =~ .*Ofrc.* ]] ; then
    export forcin=(CROCO_current CROCO_level)
    export forcww3=(current level)
elif [[ $RUNtype =~ .*frc.* ]] ; then
    export forcin=(ERA5_wind CROCO_current CROCO_level)
    export forcww3=(wind.era5 current level)
fi
```

As well as the boundary data (can be left empty):

```
# Boundary files
#---------------
# prefix for boundary files (leave empty is none), there are supposed to be in␣
↪WAV_FILES_DIR
export bouncin=
```

Finally, set output settings:

```
# Output settings
#----------------
export wav_int=21600  # output interval (s)
export wav_pnt=0  # point output interval. Put 0 if no point output
export point_output_list=${WAV_FILES_DIR}/my_point_output_test.txt # file where␣
↪to find list of point (format: lon lat name) to output spectrum
export wav_trck=0     # track output interval. Put 0 if no track output
export flagout="TRUE" # Keep (TRUE) or not (FALSE) the ww3 output binary files␣
↪(e.g. out_grd.ww3)
```

- Now that you have completed the necessary files, you are ready to run your simulation. To do so, simply do

```
./submitjob.sh
```

2.19.4.6 Outputs, logs

Once your job is launched, two repositories should appear:

- `${CHOME}/job_${CEXPER}`
- `${CWORK}/rundir`

`${CHOME}`, `${CWORK}` being variables you specified in `myenv_mypath.sh`. The first is where jobs and logs are put. The second is where your job is running and where outputs/restarts are stored:

- `${CEXPER}_execute`: running directory, here you can find all the files used to run your simulation
- `${CEXPER}_output`: output directory, here you should find the output of the different models if the run was successful
- `${CEXPER}_restart`: restart directory, here you should find the restarts files, they are used for the next job if `RESTART_FLAG` is set to `True`.

In those repositories, you will find one folder per job. Meaning if the simulation is 12 jobs long, you will have 12 folders named with dates of each job.

In case of error during your job, check the log files in: `${CHOME}/job_${CEXPER}` for the job log, or in `${CWORK}/rundir/${CEXPER}_execute` for the model and oasis logs:

- `out_run.txt`: general log
- `rsl.error.0000 rsl.out.0000 rsl.error.000* rsl.out.000*`: WRF logs
- `croco.log`: CROCO log
- `grid.out ounf.out prnc.wind.out strt.out log.ww3`: WW3 logs
- `nout.000000 debug.root.01 debug.root.02`: OASIS logs

Typical issues that you can encounter are:

- Files not found: check your file names, paths, check `myenv_mypath.sh`, `mynamelist.sh`
- Inconsistency in exchanged variables: check OASIS `namcouple`, especially variable names
- Inconsistent dimensions of the grids of the different files: check the model grid files, the OASIS grids and masks files, the OASIS remapping weight files, as well as the models and OASIS namelists. NB: OASIS grids, masks and rmp* files are not recomputed if they already exist in your directory (make sure to clean in case of error)
- Model blow up: check the model log files. If blow up is due to CFL issue (unrealistic speed, or segmentation fault), decrease the model time step in `mynamelist.sh` but be careful to keep consistency between the coupling time step and model time steps (they should be multiples)

2.20 Littoral 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 analyzed. The various aspects that will be adressed are the following :

- Compute a test-case,
- Select wave-averaged or wave-resolving options

- Modify a test-case (including a new bathymetry, modifying the forcings, ...),
- Use of the CROCO embedded WKB wave model,
- Use Bottom Boundary Layer formulation under combined waves and currents,
- Account for the sediment compartment,
- Analyze bed evolution (morphodynamics).

The tutorial is based on the *Rip Current*, *Sandbar*, *Plannar Beach* test cases. For a description of the wave-averaged equations and WKB wave model see *Wave-averaged Equations*.

For wave-resolving simulations, we refer to *Swash* and *Sandbar* (2DV) and *Flash Rips* (3D) test cases, although, with the right choice of options, *Rip Current* can also produce nearshore currents due to individual short-crested waves. In this case, *Non-Hydrostatic, Non-Boussinesq Equations* are needed, as well as an offshore *Wavemaker*.

Background:

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. There are two main types of rip currents: channeled (bathymetric) and transient (flash) rip currents. Channeled rip currents are the most commonly studied ([Marchesiello *et al.*, 2015]). Flash rips are purely wave-induced processes that can occur on uniform bathymetry along the coast. Over the last decade, the process of generating flash rips by short-crested waves (Peregrine process) has become the current consensus in the nearshore community ([Marchesiello *et al.*, 2021]).

RIP test case:

Here, we consider channeled rip currents, i.e., longshore variation of the wave-induced momentum flux due to breaking at barred bottom topography with an imposed longshore perturbation, as in Yu [2003] or Weir *et al.* [2011]. 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.*, 2010]. 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 from Yu [2003] shows what the bathymetry looks like.

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:

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
```

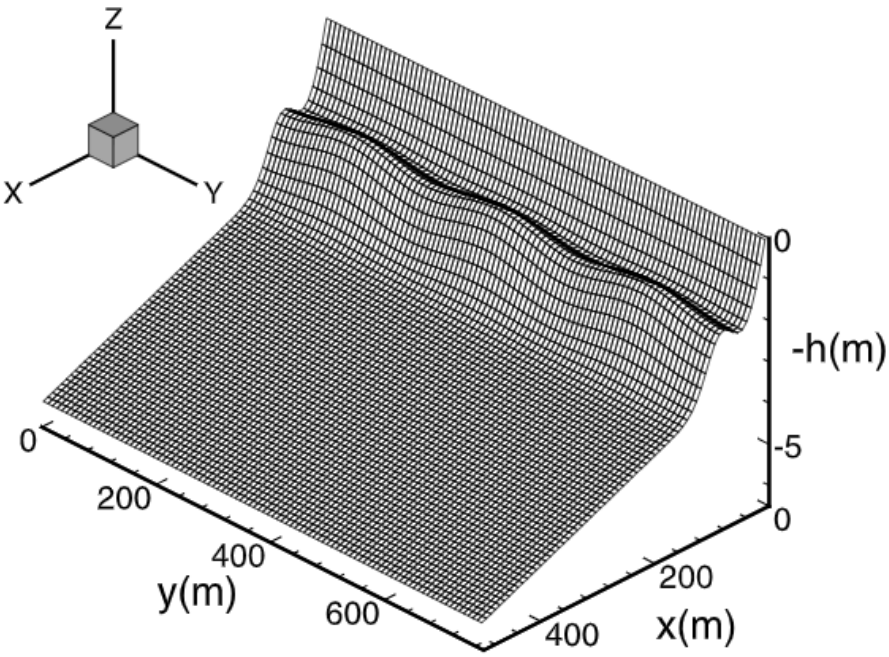


Figure 1. Beach topography with a longshore bar at $x = 80$ m and rip channels. $\epsilon = 0.1$ and $\lambda = 256$ m.

```
# 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
#  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
```

(continues on next page)

(continued from previous page)

```
#  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
# ifdef SEDIMENT
#  define ANA_SEDIMENT
#  undef  ANA_SPFLUX
#  undef  ANA_BPFLUX
# endif
```

(continues on next page)

(continued from previous page)

```
# undef  DIAGNOSTICS_UV
```

2.21 Realistic coastal configuration

Warning

This part is only given as an example of coastal configuration in tidal environment. So you have an example of which set of cpp keys to use

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`)

All inputs files can be found there :

https://data-croco.ifremer.fr/CONFIGS_EXAMPLES/Run_VILAINE.tar.gz

2.22 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
   #
   ```

(continues on next page)

(continued from previous page)

```
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:

```
./jobcomp
```

Run :

```
qsub job_croco_mpi.pbs
```

Explore files, edit and modify iodef.xml, and run again ...

2.23 Tips

2.23.1 Tips in case of errors during compilation

1. *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
2. *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

3. *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
4. *relocation truncated to fit: R_X86_64_32S against symbol* at compilation in sequential mode
   - add option mcmodel=large in compile options (FLAGS) in jobcomp
5. *m2c* error at the beginning of the compilation
   - the path to OCEAN directory in the jobcomp file may be wrong

2.23.2 TIPS for errors at runtime

2.23.2.1 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

2.23.2.2 Others possible errors

- If at runtime you got an error in reading bathymetry variable *H* (code -57)

  ::
  you may ask for too many cpus in MPI compared to the size of your grid

- **XIOS**

> **Warning**
>
> The output time step in XIOS must be a multiple of the 3D time step in croco

2.23.3 Analytical forcing

Some advices and tips on how to use the model with analytical forcings

1. How to unplug atmospherical forcing

   In cppdefs.h you should undefine *BULK_FLUX* and add some keys

```
#undef BULK_FLUX
#define ANA_SSFLUX
#define ANA_STFLUX
#define ANA_SMFLUX
#define ANA_SRFLUX
```

2. Set analytical boundary conditions
   - Edit cppdefs.h

     ```
     DEFINE ANA_BRY
     ```

   - Edit *analytical.F* routine and set your own OBC for zeta,Ubar,Vbar,U,V,T,S

2.23.4 MUSTANG tips

2.23.4.1 Get to know the CROCO/MUSTANG coupling

Read the documentation on CROCO/MUSTANG (*MUSTANG Sediment model*)

> **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

2.23.4.2 Run a test case

- Choose a test case (*Sediment test cases*)
- 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.

2.23.4.3 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
     ```

     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 needed 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
     !-----------------------------------------------------------
     ```

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" ;
    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) ;
```

(continues on next page)

(continued from previous page)

```
    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) ;
```

(continues on next page)

(continued from previous page)

```
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.

BIBLIOGRAPHY

[1] Alexander F. Shchepetkin and James C. McWilliams. The regional oceanic modeling system (ROMS): a split-explicit, free-surface, topography-following-coordinate oceanic model. *Ocean Modelling*, 9(4):347–404, January 2005. URL: https://linkinghub.elsevier.com/retrieve/pii/S1463500304000484 (visited on 2023-06-06), doi:10.1016/j.ocemod.2004.08.002.

[2] Laurent Debreu, Patrick Marchesiello, Pierrick Penven, and Gildas Cambon. Two-way nesting in split-explicit ocean models: Algorithms, implementation and validation. *Ocean Modelling*, 49-50:1–21, June 2012. URL: https://linkinghub.elsevier.com/retrieve/pii/S1463500312000480 (visited on 2023-06-06), doi:10.1016/j.ocemod.2012.03.003.

[3] F. Auclair, L. Bordois, Y. Dossmann, T. Duhaut, A. Paci, C. Ulses, and C. Nguyen. A non-hydrostatic non-Boussinesq algorithm for free-surface ocean modelling. *Ocean Modelling*, 132:12–29, December 2018. URL: https://linkinghub.elsevier.com/retrieve/pii/S1463500318302646 (visited on 2023-06-06), doi:10.1016/j.ocemod.2018.07.011.

[4] John Marshall, Chris Hill, Lev Perelman, and Alistair Adcroft. Hydrostatic, quasi-hydrostatic, and nonhydrostatic ocean modeling. *Journal of Geophysical Research: Oceans*, 102(C3):5733–5752, March 1997. URL: http://doi.wiley.com/10.1029/96JC02776 (visited on 2023-06-06), doi:10.1029/96JC02776.

[5] James C. McWilliams, Juan M. Restrepo, and Emily M. Lane. An asymptotic theory for the interaction of waves and currents in coastal waters. *Journal of Fluid Mechanics*, 511:135–178, 2004. URL: https://www.cambridge.org/core/article/an-asymptotic-theory-for-the-interaction-of-waves-and-currents-in-coastal-waters/DA4918B37321E8DF3D1DFADD776BA8F6, doi:10.1017/S0022112004009358.

[6] T. Gerkema, J. T. F. Zimmerman, L. R. M. Maas, and H. Van Haren. Geophysical and astrophysical fluid dynamics beyond the traditional approximation. *Reviews of Geophysics*, 46(2):RG2004, May 2008. URL: http://doi.wiley.com/10.1029/2006RG000220 (visited on 2023-06-06), doi:10.1029/2006RG000220.

[7] Patrick Marchesiello, Rachid Benshila, Rafael Almar, Yusuke Uchiyama, James C. McWilliams, and Alexander Shchepetkin. On tridimensional rip current modeling. *Ocean Modelling*, 96:36–48, December 2015. URL: https://linkinghub.elsevier.com/retrieve/pii/S1463500315001122 (visited on 2023-06-06), doi:10.1016/j.ocemod.2015.07.003.

[8] Yusuke Uchiyama, James C. McWilliams, and Alexander F. Shchepetkin. Wave–current interaction in an oceanic circulation model with a vortex-force formalism: Application to the surf zone. *Ocean Modelling*, 34(1-2):16–35, January 2010. URL: https://linkinghub.elsevier.com/retrieve/pii/S1463500310000594 (visited on 2023-06-06), doi:10.1016/j.ocemod.2010.04.002.

[9] Michael S. Longuet-Higgins. Mass transport in water waves. *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 245(903):535–581, March 1953. URL: https://royalsocietypublishing.org/doi/10.1098/rsta.1953.0006 (visited on 2023-06-06), doi:10.1098/rsta.1953.0006.

[10] John Casey Church and Edward B. Thornton. Effects of breaking wave induced turbulence within a longshore current model. *Coastal Engineering*, 20(1-2):1–28, July 1993. URL: https://linkinghub.elsevier.com/retrieve/pii/037838399390053B (visited on 2023-06-06), doi:10.1016/0378-3839(93)90053-B.

[11] Edward B. Thornton and R. T. Guza. Transformation of wave height distribution. *Journal of Geophysical Research*, 88(C10):5925, 1983. URL: http://doi.wiley.com/10.1029/JC088iC10p05925 (visited on 2023-06-06), doi:10.1029/JC088iC10p05925.

[12] Edward B. Thornton and R. T. Guza. Surf Zone Longshore Currents and Random Waves: Field Data and Models. *Journal of Physical Oceanography*, 16(7):1165–1178, July 1986. URL: http://journals.ametsoc.org/doi/10.1175/1520-0485(1986)016\T1\textless{}1165:SZLCAR\T1\textgreater{}2.0.CO;2 (visited on 2023-06-06), doi:10.1175/1520-0485(1986)016<1165:SZLCAR>2.0.CO;2.

[13] Aike Beckmann and Dale B. Haidvogel. Numerical Simulation of Flow around a Tall Isolated Seamount. Part I: Problem Formulation and Model Accuracy. *Journal of Physical Oceanography*, 23(8):1736–1753, August 1993. URL: http://journals.ametsoc.org/doi/10.1175/1520-0485(1993)023\T1\textless{}1736:NSOFAA\T1\textgreater{}2.0.CO;2 (visited on 2023-06-06), doi:10.1175/1520-0485(1993)023<1736:NSOFAA>2.0.CO;2.

[14] Robert L. Haney. On the Pressure Gradient Force over Steep Topography in Sigma Coordinate Ocean Models. *Journal of Physical Oceanography*, 21(4):610–619, April 1991. URL: http://journals.ametsoc.org/doi/10.1175/1520-0485(1991)021\T1\textless{}0610:OTPGFO\T1\textgreater{}2.0.CO;2 (visited on 2023-06-06), doi:10.1175/1520-0485(1991)021<0610:OTPGFO>2.0.CO;2.

[15] Yves Soufflet, Patrick Marchesiello, Florian Lemarié, Julien Jouanno, Xavier Capet, Laurent Debreu, and Rachid Benshila. On effective resolution in ocean models. *Ocean Modelling*, 98:36–50, February 2016. URL: https://linkinghub.elsevier.com/retrieve/pii/S1463500315002401 (visited on 2023-06-06), doi:10.1016/j.ocemod.2015.12.004.

[16] Alexander F. Shchepetkin and James C. McWilliams. Quasi-Monotone Advection Schemes Based on Explicit Locally Adaptive Dissipation. *Monthly Weather Review*, 126(6):1541–1580, June 1998. URL: http://journals.ametsoc.org/doi/10.1175/1520-0493(1998)126\T1\textless{}1541:QMASBO\T1\textgreater{}2.0.CO;2 (visited on 2023-06-06), doi:10.1175/1520-0493(1998)126<1541:QMASBO>2.0.CO;2.

[17] C. Ménesguen, S. Le Gentil, P. Marchesiello, and N. Ducousso. Destabilization of an Oceanic Meddy-Like Vortex: Energy Transfers and Significance of Numerical Settings. *Journal of Physical Oceanography*, 48(5):1151–1168, May 2018. URL: https://journals.ametsoc.org/view/journals/phoc/48/5/jpo-d-17-0126.1.xml (visited on 2023-06-06), doi:10.1175/jpo-d-17-0126.1.

[18] Rafael Borges, Monique Carmona, Bruno Costa, and Wai Sun Don. An improved weighted essentially non-oscillatory scheme for hyperbolic conservation laws. *Journal of Computational Physics*, 227(6):3191–3211, March 2008. URL: https://linkinghub.elsevier.com/retrieve/pii/S0021999107005232 (visited on 2023-06-06), doi:10.1016/j.jcp.2007.11.038.

[19] Alexander F. Shchepetkin. An adaptive, Courant-number-dependent implicit scheme for vertical advection in oceanic modeling. *Ocean Modelling*, 91:38–69, July 2015. URL: https://linkinghub.elsevier.com/retrieve/pii/S1463500315000530 (visited on 2023-06-07), doi:10.1016/j.ocemod.2015.03.006.

[20] P. Marchesiello and P. Estrade. Eddy activity and mixing in upwelling systems: a comparative study of Northwest Africa and California regions. *International Journal of Earth Sciences*, 98(2):299–308, March 2009. URL: http://link.springer.com/10.1007/s00531-007-0235-6 (visited on 2023-06-06), doi:10.1007/s00531-007-0235-6.

[21] F. Lemarié, L. Debreu, A.F. Shchepetkin, and J.C. McWilliams. On the stability and accuracy of the harmonic and biharmonic isoneutral mixing operators in ocean models. *Ocean Modelling*, 52-53:9–35, August 2012. URL: https://linkinghub.elsevier.com/retrieve/pii/S1463500312000674 (visited on 2023-06-06), doi:10.1016/j.ocemod.2012.04.007.

[22] Alexander F. Shchepetkin and James C. McWilliams. A method for computing horizontal pressure-gradient force in an oceanic model with a nonaligned vertical coordinate. *Journal of Geophysical Research*, 108(C3):3090, 2003. URL: http://doi.wiley.com/10.1029/2001JC001047 (visited on 2023-06-07), doi:10.1029/2001JC001047.

[23] Patrick Marchesiello, James C. McWilliams, and Alexander Shchepetkin. Open boundary conditions for long-term integration of regional oceanic models. *Ocean Modelling*, 3(1-2):1–20, January 2001. URL: https://linkinghub.elsevier.com/retrieve/pii/S1463500300000135 (visited on 2023-06-06), doi:10.1016/S1463-5003(00)00013-5.

[24] David R. Jackett and Trevor J. Mcdougall. Minimal Adjustment of Hydrographic Profiles to Achieve Static Stability. *Journal of Atmospheric and Oceanic Technology*, 12(2):381–389, April 1995. URL: http://journals.ametsoc.org/doi/10.1175/1520-0426(1995)012\T1\textless{}0381:MAOHPT\T1\textgreater{}2.0.CO;2 (visited on 2023-06-07), doi:10.1175/1520-0426(1995)012<0381:MAOHPT>2.0.CO;2.

[25] John C. Warner, Zafer Defne, Kevin Haas, and Hernan G. Arango. A wetting and drying scheme for ROMS. *Computers & Geosciences*, 58:54–61, August 2013. URL: https://linkinghub.elsevier.com/retrieve/pii/S0098300413001362 (visited on 2023-06-07), doi:10.1016/j.cageo.2013.05.004.

[26] William G. Large. Modeling and Parameterizing the Ocean Planetary Boundary Layer. In Eric P. Chassignet and Jacques Verron, editors, *Ocean Modeling and Parameterization*, pages 81–120. Springer Netherlands, Dordrecht, 1998. URL: http://link.springer.com/10.1007/978-94-011-5096-5_3 (visited on 2023-06-07), doi:10.1007/978-94-011-5096-5_3.

[27] W. G. Large, J. C. McWilliams, and S. C. Doney. Oceanic vertical mixing: A review and a model with a nonlocal boundary layer parameterization. *Reviews of Geophysics*, 32(4):363, 1994. URL: http://doi.wiley.com/10.1029/94RG01872 (visited on 2023-06-07), doi:10.1029/94RG01872.

[28] James C. McWilliams and Peter P. Sullivan. Vertical Mixing by Langmuir Circulations. *Spill Science & Technology Bulletin*, 6(3-4):225–237, June 2000. URL: https://linkinghub.elsevier.com/retrieve/pii/S135325610100041X (visited on 2023-06-07), doi:10.1016/S1353-2561(01)00041-X.

[29] L. P. Van Roekel, B. Fox-Kemper, P. P. Sullivan, P. E. Hamlington, and S. R. Haney. The form and orientation of Langmuir cells for misaligned winds and waves: LANGMUIR UNDER MISALIGNED WIND AND WAVES. *Journal of Geophysical Research: Oceans*, 117(C5):n/a–n/a, May 2012. URL: http://doi.wiley.com/10.1029/2011JC007516 (visited on 2023-06-07), doi:10.1029/2011JC007516.

[30] Qing Li, Adrean Webb, Baylor Fox-Kemper, Anthony Craig, Gokhan Danabasoglu, William G. Large, and Mariana Vertenstein. Langmuir mixing effects on global climate: WAVEWATCH III in CESM. *Ocean Modelling*, 103:145–160, July 2016. URL: https://linkinghub.elsevier.com/retrieve/pii/S1463500315001407 (visited on 2023-06-07), doi:10.1016/j.ocemod.2015.07.020.

[31] Andrej Nikolaevich Kolmogorov. Equations of turbulent motion in an incompressible fluid. *Dokl. Akad. Nauk SSSR*, 30(4):299–303, 1942.

[32] W.P Jones and B.E Launder. The prediction of laminarization with a two-equation model of turbulence. *International Journal of Heat and Mass Transfer*, 15(2):301–314, February 1972. URL: https://linkinghub.elsevier.com/retrieve/pii/0017931072900762 (visited on 2023-06-07), doi:10.1016/0017-9310(72)90076-2.

[33] Lars Umlauf and Hans Burchard. A generic length-scale equation for geophysical turbulence models. *Journal of Marine Research*, 61:235–265, 2003.

[34] M. M. Gibson and B. E. Launder. Ground effects on pressure fluctuations in the atmospheric boundary layer. *Journal of Fluid Mechanics*, 86(3):491–511, June 1978. URL: https://www.cambridge.org/core/product/identifier/S0022112078001251/type/journal_article (visited on 2023-06-08), doi:10.1017/S0022112078001251.

[35] George L. Mellor and Tetsuji Yamada. Development of a turbulence closure model for geophysical fluid problems. *Reviews of Geophysics*, 20(4):851, 1982. URL: http://doi.wiley.com/10.1029/RG020i004p00851 (visited on 2023-06-08), doi:10.1029/RG020i004p00851.

[36] Lakshmi H. Kantha and Carol Anne Clayson. An improved mixed layer model for geophysical applications. *Journal of Geophysical Research*, 99(C12):25235, 1994. URL: http://doi.wiley.com/10.1029/94JC02257 (visited on 2023-06-08), doi:10.1029/94JC02257.

[37] Patrick J. Luyten. An analytical and numerical study of surface and bottom boundary layers with variable forcing and application to the North Sea. *Journal of Marine Systems*, 8(3-4):171–189, September 1996. URL: https://linkinghub.elsevier.com/retrieve/pii/092479639600005X (visited on 2023-06-08), doi:10.1016/0924-7963(96)00005-X.

[38] Y. Cheng, V. M. Canuto, and A. M. Howard. An Improved Model for the Turbulent PBL. *Journal of the Atmospheric Sciences*, 59(9):1550–1565, May 2002. URL: http://journals.ametsoc.org/doi/10.1175/1520-0469(2002)059\T1\textless{}1550:AIMFTT\T1\textgreater{}2.0.CO;2 (visited on 2023-06-08), doi:10.1175/1520-0469(2002)059<1550:AIMFTT>2.0.CO;2.

[39] B. Galperin, L. H. Kantha, S. Hassid, and A. Rosati. A Quasi-equilibrium Turbulent Energy Model for Geophysical Flows. *Journal of the Atmospheric Sciences*, 45(1):55–62, January 1988. URL: http://journals.ametsoc.org/doi/10.1175/1520-0469(1988)045\T1\textless{}0055:AQETEM\T1\textgreater{}2.0.CO;2 (visited on 2023-06-08), doi:10.1175/1520-0469(1988)045<0055:AQETEM>2.0.CO;2.

[40] Florian Lemarié, Guillaume Samson, Jean-Luc Redelsperger, Hervé Giordani, Théo Brivoal, and Gurvan Madec. A simplified atmospheric boundary layer model for an improved representation of air–sea interactions in eddying oceanic models: implementation and first evaluation in NEMO (4.0). *Geoscientific Model Development*, 14(1):543–572, January 2021. URL: https://gmd.copernicus.org/articles/14/543/2021/ (visited on 2021-04-08), doi:10.5194/gmd-14-543-2021.

[41] Lionel Renault, S. Masson, T. Arsouze, Gurvan Madec, and James C. McWilliams. Recipes for How to Force Oceanic Model Dynamics. *Journal of Advances in Modeling Earth Systems*, February 2020. URL: https://onlinelibrary.wiley.com/doi/10.1029/2019MS001715 (visited on 2023-06-08), doi:10.1029/2019MS001715.

[42] Xubin Zeng and Anton Beljaars. A prognostic scheme of sea surface skin temperature for modeling and data assimilation: SEA SURFACE SKIN TEMPERATURE SCHEME. *Geophysical Research Letters*, 32(14):n/a–n/a, July 2005. URL: http://doi.wiley.com/10.1029/2005GL023030 (visited on 2023-06-08), doi:10.1029/2005GL023030.

[43] E. Blayo and L. Debreu. Revisiting open boundary conditions from the point of view of characteristic variables. *Ocean Modelling*, 9(3):231–252, January 2005. URL: https://linkinghub.elsevier.com/retrieve/pii/S1463500304000447 (visited on 2023-06-08), doi:10.1016/j.ocemod.2004.07.001.

[44] Simon Treillou, Patrick Marchesiello, and Christine M. Baker. Correction of coherent interference in wave-resolving nearshore models and validation with experimental data. *Ocean Modelling*, 189:102369, June 2024. doi:10.1016/j.ocemod.2024.102369.

[45] Pierrick Penven, Laurent Debreu, Patrick Marchesiello, and James C. McWilliams. Evaluation and application of the ROMS 1-way embedding procedure to the central california upwelling system. *Ocean Modelling*, 12(1-2):157–187, January 2006. URL: https://linkinghub.elsevier.com/retrieve/pii/S1463500305000491 (visited on 2023-06-08), doi:10.1016/j.ocemod.2005.05.002.

[46] Eric Blayo and Laurent Debreu. Adaptive Mesh Refinement for Finite-Difference Ocean Models: First Experiments. *Journal of Physical Oceanography*, 29(6):1239–1250, June 1999. URL: http://journals.ametsoc.org/doi/10.1175/1520-0485(1999)029\T1\textless{}1239:AMRFFD\T1\textgreater{}2.0.CO;2 (visited on 2023-06-08), doi:10.1175/1520-0485(1999)029<1239:AMRFFD>2.0.CO;2.

[47] Laurent Debreu, Christophe Vouland, and Eric Blayo. AGRIF: Adaptive grid refinement in Fortran. *Computers & Geosciences*, 34(1):8–13, January 2008. URL: https://linkinghub.elsevier.com/retrieve/pii/S009830040700115X (visited on 2023-06-08), doi:10.1016/j.cageo.2007.01.009.

[48] Meinte Blaas, Changming Dong, Patrick Marchesiello, James C. McWilliams, and Keith D. Stolzenbach. Sediment-transport modeling on Southern Californian shelves: A ROMS case study. *Continental Shelf Research*, 27(6):832–853, March 2007. URL: https://linkinghub.elsevier.com/retrieve/pii/S027843430600389X (visited on 2023-06-09), doi:10.1016/j.csr.2006.12.003.

[49] RL Soulsby. Bed shear-stresses due to combined waves and currents. *Advances in coastal morphodynamics*, 1995.

[50] William D. Grant and Ole Secher Madsen. Movable bed roughness in unsteady oscillatory flow. *Journal of Geophysical Research*, 87(C1):469, 1982. URL: http://doi.wiley.com/10.1029/JC087iC01p00469 (visited on 2023-06-09), doi:10.1029/JC087iC01p00469.

[51] Peter Nielsen. Suspended sediment concentrations under waves. *Coastal Engineering*, 10(1):23–31, May 1986. URL: https://linkinghub.elsevier.com/retrieve/pii/0378383986900372 (visited on 2023-06-09), doi:10.1016/0378-3839(86)90037-2.

[52] Michael Z Li and Carl L Amos. SEDTRANS96: the upgraded and better calibrated sediment-transport model for continental shelves. *Computers & Geosciences*, 27(6):619–645, July 2001. URL: https://linkinghub.elsevier.com/retrieve/pii/S0098300400001205 (visited on 2023-06-09), doi:10.1016/S0098-3004(00)00120-5.

[53] John C. Warner, Christopher R. Sherwood, Richard P. Signell, Courtney K. Harris, and Hernan G. Arango. Development of a three-dimensional, regional, coupled wave, current, and sediment-transport model. *Computers & Geosciences*, 34(10):1284–1306, October 2008. URL: https://linkinghub.elsevier.com/retrieve/pii/S0098300408000563 (visited on 2023-06-09), doi:10.1016/j.cageo.2008.02.012.

[54] Farshad Shafiei. Nutrient mass balance of a large riverine reservoir in the context of water residence time variability. *Environmental Science and Pollution Research*, 28(29):39082–39100, August 2021. URL: https://link.springer.com/10.1007/s11356-021-13297-8 (visited on 2023-06-09), doi:10.1007/s11356-021-13297-8.

[55] Courtney K. Harris and Patricia L. Wiberg. Approaches to quantifying long-term continental shelf sediment transport with an example from the Northern California STRESS mid-shelf site. *Continental Shelf Research*, 17(11):1389–1418, September 1997. URL: https://linkinghub.elsevier.com/retrieve/pii/S0278434397000174 (visited on 2023-06-09), doi:10.1016/S0278-4343(97)00017-4.

[56] Dale R. Durran. *Numerical Methods for Fluid Dynamics*. Volume 32 of Texts in Applied Mathematics. Springer New York, New York, NY, 2010. ISBN 9781441964113 9781441964120. URL: http://link.springer.com/10.1007/978-1-4419-6412-0 (visited on 2023-06-09), doi:10.1007/978-1-4419-6412-0.

[57] E. Meyer-Peter and R. Müller. Formulas for bed-load transport. pages 39–64, 1948.

[58] Tarandeep S. Kalra, Christopher R. Sherwood, John C. Warner, Yashar Rafati, and Tian-Jian Hsu. INVESTIGATING BEDLOAD TRANSPORT UNDER ASYMMETRICAL WAVES USING A COUPLED OCEAN-WAVE MODEL. In *Coastal Sediments 2019*, 591–604. Tampa/St. Petersburg, Florida, USA, May 2019. WORLD SCIENTIFIC. URL: https://www.worldscientific.com/doi/abs/10.1142/9789811204487_0052 (visited on 2023-06-09), doi:10.1142/9789811204487_0052.

[59] G.R. Lesser, J.A. Roelvink, J.A.T.M. Van Kester, and G.S. Stelling. Development and validation of a three-dimensional morphological model. *Coastal Engineering*, 51(8-9):883–915, October 2004. URL: https://linkinghub.elsevier.com/retrieve/pii/S0378383904000870 (visited on 2023-06-09), doi:10.1016/j.coastaleng.2004.07.014.

[60] Mohammad Dibajnia and Akira Watanabe. Sheet Flow Under Nonlinear Waves and Currents. In *Coastal Engineering 1992*, 2015–2028. Venice, Italy, June 1993. American Society of Civil Engineers. URL: http://ascelibrary.org/doi/10.1061/9780872629332.154 (visited on 2023-06-09), doi:10.1061/9780872629332.154.

[61] Jan S. Ribberink. Bed-load transport for steady flows and unsteady oscillatory flows. *Coastal Engineering*, 34(1-2):59–82, July 1998. URL: https://linkinghub.elsevier.com/retrieve/pii/S0378383998000131 (visited on 2023-06-20), doi:10.1016/S0378-3839(98)00013-1.

[62] Leo C. van Rijn. Principles of sediment transport in rivers, estuaries and coastal seas. In 1993.

[63] J.A. Roelvink. Coastal morphodynamic evolution techniques. *Coastal Engineering*, 53(2-3):277–287, February 2006. URL: https://linkinghub.elsevier.com/retrieve/pii/S0378383905001419 (visited on 2023-06-09), doi:10.1016/j.coastaleng.2005.10.015.

[64] R.L. Soulsby. Dynamics of marine sands: a manual for practical applications. *Oceanographic Literature Review*, 44(9):947, 1997.

[65] J. Dungan Smith and S. R. McLean. Spatially averaged flow over a wavy surface. *Journal of Geophysical Research*, 82(12):1735–1746, April 1977. URL: http://doi.wiley.com/10.1029/JC082i012p01735 (visited on 2023-06-20), doi:10.1029/JC082i012p01735.

[66] Romaric Verney, Robert Lafite, Jean Claude Brun-Cottan, and Pierre Le Hir. Behaviour of a floc population during a tidal cycle: Laboratory experiments and numerical modelling. *Continental Shelf Research*, 31(10):S64–S83, July 2011. URL: https://linkinghub.elsevier.com/retrieve/pii/S0278434310000415 (visited on 2023-08-04), doi:10.1016/j.csr.2010.02.005.

[67] Pierre Le Hir, Florence Cayocca, and Benoît Waeles. Dynamics of sand and mud mixtures: A multiprocess-based modelling strategy. *Continental Shelf Research*, 31(10):S135–S149, July 2011. URL: https://linkinghub.elsevier.com/retrieve/pii/S0278434310003833 (visited on 2023-06-09), doi:10.1016/j.csr.2010.12.009.

[68] Baptiste Mengual, Pierre Le Hir, Aurélie Rivier, Matthieu Caillaud, and Florent Grasso. Numerical modeling of bedload and suspended load contributions to morphological evolution of the Seine Estuary (France). *International Journal of Sediment Research*, 36(6):723–735, December 2021. URL: https://linkinghub.elsevier.com/retrieve/pii/S1001627920300755 (visited on 2023-06-26), doi:10.1016/j.ijsrc.2020.07.003.

[69] Aurélie Rivier, Pierre Le Hir, Pascal Bailly Du Bois, Philippe Laguionie, and Mehdi Morillon. Numerical modelling of heterogeneous sediment transport: new insights for particulate radionuclide transport and deposition. 2017. URL: https://archimer.ifremer.fr/doc/00394/50580/.

[70] Leo C. Van Rijn. Sediment Pick-Up Functions. *Journal of Hydraulic Engineering*, 110(10):1494–1502, October 1984. URL: https://ascelibrary.org/doi/10.1061/%28ASCE%290733-9429%281984%29110%3A10%281494%29 (visited on 2023-06-26), doi:10.1061/(ASCE)0733-9429(1984)110:10(1494).

[71] Weiming Wu and Qianru Lin. Nonuniform sediment transport under non-breaking waves and currents. *Coastal Engineering*, 90:1–11, August 2014. URL: https://linkinghub.elsevier.com/retrieve/pii/S0378383914000763 (visited on 2023-06-26), doi:10.1016/j.coastaleng.2014.04.006.

[72] Baptiste Mengual, Pierre Le Hir, Florence Cayocca, and Thierry Garlan. Modelling Fine Sediment Dynamics: Towards a Common Erosion Law for Fine Sand, Mud and Mixtures. *Water*, 9(8):564, July 2017. URL: http://www.mdpi.com/2073-4441/9/8/564 (visited on 2023-06-09), doi:10.3390/w9080564.

[73] Weiming Wu and Wei Li. Porosity of bimodal sediment mixture with particle filling. *International Journal of Sediment Research*, 32(2):253–259, June 2017. URL: https://linkinghub.elsevier.com/retrieve/pii/S1001627917300793 (visited on 2023-06-26), doi:10.1016/j.ijsrc.2017.03.005.

[74] John K. Wooster, Scott R. Dusterhoff, Yantao Cui, Leonard S. Sklar, William E. Dietrich, and Mary Malko. Sediment supply and relative size distribution effects on fine sediment infiltration into immobile gravels: FINE SEDIMENT INFILTRATION INTO IMMOBILE GRAVELS. *Water Resources Research*, March 2008. URL: http://doi.wiley.com/10.1029/2006WR005815 (visited on 2023-06-09), doi:10.1029/2006WR005815.

[75] Yantao Cui, Chris Paola, and Gary Parker. Numerical simulation of aggradation and downstream fining. *Journal of Hydraulic Research*, 34(2):185–204, March 1996. URL: https://www.tandfonline.com/doi/full/10.1080/00221689609498496 (visited on 2023-06-20), doi:10.1080/00221689609498496.

[76] L. M. Merckelbach and C. Kranenburg. Equations for effective stress and permeability of soft mud–sand mixtures. *Géotechnique*, 54(4):235–243, May 2004. URL: https://www.icevirtuallibrary.com/doi/10.1680/geot.2004.54.4.235 (visited on 2023-06-09), doi:10.1680/geot.2004.54.4.235.

[77] W. H. McAnally. *Aggregation and deposition of estuarial fine sediment*. PhD thesis, University of Florida, 1999.

[78] J.C. Winterwerp, A.J. Bale, M.C. Christie, K.R. Dyer, S. Jones, D.G. Lintern, A.J. Manning, and W. Roberts. Flocculation and settling velocity of fine sediment. In *Proceedings in Marine Science*, volume 5, pages 25–40. Elsevier, 2002. URL: https://linkinghub.elsevier.com/retrieve/pii/S1568269202800067 (visited on 2023-06-09), doi:10.1016/S1568-2692(02)80006-7.

[79] W. van Leussen. *Estuarine macroflocs and their role in fine-grained sediment transport*. PhD thesis, University of Utrecht, 1994.

[80] J.C. Winterwerp. On the dynamic of high-concentrated mud suspensions. 99:, 01 1999.

[81] Eric Wolanski, Takashi Asaeda, and Jorg Imberger. Mixing across a lutocline. *Limnology and Oceanography*, 34(5):931–938, July 1989. URL: http://doi.wiley.com/10.4319/lo.1989.34.5.0931 (visited on 2023-06-20), doi:10.4319/lo.1989.34.5.0931.

[82] M. Smoluchowski. Versuch einer mathematischen theorie des koagulations-kinetik kolloid losungen. *Zeitschrift fur Physikalisch@phdthesise Chemie*, 92:129–168, 1917.

[83] C. Kranenburg. The fractal structure of cohesive sediment aggregates. *Estuarine, Coastal and Shelf Science*, 39(6):451–460, January 1994. URL: https://linkinghub.elsevier.com/retrieve/pii/S0272771406800028 (visited on 2023-06-20), doi:10.1016/S0272-7714(06)80002-8.

[84] W.H. McAnally and A.J. Mehta. Collisional aggregation of fine estuarial sediment. In *Proceedings in Marine Science*, volume 3, pages 19–39. Elsevier, 2000. URL: https://linkinghub.elsevier.com/retrieve/pii/S1568269200801102 (visited on 2023-08-04), doi:10.1016/S1568-2692(00)80110-2.

[85] W.H. McAnally and A.J. Mehta. Significance of Aggregation of Fine Sediment Particles in Their Deposition. *Estuarine, Coastal and Shelf Science*, 54(4):643–653, April 2002. URL: https://linkinghub.elsevier.com/retrieve/pii/S0272771401908479 (visited on 2023-08-04), doi:10.1006/ecss.2001.0847.

[86] Florent Grasso, Pierre Le Hir, and Philippe Bassoullet. Numerical modelling of mixed-sediment consolidation. *Ocean Dynamics*, 65(4):607–616, 2015. URL: https://doi.org/10.1007/s10236-015-0818-x, doi:10.1007/s10236-015-0818-x.

[87] E. A. Toorman. Sedimentation and self-weight consolidation: general unifying theory. *Géotechnique*, 46(1):103–113, 1996. URL: https://doi.org/10.1680/geot.1996.46.1.103, arXiv:https://doi.org/10.1680/geot.1996.46.1.103, doi:10.1680/geot.1996.46.1.103.

[88] Alain Alexis, Philippe Bassoullet, Pierre Le Hir, and Charles Teisson. *Consolidation of Soft Marine Soils: Unifying Theories, Numerical Modelling and in Situ Experiments*, chapter, pages 2949–2961. Coastal Engineering 1992, 1992. URL: https://ascelibrary.org/doi/abs/10.1061/9780872629332.225, arXiv:https://ascelibrary.org/doi/pdf/10.1061/9780872629332.225, doi:10.1061/9780872629332.225.

[89] E. A. Toorman. Sedimentation and self-weight consolidation: constitutive equations and numerical modelling. *Géotechnique*, 49(6):709–726, 1999. URL: https://doi.org/10.1680/geot.1999.49.6.709, arXiv:https://doi.org/10.1680/geot.1999.49.6.709, doi:10.1680/geot.1999.49.6.709.

[90] G. Bartholomeeusen, G. C. Sills, D. Znidarčić, W. Van Kesteren, L. M. Merckelbach, R. Pyke, W. D. Carrier, H. Lin, D. Penumadu, H. Winterwerp, S. Masala, and D. Chan. Sidere: numerical prediction of large-strain consolidation. *Géotechnique*, 52(9):639–648, 2002. URL: https://doi.org/10.1680/geot.2002.52.9.639, arXiv:https://doi.org/10.1680/geot.2002.52.9.639, doi:10.1680/geot.2002.52.9.639.

[91] L. M. Merckelbach and C. Kranenburg. Determining effective stress and permeability equations for soft mud from simple laboratory experiments. *Géotechnique*, 54(9):581–591, 2004. URL: https://doi.org/10.1680/geot.2004.54.9.581, arXiv:https://doi.org/10.1680/geot.2004.54.9.581, doi:10.1680/geot.2004.54.9.581.

[92] Katerini Kombiadou, Florian Ganthy, Verney Romaric, Martin Plus, and Sottolichio Aldo. Modelling the effects of zostera noltei meadows on sediment dynamics: application to the arcachon lagoon. *Ocean Dynamics*, 64(10):1499–1516, 2014. URL:, doi:https://doi.org/10.1007/s10236-014-0754-1.

[93] Florian Ganthy, Romaric Verney, and Franck Dumas. Improvements of a process-based model for 2- and 3-dimensional simulation of flow in presence of various obstructions. Preprint available at SSRN: https://ssrn.com/abstract=4775274 or http://dx.doi.org/10.2139/ssrn.4775274, 2024.

[94] Mohamed Abdelrhman. Modeling coupling between eelgrass zostera marina and water flow. *Marine Ecology-progress Series - MAR ECOL-PROGR SER*, 338:81–96, 05 2007. doi:10.3354/meps338081.

[95] Mohamed Abdelrhman. Effect of eelgrass zostera marina canopies on flow and transport. *Marine Ecology-progress Series - MAR ECOL-PROGR SER*, 248:67–83, 02 2003. doi:10.3354/meps248067.

[96] Nicolas Gruber, Hartmut Frenzel, Scott C. Doney, Patrick Marchesiello, James C. McWilliams, John R. Moisan, John J. Oram, Gian-Kasper Plattner, and Keith D. Stolzenbach. Eddy-resolving simulation of plankton ecosystem dynamics in the California Current System. *Deep Sea Research Part I: Oceanographic Research Papers*, 53(9):1483–1516, September 2006. URL: https://linkinghub.elsevier.com/retrieve/pii/S0967063706001713 (visited on 2023-06-09), doi:10.1016/j.dsr.2006.06.005.

[97] Nicolas Gruber, Zouhair Lachkar, Hartmut Frenzel, Patrick Marchesiello, Matthias Münnich, James C. McWilliams, Takeyoshi Nagai, and Gian-Kasper Plattner. Eddy-induced reduction of biological production in eastern boundary upwelling systems. *Nature Geoscience*, 4(11):787–792, November 2011. URL: https://www.nature.com/articles/ngeo1273 (visited on 2023-06-09), doi:10.1038/ngeo1273.

[98] E. Gutknecht, I. Dadou, B. Le Vu, G. Cambon, J. Sudre, V. Garçon, E. Machu, T. Rixen, A. Kock, A. Flohr, A. Paulmier, and G. Lavik. Coupled physical/biogeochemical modeling including O₂-dependent processes in the Eastern Boundary Upwelling Systems: application in the Benguela. *Biogeosciences*, 10(6):3559–3591, June 2013. URL: https://bg.copernicus.org/articles/10/3559/2013/ (visited on 2023-06-09), doi:10.5194/bg-10-3559-2013.

[99] B. Aumont, S. Szopa, and S. Madronich. Modelling the evolution of organic carbon during its gas-phase tropospheric oxidation: development of an explicit model based on a self generating approach. *Atmospheric Chemistry and Physics*, 5(9):2497–2517, September 2005. URL: https://acp.copernicus.org/articles/5/2497/2005/ (visited on 2023-06-09), doi:10.5194/acp-5-2497-2005.

[100] Martin Huret, Isabelle Dadou, Franck Dumas, Pascal Lazure, and Véronique Garçon. Coupling physical and biogeochemical processes in the Río de la Plata plume. *Continental Shelf Research*, 25(5-6):629–653, March 2005. URL: https://linkinghub.elsevier.com/retrieve/pii/S0278434304002614 (visited on 2023-06-09), doi:10.1016/j.csr.2004.10.003.

[101] E.V. Yakushev, F. Pollehne, G. Jost, I. Kuznetsov, B. Schneider, and L. Umlauf. Analysis of the water column oxic/anoxic interface in the Black and Baltic seas with a numerical model. *Marine Chemistry*, 107(3):388–410, December 2007. URL: https://linkinghub.elsevier.com/retrieve/pii/S0304420307001314 (visited on 2023-06-09), doi:10.1016/j.marchem.2007.06.003.

[102] Teodoro Coba De La Peña, Francisco J. Redondo, Esteban Manrique, M. M. Lucas, and José J. Pueyo. Nitrogen fixation persists under conditions of salt stress in transgenic Medicago truncatula plants expressing a cyanobacterial flavodoxin: Flavodoxin induces salt tolerance in nodules. *Plant Biotechnology Journal*, 8(9):954–965, December 2010. URL: https://onlinelibrary.wiley.com/doi/10.1111/j.1467-7652.2010.00519.x (visited on 2023-06-09), doi:10.1111/j.1467-7652.2010.00519.x.

[103] P. Suntharalingam, J. L. Sarmiento, and J. R. Toggweiler. Global significance of nitrous-oxide production and transport from oceanic low-oxygen zones: A modeling study. *Global Biogeochemical Cycles*, 14(4):1353–1370, December 2000. URL: http://doi.wiley.com/10.1029/1999GB900100 (visited on 2023-06-09), doi:10.1029/1999GB900100.

[104] Parvadha Suntharalingam, Erik Buitenhuis, Corinne Le Quéré, Frank Dentener, Cynthia Nevison, James H. Butler, Hermann W. Bange, and Grant Forster. Quantifying the impact of anthropogenic nitrogen deposition on oceanic nitrous oxide: ANTHROPOGENIC N DEPOSITION AND OCEAN N $_\textrm 2$ O. *Geophysical Research Letters*, 39(7):n/a–n/a, April 2012. URL: http://doi.wiley.com/10.1029/2011GL050778 (visited on 2023-06-09), doi:10.1029/2011GL050778.

[105] de Boyer Montegut C., Madec G., S. Fischer A., Lazar A., and Iudicone D. Mixed layer depth over the global ocean : an examination of profile data and a profile-based climatology. *Journal of Geophysical Research*, 109():C12003, December 2004. URL: https://www.researchgate.net/publication/229010378_Mixed_layer_depth_over_the_global_ocean_An_examination_of_profile_data_and_a_profile-based_climatology (visited on 2024-10-22), doi:10.1029/2004JC002378.

[106] Giulio Boccaletti, Ronald C. Pacanowski, S. George, H. Philander, and Alexey V. Fedorov. The Thermal Structure of the Upper Ocean. *Journal of Physical Oceanography*, 34(4):888–902, April 2004. URL: http://journals.ametsoc.org/doi/10.1175/1520-0485(2004)034\T1\textless{}0888:TTSOTU\T1\textgreater{}2.0.CO;2 (visited on 2023-06-21), doi:10.1175/1520-0485(2004)034<0888:TTSOTU>2.0.CO;2.

[107] Philippe Estrade, Patrick Marchesiello, Alain Colin De Verdière, and Claude Roy. Cross-shelf structure of coastal upwelling: A two — dimensional extension of Ekman's theory and a mechanism for inner shelf upwelling shut down. *Journal of Marine Research*, 66(5):589–616, September 2008. URL: http://openurl.ingenta.com/content/xref?genre=article&issn=0022-2402&volume=66&issue=5&spage=589 (visited on 2023-06-21), doi:10.1357/002224008787536790.

[108] P. Marchesiello and P. Estrade. Upwelling limitation by onshore geostrophic flow. *Journal of Marine Research*, 68(1):37–62, January 2010. URL: http://www.ingentaconnect.com/content/10.1357/002224010793079004 (visited on 2023-06-21), doi:10.1357/002224010793079004.

[109] John P. Boyd. Equatorial Solitary Waves. Part I: Rossby Solitons. *Journal of Physical Oceanography*, 10(11):1699–1717, November 1980. URL: http://journals.ametsoc.org/doi/10.1175/1520-0485(1980)010\T1\textless{}1699:ESWPIR\T1\textgreater{}2.0.CO;2 (visited on 2023-06-21), doi:10.1175/1520-0485(1980)010<1699:ESWPIR>2.0.CO;2.

[110] William Carlisle Thacker. Some exact solutions to the nonlinear shallow-water wave equations. *Journal of Fluid Mechanics*, 107(-1):499, June 1981. URL: http://www.journals.cambridge.org/abstract_S0022112081001882 (visited on 2023-06-21), doi:10.1017/S0022112081001882.

[111] James C. McWilliams and Glenn R. Flierl. On the Evolution of Isolated, Nonlinear Vortices. *Journal of Physical Oceanography*, 9(6):1155–1182, November 1979. URL: http://journals.ametsoc.org/doi/10.1175/1520-0485(1979)009\T1\textless{}1155:OTEOIN\T1\textgreater{}2.0.CO;2 (visited on 2023-06-21), doi:10.1175/1520-0485(1979)009<1155:OTEOIN>2.0.CO;2.

[112] Emanuele Di Lorenzo, William R. Young, and Stefan Llewellyn Smith. Numerical and Analytical Estimates of M2 Tidal Conversion at Steep Oceanic Ridges. *Journal of Physical Oceanography*,

36(6):1072–1084, June 2006. URL: http://journals.ametsoc.org/doi/10.1175/JPO2880.1 (visited on 2023-06-21), doi:10.1175/JPO2880.1.

[113] B. Weir, Y. Uchiyama, E. M. Lane, J. M. Restrepo, and J. C. McWilliams. A vortex force analysis of the interaction of rip currents and surface gravity waves. *Journal of Geophysical Research*, 116(C5):C05001, May 2011. URL: http://doi.wiley.com/10.1029/2010JC006232 (visited on 2023-06-21), doi:10.1029/2010JC006232.

[114] Patrick Marchesiello, Francis Auclair, Laurent Debreu, James McWilliams, Rafael Almar, Rachid Benshila, and Franck Dumas. Tridimensional nonhydrostatic transient rip currents in a wave-resolving model. *Ocean Modelling*, 163:101816, July 2021. URL: https://linkinghub.elsevier.com/retrieve/pii/S1463500321000676 (visited on 2023-06-21), doi:10.1016/j.ocemod.2021.101816.

[115] Patrick Marchesiello, Julien Chauchat, Hassan Shafiei, Rafael Almar, Rachid Benshila, Franck Dumas, and Laurent Debreu. 3D wave-resolving simulation of sandbar migration. *Ocean Modelling*, 180:102127, December 2022. URL: https://linkinghub.elsevier.com/retrieve/pii/S146350032200141X (visited on 2023-06-21), doi:10.1016/j.ocemod.2022.102127.

[116] Hervé Michallet, B. Gerben Ruessink, Mariana Vieira Lima Matias da Rocha, Anouk de Bakker, Dominic A. van Der A, Andrea Ruju, Paulo A. Silva, Nadia Sénéchal, Vincent Marieu, Marion Tissier, Rafael Almar, Tiago Abreu, Florent Birrien, Laure Vignal, Eric Barthélemy, Dominique Mouazé, Rodrigo Cienfuegos, and Peter Wellens. GLOBEX: Wave dynamics on a shallow sloping beach. In *HYDRALAB IV Joint User Meeting, Lisbon, July 2014*. Lisbonne, Portugal, July 2014. URL: https://hal.science/hal-01084718.

[117] XinJian Chen. A fully hydrodynamic model for three-dimensional, free-surface flows. *International Journal for Numerical Methods in Fluids*, 42(9):929–952, July 2003. URL: https://onlinelibrary.wiley.com/doi/10.1002/fld.557 (visited on 2023-06-21), doi:10.1002/fld.557.

[118] J. O. Shin, S. B. Dalziel, and P. F. Linden. Gravity currents produced by lock exchange. *Journal of Fluid Mechanics*, 521:1–34, December 2004. URL: http://www.journals.cambridge.org/abstract_S002211200400165X (visited on 2023-06-21), doi:10.1017/S002211200400165X.

[119] Mehmet Ilıcak, Alistair J. Adcroft, Stephen M. Griffies, and Robert W. Hallberg. Spurious dianeutral mixing and the role of momentum closure. *Ocean Modelling*, 45-46:37–58, January 2012. URL: https://linkinghub.elsevier.com/retrieve/pii/S1463500311001685 (visited on 2023-06-21), doi:10.1016/j.ocemod.2011.10.003.

[120] D. A. Horn, J. Imberger, and G. N. Ivey. The degeneration of large-scale interfacial gravity waves in lakes. *Journal of Fluid Mechanics*, 434:181–207, May 2001. URL: https://www.cambridge.org/core/product/identifier/S0022112001003536/type/journal_article (visited on 2023-06-21), doi:10.1017/S0022112001003536.

[121] Jared Penney, Yves Morel, Peter Haynes, Francis Auclair, and Cyril Nguyen. Diapycnal mixing of passive tracers by Kelvin–Helmholtz instabilities. *Journal of Fluid Mechanics*, 900:A26, October 2020. URL: https://www.cambridge.org/core/product/identifier/S0022112020004838/type/journal_article (visited on 2023-06-21), doi:10.1017/jfm.2020.483.

[122] Wen Long, James T. Kirby, and Zhiyu Shao. A numerical scheme for morphological bed level calculations. *Coastal Engineering*, 55(2):167–180, February 2008. URL: https://linkinghub.elsevier.com/retrieve/pii/S0378383907001068 (visited on 2023-06-21), doi:10.1016/j.coastaleng.2007.09.009.

[123] Christopher R. Sherwood, Alfredo L. Aretxabaleta, Courtney K. Harris, J. Paul Rinehimer, Romaric Verney, and Bénédicte Ferré. Cohesive and mixed sediment in the Regional Ocean Modeling System (ROMS v3.6) implemented in the Coupled Ocean–Atmosphere–Wave–Sediment Transport Modeling System (COAWST r1234). *Geoscientific Model Development*, 11(5):1849–1871, May 2018. URL: https://gmd.copernicus.org/articles/11/1849/2018/ (visited on 2023-06-21), doi:10.5194/gmd-11-1849-2018.

[124] Thomas Kilpatrick, Niklas Schneider, and Bo Qiu. Boundary Layer Convergence Induced by Strong Winds across a Midlatitude SST Front*. *Journal of Climate*, 27(4):1698–1718, February 2014. URL: http://journals.ametsoc.org/doi/10.1175/JCLI-D-13-00101.1 (visited on 2022-02-09), doi:10.1175/JCLI-D-13-00101.1.

[125] Michael A. Spall. Midlatitude Wind Stress–Sea Surface Temperature Coupling in the Vicinity of Oceanic Fronts. *Journal of Climate*, 20(15):3785–3801, August 2007. URL: http://journals.ametsoc.org/doi/10.1175/JCLI4234.1 (visited on 2022-02-09), doi:10.1175/JCLI4234.1.

[126] Alex Ayet and Jean-Luc Redelsperger. An analytical study of the atmospheric boundary-layer flow and divergence over an SST front. *Quarterly Journal of the Royal Meteorological Society*, 145(723):2549–2567, July 2019. URL: https://onlinelibrary.wiley.com/doi/10.1002/qj.3578 (visited on 2022-03-31), doi:10.1002/qj.3578.

[127] Florian Ganthy, Laura Soissons, Pierre-Guy Sauriau, Romaric Verney, and Aldo Sottolichio. Effects of short flexible seagrass zostera noltei on flow, erosion and deposition processes determined using flume experiments. *Sedimentology*, 62(4):997–1023, 2015. URL: https://archimer.ifremer.fr/doc/00244/35507/, doi:https://doi.org/10.1111/sed.12170.

[128] Florian Ganthy. *Rôle des herbiers de zostères (Zostera noltii) sur la dynamique sédimentaire du Bassin d'Arcachon*. PhD thesis, Universté de Bordeaux 1, 12 2011. URL: https://archimer.ifremer.fr/doc/00060/17170/.

[129] George L. Mellor and Tetsuji Yamada. A Hierarchy of Turbulence Closure Models for Planetary Boundary Layers. *Journal of the Atmospheric Sciences*, 31(7):1791–1806, October 1974. URL: http://journals.ametsoc.org/doi/10.1175/1520-0469(1974)031\T1\textless{}1791:AHOTCM\T1\textgreater{}2.0.CO;2 (visited on 2023-06-21), doi:10.1175/1520-0469(1974)031<1791:AHOTCM>2.0.CO;2.

[130] George Mellor. The Three-Dimensional Current and Surface Wave Equations. *Journal of Physical Oceanography*, 33(9):1978–1989, September 2003. URL: http://journals.ametsoc.org/doi/10.1175/1520-0485(2003)033\T1\textless{}1978:TTCASW\T1\textgreater{}2.0.CO;2 (visited on 2023-06-21), doi:10.1175/1520-0485(2003)033<1978:TTCASW>2.0.CO;2.

[131] W. P. Budgell. Numerical simulation of ice-ocean variability in the Barents Sea region: Towards dynamical downscaling. *Ocean Dynamics*, 55(3-4):370–387, December 2005. URL: http://link.springer.com/10.1007/s10236-005-0008-3 (visited on 2023-06-21), doi:10.1007/s10236-005-0008-3.

[132] James A. Carton. Sea level rise and the warming of the oceans in the Simple Ocean Data Assimilation (SODA) ocean reanalysis. *Journal of Geophysical Research*, 110(C9):C09006, 2005. URL: http://doi.wiley.com/10.1029/2004JC002817 (visited on 2023-06-21), doi:10.1029/2004JC002817.

[133] Jean-Michel Lellouche, Eric Greiner, Romain Bourdallé-Badie, Gilles Garric, Angelique Melet, Marie Drévillon, Clément Bricaud, Mathieu Hamon, Olivier Le Galloudec, Charly Regnier, Tony Candela, Charles-Emmanuel Testut, Florent Gasparin, Giovanni Ruggiero, Mounir Benkiran, Yann Drillet, and Le Traon Pierre-Yves. *Frontiers In Earth Science*, 2021. doi:https://doi.org/10.3389/feart.2021.698876.

[134] Gary D. Egbert and Svetlana Y. Erofeeva. Efficient Inverse Modeling of Barotropic Ocean Tides. *Journal of Atmospheric and Oceanic Technology*, 19(2):183–204, February 2002. URL: http://journals.ametsoc.org/doi/10.1175/1520-0426(2002)019\T1\textless{}0183:EIMOBO\T1\textgreater{}2.0.CO;2 (visited on 2023-06-21), doi:10.1175/1520-0426(2002)019<0183:EIMOBO>2.0.CO;2.

[135] Wessel, P. and W. H. F. Smith. A global self-consistent, hierarchical, high-resolution shoreline database. *J. Geophys. Res.*, 101():8741–8743, 1996.

[136] Kenneth S. Casey and Peter Cornillon. A Comparison of Satellite and In Situ–Based Sea Surface Temperature Climatologies. *Journal of Climate*, 12(6):1848–1863, June 1999. URL: http://journals.ametsoc.org/doi/10.1175/1520-0442(1999)012\T1\textless{}1848:ACOSAI\T1\textgreater{}2.0.CO;2 (visited on 2023-06-21), doi:10.1175/1520-0442(1999)012<1848:ACOSAI>2.0.CO;2.

[137] Walter H. F. Smith and David T. Sandwell. Global Sea Floor Topography from Satellite Altimetry and Ship Depth Soundings. *Science*, 277(5334):1956–1962, September 1997. URL: https://www.science.org/doi/10.1126/science.277.5334.1956 (visited on 2023-06-21), doi:10.1126/science.277.5334.1956.

[138] O. Aumont and L. Bopp. Globalizing results from ocean in situ iron fertilization studies: GLOBALIZING IRON FERTILIZATION. *Global Biogeochemical Cycles*, 20(2):n/a–n/a, June 2006. URL: http://doi.wiley.com/10.1029/2005GB002591 (visited on 2023-06-21), doi:10.1029/2005GB002591.

[139] Marchesiello, Lefèvre, Pierrick Penven, Florian Lemarié, Laurent Debreu, Pascal Douillet, A. Vega, P. Derex, Vincent Echevin, and Boris Dewitte. Keys to affordable regional marine forecast systems. In 2008. URL: https://api.semanticscholar.org/CorpusID:135117833.

[140] R. A. Flather and A. M. Davies. Note on a preliminary scheme for storm surge prediction using numerical models. *Quarterly Journal of the Royal Meteorological Society*, 102(431):123–132, Jan-

uary 1976. URL: https://onlinelibrary.wiley.com/doi/10.1002/qj.49710243110 (visited on 2023-06-21), doi:10.1002/qj.49710243110.

[141] Jie Yu. Effects of wave-current interaction on rip currents. *Journal of Geophysical Research*, 108(C3):3088, 2003. URL: http://doi.wiley.com/10.1029/2001JC001105 (visited on 2023-06-21), doi:10.1029/2001JC001105.