# CROCO

Victor Trappler

April 22, 2020

## Introduction

CROCO is a new oceanic modeling system built upon ROMS_AGRIF and the non-hydrostatic kernel of SNH (under testing), gradually including algorithms from MARS3D (sediments) and HYCOM (vertical coordinates). An important objective for CROCO is to resolve very fine scales (especially in the coastal area), and their interactions with larger scales. It is the oceanic component of a complex coupled system including various components, e.g., atmosphere, surface waves, marine sediments, biogeochemistry and ecosystems[1].

In this document, I will try to provide a summary of my understanding of this model and its use, especially in the light of my PhD work.

## 1 Numerics

### 1.1 Parametrization of the bottom friction

**Linear friction**

$$(\tau_b^x, \tau_b^y) = -r(u_b, v_b) \tag{1}$$

**Quadratic (constant)**

$$(\tau_b^x, \tau_b^y) = C_d\sqrt{u_b^2 + v_b^2}(u_b, v_b) \tag{2}$$

**Quadratic with Von Karman log-layer**

$$c(\tau_b^x, \tau_b^y) = C_d\sqrt{u_b^2 + v_b^2}(u_b, v_b) \tag{3}$$

$$C_d = \begin{cases} \left(\frac{\kappa}{\log(\Delta z_b / r_z)}\right)^2 & \text{for } C_d \in [C_d^{\min}, C_d^{\max}] \\ C_d^{\min} \\ C_d^{\max} \end{cases} \tag{4}$$

$$\kappa = 0.41 \tag{5}$$

$$\tag{6}$$

### 1.2 Numerical methods used

## 2 Utilisation

CROCO is written mainly in FORTRAN, so it needs to be first compiled, then executed

[1]taken from `http://www.croco-ocean.org/`

## 2.1 Compilation job

### 2.1.1 param.h

Initialize parameters of the simulation, especially the number of tides to take into account:

- Physical grid

```
    #elif defined FRICTION_TIDES
      parameter (LLm0=139, MMm0=164,    N=1)
```

- NTIDES

```
!----------------------------------------------------------
! Tides, Wetting-Drying, Point sources, Floast, Stations
!----------------------------------------------------------

#if defined SSH_TIDES || defined UV_TIDES
      integer Ntides              ! Number of tides
                                  ! ====== == =====
# if defined IGW || defined S2DV
      parameter (Ntides=1)
# elif defined(FRICTION_TIDES)
      parameter (Ntides=10) ! HERE to change number
# else
      parameter (Ntides=8)
# endif
```

### 2.1.2 cppdefs.h

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

### 2.1.3 Compile

```
  #!/bin/sh
  ../OCEAN/jobcomp
```

The `jobcomp` executable file in bash prepares and compile CROCO. The relevant directory variables are:

- `RUNDIR`: The current directory, so `croco/Run/`
- `SOURCE`: The source directory, so `croco/OCEAN`
- `SCRDIR`: The *scratch* directory, so `croco/Run/Compile/`
- `ROOT_DIR`: the root directory, so `croco/`

It first set the compiler options according to the OS in place: `LINUX_FC=gfortran` with 64bits for instance. Afterwards, the source code is copied from `SOURCE` to `SCRDIR`. The local files (in `croco/Run/` then) overwrite those in `SCRDIR`. We change directory to `SCRDIR` (`Run/Compile`).

The compulation options are set:

- `CPP1=cpp -traditional -DLinux`: Preprocessing options for C, C++: "traditional" for compatibility, "DLinux" to predefine macro "Linux", with definition 1
- `CFT1 = gfortran`: Fortran compiler, with the flags:

- FFLAGS1=
  - -O3: Optimization of level 3:
  - -fdefault-real-8: set defaults real type to 8 bytes wide
  - -fdefault-double-8: set defaults double type to 8 bytes wide
  - -mcmodel=large: Might require a lot of static memory
  - -fno-align-commons: disable automatic alignment of all variables in "COMMON" block
  - -fbacktrace: Fortran runtime should output backtrace of fatal error
  - -fbounds-check: enable generation of runtime checks for array subscripts (deprecated, should be fcheck=bounds according to gfortran manual
  - -finit-real=nan: initialize REAL variables to (silent) NaN
  - -finit-integer=8888: initialize INTEGER variables to 8888

**TAPENADE** Turn on tracing (`set -x`) and exit on error (`set -e`). Copy all `.F` `.c` and `.h` files from `ROO_DIR/AD/` to `SCRDIR`, and `Makefile` as well.

It looks for tapenade with "ifexist" file:

```
  [ -f \${d}/tapenade\_3.14/bin/tapenade ]
```

**Makefile** After the sources are defined, let us take a look at the Makefile, in `croco/OCEAN/Compile/`. The basic structure of makefiles is the following:

```
product: source
    command
```

```
$(SBIN):  $(OBJS90) $(OBJS)
         $(LDR) $(FFLAGS) $(LDFLAGS) -o a.out $(OBJS90) $(OBJS) $(LCDF) $(LMPI)
         mv a.out \$(SBIN)
```

but with aliase

## 2.2 Execution

### 2.2.1 The .in file

**Timestepping**

```
  time_stepping: NTIMES   dt[sec]  NDTFAST  NINFO
                25920      10     1      1
```

`NTIMES` is the number of time steps for the simulation

`dt` is the time-step for the simulation

| Time simulated | `NTIMES` |
|---:|:---|
| 1 hour | 360 |
| 1 day | 8640 |
| 3 days | 25920 |
| 1 week (7 days) | 60480 |
| 1 month (30 days) | 259200 |
| 1 year (360 days) | 3110400 |
| 1 year (365 days) | 3153600 |

Table 1: Table of some values for NTIMES, with `dt` of 10s

```
restart:          NRST, NRPFRST / filename
                   720    -1
                   CROCO_FILES/croco_rst.nc
history: LDEFHIS, NWRT, NRPFHIS / filename
            T      180     0
            CROCO_FILES/croco_rst_obs_1mo.nc
```

`NRST` : Number of time-steps between saving a rst file

`NWRT` : Number of time-steps between saving to the history file

**Other input files**

```
forcing: filename
                          CROCO_FILES/croco_frc_M2S2K1.nc
climatology: filename
                          CROCO_FILES/croco_clm.nc
```

Here, the forcing filename is generated using MATLAB/OCTAVE and the `croco_tools`, that includes the tide

```
  bottom_drag:     RDRG [m/s],  RDRG2,  Zob [m],  Cdb_min, Cdb_max
                1.00d-04    0.00d+00    5.00d-06    1.00d-04    1.00d-01
```

## 2.3 Toward a black-box utilisation using `crocopy`

TBD

## 3 File structure, definition and calls

### 3.1 optim_driver.F

Subroutines
- state_control
- rms_fun_step
- rms_fun_step_2d

Functions
- rms

```
state_control(iicroot)
```

```
  call init_control
```

set ad_x, ad_g to 0

```
  call simul
```

- get cost and gradient on each processor (suffix _f indicates that this is the full vector (in contrast to the vector on each proc))
- ad_g_f is constructed from ad_g
- cost_f is updated

```
  call set_state(ad_x_f)
```

—

### 3.2 ATLN2/cost_fun.F

Subroutines
- ad_step
- cost_fun
- cost_fun_step
- cost_fun_step_2d
- cost_fun_step_2d_tile
- set_state
- set_state_2d
- set_state_2d_tile
- init_control
- save_croco_state
- restore_croco_state
- init_local_arrays

ad_step

```
ad_step
```

calls ad_ns (defined in adparam.h) in a row the step subroutine cost_fun

```
  cost_fun(ad_x, cost):
  call set_state
  for ta=1, ad_nt
      call ad_step()
      call cost_fun_step()
```

st_fun_step, _2d, _tile

```
    cost_fun_step(ad_x, icost, ad_nt, mode)
```

Loops over tiles etc

$$\begin{aligned} \text{if mode} = 3: &\quad \xi(\mathtt{i},\mathtt{j},\mathtt{knew}) - \mathtt{ad\_obs}(\mathtt{i},\mathtt{j},\mathtt{ta}+2) \\ \text{if mode} = 2: &\quad \mathtt{z0b} - \mathtt{z0b\_bck} \end{aligned}$$

and stores it to `cost`

set_state, _2d, _tile

```
set_state():
    z0b = ad_x
```

then saves `z0b` and $\xi$ to the files `z0b.proc.iteration` and `ssh.proc.iteration`

init_control

```
init_control():
    ad_x = 0
    ad_g = 0
    call init_local_arrays(ad_x)
```

save_croco_state

```
save_croco_state:
    *_bck = *
```

restore_croco_state

```
restore_croco_state:
    * = *_bck
```

init_local_arrays

```
init_local_arrays
```

## 3.3 adj_driver

Subroutines – simul

simul

```
simul(indic, sn, ad_x, cost, ad_g, izs, rzs, dzs)
    call save_croco_state
    call cost_fun
    rms()
    call restore_croco_state
    call cost_fun_b() ! adjoint run
    call restore_croco_state
```