New parameterizations in the MOHIDLagrangian model

Mohsen Shabani
Universidad de Santiago de Compostela

June 24, 2025

Overview

General information

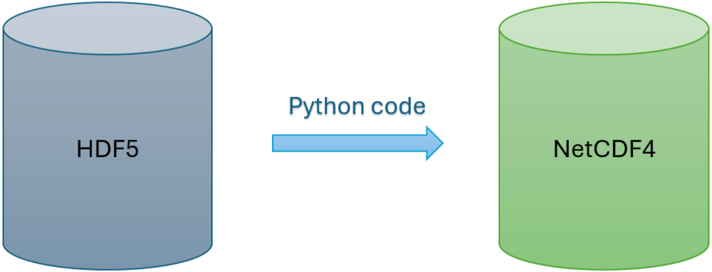


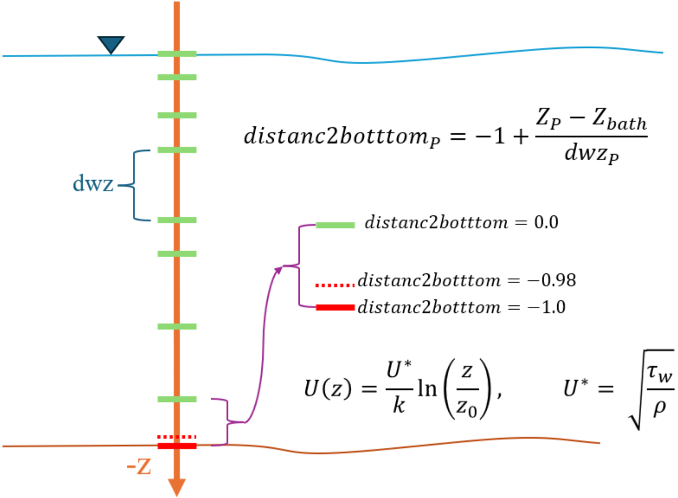
Figure: A Python code to convert HDF5 file to NetCDF4 format.

General information

	Base	Paper	Plastic	Coliform	Seed	Detritus
LagrangianKinematic	M✓	M✓	M✓	M✓	M✓	M✓
StokesDrift	M✓	M✓	M✓	M✓	M✓	M✓
Windage	M✓	M✓	M✓			
DiffusionMixingLength	M✓	M✓	M✓	M✓	M✓	M✓
Aging	M✓	M✓	M✓	M✓	M✓	M✓
DegradationLinear		L✓	L✓			
Buoyancy		V✓	V✓		V✓	V✓
Resuspension		V✓	V✓		V✓	V✓
BioFouling			L✓			
MortalityT90				C✓		
Dilution				C✓		
Degradation						D✓

kernel.f90

function Lagrangian Kinematic (self , sv , bdata , time)



function Lagrangian Kinematic (self , sv , bdata , time)

$$U^* = \frac{k \quad U(Z_{dwz})}{\ln(\frac{Z_{dwz}}{z_0})} \Rightarrow U(z) = U(Z_{dwz}) \quad \frac{k}{\ln(\frac{Z_{dwz}}{z_0})} \frac{\ln(\frac{z}{z_0})}{k}$$

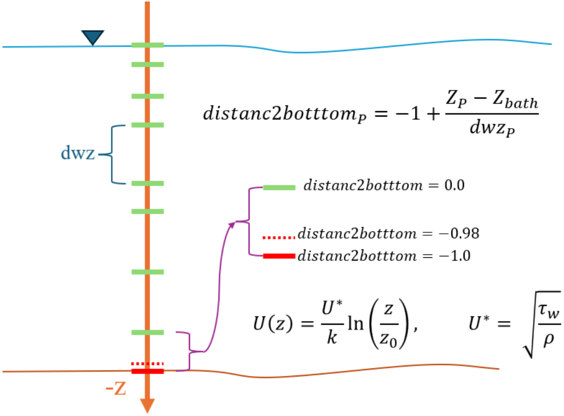
$$\Rightarrow U(z) = U(Z_{dwz}) \quad \frac{\ln(\frac{z}{z_0})}{\ln(\frac{Z_{dwz}}{z_0})}, \quad z_0 \leq z \leq Z_{dwz} \quad and \quad U : [u, v, w]$$

In MOHIDLagrangian:

```
    where (dist2bottom < threshold_bot_wat)
        aux_r8 = max((sv%state(:,col_dwz)/2),Hmin_Chezy) /
           Globals%Constants%Rugosity
        chezyZ = (VonKarman / dlog(aux_r8))**2
        sv%state(:,4) = var_hor_dt(:,col_u) * chezyZ
        sv%state(:,5) = var_hor_dt(:,col_v) * chezyZ
    end where
```

kernel.f90

function Lagrangian Kinematic (self , sv , bdata , time)



- The last measurement which is corresponed to last *dwz*, is variable (i.e., 2.0 to 0.1 meters).
- Hence, the threshold could be defined in the rugosity (z_0) scale (e.g. $10 \cdot z_0$).
- `threshold_bot_wat 0.5`, while we can define it as 0 (the last measurement).
- `function LagrangianVelModification` is created to modify the velocities.

kernel.f90

function DiffusionMixingLength (self , sv , bdata , time , dt)

- **Random velocity for direction (random walk)** $i = \{x, y, z\}$

$$v_i^{\text{rand}} = (2r_i - 1)\sqrt{\frac{\alpha \cdot D_i}{dt}} \quad \text{where} \quad r_i \sim U(0,1)$$

$$\frac{d}{dt}v_i^{\text{rand}} = (2r_i - 1) \cdot \frac{1}{dt}\sqrt{\frac{\alpha \cdot D_i \cdot |v_i|}{dt}} \quad \text{where} \quad r_i \sim U(0,1)$$

- Turbulent diffusion coefficient unit, D_i, is $[\text{m}^2\text{s}^{-1}]$.

kernel.f90

function DiffusionMixingLength (self , sv , bdata , time , dt)

- if `sv%state(:,10)` $> 2 \cdot$ `sv%resolution` and `sv%landIntMask` $<$ `landVal`

$$\texttt{DiffusionMixingLength(:,7)} = \frac{d}{dt} v_x^{rand} \quad , \alpha = 1$$

$$\texttt{DiffusionMixingLength(:,8)} = \frac{d}{dt} v_y^{rand} \quad , \alpha = 1$$

$$\texttt{DiffusionMixingLength(:,9)} = \frac{d}{dt} v_Z^{rand} \quad , \alpha = 10^{-6}$$

$$\texttt{DiffusionMixingLength(:,10)} = 0.0$$

`DiffusionMixingLength(:,i)` `i=7,8,9` are corresponded to the derivative of diffusion velocity in the directions of x, y, z. Also, `DiffusionMixingLength(:,10)` represents the derivative of diffusion mixing length.

kernel.f90

function DiffusionMixingLength (self , sv , bdata , time , dt)

Then, the new position will be modified based on the velocities calculated above.

$$dx = \mathtt{m2geo}(\frac{d}{dt}v_x^{\text{rand}}, \text{lon}) \cdot dt$$

$$dy = \mathtt{m2geo}(\frac{d}{dt}v_y^{\text{rand}}, \text{lat}) \cdot dt$$

$$dz = \frac{d}{dt}v_z^{\text{rand}} \cdot dt$$

```
DiffusionMixingLength(:,1) = Utils%m2geo(DiffusionMixingLength(:,7), sv%state(:,2), .false.)*dt

DiffusionMixingLength(:,2) = Utils%m2geo(DiffusionMixingLength(:,8), sv%state(:,2), .true.)*dt

DiffusionMixingLength(:,3) = DiffusionMixingLength(:,9)*dt
```

The kernel operates on the time derivative. Since $x = vt$, taking the derivative with respect to time yields the velocity v. Therefore, to correctly update positions, the corresponding velocity must be included in the kernel. $\frac{d}{dt}v_i^{\text{rand}} \cdot dt = v_i^{\text{rand}}$

kernel.f90

function DiffusionMixingLength (self , sv , bdata , time , dt)

$$\frac{d}{dt} v_i^{\text{rand}} = (2r_i - 1) \cdot \frac{1}{dt} \sqrt{\frac{\alpha \cdot D_i \cdot |v_i|}{dt}} \quad \text{where} \quad r_i \sim U(0,1)$$

- Why is it multiplied by v_i?
- One possible limitation of the random walk model arises in regions where the velocity v_i is zero or nearly zero. By multiplying by v_i, we ensure that the random walk component becomes zero and is effectively excluded from the calculation.
- Since multiplying by v_i introduces a unit inconsistency. It may be more appropriate to use a binary (dimensionless) switch to activate or deactivate the random walk term in such regions.

kernelVerticalMotion.f90
function Buoyancy(self, sv, bdata, time)

- Modification of z position regarding the buoyancy term in MOHIDLagrangian

$$\texttt{Buoyancy(:,3)} = \begin{cases} \mathrm{sign}_z \cdot \sqrt{-2g \cdot \frac{S}{C_d} \cdot R_\rho}, & \text{if } Re \neq 0 \text{ and dist2bottom} > \text{threshold} \\ 0, & \text{otherwise} \end{cases}$$

- While S in MOHIDLagrangian called as shape factor,

$$S_{MOHIDLagrangian} = \frac{\left(\frac{6}{\pi} V_{real}\right)^{\frac{1}{3}}}{\left(\frac{4}{\pi} A_{real}\right)^{\frac{1}{2}}} \quad [1]$$

Please note that `Buoyancy(:,3)` should be in the unit of velocity [m/s].

kernelVerticalMotion.f90
function Buoyancy(self, sv, bdata, time)

- Let's do a unit check in the mentioned formula in MOHIDLagrangian,

$$\texttt{Buoyancy(:,3)} = \begin{cases} \text{sign}_z[1] \cdot \sqrt{-2g[\frac{m}{s^2}] \cdot \frac{S[1]}{C_d[1]} \cdot R_\rho[1]}, & \text{if Cond.} \\ 0, & \text{otherwise} \end{cases}$$

$$\texttt{Buoyancy(:,3)} = \begin{cases} [1] \cdot \sqrt{[\frac{m}{s^2}] \cdot \frac{[1]}{[1]} \cdot [1]} = \sqrt{[\frac{m}{s^2}]} \neq [\frac{m}{s}], & \text{if Cond.} \\ 0, & \text{otherwise} \end{cases}$$

In MOHIDLagrangian, we have unit inconsistency.

kernelVerticalMotion.f90
Buoyancy Calculation

Lets consider a particle with 3 forces, its weightF_W, buoyancyF_B, dragF_D. Assume that the positive direction is upward and buoyancy and drag force are in the positive direction. Hence,

$$F_D + F_B = F_W.$$

By replacing the forces and considering **shape factor**,Φ,

$$\frac{1}{2}\rho_f C_{D,ref} \Phi A_{CS,real} v^2 + \rho_f g V_{real} = -\rho_p g V_{real}.$$

Now, by rearranging the above equation the velocity of particle will be:

$$v = \sqrt{-2g(\frac{\rho_p - \rho_f}{\rho_f})\frac{1}{\Phi C_{D,ref}}\frac{V_{real}}{A_{CS,real}}} = \sqrt{-2g \cdot R_\rho \cdot \frac{1}{\Phi C_{D,ref}} \cdot \frac{V_{real}}{A_{CS,real}}} \quad [m/s].$$

kernelVerticalMotion.f90
Buoyancy Calculation

On the other side, the **shape factor** can be defined as[1].

$$\Phi = \frac{\text{Actual surface area of particle}}{\text{Surface area of the sphere of same volume}}$$

where
sphericity, ψ , is the inverse of the shape factor.

- $\Phi = 1$ is the reference case and corresponds to a sphere.
- $\Phi > 1$ is corresponded to a **Prolate spheroid** (elongated).
- $\Phi < 1$ is corresponded to an **Oblate spheroid** (flattened).
- $\Phi >> 1$ is corresponded to a **Flat plate**.

$$V_{real} = \frac{4}{3}\pi r_{shpere}^3 \Rightarrow r_{shpere} = (\frac{3}{4\pi}V_{real})^{\frac{1}{3}}$$

[1] Please pay attention that we have different methods to consider shape factor, but at the end it should be dimensionless

kernelVerticalMotion.f90
Buoyancy Calculation

$$\Phi = \frac{\text{Actual surface area of particle}}{\text{Surface area of the sphere of same volume}}$$

Hence,

$$\Phi = \frac{A_{real}}{4\pi(r_{real})^2} = \frac{A_{real}}{4\pi(\frac{3}{4\pi}V_{real})^{\frac{2}{3}}} = \frac{A_{real}}{\pi^{\frac{1}{3}}(6V_{real})^{\frac{2}{3}}}$$

kernelVerticalMotion.f90
Buoyancy Calculation

- In the previous formula, C_D is a function of the Reynolds number.
- Reynolds number has to calculate in the relative velocity ($w_{rel} = |w_p - w_f|$). Hence, this form of formula needs an iterative method.
- MOHIDLagrangian uses explicit solver and using iterative method to calculate each settling velocity particle in each time step is not cost-effective.
- In MOHIDLagrangian,

```
MeanKvisco = 10^-3 ⇒ 10^-6 (kVisco = Globals%Constants%MeanKvisco)
fDensity = seaWaterDensity(sv%state(:,col_sal), sv%state(:,col_temp),sv%state(:,3))
kVisco = absoluteSeaWaterViscosity(sv%state(:,col_sal), sv%state(:,col_temp)) /fDensity
reynoldsNumber = self%Reynolds(sv%state(:,6), kvisco, sv%state(:,rIdx)*2)
```

kernelVerticalMotion.f90
Buoyancy Calculation

- Other options to calculate settling velocity

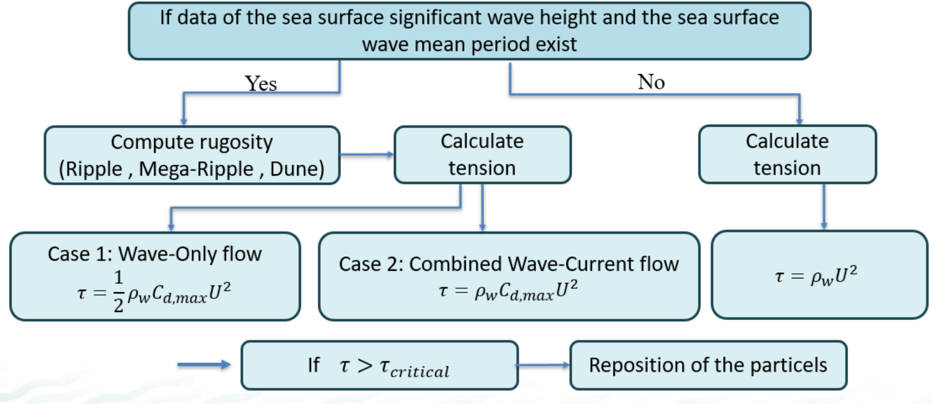
Review of formulations for terminal settling/rising velocities of plastics used in the reviewed numerical studies; $\Delta\rho = \frac{\rho_p - \rho_w}{\rho_w}$, $c_1 = 53.5\exp(-0.65\Psi)$, $c_2 = 5.65\exp(-2.5\Psi)$, $c_3 = 0.7 + 0.9\Psi$.

Reference	Expression		Particle shape
Lamb (1924)	$w_t = -\frac{g\Delta\rho D_n^2}{18\nu_w}, \quad \rho_p > \rho_w$	(C.4)	Spherical particles
Elliott (1986)	$w_t = -\frac{g D_n^2 \Delta\rho}{18\nu_w}, \quad \rho_p < \rho_w$	(C.5)	Spherical particles with $D_n < D_{cr}$
Elliott (1986)	$w_t = \left(-\frac{8}{3} g D_n \Delta\rho\right)^{1/2}, \quad \rho_p < \rho_w$	(C.6)	Spherical particles with $D_n > D_{cr}$
Wu and Wang (2006)	$w_t = -\frac{c_1 \nu_w}{c_2 D_n}\left[\sqrt{\frac{1}{4} + \left(\frac{4c_2}{3c_1^2} D_*^3\right)^{1/c_3}} - \frac{1}{2}\right]^{c_3}, \quad \rho_p > \rho_w$	(C.7)	Sediment particles
Zhiyao et al. (2008)	$w_t = -\frac{\nu_w}{D_n} D_*^3 \left(38.1 + 0.93 D_*^{12/7}\right)^{-7/8}, \quad \rho_p > \rho_w$	(C.8)	Sediment particles
Khatmullina and Isachenko (2017)	$w_t = -\frac{\pi}{2}\frac{g\,\Delta\rho}{\nu_w}\frac{D_n\, l_{max}}{55.238\, l_{max} + 12.691}, \quad \rho_p > \rho_w$	(C.9)	Cylindrical plastics
Wang et al. (2021)	$w_t = -1.0434(\Delta\rho g)^{0.495}\frac{D_n^{0.777}\Psi^{0.710}}{\nu_w^{0.124}}, \quad \rho_p > \rho_w$	(C.10)	Irregularly shaped plastics

kernelVerticalMotion.f90

function Resuspension(self, sv, bdata, time,dt)

- Resuspension model in MOHIDLagrangian (`if (dist2bottom(i) < landIntThreshold) then`)



kernelVerticalMotion.f90
function Resuspension(self, sv, bdata, time,dt)

- If sea surface wave data exists:

```
!Average velocity
U = sqrt(sv%state(i,4)**2.0 + sv%state(i,5)**2.0)/
        0.4* (dlog(bat/z0(i)) - 1.0 + z0(i)/bat)
```

Recall the velocity based on log-law, where z is calculated from the seabed.

$$U(z) = \frac{U^*}{k} \ln\left(\frac{z}{z_0}\right), \quad z = Z - Z_{bath}$$

Hence, the average velocity along this layer will be:

$$\bar{U} = \int_0^H U(z)dz = \frac{U^*}{k}\left\{\ln\left(\frac{H}{z_0}\right) - 1 + \frac{z_0}{H}\right\}, \quad H = Z_{dwz} - Z_{bath}.$$

kernelVerticalMotion.f90
function Resuspension(self, sv, bdata, time,dt)

- Deposition of the particle (`if (tension > Globals%Constants%Critical_Shear_Erosion) then`)

```
    where ((dist2bottom < landIntThreshold) .and. Tension>
        Globals%Constants%Critical_Shear_Erosion)
    !Tracer gets positive vertical velocity which corresponds
         to a percentage of the velocity modulus
    !Resuspension(:,3) = Globals%Constants%ResuspensionCoeff
        * velocity_mod
    !tracers gets brought up to 0.5m
    Resuspension(:,3) = 0.5/dt
    end where
```

Question: How does it show those suspended particles that move just above the seabed?

kernelVerticalMotion.f90
function Resuspension(self, sv, bdata, time,dt)

Question: How does it show those suspended particles that move just above the seabed?

- Find the U^* based on the log-law and later find the $\tau_{wall} = \rho U^{*2}$.
- Consider a threshold in rugosity scale (e.g. $10 \cdot z_0$).
- Lower than this threshold we Do not have a **Lagrangian kinematic** movement But if $\tau_{wall} > \tau_{cr}$ (resuspension condition) the particles move with the same velocity of the log-law.
- Buoyancy could affect on the particles which exist in this layer
- Resuspension Model can be replaced with a new model.

kernelVerticalMotion.f90
Resuspension: Deposition of the particles(Other option)

- Deposition: $M_{Dpos} = w_s \cdot C \cdot (1 - \frac{\tau_{wall}}{\tau_{cr,Dpos}}), \quad \tau_{wall} < \tau_{cr,Dpos}$
  - M_{Dpos} : flux rate of deposition (kg/m2/s)
  - w_s: settling velocity (m/s)
  - C: particle concentration (kg/m3)

  This is for the Flux rate of deposition, and it can use in the advection-diffusion equation as the flux at the boundaries.The MOHIDLagrangian works with particles and number of them. Hence, how we do we can use it in MOHIDLagrangian?

- $\frac{\text{Deposited particles (kg)}}{\text{Total mass (kg)}} = \frac{M_{Dpos} \cdot \Delta t \cdot A_{wall}}{C \cdot A_{wall} \cdot h}$
- $P_{Dpos} = \min\{1, \frac{w_s \cdot \Delta t}{h} \cdot (1 - \frac{\tau_{wall}}{\tau_{cr,Dpos}})\}$

kernelVerticalMotion.f90
Resuspension: Deposition of the particles(Other option)

- $P_{Dpos} = \min\{1, \frac{w_s \cdot \Delta t}{h} \cdot (1 - \frac{\tau_{wall}}{\tau_{cr,Dpos}})\}$
  - We can consider it as the portion of the deposited particles or probability of the deposition.
- The easiest way is to assume it as the probability of deposition of the particles, and by a random number, *Rand*, in the interval of [0,1]:
- if $z_p < h$ and $\tau_{wall} < \tau_{cr,Dpos}$
  if $Rand < P_{Dpos}$ then The particle deposits
  if $Rand > P_{Dpos}$ then particle moves with the flow
- This method does not guaranty that the exact number of particle deposit.

kernelVerticalMotion.f90
Resuspension: Erosion of the particles(Other option)

- Erosion flux rate (kg/m2/s):

$$M_{Ero} = M_0 \left(\frac{\tau_{wall}}{\tau_{cr,Ero}} - 1 \right), \quad \tau_{wall} > \tau_{cr,Ero}$$

- Fraction of eroded mass:

$$\frac{M_{Ero} \cdot \Delta t \cdot A_{wall}}{\rho_s \cdot A_{wall} \cdot h}$$

- Erosion probability:

$$P_{Ero} = \min \left\{ 1, \frac{M_0 \cdot \Delta t}{\rho_p \cdot h} \cdot \left(\frac{\tau_{wall}}{\tau_{cr,Ero}} - 1 \right) \quad or \quad \alpha \cdot \left(\frac{\tau_{wall}}{\tau_{cr,Ero}} - 1 \right) \right\}$$

  - It is assumed that all particles are distributed uniformly. Otherwise, a tuning parameter α could be used.
- If $z_p < h$ and $\tau_{wall} > \tau_{cr,Ero}$
  - If $Rand < P_{Ero}$ then particle is eroded (released)
  - If $Rand > P_{Ero}$ then particle stays on bed

Outlook

- Add a new model for windage [2]

$$U = \frac{U_c + U_w\sqrt{\frac{\rho_{air}}{\rho_{water}}\frac{S_{above}}{S_{below}}}}{1 + \sqrt{\frac{\rho_{air}}{\rho_{water}}\frac{S_{above}}{S_{below}}}} \qquad V = \frac{V_c + V_w\sqrt{\frac{\rho_{air}}{\rho_{water}}\frac{S_{above}}{S_{below}}}}{1 + \sqrt{\frac{\rho_{air}}{\rho_{water}}\frac{S_{above}}{S_{below}}}}$$

- Add a new model for biofouling [2]

Spheres:

$$\rho_p = \rho_0 \frac{R_0^3}{(R_0 + BT)^3} + \rho_D\left[1 - \frac{R_0^3}{(R_0 + BT)^3}\right],$$

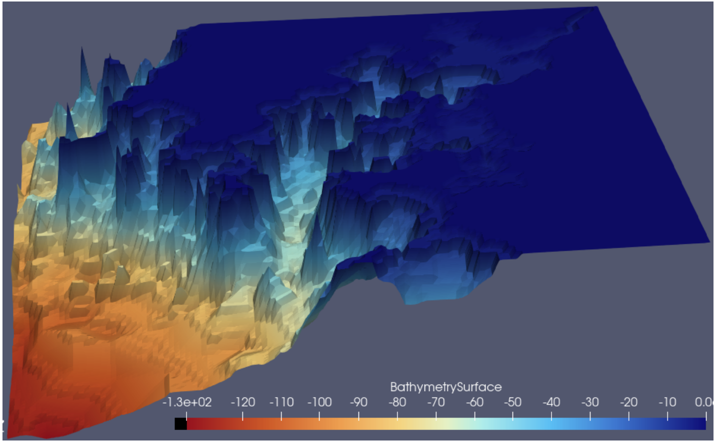
- Consider the variation of the particle radius which can effect on the buoyancy and biofouling.

[2] Jalón-Rojas, I., Wang, X. H., & Fredj, E. (2019). A 3D numerical model to Track Marine Plastic Debris (TrackMPD): Sensitivity of microplastic trajectories and fates to particle dynamical properties and physical processes. Marine pollution bulletin, 141, 256-272.

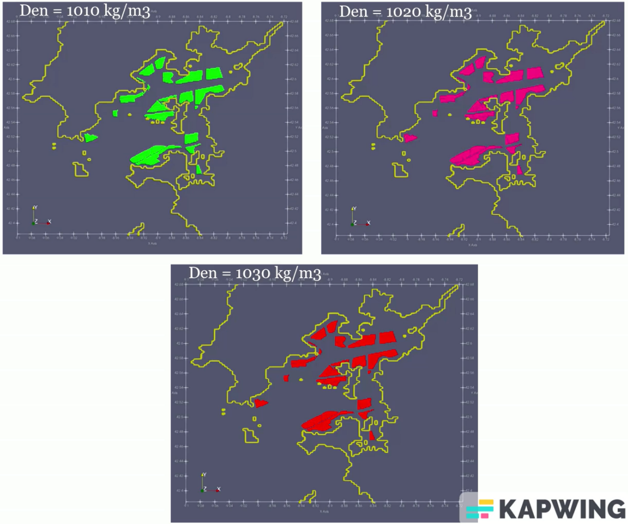
Simulation
Ría de Arousa

- Comparison of three different densities by using a new model for bouyancy term (setteling velocity).
  - Horizontal view
  - Vertical view

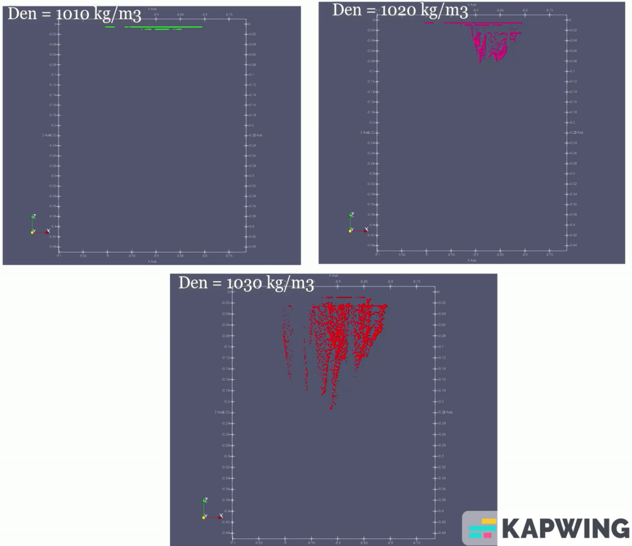
- Comparison of the old and new buoyancy models for the density= 1030 kg/m3.



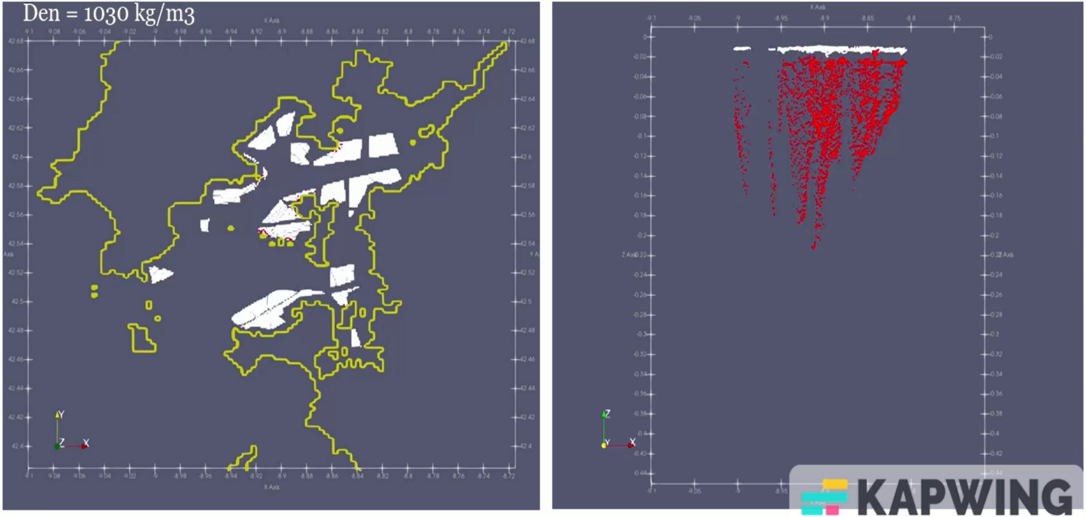
Simulation: New models
Ría de Arousa: Comparison of different densities (Horizontal view)



Simulation: New models
Ría de Arousa: Comparison of different densities (Vertical view)



Simulation
Ría de Arousa: Comparison of the old and new models



Thank you for your attention