AeoLiS Documentation

Release 1.0

Bas Hoonhout

Jan 02, 2023

CONTENTS

AeoLiS is a process-based model for simulating aeolian sediment transport in situations where supply-limiting factors are important, like in coastal environments. Supply-limitations currently supported are soil moisture contents, sediment sorting and armouring, bed slope effects, air humidity and roughness elements.

This documentation describes the Python implementation of the AeoLiS model. The source code of the Python implementation can be found at https://github.com/openearth/aeolis-python.

CHAPTER

ONE

CONTENTS

1.1 Model description

The model approach of [dVvTdVvR+14] is extended to compute the spatiotemporal varying sediment availability through simulation of the process of beach armoring. For this purpose the bed is discretized in horizontal grid cells and in vertical bed layers (2DV). Moreover, the grain size distribution is discretized into fractions. This allows the grain size distribition to vary both horizontally and vertically. A bed composition module is used to compute the sediment availability for each sediment fraction individually. This model approach is a generalization of existing model concepts, like the shear velocity threshold and critical fetch, and therefore compatible with these existing concepts.

1.1.1 Advection Scheme

A 1D advection scheme is adopted in correspondence with [dVvTdVvR+14] in which c $[\mathrm{kg/m^2}]$ is the instantaneous sediment mass per unit area in transport:

$$\frac{\partial c}{\partial t} + u_z \frac{\partial c}{\partial x} = E - D \tag{1.1}$$

t [s] denotes time and x [m] denotes the cross-shore distance from a zero-transport boundary. E and D $[\mathrm{kg/m^2/s}]$ represent the erosion and deposition terms and hence combined represent the net entrainment of sediment. Note that Equation (1.1) differs from Equation 9 in [dVvTdVvR+14] as they use the saltation height h [m] and the sediment concentration C_c $[\mathrm{kg/m^3}]$. As h is not solved for, the presented model computes the sediment mass per unit area $c = hC_\mathrm{c}$ rather than the sediment concentration C_c. For conciseness we still refer to c as the *sediment concentration*.

The net entrainment is determined based on a balance between the equilibrium or saturated sediment concentration c_sat $[\mathrm{kg/m^2}]$ and the instantaneous sediment transport concentration c and is maximized by the available sediment in the bed m_a $[\mathrm{kg/m^2}]$ according to:

$$E - D = \min\left(\frac{\partial m_\mathrm{a}}{\partial t} \quad ; \quad \frac{c_\mathrm{sat} - c}{T}\right) \tag{1.2}$$

T [s] represents an adaptation time scale that is assumed to be equal for both erosion and deposition. A time scale of 1 second is commonly used ([dVvTdVvR+14]).

The saturated sediment concentration c_sat is computed using an empirical sediment transport formulation (e.g. [Bag37b]):

$$q_\mathrm{sat} = \alpha C \frac{\rho_\mathrm{a}}{g} \sqrt{\frac{d_\mathrm{n}}{D_\mathrm{n}}} \left(u_z - u_\mathrm{th}\right)^3 \tag{1.3}$$

in which q_sat [kg/m/s] is the equilibrium or saturated sediment transport rate and represents the sediment transport capacity. u_z [m/s] is the wind velocity at height z [m] and u_th the velocity threshold [m/s]. The properties of the sediment in transport are represented by a series of parameters: C [–] is a parameter to account for the grain size distribution width, ρ_a [$\mathrm{kg/m^3}$] is the density of the air, g [$\mathrm{m/s^2}$] is the gravitational constant, d_n [m] is the nominal grain size and D_n [m] is a reference grain size. α is a constant to account for the conversion of the measured wind velocity to the near-bed shear velocity following Prandtl-Von Kármán's Law of the Wall: $\left(\frac{\kappa}{\ln z/z'}\right)^3$ in which z' [m] is the height at which the idealized velocity profile reaches zero and κ [-] is the Von Kármán constant.

The equilibrium sediment transport rate q_sat is divided by the wind velocity u_z to obtain a mass per unit area (per unit width):

$$c_\mathrm{sat} = \max\left(0 \quad ; \quad \alpha C \frac{\rho_\mathrm{a}}{g} \sqrt{\frac{d_n}{D_n}} \frac{(u_z - u_\mathrm{th})^3}{u_z}\right) \tag{1.4}$$

in which C [–] is an empirical constant to account for the grain size distribution width, ρ_a [$\mathrm{kg/m^3}$] is the air density, g [$\mathrm{m/s^2}$] is the gravitational constant, d_n [m] is the nominal grain size, D_n [m] is a reference grain size, u_z [m/s] is the wind velocity at height z [m] and α [–] is a constant to convert from measured wind velocity to shear velocity.

Note that at this stage the spatial variations in wind velocity are not solved for and hence no morphological feedback is included in the simulation. The model is initially intended to provide accurate sediment fluxes from the beach to the dunes rather than to simulate subsequent dune formation.

1.1.2 Multi-fraction Erosion and Deposition

The formulation for the equilibrium or saturated sediment concentration c_sat (Equation `equilibrium-transport`) is capable of dealing with variations in grain size through the variables u_th, d_n and C ([Bag37b]). However, the transport formulation only describes the saturated sediment concentration assuming a fixed grain size distribution, but does not define how multiple fractions coexist in transport. If the saturated sediment concentration formulation would be applied to each fraction separately and summed up to a total transport, the total sediment transport would increase with the number of sediment fractions. Since this is unrealistic behavior the saturated sediment concentration c_sat for the different fractions should be weighted in order to obtain a realistic total sediment transport. Equation (1.2) therefore is modified to include a weighting factor $\hat{w}_k$ in which k represents the sediment fraction index:

$$E_k - D_k = \min\left(\frac{\partial m_{\mathrm{a},k}}{\partial t} \quad ; \quad \frac{\hat{w}_k \cdot c_{\mathrm{sat},k} - c_k}{T}\right) \tag{1.5}$$

It is common to use the grain size distribution in the bed as weighting factor for the saturated sediment concentration (e.g. [Delft3DFManual14], section 11.6.4). Using the grain size distribution at the bed surface as a weighting factor assumes, in case of erosion, that all sediment at the bed surface is equally exposed to the wind.

Using the grain size distribution at the bed surface as weighting factor in case of deposition would lead to the behavior where deposition becomes dependent on the bed composition. Alternatively, in case of deposition, the saturated sediment concentration can be weighted based on the grain size distribution in the air. Due to the nature of saltation, in which continuous interaction with the bed forms the saltation cascade, both the grain size distribution in the bed and in the air are likely to contribute to the interaction between sediment fractions. The ratio between both contributions in the model is determined by a bed interaction parameter ζ.

The weighting of erosion and deposition of individual fractions is computed according to:

$$\hat{w}_k = \frac{w_k}{\sum_{k=1}^{n_\mathrm{k}} w_k} \tag{1.6}$$

$$\text{where} \quad w_k = (1 - \zeta) \cdot w_k^\mathrm{air} + (1 - \hat{S}_k) \cdot w_k^\mathrm{bed} \tag{1.6}$$

in which k represents the sediment fraction index, n_k the total number of sediment fractions, w_k is the unnormalized weighting factor for fraction k, $\hat{w}_k$ is its normalized counterpart, w_k^air and w_k^bed are the weighting factors based on the grain size distribution in the air and bed respectively and $\hat{S}_k$ is the effective sediment saturation of the air. The weighting factors based on the grain size distribution in the air and the bed are computed using mass ratios:

$$w_k^\mathrm{air} = \frac{c_k}{c_{\mathrm{sat},k}} \quad ; \quad w_k^\mathrm{bed} = \frac{m_{\mathrm{a},k}}{\sum_{k=1}^{n_\mathrm{k}} m_{\mathrm{a},k}} \tag{1.6}$$

The sum of the ratio w_k^{air} over the fractions denotes the degree of saturation of the air column for fraction k. The degree of saturation determines if erosion of a fraction may occur. Also in saturated situations erosion of a sediment fraction can occur due to an exchange of momentum between sediment fractions, which is represented by the bed interaction parameter ζ. The effective degree of saturation is therefore also influenced by the bed interaction parameter and defined as:

$$\hat{S}_k = \min\left(1 \quad ; \quad (1-\zeta)\cdot\sum_{k=1}^{n_\mathrm{k}} w_k^{\mathrm{air}}\right) \tag{1.7}$$

When the effective saturation is greater than or equal to unity the air is (over)saturated and no erosion will occur. The grain size distribution in the bed is consequently less relevant and the second term in Equation (1.6) is thus minimized and zero in case $\zeta = 0$. In case the effective saturation is less than unity erosion may occur and the grain size distribution of the bed also contributes to the weighting over the sediment fractions. The weighting factors for erosion are then composed from both the grain size distribution in the air and the grain size distribution at the bed surface. Finally, the resulting weighting factors are normalized to sum to unity over all fractions ($\hat{w}_k$).

The composition of weighting factors for erosion is based on the saturation of the air column. The non-saturated fraction determines the potential erosion of the bed. Therefore the non-saturated fraction can be used to scale the grain size distribution in the bed in order to combine it with the grain size distribution in the air according to Equation (1.6). The non-saturated fraction of the air column that can be used for scaling is therefore $1 - \hat{S}_k$.

For example, if bed interaction is disabled ($\zeta = 0$) and the air is 70% saturated, then the grain size distribution in the air contributes 70% to the weighting factors for erosion, while the grain size distribution in the bed contributes the other 30% (Figure Fig. 1.1, upper left panel). In case of (over)saturation the grain size distribution in transport contributes 100% to the weighting factors and the grain size distribution in the bed is of no influence. Transport progresses in downwind direction without interaction with the bed.

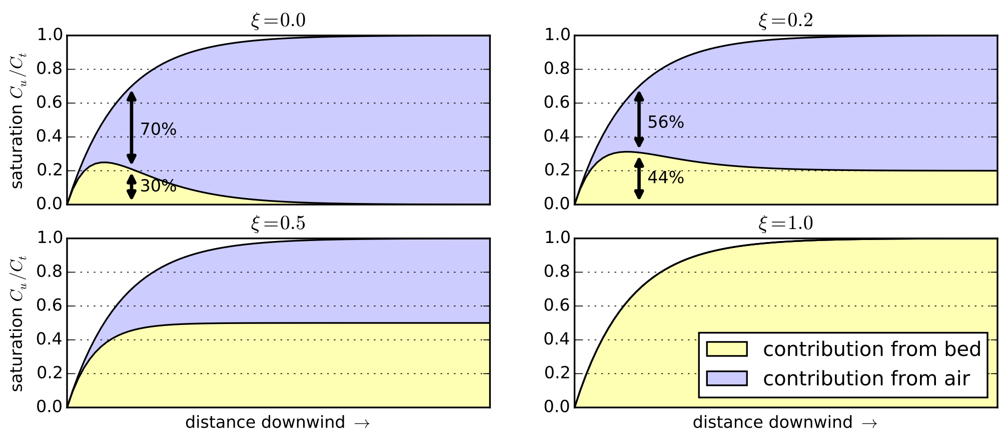

Fig. 1.1: Contributions of the grain size distribution in the bed and in the air to the weighting factors $\hat{w}_k$ for the equilibrium sediment concentration in Equation (1.5) for different values of the bed interaction parameter.

To allow for bed interaction in saturated situations in which no net erosion can occur, the bed interaction parameter ζ is used (Figure Fig. 1.1). The bed interaction parameter can take values between 0.0 and 1.0 in which the weighting factors for the equilibrium or saturated sediment concentration in an (over)saturated situation are fully determined by the grain size distribution in the bed or in the air respectively. A bed interaction value of 0.2 represents the situation in which the grain size distribution at the bed surface contributes 20% to the weighting of the saturated sediment concentration over the fractions. In the example situation where the air is 70% saturated such value for the bed interaction parameter would lead to weighting factors that are constituted for $70\% \cdot (100\% - 20\%) = 56\%$ based on the grain size distribution in the air and for the other 44% based on the grain size distribution at the bed surface (Figure Fig. 1.1, upper right panel).

The parameterization of the exchange of momentum between sediment fractions is an aspect of saltation that is still poorly understood. Therefore calibration of the bed interaction parameter ζ is necessary. The model parameters in

Equation `equilibrium-transport` can be chosen in accordance with the assumptions underlying multi-fraction sediment transport. C should be set to 1.5 as each individual sediment fraction is well-sorted, d_{n} should be chosen equal to D_{n} as the grain size dependency is implemented through u_{th}. u_{th} typically varies between 1 and 6 m/s for sand.

1.1.3 Simulation of Sediment Sorting and Beach Armoring

Since the equilibrium or saturated sediment concentration $c_{\mathrm{sat},k}$ is weighted over multiple sediment fractions in the extended advection model, also the instantaneous sediment concentration c_k is computed for each sediment fraction individually. Consequently, grain size distributions may vary over the model domain and in time. These variations are thereby not limited to the horizontal, but may also vary over the vertical since fine sediment may be deposited on top of coarse sediment or, reversely, fines may be eroded from the bed surface leaving coarse sediment to reside on top of the original mixed sediment. In order to allow the model to simulate the processes of sediment sorting and beach armoring the bed is discretized in horizontal grid cells and vertical bed layers (2DV; Figure Fig. 1.2).

The discretization of the bed consists of a minimum of three vertical bed layers with a constant thickness and an unlimited number of horizontal grid cells. The top layer is the *bed surface layer* and is the only layer that interacts with the wind and hence determines the spatiotemporal varying sediment availability and the contribution of the grain size distribution in the bed to the weighting of the saturated sediment concentration. One or more *bed composition layers* are located underneath the bed surface layer and form the upper part of the erodible bed. The bottom layer is the *base layer* and contains an infinite amount of erodible sediment according to the initial grain size distribution. The base layer cannot be eroded, but can supply sediment to the other layers.

Each layer in each grid cell describes a grain size distribution over a predefined number of sediment fractions (Figure Fig. 1.2, detail). Sediment may enter or leave a grid cell only through the bed surface layer. Since the velocity threshold depends among others on the grain size, erosion from the bed surface layer will not be uniform over all sediment fractions, but will tend to erode fines more easily than coarse sediment (Figure Fig. 1.2, detail, upper left panel). If sediment is eroded from the bed surface layer, the layer is repleted by sediment from the lower bed composition layers. The repleted sediment has a different grain size distribution than the sediment eroded from the bed surface layer. If more fines are removed from the bed surface layer in a grid cell than repleted, the median grain size increases. If erosion of fines continues the bed surface layer becomes increasingly coarse. Deposition of fines or erosion of coarse material may resume the erosion of fines from the bed.

In case of deposition the process is similar. Sediment is deposited in the bed surface layer that then passes its excess sediment to the lower bed layers (Figure Fig. 1.2, detail, upper right panel). If more fines are deposited than passed to the lower bed layers the bed surface layer becomes increasingly fine.

1.1.4 Simulation of the Emergence of Non-erodible Roughness Elements

Sediment sorting may lead to the emergence of non-erodible elements from the bed. Non-erodible roughness elements may shelter the erodible bed from wind erosion due to shear partitioning, resulting in a reduced sediment availability ([RGL93]). Therefore the equation of [RGL93] is implemented according to:

$$u_{*\mathrm{th,R}} = u_{*\mathrm{th}} \cdot \sqrt{\left(1 - m \cdot \sum_{k=k_0}^{n_{\mathrm{k}}} w_k^{\mathrm{bed}} \right) \left(1 + \frac{m \beta}{\sigma} \cdot \sum_{k=k_0}^{n_{\mathrm{k}}} w_k^{\mathrm{bed}} \right)} \tag{1.8}$$

in which σ is the ratio between the frontal area and the basal area of the roughness elements and β is the ratio between the drag coefficients of the roughness elements and the bed without roughness elements. m is a factor to account for the difference between the mean and maximum shear stress and is usually chosen 1.0 in wind tunnel experiments and may be lowered to 0.5 for field applications. The roughness density λ in the original equation of [RGL93] is obtained from the mass fraction in the bed surface layer w_k^{bed} according to:

$$\lambda = \frac{\sum_{k=k_0}^{n_{\mathrm{k}}} w_k^{\mathrm{bed}}}{\sigma} \tag{1.9}$$

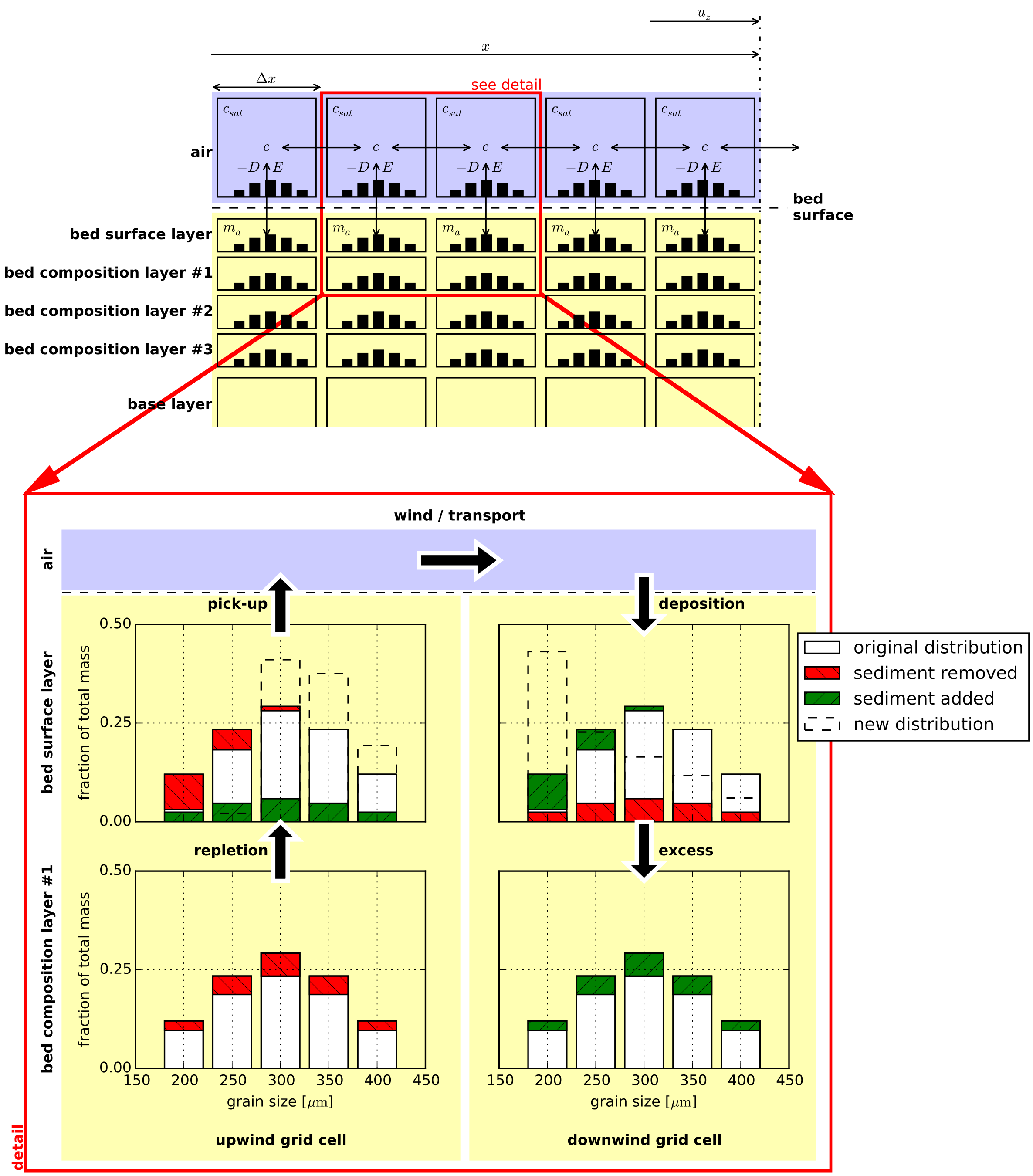

Fig. 1.2: Schematic of bed composition discretisation and advection scheme. Horizontal exchange of sediment may occur solely through the air that interacts with the *bed surface layer*. The detail presents the simulation of sorting and beach armoring where the bed surface layer in the upwind grid cell becomes coarser due to non-uniform erosion over the sediment fractions, while the bed surface layer in the downwind grid cell becomes finer due to non-uniform deposition over the sediment fractions. Symbols refer to Equations (1.1) and (1.2).

in which k_0 is the index of the smallest non-erodible sediment fraction in current conditions and n_{k} is the total number of sediment fractions. It is assumed that the sediment fractions are ordered by increasing size. Whether a fraction is erodible depends on the sediment transport capacity.

1.1.5 Simulation of the Hydraulic Mixing

As sediment sorting due to aeolian processes can lead to armoring of a beach surface, mixing of the beach surface or erosion of course material may undo the effects of armoring. To ensure a proper balance between processes that limit and enhance sediment availability in the model both types of processes need to be sufficiently represented when simulating spatiotemporal varying bed surface properties and sediment availability.

A typical upwind boundary in coastal environments during onshore winds is the water line. For aeolian sediment transport the water line is a zero-transport boundary. In the presence of tides, the intertidal beach is flooded periodically. Hydraulic processes like wave breaking mix the bed surface layer of the intertidal beach, break the beach armoring and thereby influence the availability of sediment.

In the model the mixing of sediment is simulated by averaging the sediment distribution over the depth of disturbance (Δz_{d}). The depth of disturbance is linearly related to the breaker height (e.g. [Kin51], [Wil71], [MAROHare07]). [MAROHare07] proposes an empirical factor $f_{\Delta z_{\mathrm{d}}}$ [-] that relates the depth of disturbance directly to the local breaker height according to:

$$\Delta z_{\mathrm{d}} = f_{\Delta z_{\mathrm{d}}} \cdot \min\left(H \quad ; \quad \gamma \cdot d\right) \tag{1.10}$$

in which the offshore wave height H [m] is taken as the local wave height maximized by a maximum wave height over depth ratio γ [-]. d [m] is the water depth that is provided to the model through an input time series of water levels. Typical values for $f_{\Delta z_{\mathrm{d}}}$ are 0.05 to 0.4 and 0.5 for γ.

1.1.6 Simulation of surface moisture

Wave runup, capillary rise from the beach groundwater, and precipitation periodically wet the intertidal beach temporally increasing the shear velocity threshold (Fig. 1.3). Infiltration and evaporation subsequently dry the beach.

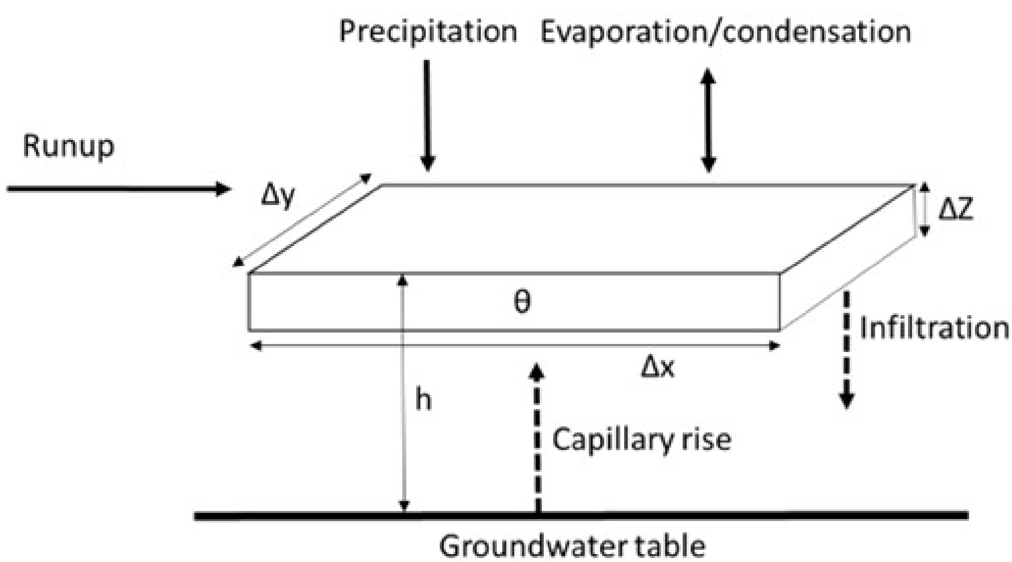

Fig. 1.3: Illustration of processes influencing the volumetric moisture content θ at the beach surface.

The structure of the surface moisture module and included processes are schematized in Fig. 1.4. The resulting surface moisture is obtained by selecting the largest of the moisture contents computed with the water balance approach (right column) and due to capillary rise from the groundwater table (left column). The method is based on the assumption that the flow of soil water is small compared to the flow of groundwater and that the beach groundwater dynamics primarily is controlled by the water level and wave action at the seaward boundary ([RGE99], [Sch14]). Thus, there is no feedback between the processes in the right column of Fig. 1.4 and the groundwater dynamics described in the left column.

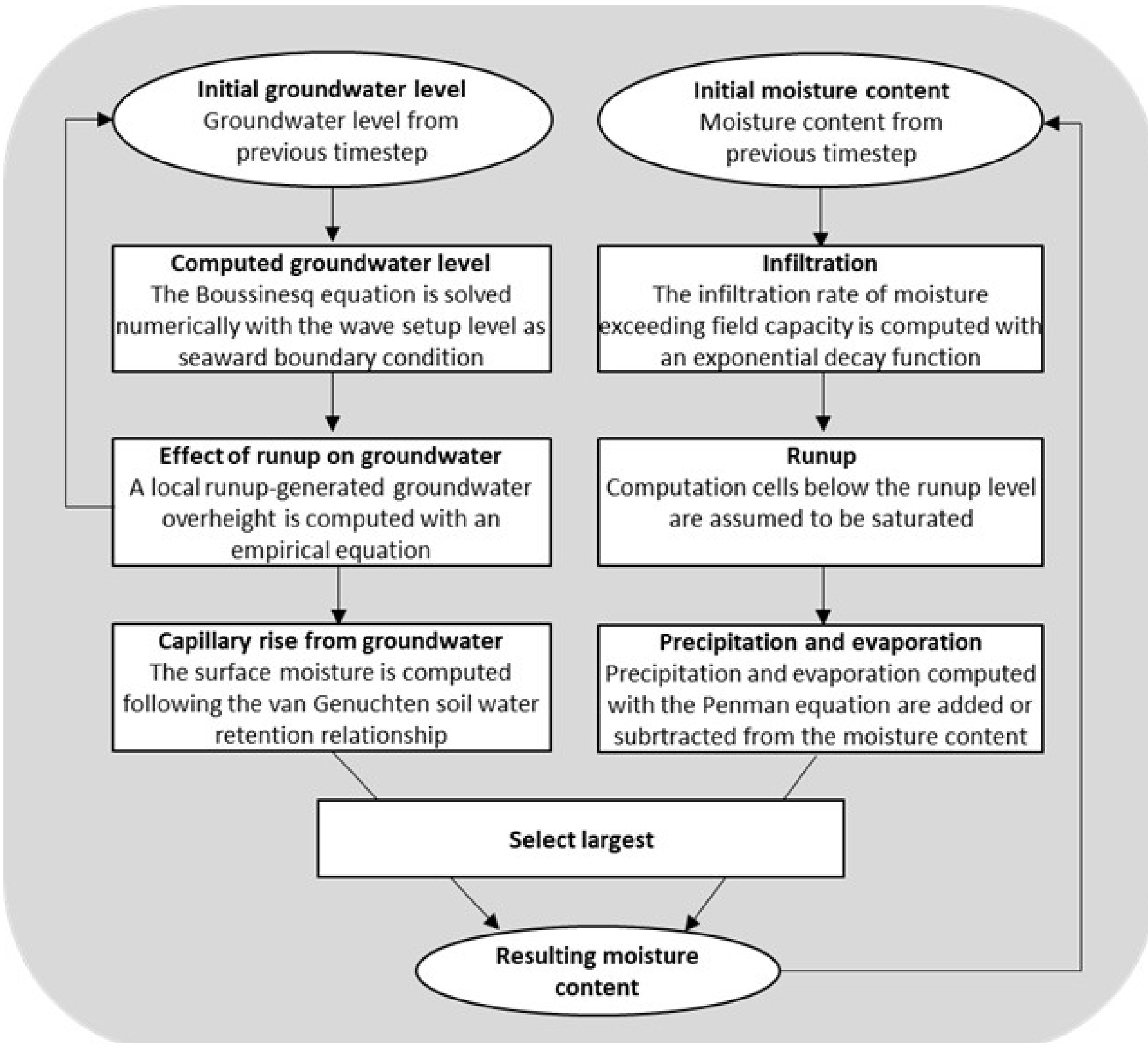

Fig. 1.4: Implementation of surface moisture processes in the AeoLiS.

Runup and wave setup

The runup height and wave setup are computed using the Stockdon formulas ([SHHS06]). Their parameterization differs depending on the dynamic beach steepness expressed through the Irribaren number:

$$\xi = \tan\beta / \sqrt{H_0/L_0} \tag{1.11}$$

where H_0 is the significant offshore wave height, L_0 is the deepwater wavelength, and $\tan\beta$ is the foreshore slope.

For dissipative conditions, ξ < 0.3, the runup, R_2, is parameterized as,

$$R_2 = 0.043\sqrt{H_0 L_0} \tag{1.12}$$

and wave setup:

$$< \eta >= 0.02\sqrt{H_0 L_0} \tag{1.13}$$

For ξ > 0.3, runup is paramterized as,

$$R_2 = 1.1\left(0.35\beta\sqrt{H_0 L_0} + \frac{\sqrt{H_0 L_0\left(0.563\beta^2 + 0.004\right)}}{2}\right) \tag{1.14}$$

and wave setup:

$$< \eta >= 0.35\xi \tag{1.15}$$

Tide- and wave-induced groundwater variations

Groundwater under sandy beaches can be considered as shallow aquifers, with only horizontal groundwater flow so that the pressure distribution is hydrostatic ([BMH98], [BSDR19], [Nie90], [RGE99]). The cross-shore flow dominates temporal variations of groundwater levels. Alongshore, groundwater table variations are typically small ([Sch14]). Although the surface moisture model can be extended over a two-dimensional grid, the groundwater simulations are performed for 1D transects cross-shore to avoid numerical instabilities at the seaward boundary and reduce computational time.

The beach aquifers is schematised as a sandy body, with saturated hydraulic conductivity, K, and effective porosity, n_e. The aquifer is assumed to rest on an impermeable surface, where D is the aquifer depth. The groundwater elevation relative to the mean sea level (MSL) is denoted η, and the shore-perpendicular x-axis is positive landwards, with an arbitrary starting point. The sand is assumed to be homogenous and isotropic. In this context, isotropy implies that hydraulic conductivity is independent of flow direction.

The horizontal groundwater discharge per unit area, u, is then governed by Darcy's law,

$$u = -K\frac{\partial \eta}{\partial x} \tag{1.16}$$

and the continuity equation (see e.g., [Nie09]),

$$\frac{\partial \eta}{\partial t} = -\frac{1}{n_e}\frac{\partial}{\partial x}((D+\eta)u) \tag{1.17}$$

where t is time.

The groundwater overheight due to runup, U_l, is computed by ([KNH94], [NDWE88]),

$$U_l = \begin{cases} C_l K f(x) & \text{if } x_S \leqslant x \leqslant x_R \\ 0, & \text{if } x > x_R \end{cases} \tag{1.18}$$

where C_l is an infiltration coefficient (-), and $f(x)$ is a function of x ranging from 0 to 1. x_S is the horizontal location of the sum of the still water level and wave setup, and x_R is the horizontal location of the runup limit:

$$f(x) = \begin{cases} \frac{x - x_s}{\frac{2}{3}(x_{ru} - x_s)} & if\, x_s < x \leqslant x_s + \frac{2}{3}(x_{ru} - x_s) \\ 3 - \frac{x - x_s}{\frac{1}{3}(x_{ru} - x_s)} & if\, x_s + \frac{2}{3}(x_{ru} - x_s) < x < x_{ru} \end{cases} \tag{1.19}$$

Substitution of u (Equation (1.16)) in the continuity equation (Equation (1.17)) with the addition of U_l/n_e gives the nonlinear Boussinesq equation:

$$\frac{\partial \eta}{\partial t} = \frac{K}{n_e}\frac{\partial}{\partial x}\left((D + \eta)\frac{\partial \eta}{\partial x}\right) + \frac{U_l}{n_e} \tag{1.20}$$

Capillary rise

Soil water retention (SWR) functions describe the surface moisture due to capillary transport of water from the groundwater table ([vG80]):

$$\theta(h) = \theta_r + \frac{\theta_s - \theta_r}{\left[1 + |\alpha h|^n\right]^m} \tag{1.21}$$

where h is the groundwater table depth, α and n are fitting parameters related to the air entry suction and the pore size distribution. The parameter m is commonly parameterised as $m = 1 - 1/n$.

The resulting surface moisture is computed for both drying and wetting conditions, i.e., including the effect of hysteresis. The moisture contents computed with drying and wetting SWR functions are denoted $\theta^d(h)$ and $\theta^w(h)$, respectively. When moving between wetting and drying conditions, the soil moisture content follows an intermediate retention curve called a scanning curve. The drying scanning curves are scaled from the main drying curve and wetting scanning curves from the main wetting curve. The drying scanning curve is then obtained from ([Mua74]):

$$\theta^d(h_\Delta, h) = \theta^w(h) + \frac{\left[\theta^w(h_\Delta) - \theta^w(h)\right]}{\left[\theta_s - \theta^w(h)\right]}\left[\theta^d(h) - \theta^w(h)\right] \tag{1.22}$$

where h_Δ is the groundwater table depth at the reversal on the wetting curve.

The wetting scanning curve is obtained from ([Mua74]):

$$\theta^w(h_\Delta, h) = \theta^w(h) + \frac{\left[\theta_s - \theta^w(h)\right]}{\left[\theta_s - \theta^w(h_\Delta)\right]}\left[\theta^d(h_\Delta) - \theta^w(h_\Delta)\right] \tag{1.23}$$

where h_Δ is the groundwater table depth at the reversal on the drying curve.

Infiltration

Infiltration is accounted for by assuming that excess water infiltrates until the moisture content reaches field capacity, $\theta_f c$. The moisture content at field capacity is the maximum amount of water that the unsaturated zone of soil can hold against the pull of gravity. For sandy soils, the matric potential at this soil moisture condition is around - 1/10 bar. In equilibrium, this potential would be exerted on the soil capillaries at the soil surface when the water table is about 100 cm below the soil surface, $\theta_{fc} = \theta^d(100)$.

Infiltration is represented by an exponential decay function that is governed by a drying time scale T_{dry}. Exploratory model runs of the unsaturated soil with the HYDRUS1D ([vSimrunekvSejnavG98]) hydrology model show that the increase of the volumetric water content to saturation is almost instantaneous with rising tide. The drying of the beach surface through infiltration shows an exponential decay. In order to capture this behavior the volumetric water content is implemented according to:

$$\frac{d\theta}{dt} = (\theta - \theta_{fc}) \left(e^{-\ln(2)\frac{dt}{T_{dry}}} \right) \tag{1.24}$$

An alternative formulation is used for simulations that does not account for ground water and SWR processes,

$$p_{\mathrm{V}}^{n+1} = \begin{cases} p & \text{if } \eta > z_{\mathrm{b}} \\ p_{\mathrm{V}}^{n} \cdot e^{\frac{\log(0.5)}{T_{\mathrm{dry}}} \cdot \Delta t^n} - E_{\mathrm{v}} \cdot \frac{\Delta t^n}{\Delta z} & \text{if } \eta \leq z_{\mathrm{b}} \end{cases} \tag{1.25}$$

where η [m+MSL] is the instantaneous water level, z_{b} [m+MSL] is the local bed elevation, p_{V}^{n} [-] is the volumetric water content in time step n, Δt^n [s] is the model time step and Δz is the bed composition layer thickness. T_{dry} [s] is the beach drying time scale, defined as the time in which the beach moisture content halves.

Precipitation and evaporation

A water balance approach accounts for the effect of precipitation and evaporation,

$$\frac{d\theta}{dt} = \frac{(P - E)}{\Delta z} \tag{1.26}$$

where P is the precipitation, E is the evaporation, and Δz is the thickness of the surface layer.

Evaporation is simulated using an adapted version of the Penman-Monteith equation ([Shu93]) that is governed by meteorological time series of solar radiation, temperature and humidity.

E_{v} [m/s] is the evaporation rate that is implemented through an adapted version of the Penman equation ([Shu93]):

$$E_{\mathrm{v}} = \frac{m_{\mathrm{v}} \cdot R_{\mathrm{n}} + 6.43 \cdot \gamma_{\mathrm{v}} \cdot (1 + 0.536 \cdot u_2) \cdot \delta e}{\lambda_{\mathrm{v}} \cdot (m_{\mathrm{v}} + \gamma_{\mathrm{v}})} \cdot 9 \cdot 10^7 \tag{1.27}$$

where m_{v} [kPa/K] is the slope of the saturation vapor pressure curve, R_{n} $[\mathrm{MJ/m^2/day}]$ is the net radiance, γ_{v} [kPa/K] is the psychrometric constant, u_2 [m/s] is the wind speed at 2 m above the bed, δe [kPa] is the vapor pressure deficit (related to the relative humidity) and λ_{v} [MJ/kg] is the latent heat vaporization. To obtain an evaporation rate in [m/s], the original formulation is multiplied by $9 \cdot 10^7$.

1.1.7 Shear velocity threshold

The shear velocity threshold represents the influence of bed surface properties in the saturated sediment transport equation. The shear velocity threshold is computed for each grid cell and sediment fraction separately based on local bed surface properties, like moisture, roughness elements and salt content. For each bed surface property supported by the model a factor is computed to increase the initial shear velocity threshold:

$$u_{*\mathrm{th}} = f_{u_{*\mathrm{th}},\mathrm{M}} \cdot f_{u_{*\mathrm{th}},\mathrm{R}} \cdot f_{u_{*\mathrm{th}},\mathrm{S}} \cdot u_{*\mathrm{th},0} \tag{1.28}$$

The initial shear velocity threshold $u_{*\mathrm{th},0}$ [m/s] is computed based on the grain size following [Bag37a]:

$$u_{*\mathrm{th},0} = A\sqrt{\frac{\rho_{\mathrm{p}} - \rho_{\mathrm{a}}}{\rho_{\mathrm{a}}} \cdot g \cdot d_{\mathrm{n}}} \tag{1.29}$$

where A [-] is an empirical constant, ρ_{p} $[\mathrm{kg/m^3}]$ is the grain density, ρ_{a} $[\mathrm{kg/m^3}]$ is the air density, g $[\mathrm{m/s^2}]$ is the gravitational constant and d_{n} [m] is the nominal grain size of the sediment fraction.

Moisture content

The shear velocity threshold is updated based on moisture content following [Bel64]:

$$f_{u_{*\mathrm{th}},\mathrm{M}} = \max(1 \quad ; \quad 1.8 + 0.6 \cdot \log(p_{\mathrm{g}})) \tag{1.30}$$

where $f_{u_{*\mathrm{th}},M}$ [-] is a factor in Equation (1.28), p_{g} [-] is the geotechnical mass content of water, which is the percentage of water compared to the dry mass. The geotechnical mass content relates to the volumetric water content p_{V} [-] according to:

$$: label : vol - water$$
$$p_{\mathrm{g}} = \frac{p_{\mathrm{V}} \cdot \rho_{\mathrm{w}}}{\rho_{\mathrm{p}} \cdot (1 - p)}$$

where ρ_{w} $[\mathrm{kg/m^3}]$ and ρ_{p} $[\mathrm{kg/m^3}]$ are the water and particle density respectively and p [-] is the porosity. Values for p_{g} smaller than 0.005 do not affect the shear velocity threshold ([PT90]). Values larger than 0.064 (or 10% volumetric content) cease transport ([DF10]), which is implemented as an infinite shear velocity threshold.

Roughness elements

The shear velocity threshold is updated based on the presence of roughness elements following [RGL93]:

$$: label : shear - rough$$
$$f_{u_{*\mathrm{th}},R} = \sqrt{(1 - m \cdot \sum_{k=k_0}^{n_k} \hat{w}_k^{\mathrm{bed}})(1 + \frac{m\beta}{\sigma} \cdot \sum_{k=k_0}^{n_k} \hat{w}_k^{\mathrm{bed}})}$$

by assuming:

$$: label : lambda - rough$$
$$\lambda = \frac{\sum_{k=k_0}^{n_k} \hat{w}_k^{\mathrm{bed}}}{\sigma}$$

where $f_{u_{*\mathrm{th}},R}$ [-] is a factor in Equation (1.28), k_0 is the sediment fraction index of the smallest non-erodible fraction in current conditions and n_k is the number of sediment fractions defined. The implementation is discussed in detail in section ref{sec:roughness}.

Salt content

The shear velocity threshold is updated based on salt content following [NE81]:

$$f_{u_{*\mathrm{th}},S} = 1.03 \cdot \exp(0.1027 \cdot p_{\mathrm{s}}) \tag{1.31}$$

where $f_{u_{*\mathrm{th}},S}$ [-] is a factor in Equation (1.28) and p_{s} [-] is the salt content [mg/g]. Currently, no model is implemented that predicts the instantaneous salt content. The spatial varying salt content needs to be specified by the user, for example through the BMI interface.

Bibliography

1.2 Numerical implementation

The numerical implementation of the equations presented in *Model description* is explained here. The implementation is available as Python package through the OpenEarth GitHub repository at: http://www.github.com/openearth/aeolis-python/

1.2.1 Advection equation

The advection equation is implemented in two-dimensional form following:

$$\frac{\partial c}{\partial t} + u_{z,\mathrm{x}} \frac{\partial c}{\partial x} + u_{z,\mathrm{y}} \frac{\partial c}{\partial y} = \frac{c_{\mathrm{sat}} - c}{T} \tag{1.32}$$

in which c $[\mathrm{kg/m^2}]$ is the sediment mass per unit area in the air, c_{sat} $[\mathrm{kg/m^2}]$ is the maximum sediment mass in the air that is reached in case of saturation, $u_{z,\mathrm{x}}$ and $u_{z,\mathrm{y}}$ are the x- and y-component of the wind velocity at height z [m], T [s] is an adaptation time scale, t [s] denotes time and x [m] and y [m] denote cross-shore and alongshore distances respectively.

The formulation is discretized in different ways to allow for different types of simulations balancing accuracy vs. computational resources. The conservative method combined with an euler backward scheme (written by Prof. Rauwoens) is the current default for most simulations. Non-conservative methods end explicit Euler forward schemes are also available.

Default scheme – Conservative Euler Backward Implicit

The default numerical method assumes the advection scheme in a conservative form in combination with an euler backward scheme. This scheme is prepared to use a TVD method but this is not implemented yet (add footnote{Total Variance Diminishing, this is explained in the lecture notes by Zijlema p94})

The fluxes at the interface of the cells are defined used in the advection terms:

$$\begin{aligned} &\frac{c^{n+1}_{i,j,k} - c^{n}_{i,j,k}}{\Delta t} + \\ &\frac{u_{\mathrm{x},i+1/2,j} \cdot c^{n+1}_{i+1/2,j,k} - u_{\mathrm{x},i-1/2,j} \cdot c^{n+1}_{i-1/2,j,k}}{\Delta x} + \\ &\frac{u_{\mathrm{y},i,j+1/2} \cdot c^{n+1}_{i,j+1/2,k} - u_{\mathrm{y},i,j-1/2} \cdot c^{n+1}_{i,j-1/2,k}}{\Delta y} + \\ &= \\ &\frac{\min(\hat{w}^{n+1}_{i,j,k} \cdot c^{n+1}_{\mathrm{sat},i,j,k}, m_{i.j.k} + c^{n+1}_{i,j,k}) - c^{n+1}_{i,j,k}}{T} \end{aligned} \tag{1.33}$$

In which n is the time step index, i and j are the cross-shore and alongshore spatial grid cell indices and k is the grain size fraction index. w [-] is the weighting factor used for the weighted addition of the saturated sediment concentrations over all grain size fractions. Note that u is spatially varying but has no temporal index. This is because u is a result of a separate wind solver and considered temporally invariant in the advection solver.

Now we use a correction algorithm where:

$$c^{n+1}_{i,j,k} = c^{n+1*}_{i,j,k} + \delta c_{i,j,k} \tag{1.34}$$

where $\delta c_{i,j,k}$ is solved for and $*$ denotes the previous iteration.

When now assuming an upwind scheme in space, we can derive 4 concentrations at the cell faces which are dependent on the velocity at the cell faces.

We assume in x direction:

$$c^{n+1}_{i+1/2,j,k} = \begin{cases} c^{n+1*}_{i,j,k} + \delta c_{i,j,k} & \text{if } u_{\text{x},i+1/2,j} > 0, \\ c^{n+1*}_{i+1,j,k} + \delta c_{i+1,j,k} & \text{if } u_{\text{x},i+1/2,j} < 0. \end{cases}$$

$$c^{n+1}_{i-1/2,j,k} = \begin{cases} c^{n+1*}_{i-1,j,k} + \delta c_{i-1,j,k} & \text{if } u_{\text{x},i-1/2,j} > 0, \\ c^{n+1*}_{i,j,k} + \delta c_{i,j,k} & \text{if } u_{\text{x},i-1/2,j} < 0. \end{cases}$$

and in y-direction:

$$c^{n+1}_{i,j+1/2,k} = \begin{cases} c^{n+1*}_{i,j,k} + \delta c_{i,j,k} & \text{if } u_{\text{y},i,j+1/2} > 0, \\ c^{n+1*}_{i,j+1,k} + \delta c_{i,j+1,k} & \text{if } u_{\text{y},i,j+1/2} < 0. \end{cases}$$

$$c^{n+1}_{i,j-1/2,k} = \begin{cases} c^{n+1*}_{i,j-1,k} + \delta c_{i,j-1,k} & \text{if } u_{\text{y},i,j-1/2} > 0, \\ c^{n+1*}_{i,j,k} + \delta c_{i,j,k} & \text{if } u_{\text{y},i,j-1/2} < 0. \end{cases}$$

Now we assume:

- $\Gamma_x = 1$ if $u_{\text{x},i+1/2,j,k} > 0$ and $\Gamma_x = 0$ if $u_{\text{x},i+1/2,j,k} \leq 0$
- $\Gamma_y = 1$ if $u_{\text{y},i,j+1/2,k} > 0$ and $\Gamma_x = 0$ if $u_{\text{y},i,j+1/2,k} \leq 0$

(We did not test if this works well with diverging and converging flows. We may need another term that describes the conditions at the negative cell faces if they are of opposite direction than the positive cell faces and vice versa)

Let's continue for the moment so that

$$\begin{gathered} \frac{c^{n+1*}_{i,j,k} + \delta c_{i,j,k} - c^{n}_{i,j,k}}{\Delta t} + \\ \Gamma_x \cdot \frac{u_{\text{x},i+1/2,j} \cdot (c^{n+1*}_{i,j,k} + \delta c_{i,j,k}) - u_{\text{x},i-1/2,j} \cdot (c^{n+1*}_{i-1,j,k} + \delta c_{i-1,j,k})}{\Delta x} + \\ (1 - \Gamma_x) \cdot \frac{u_{\text{x},i+1/2,j} \cdot (c^{n+1*}_{i+1,j,k} + \delta c_{i+1,j,k}) - u_{\text{x},i-1/2,j} \cdot (c^{n+1*}_{i,j,k} + \delta c_{i,j,k})}{\Delta x} + \\ \Gamma_y \cdot \frac{u_{\text{y},i,j+1/2} \cdot (c^{n+1*}_{i,j,k} + \delta c_{i,j,k}) - u_{\text{y},i,j-1/2} \cdot (c^{n+1*}_{i,j-1,k} + \delta c_{i,j-1,k})}{\Delta y} + \\ (1 - \Gamma_y) \cdot \frac{u_{\text{y},i,j+1/2} \cdot (c^{n+1*}_{i,j+1,k} + \delta c_{i,j+1,k}) - u_{\text{y},i,j-1/2} \cdot (c^{n+1*}_{i,j,k} + \delta c_{i,j,k})}{\Delta y} + \\ = \\ \frac{\min(\hat{w}^{n+1}_{i,j,k} \cdot c^{n+1}_{\text{sat},i,j,k}, m_{i,j,k} + c^{n+1*}_{i,j,k} + \delta c_{i,j,k}) - c^{n+1*}_{i,j,k} + \delta c_{i,j,k}}{T} \end{gathered}$$

(note that the above does not take converging and diverging flows into account, also $\delta c_{i,j,k}$ at the right hand side in the "min" brackets is difficult to solve for. In the code, this term is neglected which may cause some inaccuracy when calculating pickup. Although mass continuity is corrected for in the implicit scheme when calculating pickup using equation ???)

Now we simplify:

$$\begin{aligned}(\frac{\Delta x \Delta y}{\Delta t} + \Gamma_x \Delta y \cdot u_{\mathrm{x},i+1/2,j} - (1-\Gamma_x)\Delta y \cdot u_{\mathrm{x},i-1/2,j} + \Gamma_y \Delta x \cdot u_{\mathrm{y},i,j+1/2} \\ -(1-\Gamma_y)\Delta x \cdot u_{\mathrm{y},i,j-1/2} + \frac{\Delta x \Delta y}{T_s}) \cdot \delta c_{i,j,k} \\ -(\Gamma_x \Delta y \cdot u_{\mathrm{x},i-1/2,j}) \cdot \delta c_{i-1,j,k} \\ +((1-\Gamma_x)\Delta y \cdot u_{\mathrm{x},i+1/2,j}) \cdot \delta c_{i+1,j,k} \\ -(\Gamma_y \Delta x \cdot u_{\mathrm{y},i,j-1/2}) \cdot \delta c_{i,j-1,k} \\ +((1-\Gamma_y)\Delta x \cdot u_{\mathrm{y},i,j+1/2}) \cdot \delta c_{i,j+1,k}\end{aligned}$$

or

$$\begin{aligned}A0 \cdot \delta c_{i,j,k} + A\mathrm{m1} \cdot \delta c_{i-1,j,k} + A\mathrm{p1} \cdot \delta c_{i+1,j,k} \\ +A\mathrm{mx} \cdot \delta c_{i,j-1,k} + A\mathrm{px} \cdot \delta c_{i,j+1,k} = y_{i,j,k}\end{aligned}$$

or the linear system of equations in general form:

$$A \cdot \delta c_{i,j,k} = y_{i,j,k} \tag{1.35}$$

Where A is a 3-dimensional sparse matrix that is compiled using the matrix diagonals ($A0$, $Am1$, $Ap1$, Amx, Apx) which are defined as:

$$\begin{aligned}A0 = &+ \frac{\Delta x \Delta y}{\Delta t} \\ &+ \frac{\Delta x \Delta y}{T_s} \\ &- (1-\Gamma_x)\Delta y \cdot u_{\mathrm{x},i-1/2,j} \\ &+ \Gamma_x \Delta y \cdot u_{\mathrm{x},i+1/2,j} \\ &- (1-\Gamma_y)\Delta x \cdot u_{\mathrm{y},i,j-1/2} \\ &+ \Gamma_y \Delta x \cdot u_{\mathrm{y},i,j+1/2}\end{aligned}$$

and

$$A\mathrm{m1} = -\Gamma_x \Delta y \cdot u_{\mathrm{x},i-1/2,j}$$

and

$$A\mathrm{p1} = (1-\Gamma_x)\Delta y \cdot u_{\mathrm{x},i+1/2,j}$$

and

$$A\mathrm{mx} = -\Gamma_y \Delta x \cdot u_{\mathrm{y},i,j-1/2}$$

and

$$A\mathrm{px} = (1-\Gamma_y)\Delta x \cdot u_{\mathrm{y},i,j+1/2}$$

Let’s go towards the RHS

$$\begin{aligned}y_{i,j,k} = &- \frac{\Delta x \Delta y}{\Delta t}(c^{n+1*}_{i,j,k} - c^{n}_{i,j,k}) \\ &+ \frac{\Delta x \Delta y}{T_s}(\min(\hat{w}^{n+1}_{i,j,k} \cdot c^{n+1}_{\mathrm{sat},i,j,k}, m_{i,j,k} + c^{n+1*}_{i,j,k}) - c^{n+1*}_{i,j,k}) \\ &+ \Delta y \cdot u_{\mathrm{x},i-1/2,j} \cdot (\Gamma_x \cdot c^{n+1*}_{i-1,j,k} + (1-\Gamma_x)c^{n+1*}_{i,j,k}) \\ &- \Delta y \cdot u_{\mathrm{x},i+1/2,j} \cdot (\Gamma_x \cdot c^{n+1*}_{i,j,k} + (1-\Gamma_x)c^{n+1*}_{i+1,j,k}) \\ &+ \Delta x \cdot u_{\mathrm{y},i,j-1/2} \cdot (\Gamma_y \cdot c^{n+1*}_{i,j-1,k} + (1-\Gamma_y)c^{n+1*}_{i,j,k}) \\ &- \Delta x \cdot u_{\mathrm{y},i,j+1/2} \cdot (\Gamma_y \cdot c^{n+1*}_{i,j,k} + (1-\Gamma_y)c^{n+1*}_{i,j+1,k})\end{aligned}$$

in the python code some intermediate variable is defined to make it easier to shift indexes

$$\text{Ctxfs} = (\Gamma_x \cdot c^{n+1*}_{i,j,k} + (1 - \Gamma_x) c^{n+1*}_{i+1,j,k})$$

and

$$\text{Ctxfn} = (\Gamma_y \cdot c^{n+1*}_{i,j,k} + (1 - \Gamma_y) c^{n+1*}_{i,j+1,k})$$

also Erosion and deposition are defined using seperate variables.

$$D_{i,j,k} = \frac{\Delta x \Delta y}{T_s} c^{n+1*}_{i,j,k}$$

and

$$A_{i,j,k} = \frac{\Delta x \Delta y}{T_s} m_{i,j,k} + D_{i,j,k}$$

and

$$U_{i,j,k} = \frac{\Delta x \Delta y}{T_s} \hat{w}^{n+1}_{i,j,k} \cdot c^{n+1}_{\mathrm{sat},i,j,k}$$

and

$$E_{i,j,k} = \min(U_{i,j,k}, A_{i,j,k})$$

After solving equation $\delta c_{i,j,k}$ using (1.35), $c^{n+1}_{i,j,k}$ can be calculated using equation (1.34).

Also, the pickup per grid cell can be calculated using:

$$\text{pickup} = \frac{\hat{w}^{n+1}_{i,j,k} \cdot c^{n+1}_{\mathrm{sat},i,j,k} - c^{n+1}_{i,j,k}}{T_s} \Delta t$$

note that this is only valid when using an Euler backward scheme.

Solving the Linear System of Equations

The linear system of equations can be elaborated :

$$\begin{bmatrix}
A^0_1 & A^1_1 & \mathbf{0} & \cdots & \mathbf{0} & A^{n_\mathrm{y}+1}_1 \\
A^{-1}_2 & A^0_2 & \ddots & \ddots & & \mathbf{0} \\
\mathbf{0} & \ddots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \ddots & \mathbf{0} \\
\mathbf{0} & & \ddots & \ddots & A^0_{n_\mathrm{y}} & A^1_{n_\mathrm{y}} \\
A^{-n_\mathrm{y}-1}_{n_\mathrm{y}+1} & \mathbf{0} & \cdots & \mathbf{0} & A^{-1}_{n_\mathrm{y}+1} & A^0_{n_\mathrm{y}+1}
\end{bmatrix}
\begin{bmatrix}
\vec{\delta c}_1 \\ \vec{\delta c}_2 \\ \vdots \\ \vdots \\ \vec{\delta c}_{n_\mathrm{y}} \\ \vec{\delta c}_{n_\mathrm{y}+1}
\end{bmatrix}
=
\begin{bmatrix}
\vec{y}_1 \\ \vec{y}_2 \\ \vdots \\ \vdots \\ \vec{y}_{n_\mathrm{y}} \\ \vec{y}_{n_\mathrm{y}+1}
\end{bmatrix} \tag{1.36}$$

where each item in the matrix is again a matrix A^l_j and each item in the vectors is again a vector $\vec{\delta c}_j$ and $\vec{y}_j$ respectively. The form of the matrix A^l_j depends on the diagonal index l and reads:

$$A^0_j = \begin{bmatrix}
0 & 0 & 0 & 0 & \cdots & \cdots & 0 \\
a^{0,-1}_{2,j} & a^{0,0}_{2,j} & a^{0,1}_{2,j} & \ddots & & & \vdots \\
0 & a^{0,-1}_{3,j} & a^{0,0}_{3,j} & a^{0,1}_{3,j} & \ddots & & \vdots \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
\vdots & & \ddots & a^{0,-1}_{n_\mathrm{x}-1,j} & a^{0,0}_{n_\mathrm{x}-1,j} & a^{0,1}_{n_\mathrm{x}-1,j} & 0 \\
\vdots & & & 0 & a^{0,-1}_{n_\mathrm{x},j} & a^{0,0}_{n_\mathrm{x},j} & a^{0,1}_{n_\mathrm{x},j} \\
0 & \cdots & \cdots & 0 & 1 & -2 & 1
\end{bmatrix} \tag{1.37}$$

for $l = 0$ and

$$A_j^l = \begin{bmatrix} 1 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & a_{2,j}^{l,0} & \ddots & & & & \vdots \\ \vdots & \ddots & a_{3,j}^{l,0} & \ddots & & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & \ddots & a_{n_{\mathrm{x}}-1,j}^{l,0} & \ddots & \vdots \\ \vdots & & & & \ddots & a_{n_{\mathrm{x}},j}^{l,0} & 0 \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & 1 \end{bmatrix} \tag{1.38}$$

for $l \neq 0$. The vectors $\vec{\delta c}_{j,k}$ and $\vec{y}_{j,k}$ read:

$$\vec{\delta c}_{j,k} = \begin{bmatrix} \delta c_{1,j,k}^{n+1} \\ \delta c_{2,j,k}^{n+1} \\ \delta c_{3,j,k}^{n+1} \\ \vdots \\ \delta c_{n_{\mathrm{x}}-1,j,k}^{n+1} \\ \delta c_{n_{\mathrm{x}},j,k}^{n+1} \\ \delta c_{n_{\mathrm{x}}+1,j,k}^{n+1} \end{bmatrix} \text{ and } \vec{y}_{j,k} = \begin{bmatrix} 0 \\ y_{2,j,k}^{n} \\ y_{3,j,k}^{n} \\ \vdots \\ y_{n_{\mathrm{x}}-1,j,k}^{n} \\ y_{n_{\mathrm{x}},j,k}^{n} \\ 0 \end{bmatrix} \tag{1.39}$$

n_{x} and n_{y} denote the number of spatial grid cells in x- and y-direction.

Iterations to solve for multiple fractions

The linear system defined in Equation (1.36) is solved by a sparse matrix solver for each sediment fraction separately in ascending order of grain size. Initially, the weights $\hat{w}_{i,j,k}^{n+1}$ are chosen according to the grain size distribution in the bed and the air. The sediment availability constraint is checked after each solve:

$$m_{\mathrm{a}} \geq \frac{\hat{w}_{i,j,k}^{n+1} c_{\mathrm{sat},i,j,k}^{n+1} - c_{i,j,k}^{n+1}}{T} \Delta t^n \tag{1.40}$$

If the constraint if violated, a new estimate for the weights is back-calculated following:

$$\hat{w}_{i,j,k}^{n+1} = \frac{c_{i,j,k}^{n+1} + m_{\mathrm{a}} \frac{T}{\Delta t^n}}{c_{\mathrm{sat},i,j,k}^{n+1}} \tag{1.41}$$

The system is solved again using the new weights. This procedure is repeated until a weight is found that does not violate the sediment availability constraint. If the time step is not too large, the procedure typically converges in only a few iterations. Finally, the weights of the larger grains are increased proportionally as to ensure that the sum of all weights remains unity. If no larger grains are defined, not enough sediment is available for transport and the grid cell is truly availability-limited. This situation should only occur occasionally as the weights in the next time step are computed based on the new bed composition and thus will be skewed towards the large fractions. If the situation occurs regularly, the time step is chosen too large compared to the rate of armoring.

Euler Schemes in non-conservative form

Early model results relied on Euler schemes in a non conservative form. This allowed for a relatively easy implementation but did not guarantee mass conservation. In version 2 of AEOLIS the conservative form became the default. However, some users still use the older scheme.

The formulation is discretized following a first order upwind scheme assuming that the wind velocity u_z is positive in both x-direction and y-direction:

$$\frac{c^{n+1}_{i,j,k} - c^{n}_{i,j,k}}{\Delta t^n} + u^n_{z,\mathrm{x}} \frac{c^{n}_{i+1,j,k} - c^{n}_{i,j,k}}{\Delta x_{i,j}} + u^n_{z,\mathrm{y}} \frac{c^{n}_{i,j+1,k} - c^{n}_{i,j,k}}{\Delta y_{i,j}} = \frac{\hat{w}^{n}_{i,j,k} \cdot c^{n}_{\mathrm{sat},i,j,k} - c^{n}_{i,j,k}}{T} \tag{1.42}$$

in which n is the time step index, i and j are the cross-shore and alongshore spatial grid cell indices and k is the grain size fraction index. w [-] is the weighting factor used for the weighted addition of the saturated sediment concentrations over all grain size fractions.

The discretization can be generalized for any wind direction as:

$$\frac{c^{n+1}_{i,j,k} - c^{n}_{i,j,k}}{\Delta t^n} + u^n_{z,\mathrm{x+}} c^{n}_{i,j,k,\mathrm{x+}} + u^n_{z,\mathrm{y+}} c^{n}_{i,j,k,\mathrm{y+}} + u^n_{z,\mathrm{x-}} c^{n}_{i,j,k,\mathrm{x-}} + u^n_{z,\mathrm{y-}} c^{n}_{i,j,k,\mathrm{y-}} = \frac{\hat{w}^{n}_{i,j,k} \cdot c^{n}_{\mathrm{sat},i,j,k} - c^{n}_{i,j,k}}{T} \tag{1.43}$$

in which:

$$\begin{array}{rclcrcl} u^n_{z,\mathrm{x+}} & = & \max(0, u^n_{z,\mathrm{x}}) & ; & u^n_{z,\mathrm{y+}} & = & \max(0, u^n_{z,\mathrm{y}}) \\ u^n_{z,\mathrm{x-}} & = & \min(0, u^n_{z,\mathrm{x}}) & ; & u^n_{z,\mathrm{y-}} & = & \min(0, u^n_{z,\mathrm{y}}) \end{array} \tag{1.44}$$

and

$$\begin{array}{rclcrcl} c^n_{i,j,k,\mathrm{x+}} & = & \frac{c^n_{i+1,j,k} - c^n_{i,j,k}}{\Delta x} & ; & c^n_{i,j,k,\mathrm{y+}} & = & \frac{c^n_{i,j+1,k} - c^n_{i,j,k}}{\Delta y} \\ c^n_{i,j,k,\mathrm{x-}} & = & \frac{c^n_{i,j,k} - c^n_{i-1,j,k}}{\Delta x} & ; & c^n_{i,j,k,\mathrm{y-}} & = & \frac{c^n_{i,j,k} - c^n_{i,j-1,k}}{\Delta y} \end{array} \tag{1.45}$$

Equation (1.43) is explicit in time and adheres to the Courant-Friedrich-Lewis (CFL) condition for numerical stability. Alternatively, the advection equation can be discretized implicitly in time for unconditional stability:

$$\frac{c^{n+1}_{i,j,k} - c^{n}_{i,j,k}}{\Delta t^n} + u^{n+1}_{z,\mathrm{x+}} c^{n+1}_{i,j,k,\mathrm{x+}} + u^{n+1}_{z,\mathrm{y+}} c^{n+1}_{i,j,k,\mathrm{y+}} + u^{n+1}_{z,\mathrm{x-}} c^{n+1}_{i,j,k,\mathrm{x-}} + u^{n+1}_{z,\mathrm{y-}} c^{n+1}_{i,j,k,\mathrm{y-}} = \frac{\hat{w}^{n+1}_{i,j,k} \cdot c^{n+1}_{\mathrm{sat},i,j,k} - c^{n+1}_{i,j,k}}{T} \tag{1.46}$$

Equation (1.43) and :eq:apx-implicit-generalized` can be rewritten as:

$$c^{n+1}_{i,j,k} = c^{n}_{i,j,k} - \Delta t^n \left[u^n_{z,\mathrm{x+}} c^{n}_{i,j,k,\mathrm{x+}} + u^n_{z,\mathrm{y+}} c^{n}_{i,j,k,\mathrm{y+}} + u^n_{z,\mathrm{x-}} c^{n}_{i,j,k,\mathrm{x-}} + u^n_{z,\mathrm{y-}} c^{n}_{i,j,k,\mathrm{y-}} + \frac{\hat{w}^{n}_{i,j,k} \cdot c^{n}_{\mathrm{sat},i,j,k} - c^{n}_{i,j,k}}{T} \right] \tag{1.47}$$

and

$$c^{n+1}_{i,j,k} + \Delta t^n \left[u^{n+1}_{z,\mathrm{x+}} c^{n+1}_{i,j,k,\mathrm{x+}} + u^{n+1}_{z,\mathrm{y+}} c^{n+1}_{i,j,k,\mathrm{y+}} + u^{n+1}_{z,\mathrm{x-}} c^{n+1}_{i,j,k,\mathrm{x-}} + u^{n+1}_{z,\mathrm{y-}} c^{n+1}_{i,j,k,\mathrm{y-}} + \frac{\hat{w}^{n+1}_{i,j,k} \cdot c^{n+1}_{\mathrm{sat},i,j,k} - c^{n+1}_{i,j,k}}{T} \right] = c^{n}_{i,j,k} \tag{1.48}$$

and combined using a weighted average:

$$\begin{aligned}
& c^{n+1}_{i,j,k} + \Gamma \Delta t^n \left[u^{n+1}_{z,\mathrm{x+}} c^{n+1}_{i,j,k,\mathrm{x+}} + u^{n+1}_{z,\mathrm{y+}} c^{n+1}_{i,j,k,\mathrm{y+}} \right. \\
& \left. + u^{n+1}_{z,\mathrm{x-}} c^{n+1}_{i,j,k,\mathrm{x-}} + u^{n+1}_{z,\mathrm{y-}} c^{n+1}_{i,j,k,\mathrm{y-}} + \frac{\hat{w}^{n+1}_{i,j,k} \cdot c^{n+1}_{\mathrm{sat},i,j,k} - c^{n+1}_{i,j,k}}{T} \right] \\
& = c^n_{i,j,k} - (1-\Gamma) \Delta t^n \left[u^n_{z,\mathrm{x+}} c^n_{i,j,k,\mathrm{x+}} + u^n_{z,\mathrm{y+}} c^n_{i,j,k,\mathrm{y+}} \right. \\
& \left. + u^n_{z,\mathrm{x-}} c^n_{i,j,k,\mathrm{x-}} + u^n_{z,\mathrm{y-}} c^n_{i,j,k,\mathrm{y-}} + \frac{\hat{w}^n_{i,j,k} \cdot c^n_{\mathrm{sat},i,j,k} - c^n_{i,j,k}}{T} \right]
\end{aligned} \tag{1.49}$$

in which Γ is a weight that ranges from 0 – 1 and determines the implicitness of the scheme. The scheme is implicit with $\Gamma = 0$, explicit with $\Gamma = 1$ and semi-implicit otherwise. $\Gamma = 0.5$ results in the semi-implicit Crank-Nicolson scheme.

Equation (1.45) is back-substituted in Equation (1.49):

$$\begin{aligned}
& c^{n+1}_{i,j,k} + \Gamma \Delta t^n \left[u^{n+1}_{z,\mathrm{x+}} \frac{c^{n+1}_{i+1,j,k} - c^{n+1}_{i,j,k}}{\Delta x} + u^{n+1}_{z,\mathrm{y+}} \frac{c^{n+1}_{i,j+1,k} - c^{n+1}_{i,j,k}}{\Delta y} \right. \\
& \left. + u^{n+1}_{z,\mathrm{x-}} \frac{c^{n+1}_{i,j,k} - c^{n+1}_{i-1,j,k}}{\Delta x} + u^{n+1}_{z,\mathrm{y-}} \frac{c^{n+1}_{i,j,k} - c^{n+1}_{i,j-1,k}}{\Delta y} + \frac{\hat{w}^{n+1}_{i,j,k} \cdot c^{n+1}_{\mathrm{sat},i,j,k} - c^{n+1}_{i,j,k}}{T} \right] \\
& = c^n_{i,j,k} - (1-\Gamma) \Delta t^n \left[u^n_{z,\mathrm{x+}} \frac{c^n_{i+1,j,k} - c^n_{i,j,k}}{\Delta x} + u^n_{z,\mathrm{y+}} \frac{c^n_{i,j+1,k} - c^n_{i,j,k}}{\Delta y} \right. \\
& \left. + u^n_{z,\mathrm{x-}} \frac{c^n_{i,j,k} - c^n_{i-1,j,k}}{\Delta x} + u^n_{z,\mathrm{y-}} \frac{c^n_{i,j,k} - c^n_{i,j-1,k}}{\Delta y} + \frac{\hat{w}^n_{i,j,k} \cdot c^n_{\mathrm{sat},i,j,k} - c^n_{i,j,k}}{T} \right]
\end{aligned} \tag{1.50}$$

and rewritten:

$$\begin{aligned}
& \left[1 - \Gamma \left(u^{n+1}_{z,\mathrm{x+}} \frac{\Delta t^n}{\Delta x} + u^{n+1}_{z,\mathrm{y+}} \frac{\Delta t^n}{\Delta y} - u^{n+1}_{z,\mathrm{x-}} \frac{\Delta t^n}{\Delta x} - u^{n+1}_{z,\mathrm{y-}} \frac{\Delta t^n}{\Delta y} + \frac{\Delta t^n}{T} \right) \right] c^{n+1}_{i,j,k} \\
& + \Gamma \left(u^{n+1}_{z,\mathrm{x+}} \frac{\Delta t^n}{\Delta x} c^{n+1}_{i+1,j,k} + u^{n+1}_{z,\mathrm{y+}} \frac{\Delta t^n}{\Delta y} c^{n+1}_{i,j+1,k} - u^{n+1}_{z,\mathrm{x-}} \frac{\Delta t^n}{\Delta x} c^{n+1}_{i-1,j,k} - u^{n+1}_{z,\mathrm{y-}} \frac{\Delta t^n}{\Delta y} c^{n+1}_{i,j-1,k} \right) \\
& = \left[1 + (1-\Gamma) \left(u^n_{z,\mathrm{x+}} \frac{\Delta t^n}{\Delta x} + u^n_{z,\mathrm{y+}} \frac{\Delta t^n}{\Delta y} - u^n_{z,\mathrm{x-}} \frac{\Delta t^n}{\Delta x} - u^n_{z,\mathrm{y-}} \frac{\Delta t^n}{\Delta y} + \frac{\Delta t^n}{T} \right) \right] c^n_{i,j,k} \\
& + (1-\Gamma) \left(u^n_{z,\mathrm{x+}} \frac{\Delta t^n}{\Delta x} c^n_{i+1,j,k} + u^n_{z,\mathrm{y+}} \frac{\Delta t^n}{\Delta y} c^n_{i,j+1,k} - u^n_{z,\mathrm{x-}} \frac{\Delta t^n}{\Delta x} c^n_{i-1,j,k} - u^n_{z,\mathrm{y-}} \frac{\Delta t^n}{\Delta y} c^n_{i,j-1,k} \right) \\
& - \Gamma \hat{w}^{n+1}_{i,j,k} \cdot c^{n+1}_{\mathrm{sat},i,j,k} \frac{\Delta t^n}{T} - (1-\Gamma) \hat{w}^n_{i,j,k} \cdot c^n_{\mathrm{sat},i,j,k} \frac{\Delta t^n}{T}
\end{aligned} \tag{1.51}$$

and simplified:

$$a^{0,0}_{i,j} c^{n+1}_{i,j,k} + a^{1,0}_{i,j} c^{n+1}_{i+1,j,k} + a^{0,1}_{i,j} c^{n+1}_{i,j+1,k} - a^{-1,0}_{i,j} c^{n+1}_{i-1,j,k} - a^{0,-1}_{i,j} c^{n+1}_{i,j-1,k} = y_{i,j,k} \tag{1.52}$$

where the implicit coefficients are defined as:

$$\begin{aligned}
a^{0,0}_{i,j} &= \left[1 - \Gamma \left(u^{n+1}_{z,\mathrm{x+}} \frac{\Delta t^n}{\Delta x} + u^{n+1}_{z,\mathrm{y+}} \frac{\Delta t^n}{\Delta y} - u^{n+1}_{z,\mathrm{x-}} \frac{\Delta t^n}{\Delta x} - u^{n+1}_{z,\mathrm{y-}} \frac{\Delta t^n}{\Delta y} + \frac{\Delta t^n}{T} \right) \right] \\
a^{1,0}_{i,j} &= \Gamma u^{n+1}_{z,\mathrm{x+}} \frac{\Delta t^n}{\Delta x} \\
a^{0,1}_{i,j} &= \Gamma u^{n+1}_{z,\mathrm{y+}} \frac{\Delta t^n}{\Delta y} \\
a^{-1,0}_{i,j} &= \Gamma u^{n+1}_{z,\mathrm{x-}} \frac{\Delta t^n}{\Delta x} \\
a^{0,-1}_{i,j} &= \Gamma u^{n+1}_{z,\mathrm{y-}} \frac{\Delta t^n}{\Delta y}
\end{aligned} \tag{1.53}$$

and the explicit right-hand side as:

$$
\begin{aligned}
y^n_{i,j,k} = \left[1 + (1-\Gamma)\left(u^n_{z,\mathrm{x}+}\frac{\Delta t^n}{\Delta x} + u^n_{z,\mathrm{y}+}\frac{\Delta t^n}{\Delta y} - u^n_{z,\mathrm{x}-}\frac{\Delta t^n}{\Delta x} - u^n_{z,\mathrm{y}-}\frac{\Delta t^n}{\Delta y} + \frac{\Delta t^n}{T}\right)\right] c^n_{i,j,k} \\
+(1-\Gamma)\left(u^n_{z,\mathrm{x}+}\frac{\Delta t^n}{\Delta x}c^n_{i+1,j,k} + u^n_{z,\mathrm{y}+}\frac{\Delta t^n}{\Delta y}c^n_{i,j+1,k} - u^n_{z,\mathrm{x}-}\frac{\Delta t^n}{\Delta x}c^n_{i-1,j,k} - u^n_{z,\mathrm{y}-}\frac{\Delta t^n}{\Delta y}c^n_{i,j-1,k}\right) \\
-\Gamma \hat{w}^{n+1}_{i,j,k} \cdot c^{n+1}_{\mathrm{sat},i,j,k}\frac{\Delta t^n}{T} - (1-\Gamma)\hat{w}^n_{i,j,k} \cdot c^n_{\mathrm{sat},i,j,k}\frac{\Delta t^n}{T}
\end{aligned}
\tag{1.54}
$$

The offshore boundary is defined to be zero-flux, the onshore boundary has a constant transport gradient and the lateral boundaries are circular:

$$
\begin{aligned}
c^{n+1}_{1,j,k} &= 0 \\
c^{n+1}_{n_\mathrm{x}+1,j,k} &= 2c^{n+1}_{n_\mathrm{x},j,k} - c^{n+1}_{n_\mathrm{x}-1,j,k} \\
c^{n+1}_{i,1,k} &= c^{n+1}_{i,n_\mathrm{y}+1,k} \\
c^{n+1}_{i,n_\mathrm{y}+1,k} &= c^{n+1}_{i,1,k}
\end{aligned}
\tag{1.55}
$$

1.2.2 Shear stress perturbation for non-perpendicular wind directions

The shear stress perturbation is estimated following the analytical description of the influence of alow and smooth hill in the wind profile by Weng et al. (1991). The perturbation is given by the Fouriertransformed components of the shear stress perturbation in the unperturbed wind direction which are the functions () and (). The x-direction is defined by the direction of the wind velocity 0 on a flat bed, while the y direction is then the transverse.

As a result, the perturbation theory can only estimate the shear stress induced by the morphology-wind interaction in parallel direction of wind. Therefore, model simulations were, up to now, limited to input wind directions parallel to the crossshore axis of the grid.

To overcome this limitation and to allow for modelling directional winds, an overlaying computational grid is introduced in AeoLiS, which rotates with the changing wind direction per time step. By doing this, the shear stresses are always estimated in the positive x-direction of the computational grid. The following steps are executed for each time step:

1. Create a computational grid alligned with the wind direction (set_computational_grid)
2. Add and fill buffer around the original grid

3. Populate computation grid by rotating it to the current wind direction and interpolate the original topography on it. Additionally, edges around 4. Compute the morphology-wind induced shear stress by using the perturbation theory 5. Add the only wind induced wind shear stresses to the computational grid 6. Rotate both the grids and the total shear stress results in opposite direction 7. Interpolate the total shear stress results from the computational grid to the original grid 8. Rotate the wind shear stress results and the original grid back to the original orientation Note: the extra rotations in the last two steps are necessary as a simplified, but faster in terms of computational time, interpolation method is used.

1.2.3 Boussinesq groundwater equation

The Boussinesq equation is solved numerically with a central finite difference method in space and a fourth-order Runge-Kutta integration technique in time:

$$
f(\eta) = \frac{K}{n_e}\left[D\underbrace{\frac{\partial^2 \eta}{\partial x^2}}_{a} + \underbrace{\frac{\partial}{\partial x}\underbrace{\left\{\eta\frac{\partial \eta}{\partial x}\right\}}_{b}}_{c}\right]
\tag{1.56}
$$

The Runge-Kutta time-stepping, where Δt is the length of the timestep, is defined as,

$$\begin{aligned}
\eta_i^{t+1} &= \eta_i^t + \frac{\Delta t}{6}\left(f_1 + 2f_2 + 2f_3 + f_4\right) \\
f_1 &= f(\eta_i^t) \\
f_2 &= f\left(\eta_i^t + \frac{\Delta t}{2} f_1\right) \\
f_3 &= f\left(\eta_i^t + \frac{\Delta t}{2} f_2\right) \\
f_4 &= f\left(\eta_i^t + \Delta t f_3\right)
\end{aligned} \tag{1.57}$$

where, i is the grid cell in x-direction and t is the timestep. The central difference solution to $f(\eta)$ is obtained through discretisation of the Boussinesq equation,

$$a_i = \frac{\eta_{i+1} - 2\eta_i + \eta_{i-1}}{(\Delta x)^2} \tag{1.58}$$

$$b_i = \frac{\eta_i\left(\eta_{i+1} - \eta_{i-1}\right)}{\Delta x}$$

$$c_i = \frac{\left(b_{i+1} - b_{i-1}\right)}{\Delta x}$$

The seaward boundary condition is defined as the still water level plus the wave setup . If the groundwater elevation is larger than the bed elevation, there is a seepage face, and the groundwater elevation is set equal to the bed elevation. On the landward boundary, a no-flow condition, $\frac{\partial \eta}{\partial t} = 0$ (Neumann condition), or constant head, $\eta = constant$ (Dirichlet condition), is prescribed.

1.2.4 Basic Model Interface (BMI)

A Basic Model Interface (BMI, [PHN13]) is implemented that allows interaction with the model during run time. The model can be implemented as a library within a larger framework as the interface exposes the initialization, finalization and time stepping routines. As a convenience functionality the current implementation supports the specification of a callback function. The callback function is called at the start of each time step and can be used to exchange data with the model, e.g. update the topography from measurements.

An example of a callback function, that is referenced in the model input file or through the model command-line options as `callback.py:update`, is:

```
import numpy as np

def update(model):
  val = model.get_var('zb')
  val_new = val.copy()
  val_new[:,:] = np.loadtxt('measured_topography.txt')
  model.set_var('zb', val_new)
```

Bibliography

1.3 Source code documentation

1.3.1 Use of documentation

Here you can find the documentation with direct links to the actual AeoLiS code. You can click on the green [source] button next to the classes and modules below to access the specific source code. You can use ctr-f to look for a specific functionality or variable. It still may be a bit difficult to browse through, in addition you can find an overview of all module code here

1.3.2 Model classes

The AeoLiS model is based on two main model classes: *AeoLiS* and *AeoLiSRunner*. The former is the actual, low-level, BMI-compatible class that implements the basic model functions and numerical schemes. The latter is a convenience class that implements a time loop, netCDF4 output, a progress indicator and a callback function that allows the used to interact with the model during runtime.

The additional *WindGenerator* class to generate realistic wind time series is available from the same module.

AeoLiS

class model.**AeoLiS**(*configfile*)

AeoLiS model class

AeoLiS is a process-based model for simulating supply-limited aeolian sediment transport. This model class is compatible with the Basic Model Interface (BMI) and provides basic model operations, like initialization, time stepping, finalization and data exchange. For higher level operations, like a progress indicator and netCDF4 output is refered to the AeoLiS model runner class, see *AeoLiSRunner*.

Examples

```
>>> with AeoLiS(configfile='aeolis.txt') as model:
>>>     while model.get_current_time() <= model.get_end_time():
>>>         model.update()
```

```
>>> model = AeoLiS(configfile='aeolis.txt')
>>> model.initialize()
>>> zb = model.get_var('zb')
>>> model.set_var('zb', zb + 1)
>>> for i in range(10):
>>>     model.update(60.) # step 60 seconds forward
>>> model.finalize()
```

__init__(*configfile*)

Initialize class

Parameters
`configfile` (*str*) – Model configuration file. See *read_configfile()*.

`crank_nicolson()`

Convenience function for semi-implicit solver based on Crank-Nicolson scheme

See also:

model.AeoLiS.solve()

`static dimensions(var=None)`

Static method that returns named dimensions of all spatial grids

Parameters
`var` (*str, optional*) – Name of spatial grid

Returns
Tuple with named dimensions of requested spatial grid or dictionary with all named dimensions of all spatial grids. Returns nothing if requested spatial grid is not defined.

Return type
tuple or dict

`euler_backward()`

Convenience function for implicit solver based on Euler backward scheme

See also:

model.AeoLiS.solve()

`euler_forward()`

Convenience function for explicit solver based on Euler forward scheme

See also:

model.AeoLiS.solve()

`finalize()`

Finalize model

`get_count(name)`

Get counter value

Parameters
`name` (*str*) – Name of counter

`get_current_time()`

Returns
Current simulation time

Return type
float

`get_end_time()`

Returns
Final simulation time

Return type
float

get_start_time()

Returns
Initial simulation time

Return type
float

get_var(*var*)

Returns spatial grid or model configuration parameter

If the given variable name matches with a spatial grid, the spatial grid is returned. If not, the given variable name is matched with a model configuration parameter. If a match is found, the parameter value is returned. Otherwise, nothing is returned.

Parameters
var (*str*) – Name of spatial grid or model configuration parameter

Returns
Spatial grid or model configuration parameter

Return type
np.ndarray or int, float, str or list

Examples

```
>>> # returns bathymetry grid
... model.get_var('zb')
```

```
>>> # returns simulation duration
... model.get_var('tstop')
```

See also:

model.AeoLiS.set_var()

get_var_count()

Returns
Number of spatial grids

Return type
int

get_var_name(*i*)

Returns name of spatial grid by index (in alphabetical order)

Parameters
i (*int*) – Index of spatial grid

Returns
Name of spatial grid or -1 in case index exceeds the number of grids

Return type
str or -1

get_var_rank(*var*)

Returns rank of spatial grid

Parameters
`var` (*str*) – Name of spatial grid

Returns
Rank of spatial grid or -1 if not found

Return type
int

`get_var_shape`(*var*)

Returns shape of spatial grid

Parameters
`var` (*str*) – Name of spatial grid

Returns
Dimensions of spatial grid or -1 if not found

Return type
tuple or int

`get_var_type`(*var*)

Returns variable type of spatial grid

Parameters
`var` (*str*) – Name of spatial grid

Returns
Variable type of spatial grid or -1 if not found

Return type
str or int

`initialize`()

Initialize model

Read model configuration file and initialize parameters and spatial grids dictionary and load bathymetry and bed composition.

`inq_compound`()

Return the number of fields of a compound type.

`inq_compound_field`()

Lookup the type,rank and shape of a compound field

`set_timestep`(*dt=-1.0*)

Determine optimal time step

If no time step is given the optimal time step is determined. For explicit numerical schemes the time step is based in the Courant-Frierichs-Lewy (CFL) condition. For implicit numerical schemes the time step specified in the model configuration file is used. Alternatively, a preferred time step is given that is maximized by the CFL condition in case of an explicit numerical scheme.

Returns True except when:

1. No time step could be determined, for example when there is no wind and the numerical scheme is explicit. In this case the time step is set arbitrarily to one second.

2. Or when the time step is smaller than -1. In this case the time is updated with the absolute value of the time step, but no model execution is performed. This funcionality can be used to skip fast-forward in time.

Parameters
`df` (*float, optional*) – Preferred time step

Returns
False if determination of time step was unsuccessful, True otherwise

Return type
bool

`set_var`(*var*, *val*)

Sets spatial grid or model configuration parameter

If the given variable name matches with a spatial grid, the spatial grid is set. If not, the given variable name is matched with a model configuration parameter. If a match is found, the parameter value is set. Otherwise, nothing is set.

Parameters

- **var** (*str*) – Name of spatial grid or model configuration parameter
- **val** (*np.ndarray or int, float, str or list*) – Spatial grid or model configuration parameter

Examples

```
>>> # set bathymetry grid
... model.set_var('zb', np.array([[0.,0., ... ,0.]]))
```

```
>>> # set simulation duration
... model.set_var('tstop', 3600.)
```

See also:

model.AeoLiS.get_var()

`set_var_index`(*i*, *val*)

Set spatial grid by index (in alphabetical order)

Parameters

- **i** (*int*) – Index of spatial grid
- **val** (*np.ndarray*) – Spatial grid

`set_var_slice`()

Overwrite the values in variable name with data from var, in the range (start:start+count). Start, count can be integers for rank 1, and can be tuples of integers for higher ranks. For some implementations it can be equivalent and more efficient to do: *get_var(name)[start[0]:start[0]+count[0], …, start[n]:start[n]+count[n]] = var*

`solve`(*alpha=0.5*, *beta=1.0*)

Implements the explicit Euler forward, implicit Euler backward and semi-implicit Crank-Nicolson numerical schemes

Determines weights of sediment fractions, sediment pickup and instantaneous sediment concentration. Returns a partial spatial grid dictionary that can be used to update the global spatial grid dictionary.

Parameters

- **alpha** (*float, optional*) – Implicitness coefficient (0.0 for Euler forward, 1.0 for Euler backward or 0.5 for Crank-Nicolson, default=0.5)

- **beta** (*float, optional*) – Centralization coefficient (1.0 for upwind or 0.5 for centralized, default=1.0)

Returns
Partial spatial grid dictionary

Return type
dict

Examples

```
>>> model.s.update(model.solve(alpha=1., beta=1.) # euler backward
```

```
>>> model.s.update(model.solve(alpha=.5, beta=1.) # crank-nicolson
```

See also:

model.AeoLiS.euler_forward(), *model.AeoLiS.euler_backward()*, *model.AeoLiS.crank_nicolson()*, *transport.compute_weights()*, *transport.renormalize_weights()*

solve_pieter(*alpha=0.5*, *beta=1.0*)

Implements the explicit Euler forward, implicit Euler backward and semi-implicit Crank-Nicolson numerical schemes

Determines weights of sediment fractions, sediment pickup and instantaneous sediment concentration. Returns a partial spatial grid dictionary that can be used to update the global spatial grid dictionary.

Parameters

- **alpha** (*float, optional*) – Implicitness coefficient (0.0 for Euler forward, 1.0 for Euler backward or 0.5 for Crank-Nicolson, default=0.5)
- **beta** (*float, optional*) – Centralization coefficient (1.0 for upwind or 0.5 for centralized, default=1.0)

Returns
Partial spatial grid dictionary

Return type
dict

Examples

```
>>> model.s.update(model.solve(alpha=1., beta=1.) # euler backward
```

```
>>> model.s.update(model.solve(alpha=.5, beta=1.) # crank-nicolson
```

See also:

model.AeoLiS.euler_forward(), *model.AeoLiS.euler_backward()*, *model.AeoLiS.crank_nicolson()*, *transport.compute_weights()*, *transport.renormalize_weights()*

solve_steadystate()

Implements the steady state solution

update(*dt=-1*)

Time stepping function

Takes a single step in time. Interpolates wind and hydrodynamic time series to the current time, updates the soil moisture, mixes the bed due to wave action, computes wind velocity threshold and the equilibrium sediment transport concentration. Subsequently runs one of the available numerical schemes to compute the instantaneous sediment concentration and pickup for the next time step and updates the bed accordingly.

For explicit schemes the time step is maximized by the Courant-Friedrichs-Lewy (CFL) condition. See *set_timestep()*.

Parameters

dt (*float, optional*) – Time step in seconds. The time step specified in the model configuration file is used in case dt is smaller than zero. For explicit numerical schemes the time step is maximized by the CFL confition.

AeoLiSRunner

class model.**AeoLiSRunner**(*configfile='aeolis.txt'*)

AeoLiS model runner class

This runner class is a convenience class for the BMI-compatible AeoLiS model class (*AeoLiS()*). It implements a time loop, a progress indicator and netCDF4 output. It also provides the definition of a callback function that can be used to interact with the AeoLiS model during runtime.

The command-line function `aeolis` is available that uses this class to start an AeoLiS model run.

Examples

```
>>> # run with default settings
... AeoLiSRunner().run()
```

```
>>> AeoLiSRunner(configfile='aeolis.txt').run()
```

```
>>> model = AeoLiSRunner(configfile='aeolis.txt')
>>> model.run(callback=lambda model: model.set_var('zb', zb))
```

```
>>> model.run(callback='bar.py:add_bar')
```

See also:

console.aeolis

__init__(*configfile='aeolis.txt'*)

Initialize class

Reads model configuration file without parsing all referenced files for the progress indicator and netCDF output. If no configuration file is given, the default settings are used.

Parameters

configfile (*str, optional*) – Model configuration file. See *read_configfile()*.

dump_restartfile()

Dump model state to restart file

get_statistic(*var*, *stat='avg'*)

Return statistic of spatial grid

Parameters

- **var** (*str*) – Name of spatial grid
- **stat** (*str*) – Name of statistic (avg, sum, var, min or max)

Returns

Statistic of spatial grid

Return type

numpy.ndarray

get_var(*var*, *clear=True*)

Returns spatial grid, statistic or model configuration parameter

Overloads the *get_var()* function and extends it with the functionality to return statistics on spatial grids by adding a postfix to the variable name (e.g. Ct_avg). Supported statistics are avg, sum, var, min and max.

Parameters

- **var** (*str*) – Name of spatial grid or model configuration parameter. Spatial grid name can be extended with a postfix to request a statistic (_avg, _sum, _var, _min or _max).
- **clear** (*bool*) – Clear output statistics afterwards.

Returns

Spatial grid, statistic or model configuration parameter

Return type

np.ndarray or int, float, str or list

Examples

```
>>> # returns average sediment concentration
... model.get_var('Ct_avg')
```

```
>>> # returns variance in wave height
... model.get_var('Hs_var')
```

See also:

model.AeoLiS.get_var()

initialize()

Initialize model

Overloads the *initialize()* function, but also initializes output statistics.

load_hotstartfiles()

Load model state from hotstart files

Hotstart files are plain text representations of model state variables that can be used to hotstart the (partial) model state. Hotstart files should have the name of the model state variable it contains and have the extension *.hotstart*. Hotstart files differ from restart files in that restart files contain entire model states and are pickled Python objects.

See also:

model.AeoLiSRunner.load_restartfile()

`load_restartfile`(*restartfile*)

Load model state from restart file

Parameters
`restartfile` (*str*) – Path to previously written restartfile.

`output_clear`()

Clears output statistics dictionary

Creates a matrix for minimum, maximum, variance and summed values for each output variable and sets the time step counter to zero.

`output_init`()

Initialize netCDF4 output file and output statistics dictionary

`output_update`()

Updates output statistics dictionary

Updates matrices with minimum, maximum, variance and summed values for each output variable with current spatial grid values and increases time step counter with one.

`output_write`()

Appends output to netCDF4 output file

If the time since the last output is equal or larger than the set output interval, append current output to the netCDF4 output file. Computes the average and variance values based on available output statistics and clear output statistics dictionary.

`parse_callback`(*callback*)

Parses callback definition and returns function

The callback function can be specified in two formats:

- As a native Python function
- As a string refering to a Python script and function, separated by a colon (e.g. `example/callback.py:function`)

Parameters
`callback` (*str or function*) – Callback definition

Returns
Python callback function

Return type
function

`print_params`()

Print model configuration parameters to screen

`print_progress`(*fraction=0.01*, *min_interval=1.0*, *max_interval=60.0*)

Print progress to screen

Parameters

- `fraction` (*float, optional*) – Fraction of simulation at which to print progress (default: 1%)
- `min_interval` (*float, optional*) – Minimum time in seconds between subsequent progress prints (default: 1s)

- **max_interval** (*float, optional*) – Maximum time in seconds between subsequent progress prints (default: 60s)

print_stats()

Print model run statistics to screen

run(*callback=None*, *restartfile=None*)

Start model time loop

Changes current working directory to the model directory, prints model configuration parameters and progress indicator to the screen, writes netCDF4 output and calls a callback function upon request.

Parameters

- **callback** (*str or function*) – The callback function is called at the start of every single time step and takes the AeoLiS model object as input. The callback function can be used to interact with the model during simulation (e.g. update the bed with new measurements). See for syntax `parse_callback()`.
- **restartfile** (*str*) – Path to previously written restartfile. The model state is loaded from this file after initialization of the model.

See also:

`model.AeoLiSRunner.parse_callback()`

set_configfile(*configfile*)

Set model configuration file name

set_params(***kwargs*)

Set model configuration parameters

update(*dt=-1*)

Time stepping function

Overloads the `update()` function, but also updates output statistics and clears output statistics upon request.

Parameters

dt (*float, optional*) – Time step in seconds.

write_params()

Write updated model configuration to configuration file

Creates a backup in case the model configration file already exists.

See also:

`inout.backup()`

WindGenerator

class model.**WindGenerator**(*mean_speed=9.0*, *max_speed=30.0*, *dt=60.0*, *n_states=30*, *shape=2.0*, *scale=2.0*)

Wind velocity time series generator

Generates a random wind velocity time series with given mean and maximum wind speed, duration and time resolution. The wind velocity time series is generated using a Markov Chain Monte Carlo (MCMC) approach based on a Weibull distribution. The wind time series can be written to an AeoLiS-compatible wind input file assuming a constant wind direction of zero degrees.

The command-line function `aeolis-wind` is available that uses this class to generate AeoLiS wind input files.

Examples

```
>>> wind = WindGenerator(mean_speed=10.).generate(duration=24*3600.)
>>> wind.write_time_series('wind.txt')
>>> wind.plot()
>>> wind.hist()
```

See also:

console.wind

__init__(*mean_speed=9.0*, *max_speed=30.0*, *dt=60.0*, *n_states=30*, *shape=2.0*, *scale=2.0*)

__weakref__
list of weak references to the object (if defined)

1.3.3 Physics modules

Bathymetry and bed composition

bed.**initialize**(*s*, *p*)
Initialize bathymetry and bed composition

Initialized bathymetry, computes cell sizes and orientation, bed layer thickness and bed composition.

Parameters

- **s** (*dict*) – Spatial grids
- **p** (*dict*) – Model configuration parameters

Returns
Spatial grids

Return type
dict

bed.**mixtoplayer**(*s*, *p*)
Mix grain size distribution in top layer of the bed.

Simulates mixing of the top layers of the bed by wave action. The wave action is represented by a local wave height maximized by a maximum wave hieght over depth ratio `gamma`. The mixing depth is a fraction of the local wave height indicated by `facDOD`. The mixing depth is used to compute the number of bed layers that should be included in the mixing. The grain size distribution in these layers is then replaced by the average grain size distribution over these layers.

Parameters

- **s** (*dict*) – Spatial grids
- **p** (*dict*) – Model configuration parameters

Returns
Spatial grids

Return type
dict

`bed.prevent_negative_mass(m, dm, pickup)`

Handle situations in which negative mass may occur due to numerics

Negative mass may occur by moving sediment to lower layers down to accomodate deposition of sediments. In particular two cases are important:

1. A net deposition cell has some erosional fractions.

   In this case the top layer mass is reduced according to the existing sediment distribution in the layer to accomodate deposition of fresh sediment. If the erosional fraction is subtracted afterwards, negative values may occur. Therefore the erosional fractions are subtracted from the top layer beforehand in this function. An equal mass of deposition fractions is added to the top layer in order to keep the total layer mass constant. Subsequently, the distribution of the sediment to be moved to lower layers is determined and the remaining deposits are accomodated.

2. Deposition is larger than the total mass in a layer.

   In this case a non-uniform distribution in the bed may also lead to negative values as the abundant fractions are reduced disproportionally as sediment is moved to lower layers to accomodate the deposits. This function fills the top layers entirely with fresh deposits and moves the existing sediment down such that the remaining deposits have a total mass less than the total bed layer mass. Only the remaining deposits are fed to the routine that moves sediment through the layers.

Parameters

- **m** (*np.ndarray*) – Sediment mass in bed (nx*ny, nl, nf)
- **dm** (*np.ndarray*) – Total sediment mass exchanged between layers (nx*ny, nf)
- **pickup** (*np.ndarray*) – Sediment pickup (nx*ny, nf)

Returns

- *np.ndarray* – Sediment mass in bed (nx*ny, nl, nf)
- *np.ndarray* – Total sediment mass exchanged between layers (nx*ny, nf)
- *np.ndarray* – Sediment pickup (nx*ny, nf)

Note: The situations handled in this function can also be prevented by reducing the time step, increasing the layer mass or increasing the adaptation time scale.

`bed.update(s, p)`

Update bathymetry and bed composition

Update bed composition by moving sediment fractions between bed layers. The total mass in a single bed layer does not change as sediment removed from a layer is repleted with sediment from underlying layers. Similarly, excess sediment added in a layer is moved to underlying layers in order to keep the layer mass constant. The lowest bed layer exchanges sediment with an infinite sediment source that follows the original grain size distribution as defined in the model configuration file by `grain_size` and `grain_dist`. The bathymetry is updated following the cummulative erosion/deposition over the fractions if `bedupdate` is `True`.

Parameters

- **s** (*dict*) – Spatial grids
- **p** (*dict*) – Model configuration parameters

Returns

Spatial grids

Return type
dict

`bed.wet_bed_reset(s, p)`

Text

Parameters

- **s** (*dict*) – Spatial grids
- **p** (*dict*) – Model configuration parameters

Returns
Spatial grids

Return type
dict

Wind velocity and direction

`wind.calculate_z0(p, s)`

Calculate z0 according to chosen roughness method

The z0 is required for the calculation of the shear velocity. Here, z0 is calculated based on a user-defined method. The constant method defines the value of z0 as equal to k (z0 = ks). This was implemented to ensure backward compatibility and does not follow the definition of Nikuradse (z0 = k / 30). For following the definition of Nikuradse use the method constant_nikuradse. The mean_grainsize_initial method uses the intial mean grain size ascribed to the bed (grain_dist and grain_size in the input file) to calculate the z0. The median_grainsize_adaptive bases the z0 on the median grain size (D50) in the surface layer in every time step. The resulting z0 is variable accross the domain (x,y). The strypsteen_vanrijn method is based on the roughness calculation in their paper.

Parameters

- **s** (*dict*) – Spatial grids
- **p** (*dict*) – Model configuration parameters

Returns
z0

Return type
array

`wind.compute_shear1d(s, p)`

Compute wind shear perturbation for given free-flow wind speed on computational grid. based on same implementation in Duna

`wind.initialize(s, p)`

Initialize wind model

`wind.interpolate(s, p, t)`

Interpolate wind velocity and direction to current time step

Interpolates the wind time series for velocity and direction to the current time step. The cosine and sine of the direction angle are interpolated separately to prevent zero-crossing errors. The wind velocity is decomposed in two grid components based on the orientation of each individual grid cell. In case of a one-dimensional model only a single positive component is used.

Parameters

- **s** (*dict*) – Spatial grids

- **p** (*dict*) – Model configuration parameters
- **t** (*float*) – Current time

Returns
Spatial grids

Return type
dict

class shear.WindShear(*x, y, z, dx, dy, L, l, z0, buffer_width, buffer_relaxation=None*)

Class for computation of 2DH wind shear perturbations over a topography.

The class implements a 2D FFT solution to the wind shear perturbation on curvilinear grids. As the FFT solution is only defined on an equidistant rectilinear grid with circular boundary conditions that is aligned with the wind direction, a rotating computational grid is automatically defined for the computation. The computational grid is extended in all directions using a logistic sigmoid function as to ensure full coverage of the input grid for all wind directions, circular boundaries and preservation of the alongshore uniformity. An extra buffer distance can be used as to minimize the disturbence from the borders in the input grid. The results are interpolated back to the input grid when necessary.

Frequencies related to wave lengths smaller than a computational grid cell are filtered from the 2D spectrum of the topography using a logistic sigmoid tapering. The filtering aims to minimize the disturbance as a result of discontinuities in the topography that may physically exists, but cannot be solved for in the computational grid used.

Example

```
>>> w = WindShear(x, y, z)
>>> w(u0=10., udir=30.).add_shear(taux, tauy)
```

Notes

To do:

- **Actual resulting values are still to be compared with the results**
  from Kroy et al. (2002)
- Grid interpolation can still be optimized
- Separation bubble is still to be improved

add_shear()

Add wind shear perturbations to a given wind shear field

Parameters

- **taux** (*numpy.ndarray*) – Wind shear in x-direction
- **tauy** (*numpy.ndarray*) – Wind shear in y-direction

Returns

- **taux** (*numpy.ndarray*) – Wind shear including perturbations in x-direction
- **tauy** (*numpy.ndarray*) – Wind shear including perturbations in y-direction

`compute_shear`(*u0*, *nfilter=(1.0, 2.0)*)

Compute wind shear perturbation for given free-flow wind speed on computational grid

Parameters

- **u0** (*float*) – Free-flow wind speed
- **nfilter** (*2-tuple*) – Wavenumber range used for logistic sigmoid filter. See `filter_highfrequencies()`

`filter_highfrequenies`(*kx*, *ky*, *hs*, *nfilter=(1, 2)*)

Filter high frequencies from a 2D spectrum

A logistic sigmoid filter is used to taper higher frequencies from the 2D spectrum. The range over which the sigmoid runs from 0 to 1 with a precision `p` is given by the 2-tuple `nfilter`. The range is defined as wavenumbers in terms of gridcells, i.e. a value 1 corresponds to a wave with length `dx`.

Parameters

- **kx** (*numpy.ndarray*) – Wavenumbers in x-direction
- **ky** (*numpy.ndarray*) – Wavenumbers in y-direction
- **hs** (*numpy.ndarray*) – 2D spectrum
- **nfilter** (*2-tuple*) – Wavenumber range used for logistic sigmoid filter
- **p** (*float*) – Precision of sigmoid range definition

Returns

hs – Filtered 2D spectrum

Return type

numpy.ndarray

`static get_borders`(*x*)

Returns borders of a grid as one-dimensional array

`static get_exact_grid`(*xmin*, *xmax*, *ymin*, *ymax*, *dx*, *dy*)

Returns a grid with given gridsizes approximately within given bounding box

`get_separation`()

Returns difference in height between z-coordinate of the separation polynomial and of the bed level

Returns

hsep – Height of seperation bubble

Return type

numpy.ndarray

`get_shear`()

Returns wind shear perturbation field

Returns

- **taux** (*numpy.ndarray*) – Wind shear perturbation in x-direction
- **tauy** (*numpy.ndarray*) – Wind shear perturbation in y-direction

`interpolate`(*x*, *y*, *z*, *xi*, *yi*, *z0*)

Interpolate one grid to an other

`plot`(*ax=None*, *cmap='Reds'*, *stride=10*, *computational_grid=False*, ***kwargs*)

Plot wind shear perturbation

Parameters

- **ax** (*matplotlib.pyplot.Axes, optional*) – Axes to plot onto
- **cmap** (*matplotlib.cm.Colormap or string, optional*) – Colormap for topography (default: Reds)
- **stride** (*int, optional*) – Stride to apply to wind shear vectors (default: 10)
- **computational_grid** (*bool, optional*) – Plot on computational grid rather than input grid (default: False)
- **kwargs** (*dict*) – Additional arguments to `matplotlib.pyplot.quiver()`

Returns
ax – Axes used for plotting

Return type
matplotlib.pyplot.Axes

static `rotate`(*x*, *y*, *alpha*, *origin=(0, 0)*)

Rotate a matrix over given angle around given origin

`separation_shear`(*hsep*)

Reduces the computed wind shear perturbation below the separation surface to mimic the turbulence effects in the separation bubble

Parameters
hsep (*numpy.ndarray*) – Height of seperation bubble (in x direction)

`set_computational_grid`(*udir*)

Define computational grid

The computational grid is square with dimensions equal to the diagonal of the bounding box of the input grid, plus twice the buffer width.

Wind velocity threshold

`threshold.compute`(*s*, *p*)

Compute wind velocity threshold based on bed surface properties

Computes wind velocity threshold based on grain size fractions, bed slope, soil moisture content, air humidity, the presence of roughness elements and a non-erodible layer. All bed surface properties increase the current wind velocity threshold, except for the grain size fractions. Therefore, the computation is initialized by the grain size fractions and subsequently altered by the other bed surface properties.

Parameters

- **s** (*dict*) – Spatial grids
- **p** (*dict*) – Model configuration parameters

Returns
Spatial grids

Return type
dict

See also:

compute_grainsize(), *compute_bedslope()*, *compute_moisture()*, compute_humidity(), *compute_sheltering()*, *non_erodible()*

threshold.**compute_bedslope**(*s*, *p*)

Modify wind velocity threshold based on bed slopes following Dyer (1986)

Parameters

- **s** (*dict*) – Spatial grids
- **p** (*dict*) – Model configuration parameters

Returns
Spatial grids

Return type
dict

threshold.**compute_grainsize**(*s*, *p*)

Compute wind velocity threshold based on grain size fractions following Bagnold (1937)

Parameters

- **s** (*dict*) – Spatial grids
- **p** (*dict*) – Model configuration parameters

Returns
Spatial grids

Return type
dict

threshold.**compute_moisture**(*s*, *p*)

Modify wind velocity threshold based on soil moisture content following Belly (1964) or Hotta (1984)

Parameters

- **s** (*dict*) – Spatial grids
- **p** (*dict*) – Model configuration parameters

Returns
Spatial grids

Return type
dict

threshold.**compute_salt**(*s*, *p*)

Modify wind velocity threshold based on salt content following Nickling and Ecclestone (1981)

Parameters

- **s** (*dict*) – Spatial grids
- **p** (*dict*) – Model configuration parameters

Returns
Spatial grids

Return type
dict

`threshold.compute_sheltering(s, p)`

Modify wind velocity threshold based on the presence of roughness elements following Raupach (1993)

Raupach (1993) presents the following amplification factor for the shear velocity threshold due to the presence of roughness elements.

$$R_t = \frac{u_{*,th,s}}{u_{*,th,r}} = \sqrt{\frac{\tau_s''}{\tau}} = \frac{1}{\sqrt{(1 - m\sigma\lambda)\,(1 + m\beta\lambda)}}$$

m is a constant smaller or equal to unity that accounts for the difference between the average stress on the bed surface τ_s and the maximum stress on the bed surface τ_s''.

β is the stress partition coefficient defined as the ratio between the drag coefficient of the roughness element itself C_r and the drag coefficient of the bare surface without roughness elements C_s.

σ is the shape coefficient defined as the basal area divided by the frontal area: $\frac{A_b}{A_f}$. For hemispheres $\sigma = 2$, for spheres $\sigma = 1$.

λ is the roughness density defined as the number of elements per surface area $\frac{n}{S}$ multiplied by the frontal area of a roughness element A_f, also known as the frontal area index:

$$\lambda = \frac{nbh}{S} = \frac{nA_f}{S}$$

If multiplied by σ the equation simplifies to the mass fraction of non-erodible elements:

$$\sigma\lambda = \frac{nA_b}{S} = \sum_{k=n_0}^{n_k} \hat{w}_k^{\mathrm{bed}}$$

where k is the fraction index, n_0 is the smallest non-erodible fraction, n_k is the largest non-erodible fraction and $\hat{w}_k^{\mathrm{bed}}$ is the mass fraction of sediment fraction k. It is assumed that the fractions are ordered by increasing size.

Substituting the derivation in the Raupach (1993) equation gives the formulation implemented in this function:

$$u_{*,th,r} = u_{*,th,s} * \sqrt{\left(1 - m \sum_{k=n_0}^{n_k} \hat{w}_k^{\mathrm{bed}}\right) \left(1 + m\frac{\beta}{\sigma} \sum_{k=n_0}^{n_k} \hat{w}_k^{\mathrm{bed}}\right)}$$

Parameters

- **s** (*dict*) – Spatial grids
- **p** (*dict*) – Model configuration parameters

Returns
Spatial grids

Return type
dict

`threshold.non_erodible(s, p)`

Modify wind velocity threshold based on the presence of a non-erodible layer.

Parameters

- **s** (*dict*) – Spatial grids
- **p** (*dict*) – Model configuration parameters

Returns
Spatial grids

Return type
dict

Tides, meteorology and soil moisture content

Sediment transport

`transport.`**`compute_weights`**(s, p)

Compute weights for sediment fractions

Multi-fraction sediment transport needs to weigh the transport of each sediment fraction to prevent the sediment transport to increase with an increasing number of sediment fractions. The weighing is not uniform over all sediment fractions, but depends on the sediment availibility in the air and the bed and the bed interaction parameter `bi`.

Parameters

- **s** (*dict*) – Spatial grids
- **p** (*dict*) – Model configuration parameters

Returns
Array with weights for each sediment fraction

Return type
numpy.ndarray

`transport.`**`constant_grainspeed`**(s, p)

Define saltation velocity u [m/s]

Parameters

- **s** (*dict*) – Spatial grids
- **p** (*dict*) – Model configuration parameters

Returns
Spatial grids

Return type
dict

`transport.`**`duran_grainspeed`**(s, p)

Compute grain speed according to Duran 2007 (p. 42)

Parameters

- **s** (*dict*) – Spatial grids
- **p** (*dict*) – Model configuration parameters

Returns
Spatial grids

Return type
dict

`transport.`**`equilibrium`**(s, p)

Compute equilibrium sediment concentration following Bagnold (1937)

Parameters

- **s** (*dict*) – Spatial grids
- **p** (*dict*) – Model configuration parameters

Returns
Spatial grids

Return type
dict

`transport.`**`renormalize_weights`**`(`*w*, *ix*`)`

Renormalizes weights for sediment fractions

Renormalizes weights for sediment fractions such that the sum of all weights is unity. To ensure that the erosion of specific fractions does not exceed the sediment availibility in the bed, the normalization only modifies the weights with index equal or larger than `ix`.

Parameters

- **w** (*numpy.ndarray*) – Array with weights for each sediment fraction
- **ix** (*int*) – Minimum index to be modified

Returns
Array with weights for each sediment fraction

Return type
numpy.ndarray

Avalanching

`avalanching.`**`angele_of_repose`**`(`*s*, *p*`)`

Determine the dynamic and static angle of repose.

Both the critical dynamic and static angle of repose are spatial varying and depend on surface moisture content and roots of present vegetation and . . . .

Parameters

- **s** (*dict*) – Spatial grids
- **p** (*dict*) – Model configuration parameters

Returns
Spatial grids

Return type
dict

`avalanching.`**`avalanche`**`(`*s*, *p*`)`

Avalanching occurs if bed slopes exceed critical slopes.

Simulates the process of avalanching that is triggered by the exceedence of a critical static slope `theta_stat` by the bed slope. The iteration stops if the bed slope does not exceed the dynamic critical slope `theta_dyn`.

Parameters

- **s** (*dict*) – Spatial grids
- **p** (*dict*) – Model configuration parameters

Returns
Spatial grids

Return type
dict

avalanching.**calc_gradients**(*zb*, *nx*, *ny*, *ds*, *dn*, *zne*)

Calculates the downslope gradients in the bed that are needed for avalanching module

Returns
Downslope gradients in 4 different directions (nx*ny, 4)

Return type
np.ndarray

Vegetation

vegetation.**initialize**(*s*, *p*)

Initialise vegetation based on vegetation file.

Parameters

- **s** (*dict*) – Spatial grids
- **p** (*dict*) – Model configuration parameters

Returns
Spatial grids

Return type
dict

Marine Erosion

erosion.**run_ph12**(*s*, *p*, *t*)

Calculates bed level change due to dune erosion

Calculates bed level change due to dune erosion accoording to Palmsten and Holman (2012).

Parameters

- **s** (*dict*) – Spatial grids
- **p** (*dict*) – Model configuration parameters
- **t** (*float*) – Model time

Returns
Spatial grids

Return type
dict

1.3.4 Helper modules

Input/Output

inout.**backup**(*fname*)

Creates a backup file of the provided file, if it exists inout.**check_configuration**(*p*)

Check model configuration validity

Checks if required parameters are set and if the references files for bathymetry, wind, tide and meteorological input are valid. Throws an error if one or more requirements are not met.

Parameters
p (*dict*) – Model configuration dictionary with parsed files

See also:

read_configfile()

inout.**get_backupfilename**(*fname*)

Returns a non-existing backup filename

inout.**parse_value**(*val*, *parse_files=True*, *force_list=False*)

Casts a string to the most appropriate variable type

Parameters

- **val** (*str*) – String representation of value
- **parse_files** (*bool*) – If True, files referred to by string parameters are parsed by `numpy.loadtxt`
- **force_list** – If True, interpret the value as a list, even if it consists of a single value

Returns
Casted value

Return type
misc

Examples

```
>>> type(parse_value('T'))
    bool
>>> type(parse_value('F'))
    bool
>>> type(parse_value('123'))
    int
>>> type(parse_value('123.2'))
    float
>>> type(parse_value('euler_forward'))
    str
>>> type(parse_value(''))
    NoneType
>>> type(parse_value('zb zs Ct'))
    numpy.ndarray
>>> type(parse_value('zb', force_list=True))
    numpy.ndarray
>>> type(parse_value('0.1 0.2 0.3')[0])
    float
>>> type(parse_value('wind.txt'), parse_files=True)
    numpy.ndarray
>>> type(parse_value('wind.txt'), parse_files=False)
    str
```

`inout.read_configfile(configfile, parse_files=True, load_defaults=True)`

Read model configuration file

Updates default model configuration based on a model configuration file. The model configuration file should be a text file with one parameter on each line. The parameter name and value are seperated by an equal sign (=). Any lines that start with a percent sign (%) or do not contain an equal sign are omitted.

Parameters are casted into the best matching variable type. If the variable type is `str` it is optionally interpreted as a filename. If the corresponding file is found it is parsed using the `numpy.loadtxt` function.

Parameters

- **configfile** (*str*) – Model configuration file
- **parse_files** (*bool*) – If True, files referred to by string parameters are parsed
- **load_defaults** (*bool*) – If True, default settings are loaded and overwritten by the settings from the configuration file

Returns

Dictionary with casted and optionally parsed model configuration parameters

Return type

dict

See also:

write_configfile(), *check_configuration()*

`inout.visualize_grid(s, p)`

Create figures and tables for the user to check whether the grid-input is correctly interpreted

`inout.visualize_spatial(s, p)`

Create figures and tables for the user to check whether the input is correctly interpreted

`inout.visualize_timeseries(p, t)`

Create figures and tables for the user to check whether the timeseries-input is correctly interpreted

`inout.write_configfile(configfile, p=None)`

Write model configuration file

Writes model configuration to file. If no model configuration is given, the default configuration is written to file. Any parameters with a name ending with *_file* and holding a matrix are treated as separate files. The matrix is then written to an ASCII file using the `numpy.savetxt` function and the parameter value is replaced by the name of the ASCII file.

Parameters

- **configfile** (*str*) – Model configuration file
- **p** (*dict, optional*) – Dictionary with model configuration parameters

Returns

Dictionary with casted and optionally parsed model configuration parameters

Return type

dict

See also:

read_configfile()

netCDF4 output

`netcdf.append`(*outputfile*, *variables*)

Append variables to existing netCDF4 output file

Increments the time axis length with one and appends the provided spatial grids along the time axis. The `variables` dictionary should at least have the `time` field indicating the current simulation time. The CF time bounds are updated accordingly.

Parameters

- **outputfile** (*str*) – Name of netCDF4 output file
- **variables** (*dict*) – Dictionary with spatial grids and time

Examples

```
>>> netcdf.append('aeolis.nc', {'time', 3600.,
...                              'Ct', np.array([[0.,0., ... ,0.]]),
...                              'Cu', np.array([[1.,1., ... ,1.]]))
```

See also:

set_bounds()

`netcdf.dump`(*outputfile*, *dumpfile*, *var='mass'*, *ix=-1*)

Dumps time slice from netCDF4 output file to ASCII file

This function can be used to use a specific time slice from a netCDF4 output file as input file for another AeoLiS model run. For example, the bed composition from a spinup run can be used as initial composition for other runs reducing the spinup time.

Parameters

- **outputfile** (*str*) – Name of netCDF4 output file
- **dumpfile** (*str*) – Name of ASCII dump file
- **var** (*str, optional*) – Name of spatial grid to be dumped (default: mass)
- **ix** (*int*) – Time slice index to be dumped (default: -1)

Examples

```
>>> # use bedcomp_file = bedcomp.txt in model configuration file
... netcdf.dump('aeolis.nc', 'bedcomp.txt', var='mass')
```

`netcdf.initialize`(*outputfile*, *outputvars*, *s*, *p*, *dimensions*)

Create empty CF-compatible netCDF4 output file

Parameters

- **outputfile** (*str*) – Name of netCDF4 output file
- **outputvars** (*dictionary*) – Spatial grids to be written to netCDF4 output file
- **s** (*dict*) – Spatial grids
- **p** (*dict*) – Model configuration parameters

- **dimensions** (*dict*) – Dictionary that specifies a tuple with the named dimensions for each spatial grid (e.g. ('ny', 'nx', 'nfractions'))

Examples

```
>>> netcdf.initialize('aeolis.nc',
...                   ['Ct', 'Cu', 'zb'],
...                   ['avg', 'max'],
...                   s, p, {'Ct':('ny','nx','nfractions'),
...                          'Cu':('ny','nx','nfractions'),
...                          'zb':('ny','nx')})
```

`netcdf.parse_metadata(outputvars)`

Parse metadata from constants.py

Parses the Python comments in constants.py to extract meta data, like units, for the model state variables that can be used as netCDF4 meta data.

Parameters
outputvars (*dictionary*) – Spatial grids to be written to netCDF4 output file

Returns
meta – Dictionary with meta data for the output variables

Return type
dict

`netcdf.set_bounds(outputfile)`

Sets CF time bounds

Parameters
outputfile (*str*) – Name of netCDF4 output file

Plotting

Command-line tools

`console.aeolis()`

aeolis : a process-based model for simulating supply-limited aeolian sediment transport

Usage:
aeolis <config> [options]

Positional arguments:
config configuration file

Options:

-h, --help	show this help message and exit
--callback=FUNC	reference to callback function (e.g. example/callback.py:callback)
--restart=FILE	model restart file
--verbose=LEVEL	logging verbosity [default: 20]
--debug	write debug logs

`console.wind()`

aeolis-wind : a wind time series generation tool for the aeolis model

Usage:
aeolis-wind <file> [–mean=MEAN] [–max=MAX] [–duration=DURATION] [–timestep=TIMESTEP]

Positional arguments:
file output file

Options:

-h, --help	show this help message and exit
--mean=MEAN	mean wind speed [default: 10]
--max=MAX	maximum wind speed [default: 30]
--duration=DURATION	duration of time series [default: 3600]
--timestep=TIMESTEP	timestep of time series [default: 60]

Miscellaneous

`utils.apply_mask(arr, mask)`

Apply complex mask

The real part of the complex mask is multiplied with the input array. Subsequently the imaginary part is added and the result returned.

The shape of the mask is assumed to match the first few dimensions of the input array. If the input array is larger than the mask, the mask is repeated for any additional dimensions.

Parameters

- **arr** (*numpy.ndarray*) – Array or matrix to which the mask needs to be applied
- **mask** (*numpy.ndarray*) – Array or matrix with complex mask values

Returns
arr – Array or matrix to which the mask is applied

Return type
numpy.ndarray

`utils.calc_grain_size(p, s, percent)`

Calculate grain size characteristics based on mass in each fraction

Calculate grain size distribution for each cell based on weight distribution over the fractions. Interpolates to the requested percentage in the grain size distribution. For example, percent=50 will result in calculation of the D50. Calculation is only executed for the top layer

Parameters

- **s** (*dict*) – Spatial grids
- **p** (*dict*) – Model configuration parameters
- **percent** (*float*) – Requested percentage in grain size dsitribution

Returns
grain size per grid cell

Return type
array

`utils.calc_mean_grain_size(p, s)`

Calculate mean grain size based on mass in each fraction

Calculate mean grain size for each cell based on weight distribution over the fractions. Calculation is only executed for the top layer.

Parameters

- **s** (*dict*) – Spatial grids
- **p** (*dict*) – Model configuration parameters
- **percent** (*float*) – Requested percentage in grain size dsitribution

Returns
mean grain size per grid cell

Return type
array

`utils.format_log(msg, ncolumns=2, **props)`

Format log message into columns

Prints log message and additional data into a column format that fits into a 70 character terminal.

Parameters

- **msg** (*str*) – Main log message
- **ncolumns** (*int*) – Number of columns
- **props** (*key/value pairs*) – Properties to print in column format

Returns
Formatted log message

Return type
str

Note: Properties names starting with `min`, `max` or `nr` are respectively replaced by `min.`, `max.` or `#`.

`utils.interp_array(x, xp, fp, circular=False, **kwargs)`

Interpolate multiple time series at once

Parameters

- **x** (*array_like*) – The x-coordinates of the interpolated values.
- **xp** (*1-D sequence of floats*) – The x-coordinates of the data points, must be increasing.
- **fp** (*2-D sequence of floats*) – The y-coordinates of the data points, same length as `xp`.
- **circular** (*bool*) – Use the `interp_circular()` function rather than the `numpy.interp()` function.
- **kwargs** (*dict*) – Keyword options to the `numpy.interp()` function

Returns
The interpolated values, same length as second dimension of `fp`.

Return type
ndarray

`utils.`**`interp_circular`**(*x*, *xp*, *fp*, ***kwargs*)

One-dimensional linear interpolation.

Returns the one-dimensional piecewise linear interpolant to a function with given values at discrete data-points. Values beyond the limits of `x` are interpolated in circular manner. For example, a value of `x > x.max()` evaluates as `f(x-x.max())` assuming that `x.max() - x < x.max()`.

Parameters

- **x** (*array_like*) – The x-coordinates of the interpolated values.
- **xp** (*1-D sequence of floats*) – The x-coordinates of the data points, must be increasing.
- **fp** (*1-D sequence of floats*) – The y-coordinates of the data points, same length as `xp`.
- **kwargs** (*dict*) – Keyword options to the `numpy.interp()` function

Returns
y – The interpolated values, same shape as `x`.

Return type
{float, ndarray}

Raises
ValueError – If `xp` and `fp` have different length

`utils.`**`isarray`**(*x*)

Check if variable is an array

`utils.`**`isiterable`**(*x*)

Check if variable is iterable

`utils.`**`makeiterable`**(*x*)

Ensure that variable is iterable

`utils.`**`normalize`**(*x*, *ref=None*, *axis=0*, *fill=0.0*)

Normalize array

Normalizes an array to make it sum to unity over a specific axis. The procedure is safe for dimensions that sum to zero. These dimensions return the `fill` value instead.

Parameters

- **x** (*array_like*) – The array to be normalized
- **ref** (*array_like, optional*) – Alternative normalization reference, if not specified, the sum of x is used
- **axis** (*int, optional*) – The normalization axis (default: 0)
- **fill** (*float, optional*) – The return value for all-zero dimensions (default: 0.)

`utils.`**`prevent_tiny_negatives`**(*x*, *max_error=1e-10*, *replacement=0.0*)

Replace tiny negative values in array

Parameters

- **x** (*np.ndarray*) – Array with potential tiny negative values
- **max_error** (*float*) – Maximum absolute value to be replaced
- **replacement** (*float*) – Replacement value

Returns
Array with tiny negative values removed

Return type
np.ndarray

`utils.print_value`(*val*, *fill='<novalue>'*)

Construct a string representation from an arbitrary value

Parameters

- **val** (*misc*) – Value to be represented as string
- **fill** (*str, optional*) – String representation used in case no value is given

Returns
String representation of value

Return type
str

`utils.rotate`(*x*, *y*, *alpha*, *origin=(0, 0)*)

Rotate a matrix over given angle around given origin

Sierd's favorite function is: `aeolis.bed.prevent_tiny_negatives`

1.4 Input files

The computational grid and boundary conditions for AeoLiS are specified through external input files called by the model parameter file aeolis.txt. The computational grid is defined with an x grid, y grid, and z grid. Boundary conditions for wind, wave, and tides are also specified with external text files. A list of additional grid and boundary files can be found in the table below. Each file is further defined below.

Input File	File Description
aeolis.txt	File containing parameter definitions
x.grd	File containing cross-shore grid
y.grd	File containing alongshore grid (can be all zeros for 1D cases)
z.grd	File containing topography and bathymetry data
veg.grd	File containing initial vegetation density
mass.txt	File containing sediment mass data when using space varying grain size distribution
wind.txt	File containing wind speed and direction data
tide.txt	File containing water elevation data
wave.txt	File containing wave height and period data
meteo.txt	File containing meteorological time series data

1.4.1 aeolis.txt

This is the parameter file for AeoLiS that defines the model processes and boundary conditions. Parameters in the file are specified by various keywords; each keyword has a pre-defined default value that will be used if it is not direclty specified in aeolis.txt (a list of default parameter values can be found in the Default settings tab on the left). Among the keywords in aeolis.txt are the keywords to define the external computational grid files (xgrid_file, ygrid_file, and bed_file) and external boundary condition files (tide_file, wave_file, wind_file, etc.). The different physical processes in AeoLiS can be turned on and off by changing the process keywords in aeolis.txt to T (True) and F (False). Example aeolis.txt parameters files can be found in the examples folder on the AeoLiS GitHub.

1.4.2 x.grd

The x.grd file defines the computational grid in the cross-shore direction defined in meters. In a 1-dimensional (1D) case, the file contains a single column of cross-shore locations starting at zero for a location of choice. In a 2-dimesional (2D) case, the file contains multiple columns (cross-shore positions) and rows (alongshore positions) where each value corresponds to a specific location in the 2D grid. The file can be renamed and is referenced from the parameters file with the xgrid_file keyword.

1.4.3 y.grd

This file defines the computational grid in the alongshore direction. In a 1D case, y.grd will contain a single column of zeros. In a 2D case, similar to the x.grd file, y.grd has multiple columns (cross-shore positions) and rows (alongshore positions) where each row, column position corresponds to a specific location in the 2D gird. x.grd and y.grd will always be the same size regardless of running a 1D or 2D simulation. As with the x.grd file, this file can be renamed and is referenced from the parameters file with the keyword: ygrid_file.

1.4.4 z.grd

The z.grd file provides the model with the elevation information for the computational grid defined in x.grd and y.grd. Similar to x.grd and y.grd, when running AeoLis in 1D the file contains a single column with the number of rows equal to the number of rows in x.grd and y.grd. In 2D cases, z.grd has multiple columns and rows of equal size to x.grd and y.grd. Elevation values in the file should be defined such that positive is up and negative is down. The file can be renamed and is referenced from the parameters file with the keyword: bed_file.

1.4.5 veg.grd

The veg.grd file is an optional grid providing initial vegetation coverage (density) at each position in the model domain defined in x.grd and y.grd. Similar to the grid files, if simulations are in 2D there will be multiple columns for each cross-shore position (x) and multiple rows for each alongshore position (y). The format of a 1D vegetation grid file can be seen below where each red dots represent vegetation cover at each cross-shore position.

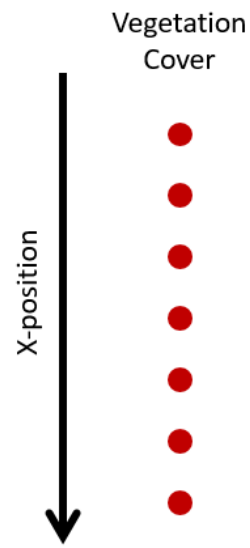

Fig. 1.5: File format for a 1D AeoLis vegetation grid. Each red dot is the vegetation density at a specific location in the computational grid.

1.4.6 mass.txt

The mass.txt file allows users to specify variations in grain size distribution in both horizontal and vertical directions. If the grain size distribution is constant throughout the model domain, multifraction sediment transport is possilbe without this file. The file contains the mass of each sediment fraction in each grid cell and bed layer. The file is formatted such that each row corresponds to a specific location in the computational domain and the columns are grouped by bed layers and each individual column represents a single sediment fraction present in the model domain. An infinite number of sediment fractions can be defined in the model; however, it should be noted the more sediment fractions present the longer the simulation time and larger the output files.

In a 1D case, the text file will have dimensions of number of cross-shore locations (x) by number of sediment fractions times the number of bed layers. For example if you have 200 cross-shore positions in your model domain and 4 different sediment fractions with 3 bed layers, your mass.txt file will contain a matrix of 200 rows by 12 columns. An example of a 1D mass.txt file can be seen below where each red dot represents a sediment fraction mass at a specific location in the model domain.

In a 2D case, the mass.txt file will have dimensions of number of cross-shore positions (x) times the number of alongshore positions (x) by number of sediment fractions times the number of bed layers. The file will be formatted such that the columns are grouped by bed layer with all available sediment fractions present in each bed layer and rows are grouped by alongshore position with all cross-shore prositions given for each alongshore position. An visual example of a 2D mass.txt input file for AeoLis can be seen below.

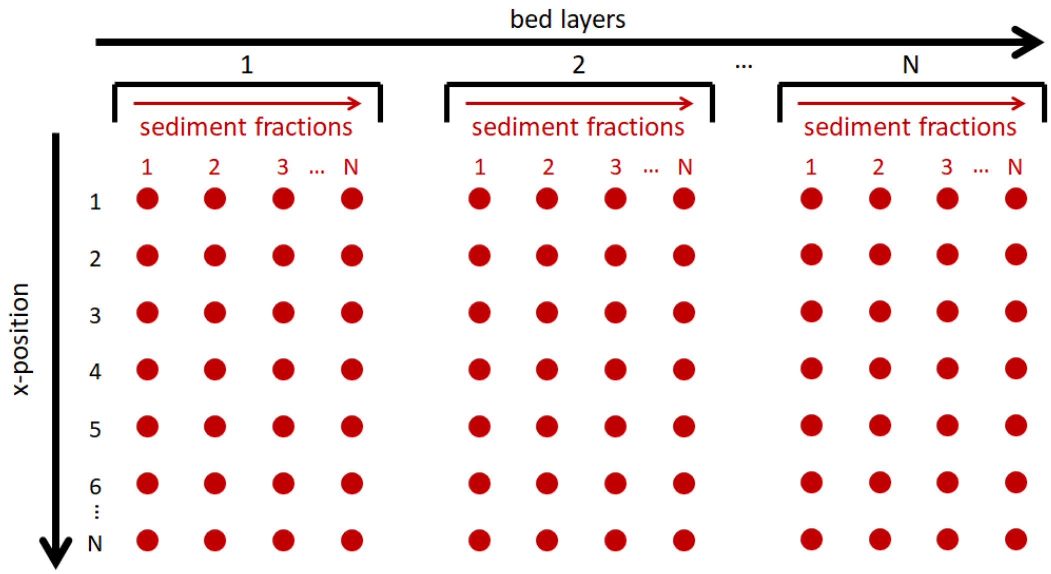

Fig. 1.6: File format for a 1D AeoLis mass for spatially variable grain size distributions. Each red dot is the mass for each sediment fraction at each location in the computational grid (x, y, bed layer).

1.4.7 wind.txt

The wind.txt file provides the model with wind boundary conditions and is formatted similar to the tide.txt and wave.txt files. The first column is time in seconds from start, the second column is wind speed, and the third column is wind direction. The wind directions can be specified in either nautical or cartesian convention (specified in aeolis.txt with keyword: wind_convention). The format of this file can be seen below were each of the red dots represents a data value of time, wind speed, or wind direction. As AeoLiS is an aeolian sediment transport model, the wind boundary conditions are of particular importance.

1.4.8 tide.txt

The tide.txt file contains the water elevation data for the duration of the simulation. It is formatted such that the first column is time in seconds and the second column is the water elevation data at each time step. An example of the file format can be seen below where each red dot represents a data value for time or water elevation.

1.4.9 wave.txt

The wave.txt file provides the model with wave data used in AeoLiS for runup calculations. The file is formatted similar to tide.txt but has three columns instead of two. Here, the first column is time in seconds, the second column is wave height, and the third column is the wave period. The format of this file can be seen below where each red dot represents a data value.

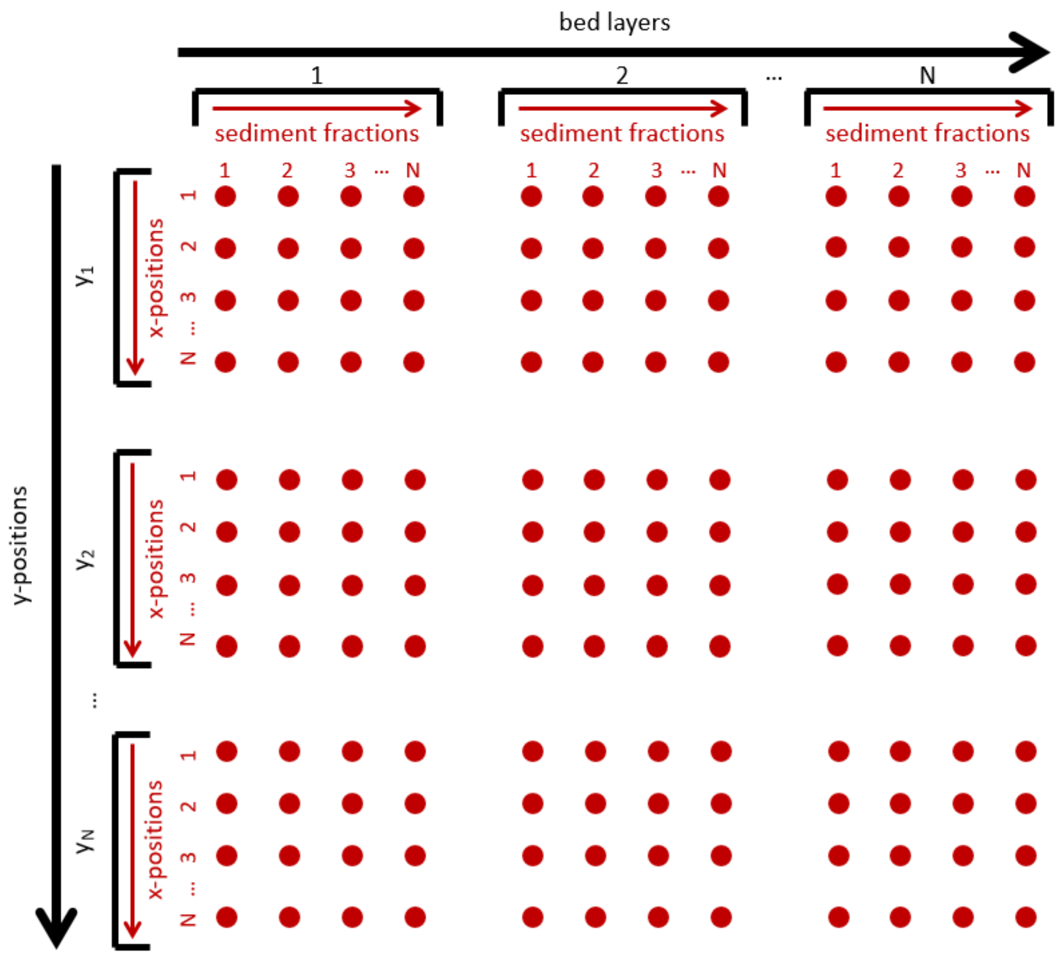

Fig. 1.7: File format for a 2D AeoLis mass file for spatially variable grain size distributions. Each red dot is the mass for each sediment fraction at each location in the computational grid (x, y, bed layer).

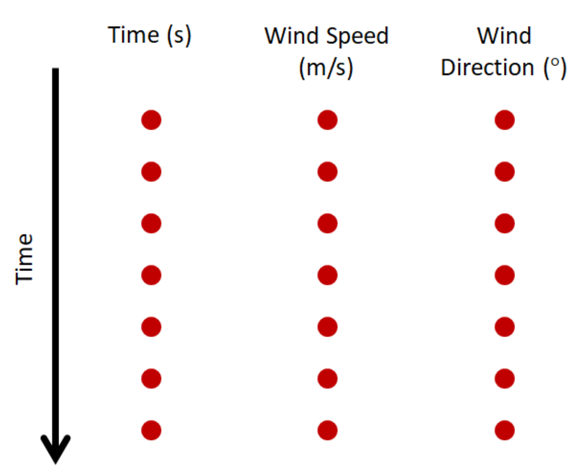

Fig. 1.8: File format for wind boundary conditions file for AeoLis input.

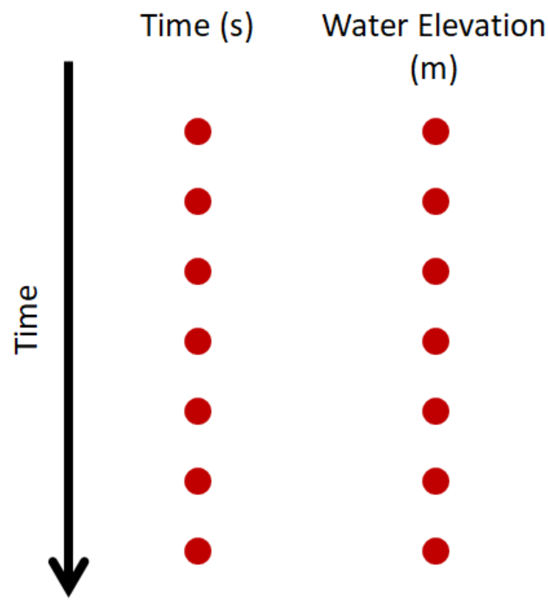

Fig. 1.9: File format for the water elevation conditions file for AeoLis input.

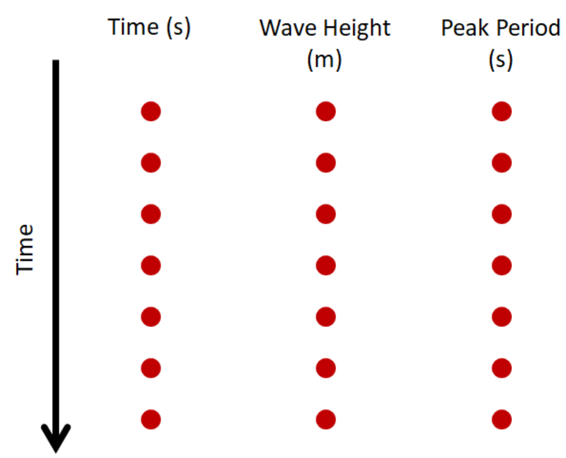

Fig. 1.10: File format for the wave conditions file for AeoLis input.

1.4.10 meteo.txt

The meteo.txt file contains meteorological data used to simulate surface moisture in the model domain (see Simulation of surface moisture in Model description on for surface moisture implementation in AeoLiS). This file is formatted similar to the other environmental boundary condition files (wind, wave, and tide) such that it contains a time series of environmental data read into AeoLiS through keyword specification. The keywords required to process surface moisture with evaporation and infiltration are process_moist = True, method_moist_process = surf_moisture, th_moisture = True, and meteo_file = meteo.txt (or name of file containing meteorological data). An example of the meteo.txt file can be seen in the figure below where each red dot represents a time series data value. The first column contains time (s), the second column is temperature (degrees C), the thrid column is precipitation (mm/hr), the fourth column is relative humidity (%), the fifth column is global radiation (MJ/m^2/day), and the sixth column is air pressure (kPa).

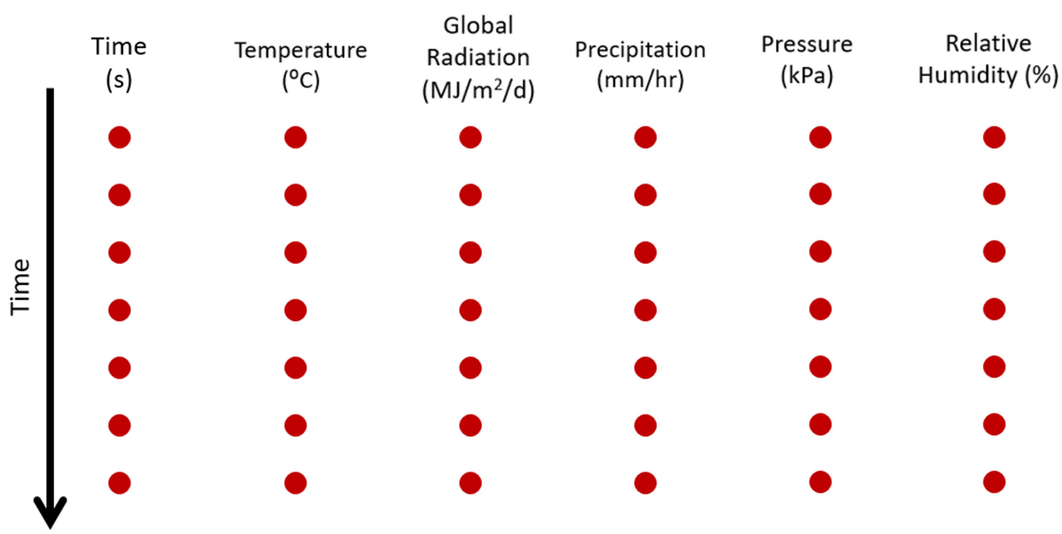

Fig. 1.11: File format for meteorological data used to simulate surface moisture in AeoLiS where each red dot represents a time series value.

1.5 Default settings

The AeoLiS model can be configured using a model configuration file. For any configuration parameters not defined in the model configuration file, or in case the model configuration file is absent, the default model configuration is used. The default model configuration is listed below.

```
DEFAULT_CONFIG = {
    'process_wind'                  : True,             # Enable the process of wind
    'process_transport'             : True,             # Enable the process of␣
↪transport
    'process_bedupdate'             : True,             # Enable the process of bed␣
↪updating
    'process_threshold'             : True,             # Enable the process of␣
↪threshold
    'th_grainsize'                  : True,             # Enable wind velocity␣
↪threshold based on grainsize
    'th_bedslope'                   : False,            # Enable wind velocity␣
↪threshold based on bedslope
```

(continues on next page)

(continued from previous page)

```
    'th_moisture'                  : False,              # Enable wind velocity␣
↪threshold based on moisture
    'th_drylayer'                  : False,             # Enable threshold based on␣
↪drying of layer
    'th_humidity'                  : False,             # Enable wind velocity␣
↪threshold based on humidity
    'th_salt'                      : False,             # Enable wind velocity␣
↪threshold based on salt
    'th_sheltering'                : False,              # Enable wind velocity␣
↪threshold based on sheltering by roughness elements
    'th_nelayer'                   : False,             # Enable wind velocity␣
↪threshold based on a non-erodible layer
    'process_avalanche'            : False,              # Enable the process of␣
↪avalanching
    'process_shear'                : False,              # Enable the process of wind␣
↪shear
    'process_tide'                 : False,              # Enable the process of tides
    'process_wave'                 : False,              # Enable the process of waves
    'process_runup'                : False,              # Enable the process of wave␣
↪runup
    'process_moist'                : False,              # Enable the process of moist
    'process_mixtoplayer'          : False,              # Enable the process of␣
↪mixing
    'process_wet_bed_reset'        : False,              # Enable the process of bed-
↪reset in the intertidal zone
    'process_meteo'                : False,             # Enable the process of meteo
    'process_salt'                 : False,             # Enable the process of salt
    'process_humidity'             : False,             # Enable the process of␣
↪humidity
    'process_groundwater'          : False,       #NEWCH      # Enable the process of␣
↪groundwater
    'process_scanning'             : False,        #NEWCH      # Enable the process of␣
↪scanning curves
    'process_inertia'              : False,             # NEW
    'process_separation'           : False,              # Enable the including of␣
↪separation bubble
    'process_vegetation'           : False,             # Enable the process of␣
↪vegetation
    'process_fences'               : False,             # Enable the process of sand␣
↪fencing
    'process_dune_erosion'         : False,             # Enable the process of wave-
↪driven dune erosion
    'visualization'                : False,              # Boolean for visualization␣
↪of model interpretation before and just after initialization
    'xgrid_file'                   : None,              # Filename of ASCII file with␣
↪x-coordinates of grid cells
    'ygrid_file'                   : None,              # Filename of ASCII file with␣
↪y-coordinates of grid cells
    'bed_file'                     : None,              # Filename of ASCII file with␣
↪bed level heights of grid cells
    'wind_file'                    : None,              # Filename of ASCII file with␣
↪time series of wind velocity and direction
```

(continues on next page)

(continued from previous page)

```
    'tide_file'                      : None,              # Filename of ASCII file with␣
↪time series of water levels
    'wave_file'                      : None,              # Filename of ASCII file with␣
↪time series of wave heights
    'meteo_file'                     : None,              # Filename of ASCII file with␣
↪time series of meteorlogical conditions
    'bedcomp_file'                   : None,              # Filename of ASCII file with␣
↪initial bed composition
    'threshold_file'                 : None,              # Filename of ASCII file with␣
↪shear velocity threshold
    'fence_file'                     : None,              # Filename of ASCII file with␣
↪sand fence location/height (above the bed)
    'ne_file'                        : None,              # Filename of ASCII file with␣
↪non-erodible layer
    'veg_file'                       : None,              # Filename of ASCII file with␣
↪initial vegetation density
    'wave_mask'                      : None,              # Filename of ASCII file with␣
↪mask for wave height
    'tide_mask'                      : None,              # Filename of ASCII file with␣
↪mask for tidal elevation
    'runup_mask'                     : None,              # Filename of ASCII file with␣
↪mask for run-up
    'threshold_mask'                 : None,              # Filename of ASCII file with␣
↪mask for the shear velocity threshold
    'gw_mask'                        : None,        #NEWCH      # Filename of ASCII file␣
↪with mask for the groundwater level
    'nx'                             : 0,                 # [-] Number of grid cells in␣
↪x-dimension
    'ny'                             : 0,                 # [-] Number of grid cells in␣
↪y-dimension
    'dt'                             : 60.,               # [s] Time step size
    'dx'                             : 1.,
    'dy'                             : 1.,
    'CFL'                            : 1.,                # [-] CFL number to determine␣
↪time step in explicit scheme
    'accfac'                         : 1.,                # [-] Numerical acceleration␣
↪factor
    'max_bedlevel_change'            : 999.,              # [m] Maximum bedlevel change␣
↪after one timestep. Next timestep dt will be modified (use 999. if not used)
    'tstart'                         : 0.,                # [s] Start time of simulation
    'tstop'                          : 3600.,             # [s] End time of simulation
    'restart'                        : None,              # [s] Interval for which to␣
↪write restart files
    'dzb_interval'                   : 86400,             # [s] Interval used for␣
↪calcuation of vegetation growth
    'output_times'                   : 60.,               # [s] Output interval in␣
↪seconds of simulation time
    'output_file'                    : None,              # Filename of netCDF4 output␣
↪file
    'output_vars'                    : ['zb', 'zs',
                                        'Ct', 'Cu',
                                        'uw', 'udir',
```

(continues on next page)

(continued from previous page)

```
                                       'uth', 'mass'
                                       'pickup', 'w'],     # Names of spatial grids to be␣
↪included in output
    'output_types'                : [],                    # Names of statistical␣
↪parameters to be included in output (avg, sum, var, min or max)
    'external_vars'               : [],                    # Names of variables that are␣
↪overwritten by an external (coupling) model, i.e. CoCoNuT
    'grain_size'                  : [225e-6],              # [m] Average grain size of␣
↪each sediment fraction
    'grain_dist'                  : [1.],                  # [-] Initial distribution of␣
↪sediment fractions
    'nlayers'                     : 3,                     # [-] Number of bed layers
    'layer_thickness'             : .01,                   # [m] Thickness of bed layers
    'g'                           : 9.81,                  # [m/s^2] Gravitational␣
↪constant
    'v'                           : 0.000015,              # [m^2/s] Air viscosity
    'rhoa'                        : 1.225,                 # [kg/m^3] Air density
    'rhog'                        : 2650.,                 # [kg/m^3] Grain density
    'rhow'                        : 1025.,                 # [kg/m^3] Water density
    'porosity'                    : .4,                    # [-] Sediment porosity
    'Aa'                          : .085,                  # [-] Constant in formulation␣
↪for wind velocity threshold based on grain size
    'z'                           : 10.,                   # [m] Measurement height of␣
↪wind velocity
    'h'                           : None,                  # [m] Representative height of␣
↪saltation layer
    'k'                           : 0.001,                 # [m] Bed roughness
    'L'                           : 100.,                  # [m] Typical length scale of␣
↪dune feature (perturbation)
    'l'                           : 10.,                   # [m] Inner layer height␣
↪(perturbation)
    'c_b'                         : 0.2,                   # [-] Slope at the leeside of␣
↪the separation bubble # c = 0.2 according to Durán 2010 (Sauermann 2001: c = 0.25 for␣
↪14 degrees)
    'mu_b'                        : 30,                    # [deg] Minimum required slope␣
↪for the start of flow separation
    'buffer_width'                : 10,                    # [m] Width of the bufferzone␣
↪around the rotational grid for wind perturbation
    'sep_filter_iterations'       : 0,                     # [-] Number of filtering␣
↪iterations on the sep-bubble (0 = no filtering)
    'zsep_y_filter'               : False,                 # [-] Boolean for turning on/
↪off the filtering of the separation bubble in y-direction
    'Cb'                          : 1.5,                   # [-] Constant in bagnold␣
↪formulation for equilibrium sediment concentration
    'Ck'                          : 2.78,                  # [-] Constant in kawamura␣
↪formulation for equilibrium sediment concentration
    'Cl'                          : 6.7,                   # [-] Constant in lettau␣
↪formulation for equilibrium sediment concentration
    'Cdk'                         : 5.,                    # [-] Constant in DK␣
↪formulation for equilibrium sediment concentration
    # 'm'                          : 0.5,                   # [-] Factor to account for␣
↪difference between average and maximum shear stress
```

(continues on next page)

(continued from previous page)

```
#    'alpha'                          : 0.4,              # [-] Relation of vertical␣
↪component of ejection velocity and horizontal velocity difference between impact and␣
↪ejection
    'kappa'                          : 0.41,              # [-] Von Kármán constant
    'sigma'                          : 4.2,               # [-] Ratio between basal area␣
↪and frontal area of roughness elements
    'beta'                           : 130.,              # [-] Ratio between drag␣
↪coefficient of roughness elements and bare surface
    'bi'                             : 1.,                # [-] Bed interaction factor
    'T'                              : 1.,                # [s] Adaptation time scale in␣
↪advection equation
    'Tdry'                           : 3600.*1.5,         # [s] Adaptation time scale␣
↪for soil drying
    'Tsalt'                          : 3600.*24.*30.,     # [s] Adaptation time scale␣
↪for salinitation
    'Tbedreset'                      : 86400.,            # [s]
    'eps'                            : 1e-3,              # [m] Minimum water depth to␣
↪consider a cell "flooded"
    'gamma'                          : .5,                # [-] Maximum wave height over␣
↪depth ratio
    'xi'                             : .3,                # [-] Surf similarity parameter
    'facDOD'                         : .1,                # [-] Ratio between depth of␣
↪disturbance and local wave height
    'csalt'                          : 35e-3,             # [-] Maximum salt␣
↪concentration in bed surface layer
    'cpair'                          : 1.0035e-3,         # [MJ/kg/oC] Specific heat␣
↪capacity air

    'fc'                             : 0.11,        # NEWCH      # [-] Moisture content␣
↪at field capacity (volumetric)
    'w1_5'                           : 0.02,        # NEWCH      # [-] Moisture content␣
↪at wilting point (gravimetric)
    'resw_moist'                     : 0.01,        # NEWCH      # [-] Residual soil␣
↪moisture content (volumetric)
    'satw_moist'                     : 0.35,        # NEWCH      # [-] Satiated soil␣
↪moisture content (volumetric)
    'resd_moist'                     : 0.01,        # NEWCH      # [-] Residual soil␣
↪moisture content (volumetric)
    'satd_moist'                     : 0.5,         # NEWCH      # [-] Satiated soil␣
↪moisture content (volumetric)
    'nw_moist'                       : 2.3,         # NEWCH      # [-] Pore-size␣
↪distribution index in the soil water retention function
    'nd_moist'                       : 4.5,         # NEWCH      # [-] Pore-size␣
↪distribution index in the soil water retention function
    'mw_moist'                       : 0.57,        # NEWCH      # [-] m, van Genucthen␣
↪param (can be approximated as 1-1/n)
    'md_moist'                       : 0.42,        # NEWCH      # [-] m, van Genucthen␣
↪param (can be approximated as 1-1/n)
    'alfaw_moist'                    : -0.070,      # NEWCH      # [cm^-1] Inverse of␣
↪the air-entry value for a wetting branch of the soil water retention function (Schmutz,
↪ 2014)
    'alfad_moist'                    : -0.035,      # NEWCH      # [cm^-1] Inverse of␣
```

(continues on next page)

(continued from previous page)

```
↪the air-entry value for a drying branch of the soil water retention function (Schmutz,␣
↪2014)
    'thick_moist'                    : 0.002,        # NEWCH      # [m] Thickness of␣
↪surface moisture soil layer
    'K_gw'                           : 0.00078,      # NEWCH      # [m/s] Hydraulic␣
↪conductivity (Schmutz, 2014)
    'ne_gw'                          : 0.3,          # NEWCH      # [-] Effective porosity
    'D_gw'                           : 12,           # NEWCH      # [m] Aquifer depth
    'tfac_gw'                        : 10,           # NEWCH      # [-] Reduction factor␣
↪for time step in ground water calculations
    'Cl_gw'                          : 0.7,          # NEWCH      # [m] Groundwater␣
↪overheight due to runup
    'in_gw'                          : 0,            # NEWCH      # [m] Initial␣
↪groundwater level
    'GW_stat'                        : 1,            # NEWCH      # [m] Landward static␣
↪groundwater boundary (if static boundary is defined)
    'theta_dyn'                      : 33.,                 # [degrees] Initial Dynamic␣
↪angle of repose, critical dynamic slope for avalanching
    'theta_stat'                     : 34.,                 # [degrees] Initial Static␣
↪angle of repose, critical static slope for avalanching
    'avg_time'                       : 86400.,              # [s] Indication of the time␣
↪period over which the bed level change is averaged for vegetation growth
    'gamma_vegshear'                 : 16.,                 # [-] Roughness factor for the␣
↪shear stress reduction by vegetation
    'hveg_max'                       : 1.,                  # [m] Max height of vegetation
    'dzb_opt'                        : 0.,                  # [m/year] Sediment burial for␣
↪optimal growth
    'V_ver'                          : 0.,                  # [m/year] Vertical growth
    'V_lat'                          : 0.,                  # [m/year] Lateral growth
    'germinate'                      : 0.,                  # [1/year] Possibility of␣
↪germination per year
    'lateral'                        : 0.,                  # [1/year] Posibility of␣
↪lateral expension per year
    'veg_gamma'                      : 1.,                  # [-] Constant on influence of␣
↪sediment burial
    'veg_sigma'                      : 0.8,                 # [-] Sigma in gaussian␣
↪distrubtion of vegetation cover filter
    'sedimentinput'                  : 0.,                  # [-] Constant boundary␣
↪sediment influx (only used in solve_pieter)
    'scheme'                         : 'euler_backward',    # Name of numerical scheme␣
↪(euler_forward, euler_backward or crank_nicolson)
    'solver'                         : 'trunk',              # Name of the solver (trunk,␣
↪pieter, steadystate,steadystatepieter)
    'boundary_lateral'               : 'circular',          # Name of lateral boundary␣
↪conditions (circular, constant ==noflux)
    'boundary_offshore'              : 'constant',          # Name of offshore boundary␣
↪conditions (flux, constant, uniform, gradient)
    'boundary_onshore'               : 'gradient',          # Name of onshore boundary␣
↪conditions (flux, constant, uniform, gradient)
    'boundary_gw'                    : 'no_flow',           # Landward groundwater␣
↪boundary, dGw/dx = 0 (or 'static')
    'method_moist_threshold'         : 'belly_johnson',     # Name of method to compute␣
```

(continues on next page)

(continued from previous page)

```
↪wind velocity threshold based on soil moisture content
    'method_moist_process'          : 'infiltration',     # Name of method to compute␣
↪soil moisture content(infiltration or surface_moisture)
    'offshore_flux'                 : 0.,                 # [-] Factor to determine␣
↪offshore boundary flux as a function of Q0 (= 1 for saturated flux , = 0 for noflux)
    'constant_offshore_flux'        : 0.,                 # [kg/m/s] Constant input flux␣
↪at offshore boundary
    'onshore_flux'                  : 0.,                 # [-] Factor to determine␣
↪onshore boundary flux as a function of Q0 (= 1 for saturated flux , = 0 for noflux)
    'constant_onshore_flux'         : 0.,                 # [kg/m/s] Constant input flux␣
↪at offshore boundary
    'lateral_flux'                  : 0.,                 # [-] Factor to determine␣
↪lateral boundary flux as a function of Q0 (= 1 for saturated flux , = 0 for noflux)
    'method_transport'              : 'bagnold',          # Name of method to compute␣
↪equilibrium sediment transport rate
    'method_roughness'              : 'constant',         # Name of method to compute␣
↪the roughness height z0, note that here the z0 = k, which does not follow the␣
↪definition of Nikuradse where z0 = k/30.
    'method_grainspeed'             : 'windspeed',        # Name of method to assume/
↪compute grainspeed (windspeed, duran, constant)
    'max_error'                     : 1e-6,               # [-] Maximum error at which␣
↪to quit iterative solution in implicit numerical schemes
    'max_iter'                      : 1000,               # [-] Maximum number of␣
↪iterations at which to quit iterative solution in implicit numerical schemes
    'max_iter_ava'                  : 1000,               # [-] Maximum number of␣
↪iterations at which to quit iterative solution in avalanching calculation
    'refdate'                       : '2020-01-01 00:00', # [-] Reference datetime in␣
↪netCDF output
    'callback'                      : None,               # Reference to callback␣
↪function (e.g. example/callback.py':callback)
    'wind_convention'               : 'nautical',         # Convention used for the wind␣
↪direction in the input files (cartesian or nautical)
    'alfa'                          : 0,                  # [deg] Real-world grid cell␣
↪orientation wrt the North (clockwise)
    'dune_toe_elevation'            : 3,                  # Choose dune toe elevation,␣
↪only used in the PH12 dune erosion solver
    'beach_slope'                   : 0.1,                # Define the beach slope, only␣
↪used in the PH12 dune erosion solver
    'veg_min_elevation'             : 3,                  # Choose the minimum elevation␣
↪where vegetation can grow
    'vegshear_type'                 : 'raupach',          # Choose the Raupach grid␣
↪based solver (1D or 2D) or the Okin approach (1D only)
    'okin_c1_veg'                   : 0.48,               #x/h spatial reduction factor␣
↪in Okin model for use with vegetation
    'okin_c1_fence'                 : 0.48,               #x/h spatial reduction factor␣
↪in Okin model for use with sand fence module
    'okin_initialred_veg'           : 0.32,               #initial shear reduction␣
↪factor in Okin model for use with vegetation
    'okin_initialred_fence'         : 0.32,               #initial shear reduction␣
↪factor in Okin model for use with sand fence module
    'veggrowth_type'                : 'orig',     #'orig', 'duranmoore14'
    'rhoveg_max'                    : 0.5,                #maximum vegetation density,␣
```

(continues on next page)

(continued from previous page)

```
↪only used in duran and moore 14 formulation
    't_veg'                       : 3,                  #time scale of vegetation␣
↪growth (days), only used in duran and moore 14 formulation
    'v_gam'                       : 1,                  # only used in duran and moore␣
↪14 formulation
}

REQUIRED_CONFIG = ['nx', 'ny']
```

1.6 Model state/output

The AeoLiS model state is described by a collection of spatial grid variables with at least one value per horizontal grid cell. Specific model state variables can also be subdivided over bed composition layers and/or grain size fractions. All model state variables can be part of the model netCDF4 output. The current model state variables are listed below.

```
INITIAL_STATE = {
    ('ny', 'nx') : (
        'uw',                             # [m/s] Wind velocity
        'uws',                            # [m/s] Component of wind velocity in x-
↪direction
        'uwn',                            # [m/s] Component of wind velocity in y-
↪direction

        'tau',                            # [N/m^2] Wind shear stress
        'taus',                           # [N/m^2] Component of wind shear stress in␣
↪x-direction
        'taun',                           # [N/m^2] Component of wind shear stress in␣
↪y-direction
        'tau0',                           # [N/m^2] Wind shear stress over a flat bed
        'taus0',                          # [N/m^2] Component of wind shear stress in␣
↪x-direction over a flat bed
        'taun0',                          # [N/m^2] Component of wind shear stress in␣
↪y-direction over a flat bed
        'taus_u',                         # [N/m^2] Saved direction of wind shear␣
↪stress in x-direction
        'taun_u',                         # [N/m^2] Saved direction of wind shear␣
↪stress in y-direction
        'dtaus',                          # [-] Component of the wind shear␣
↪perturbation in x-direction
        'dtaun',                          # [-] Component of the wind shear␣
↪perturbation in y-direction

        'ustar',                          # [m/s] Wind shear velocity
        'ustars',                         # [m/s] Component of wind shear velocity in␣
↪x-direction
        'ustarn',                         # [m/s] Component of wind shear velocity in␣
↪y-direction
        'ustar0',                         # [m/s] Wind shear velocity over a flat bed
        'ustars0',                        # [m/s] Component of wind shear velocity in␣
```

(continues on next page)

(continued from previous page)

```
↪x-direction over a flat bed
        'ustarn0',                          # [m/s] Component of wind shear velocity in␣
↪y-direction over a flat bed

        'udir',                             # [rad] Wind direction
        'zs',                               # [m] Water level above reference (or equal␣
↪to zb if zb > zs)
        'SWL',                              # [m] Still water level above reference
        'Hs',                               # [m] Wave height
        'Hsmix',                            # [m] Wave height for mixing (including␣
↪setup, TWL)
        'Tp',                               # [s] Wave period for wave runup calculations
        'zne',                              # [m] Non-erodible layer
    ),
}

MODEL_STATE = {
    ('ny', 'nx') : (
        'x',                                # [m] Real-world x-coordinate of grid cell␣
↪center
        'y',                                # [m] Real-world y-coordinate of grid cell␣
↪center
        'ds',                               # [m] Real-world grid cell size in x-
↪direction
        'dn',                               # [m] Real-world grid cell size in y-
↪direction
        'dsdn',                             # [m^2] Real-world grid cell surface area
        'dsdni',                            # [m^-2] Inverse of real-world grid cell␣
↪surface area
#        'alfa',                            # [rad] Real-world grid cell orientation
↪#Sierd_comm in later releases this needs a revision
        'zb',                               # [m] Bed level above reference
        'zs',                               # [m] Water level above reference
        'zne',                              # [m] Height above reference of the non-
↪erodible layer
        'zb0',                              # [m] Initial bed level above reference
        'zdry',                             # [m]
        'dzdry',                            # [m]
        'dzb',                              # [m/dt] Bed level change per time step␣
↪(computed after avalanching!)
        'dzbyear',                          # [m/yr] Bed level change translated to m/y
        'dzbavg',                           # [m/year] Bed level change averaged over␣
↪collected time steps
        'S',                                # [-] Level of saturation
        'moist',                      #NEWCH      # [-] Moisture content (volumetric)
        'moist_swr',                  #NEWCH      # [-] Moisture content soil water␣
↪retention relationship (volumetric)
        'h_delta',                    #NEWCH      # [-] Suction at reversal between␣
↪wetting/drying conditions
        'gw',                         #NEWCH      # [m] Groundwater level above reference
        'gw_prev',                    #NEWCH      # [m] Groundwater level above␣
↪reference in previous timestep
```

(continues on next page)

(continued from previous page)

```
        'wetting',                    #NEWCH      # [bool] Flag indicating wetting or␣
↪drying of soil profile
        'scan_w',                     #NEWCH      # [bool] Flag indicating that the␣
↪moisture is calculated on the wetting scanning curve
        'scan_d',                     #NEWCH      # [bool] Flag indicating that the␣
↪moisture is calculated on the drying scanning curve
        'scan_w_moist',               #NEWCH      # [-] Moisture content (volumetric)␣
↪computed on the wetting scanning curve
        'scan_d_moist',               #NEWCH      # [-] Moisture content (volumetric)␣
↪computed on the drying scanning curve
        'w_h',                        #NEWCH      # [-] Moisture content (volumetric)␣
↪computed on the main wetting curve
        'd_h',                        #NEWCH      # [-] Moisture content (volumetric)␣
↪computed on the main drying curve
        'w_hdelta',                   #NEWCH      # [-] Moisture content (volumetric)␣
↪computed on the main wetting curve for hdelta
        'd_hdelta',                    #NEWCH      # [-] Moisture content (volumetric)␣
↪computed on the main drying curve for hdelta
        'ustar',                            # [m/s] Shear velocity by wind
        'ustars',                           # [m/s] Component of shear velocity in x-
↪direction by wind
        'ustarn',                           # [m/s] Component of shear velocity in y-
↪direction by wind
        'ustar0',                           # [m/s] Initial shear velocity (without␣
↪perturbation)
        'zsep',                             # [m] Z level of polynomial that defines the␣
↪separation bubble
        'hsep',                             # [m] Height of separation bubbel =␣
↪difference between z-level of zsep and of the bed level zb
        'theta_stat',                       # [degrees] Updated, spatially varying␣
↪static angle of repose
        'theta_dyn',                        # [degrees] Updated, spatially varying␣
↪dynamic angle of repose
        'rhoveg',                           # [-] Vegetation cover
        'drhoveg',                          # Change in vegetation cover
        'hveg',                             # [m] height of vegetation
        'dhveg',                            # [m] Difference in vegetation height per␣
↪time step
        'dzbveg',                           # [m] Bed level change used for calculation␣
↪of vegetation growth
        'germinate',                        # vegetation germination
        'lateral',                          # vegetation lateral expansion
        'vegfac',                           # Vegetation factor to modify shear stress␣
↪by according to Raupach 1993
        'R',                                # [m] wave runup
        'eta',                              # [m] wave setup
        'sigma_s',                          # [m] swash
        'TWL',                              # [m] Total Water Level above reference (SWL␣
↪+ Run-up)
        'SWL',                              # [m] Still Water Level above reference
        'DSWL',                             # [m] Dynamic Still water level above␣
↪reference (SWL + Set-up)
```

(continues on next page)

(continued from previous page)

```
        'Rti',                          # [-] Factor taking into account sheltering␣
↪by roughness elements
    ),
    ('ny','nx','nfractions') : (
        'Cu',                           # [kg/m^2] Equilibrium sediment␣
↪concentration integrated over saltation height
        'Cuf',                          # [kg/m^2] Equilibrium sediment␣
↪concentration integrated over saltation height, assuming the fluid shear velocity␣
↪threshold
        'Cu0',                          # [kg/m^2] Flat bad equilibrium sediment␣
↪concentration integrated over saltation height
        'Ct',                           # [kg/m^2] Instantaneous sediment␣
↪concentration integrated over saltation height
        'q',                            # [kg/m/s] Instantaneous sediment flux
        'qs',                           # [kg/m/s] Instantaneous sediment flux in x-
↪direction
        'qn',                           # [kg/m/s] Instantaneous sediment flux in y-
↪direction
        'pickup',                       # [kg/m^2] Sediment entrainment
        'w',                            # [-] Weights of sediment fractions
        'w_init',                       # [-] Initial guess for ``w''
        'w_air',                        # [-] Weights of sediment fractions based on␣
↪grain size distribution in the air
        'w_bed',                        # [-] Weights of sediment fractions based on␣
↪grain size distribution in the bed
        'uth',                          # [m/s] Shear velocity threshold
        'uthf',                         # [m/s] Fluid shear velocity threshold
        'uth0',                         # [m/s] Shear velocity threshold based on␣
↪grainsize only (aerodynamic entrainment)
        'u',                            # [m/s] Mean horizontal saltation velocity␣
↪in saturated state
        'us',                           # [m/s] Component of the saltation velocity␣
↪in x-direction
        'un',                           # [m/s] Component of the saltation velocity␣
↪in y-direction
        'u0',
    ),
    ('ny','nx','nlayers') : (
        'thlyr',                        # [m] Bed composition layer thickness
        'salt',                         # [-] Salt content
    ),
    ('ny','nx','nlayers','nfractions') : (
        'mass',                         # [kg/m^2] Sediment mass in bed
    ),
}
```

1.7 Installation

1.7.1 Requirements

Python packages

- bmi-python: http://github.com/openearth/bmi-python
- numpy
- scipy
- netCDF4
- docopt

External libraries (Windows)

These libraries are needed on Windows if the Python package netCDF4 is installed manually.

- Microsoft Visual C++ Compiler for Python 2.7: http://aka.ms/vcpython27
- msinttypes for stdint.h: https://code.google.com/archive/p/msinttypes/
- HDF5 headers: https://www.hdfgroup.org/HDF5/release/obtain5.html
- netCDF4 headers: https://github.com/Unidata/netcdf-c/releases
- Set environment variables HDF5_DIR and NETCDF_DIR to the respective installation paths

1.8 What's New

1.8.1 v2.0.0 (April 2022)

Breaking changes

- New vegetation growth/expansion capabilities (Bart Van Westen)
- Addition of groundwater module and new moisture routines (Caroline Hallin)
- Incorporation of Okin (2008) vegetation shear coupler (Nick Cohn)
- Addition of Palmsten and Holman (2012) dune erosion module (Nick Cohn)
- Approach to add sand fences into model (Nick Cohn)

Improvements

- Replacement of wave runup driver with Stockdon et al. (2006) (Nick Cohn)
- Non-FFT 1D based topographic shear coupler added for computational speed up (Nick Cohn)

1.8.2 v1.2.2 (18 April 2020)

Breaking changes

- Removed support for statistical variable names with dot-notation (e.g. *.avg* and *.sum*) (Bas Hoonhout)

Improvements

- Logger shows minute by minute updates (Tom Pak)

New functions/methods

- Avalanching process included in bed.py (Tom Pak)
- Implementation of non-erodible layers (Tom Pak)

Bug fixes

- boundary condition definition updated (Tom Pak)
- compatiblity with new NETCDF4 version restored (Sierd de Vries)
- compatiblity with 1D domains (Sierd de Vries)

Tests

None.

1.8.3 v1.1.5 (unreleased)

Breaking changes

None.

Improvements

- Also enable inundation if process_tide is True, but tide_file not specified. In this case the water level is constant zero.
- Changed class attributes into instance attributes to support parallel independent model instances.

New functions/methods

None.

Bug fixes

- Fixed double definition of statistics variables in netCDF file in case both *output_types* is specified and individual statistics variables are specified in *output_vars*.

Tests

None.

1.8.4 v1.1.4 (15 February 2018)

Improvements

- Route all log messages and exceptions through the logging module. Consequently, all information, warnings, and exceptions, including tracebacks can be logged to file.
- Added model version number and Git hash to log files and model output.

1.8.5 v1.1.3 (9 February 2018)

Bug fixes

- Apply precipitation/eaporation only in top bed layer to prevent mismatching matrix shapes in the multiplication. In the future, precipitation might be distributed over multiple layers depending on the porosity.

1.8.6 v1.1.2 (21 December 2017)

Breaking changes

- Changed name of statistics variables that describe the average, minimum, maximum, cumulative values, or variance of a model state variable. The variables names that used to end with *.avg*, *.sum*, etc. now end with *_avg*, *_sum*, etc. The new naming convention was already adopted in the netCDF output in order to be compatible with the CF-1.6 convention, but is now also adopted in, for example, the Basic Model Interface (BMI). Old notation is deprecated but still supported.

Improvements

- Prepared for continuous integration through CircleCI.
- Prepared for code coverage checking through codecov.

Bug fixes

- Use percentages (0-100) rather than fractions (0-1) in the formulation of Belly and Johnson that describes the effect of soil moisture on the shear velocity threshold. Thanks to Dano Roelvink and Susana Costas (b3d992b).

Tests

- Reduced required accuracy for mass conservation tests from 0.00000000000001% to 1%.

1.8.7 v1.1.1 (15 November 2017)

Improvements

- Made code compatible with Python 3.x.
- Prepared and uploaded package to PyPI.
- Switch back to original working directory after finishing simulation.
- Removed double definition of model state. Now only defined in *constants.MODEL_STATE*.
- Also write initial model state to output.
- Made netCDF output compatible with CF-1.6 convention.

New functions/methods

- Added support to run a default model for testing purposes by setting the configuration file as "DEFAULT".
- Added generic framework for reading and applying spatial masks. Implemented support for wave, tide and threshold masks specifically.
- Added option to include a reference date in netCDF output.
- Added experimental option for constant boundary conditions.
- Added support for reading and writing hotstart files to load a (partial) model state upon initialisation.
- Added preliminary wind shear perturbation module. Untested.
- Added support to switch on or off specific processes.
- Added support for immutable model state variables. This functionality can be combined with BMI or hotstart files to prevent external process results to be overwritten by the model.
- Added option to specify wind direction convention (nautical or cartesian).

Bug fixes

- Fixed conversion from volume to mass using porosity and density (fe9aa52).
- Update water level with bed updates to prevent loss of water due to bed level change (fe9aa52).
- Fixed mass bug in base layer that drained sediment from bottom layers, resulting in empty layers (f612760).
- Made removal of negative concentrations mass conserving by scraping the concentrations from all other grid cells (03de813).

Tests

- Added tests to check mass conservation in bed mixing routines.
- Added integration tests.

1.8.8 v1.1.0 (27 July 2016)

Initial release

CHAPTER

TWO

ACKNOWLEDGEMENTS

AeoLiS is initially developed at Delft University of Technology with support from the ERC-Advanced Grant 291206 Nearshore Monitoring and Modeling (NEMO) and Deltares. AeoLiS is currently maintained by Bart van Westen at Deltares, Nick Cohn at U.S. Army Engineer Research and Development Center (ERDC) and Sierd de Vries at Delft University of Technology.

CHAPTER THREE

INDICES AND TABLES

- genindex
- search

BIBLIOGRAPHY

[vSimrunekvSejnavG98] J. Šimůnek, M. Šejna, and M. Th. van Genuchten. *The HYDRUS-1D software package for simulating the one-dimensional movement of water, heat, and multiple solutes in variably- saturated media*. International Ground Water Modeling Center, Colorado School of Mines, Golden, Colorado, version 1.0. igwmc - tps - 70 edition, 1998. 186pp.

[Bag37a] RA Bagnold. The size-grading of sand by wind. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, pages 250–264, 1937.

[Bag37b] RA Bagnold. The transport of sand by wind. *Geographical journal*, pages 409–438, 1937.

[BMH98] Andrew J. Baird, Travis Mason, and Diane P. Horn. Validation of a Boussinesq model of beach ground water behaviour. *Marine Geology*, 148(1-2):55–69, 1998. doi:10.1016/S0025-3227(98)00026-7.

[Bel64] P Y Belly. Sand movement by wind. Technical Report 1, U.S. Army Corps of Engineers CERC, 1964. 38 pp.

[BSDR19] Laura B. Brakenhoff, Yvonne Smit, Jasper J.A. Donker, and Gerben Ruessink. Tide-induced variability in beach surface moisture: Observations and modelling. *Earth Surface Processes and Landforms*, 44(1):317–330, 2019. doi:10.1002/esp.4493.

[dVvTdVvR+14] S de Vries, J S M van Thiel de Vries, L C van Rijn, S M Arens, and R Ranasinghe. Aeolian sediment transport in supply limited situations. *Aeolian Research*, 12:75–85, 2014. doi:10.1016/j.aeolia.2013.11.005.

[DF10] I Delgado-Fernandez. A review of the application of the fetch effect to modelling sand supply to coastal foredunes. *Aeolian Research*, 2:61–70, 2010. doi:10.1016/j.aeolia.2010.04.001.

[KNH94] H.Y. Kang, P. Nielsen, and D.J Hanslow. Watertable overheight due to wave runup on a sandy beach. In *Coastal Engineering 1994*, 2115–2124. 1994.

[Kin51] C. A. M. King. Depth of disturbance of sand on sea beaches by waves. *Journal of Sedimentary Petrology*, 21(3):131–140, 1951. URL: http://archives.datapages.com/data/sepm/journals/v01-32/data/021/021003/pdfs/0131.pdf.

[MAROHare07] G Masselink, N Auger, P Russell, and T O'Hare. Short-term morphological change and sediment dynamics in the intertidal zone of a macrotidal beach. *Sedimentology*, 54:39–53, 2007. doi:10.1111/j.1365-3091.2006.00825.x.

[Mua74] Y Mualem. A Conceptual Model of Hysteresis. *Water Resources Research*, 10(3):514–520, 1974.

[NE81] W G Nickling and M Ecclestone. The effects of soluble salts on the threshold shear velocity of fine sand. *Sedimentology*, 28:505–510, 1981.

[NDWE88] P Nielsen, GA Davis, JM Winterbourne, and G Elias. Wave setup and the watertable in sandy beaches, Technical Report Technical Memo- randum 88/1, New South Wales Public Works Department Coastal Branch. Technical Report, -, 1988.

[Nie90] Peter Nielsen. Tidal dynamics of the water table in beaches. *Water Resources Research*, 26(9):2127–2134, 1990. doi:10.1029/WR026i009p02127.

[Nie09] Peter Nielsen. *Coastal and estuarine processes*. World Scientific Publishing Company, 2009.

[PHN13] S. D. Peckham, E. W. H. Hutton, and B. Norris. A component-based approach to integrated modeling in the geosciences: the design of CSDMS. *Computers and Geosciences*, 53:3–12, 2013. doi:10.1016/j.cageo.2012.04.002.

[PT90] K. Pye and H. Tsoar. *Aeolian Sand and Sand Dunes*. Unwin Hyman, London, 1990.

[RGE99] B. Raubenheimer, R. T. Guza, and Steve Elgar. Tidal water table fluctuations in a sandy ocean beach. *Water Resources Research*, 35(8):2313–2320, 1999. doi:10.1029/1999WR900105.

[RGL93] MR Raupach, DA Gillette, and JF Leys. The effect of roughness elements on wind erosion threshold. *Journal of Geophysical Research: Atmospheres*, 98(D2):3023–3029, 1993. doi:10.1029/92JD01922.

[Sch14] Phillip P Schmutz. *Investigation of Factors Controlling the Dynamics of Beach Surface Moisture*. PhD thesis, Louisiana State University, 2014.

[Shu93] W J Shuttleworth. Evaporation. In D R Maidment, editor, *Handbook of Hydrology*, pages 4.1–4.53. McGraw-Hill, New York, 1993.

[SHHS06] Hilary F. Stockdon, Rob A. Holman, Peter A. Howd, and Asbury H. Sallenger. Empirical parameterization of setup, swash, and runup. *Coastal Engineering*, 53(7):573–588, 2006. URL: https://www.sciencedirect.com/science/article/pii/S0378383906000044, doi:https://doi.org/10.1016/j.coastaleng.2005.12.005.

[vG80] M. Th van Genuchten. Closed-Form Equation for Predicting the Hydraulic Conductivity of Unsaturated Soils. *Soil Science Society of America Journal*, 44(5):892–898, 1980. doi:10.2136/sssaj1980.03615995004400050002x.

[Wil71] A. T. Williams. An analysis of some factors involved in the depth of disturbance of beach sand by waves. *Marine Geology*, 11(3):145–158, 1971. doi:10.1016/0025-3227(71)90003-X.

[Delft3DFManual14] Delft3D-FLOW Manual. *Delft3D - 3D/2D modelling suite for integral water solutions - Hydro-Morphodynamics*. Deltares, Delft, May 2014. Version 3.15.34158.

[PHN13] S. D. Peckham, E. W. H. Hutton, and B. Norris. A component-based approach to integrated modeling in the geosciences: the design of CSDMS. *Computers and Geosciences*, 53:3–12, 2013. doi:10.1016/j.cageo.2012.04.002.

PYTHON MODULE INDEX

INDEX

Symbols

A

B

C

D

E

F

G

I

W