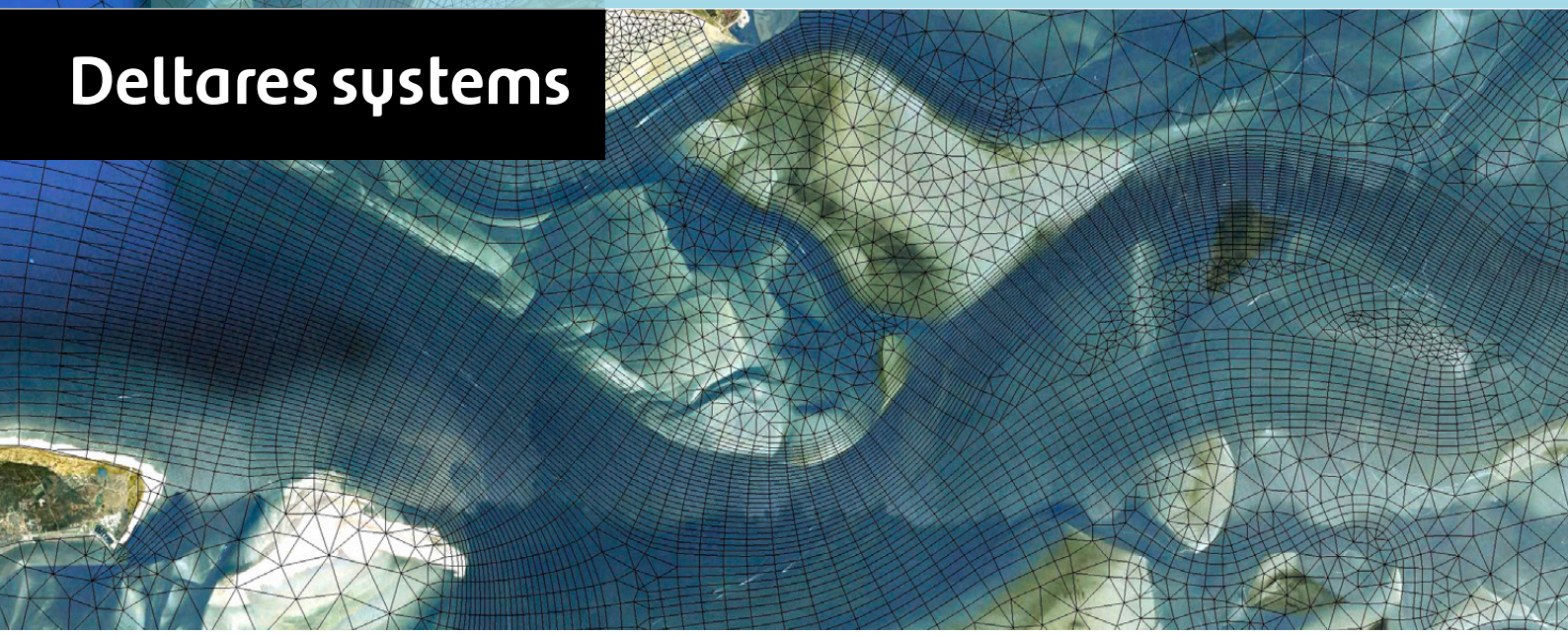
1D/2D/3D Modelling suite for integral water solutions

Delft3D Flexible Mesh Suite

Deltares systems



D-Flow Flexible Mesh

Deltares

Enabling Delta Life

Technical Reference Manual

D-Flow Flexible Mesh

Technical Reference Manual

Version: 1.1.148
Revision: 41880

14 September 2015

D-Flow Flexible Mesh, Technical Reference Manual

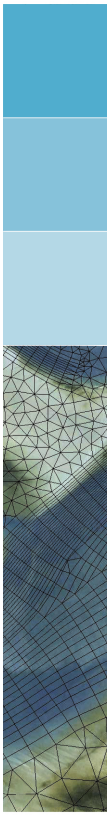
Published and printed by:
Deltares
Boussinesqweg 1
2629 HV Delft
P.O. 177
2600 MH Delft
The Netherlands

telephone: +31 88 335 82 73
fax: +31 88 335 85 82
e-mail: info@deltares.nl
www: https://www.deltares.nl

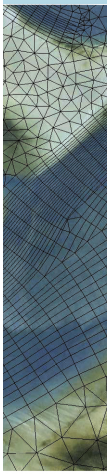
For sales contact:
telephone: +31 88 335 81 88
fax: +31 88 335 81 11
e-mail: sales@deltaressystems.nl
www: http://www.deltaressystems.nl

For support contact:
telephone: +31 88 335 81 00
fax: +31 88 335 81 11
e-mail: support@deltaressystems.nl
www: http://www.deltaressystems.nl

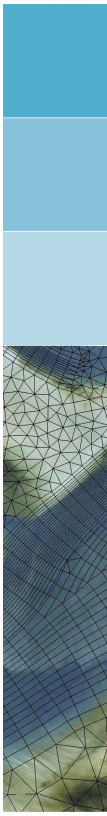
Contents



List of Figures



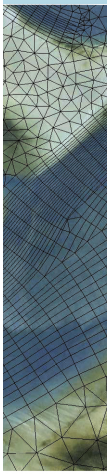
List of Tables



1 Problem specification

The specification of a problem to be run should resemble the procedure for Delft3D-FLOW, i.e., through a Master Definition file. The Master Definition Unstructured (MDU) file standards are evidently not equal to those for Delft3D (yet?).

1.1 The master definition file

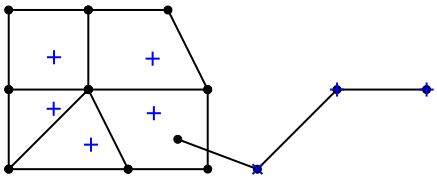


2 Data structures

The data structures used for flow simulations on unstructured meshes are fundamentally different from those on curvilinear meshes, which fit in standard rank-2 arrays. Section 2.1 contains the conceptual hierarchy of mesh and flow data. Section 2.2 contains implementation details of the variables and IO-routines available.

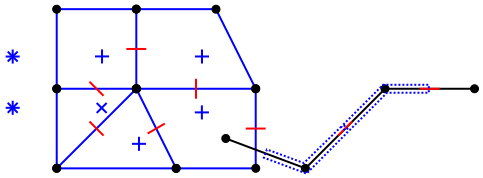
2.1 Hierarchy of unstructured nets

1. Net (domain discretization)



net node	(1..NUMK)
net link (1D)	(1..NUML1D)
net link (2D)	(NUML1D+1..NUML)

2. Flow data (2D)



netcell/flow node (2D)	(1..NDX2D=NUMP)
netcell/flow node (1D)	(NDX2D+1..NDXI)
boundary flow node	(NDXI+1..NDX)
netcell circumcenter	
netcell link	(1..LNX1D) (LNX1D..LNXI) (LNXI+1..LNX)

2.2 Implementation details of unstructured nets

2.3 ...

2.3.1 Improved cache use by node renumbering

The order of nodes in unstructured nets can be arbitrary, as opposed to structured nets, where neighbouring grid points generally lie at offsets ± 1 and $\pm N_x$ in computer memory.

The order of net nodes in memory should not affect the numerical outcomes in any way, so it is safe to apply any permutation to the net- and/or flow nodes. A permutation that puts nodes that are close to each other in the net also close to each other in memory likely improves cache effiency.

The basic problem is: given a set of nodes and their adjacency matrix, find a permutation for the nodes such that, when applied, the new adjacency matrix has a smaller bandwidth. The Reverse Cuthill–McKee (RCM) algorithm is a possible way to achieve this.

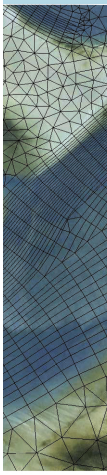
Net nodes can be renumbered, with the net links used as adjacency information. This is only done upon the user's request (*Operations* > *Renumber net nodes*), since net node ordering does not affect the flow simulation times very much. For flow nodes it is done automatically, as part of `flow_geominit()`. It can be switched off in *Various* > *Change geometry settings*. Technical detail: for true efficiency the flow links should be ordered approximately in the same pace as the flow nodes. Specifically: `lne` is reordered, based on its first node `lne(1,:)`.

Other code parts require (assume) that net links are indexed identical to flow links, so `kn` is reordered in the same way as `lne` was.

3 Unstructured grid generation

The grid generation parts in D-Flow FM are standard grid generation techniques for either curvilinear grids, triangular grids or 2D networks. D-Flow FM does not generate a hybrid unstructured net of arbitrary polygons at once, but facilitates easy combination of beforementioned grids and nets in subdomains. It *does* offer grid optimization over the entire hybrid net, such as orthogonalization, automated removal of small cells and more.

Most of this functionality will be moved to RGFGRID.



3.1 Curvilinear grids

Curvilinear grid generation is done by (old) code from RGFGRID, within polygons of splines.

3.2 Triangular grids

Unstructured triangular grid generation is done with the Triangle code by J.R. Shewchuk from Berkely. This is an implementation of Delaunay triangulation. In RGFGRID, this will be replaced by SEPRAN routines.

3.3 2D networks

Two-dimensional (SOBEK-like) networks are interactively clicked by the user.

3.4 Grid optimizations

There are two grid optimization procedures: orthogonalization and smoothing. They will be explained in the following sections.

3.5 Grid orthogonalization

D-Flow FM adopts a staggered scheme for the discretization of the two-dimensional shallow water equations. Due to our implementation of the staggered scheme, the computational grid needs to be *orthogonal*.

Definition 3.5.1. *We say that a grid is orthogonal if its edges are perpendicular to the edges of the dual grid.*

To this end, we will firstly construct a local grid mapping $\boldsymbol{x}(\xi, \eta)$ attached to some node k, see Figure 3.1. Since the ξ and η grid lines are aligned with the primary and dual edges, the grid will be orthogonal if the grid mapping satisfies the relation

$$\boldsymbol{x}_\eta = -a\boldsymbol{x}_\xi^\perp, \tag{3.1}$$

where a is the aspect ratio defined by:

$$a = \frac{\|\boldsymbol{x}_\eta\|}{\|\boldsymbol{x}_\xi\|}. \tag{3.2}$$

A grid mapping that satisfies relation (3.1), also satisfies the scaled Laplace equation:

$$\frac{\partial}{\partial\xi}\left(a\frac{\partial\boldsymbol{x}}{\partial\xi}\right) + \frac{\partial}{\partial\eta}\left(\frac{1}{a}\frac{\partial\boldsymbol{x}}{\partial\eta}\right) = 0, \tag{3.3}$$

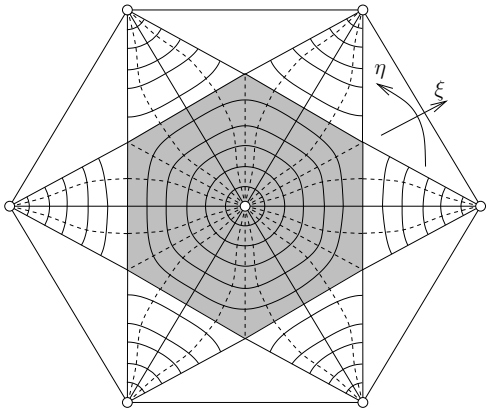


Figure 3.1: *Local grid mapping $\boldsymbol{x}(\xi,\eta)$ around a node for orthogonalization; ξ-lines are dashed; the dual cell is shaded*

which can be written in the following weak form

$$\int_{\partial\Omega_\xi} a\frac{\partial \boldsymbol{x}}{\partial \xi}n_\xi + \frac{1}{a}\frac{\partial \boldsymbol{x}}{\partial \eta}n_\eta \,\mathrm{d}S_\xi = 0, \tag{3.4}$$

where Ω_ξ is a control volume in (ξ,η) space and $\mathbf{n}_\xi = (n_\xi, n_\eta)^T$ is a normal vector. For nodes on the grid boundary, the following condition expresses orthogonal grid lines at the grid boundary:

$$\frac{\partial \boldsymbol{x}}{\partial \eta}\cdot\frac{\partial \boldsymbol{x}}{\partial \xi} = 0. \tag{3.5}$$

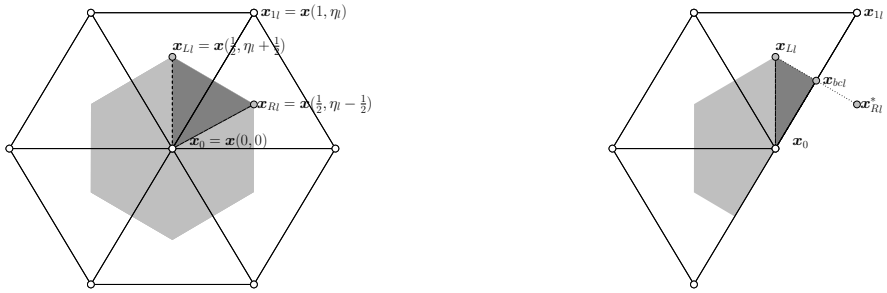
Discretization of Equation 3.3 yields

$$\begin{aligned}\sum_{l\in\mathcal{L}_i}\frac{\|\boldsymbol{x}_{Rl}-\boldsymbol{x}_{Ll}\|}{\|\boldsymbol{x}_{1l}-\boldsymbol{x}_0\|}(\boldsymbol{x}_{1l}-\boldsymbol{x}_0) \quad + \\ \sum_{l\in\mathcal{L}_e}\left\{\frac{1}{2}\frac{\|\boldsymbol{x}^*_{Rl}-\boldsymbol{x}_{Ll}\|}{\|\boldsymbol{x}_{1l}-\boldsymbol{x}_0\|}(\boldsymbol{x}_{1l}-\boldsymbol{x}_0) \quad + \quad \frac{1}{2}\frac{\|\boldsymbol{x}_{1l}-\boldsymbol{x}_0\|}{\|\boldsymbol{x}^*_{Rl}-\boldsymbol{x}_{Ll}\|}(\boldsymbol{x}^*_{Rl}-\boldsymbol{x}_{Ll})\right\} \quad = \quad 0,\end{aligned} \tag{3.6}$$

with the following nomenclature:

Definition 3.5.2. *$\mathcal{L}_i$ is the set of* internal *primary edges connected to node k and $\boldsymbol{x}_0$ and $\boldsymbol{x}_{1l}$ are the coordinates of that node and of the neighboring node connected through edge l, respectively. Furthermore, $\boldsymbol{x}_{Ll}$ and $\boldsymbol{x}_{Rl}$ are the left and right neighboring cell-center coordinates, see Figure 3.2a.*

The second summation in Equation 3.6 accounts for boundary edges.



(a) *internal edge*

(b) *boundary edge; x_{bcl} are the coordinates of a node projected onto the grid boundary, x^*_{Rl} are ghost node coordinates*

Figure 3.2: *part of the control volume that surrounds edge l (dark shading) and the nodes involved*

Definition 3.5.3. *$\mathcal{L}_e$ is the set of boundary edges (nonempty if node k is on the grid boundary only), x^*_{Rl} are the coordinates of a ghost node and x_{bcl} are boundary node coordinates, see Figure 3.2b.*

The ghost node is constructed by extrapolation from the center and boundary nodes, i.e.

$$x^*_{Rl} = 2x_{bcl} - x_{Ll}, \tag{3.7}$$

and using the boundary condition of Equation 3.8 to project the left cell-center orthogonally onto the grid boundary:

$$x_{bcl} = x_0 + \frac{(x_{1l} - x_0) \cdot (x_{Ll} - x_0)}{\|x_{1l} - x_0\|^2}(x_{1l} - x_0). \tag{3.8}$$

Remark 3.5.4. We will always assume that the grid is on the left-hand side of a boundary edge.

Finally, Equation 3.6 can be put in the following form

$$\sum_{l \in \mathcal{L}_i} a_l \left(x_{1l} - x_0\right) \quad + \quad \sum_{l \in \mathcal{L}_e} \left\{ \frac{1}{2} a_l \left(x_{1l} - x_0\right) \quad + \quad \frac{1}{2} \|x_{1l} - x_0\| \mathbf{n}_l, \right\} \quad = \quad 0, \tag{3.9}$$

where $\mathbf{n}_l = \frac{x^*_{Rl} - x_{Ll}}{\|x^*_{Rl} - x_{Ll}\|}$ is the outward normal at edge l and a_l is the aspect ratio of edge l, i.e.

$$a_l = \begin{cases} \dfrac{\|x_{Rl} - x_{Ll}\|}{\|x_{1l} - x_0\|}, & l \in \mathcal{L}_i, \\ \dfrac{\|x^*_{Rl} - x_{Ll}\|}{\|x_{1l} - x_0\|}, & l \in \mathcal{L}_e. \end{cases} \tag{3.10}$$

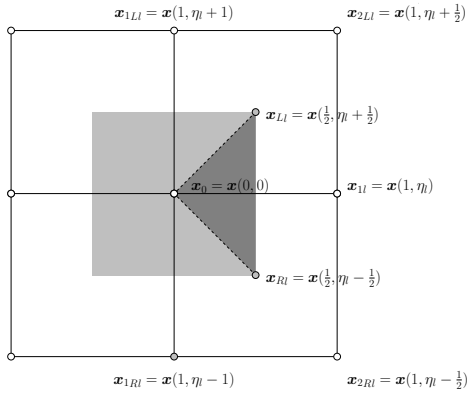


Figure 3.3: *part of the control volume that surrounds edge l (dark shading) and the nodes involved; quadrilateral grid cells*

3.5.1 Curvilinear-like discretization

The previous formulation may lead to distorted quadrilateral grids. This is remedied by mimicking a curvilinear formulation in the quadrilateral parts of the grid. Then, in Equation 3.9 the aspect ratio of Equation 3.10 is replaced by

$$a_l = \begin{cases} \dfrac{4\,\|\boldsymbol{x}_{Rl} - \boldsymbol{x}_{Ll}\|}{2\|\boldsymbol{x}_{1l} - \boldsymbol{x}_0\| + \|\boldsymbol{x}_{2Rl} - \boldsymbol{x}_{1Rl}\| + \|\boldsymbol{x}_{2Ll} - \boldsymbol{x}_{1Ll}\|}, & l \in \mathcal{L}_i, \\ \dfrac{2\,\|\boldsymbol{x}^*_{Rl} - \boldsymbol{x}_{Ll}\|}{\|\boldsymbol{x}_{1l} - \boldsymbol{x}_0\| + \|\boldsymbol{x}_{2Ll} - \boldsymbol{x}_{1Ll}\|}, & l \in \mathcal{L}_e, \end{cases} \tag{3.11}$$

where the nodes involved are depicted in Fig. 3.3.

3.6 Inverse map elliptic grid smoothing

Enhancing the smoothness of the grid is performed by means of an elliptic smoother. This work is based on Huang (2001, 2005). In order to prevent grid folds in non-convex domains, the smoother is formulated in terms of a so-called inverse map, i.e. $\boldsymbol{\xi}(x, y)$, and leads to

$$\nabla \cdot G \nabla \xi = 0, \tag{3.12}$$

where G is the monitor function for grid adaptivity which will be explained later.

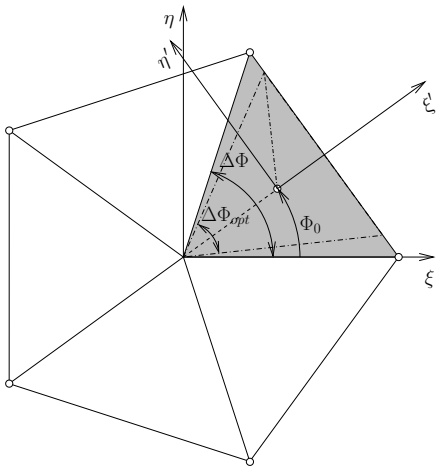
Remark 3.6.1. Although the method is based on an inverse mapping $\boldsymbol{\xi}(x, y)$, it is more convenient to work with the direct mapping $\boldsymbol{x}(\xi, \eta)$.

By interchanging the role of dependent and independent variables, Equation 3.12 can be transformed into an expression for the direct grid mapping $\boldsymbol{x}(\xi, \eta)$:

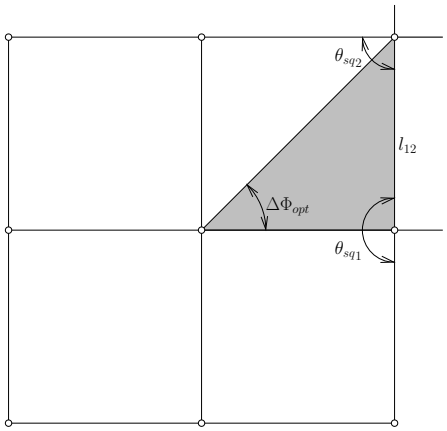
$$(\boldsymbol{a}^1, G^{-1}\boldsymbol{a}^1)\frac{\partial^2 \boldsymbol{x}}{\partial \xi^2} + (\boldsymbol{a}^1, G^{-1}\boldsymbol{a}^2)\frac{\partial^2 \boldsymbol{x}}{\partial \xi \partial \eta} + (\boldsymbol{a}^2, G^{-1}\boldsymbol{a}^1)\frac{\partial^2 \boldsymbol{x}}{\partial \eta \partial \xi} + (\boldsymbol{a}^2, G^{-1}\boldsymbol{a}^2)\frac{\partial^2 \boldsymbol{x}}{\partial \eta^2} = 0, \tag{3.13}$$

where by $(\ldots, \ldots)$ an inner product is meant and $\boldsymbol{a}^1 = \nabla \xi$ and $\boldsymbol{a}^2 = \nabla \eta$ are the contravariant base vectors.

Obviously we need to start by defining the node coordinates in (ξ, η)-space based on their connectivity with neighboring grid nodes.



(a) *non-square triangular cell; the dashed cell is an optimal equiangular polygon, while the shaded cell is the resulting cell after scaling in η' direction; Φ_0 is the angle of the ξ'-axis in the (ξ, η) frame*



(b) *square triangular cell; additional node angles $\theta_s q_1$ and θ_{sq_2} and edge l_{12} are used to determine optimal angle $\Delta\Phi_{opt}$*

Figure 3.4: *Optimal angle $\Delta\Phi_{opt}$, true angle $\Delta\Phi$, element-center angle Φ_0 and (ξ', η') coordinate frame*

3.6.1 Assigning the node coordinates in computational space

By assigning the node coordinates in computational space (ξ, η), we postulate the optimal smooth grid. Compare with a curvi-linear grid in this respect. To see how we have to choose the (ξ, η) coordinates, first consider a linearization of the grid mapping around a node:

$$\boldsymbol{x} = \boldsymbol{x}_0 + J(\boldsymbol{\xi} - \boldsymbol{\xi}_0) + \mathcal{O}\left(\|\boldsymbol{\xi}\|^2\right), \tag{3.14}$$

where $\boldsymbol{x}_0$ and $\boldsymbol{\xi}_0$ are the node coordinates in physical and computational space respectively and J is the Jacobian matrix of the transformation. Following Huang (2005), the Jacobian matrix J can be decomposed into (singular value decomposition):

$$J = U\Sigma V^T, \tag{3.15}$$

where V is a rotation in (ξ, η)-space, Σ a compression/expansion and U a rotation in (x, y)-space. Since Equation 3.13 is invariant to rotation of the (ξ, η)-axes, rotation V is irrelevant and we may start by assigning $\boldsymbol{\xi} = (0, 0)^T$ to the center node k and $\boldsymbol{\xi} = (1, 0)^T$ to an arbitrary neighboring node.

We now proceed by considering a cell attached to a node k in coordinate frame (ξ', η'), see Figure 3.4a, and define an *optimal* angle $\Delta\Phi_{opt}$ between the two of its edges that are connected to node k.

Definition 3.6.2. *The optimal angle $\Delta\Phi_{opt}$ is the angle between two adjacent edges of a cell, both connected to node k, that would lead to the desired optimal smooth grid cell.*

Remark 3.6.3. In general, the optimal smooth cell is an equiangular polygon, with the exception of square triangles. The optimal angle of a square triangle is either $\frac{1}{4}\pi$ or $\frac{1}{2}\pi$, depending on the grid connectivity.

For a non-square triangle this angle would be $\frac{1}{3}\pi$. However, by considering a node with five non-square triangles attached, one can easily understand that this angle is unsuitable in general, as five of such angles do not sum up to 2π. Therefore, we define a *true* angle as follows.

Definition 3.6.4. *The true angle $\Delta\Phi_0$ is the angle between two adjacent edges of a cell, both connected to node k, such that sum of all cell true-angles equals its prescribed value of either 2π (internal nodes), π (boundary nodes) or $\frac{1}{2}\pi$ (corner nodes).*

The true cell is obtained from the optimal cell by scaling the cell as will be explained later.

Returning to the optimal angle, we first discriminate between *square* cells and non-square cells to account for (partly) quadrilateral grids.

Definition 3.6.5. *The* stencil *is the set of cells that are connected to node k. A node angle is the angle between two stencil-boundary edges connected to that node. A square node, not being node k itself, is a node that is connected to three or less quadrilateral non-stencil cells and no other non-stencil nodes. Square nodes have node angles as indicated in Figure 3.4b, which can be $\frac{1}{2}\pi$, π or $\frac{3}{2}\pi$. A square cell is a quadrilateral cell or a triangular cell which contains at least one square node.*

The node angles are computed by

$$\theta_{sq_i} = \begin{cases} (2 - \frac{1}{2}N_{nsq})\,\pi, & \text{node } i \text{ is a square internal node,} \\ (1 - \frac{1}{2}N_{nsq})\,\pi, & \text{node } i \text{ is a square boundary node,} \\ \frac{1}{2}\pi, & \text{node } i \text{ is a square corner node,} \end{cases} \tag{3.16}$$

where N_{nsq} is the number of non-stencil quadrilaterals connected to node i. The optimal angle $\Delta\Phi_{opt}$ is finally determined by

$$\Delta\Phi_{opt} = \begin{cases} (1 - \frac{2}{N})\,\pi, & N \geq 4 \vee \text{non-square cell,} \\ \frac{1}{2}\pi, & N = 3 \wedge \text{square cell with two square nodes '1' and '2'} \wedge \\ & \quad \frac{\theta_{sq_1}}{2} + \frac{\theta_{sq_2}}{2} = \frac{\pi}{2} \wedge l_{12} \text{ is not a boundary edge,} \\ \frac{1}{4}\pi, & \text{other,} \end{cases} \tag{3.17}$$

where N is the number of nodes that comprise the cell. In the example of Figure 3.4b, nodes 1 and 2 are square nodes with angles of π and $\frac{1}{2}\pi$ respectively and the shaded cell is a square triangle with optimal angle $\frac{1}{4}\pi$.

3.6.1.1 Determining the true cell angles

Having defined the optimal angles for all cells, we can derive the true angles by demanding that the cells fit in the stencil. To this end, we consider the number and type of cells connected to node k.

Definition 3.6.6. *The sum of all cell-optimal and true angles are called $\Delta\Phi_{tot}$ and $\Delta\Phi_{0tot}$ respectively. Furthermore, the sum of all optimal and true angles of quadrilateral cells are called $\Delta\Phi_{quad}$ and $\Delta\Phi_{0quad}$ respectively. The number of quadrilateral cells is N_{quad}. The same definitions hold for the square triangular cells: $\Delta\Phi_{tri_{sq}}$, $\Delta\Phi_{0tri_{sq}}$ and $N_{tri_{sq}}$ respectively and for the remaining cells: $\Delta\Phi_{tri}$, $\Delta\Phi_{0tri}$ and N_{tri} respectively.*

Remark 3.6.7. The remaining cells are not necessarily non-square triangles only, but can also be pentagons and/or hexagons, et cetera.

Of course holds

$$\Delta\Phi_{tot} = \Delta\Phi_{quad} + \Delta\Phi_{tri_{sq}} + \Phi_{tri}, \tag{3.18}$$

$$N_{tot} = N_{quad} + N_{tri_{sq}} + N_{tri}. \tag{3.19}$$

In a similar fashion, the sum of all true angles should sum up to $f2\pi$, where

$$f = \begin{cases} 1, & \text{internal node,} \\ \frac{1}{2}, & \text{boundary node,} \\ \frac{1}{4}, & \text{corner node.} \end{cases} \tag{3.20}$$

In other words, we seek true angles $\Delta\Phi_{0quad}$, . . . , such that:

$$\Delta\Phi_{0quad} + \Delta\Phi_{0tri_{sq}} + \Phi_{0tri} = f\,2\pi. \tag{3.21}$$

This is achieved by setting

$$\Delta\Phi_{0quad} = \mu \quad \Delta\Phi_{quad}, \tag{3.22}$$
$$\Delta\Phi_{0tri_{sq}} = \mu\ \mu_{tri_{sq}}\Delta\Phi_{tri_{sq}}, \tag{3.23}$$
$$\Delta\Phi_{0tri} = \mu\ \mu_{tri}\ \ \Delta\Phi_{tri}. \tag{3.24}$$

We give highest precedence to the optimal angles of quadrilateral cells, followed by square triangular cells and lowest to the remaining cells, by taking

$$\mu_{tri} = \max(\frac{f\,2\pi - (\Delta\Phi_{quad} + \Delta\Phi_{tri_{sq}})}{\Delta\Phi_{tri}},\ \mu_{tri_{min}}), \tag{3.25}$$

$$\mu_{tri_{sq}} = \begin{cases} 1 & N_{tri} > 0 \\ \max(\frac{f\,2\pi - \Delta\Phi_{quad}}{\Delta\Phi_{tri_{sq}}},\ \mu_{tri_{sq_{min}}}), & N_{tri} = 0, \end{cases} \tag{3.26}$$

$$\mu = \frac{f\,2\pi}{\Delta\Phi_{quad} + \mu_{tri}\Delta\Phi_{tri} + \mu_{tri_{sq}}\Delta\Phi_{tri_{sq}}}, \tag{3.27}$$

where the lower bounds are

$$\mu_{tri_{min}} = N_{tri}\Delta\Phi_{min}/\Delta\Phi_{tri}, \tag{3.28}$$
$$\mu_{tri_{sq_{min}}} = N_{tri_{sq}}\Delta\Phi_{min}/\Delta\Phi_{tri_{sq}} \tag{3.29}$$

and $\Delta\Phi_{min} = \frac{1}{12}\pi$ is the minimum cell angle.

3.6.1.2 Assigning the node coordinates

With the the optimal angles of the cell defined, the (ξ', η') coordinates can be assigned to the cell nodes. We require that all edges connected to node k have unit length, which has its consequences for square triangles.

Remark 3.6.8. Since all edges connected to node k are required to have unit length, square triangles may be transformed into non-square triangles, but maintain their cell angle $\Delta\Phi_0$, see Figure 3.5 for an example.

Since the cell in (ξ', η') coordinates is an equiangular polygon in (ξ', η') space $\Delta\theta$, the coordinates of the i^{th} node are

$$\Delta\theta = \frac{2\pi}{N} \tag{3.30}$$
$$\xi' = R_0\,(1 - \cos(i\,\Delta\theta)), \tag{3.31}$$
$$\eta' = R_0\,\sin(i\,\Delta\theta), \tag{3.32}$$

where $i = 0$ corresponds to the center node k and counting counterclockwise, N is the number of nodes in the cell and R_0 the cell center distance from the origin:

$$R_0 = \cos(\frac{1}{2}\Delta\Phi)\,(1 - \cos(\Delta\theta)). \tag{3.33}$$

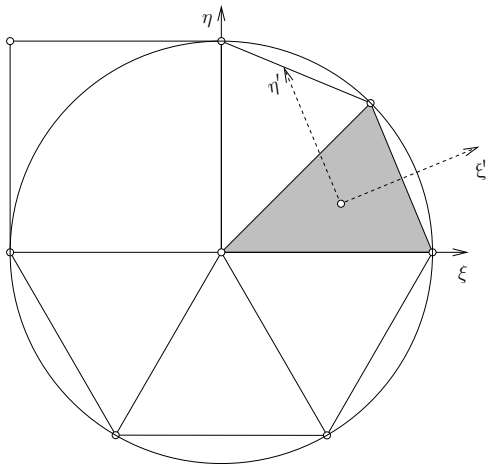
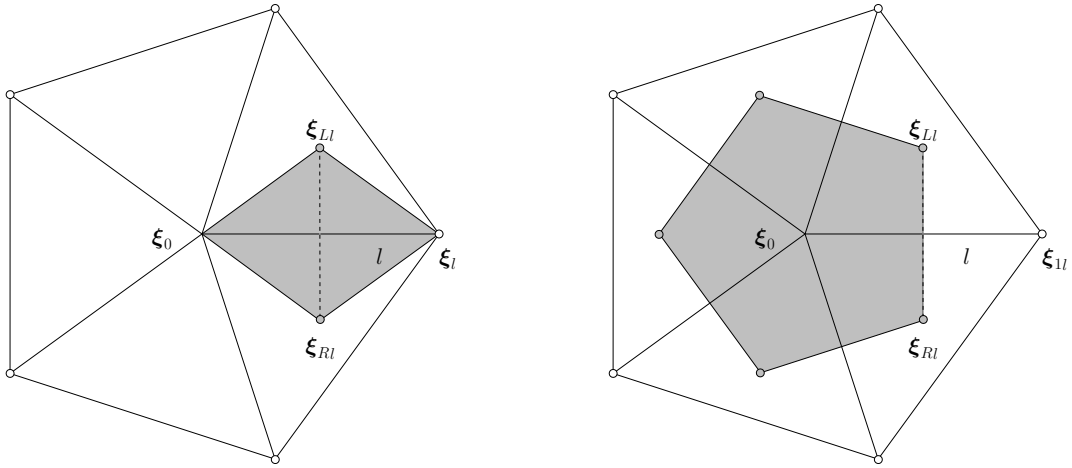


Figure 3.5: *Computational coordinates for one quadrilateral and five triangular cells, two of which were square before transformation to (ξ, η) coordinates*



(a) *node-to-edge gradient at edge l*

(b) *edge-to-node gradient at the central node, where $\boldsymbol{\xi} = \boldsymbol{\xi}_0 = \mathbf{0}$*

Figure 3.6: *control volumes for computing the discrete operators*

The coordinates (ξ, η) of the cell nodes are obtained by scaling and rotating the cell in such a way that it fits in the stencil, see Figure 3.4a. The transformation from (ξ', η') to (ξ, η) coordinates is:

$$\xi = \cos(\Phi_0)\,\xi' - A\,\sin(\Phi_0)\,\eta', \tag{3.34}$$

$$\eta = \sin(\Phi_0)\,\xi' + A\,\cos(\Phi_0)\,\eta', \tag{3.35}$$

where N is the number of nodes that comprise the cell, A is the cell aspect ratio :

$$A = \frac{(1 - \cos(\Delta\theta))\,\tan(\frac{1}{2}\Delta\Phi)}{\sin(\Delta\theta)}, \tag{3.36}$$

where $\Delta\theta = \frac{2\pi}{N}$, with N being the number of nodes that comprise the cell.

3.6.2 Computing the operators

For the solution of Equation 3.13, we approximate at k

$$\frac{\partial^2 \mathbf{x}}{\partial \xi \partial \eta} \approx \sum_{l \in \mathcal{L}} D_{\xi l} \sum_{j \in \mathcal{N}} G_{\eta l j} \mathbf{x}_j, \tag{3.37}$$

and similar for the other derivatives, where:

Definition 3.6.9. *$\mathcal{L}$ is the set of edges attached to node k and $\mathcal{N}$ is the set of nodes in the stencil of node k. Furthermore, G_ξ and G_η are the node-to-edge approximations and D_ξ and D_η the edge-to-node approximations of the ξ and η derivatives respectively.*

The discretization is as follows. For some quantity Φ, its gradient can be approximated in the usual finite-volume way

$$\nabla_\xi \Phi \approx \frac{1}{\text{vol}(\Omega_\xi)} \int_{\partial \Omega_\xi} \Phi \, \boldsymbol{n}_\xi \, \mathrm{d}S_\xi. \tag{3.38}$$

For the $node - to - edge$ gradient $(G_\xi, G_\eta)^T$ we take the control volume as indicated in Figure 3.6a and obtain for some node-based quantity Φ

$$(G_\xi, G_\eta)^T \Phi\big|_l = \begin{cases} \dfrac{(\boldsymbol{\xi}_{Rl} - \boldsymbol{\xi}_{Ll})^\perp (\Phi_{1l} - \Phi_0) \; - \; (\boldsymbol{\xi}_{1l} - \boldsymbol{\xi}_0)^\perp (\Phi_{Rl} - \Phi_{Ll})}{\|(\boldsymbol{\xi}_{1l} - \boldsymbol{\xi}_0) \times (\boldsymbol{\xi}_{Rl} - \boldsymbol{\xi}_{Ll})\|}, & l \in \mathcal{L}_i, \\ \dfrac{(\boldsymbol{\xi}^*_{Rl} - \boldsymbol{\xi}_{Ll})^\perp (\Phi_{1l} - \Phi_0) \; - \; (\boldsymbol{\xi}_{1l} - \boldsymbol{\xi}_0)^\perp (\Phi^*_{Rl} - \Phi_{Ll})}{\|(\boldsymbol{\xi}_{1l} - \boldsymbol{\xi}_0) \times (\boldsymbol{\xi}_{Rl} - \boldsymbol{\xi}_{Ll})\|}, & l \in \mathcal{L}_e, \end{cases} \tag{3.39}$$

where we use similar definitions as Definitions 3.5.2 and 3.5.3, and Remark 3.5.4 also holds. Furthermore, $\boldsymbol{\xi}^\perp = (-\eta, \xi)^T$ and $\boldsymbol{\xi}_0 = \mathbf{0}$ by construction.

The cell centers $\boldsymbol{\xi}_{Ll}$ and $\boldsymbol{\xi}_{Rl}$ can be expressed in general form as

$$\boldsymbol{\xi}_{Ll} = \sum_{j \in \mathcal{N}} A_{ic_L(l),j} \boldsymbol{\xi}_j, \tag{3.40}$$

$$\boldsymbol{\xi}_{Rl} = \sum_{j \in \mathcal{N}} A_{ic_R(l),j} \boldsymbol{\xi}_j, \tag{3.41}$$

where:

Definition 3.6.10. *ic_L and ic_R are the left and right edge-neighbor-to-cell mappings respectively.*

For non-triangular cells the cell centroid is taken. For triangular cells on the other hand, the circumcenter is used and computed as follows:

$$\boldsymbol{\xi}_{Ll} = \boldsymbol{\xi}_0 \; + \; \alpha(\boldsymbol{\xi}_{1l+1} - \boldsymbol{\xi}_0) \; + \; \beta(\boldsymbol{\xi}_{1l} - \boldsymbol{\xi}_0), \tag{3.42}$$

$$\boldsymbol{\xi}_{Rl} = \boldsymbol{\xi}_{Ll-1}, \tag{3.43}$$

where

$$\alpha = \frac{1 - \frac{1}{\gamma} c}{2(1 - c^2)}, \tag{3.44}$$

$$\beta = \frac{1 - \gamma c}{2(1 - c^2)}, \tag{3.45}$$

and

$$\gamma = \frac{\|\boldsymbol{\xi}_{1l+1} - \boldsymbol{\xi}_0\|}{\|\boldsymbol{\xi}_{1l} - \boldsymbol{\xi}_0\|}, \tag{3.46}$$

$$c = \frac{(\boldsymbol{\xi}_{1l} - \boldsymbol{\xi}_0) \cdot (\boldsymbol{\xi}_{1l+1} - \boldsymbol{\xi}_0)}{\|\boldsymbol{\xi}_{1l} - \boldsymbol{\xi}_0\| \, \|\boldsymbol{\xi}_{1l+1} - \boldsymbol{\xi}_0\|}. \tag{3.47}$$

Remark 3.6.11. The edges l around node k are arranged in counterclockwise order.

The cell center values Φ in Eqn. (3.39) are computed in the same manner, i.e.:

$$\Phi_{Ll} = \sum_{l \in \mathcal{L}} A_{ic_L(l),j} \Phi_j, \tag{3.48}$$

$$\Phi_{Rl} = \sum_{l \in \mathcal{L}} A_{ic_R(l),j} \Phi_j. \tag{3.49}$$

Combining Equations 3.39, 3.48 and 3.49 yields for internal edges

$$G_{\xi_{lj}} = \frac{-(\eta_{Rl} - \eta_{Ll})(\delta_{in(l),j} - \delta_{0,j}) \; + \; (\eta_{1l} - \eta_0)(A_{ic_R(l),j} - A_{ic_L(l),j})}{\|(\boldsymbol{\xi}_{1l} - \boldsymbol{\xi}_0) \times (\boldsymbol{\xi}_{Rl} - \boldsymbol{\xi}_{Ll})\|}, \quad l \in \mathcal{L}_i, \tag{3.50}$$

and

$$G_{\eta_{lj}} = \frac{(\xi_{Rl} - \xi_{Ll})(\delta_{in(l),j} - \delta_{0,j}) \; - \; (\xi_{1l} - \xi_0)(A_{ic_R(l),j} - A_{ic_L(l),j})}{\|(\boldsymbol{\xi}_{1l} - \boldsymbol{\xi}_0) \times (\boldsymbol{\xi}_{Rl} - \boldsymbol{\xi}_{Ll})\|}, \quad l \in \mathcal{L}_i, \tag{3.51}$$

where δ is the Kronecker delta and:

Definition 3.6.12. *in is the edge to node mapping, i.e.* $in(l)$ *is the node connected to the central node through edge* l*.*

Boundary edges are treated in a similar fashion as before, see Equation 3.7, by creating a ghost node:

$$\boldsymbol{\xi}^*_{Rl} = 2\boldsymbol{\xi}_{bcl} - \boldsymbol{\xi}_{Ll}, \tag{3.52}$$

$$\Phi^*_{Rl} = 2\Phi_{bcl} - \Phi_{Ll}, \tag{3.53}$$

and

$$\boldsymbol{\xi}_{bcl} = \boldsymbol{\xi}_0 + \alpha_\xi(\boldsymbol{\xi}_{1l} - \boldsymbol{\xi}_0), \tag{3.54}$$

$$\Phi_{bcl} = \Phi_0 + \alpha_x(\Phi_{1l} - \Phi_0), \tag{3.55}$$

with

$$\alpha_\xi = (\boldsymbol{\xi}_{Ll} - \boldsymbol{\xi}_0) \cdot \frac{\boldsymbol{\xi}_{1l} - \boldsymbol{\xi}_0}{\|\boldsymbol{\xi}_{1l} - \boldsymbol{\xi}_0\|}, \tag{3.56}$$

$$\alpha_x = \alpha_\xi. \tag{3.57}$$

Remark 3.6.13. Note that $\alpha_\xi = \frac{1}{2}$ for triangular and quadrilateral cells. The boundary conditions are non-orthogonal, in contrast to Eqn. (3.8). This maintains the linearity of operators G_ξ and G_η.

Substitution in Eqn. (3.39) yields for the boundary edges

$$G_{\xi_{lj}} = \frac{1}{\|(\boldsymbol{\xi}_{1l} - \boldsymbol{\xi}_0) \times (\boldsymbol{\xi}^*_{Rl} - \boldsymbol{\xi}_{Ll})\|} \Big[\\ \left(-(\eta^*_{Rl} - \eta_{Ll}) + 2\alpha_x(\eta_{1l} - \eta_0)\right) \delta_{in(l),j} + \\ \left((\eta^*_{Rl} - \eta_{Ll}) + 2(1-\alpha_x)(\eta_{1l} - \eta_0)\right) \delta_{0,j} - \\ 2(\eta_{1l} - \eta_0) A_{ic_L(l),j} \Big], \qquad l \in \mathcal{L}_e, \tag{3.58}$$

and

$$G_{\eta_{lj}} = \frac{1}{\|(\boldsymbol{\xi}_{1l} - \boldsymbol{\xi}_0) \times (\boldsymbol{\xi}^*_{Rl} - \boldsymbol{\xi}_{Ll})\|} \Big[\\ \left((\xi^*_{Rl} - \xi_{Ll}) - 2\alpha_x(\eta_{1l} - \eta_0)\right) \delta_{in(l),j} - \\ \left((\xi^*_{Rl} - \xi_{Ll}) + 2(1-\alpha_x)(\xi_{1l} - \xi_0)\right) \delta_{0,j} + \\ 2(\xi_{1l} - \xi_0) A_{ic_L(l),j} \Big], \qquad l \in \mathcal{L}_e. \tag{3.59}$$

For the $edge - to - node$ gradient we take the control volume as indicated in Figure 3.6b and obtain

$$(D_{\xi_l}, D_{\eta_l})^T = \frac{1}{V} \boldsymbol{d}_l, \tag{3.60}$$

where

$$\boldsymbol{d}_l = \begin{cases} (\boldsymbol{\xi}_{Rl} - \boldsymbol{\xi}_{Ll})^\perp, & l \in \mathcal{L}_i, \\ (\boldsymbol{\xi}_{bcl} - \boldsymbol{\xi}_{Ll})^\perp - (\boldsymbol{\xi}_{bcl} - \boldsymbol{\xi}_0)^\perp, & l \in \mathcal{L}_e, \end{cases} \tag{3.61}$$

and

$$V = \sum_{l \in \mathcal{L}_i} \frac{1}{2} \boldsymbol{d}_l \cdot \frac{\boldsymbol{\xi}_{Ll} + \boldsymbol{\xi}_{Rl}}{2} + \sum_{l \in \mathcal{L}_e} \frac{1}{2} \boldsymbol{d}_l \cdot \frac{\boldsymbol{\xi}_{Ll} + \boldsymbol{\xi}^*_{Rl}}{2}. \tag{3.62}$$

The computation of the Jacobian requires the *node-to-node* gradient.

Definition 3.6.14. *J_ξ and J_η are the node-to-node approximations of the ξ and η derivatives respectively.*

They can be constructed as

$$J_{\xi_j} = \sum_{l \in \mathcal{L}_i} D_{\xi l} \frac{1}{2} (A_{ic_L(l),j} + A_{ic_R(l),j}) + \sum_{l \in \mathcal{L}_e} D_{\xi l} \frac{1}{2} (\delta_{0,j} + \delta_{in(l),j}), \tag{3.63}$$

and

$$J_{\eta_j} = \sum_{l \in \mathcal{L}_i} D_{\eta l} \frac{1}{2} (A_{ic_L(l),j} + A_{ic_R(l),j}) + \sum_{l \in \mathcal{L}_e} D_{\eta l} \frac{1}{2} (\delta_{0,j} + \delta_{in(l),j}). \tag{3.64}$$

3.6.3 Computing the mesh monitor matrix

In the discretization of Equation 3.13, we approximate the contravariant base vectors by firstly computing the Jacobian by applying Equations 3.63 and 3.64, and using $\boldsymbol{a}^1 = \nabla\xi$ and $\boldsymbol{a}^2 = \nabla\eta$:

$$\boldsymbol{a}^1 = (\ \ J_{22}, -J_{12})^T / \det J, \tag{3.65}$$

$$\boldsymbol{a}^2 = (-J_{21}, \ \ J_{11})^T / \det J. \tag{3.66}$$

The mesh monitor matrix G is computed as explained in Huang (2001). It is bases on a *solution* value at grid nodes, that determines the mesh refinement direction $\boldsymbol{v}$:

$$\boldsymbol{v} = \nabla u, \tag{3.67}$$

which is approximated by firstly smoothing u, and computing

$$\boldsymbol{v} = \sum_{j\in\mathcal{N}} \boldsymbol{a}^1 J_\xi u_j + \boldsymbol{a}^2 J_\eta u_j. \tag{3.68}$$

This direction vector is directly inserted in the mesh monitor matrix, see Huang (2001) for details. The obtained mesh monitor matrix is smoothed, after which the inverse G^{-1} is calculated.

3.6.4 Composing the discretization

With the operators D_ξ, D_η, G_ξ and G_η available, and the contravariant base vectors $\boldsymbol{a}^1$ and $\boldsymbol{a}^2$ and the inverse mesh monitor matrix G^{-1} computed, the discretization of Equation 3.13 is a straightforward task. We obtain

$$\sum_{j\in\mathcal{N}} w_j \mathbf{x}_j = \mathbf{0}, \tag{3.69}$$

where

$$\begin{aligned} w_j =& (\boldsymbol{a}^1, G^{-1}\boldsymbol{a}^1) \sum_{l\in\mathcal{L}} D_{\xi l} G_{\xi lj} \ + \ (\boldsymbol{a}^1, G^{-1}\boldsymbol{a}^2) \sum_{l\in\mathcal{L}} D_{\xi l} G_{\eta lj} + \\ & (\boldsymbol{a}^2, G^{-1}\boldsymbol{a}^1) \sum_{l\in\mathcal{L}} D_{\eta l} G_{\xi lj} \ + \ (\boldsymbol{a}^2, G^{-1}\boldsymbol{a}^2) \sum_{l\in\mathcal{L}} D_{\eta l} G_{\eta lj}. \end{aligned} \tag{3.70}$$

4 Numerical schemes

4.1 Time integration

...

4.2 Matrix solver: Gauss and CG

The implicit part of the discretized PDEs is solved by a combination of Gauss elimination, based on minimum degree, and CG.[1] The procedure solves an equation $A\boldsymbol{s}_1 = \boldsymbol{b}$, where A is a sparse, diagonally dominant and symmetric matrix. The array `s1(1:nodtot)` contains the unknown values to be solved. The value of `nodtot` describes the number of nodal points. The sample program calls two routines:

1 the routine `prepare`
2 the routine `solve_matrix`

4.2.1 Preparation

`prepare` determines which rows of matrix A, i.e., which nodes, are solved by Gauss elimination and which by CG, based on the nodes' degree. It need to be applied just once, thereafter `solve_matrix` can be called as many times as needed. The inputs of `prepare` are the following arrays and variables:

`nodtot`	the total number of nodes or unknowns
`lintot`	the total number of initial upper-diagonal non-zero entries of the original equation not affected by Gaussian elimination, or the total number of lines between two nodes.
`maxdgr`	the maximum degree of a node that is eliminated by Gaussian elimination
`line(1:lintot,1:2)`	the adjacency graph of A or the list of the indices of non-zero entries.

The outputs of `prepare` are the following arrays and variables:

`nogauss`	the number of nodes that will be eliminated by Gaussian elimination
`nocg`	the number of unknowns of the remaining equation to be solved by CG.
`ijtot`	the total number of upper-diagonal non-zero entries including the fill-ins due to Gaussian elimination.
`ijl(1:lintot)`	contains the addresses of `aij(1:ijtot)` (`lintot<=ijtot`) where the non-zero entries of the original equation are to be stored.
`noel(1:nogauss)`	numbers of the nodes that will be eliminated by Gaussian elimination in the order given by `noel(1:nogauss)`. The remaining unknowns, given by `noel(nogauss+1:nogauss+nocg)`, are solved by CG.
`row(1:nodtot)`	sparse matrix administration used by `solve_matrix` (see program listing)

[1] The Gauss+CG solver was designed and implemented by Guus Stelling. This section is largely a copy of his original Word document accompanying a test program.

4.2.2 Solving the matrix

The output of `prepare` is input to `solve_matrix`. Other input to `solve_matrix` is given by:

`aii(1:nodtot)`	the main diagonal elements of A
`aij(ijl(1:lintot))`	the non-zero upper-diagonal elements of A
`bi(1:nodtot)`	the components of the right hand side vector $\boldsymbol{b}$
`s0(1:nodtot)`	initial estimate of the final solution
`ipre`	if `ipre=1` then point Jacobi preconditioning is applied otherwise LUD preconditioning will be applied

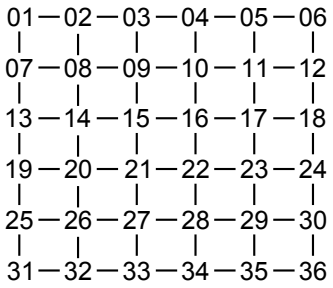
The subroutine does the following steps:

1 call `gauss_elimination`
2 call `cg(ipre)`
3 call `gauss_substitution`

After this the unknown vector `s1(1:nodtot)` has been found.

4.2.3 Example

To illustrate the `solve_matrix` routine the following example is given:



This is the adjacency graph of a 36×36 matrix A. For this graph `nodtot=36` and `lintot=60`. The graph is described by the following set of lines:

(01,02) (07,08) (13,14) (19,20) (25,26) (31,32) (02,03) (08,09) (14,15) (20,21) (26,27) (32,33) (03,04) (09,10) (15,16) (21,22) (27,28) (33,34) (04,05) (10,11) (16,17) (22,23) (28,29) (34,35) (05,06) (11,12) (17,18) (23,24) (29,30) (35,36) (01,07) (07,13) (13,19) (19,25) (25,31) (02,08) (08,14) (14,20) (20,26) (26,32) (03,09) (09,15) (15,21) (21,27) (27,33) (04,10) (10,16) (16,22) (22,28) (28,34) (05,11) (11,17) (17,23) (23,29) (29,35) (06,12) (12,18) (18,24) (24,30) (30,36),

as can be verified in the picture. The degree of each node and its connecting node numbers are given by the following table:

node	1 :	2	2	7		
node	2 :	3	1	3	8	
node	3 :	3	2	4	9	
node	4 :	3	3	5	10	
node	5 :	3	4	6	11	
node	6 :	2	5	12		
node	7 :	3	8	1	13	
node	8 :	4	7	9	2	14
node	9 :	4	8	10	3	15

```
node   10 :  4   9  11   4  16
node   11 :  4  10  12   5  17
node   12 :  3  11   6  18
node   13 :  3  14   7  19
node   14 :  4  13  15   8  20
node   15 :  4  14  16   9  21
node   16 :  4  15  17  10  22
node   17 :  4  16  18  11  23
node   18 :  3  17  12  24
node   19 :  3  20  13  25
node   20 :  4  19  21  14  26
node   21 :  4  20  22  15  27
node   22 :  4  21  23  16  28
node   23 :  4  22  24  17  29
node   24 :  3  23  18  30
node   25 :  3  26  19  31
node   26 :  4  25  27  20  32
node   27 :  4  26  28  21  33
node   28 :  4  27  29  22  34
node   29 :  4  28  30  23  35
node   30 :  3  29  24  36
node   31 :  2  32  25
node   32 :  3  31  33  26
node   33 :  3  32  34  27
node   34 :  3  33  35  28
node   35 :  3  34  36  29
node   36 :  2  35  30
```

If no Gaussian elimination is is applied, but if the equation is solved entirely by CG then this administration is used by the `cg` subroutine. However if every point up to degree 4 (i.e. `maxdgr=5`) is eliminated by Gauss then the following table might result:

```
gauss    1 :  2   2   7
gauss    6 :  2   5  12
gauss   31 :  2  32  25
gauss   36 :  2  35  30
gauss    2 :  3   3   8   7
gauss    4 :  3   3   5  10
gauss    7 :  3   8  13   3
gauss   12 :  3  11  18   5
gauss   19 :  3  20  13  25
gauss   24 :  3  23  18  30
gauss   32 :  3  33  26  25
gauss   34 :  3  33  35  28
gauss    5 :  4  11   3  10  18
gauss    8 :  4   9  14   3  13
gauss   11 :  4  10  17  18   3
gauss   15 :  4  14  16   9  21
gauss   22 :  4  21  23  16  28
gauss   25 :  4  26  20  13  33
gauss   26 :  4  27  20  33  13
gauss   29 :  4  28  30  23  35
gauss   30 :  4  35  23  18  28
```

gauss	35 :	4	33	28	23	18			
cg	3 :	6	9	10	13	18	14	17	
cg	9 :	6	10	3	14	13	16	21	
cg	10 :	5	9	16	3	18	17		
cg	13 :	6	14	3	20	9	33	27	
cg	14 :	6	13	20	9	3	16	21	
cg	16 :	7	17	10	14	9	21	23	28
cg	17 :	5	16	18	23	10	3		
cg	18 :	6	17	23	3	10	28	33	
cg	20 :	5	21	14	13	33	27		
cg	21 :	7	20	27	14	16	9	23	28
cg	23 :	6	17	18	21	16	28	33	
cg	27 :	5	28	21	33	20	13		
cg	28 :	6	27	33	21	23	16	18	
cg	33 :	6	27	28	20	13	23	18	

The corner nodes have the lowest degree so they are eliminated first as the table shows. These are followed by other nodes on the boundary before internal nodes are eliminated. After each elimination step the degree of neighboring points, due to fill-in, might be increased, so minimum degree automatically imposes some kind of colored ordering of the nodal points. Elimination of such points is known to improve the convergence properties of CG, see e.g. **?**. The nodes, which are left over for CG, clearly show the increased degree due to fill in.

In general the fastest convergence, in terms of number of iterations, is obtained by choosing `maxdgr` as large as memory allows in combination with LUD pre-conditioning. However in terms of computational time the fastest convergence is obtained by a moderate choice of `maxdgr`, such that approximately 50 % of the total number of grid points is eliminated by Gauss in combination with point Jacobi preconditioning.

5 Conceptual description

5.1 Introduction

[yet empty]

5.2 General background

[yet empty]

5.3 Governing equations

[yet empty]

5.4 Boundary conditions

[yet empty]

5.5 Turbulence

Reynold's stresses

The forces F_ξ and F_η in the horizontal momentum equations represent the unbalance of horizontal Reynolds stresses. The Reynolds stresses are modelled using the eddy viscosity concept, (for details e.g. Rodi (1984)). This concept expresses the Reynolds stress component as the product between a flow as well as grid-dependent eddy viscosity coefficient and the corresponding components of the mean rate-of-deformation tensor. The meaning and the order of the eddy viscosity coefficients differ for 2D and 3D, for different horizontal and vertical turbulence length scales and fine or coarse grids. In general the eddy viscosity is a function of space and time.

For 3D shallow water flow the stress tensor is an-isotropic. The horizontal eddy viscosity coefficient, ν_H, is much larger than the vertical eddy viscosity ν_V ($\nu_H \gg \nu_V$). The horizontal viscosity coefficient may be a superposition of three parts:

1 a part due to "sub-grid scale turbulence",
2 a part due to "3D-turbulence" see Uittenbogaard *et al.* (1992) and
3 a part due to dispersion for depth-averaged simulations.

In simulations with the depth-averaged momentum and transport equations, the redistribution of momentum and matter due to the vertical variation of the horizontal velocity is denoted as dispersion. In 2D simulations this dispersion is not simulated as the vertical profile of the horizontal velocity is not resolved. Then this dispersive effect may be modelled as the product of a viscosity coefficient and a velocity gradient. The dispersion term may be estimated by the Elder formulation.

If the vertical profile of the horizontal velocity is not close to a logarithmic profile (e.g. due to stratification or due to forcing by wind) then a 3D-model for the transport of matter is recommended.

The horizontal eddy viscosity is mostly associated with the contribution of horizontal turbulent motions and forcing that are not resolved by the horizontal grid ("sub-grid scale turbulence") or by (a priori) the Reynolds-averaged shallow-water equations. For the former we introduce the sub-grid scale (SGS) horizontal eddy viscosity ν_{SGS} and for the latter the horizontal eddy vis-

cosity ν_H^{back}. Delft3D-FLOW simulates the larger scale horizontal turbulent motions through a methodology called Horizontal Large Eddy Simulation (HLES). The associated horizontal viscosity coefficient ν_{SGS} will then be computed by a dedicated SGS-turbulence model, including the Elder contribution if requested. For details of this approach, see **??**.

The background horizontal viscosity, user-defined through the input file is represented by ν_H^{back}. Consequently, in Delft3D-FLOW the horizontal eddy viscosity coefficient is defined by

$$\nu_H = \nu_{SGS} + \nu_V + \nu_H^{back}. \tag{5.1}$$

The 3D part ν_V is referred to as the three-dimensional turbulence and in 3D simulations it is computed following a 3D-turbulence closure model.

For turbulence closure models responding to shear production only, it may be convenient to specify a background or "ambient" vertical mixing coefficient in order to account for all other forms of unresolved mixing, ν_V^{back}. Therefore, in addition to all turbulence closure models in Delft3D-FLOW a constant (space and time) background mixing coefficient may be specified by you, which is a background value for the vertical eddy viscosity in the momentum Eqs. **??** and **??**. Consequently, the vertical eddy viscosity coefficient is defined by:

$$\nu_V = \nu_{mol} + \max(\nu_{3D}, \nu_V^{back}), \tag{5.2}$$

with ν_{mol} the kinematic viscosity of water. The 3D part ν_{3D} is computed by a 3D-turbulence closure model, see **??**. Summarizing, since in Delft3D-FLOW several combinations of horizontal and vertical eddy viscosity are optional, Table **??** presents an overview of these combinations.

5.6 Secondary flow

This section presents developments regarding to the secondary flow by means of radius of flow curvature and the spiral intensity equation. Then the spiral flow intensity is used to calculate the deviation angle of shear stress, and the effect of secondary flow on depth averaged equations. The governing equations are first explained, then, the numerical techniques for reconstruction of velocity gradients are described.

5.6.1 Governing equations

5.6.1.1 Streamline curvature

The curvature of flow streamlines, $1/R_s$, can be defined by

$$\frac{1}{R_s} = \frac{\frac{dx}{dt}\frac{d^2y}{dt^2} - \frac{dy}{dt}\frac{d^2x}{dt^2}}{\left[\left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dt}\right)^2\right]^{3/2}} \tag{5.3}$$

where x and y are the coordinate components of flow element and t is time. Substituting $u = dx/dt$ and $v = dy/dt$ gives

$$\frac{1}{R_s} = \frac{u\frac{dv}{dt} - v\frac{du}{dt}}{\left(u^2 + v^2\right)^{3/2}} \tag{5.4}$$

Expanding the material derivatives du/dt and dv/dt gives,

$$\frac{1}{R_s} = \frac{u\left(\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y}\right) - v\left(\frac{du}{dt} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y}\right)}{\left(u^2+v^2\right)^{3/2}} \tag{5.5}$$

Under the assumption of a steady flow, Equation 5.5 changes to,

$$\frac{1}{R_s} = \frac{u^2\frac{\partial v}{\partial x} + uv\frac{\partial v}{\partial y} - uv\frac{\partial u}{\partial x} - v^2\frac{\partial u}{\partial y}}{\left(u^2+v^2\right)^{3/2}} \tag{5.6}$$

Equation 5.6 describes the curvature of flow streamlines by means of the velocity field. The sign of the streamline curvature indicates the direction in which the velocity vector rotates along the curve. If the velocity vector rotates counterclockwise, then $1/R > 0$ and if it rotates clockwise, then $1/R < 0$. Following this convention, the spiral flow intensity will be negative for bends with flows from right to left, and positive for bends with flows from left to the right. This convention follows the right hand side rule.

5.6.1.2 Spiral flow intensity

As the curvature is calculated, it can be contributed in the solution of spiral flow intensity. The spiral flow intensity, I, is calculated by

$$\frac{\partial hI}{\partial t} + \frac{\partial uhI}{\partial x} + \frac{\partial vhI}{\partial y} = h\frac{\partial}{\partial x}\left(D_H\frac{\partial I}{\partial x}\right) + h\frac{\partial}{\partial x}\left(D_H\frac{\partial I}{\partial y}\right) + hS \tag{5.7}$$

where h is the water depth and

$$S = -\frac{I - I_e}{T_a} \tag{5.8}$$

$$I_e = I_{be} - I_{ce} \tag{5.9}$$

$$I_{be} = \frac{h}{R_s}\left|\boldsymbol{u}\right| \tag{5.10}$$

$$I_{ce} = f\frac{h}{2} \tag{5.11}$$

$$\left|\boldsymbol{u}\right| = \sqrt{u^2+v^2} \tag{5.12}$$

$$T_a = \frac{L_a}{\left|\boldsymbol{u}\right|} \tag{5.13}$$

$$L_a = \frac{(1-2\alpha)\,h}{2\kappa^2\alpha} \tag{5.14}$$

As the spiral motion intensity is found, it can be used in calculating the bedload transport direction and the dispersion stresses (and the effect on the momentum equations).

5.6.1.3 Bedload transport direction

In the case of depth-averaged simulation (two-dimension shallow water), the spiral motion intensity is used to calculate the bedload transport direction ϕ_τ, which is given by

$$\tan\phi_\tau = \frac{v - \alpha_I \frac{u}{|\boldsymbol{u}|} I}{u - \alpha_I \frac{v}{|\boldsymbol{u}|} I} \tag{5.15}$$

in which

$$\alpha_I = \frac{2}{\kappa^2} E_s \left(1 - \frac{1}{2}\frac{\sqrt{g}}{\kappa C}\right) \tag{5.16}$$

Here g is the gravity, κ is the von Kármán constant and C is the Chezy coefficient. E_s is a coefficient specified by the user to control the effect of the spiral motion on bedlead transport. Value 0 implies that the effect of the spiral motion is not included in the bedload transport direction.

5.6.1.4 Dispersion stresses

The momentum equations for shallow water are given as

$$\frac{\partial uh}{\partial t} + \frac{\partial uuh}{\partial x} + \frac{\partial vuh}{\partial y} = -gh\frac{\partial z_s}{\partial x} - C_f u\,|\boldsymbol{u}| - \frac{\partial hT_{xx}}{\partial x} - \frac{\partial hT_{yx}}{\partial y} - \frac{\partial hS_{xx}}{\partial x} - \frac{\partial hS_{yx}}{\partial y} \tag{5.17}$$

$$\frac{\partial vh}{\partial t} + \frac{\partial vuh}{\partial x} + \frac{\partial vvh}{\partial y} = gh\frac{\partial z_s}{\partial y} - C_f v\,|\boldsymbol{u}| - \frac{\partial hT_{yx}}{\partial x} - \frac{\partial hT_{yy}}{\partial y} - \frac{\partial hS_{yx}}{\partial x} - \frac{\partial hS_{yy}}{\partial y} \tag{5.18}$$

The 3D velocity, can be decomposed into three components

$$U = u + u^* + u' \tag{5.19}$$

where u is the depth-averaged velocity component, u^* is the depth-varying and u' is the time varying component. The depth-averaged Reynolds stresses are represented as S_{xx}, S_{xy}, S_{yx} and S_{yy} following from an averaging operations in time and depth. The so-called dispersion terms are found on the right hand side

$$\begin{aligned} T_{xx} &= \langle u^* u^* \rangle \;,\; T_{xy} = \langle u^* v^* \rangle \\ T_{yx} &= \langle v^* u^* \rangle \;,\; T_{yy} = \langle v^* v^* \rangle \end{aligned} \tag{5.20}$$

The dispersion stresses need closure, similar to the Reynolds stresses. The used approach is to consider a fully developed flow in the streamwise direction (i.e. primary flow = logarithmic), and from a 1DV model it is possible to reconstruct the secondary flow profile. The time averaged velocity can be written as:

$$\overline{u} = u + u^* = u_s\,(1 + f_s)\cos\theta - u_s \frac{H}{R_s} f_n \sin\theta \tag{5.21}$$

$$\overline{v} = v + v^* = u_s \frac{H}{R_s} f_n \cos\theta + u_s\,(1 + f_s)\sin\theta \tag{5.22}$$

The depth varying component can subsequently be written as:

$$u^* = u f_s - v\frac{H}{R_s} f_n \tag{5.23}$$

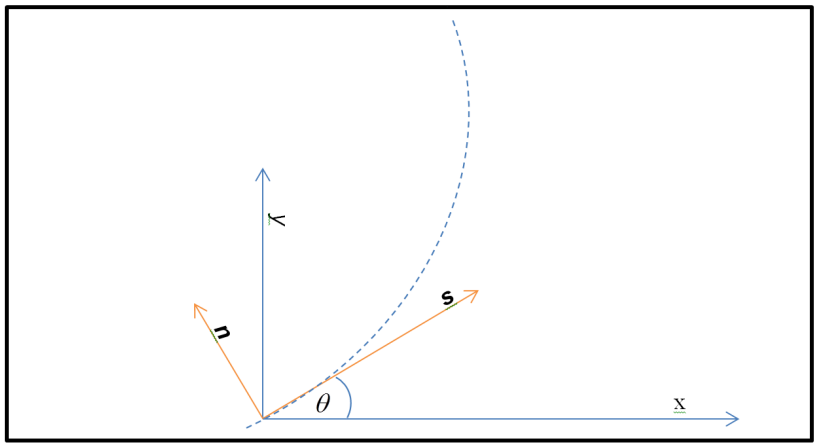


Figure 5.1: *The flow streamline path and the direction of dispersion stresses.*

$$v^* = u\frac{H}{R_s}f_n + vf_s \tag{5.24}$$

Which can subsequently be rewritten as:

$$u^* = uf_s - \frac{v}{|\boldsymbol{u}|}If_n \tag{5.25}$$

$$v^* = \frac{u}{|\boldsymbol{u}|}If_n + vf_s \tag{5.26}$$

The dispersion terms can be evaluated as:

$$\langle u^*u^*\rangle = u^2\left\langle f_s^2\right\rangle - 2\frac{uv}{|\boldsymbol{u}|}I\left\langle f_sf_n\right\rangle + \frac{v^2}{|\boldsymbol{u}|^2}I^2\left\langle f_n^2\right\rangle \tag{5.27}$$

$$\langle u^*v^*\rangle = uv\left\langle f_s^2\right\rangle + 2\frac{u^2-v^2}{|\boldsymbol{u}|}I\left\langle f_sf_n\right\rangle - \frac{uv}{|\boldsymbol{u}|^2}I^2\left\langle f_n^2\right\rangle \tag{5.28}$$

$$\langle v^*v^*\rangle = v^2\left\langle f_s^2\right\rangle + 2\frac{uv}{|\boldsymbol{u}|}I\left\langle f_sf_n\right\rangle + \frac{u^2}{|\boldsymbol{u}|^2}I^2\left\langle f_n^2\right\rangle \tag{5.29}$$

Here, we applied Delft3D approach. In Delft3D approach, the following propositions are applied:

- $\langle f_s^2\rangle$ is $\mathcal{O}(1)$ but hardly varies Olesen(1987, p49)
- $I^2\langle f_n^2\rangle$ is small for mildly curving, shallow water flow
- $\langle f_sf_n\rangle = 5\alpha - 15.6\alpha^2 + 37.5\alpha^3$ (cf. Delft3D-FLOW manual eq. 9.155)

Under these assumptions the dispersion stresses can be simplified to:

$$T_{xx} = \langle u^*u^*\rangle = -2\frac{uv}{|\boldsymbol{u}|}I\left\langle f_sf_n\right\rangle \tag{5.30}$$

$$T_{xy} = \langle u^* v^* \rangle = \frac{u^2 - v^2}{|\boldsymbol{u}|} I \langle f_s f_n \rangle \tag{5.31}$$

$$T_{yy} = \langle v^* v^* \rangle = 2\frac{uv}{|\boldsymbol{u}|} I \langle f_s f_n \rangle \tag{5.32}$$

5.6.2 Numerical schemes

In this section, the numerical techniques, implemented for calculation of secondary flow, are described. It contains the calculation of the streamline curvature, spiral motion intensity, direction of bedload transport and the effect on the momentum equations.

5.6.2.1 Calculation of streamline curvature

It is known that Perot reconstruction leads to inaccuracies in calculation of the streamlines curvature for the case with unstructured non-uniform grids. In general it is only first order accurate on unstructured meshes (Perot, 2000) and the velocity gradients derived from these reconstructed fields are inconsistent (Shashkov *et al.*, 1998) and can result in erroneous estimates of the streamline curvature, leading to non-physical solutions. However, on uniform meshes, owing to fortunate cancellations on account of grid uniformity, this methodology leads to second order accurate velocities and consistent gradients (Shashkov *et al.*, 1998; Natarajan and Sotiropoulos, 2011).

In order to avoid the inaccuracy leading from Perot reconstruction on non-uniform grids, we reconstructed the velocity gradients by a higher order reconstruction method. There are two popular methods, namely Green-Gauss and least square reconstructions, which are widely used in the pervious studies (Mavriplis, 2003) and they are also widely implemented in the existing commercial software (i.e. ANSYS Fluent). The least-squares constructions represent a linear function exactly for vertex and cell-centered discretizations on arbitrary mesh types, unrelated to mesh topology, while the Green-Gauss construction represents a linear function exactly only for a vertex-based discretization on simple elements, such as triangles or tetrahedra (Mavriplis, 2003). Hence, we used least square reconstruction for its ability in handling with all type of grid structures.

The least-squares gradient construction is obtained by solving for the values of the gradients which minimize the sum of the squares of the differences between neighboring values and values extrapolated from the point i under consideration to the neighboring locations. The objective to be minimized is given as

$$\sum_{k=1}^{N} w_{ik}^2 E_{ik}^2 \tag{5.33}$$

where w is a weighting function and E represents the error. Considering a linear reconstruction, and using Taylor series, we have

$$u_k = u_i + \left.\frac{\partial u}{\partial x}\right|_i (x_k - x_i) + \left.\frac{\partial u}{\partial y}\right|_i (y_k - y_i) + E\left(\Delta x^2, \Delta y^2\right) \tag{5.34}$$

Considering $dx_{ik} = x_k - x_i$, $dy_{ik} = y_k - y_i$ and $du_{ik} = u_k - u_i$, it yields

$$E_{ik}^2 = \left(-du_{ik} + \left.\frac{\partial u}{\partial x}\right|_i dx_{ik} + \left.\frac{\partial u}{\partial y}\right|_i dy_{ik} \right)^2 \tag{5.35}$$

A system of two equations for the two gradients $\partial u/\partial x$ and $\partial u/\partial y$ is obtained by solving the minimization problem

$$\frac{\partial \sum_{k=1}^{N} w_{ik}^2 E_{ik}^2}{\partial u_x} = 0 \tag{5.36}$$

$$\frac{\partial \sum_{k=1}^{N} w_{ik}^2 E_{ik}^2}{\partial u_y} = 0 \tag{5.37}$$

Equations 5.36 and 5.37 lead to the following set of equations

$$a_i \frac{\partial u}{\partial x} + b_i \frac{\partial u}{\partial y} = d_i \tag{5.38}$$

$$b_i \frac{\partial u}{\partial x} + c_i \frac{\partial u}{\partial y} = e_i \tag{5.39}$$

where

$$a_i = \sum_{k=1}^{N} w_{ik}^2 dx_{ik}^2 \tag{5.40}$$

$$b_i = \sum_{k=1}^{N} w_{ik}^2 dx_{ik} dy_{ik} \tag{5.41}$$

$$c_i = \sum_{k=1}^{N} w_{ik}^2 dy_{ik}^2 \tag{5.42}$$

$$d_i = \sum_{k=1}^{N} w_{ik}^2 du_{ik} dx_{ik} \tag{5.43}$$

$$e_i = \sum_{k=1}^{N} w_{ik}^2 du_{ik} dy_{ik} \tag{5.44}$$

The above system of equations for the gradients is then easily solved using Cramer's rule. This method is shown to have a second order accuracy (Mavriplis, 2003).

For the unweighted case ($w_{ik} = 1$), the determinant corresponds to a difference in quantities of the order $\mathcal{O}\left(dx^4\right)$, which may lead to ill-conditioned systems. This may be the motivation for investigations into alternate solution techniques for the least-squares construction, such as the QR factorization method advocated in. Note that when inverse distance weighting is used ($w_{ik} = \frac{1}{\sqrt{dx_{ik}^2 + dy_{ik}^2}}$), the determinant scales as $\mathcal{O}\left(1\right)$, and the system is much better conditioned.

5.6.2.2 Calculation of spiral flow intensity

As the spiral flow intensity is in the form of transport equation, it is calculated using the existing transport function in D-Flow FM. This is achieved by calculating the source term of Equation 5.7 and linking it to the existing code.

5.6.2.3 Calculation of bedload sediment direction

The direction of bedload sediment is calculated by implementing Equation 5.15 in D-Flow FM. The calculated spiral intensity and velocity field is used to find the final angle of the acting shear stress.

5.6.2.4 Calculation of dispersion stresses

The dispersion stresses T_{xx}, T_{xy} and T_{yy} are calculated parametrically by Equations 5.27 to 5.29. In order to calculate the effect of these stresses on the momentum equations, calculation of derivatives, and hence a reconstruction technique, is necessary. This is achieved by implementing the same reconstruction technique used in section 5.6.2.

5.7 Wave-current interaction

[yet empty]

5.8 Heat flux models

[yet empty]

5.9 Tide generating forces

[yet empty]

5.10 Hydraulic structures

[yet empty]

5.11 Flow resistance: bedforms and vegetation

[yet empty]

6 Numerical approach

D-Flow FM solves the two- and three-dimensional shallow-water equations. We will focus on two dimensions first. The shallow-water equations express conservation of mass and momentum and can be put into the following form:

$$\frac{\partial h}{\partial t} + \nabla \cdot (h\boldsymbol{u}) = 0, \tag{6.1}$$

$$\frac{\partial \boldsymbol{u}}{\partial t} + \frac{1}{h}(\nabla \cdot (h\boldsymbol{u}\boldsymbol{u}) - \boldsymbol{u}\nabla \cdot (h\boldsymbol{u})) = -g\nabla\zeta + \frac{1}{h}\nabla \cdot (\nu h(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T)) + \frac{\boldsymbol{\tau}}{h}, \tag{6.2}$$

where ζ is the water level, h the water depth, $\boldsymbol{u}$ the velocity vector, g the gravitational acceleration, ν the viscosity and $\boldsymbol{\tau}$ is the bottom friction:

$$\boldsymbol{\tau} = -\frac{g}{C^2}\|\boldsymbol{u}\|\boldsymbol{u}, \tag{6.3}$$

with C being the Chézy coefficient. The equations are complemented with appropriate initial conditions and water-level and/or velocity boundary conditions. The boundary conditions are explained in Section 6.3. The initial conditions will not be discussed further.

6.1 Spatial discretization

The spatial discretization is performed in a staggered manner, i.e. velocity normal components u_j are defined at the cell faces j, with face normal vector $\boldsymbol{n}_j$, and the water levels ζ_k at cell centers k. See Fig. 6.1 for an example. Note that in this example $k \in \{1, 2\}$ and $j \in \{1, 2, \ldots, 5\}$.

6.1.1 Connectivity

We will firstly introduce some notation that expresses the connectivity of computational cells, faces and mesh nodes, see Fig. 6.2. We say that

- cell k contains vertical faces j that are in the set $\mathcal{J}(k)$,
- cell k contains mesh nodes i that are in the set $\mathcal{I}(k)$,
- face j contains mesh nodes $i_L(j)$ and $i_R(j)$ respectively, given some orientation of face j,
- face j neighbors cells $L(j)$ and $R(j)$ respectively, given some orientation of face j.

Thus, in the example of Fig. 6.1:

$\mathcal{J}(1) = \{1, 2, 3\}, \mathcal{J}(2) = \{4, 5, 1\}$,
$\mathcal{I}(1) = \{1, 2, 3\}, \mathcal{I}(2) = \{4, 2, 1\}$,
$i_L(1) = 2, i_R(1) = 1, i_L(2) = 3, i_R(2) = 2$, et cetera, and
$L(1) = 1$ and $R(1) = 2$.

The orientation of face j with respect to cell $i \in \{i_L(j), i_R(j)\}$ is accounted for by $1_{j,k}$ in the following manner:

$$1_{j,k} = \begin{cases} 1, & L(j) = k \quad (\boldsymbol{n}_j \text{ is outward normal of cell } k), \\ -1, & R(j) = k \quad (\boldsymbol{n}_j \text{ is inward normal of cell } k), \end{cases} \tag{6.4}$$

where $\boldsymbol{n}_j$ is the normal vector of face j, defined positive in the direction from (neighboring) cell $L(j)$ to $R(j)$. In the example of Fig. 6.1 $1_{1,1} = 1$ and $1_{1,2} = -1$.

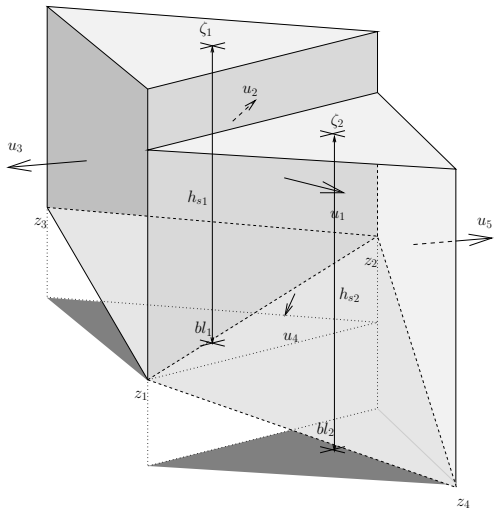


Figure 6.1: *Discretization of the water-level ζ_k, bed-levels z_i and bl_k, water depth h_i and face-normal velocities u_j; $i \in \{1, \ldots, 4\}$, $k \in \{1, 2\}$ and $j \in \{1, \ldots, 5\}$*

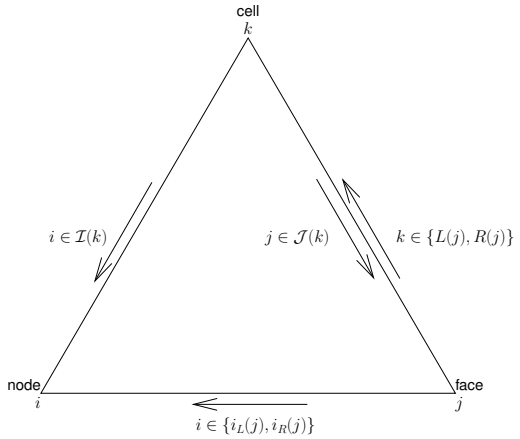


Figure 6.2: *Connectivity of cells, faces and nodes*

The connectivity translates directly to administration in the D-Flow FM code as follows:
$\mathcal{J}(k)$: `nd(k)%ln`,
$\mathcal{I}(k)$: `nd(k)%nod`,
$i_L(j)$: `lncn(2,j)`, $i_R(j)$: `lncn(1,j)`,
$L(j)$: `ln(1,j)`, $R(j)$: `ln(2,j)`.

6.1.2 Bed geometry: bed level types

The bed geometry is user defined by specifying the cell-centered values ("bed level type"=1), by its face-based values ("bed level type=2"), or by the values at the mesh nodes (other bed level types). In the first two cases, the bed is assumed piecewise constant. In the other cases, the bed is assumed piecewise linear or piecewise constant, depending on the "bed level type" and the term to be discretized at hand.

The bed geometry appears in the discretization of the governing equations by means of its cell-centered value bl_i and its face-based values bob_{1j} and bob_{2j}. Given some orientation, bob_{1j} represents the left-hand side bed level at face j and bob_{2j} the right-hand side level, respectively. In such a manner a linear representation of the bed from bob_{1j} to bob_{2j} is obtained at face j. It is used, for example, in the computation of the flow area A_{uj} as we will see later. Note that for the sake of clarity we will not discuss one-dimensional modeling at this occasion.

In case of bed level type "1", the cell-centered levels are (user) defined in bl_k and the node-based levels z_i from the mesh are disregarded. Similarly, for bed level type "2" the face-based bed levels are defined in b_{uj}. In the other cases the bed levels z_i are defined at the mesh nodes. The cell-based bed levels bl_k are then derived from the nodal values as shown in Algorithm (1). Refer to Section 6.1.1 for the definitions of sets $\mathcal{I}(k)$ and $\mathcal{J}(k)$ respectively.

The face-based bed levels bob_{1j} and bob_{2j} are derived as is also shown in Algorithm (1). Refer to Section 6.1.1 for the definitions of $i_L(j)$, $i_R(j)$, $L(j)$ and $R(j)$ respectively. We will not discuss the "conveyance type", but only mention that it's something bad, something really bad.

6.1.3 Continuity equation

The continuity equation Eqn. (6.1) is spatially discretized as

$$\frac{\mathrm{d}V_k}{\mathrm{d}t} = -\sum_{j\in\mathcal{J}(k)} A_{uj}u_j 1_{j,k}, \tag{6.7}$$

where $\mathcal{J}(k)$ is the set of vertical faces that bound cell k and $1_{j,k}$ accounts for the orientation of face j with respect to cell k, see Section 6.1.1.

Furthermore, V_k is the volume of the water column at cell k computed with Algorithm (18), not discussed here, A_{uj} approximates the flow area of face j, computed with Algorithm (3), and h_{uj} is the water depth at face j, computed with Algorithm (2). Algorithms (3) and (2) will be discussed momentarily.

Face-based water depth h_{uj}

In contrast to the bed, which may vary linearly for bed level types 3, 4 and 5 and conveyance types ≥ 1, the water level is assumed constant within a face. The water level at the faces are reconstructed from the cell-centered water levels with an upwind approximation. The face-based water depth h_{uj} is then defined as the maximum water depth in face j, see Fig. 6.3. It

Algorithm 1 setbobs: compute face-based bed-levels bob_{1j} and bob_{2j} and cell-based bed-level bl_k

$$bl_k = \begin{cases} \text{user specified}, & \text{bed level type} = 1, \\ \min\limits_{j\in\mathcal{J}(k)} (b_{uj}), & \text{bed level type} = 2, \\ \min\limits_{j\in\mathcal{J}(k)} \left[\min(bob_{1j}, bob_{2j})\right], & \text{bed level type} \in \{3,4,5\}, \\ \sum\limits_{i\in\mathcal{I}(k)} z_i / \sum\limits_{i\in\mathcal{I}(k)} 1, & \text{bed level type} = 6 \\ z_{k\text{uni}}, & \text{otherwise (default value).} \end{cases} \tag{6.5}$$

$$bob_{1,2_j} = \begin{cases} \max(bl_{L(j)}, bl_{R(j)}), & \text{bed level type} = 1, \\ b_{uj}, & \text{bed level type} = 2, \\ z_{i_{L,R}(j)}, & \text{bed level type} \in \{3,4,5\} \quad \wedge \quad \text{conveyance type} \geq 1, \\ \frac{1}{2}(z_{i_L(j)} + z_{i_R(j)}), & \text{bed level type} = 3 \quad \wedge \quad \text{conveyance type} < 1, \\ \min(z_{i_L(j)}, z_{i_R(j)}), & \text{bed level type} = 4 \quad \wedge \quad \text{conveyance type} < 1, \\ \max(z_{i_L(j)}, z_{i_R(j)}), & \text{bed level type} = 5 \quad \wedge \quad \text{conveyance type} < 1, \\ \max(bl_{L(j)}, bl_{R(j)}), & \text{bed level type} = 6. \end{cases} \tag{6.6}$$

is computed with Algorithm (2).

Algorithm 2 sethu: approximate the face-based water depth h_{uj} with an upwind reconstruction of the water level at the faces

$$h_{uj} = \begin{cases} \zeta_{L(j)} - \min(bob_{1j}, bob_{2j}), & u_j > 0 \quad \vee \quad u_j = 0 \wedge \zeta_{L(j)} > \zeta_{R(j)}, \\ \zeta_{R(j)} - \min(bob_{1j}, bob_{2j}), & u_j < 0 \quad \vee \quad u_j = 0 \wedge \zeta_{L(j)} \leq \zeta_{R(j)}. \end{cases} \tag{6.8}$$

In the example of Fig. 6.1, the water-level at face 1, assumed constant in the face as indicted in the figure, is approximated by ζ_1 if $u_1 > 0$, ζ_2 if $u_2 < 0$ and $\max(\zeta_1, \zeta_2)$ if $u_1 = 0$, respectively.

Remark 6.1.1. We will see later in Eqn. (6.73) that the time-integration of the continuity equation is implicit/explicit. Nonetheless, the upwind direction of h_{uj} is based on the velocity at the beginning of the time-step only.

Remark 6.1.2. The *upwind* reconstruction of h_{uj} from the cell-centered water levels is a first-order approximation (possibly based on the incorrect upwind direction, see previous remark). Higher-order reconstruction is not available at this moment, regardless of the option `limtyphu`.

Wet cross-sectional area A_{uj}

By the flow area A_{uj} the wet cross-sectional area of the face j is meant. Since the bed in face j is linearly varying from bob_{1j} to bob_{2j}, and the water in the face is assumed at constant level $\min(bob_{1j}, bob_{2j}) + h_{uj}$, the wet area can be computed with Algorithm (3). Note that w_{uj} is the width of face j, see Fig. 6.3, and Δb_j is the cross-sectional bed variation.

Remark 6.1.3. The exception for the case $\Delta b_j < 10^{-3} w_{uj}$ in Eqn. (6.10) should maybe be

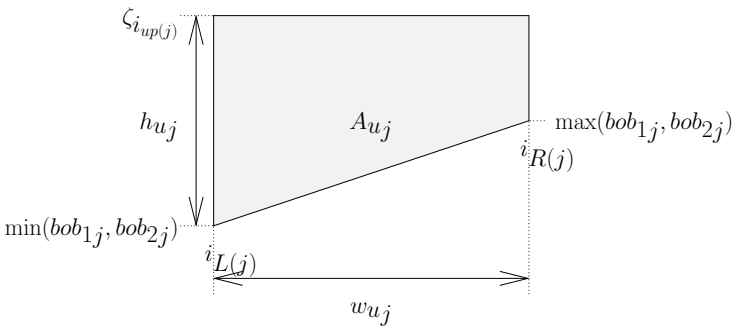


Figure 6.3: *Flow area A_{uj} and face-based water depth h_{uj}*

Algorithm 3 setau: compute the flow area A_{uj}

$$\Delta b_j = \max(bob_{1j}, bob_{2j}) - \min(bob_{1j}, bob_{2j}), \tag{6.9}$$

$$A_{uj} = \begin{cases} w_{uj} h_{uj}, & \Delta b_j < 10^{-3} w_{uj}, \\ w_{uj} h_{uj} \min(\frac{h_{uj}}{\Delta b_j}, 1) \left(1 - \frac{1}{2} \min(\frac{\Delta b_j}{h_{uj}}, 1)\right), & \text{otherwise.} \end{cases} \tag{6.10}$$

reconsidered.

Remark 6.1.4. In case of bed level type 3 and conveyance types ≥ 1, the bed is assumed lineary varying within a face, see Algorithm (1). This is accounted for in the computation of the wet cross-sectional areas of the vertical faces, see Algorithm (3). A lineary varying bed, on the other hand, is *not* accounted for in the computation water column volumes V_k in Algorithm (18), without non-linear iterations (explained later). This seems *inconsistent* when we are employing a finite volume approximation as in e.g. Eqn. (6.7).

6.1.4 Momentum equation

The momentum equation is discretized at the *faces* and in face-normal direction, see Fig. 6.4. In this figure Δx_j is the distance between the two neigboring cell centers, i.e. $\Delta x_j = \|\boldsymbol{x}_{R(j)} - \boldsymbol{x}_{L(j)}\|$, and w_{uj} is, as explained before, the width of face j.

Making use of the properties of an orthogonal mesh, the water level-gradient term projected in the face-normal direction is discretized as

$$g \nabla \zeta|_j \cdot \boldsymbol{n}_j \approx \frac{g}{\Delta x_j} (\zeta_{R(j)} - \zeta_{L(j)}). \tag{6.11}$$

The bed friction term is discretized as

$$\frac{\boldsymbol{\tau}}{h}\Big|_j \cdot \boldsymbol{n}_j \approx -\frac{g \|\boldsymbol{u}_j\|}{C^2 \hat{h}_j} u_j, \tag{6.12}$$

where $\hat{h}_j$ acts as an hydraulic radius whose computation depends on the "conveyance type". Its precise discretization is not discussed here. Furthermore, advection and diffusion are discretized as

$$\boldsymbol{n}_j \cdot \left[\frac{1}{h}(\nabla \cdot (h\boldsymbol{u}\boldsymbol{u}) - \boldsymbol{u}\nabla \cdot (h\boldsymbol{u})) - \frac{1}{h}\nabla \cdot (\nu h(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^t))\right]_j \approx \mathcal{A}_{\mathrm{ij}} u_j + \mathcal{A}_{\mathrm{ej}}. \tag{6.13}$$

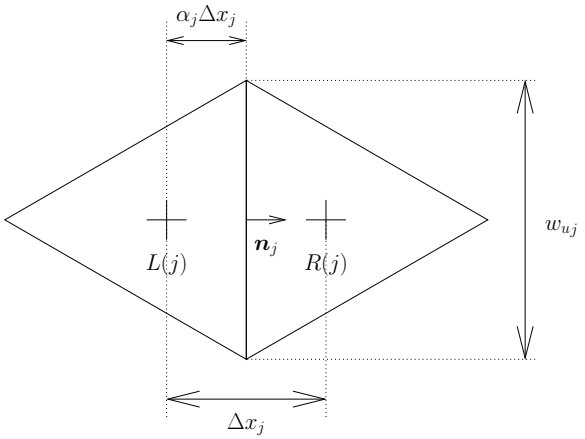


Figure 6.4: *Computational cells $L(j)$ and $R(j)$ neighboring face j; water levels are stored at the cell circumcenters, indicated with the $+$-sign*

The terms $\mathcal{A}_{\mathrm{i}j}$ and $\mathcal{A}_{\mathrm{e}j}$ will be discussed in more detail hereafter. These terms play an important role in the D-Flow FM code and are called `advi` and `adve`, respectively.

Summing up, the spatial discretization of Eqn. (6.2) reads

$$\frac{\mathrm{d}u_j}{\mathrm{d}t} = -\frac{g}{\Delta x_j}\left(\zeta_{R(j)} - \zeta_{L(j)}\right) - \mathcal{A}_{\mathrm{i}j} u_j - \mathcal{A}_{\mathrm{e}j} - \frac{g\|\boldsymbol{u}_j\|}{C^2 h} u_j. \tag{6.14}$$

Momentum advection

It would be a clear advantage if the momentum equation was discretized conservatively, especially for flows with discontinuities such as hydraulic jumps. This is not easily achieved in case of staggered, unstructured meshes. Nonetheless, Perot (2000) shows how to achieve momentum conservation in similar circumstances for the incompressible Navier Stokes equations. This approach is applied to the shallow water equations in Kramer and Stelling (2008) and Kleptsova *et al.* (2010). However, subtleties exist in the formulations as pointed out in Borsboom (2013). The various advection schemes in D-Flow FM differ on these subtleties.

The difficulty with the staggered layout on unstructured meshes is that we only solve the momentum equation in face-normal direction. We could, in principle, formulate a control volume for each face-normal velocity, but are unable to define conservative fluxes, as we do not solve for the *whole* momentum vector as we would do with a collocated arrangement of the unknowns. To this end, Perot (2000) pursues conservation of the full *reconstructed* cell-centered momentum vector. The advection operator is firstly discretized at cell centers, as if we were dealing with a collocated layout of our unknowns, and subsequently interpolated back to the faces and projected in face-normal direction.

Remark 6.1.5. Perot (2000) shows that the reconstruction from face-normal quantities to cell-centered vectors and the interpolation from cell-centered vector to face-normal quantities need to satisfy certain demands. We are not free to choose a reconstruction to our liking and the accuracy may even be compromised on irregular meshes.

The application of this approach to the shallow-water equations as in Kramer and Stelling (2008) and Kleptsova *et al.* (2010) is non-trivial. Complicating matters significantly is that we are not solving in conservative, but in primitive variables. As pointed out in Borsboom (2013),

the discretization of advection is subject to many subtleties. In D-Flow FM various advection schemes exist that vary on these subtleties.

Remark 6.1.6. It's fair to say that, formally speaking, *none* of the momentum advection schemes in D-Flow FM is conservative in the sense of Perot (2000).

The approach in Perot (2000) is as follows. Given some cell k, assume that cell-centered *conservative* advection is approximated by

$$\nabla \cdot (h\boldsymbol{u}\boldsymbol{u})|_{\Omega_k} \approx \boldsymbol{a}_k. \tag{6.15}$$

Face-normal advection at face j is then interpolated from its neighboring cells $L(j)$ and $R(j)$ as

$$\nabla \cdot (h\boldsymbol{u}\boldsymbol{u})|_{\Gamma_j} \cdot \boldsymbol{n}_j \approx \alpha_j \boldsymbol{a}_{L(j)} \cdot \boldsymbol{n}_j + (1-\alpha_j)\boldsymbol{a}_{R(j)} \cdot \boldsymbol{n}_j, \tag{6.16}$$

where α_j is the non-dimensional distance from the left cell center to the face, see Fig. 6.4. Note that the terms $-\boldsymbol{u}\nabla \cdot (h\boldsymbol{u})$ and $\frac{1}{h}$ in Eqn. (6.13) are due to our non-conservative formulation and do not appear in Eqn. (6.16). In the non-conservative formulation of Eqn. (6.13), their discretization contributes significantly to the subtle differences in the various schemes. See Borsboom (2013) for more details.

The cell-centered advection is discretized as

$$\boldsymbol{a}_k = \frac{1}{A(\Omega_k)} \sum_{j \in \mathcal{J}(k)} \boldsymbol{u}_{uj} q_j 1_{j,k}, \tag{6.17}$$

where $\boldsymbol{u}_{uj}$ is the reconstructed full velocity at the faces and $A(\Omega_k)$ the area of the control volume Ω_k, i.e. the cell. It is reconstructed from the cell-centered velocities $\boldsymbol{u}_c$ with an upwind scheme, e.g.

$$\boldsymbol{u}_{uj} = \begin{cases} \boldsymbol{u}_{L(j)}, & u_j > 0, \\ \boldsymbol{u}_{R(j)}, & u_j < 0. \end{cases}, \tag{6.18}$$

or with a higher-order limited version, discussed later. The cell-centered velocities in turn are reconstructed from the (primitive) face-normal velocities with Algorithm (6), also discussed later. Furthermore, flux q_j is derived from the face-normal velocity as

$$q_j = A_{uj} u_j, \tag{6.19}$$

see also Algorithm (19), explained later when we will discuss the time discretization.

The term $-\boldsymbol{u}\nabla \cdot (h\boldsymbol{u})$ is the so-called "storage term" and is due to our non-conservative formulation of the momentum equation. It originates from the substitution of the continuity equation in the *conservatively* formulated momentum equation. Glancing ahead at our temporal discretization, we observe the following. If we want our discretization to be discretely conservative, we need to substitute the continuity equation after spatial *and* temporal discretization, see Eqn. (6.84) (explained later). This means that we require the fluxes in the storage term to be identical to the fluxes in the discrete continuity equation, Eqn. (6.73), i.e. q_j^{n+1}, where n denotes the time level:

$$-\boldsymbol{u}\nabla \cdot (h\boldsymbol{u})|_k^n \approx -\frac{\boldsymbol{u}}{A(\Omega_k)} \sum_{j \in \mathcal{J}(k)} q_j^{n+1} 1_{j,k}, \tag{6.20}$$

where we do not mention at which time level term $\frac{\boldsymbol{u}}{A(\Omega_k)}$ is to be evaluated. Eqn. (6.20) leads to an implicit contribution to the discrete advection for $\theta > 0$. However, In D-Flow FM the storage term is always discretized explicitly in time. It is based on explicit fluxes q_j^n or on $q_{a_j}^n$, depending on the advection scheme.

Remark 6.1.7. Since the fluxes in the storage term are at the old time level, in contrast to the fluxes in the continuity equation, advection in D-Flow FM is non-conservative for non-stationary flows and $\theta > 0$.

Given the discretization of the conservatively formulated advection of Eqns. (6.16) and (6.17) and the storage term of Eqn. (6.20), the advection can now be composed in general form as

$$\begin{aligned} \mathcal{A}_{\mathrm{e}j} \;=\; & A_{Lj} \sum_{l \in \mathcal{J}^*(L(j))} q_l^* 1_{l,L(j)} \boldsymbol{u}_{ul} \cdot \boldsymbol{n}_j - q_l^{**} 1_{l,L(j)} (1 - \theta_{l,L(j)}) u^*_{Lj} \;+ \\ & A_{Rj} \sum_{l \in \mathcal{J}^*(R(j))} q_l^* 1_{l,R(j)} \boldsymbol{u}_{ul} \cdot \boldsymbol{n}_j - q_l^{**} 1_{l,R(j)} (1 - \theta_{l,R(j)}) u^*_{Rj}, \end{aligned} \tag{6.21}$$

and

$$\mathcal{A}_{\mathrm{i}j} = -A_{Lj} \sum_{l \in \mathcal{J}^*(L(j))} q_l^{**} 1_{l,L(j)} \theta_{l,L(j)} \;-\; A_{Rj} \sum_{l \in \mathcal{J}^*(R(j))} q_l^{**} 1_{l,R(j)} \theta_{l,R(j)}, \tag{6.22}$$

where $\mathcal{J}^*$, q_l^*, q_l^{**}, $\theta_{l,\{L,R\}(j)}$, $u^*_{\{L,R\}_j}$, A_{Lj}, A_{Rj} vary for the different advection schemes as described in Algorithm (4) and $\mathcal{J}^i$ is the set of faces with inward fluxes, i.e.

$$\mathcal{J}^i(k) = \{ j \in \mathcal{J}(k) | u_j 1_{j,k} < 0 \} \tag{6.23}$$

and

$$h_{uvj} = \max(\alpha_j h_{L(j)} + (1 - \alpha_j) h_{R(j)}, \epsilon_{h_\zeta}), \tag{6.24}$$

where ϵ_{h_ζ} is a threshold and $\sigma_{j,L(j)}$ and $\sigma_{j,R(j)}$ are face-based Courant numbers, computed as:

$$\sigma_{j,L(j)} = \begin{cases} \frac{1.4 \Delta t |q_{a\,j}|}{\alpha_j V_{L(j)} + (1-\alpha_j) V_{R(j)}}, & \sum_{j \in \mathcal{J}(L(j))} 1 = 3, \\ \frac{\Delta t |q_{a\,j}|}{\alpha_j V_{L(j)} + (1-\alpha_j) V_{R(j)}}, & \text{other}, \end{cases} \tag{6.25}$$

and

$$\sigma_{j,R(j)} = \begin{cases} \frac{1.4 \Delta t |q_{a\,j}|}{\alpha_j V_{L(j)} + (1-\alpha_j) V_{R(j)}}, & \sum_{j \in \mathcal{J}(R(j))} 1 = 3, \\ \frac{\Delta t |q_{a\,j}|}{\alpha_j V_{L(j)} + (1-\alpha_j) V_{R(j)}}, & \text{other}, \end{cases} \tag{6.26}$$

respectively. Note that $V_{A_u L(j)}$ and $V_{A_u R(j)}$ are undefined.

Cell center interpolation

We saw in the previous section that the cell-centered reconstructed full velocity vector $\boldsymbol{u}_c$ plays an important role in the discretization of the momentum advection. This section elaborates on its computation.

Following Perot (2000), and taking a cell k as a control volume, the full cell-centered velocity vector can be reconstructed from the face-normal components u_j by using the following approximation

$$\boldsymbol{u}_{ck} \approx \frac{1}{A(\Omega_k)} \int_{\Omega_k} \nabla \cdot (\boldsymbol{u}(\boldsymbol{x} - \boldsymbol{x}_{ck}))\, \mathrm{d}\Omega = \frac{1}{A(\Omega_k)} \int_{\partial\Omega_k} (\boldsymbol{x} - \boldsymbol{x}_{ck}) \boldsymbol{u} \cdot \boldsymbol{n}\, \mathrm{d}\Gamma, \tag{6.29}$$

where Ω_k is the control volume, i.e. the cell, $A(\Omega_k)$ its area and $\boldsymbol{n}$ is an outward normal.

Algorithm 4 advec:compute advection terms of the form
$\boldsymbol{n}_j \cdot \left[\frac{1}{h}(\nabla\cdot(h\boldsymbol{u}\boldsymbol{u}) - \boldsymbol{u}\nabla\cdot(h\boldsymbol{u}))\right]_j \approx \mathcal{A}_{\mathsf{i}j}u_j + \mathcal{A}_{\mathsf{e}j}$

compute higher-order accurate reconstructions of face-based velocity vector $\boldsymbol{u}_u$ from cell-centered velocity vectors u_c with Algorithm (10)
compute $\mathcal{A}_{\mathsf{e}}$ and $\mathcal{A}_{\mathsf{i}}$:

$$\begin{aligned}\mathcal{A}_{\mathsf{e}j} \;=\; & A_{Lj}\sum_{l\in\mathcal{J}^*(L(j))} q_l^* 1_{l,L(j)}\boldsymbol{u}_{ul}\cdot\boldsymbol{n}_j - q_l^{**}1_{l,L(j)}(1-\theta_{l,L(j)})u^*_{Lj} \;+\\ & A_{Rj}\sum_{l\in\mathcal{J}^*(R(j))} q_l^* 1_{l,R(j)}\boldsymbol{u}_{ul}\cdot\boldsymbol{n}_j - q_l^{**}1_{l,R(j)}(1-\theta_{l,R(j)})u^*_{Rj}\end{aligned} \tag{6.27}$$

$$\mathcal{A}_{\mathsf{i}j} \;=\; -A_{Lj}\sum_{l\in\mathcal{J}^*(L(j))} q_l^{**}1_{l,L(j)}\theta_{l,L(j)} \;-\; A_{Rj}\sum_{l\in\mathcal{J}^*(R(j))} q_l^{**}1_{l,R(j)}\theta_{l,R(j)} \tag{6.28}$$

with

iadv	$\mathcal{J}^*$	q_l^*	q_l^{**}	$\theta_{l,\{L,R\}(j)}$	$u^*_{\{L,R\}j}$	A_{Lj}	A_{Rj}
0	-	-	-	-	0	0	$\frac{1}{V_{L(j)}+V_{R(j)}}$
1	$\mathcal{J}$	q_{al}	q_l	0	u_j	$\frac{1}{V_{L(j)}+V_{R(j)}}$	$\frac{1}{V_{L(j)}+V_{R(j)}}$
2	$\mathcal{J}$	q_{al}	q_{al}	0	u_j	$\frac{1}{V_{L(j)}+V_{R(j)}}$	$\frac{1}{V_{L(j)}+V_{R(j)}}$
3, 33	$\mathcal{J}$	q_{al}	q_{al}	0	u_j	$\frac{\alpha_j}{\alpha_j V_{L(j)}+(1-\alpha_j)V_{R(j)}}$	$\frac{1-\alpha_j}{\alpha_j V_{L(j)}+(1-\alpha_j)V_{R(j)}}$
4	$\mathcal{J}^i$	q_{al}	q_{al}	0	u_j	$\frac{\alpha_j}{\alpha_j V_{L(j)}+(1-\alpha_j)V_{R(j)}}$	$\frac{1-\alpha_j}{\alpha_j V_{L(j)}+(1-\alpha_j)V_{R(j)}}$
5	$\mathcal{J}$	q_{al}	q_{al}	$\max(1-\frac{1}{\sigma_{l,\{L,R\}(j)}},0)$	u_j	$\frac{\alpha_j}{\alpha_j V_{L(j)}+(1-\alpha_j)V_{R(j)}}$	$\frac{1-\alpha_j}{\alpha_j V_{L(j)}+(1-\alpha_j)V_{R(j)}}$
6	$\mathcal{J}^i$	q_{al}	q_{al}	$\max(1-\frac{1}{\sigma_{l,\{L,R\}(j)}},0)$	u_j	$\frac{\alpha_j}{\alpha_j V_{L(j)}+(1-\alpha_j)V_{R(j)}}$	$\frac{1-\alpha_j}{\alpha_j V_{L(j)}+(1-\alpha_j)V_{R(j)}}$
7,9,11	$\mathcal{J}$	q_{al}	q_{al}	1	u_j	$\frac{\alpha_j}{\alpha_j V_{L(j)}+(1-\alpha_j)V_{R(j)}}$	$\frac{1-\alpha_j}{\alpha_j V_{L(j)}+(1-\alpha_j)V_{R(j)}}$
8,10,12	$\mathcal{J}^i$	q_{al}	q_{al}	1	u_j	$\frac{\alpha_j}{\alpha_j V_{L(j)}+(1-\alpha_j)V_{R(j)}}$	$\frac{1-\alpha_j}{\alpha_j V_{L(j)}+(1-\alpha_j)V_{R(j)}}$
30	$\mathcal{J}$	q_{al}	q_{al}	0	u_j	$\frac{1}{V_{L(j)}+V_{R(j)}}$	$\frac{1}{V_{L(j)}+V_{R(j)}}$
31	$\mathcal{J}$	q_{al}	q_{al}	0	$\boldsymbol{u}_{c\{L,R\}(j)}\cdot\boldsymbol{n}_j$	$\frac{\alpha_j}{V_{L(j)}}$	$\frac{1-\alpha_j}{V_{R(j)}}$
34	$\mathcal{J}$	q_{al}	q_{al}	0	u_j	$\frac{\alpha_j}{h_{wvj}b_{AL(j)}}$	$\frac{1-\alpha_j}{h_{wvj}b_{AR(j)}}$
36	$\mathcal{J}$	q_l	q_l	0	u_j	$\frac{\alpha_j}{V_{L(j)}}$	$\frac{1-\alpha_j}{V_{R(j)}}$
37	$\mathcal{J}$	q_l	q_l	0	u_j	$\frac{\alpha_j}{h_{wvj}b_{AL(j)}}$	$\frac{1-\alpha_j}{h_{wvj}b_{AR(j)}}$
38	$\mathcal{J}$	q_l	q_l	0	$\boldsymbol{u}_{c\{L,R\}(j)}\cdot\boldsymbol{n}_j$	$\frac{\alpha_j}{V_{Au\,L(j)}}$	$\frac{1-\alpha_j}{V_{Au\,R(j)}}$
333	$\mathcal{J}$	q_{al}	q_{al}	0	u_j	$\frac{\alpha_j}{V_{Au\,L(j)}}$	$\frac{1-\alpha_j}{V_{Au\,R(j)}}$

Remark 6.1.8. We will *not* discuss whether $\boldsymbol{u}_{ck}$ represents a cell-averaged or nodal value. Nevertheless, Eqn. (6.29) is a second order approximation if the center point is sufficiently close to the mass center of the control volume. Note that in our case the center point is the cell circumcenter, which can deviate considerably from the mass center for irregular meshes.

The discretization of Eqn. (6.29) in cell k is

$$\boldsymbol{u}_{ck} = \sum_{j\in\{l\in\mathcal{J}(k)|1_{l,k}=1\}} \boldsymbol{w}_{cLj}u_j \quad + \sum_{j\in\{l\in\mathcal{J}(k)|1_{l,k}=-1\}} \boldsymbol{w}_{cRj}u_j \tag{6.30}$$

with weight vectors $\boldsymbol{w}_{cLj}$ and $\boldsymbol{w}_{cRj}$ are computed with Algorithm (5), where b_{Ak} is the bed area of cell k.

Algorithm 5 setlinktocenterweights: compute weight vectors $\boldsymbol{w}_{cLj}$ and $\boldsymbol{w}_{cRj}$ in the cell-center reconstruction of Eqn. (6.30)

$$\boldsymbol{w}_{cLj} = \frac{\alpha_j \Delta x_j \boldsymbol{n}_j w_{uj}}{b_{AL(j)}} \tag{6.31}$$

$$\boldsymbol{w}_{cRj} = \frac{(1-\alpha_j) \Delta x_j \boldsymbol{n}_j w_{uj}}{b_{AR(j)}} \tag{6.32}$$

Remark 6.1.9. Cells that are cut by a dry-wet interface do not get any special treatment, i.e. dry faces (with formally undefined velocities) still appear in the reconstruction, with assumed zero velocity. Hence, cell centered velocity vectors near the dry-wet interface may be inconsistent with the local flow.

The cell centered velocities are computed with Algorithm (6), where h_{ζ_k} is the cell centered water depth at cell k, defined as $h_{\zeta_k} = \zeta_k - bl_\zeta$. Note that $\boldsymbol{u}_q$ is a 'discharge-averaged' reconstructed velocity vector. It is used for the tangential velocity v_j component at the faces:

$$v_j = \boldsymbol{n}_j \times \left(\alpha_j \boldsymbol{u}_{qL(j)} + (1-\alpha_j)\boldsymbol{u}_{qR(j)} \right) \tag{6.37}$$

Remark 6.1.10. It is not hard to see that the interpolation of $\boldsymbol{u}_q$ may be inconsistent, depending on the "bed level type", see Algorithms (1) and (2), and the bed geometry itself.

Note that Algorithm (6) also sets a first-order upwind approximation of $\boldsymbol{u}_u$, necessary in momentum advection, see Eqn. (6.17). Higher order corrections are added in Algorithm (10), explained later.

Nodal interpolation

In the discretization of horizontal momentum diffusion and in the bed friction for "conveyance type" ≥ 3, node-based velocity vectors $\boldsymbol{u}_n$ appear. This section elaborates on their computation.

The nodal velocity vectors are interpolated from the cell-centered velocity vectors $\boldsymbol{u}_c$, which were, in turn, interpolated from the face-normal velocities u, see the previous section.

Given some available cell-centered data, say Φ_c (e.g. one of the components of the velocity vector), we can define a control volume as indicated in Fig. 6.5 and interpolate to the nodes, say Φ_n, as follows:

Algorithm 6 setucxucyucxuucyu: reconstruct cell centered velocity vectors $\boldsymbol{u}_c$ and $\boldsymbol{u}_q$, and set first-order upwind fluxes $\boldsymbol{u}_u^L$

$$\boldsymbol{u}_{q_k} = \frac{1}{h_{\zeta_k}} \left(\sum_{j \in \{l \in \mathcal{J}(k) | 1_{l,k}=1\}} h_{uj} \boldsymbol{w}_{cL_j} u_j \quad + \sum_{j \in \{l \in \mathcal{J}(k) | 1_{l,k}=-1\}} h_{uj} \boldsymbol{w}_{cR_j} u_j \right) \tag{6.33}$$

if iPerot $= 2$ **then**

$$\boldsymbol{u}_{ck} = \boldsymbol{u}_{q_k} \tag{6.34}$$

else

$$\boldsymbol{u}_{ck} = \sum_{j \in \{l \in \mathcal{J}(k) | 1_{l,k}=1\}} \boldsymbol{w}_{cL_j} u_j \quad + \sum_{j \in \{l \in \mathcal{J}(k) | 1_{l,k}=-1\}} \boldsymbol{w}_{cR_j} u_j \tag{6.35}$$

end if

$$\boldsymbol{u}_{uj} = \begin{cases} \boldsymbol{u}_{cL(j)}, & q_{aj} > 0 \\ \boldsymbol{u}_{cR(j)}, & q_{aj} < 0 \\ 0, & q_{aj} = 0 \end{cases} \tag{6.36}$$

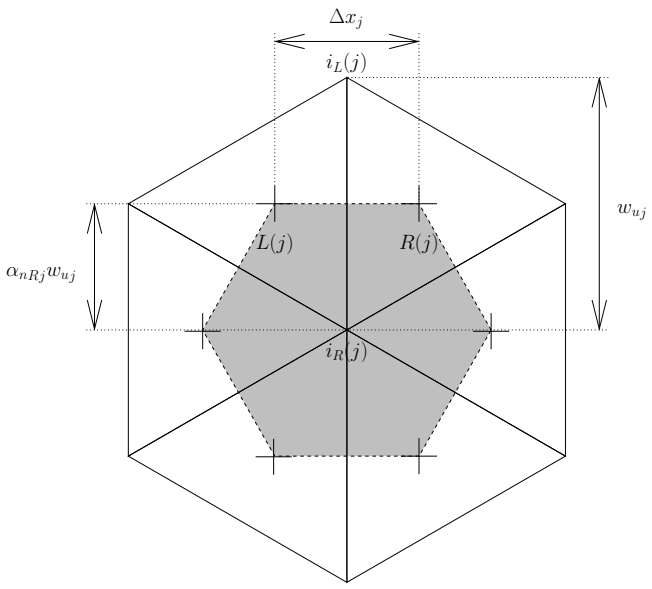


Figure 6.5: *Nodal interpolation from cell-centered values; contribution from face j to node $i_R(j)$; the shaded area indicates the control volume*

$$\Phi_n \approx \frac{1}{2A(\Omega_n)} \int_{\Omega_n} \nabla \cdot (\Phi(\boldsymbol{x} - \boldsymbol{x}_n))\, \mathrm{d}\Omega = \frac{1}{2A(\Omega_n)} \int_{\partial\Omega_n} \Phi(\boldsymbol{x} - \boldsymbol{x}_n) \cdot \boldsymbol{n}\, \mathrm{d}\Gamma, \quad (6.38)$$

where Ω_n is the node-based control volume, $A(\Omega_n)$ its area, $\boldsymbol{n}$ the outward normal vector and $\mathbf{x}_n$ are the node coordinates.

Remark 6.1.11. Eqn. (6.41) is a second order approximation if the node is located sufficiently close to the mass center of the control volume. In the example of Fig. 6.5 this is indeed the case, but not necessarily for general meshes.

The discretization of Eqn. (6.41) at node i is

$$\Phi_{ni} = \sum_{j\in\{l|i_L(l)=i\}} w_{nLj} \frac{1}{2}(\Phi_{cL(j)} + \Phi_{cR(j)}) + \sum_{j\in\{l|i_R(l)=i\}} w_{nRj} \frac{1}{2}(\Phi_{cL(j)} + \Phi_{cR(j)}), \quad (6.39)$$

with weights w_{nLj} and w_{nRj} computed as

$$w_{nLj} = \frac{\frac{1}{2}\alpha_{nLj}\Delta x_j w_{uj}}{\sum_{l\in\{m|i_L(m)=i_L(j)\}} \frac{1}{2}\alpha_{nLj}\Delta x_l w_{ul} + \sum_{l\in\{l|i_R(m)=i_L(j)\}} \frac{1}{2}\alpha_{nRl}\Delta x_l w_{ul}} \quad (6.40)$$

$$w_{nRj} = \frac{\frac{1}{2}\alpha_{nRj}\Delta x_j w_{uj}}{\sum_{l\in\{m|i_L(m)=i_R(j)\}} \frac{1}{2}\alpha_{nLl}\Delta x_l w_{ul} + \sum_{l\in\{m|i_R(m)=i_R(j)\}} \frac{1}{2}\alpha_{nRl}\Delta x_l w_{ul}} \quad (6.41)$$

Note that $\alpha_{nLj}\Delta x_j$ and $\alpha_{nRj}\Delta x_j$ approximate the components of $(\boldsymbol{x}-\boldsymbol{x}_n)\cdot\boldsymbol{n}$ in Eqn. (6.41) for node i and face j, which in D-Flow FM are computed as

$$\alpha_{nLj} = \frac{\|\frac{1}{2}(\boldsymbol{x}_{\zeta_{L(j)}} + \boldsymbol{x}_{\zeta_{R(j)}}) - \boldsymbol{x}_{ni_L(j)}\|}{\|\frac{1}{2}(\boldsymbol{x}_{\zeta_{L(j)}} + \boldsymbol{x}_{\zeta_{R(j)}}) - \boldsymbol{x}_{ni_L(j)}\| + \|\frac{1}{2}(\boldsymbol{x}_{\zeta_{L(j)}} + \boldsymbol{x}_{\zeta_{R(j)}}) - \boldsymbol{x}_{ni_R(j)}\|} \quad (6.42)$$

$$\alpha_{nRj} = 1 - \alpha_{nLj}, \quad (6.43)$$

where $\boldsymbol{x}_{\zeta_k}$ are the coordinates of cell-center k and $\boldsymbol{x}_{ni}$ are the coordinates of mesh node i, respectively.

Remark 6.1.12. A more straightforward approach, employing the properties of an orthogonal mesh and using $w_{uj} := \|\boldsymbol{x}_{nR(j)} - \boldsymbol{x}_{nL(j)}\|$, would be:

$$\alpha_{nLj} = \frac{(\frac{1}{2}(\boldsymbol{x}_{\zeta_{L(j)}} + \boldsymbol{x}_{\zeta_{R(j)}}) - \boldsymbol{x}_{ni_L(j)}) \cdot (\boldsymbol{x}_{ni_R(j)} - \boldsymbol{x}_{ni_L(j)})}{w_{uj}^2}. \quad (6.44)$$

In D-Flow FM the interpolation of cell-centered velocity vectors to nodal velocity vectors is as in Eqn. (6.41). That is, when "jacomp" $\neq 1$ and we do not consider mesh boundaries. The quantity Φ is to be replaced by both components of the velocity vector, respectively, i.e.

$$\boldsymbol{u}_{ni} = \sum_{j\in\{l|i_L(l)=i\}} w_{nLj} \frac{1}{2}(\boldsymbol{u}_{cL(j)} + \boldsymbol{u}_{cR(j)}) + \sum_{j\in\{l|i_R(l)=i\}} w_{nRj} \frac{1}{2}(\boldsymbol{u}_{cL(j)} + \boldsymbol{u}_{cR(j)}) \quad (6.45)$$

When "jacomp" $= 1$, however, the two components of the velocity vector (u_x, u_y), get different weights. If we say $\boldsymbol{u}_c =: (u_{cx}, u_{cy})^t$ and $\boldsymbol{u}_n =: (u_{nx}, u_{ny})^t$, then the interpolation becomes as performed by Algorithm (7). The weights w_{nxL}, w_{nxR}, w_{nyL} and w_{nyR} are computed with Algorithm (8), where $\boldsymbol{e}_x$ and $\boldsymbol{e}_y$ are the unit vectors in x- and y-direction respectively. The exceptions at mesh boundaries remain unmentioned.

Algorithm 7 setcornervelocities: interpolate nodal velocity vectors $\boldsymbol{u}_n = (u_{nx}, u_{ny})^t$ from cell-centered velocity vectors $\boldsymbol{u}_c = (u_{cx}, u_{cy})^t$

$$u_{nxi} = \sum_{j\in\{l|i_L(l)=i\}} w_{nxLj}\frac{1}{2}(u_{cxL(j)} + u_{cxR(j)}) + \sum_{j\in\{l|i_R(l)=i\}} w_{nxRj}\frac{1}{2}(u_{cxL(j)} + u_{cxR(j)}) \tag{6.46}$$

$$u_{nyi} = \sum_{j\in\{l|i_L(l)=i\}} w_{nyLj}\frac{1}{2}(u_{cyL(j)} + u_{cyR(j)}) + \sum_{j\in\{l|i_R(l)=i\}} w_{nyRj}\frac{1}{2}(u_{cyL(j)} + u_{cyR(j)}) \tag{6.47}$$

Algorithm 8 setlinktocornerweights: compute weights w_{nxLj}, w_{nxRj}, w_{nyLj} and w_{nyRj} in the nodal interpolation of Algorithm (7), Eqns. (6.46) and (6.47)

$$\chi_{xj} = \begin{cases} 2\max(10^{-6}, |\boldsymbol{n}_j \cdot \boldsymbol{e}_x|), & \text{jacomp} = 1 \\ 1, & \text{otherwise} \end{cases} \tag{6.48}$$

$$\chi_{yj} = \begin{cases} 2\max(10^{-6}, |\boldsymbol{n}_j \cdot \boldsymbol{e}_y|), & \text{jacomp} = 1 \\ 1, & \text{otherwise} \end{cases} \tag{6.49}$$

if $i_L(j)$ and $i_R(j)$ are not boundary nodes **then**

$$w_{nxLj} = \frac{\frac{1}{2}\alpha_{nLj}\Delta x_j w_{uj}\chi_{xj}}{\sum_{l\in\{m|i_L(m)=i_L(j)\}} \frac{1}{2}\alpha_{nLj}\Delta x_l w_{ul}\chi_{xj} + \sum_{j\in\{l|i_R(m)=i_L(j)\}} \frac{1}{2}\alpha_{nRl}\Delta x_l w_{ul}\chi_{xj}}$$

$$w_{nyLj} = \frac{\frac{1}{2}\alpha_{nLj}\Delta x_j w_{uj}\chi_{yj}}{\sum_{l\in\{m|i_L(m)=i_L(j)\}} \frac{1}{2}\alpha_{nLj}\Delta x_l w_{ul}\chi_{yj} + \sum_{l\in\{m|i_R(m)=i_L(j)\}} \frac{1}{2}\alpha_{nRl}\Delta x_l w_{ul}\chi_{yj}}$$

$$w_{nxRj} = \frac{\frac{1}{2}\alpha_{nRj}\Delta x_j w_{uj}\chi_{xj}}{\sum_{l\in\{m|i_L(m)=i_R(j)\}} \frac{1}{2}\alpha_{nLl}\Delta x_l w_{ul}\chi_{xj} + \sum_{l\in\{m|i_R(m)=i_R(j)\}} \frac{1}{2}\alpha_{nRl}\Delta x_l w_{ul}\chi_{xj}}$$

$$w_{nyRj} = \frac{\frac{1}{2}\alpha_{nRj}\Delta x_j w_{uj}\chi_{yj}}{\sum_{l\in\{m|i_L(m)=i_R(j)\}} \frac{1}{2}\alpha_{nLl}\Delta x_l w_{ul}\chi_{yj} + \sum_{l\in\{m|i_R(m)=i_R(j)\}} \frac{1}{2}\alpha_{nRl}\Delta x_l w_{ul}\chi_{yj}}$$

else
unmentioned
end if

Remark 6.1.13. For nodes *not* on the mesh boundary, it is unclear why the weights in Algorithm (7) for vector interpolation should differ from Eqns. (6.40) and (6.41) for scalar interpolation, which is the case if "jacomp" $= 1$ in Algorithm (8). The option "jacomp" $= 1$ may need to be reconsidered.

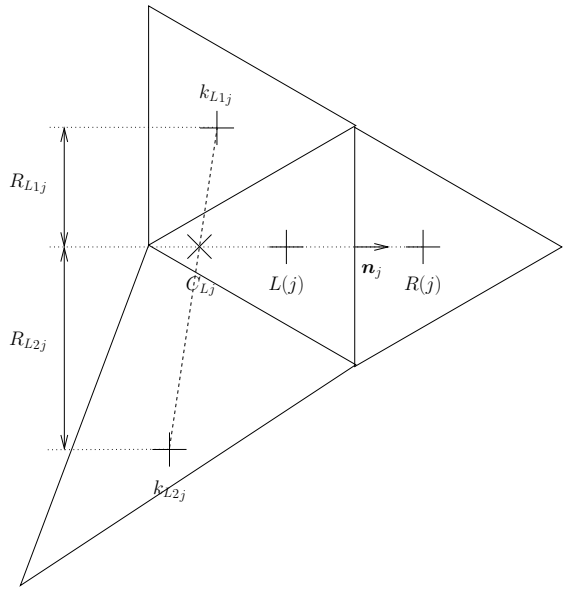


Figure 6.6: *Higher-order reconstruction of face-based velocity $\boldsymbol{u}_{uj}$, from the left*

The various variables used in the nodal interpolation have the following names in the D-Flow FM code:

$\boldsymbol{x}_{\zeta_k} \cdot \boldsymbol{e}_x$:	`xz(k)`,	$\boldsymbol{x}_{\zeta_k} \cdot \boldsymbol{e}_y$:	`yz(k)`,
$\boldsymbol{x}_{ni} \cdot \boldsymbol{e}_x$:	`xk(i)`,	$\boldsymbol{x}_{ni} \cdot \boldsymbol{e}_y$:	`yk(i)`,
u_{nx_i}:	`ucnx(i)`,	u_{ny_i}:	`ucny(i)`,
α_{nLj}:	`acn(2,j)`,	α_{nRj}:	`acn(1,j)`,
w_{nxLj}:	`wcnx4(j)`,	w_{nyLj}:	`wcny4(j)`,
w_{nxRj}:	`wcnx3(j)`,	w_{nyRj}:	`wcny3(j)`.

Higher-order reconstruction: limtypmom

A higher-order accurate discretization of advection may be achieved by higher-order accurate reconstruction of the face-based full velocity vectors $\boldsymbol{u}_u$ in Eqn. (6.17). A first-order approximation is already available from Algorithm (6), Eqn. (6.36). This section elaborates on the higher-order corrections added to $\boldsymbol{u}_u$.

The reconstruction at the faces is a *one-dimensional* reconstruction on a line through both neighboring cells $L(j)$ and $R(j)$. Besides the neighboring cell-centered values, a third value is sought on the line, which is interpolated from cells connected to the left-hand side neighboring cell $L(j)$ for reconstruction from the left, and cells connected to the right-hand side neighboring cell $R(j)$ for reconstruction from the right, respectively. We will refer the these third locations on the line as C_{Lj} and C_{Rj} respectively. For the reconstruction along the line a one-dimensional limiter is used with the purpose to obtain a TVD scheme. In D-Flow FM various limiters are available by means of the option `limtypmom`.

Remark 6.1.14. It is not immediately clear why a TVD limiter based on *interpolated* values would guarantee TVD properties of the primitive variables.

We will firstly consider the stencil for the reconstruction. Assume that we want to reconstruct at face j from the left, then the cells in the stencil are $\{R(j), L(j), k_{L1j}, k_{R2j}\}$. An example is shown in Fig. 6.6. If we let R_{L1j} measure the distance from the cell center k_{L1j} to the line through $L(j)$ and $R(j)$, and similarly for R_{L2j}, then the cells k_{L1j} and k_{L2j} are chosen according to the rules stated in Algorithm (9). These cells are the cells whose circumcenters

are closest to the line through $L(j)$ and $R(j)$.

The values in cells k_{L1j} and k_{R2j} are used to interpolate a value at C_{Lj}, which is located on the line trough the left and right cell centers $L(j)$ and $R(j)$. The higher-order reconstruction is then performed based on the values of cells C_{Lj}, $L(j)$ and $R(j)$ in a one-dimensional fashion.

A value, say Φ, at C_{Lj}, i.e. $\Phi_{C_{L_j}}$ (being one of the two cell-centered velocity vector components as we will see later), is interpolated as follows:

$$\Phi_{C_{L_j}} = s_{L1j}\Phi_{k_{L1_j}} + s_{L2j}\Phi_{k_{L2_j}}. \tag{6.50}$$

The weights are computed with Algorithm (9). Note that the exception for $R_{L_{1_j}} < 0.1\Delta x_j$

Algorithm 9 setupwslopes: determine the cells k_{L1} and k_{L2}, and compute weights s_{L1} and s_{L2} in Eqn. (6.50) for the higher-order reconstruction from the left at the faces, and similar for reconstruction from the right to obtain k_{R1}, k_{R2}, s_{R1} and s_{R2}

if reconstruction from the left **then**
determine the cells k_{L1j} and k_{R2j} according to the following rules

- cells k_{L1j} and k_{R2j} each share a face with cell $L(j)$
- the cell center is at the left-hand side from cell center $L(j)$, i.e. $(\boldsymbol{x}_{\zeta_k} \cdot \boldsymbol{n}_j < 0$ for $k \in \{k_{L1j}, k_{L2j}\}$
- cell center k_{L1j} is closer to the line through cell centers $L(j)$ and $R(j)$ than, or as close as, any other cell that obeys the two rules above
- cell center k_{L2j} is closer to the line through cell centers $L(j)$ and $R(j)$ than, or as close as, any other cell that is not k_{L1j} and obeys the first two rules above and results in a intersection point C_{Lj} that is sufficiently far from cell center $L(j)$, as expressed by $(\boldsymbol{x}_{C_{L_j}} - \boldsymbol{x}_{\zeta_{R(j)}}) \cdot (\boldsymbol{x}_{\zeta_{L(j)}} - \boldsymbol{x}_{\zeta_{R(j)}}) > 1.2$

if $R_{L_{1_j}} < 0.1\Delta x_j$ and the "advection type" of face between k_{L1j} and $L(j) \notin \{6, 8\}$ **then**
$s_{L1j} = 1$, $s_{L2j} = 0$, $\gamma_{Lj} = \frac{\Delta x_j}{\|\boldsymbol{x}_{\zeta_{L(j)}} - \boldsymbol{x}_{\zeta_{k_{L1_j}}}\|}$
else if cell k_{R2j} found and the "advection type" of faces between k_{L1j} and $L(j)$, and between k_{L2j} and $L(j) \notin \{6, 8\}$ **then**
$s_{L1j} = \frac{\|\boldsymbol{x}_{C_{L_j}} - \boldsymbol{x}_{\zeta_{k_{L2_j}}}\|}{\|\boldsymbol{x}_{\zeta_{k_{L1_j}}} - \boldsymbol{x}_{\zeta_{k_{L2_j}}}\|}$, $s_{L2j} = 1 - s_{L1j}$, $\gamma_{Lj} = \frac{\Delta x_j}{\|\boldsymbol{x}_{\zeta_{L(j)}} - \boldsymbol{x}_{C_{L_j}}\|}$
else
$s_{L1j} = 0$, $s_{L2j} = 0$, $\gamma_{Lj} = 0$ (no higher-order reconstruction)
end if
else
similar as reconstruction from the left by replacing L with R, vice versa, and taking the reversed orientation into account
end if

only adds *one* cell to the stencil for higher-order reconstruction, mimicking stencils on e.g. curvilinear meshes. Note also the exception for the (face-specified) advection types 6 and 8, not discussed further.

The interpolation of Eqn. (6.50) is applied to the Cartesian components of the velocity vector. In such a manner values at $u_{xC_{L_j}}$ and $u_{yC_{L_j}}$ are obtained. The reconstruction is then performed with Algorithm (10). Note that in Algorithm (10), Eqns. (6.53) and (6.54), γ_{Lj} ac-

Algorithm 10 sethigherorderadvectionvelocities: higher-order accurate reconstructions of face-based velocity vector $\boldsymbol{u}_u$ from cell-centered velocity vectors $\boldsymbol{u}_c$

interpolate velocity vectors on a line through cell centers $L(j)$ and $R(j)$, from the left and from the right:

$$\boldsymbol{u}_{C_{Lj}} = s_{L1j}\boldsymbol{u}_{ck_{L1_j}} + s_{L2j}\boldsymbol{u}_{ck_{L2_j}} \tag{6.51}$$

$$\boldsymbol{u}_{C_{Rj}} = s_{R1j}\boldsymbol{u}_{ck_{R1_j}} + s_{R2j}\boldsymbol{u}_{ck_{R2_j}} \tag{6.52}$$

if $q_{aj} > 0$ **then**

compute slope ratio in limiter, for each velocity component

$$r_x = \frac{u_{cR(j)_x} - u_{cL(j)_x}}{u_{cL(j)_x} - u_{C_{Lj}x}} \frac{1}{\gamma_{Lj}} \tag{6.53}$$

$$r_y = \frac{u_{cR(j)_y} - u_{cL(j)_y}}{u_{cL(j)_y} - u_{C_{Lj}y}} \frac{1}{\gamma_{Lj}} \tag{6.54}$$

apply limiter Ψ to each velocity component and reconstruct the velocity vector at the face

$$\boldsymbol{u}_{uj} = \boldsymbol{u}_{cL(j)} + \alpha_j \max(1 - \frac{\Delta t|u_j|}{\Delta x_j}, 0) \begin{pmatrix} \Psi(r_x) & 0 \\ 0 & \Psi(r_y) \end{pmatrix} (\boldsymbol{u}_{cR(j)} - \boldsymbol{u}_{cL(j)}) \tag{6.55}$$

else

reconstruction from the right similar as reconstruction from the left by replacing L with R, vice versa, α_j by $1 - \alpha_j$ and taking the reversed orientation into account

end if

counts for the non-uniform spacing along the line through cell centers $L(j)$ and $R(j)$. It is computed along with the stencil and weights in Algorithm (9) and similarly for γ_{Rj}.

In D-Flow FM various limiters (Ψ in Algorithm (10)) are available. They are set with the keyword `limtypmom`, see Table 6.1.

Table 6.1: *Various limiters available in D-Flow FM for the reconstruction of face-based velocities in momentum advection*

`limtypmom`	limiter	$\Psi(r)$
0	first-order upwind	0
$1, 5, 15$	minmod	$\max(\min(r, 1), 0)$
2	Van Leer	$\frac{r+\|r\|}{1+\|r\|}$
3	*incorrect* Koren	$r \max(\min(\min(2/r, \frac{1+2/r}{3}), 2), 0)$
$4, 14$	monotonized central	$\max(\min(\min(2r, \frac{1+r}{2}), 2), 0)$
11	*incorrect* minmod	$\frac{1}{r}\max(\min(r, 1), 0)$
12	*incorrect* Van Leer	$\frac{1}{r}\frac{r+\|r\|}{1+\|r\|}$
13	another *incorrect* Koren	$\max(\min(\min(2/r, \frac{1+2/r}{3}), 2), 0)$
20	Beam-Warming	r
21	Lax-Wendroff	1

Remark 6.1.15. In the D-Flow FM limiters 1 to 4 are formulated using the property $\Psi(r) = r\Psi(\frac{1}{r})$ for symmetric limiters. Since the Koren limiter is not symmetric, its implementation is incorrect. Limiters 11, 12 and 13 are also incorrect implementations.

Remark 6.1.16. Since the limiter functions are non-linear in general, and the velocity field is represented by face-normal components, the component-wise reconstruction is orientation dependent. Hence, the discretization is not invariant for a rotation of the coordinate frame. This may be circumvented by reconstructing face-normal and tangential components instead.

The translation of the various variables introduced here to the D-Flow FM code is as follows:

k_{L1_j}:	`klnup(1,j)`,	k_{R1_j}:	`klnup(4,j)`,
k_{L2_j}:	`klnup(2,j)`,	k_{R2_j}:	`klnup(5,j)`,
s_{L1_j}:	`slnup(1,j)`,	s_{R1_j}:	`slnup(4,j)`,
s_{L2_j}:	`slnup(2,j)`,	s_{R2_j}:	`slnup(5,j)`,
γ_{L_j}:	`slnup(3,j)`,	γ_{R_j}:	`slnup(6,j)`.

Momentum diffusion

The momentum diffusion term in Eqn. (6.2) is

$$\frac{1}{h}\nabla\cdot(\nu h(\nabla\boldsymbol{u}+\nabla\boldsymbol{u}^T)).$$

In D-Flow FM, this term is modified as

$$\frac{1}{h^p}\nabla\cdot(\nu h^p(\nabla\boldsymbol{u}+\nabla\boldsymbol{u}^T)),$$

where

$$p=\begin{cases}1, & \texttt{istresstype}=3,\\ 1, & \texttt{istresstype}=5,\\ 0, & \text{otherwise.}\end{cases} \tag{6.56}$$

Remark 6.1.17. It is unclear why, for $\texttt{istresstype}\neq 3\wedge\texttt{istresstype}\neq 5$, a modified, incorrect form of momentum diffusion, i.e. $p\neq 1$, is employed in D-Flow FM.

Obviously, the momentum diffusion term needs to be discretized at the faces and projected in face normal direction. The approach undertaken is similar to the discretization of momentum advection. First a cell-centered conservative discretization of $\nabla\cdot(\nu h(\nabla\boldsymbol{u}+\nabla\boldsymbol{u}^T))$ is formulated which is subsequently interpolated to the faces, projected in the face normal direction and divided by a water height h to bring it in non-conservative form.

If we call the cell-centered discretization $\boldsymbol{d}_k$, or more precisely

$$\nabla\cdot\left(\nu h^p(\nabla\boldsymbol{u}+\nabla\boldsymbol{u}^t)\right)\big|_{\Omega_k}\approx\boldsymbol{d}_k, \tag{6.57}$$

then the face-normal momentum diffusion at face j is interpolated from its neighboring cells $L(j)$ and $R(j)$ as

$$\nabla\cdot(\nu h^p(\nabla\boldsymbol{u}+\nabla\boldsymbol{u}^T))\big|_{\Gamma_j}\cdot\boldsymbol{n}_j\approx\alpha_j\boldsymbol{d}_{L(j)}\cdot\boldsymbol{n}_j+(1-\alpha_j)\boldsymbol{d}_{R(j)}\cdot\boldsymbol{n}_j, \tag{6.58}$$

where again α_j is the non-dimensional distance from the left cell center to the face, see Fig. 6.4.

The cell-averaged diffusion in cell k can be written in the usual manner as

$$\nabla\cdot\left(\nu h^p(\nabla\boldsymbol{u}+\nabla\boldsymbol{u}^t)\right)\big|_{\Omega_k}=\frac{1}{A(\Omega)}\int_{\partial\Omega_k}\nu h^p(\frac{\partial\boldsymbol{u}}{\partial n}+\nabla\boldsymbol{u}\cdot\boldsymbol{n})\,\mathrm{d}\Gamma, \tag{6.59}$$

where $A(\Omega_k) = b_{Ak}$ is the bed area of cell k. This expression is discretized as

$$\nabla \cdot \left(\nu h^p (\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^t)\right)\big|_{\Omega_k} \approx \boldsymbol{d}_k = \begin{cases} \frac{1}{b_{Ak}} \sum\limits_{j \in \mathcal{J}(k)} \boldsymbol{t}_{uj} w_{uj} 1_{j,k}, & p = 0, \\ \frac{1}{b_{Ak}} \sum\limits_{j \in \mathcal{J}(k)} \boldsymbol{t}_{uj} w_{uj} \min(h_{sL(j)}, h_{sR(j)}) 1_{j,k}, & p = 1, \quad \texttt{istresstype} = 3, \\ \frac{1}{b_{Ak}} \sum\limits_{j \in \mathcal{J}(k)} \boldsymbol{t}_{uj} A_{uj} 1_{j,k}, & p = 1, \quad \texttt{istresstype} = 5. \end{cases} \tag{6.60}$$

Note that $\boldsymbol{t}_j$ is the viscous stress at face j. By using $\boldsymbol{n} = (n_x, n_y)^t$, setting $\boldsymbol{s} = \boldsymbol{n}^\perp = (-n_y, n_x)^t$ and noting that

$$\nabla \boldsymbol{u} \cdot \boldsymbol{n} = \begin{pmatrix} n_x & -n_y \\ n_y & n_x \end{pmatrix} \begin{pmatrix} \boldsymbol{n} \cdot \dfrac{\partial \boldsymbol{u}}{\partial n} \\ \boldsymbol{n} \cdot \dfrac{\partial \boldsymbol{u}}{\partial s} \end{pmatrix} = \begin{pmatrix} {n_x}^2 & n_x n_y \\ n_x n_y & {n_y}^2 \end{pmatrix} \frac{\partial \boldsymbol{u}}{\partial n} + \begin{pmatrix} -n_x n_y & -{n_y}^2 \\ {n_x}^2 & n_x n_y \end{pmatrix} \frac{\partial \boldsymbol{u}}{\partial s},$$

the viscous stresses $\boldsymbol{t}_{uj}$ for $\texttt{istresstype} \neq 6$ are computed as

$$\boldsymbol{t}_{uj} = \nu_j \Bigg(\begin{pmatrix} 1 + {n_x}_j^2 & {n_x}_j {n_y}_j \\ {n_x}_j {n_y}_j & 1 + {n_y}_j^2 \end{pmatrix} \frac{\boldsymbol{u}_{cR(j)} - \boldsymbol{u}_{cL(j)}}{\Delta x_j} + \begin{pmatrix} -{n_x}_j {n_y}_j & -{n_y}_j^2 \\ {n_x}_j^2 & {n_x}_j {n_y}_j \end{pmatrix} \frac{\boldsymbol{u}_{ni_L(j)} - \boldsymbol{u}_{ni_R(j)}}{w_{uj}} \Bigg). \tag{6.61}$$

For $\texttt{istresstype} = 6$ the viscous stresses are completely expressed in normal and tangential components and essentially the same expression as Eqn. (6.61) is obtained.

Note that $\boldsymbol{u}_{ck}$ (here with $k \in \{L(j), R(j)\}$) and $\boldsymbol{u}_{ni}$ (here with $i \in \{i_L(j), i_R(j)\}$) are cell-centered and node-based velocity vectors, respectively. Their reconstruction from the face-normal velocity components and interpolation has been discussed in the foregoing sections, see Algorithms (6) and (7) respectively.

The contribution of the horizontal momentum diffusion term to the discrete momentum equation Eqn. (6.14) is finally obtained by bringing it in non-conservative form and interpolation at the faces

$$\mathcal{A}_{\text{e}j} = -\left[\frac{\alpha_j \boldsymbol{d}_{L(j)} \cdot \boldsymbol{n}_j}{H_{L_j}^p} + \frac{(1 - \alpha_j) \boldsymbol{d}_{R(j)} \cdot \boldsymbol{n}_j}{H_{R_j}^p} \right], \tag{6.62}$$

as performed by Algorithm (11). It shows that the choice for H_{Lj} and H_{Rj} depends on $\texttt{istresstype}$.

Remark 6.1.18. Momentum diffusion is discretized in a similar fashion as momentum advection, namely based on a cell-centered expression of the conservative formulation, interpolation to the faces and bringing it into a non-conservative form, i.e. dividing it by the water depth. Consequently, the discretizations of the terms $\frac{1}{H_{Lj}}$ and $\frac{1}{H_{Rj}}$, due to the non-conservative formulation, are expected to equal their counterparts in momentum advection. However, they do not, as can be seen by comparing Algorithm (11) with Algorithm (4).

Remark 6.1.19. In the discretization of the diffusive fluxes, the area of face j is approximated by $w_{uj} \min(h_{sL(j)}, h_{sR(j)})$ for $\texttt{istresstype}$ '3'. It is unclear why the actual cross-sectional area A_{uj} does not suffice. For the other $\texttt{istresstypes}$, see Remark 6.1.17.

Algorithm 11 setumod|momentum diffusion: compute momentum diffusion terms of the form $\boldsymbol{n}_j \cdot \left[-\frac{1}{h^p}\left(\nu h^p(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^t)\right)\right]_j \approx \mathcal{A}_{\mathrm{e}j}$

compute viscous stresses

$$\boldsymbol{t}_l = \nu_l \left[\begin{pmatrix} 1+n_{xl}^2 & n_{xl}n_{yl} \\ n_{xl}n_{yl} & 1+n_{yl}^2 \end{pmatrix} \frac{\boldsymbol{u}_{cR(l)} - \boldsymbol{u}_{cL(l)}}{\Delta x_l} + \begin{pmatrix} -n_{xl}n_{yl} & -n_{yl}^2 \\ n_{xl}^2 & n_{xl}n_{yl} \end{pmatrix} \frac{\boldsymbol{u}_{ni_L(l)} - \boldsymbol{u}_{ni_R(l)}}{w_{ul}} \right]$$

$$\mathcal{A}_{\mathrm{e}j} = -\left[\frac{\alpha_j}{b_{AL(j)} H_{Lj}} \sum_{l \in \mathcal{J}(L(j))} \nu_l \, A_l \, \boldsymbol{t}_l \cdot \boldsymbol{n}_j \, 1_{l,L(j)} + \frac{1-\alpha_j}{b_{AR(j)} H_{Rj}} \sum_{l \in \mathcal{J}(R(j))} \nu_l \, A_l \, \boldsymbol{t}_l \cdot \boldsymbol{n}_j \, 1_{l,R(j)} \right]$$

with A_l, H_{Lj}, H_{Rj} defined by:

`istresstype`	A_l	H_{Lj}	H_{Rj}
2, 4, 6	w_{ul}	1	1
3	$w_{ul} \min(h_{sL(l)}, h_{sR(l)})$	$\frac{1}{2}(h_{sL(j)} + h_{sR(j)})$	$\frac{1}{2}(h_{sL(j)} + h_{sR(j)})$
5	A_{ul}	$h_{sL(j)}$	$h_{sR(j)}$

Turbulence modelling: Smagorinsky, Elder

For `istressype` 2 and 3, the viscosity coefficient ν_j can be computed with Elder's formula or a Smagorinsky model. Note that the background viscosity is added, not mentioned here further for simplicity. In the first case, it is

$$\nu_j = E \; 0.0045 \; (h_{sL(j)} + h_{sR(j)}) \sqrt{u_j^2 + v_j^2}, \tag{6.63}$$

where E is the user-specified Elder coefficient. And in case of the Smagorinsky model

$$\nu_j = (C_S \sqrt{\Delta x_j w_{uj}})^2 \left| \sqrt{2\frac{\partial u_n}{\partial n}^2 + (\frac{\partial u_n}{\partial t} + \frac{\partial u_t}{\partial n})^2 + 2\frac{\partial u_t}{\partial t}^2} \right|_j, \tag{6.64}$$

where C_S is a user-specified Smagorinsky coefficient and the velocity derivatives at face j are approximated with finite differences similarly to Eqn. (6.61).

Limitation of viscosity coefficient

The explicit time integration of momentum diffusion is subject to a time-step limitation for numerical stability. We however maintain our time step and limit the eddy viscosity coefficient instead. We assume that it is sufficient to consider the model equation

$$\frac{\partial \boldsymbol{u}}{\partial t} = -\nabla \nu \cdot \nabla \boldsymbol{u}. \tag{6.65}$$

We also assume that it is sufficient to only consider a cell-based discretization, which for cell k would be

$$\frac{\boldsymbol{u}_{ck}^{n+1} - \boldsymbol{u}_{ck}^{n}}{\Delta t} = \frac{1}{V_k^n} \sum_{j \in \mathcal{J}(k)} \nu_j {A_u}_j^n \frac{\boldsymbol{u}_{R(j)}^n - \boldsymbol{u}_{L(j)}^n}{\Delta x_j} 1_{j,k}. \tag{6.66}$$

Remark 6.1.20. If we disregard the differences due the interpolation to the faces, and the missing terms $\nabla \cdot (h\nu \nabla \boldsymbol{u}^T)$, the discretization of the model equation only conforms to the form of the momentum diffusion term if `istresstype`=5, and if $V_k = b_{Ak} h_{sk}$ (no non-linear iterations, see Algorithm (18)), as can be seen by comparing with Algorithm (11).

Eqn. (6.66) can be rewritten as

$$\boldsymbol{u}_{ck}^{n+1} = \left(1 - \frac{\Delta t}{V_k^n} \sum_{j \in \mathcal{J}(k)} \frac{\nu_j {A_{u_j}}^n}{\Delta x_j}\right) \boldsymbol{u}_{ck}^n + \frac{\Delta t}{V_k^n} \sum_{j \in \mathcal{J}(k)} \frac{\nu_j {A_{u_j}}^n}{\Delta x_j} \boldsymbol{u}_{O(k,j)}^n, \tag{6.67}$$

where $O(k, j) =$ is the cell that shares face j with cell k, i.e.

$$O(k, j) = L(j) + R(j) - k. \tag{6.68}$$

We require that

$$0 \leq \frac{\Delta t}{V_k^n} \sum_{j \in \mathcal{J}(k)} \frac{\nu_j {A_{u_j}}^n}{\Delta x_j} \leq 1, \tag{6.69}$$

which is satisfied if we limit the viscosity coefficient by

$$\nu_j \leq \frac{1}{N} \frac{\Delta x_j}{{A_{u_j}}^n \Delta t} \min(V_{L(j)}^n, V_{R(j)}^n), \tag{6.70}$$

where N is the maximum number of faces in a cell. It is set to $N = 5$, although cells with more than five faces may occasionally be encountered.

Boundary stresses: irov

The viscous stress in Eqn. (6.60) at the *closed boundaries* need special attention, where three conditions that may be applied:

`irov=0`: full slip,
`irov=1`: partial slip,
`irov=2`: no slip.

The boundary conditions are further explained in Section 6.3.7.

Bed friction

Coriolis forces

6.2 Temporal discretization

The spatial discretization is, as explained in Section 6.1, performed in a staggered manner, i.e. velocity normal components u_j are defined at the cell faces j, with face normal vector $\boldsymbol{n}_j$, and the water levels ζ_k at cell centers k. If advection and diffusion are spatially discretized as in Eqn. (6.13)

$$\boldsymbol{n}_j \cdot \left[\frac{1}{h}(\nabla \cdot (h\boldsymbol{u}\boldsymbol{u}) - \boldsymbol{u}\nabla \cdot (h\boldsymbol{u})) - \nabla \cdot (\nu(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T))\right]_j \approx \mathcal{A}_{\mathrm{i}j} u_j + \mathcal{A}_{\mathrm{e}j},$$

then the temporal discretization of Eqn. (6.2) is

$$\left(\frac{1}{\Delta t}+\mathcal{A}_{\mathrm{i}j}^{n}+\frac{g\|\hat{\boldsymbol{u}}_j\|}{C^2h}\right)u_j^{n+1}=\frac{1}{\Delta t}u_j^n-\frac{g\theta_j}{\Delta x_j}\left(\zeta_{R(j)}^{n+1}-\zeta_{L(j)}^{n+1}\right)-\mathcal{A}_{\mathrm{e}j}{}^{n}-\frac{g(1-\theta_j)}{\Delta x_j}\left(\zeta_{R(j)}^{n}-\zeta_{L(j)}^{n}\right), \tag{6.71}$$

where superscript n denotes the time level, $\hat{\boldsymbol{u}}_j$ is obtained by substituting $\hat{\boldsymbol{u}}_j=(\hat{u}_j,\boldsymbol{u}_j^n\cdot\boldsymbol{n}_j^{\perp})^T$, $u_j^{n+1}=\hat{u}_j,\theta_j=0$ in Eqn. (6.71) and solving for $\hat{u}_j$, and $\Delta x_j=\|\boldsymbol{x}_{R(j)}-\boldsymbol{x}_{L(j)}\|$ measures the distance between the two water-level points of cells $L(j)$ and $R(j)$ of face j. Note that we have assumed that the face normal $\boldsymbol{n}_j$ is in the direction from cell $L(j)$ to $R(j)$.

The velocity update of Eqn. (6.71) is summarized as

$$u_j^{n+1}=-f_{u_j}{}^{n}(\zeta_{R(j)}^{n+1}-\zeta_{L(j)}^{n+1})+r_{u_j}{}^{n}, \tag{6.72}$$

where $f_{u_j}{}^{n}$ and $r_{u_j}{}^{n}$ are determined iteratively by Algorithm (12). Here MAXITER $=4$, $\epsilon=$

Algorithm 12 furu: compute $f_{u_j}{}^{n}$ and $r_{u_j}{}^{n}$ in $u_j^{n+1}=-f_{u_j}{}^{n}(\zeta_{R(j)}^{n+1}-\zeta_{L(j)}^{n+1})+r_{u_j}{}^{n}$

$$\begin{aligned}\hat{u}_j^{(0)}&=u_j{}^{n}\\ i&=0\end{aligned}$$

while $\left(i<\text{MAXITER}\wedge|\hat{u}_j^{(i)}-\hat{u}_j^{(i-1)}|>\epsilon\right)\vee i=0$ **do**

$$\begin{aligned}i&=i+1\\ f_{rj}&=\frac{g}{C^2h}\sqrt{(\hat{u}_j^{(i-1)})^2+(v_j{}^{n})^2}\\ B_u&=\frac{1}{\Delta t}+\mathcal{A}_{\mathrm{i}j}+f_{rj}\\ f_{u_j}{}^{n}&=\frac{1}{B_u}\frac{g\theta_j}{\Delta x_j}\\ r_{u_j}{}^{n}&=\frac{1}{B_u}\left(\frac{1}{\Delta t}u_j^n-\mathcal{A}_{\mathrm{e}j}-\frac{g(1-\theta_j)}{\Delta x_j}\left(\zeta_{R(j)}^{n}-\zeta_{L(j)}^{n}\right)\right)\\ \hat{u}_j^{(i)}&=-f_{u_j}{}^{n}(\zeta_{R(j)}^{n}-\zeta_{L(j)}^{n})+r_{u_j}{}^{n}\end{aligned}$$

end while

10^{-2} is a tolerance and v_j is the tangential velocity component at face j whose computation is discussed on a different occasion.

The continuity equation is discretized as

$$\frac{V_k^{n+1}-V_k^n}{\Delta t}=-\sum_{j\in\mathcal{J}(k)}A_{u_j}{}^{n}\left(\theta_j u_j{}^{n+1}+(1-\theta_j)u_j{}^{n}\right)1_{j,k}, \tag{6.73}$$

where $\mathcal{J}(k)$ is the set of faces that bound cell k and $1_{j,k}$ accounts for the orientation of face

j with respect to cell k, i.e.

$$1_{j,k} = \begin{cases} 1, & L(j) = k \quad (\boldsymbol{n}_j \text{ is outward normal of cell } k), \\ -1, & R(j) = k \quad (\boldsymbol{n}_j \text{ is inward normal of cell } k). \end{cases} \tag{6.74}$$

Furthermore, V_k^{n+1} is the volume of the water column at cell k and A_{uj} approximates the flow area of face j, i.e.

$$A_{uj} = h_{uj} w_j, \tag{6.75}$$

with h_{uj} the water level at face j (details not discussed here) and w_j the width of face j.

Substitution of Eqn. (6.72) in Eqn. (6.73) yields the following system for the water column volume at the next time instant:

$$\frac{V_k^{n+1} - V_k^n}{\Delta t} + \sum_{j \in \mathcal{J}(k)} {A_{uj}}^n \theta_j {f_{uj}}^n \, \zeta_k^{n+1} - \sum_{j \in \mathcal{J}(k)} {A_{uj}}^n \theta_j {f_{uj}}^n \, \zeta_{O(k,j)}^{n+1} = \\ - \sum_{j \in \mathcal{J}(k)} {A_{uj}}^n \left[(1-\theta_j) u_j^n + \theta_j {r_{uj}}^n \right] 1_{j,k}, \tag{6.76}$$

where $O(k,j) =$ is the cell that shares face j with cell k, i.e.

$$O(k,j) = L(j) + R(j) - k. \tag{6.77}$$

Remark 6.2.1. The flow area of face j, A_{uj}, always appears explicitly in the continuity equation, Eqn. (6.73).

The water-level equation, Eqn. (6.76), is summarized as

$$\frac{V_k^{n+1} - V_k^n}{\Delta t} + B_k^n \, \zeta_k^{n+1} + \sum_{j \in \mathcal{J}(k)} C_j^n \, \zeta_{O(k,j)}^{n+1} = d_k^n, \tag{6.78}$$

where the coefficients B_k^n (diagonal entries), C_j^n (off-diagonal entries) and d_k^n (right-hand side) are computed by Algorithm (13).

Algorithm 13 s1ini: compute the matrix entries and right-hand side in the water-level equation $\frac{V_k^{n+1} - V_k^n}{\Delta t} + B_k^n \, \zeta_k^{n+1} + \sum_{j \in \mathcal{J}(k)} C_j^n \, \zeta_{O(k,j)}^{n+1} = d_k^n$, Eqn. (6.78)

$$\begin{aligned} C_j^n &= - {A_{uj}}^n \theta_j {f_{uj}}^n \\ B_k^n &= - \sum_{j \in \mathcal{J}(k)} C_j^n \\ d_k^n &= - \sum_{j \in \mathcal{J}(k)} {A_{uj}}^n \left[(1-\theta_j) u_j^n + \theta_j {r_{uj}}^n \right] 1_{j,k} \end{aligned}$$

The continuity equation is only applied at water-level cells that are or may become (partially) wet at the next time level. These cells are marked with $k_{fs}(k) = 1$ and is based on the water height of the surrounding faces, see Algorithm (14).

Algorithm 14 setkfs: mark the water-level cells that are or may become (partially) wet with $k_{fs}(k) = 1$

$$k_{fs}(k) = \begin{cases} 0, & h_{uj} = 0 \ \forall j \in \mathcal{J}(k), \\ 1, & \text{otherwise.} \end{cases}$$

Algorithm 15 pack_matrix: determine the set $\mathcal{K}$ of water-level cells for which the continuity equation is solved

mark wet/dry cells, Algorithm (14)
$\mathcal{K} = \{k : k_{fs}(k) = 1\}$

The resulting set of water-level cells is called $\mathcal{K}$, see Algorithm (15). The continuity equation is only applied at cells k for $k \in \mathcal{K}$.

In order to solve Eqn. (6.78), we need to express the V_k^{n+1} volume of the water column at cell k at time level $n+1$ in terms of the water level ζ^{n+1}. Since this relation is non-linear in general, Eqn. (6.78) is solved iteratively by means of Newton iterations. We firstly linearize the expression for the volume of the water column and obtain for some iteration i

$$V_k^{n+1(i+1)} = V_k^{n+1(i)} + A_k^{n+1(i)} \left(\zeta_k^{n+1(i+1)} - \zeta_k^{n+1(i)} \right), \tag{6.79}$$

where $A_k^{n+1(i)}$ is the wet bed area of cell k at (iterative) time level $n+1(i)$. Substitution in Eqn. (6.78) yields

$$\left(\frac{1}{\Delta t} A_k^{n+1(i)} + B_k^n \right) \zeta_k^{n+1(i+1)} + \sum_{j \in \mathcal{J}(k)} C_j^n \, \zeta_{O(k,j)}^{n+1(i+1)} = d_k^n - \frac{1}{\Delta t} \left(V_k^{n+1(i)} - V_k^n - A_k^{n+1(i)} \zeta_k^{n+1(i)} \right), \tag{6.80}$$

which is summarized as

$$B_{rk}^{\ n+1(i)} \, \zeta_k^{n+1(i+1)} + \sum_{j \in \mathcal{J}(k)} C_{rj}^{\ n} \, \zeta_{O(k,j)}^{n+1(i+1)} = d_{rk}^{\ n+1(i)}, \tag{6.81}$$

where the coefficients $B_{rk}^{\ n}$ (diagonal entries), $C_{rj}^{\ n}$ (off-diagonal entries) and $d_{rk}^{\ n}$ (right-hand side) are computed by Algorithm (16). Note that we did not describe the water-level boundary

Algorithm 16 s1nod: compute the matrix entries and right-hand side in the water-level equation $B_{rk}^{\ n+1(i)} \, \zeta_k^{n+1(i+1)} + \sum_{j \in \mathcal{J}(k)} C_{rj}^{\ n} \, \zeta_{O(k,j)}^{n+1(i+1)} = d_{rk}^{\ n+1(i)}$, Eqn. (6.81)

$$\begin{aligned} B_{rk}^{\ n+1(i)} &= B_k^n + \frac{1}{\Delta t} A_k^{n+1(i)} \\ C_{rj}^{\ n} &= C_j^n \\ d_{rk}^{\ n+1(i)} &= d_k^n - \frac{1}{\Delta t} \left(V_k^{n+1(i)} - V_k^n - A_k^{n+1(i)} \zeta_k^{n+1(i)} \right). \end{aligned}$$

conditions in Algorithm (16).

The unknown water-levels $k \in \mathcal{K}$ in Eqn. (6.81) are solved with a Krylov solver as will be explained in Section 6.2.1.

During the iterative process, the water level $\zeta_k^{n+1(i+1)}$ may have dropped below the bed level bl_k resulting in a negative water height. In these cases the time step is repeated with either a reduced time step with factor $f = 0.7$ (type 1), or the water-level cell k eliminated from the system by setting the water levels of its bounding faces to zero (type 2, default), see Algorithm (17).

Algorithm 17 poshcheck: check positivity of water height

if $\zeta_k^{n+1(i+1)} < bl_k$ **then**
  if type 1 **then**
    $\Delta t = f\Delta t$,
    repeat time-step
  else if type 2 (default) **then**
    $h_{u_j}^{\ n} = 0, \quad j \in \mathcal{J}(k)$.
    repeat time-step
  end if
end if

Having computed a new iterate of the water level, the water column volume $V_k^{n+1(i+1)}$ and wet bed area $A_k^{n+1(i+1)}$ of cell k are computed with Algorithm (18). Note that if no non-linear iterations are performed, the wet bed area is set to equal the cell bed area b_{Ak}.

Algorithm 18 volsur: compute water-column volume $V_k^{n+1(i+1)}$ and wet bed area $A_k^{n+1(i+1)}$

if no non-linear iterations **then**

$$V_k^{n+1(i+1)} = b_{Ak} \max(\zeta_k^{n+1(i+1)} - bl_k, 0)$$
$$A_k^{n+1(i+1)} = b_{Ak}$$

else
  compute actual wet bed area and water column volume of cell k based on a constant water level in a cell and linearly varying bed levels at the faces
  not elaborated further
end if

The time-step is finalized by employing Eqn. (6.72), see Algorithm (19). Two discharges are computed at the next time level, namely

$$q_j^{n+1} = A_{u_j}^{\ n} \left(\theta_j u_j^{n+1} + (1-\theta_j) u_j^n\right), \tag{6.82}$$
$$q_{a_j}^{\ n+1} = A_{u_j}^{\ n} u_j^{n+1}. \tag{6.83}$$

Discharge q_j^{n+1} satisfies the continuity equation Eqn. (6.73)

$$\frac{V_k^{n+1} - V_k^n}{\Delta t} = -\sum_{j \in \mathcal{J}(k)} q_j^{n+1} 1_{j,k}, \tag{6.84}$$

and appears for example in the discretization of advection in Eqn. (6.2). The use of $q_{a_j}^{\ n+1}$ is not discussed here, but it is important to note that it does not satisfy the continuity equation.

Algorithm 19 u1q1: update velocity u_j^{n+1} and discharges q_j^{n+1} and ${q_a}_j^{n+1}$

if ${h_u}_j^n > 0$ **then**

$$u_j^{n+1} = -\,{f_u}_j^n(\zeta_{R(j)}^{n+1} - \zeta_{L(j)}^{n+1}) + {r_u}_j^n \tag{6.85}$$
$$q_j^{n+1} = {A_u}_j^n\left(\theta_j u_j^{n+1} + (1-\theta_j)u_j^n\right) \tag{6.86}$$
$${q_a}_j^{n+1} = {A_u}_j^n u^{n+1} \tag{6.87}$$

else

$$u_j^{n+1} = 0 \tag{6.88}$$
$$q_j^{n+1} = 0 \tag{6.89}$$
$${q_a}_j^{n+1} = 0 \tag{6.90}$$

end if

The time-step is summed up in Algorithm (20).

6.2.1 Solving the water-level equation

The unknown water-levels $k \in \mathcal{K}$ in Eqn. (6.81) are solved with a Krylov solver, Algorithm (21). However, prior to solving the system, a Minimum Degree algorithm is applied to reduce the system size. The somewhat misleading terms Gauss elimination and substitution in Algorithm (21) are due to the minimum degree algorithm. The permutation order is only computed during the initialization of the computations. It will not be discussed further.

The (reduced) water-level equation to be solved has the form of

$$A\boldsymbol{s} = \boldsymbol{d}, \tag{6.91}$$

where according to Eqn. (6.81)

$$A\boldsymbol{s} = \begin{pmatrix} B_{r1}\ \zeta_1 + \sum_{j\in\mathcal{J}(1)} C_{rj}\ \zeta_{O(1,j)} \\ B_{r2}\ \zeta_2 + \sum_{j\in\mathcal{J}(2)} C_{rj}\ \zeta_{O(2,j)} \\ \vdots \end{pmatrix} \tag{6.92}$$

and $\boldsymbol{d} = (d_{r1}, d_{r2}, \dots)^T$. Note that we have omitted the superscripts for the sake of brevity. Note also that for simplicity, we assumed all unknowns $\zeta_1, \zeta_2, \dots$ appear in the solution vector, although due to the minimum degree algorithm and possible dry cells, they do not.

The system is solved by a preconditioned Conjugate Gradient method as shown in Algorithm (22). The preconditioner P can either be diagonal scaling, or an incomplete Cholesky decomposition.

Remark 6.2.2. The iterations in Algorithm (22) could be stopped after the computation of the absolute error. However, we want the possibility to base our stopping criterion on the preconditioned residual $\|\boldsymbol{z}_r^{(i)}\|_\infty$. For now, we will base our stopping criterion only on the residual $\boldsymbol{r}^{(i)}$.

Algorithm 20 step_reduce: perform a time step

while first iteration or repeat time-step (type 1) **do**
$t^{n+1} = t^n + \Delta t$
compute ${f_u}_j^n$ and ${r_u}_j^n$ with Algorithm (12)
while first iteration or repeat time-step (type 2) **do**
compute the matrix entries B_k^n, C_j^n and right-hand side d_k^n in the water-level equation with Algorithm (13)
determine the set of water-levels that need to be solved, Algorithm (15)
$i = 0$
$\zeta_k^{n+1(0)} = \zeta_k^n$
while $\left(\max_k \left| \zeta_k^{n+1(i)} - \zeta_k^{n+1(i-1)} \right| > \epsilon \ \wedge \$ not repeat time-step$\right) \ \vee \ i = 0$ **do**
$i = i + 1$
compute the matrix entries B_{rk}^n, ${C_r}_j^n$ and right-hand side d_{rk}^n in the water-level equation with Algorithm (16)
solve the unknown water-levels and obtain $\zeta_k^{n+1(i+1)}$, Algorithm (21)
check positivity of water height with Algorithm (17) and repeat time-step if necessary with modified Δt (type 1) or ${h_u}_j^n$ (type 2, default)
if not repeat time-step **then**
compute water-column volume $V_k^{n+1(i+1)}$ and wet bed area $A_k^{n+1(i+1)}$ with Algorithm (18)
end if
end while
end while
end while
$\zeta_k^{n+1} = \zeta_k^{n+1(i+1)}$
compute velocities u_j^{n+1} and discharges q_j^{n+1} and ${q_a}_j^{n+1}$ are defined at the next time level, Algorithm (19)

Algorithm 21 solve_matrix: solve the unknown water-levels in Eqn. (6.81)

perform Gauss elimination to reduce the number of unknowns in the system
solve system with Algorithm (22)
perform the Gauss substitution and obtain $\zeta_k^{n+1(i+1)}, \quad k \in \mathcal{K}$
set $\zeta_k^{n+1(i+1)} = \zeta_k^{n+1(i)}, \quad k \notin \mathcal{K}$

Algorithm 22 conjugategradient: solve water-level equation with a preconditioned Conjugate Gradient method

compute preconditioner P
compute initial residual $\boldsymbol{r}^{(0)} = \boldsymbol{d} - A\boldsymbol{s}^{(0)}$
compute maximum error $\epsilon = \|\boldsymbol{r}^{(0)}\|_\infty$
apply preconditioner $P\boldsymbol{z}_r^{(0)} = \boldsymbol{r}^{(0)}$
set $\boldsymbol{p}^{(0)} = \boldsymbol{z}_r^{(0)}$
compute inner product $(\boldsymbol{r}^{(0)}, \boldsymbol{z}_r^{(0)})$
$i = 0$
while $\epsilon >$ tol **do**
  compute $A\boldsymbol{p}^{(i)}$
  compute $(\boldsymbol{p}^{(i)}, A\boldsymbol{p}^{(i)})$
  $\alpha^{(i)} = \frac{(\boldsymbol{r}^{(i)}, \boldsymbol{z}_r^{(i)})}{(\boldsymbol{p}^{(i)}, A\boldsymbol{p}^{(i)})}$
  $\boldsymbol{s}^{(i+1)} = \boldsymbol{s}^{(i)} + \alpha^{(i)}\boldsymbol{p}^{(i)}$
  $\boldsymbol{r}^{(i+1)} = \boldsymbol{r}^{(i)} - \alpha^{(i)}A\boldsymbol{p}^{(i)}$
  compute maximum error $\epsilon = \|\boldsymbol{r}^{(i+1)}\|_\infty$
  apply preconditioner $P\boldsymbol{z}_r^{(i+1)} = \boldsymbol{r}^{(i+1)}$
  if $\epsilon >$ tol **then**
    compute $(\boldsymbol{r}^{(i+1)}, \boldsymbol{z}_r^{(i+1)})$
    $\beta^{(i+1)} = \frac{(\boldsymbol{r}^{(i+1)}, \boldsymbol{z}_r^{(i+1)})}{(\boldsymbol{r}^{(i)}, \boldsymbol{z}_r^{(i)})}$
    $\boldsymbol{p}^{(i+1)} = \boldsymbol{z}_r^{(i+1)} + \beta^{(i+1)}\boldsymbol{p}^{(i)}$
    $i = i + 1$
  end if
end while

6.3 Boundary Conditions

We can identify three type boundary conditions in D-Flow FM. These are:

- boundary conditions that complement the governing equations, Eqns. (6.1) and (6.2),
- supplementary boundary conditions that impose additional constraints at the boundaries,
- boundary conditions for constituents, such as salinity.

We will not discuss the last category. The following boundary conditions in the first category may be imposed:

0 default: full-slip,
1 "waterlevel",
2 "Neumann",
3 "velocity",
4 "discharge",
5 "Riemann",
6 "outflow"
7 "Qh",

where, except for the default full-slip condition, we have adopted the terminology and numbering of D-Flow FM. Additionally, the following boundary conditions in the second category may be imposed:

8 "tangentialvelocity",
9 "ucxucyadvectionvelocity",
10 "normalvelocity",

were again we have used D-Flow FM terminology, but extended the numbering for our convenience. We will use the numbering for the identification of the (parts of) the boundary, at which these conditions are implied. i.e. Γ_1 is the part of the boundary with water-level boundary conditions, et cetera. Since the boundary conditions 8 to 10 are supplemental, they may be combined with conditions 1 to 9.

Disregarding the effects of atmospheric pressure and time relaxation (discussed later), the

boundary conditions may be summarized as:

$$\boldsymbol{u}\cdot\boldsymbol{n} = 0, \qquad \boldsymbol{x}\in\Gamma_0, \quad \text{default}, \tag{6.93}$$
$$\zeta = \zeta_b, \qquad \boldsymbol{x}\in\Gamma_1, \quad \text{"waterlevel"}, \tag{6.94}$$
$$\frac{\partial\zeta}{\partial n} = s_b, \qquad \boldsymbol{x}\in\Gamma_2, \quad \text{"Neumann"}, \tag{6.95}$$
$$\boldsymbol{u}\cdot\boldsymbol{n} = u_b, \qquad \boldsymbol{x}\in\Gamma_3, \quad \text{"velocity"}, \tag{6.96}$$
$$\int_{\Gamma_4} h\boldsymbol{u}\cdot\boldsymbol{n}\,\mathrm{d}\Gamma = Q_b, \qquad \text{"discharge"}, \tag{6.97}$$
$$\zeta + \sqrt{\frac{h}{g}}\boldsymbol{u}\cdot\boldsymbol{n} = 2\zeta_b - \zeta^0, \qquad \boldsymbol{x}\in\Gamma_5, \quad \text{"Riemann"}, \tag{6.98}$$
$$\frac{\partial\zeta}{\partial n} = 0, \boldsymbol{u}\cdot\boldsymbol{n} > 0, \qquad \boldsymbol{x}\in\Gamma_6, \quad \text{"outflow"}, \tag{6.99}$$
$$\frac{\partial\zeta}{\partial t} - \sqrt{gh}(\frac{\partial\zeta}{\partial n} + s_b) = 0, \boldsymbol{u}\cdot\boldsymbol{n} \le 0, \qquad \boldsymbol{x}\in\Gamma_6, \quad \text{"outflow"}, \tag{6.100}$$
$$\zeta = h_b(\int_{\Gamma_7} h\boldsymbol{u}\cdot\boldsymbol{n}\,\mathrm{d}\Gamma), \qquad \boldsymbol{x}\in\Gamma_7, \quad \text{"Qh"}, \tag{6.101}$$

and

$$\boldsymbol{u}\cdot\boldsymbol{t} = v_b, \qquad \boldsymbol{x}\in\Gamma_8, \quad \text{"tangentialvelocity"}, \tag{6.102}$$
$$\boldsymbol{u} = \boldsymbol{u}_b, \qquad \boldsymbol{x}\in\Gamma_9, \quad \text{"ucxucyadvectionvelocity"}, \tag{6.103}$$
$$\boldsymbol{u} = u_b\boldsymbol{n}, \qquad \boldsymbol{x}\in\Gamma_{10}, \quad \text{"normalvelocity"}, \tag{6.104}$$

where ζ_b, s_b, u_b, v_b, $\boldsymbol{u}_b$, Q_b and h_b are user prescribed at the boundary where appropriate, $\boldsymbol{n}$ is the inward-positive normal vector, $\boldsymbol{t}$ is a unit tangential vector and ζ^0 is the initial water level.

Remark 6.3.1. The condition $\boldsymbol{u}\cdot\boldsymbol{n} > 0$ in Eqn. (6.99) is satisfied at *inflow* only.

Remark 6.3.2. At the "Qh" boundary a $Q-\zeta$ condition is imposed.

6.3.1 Fictitious boundary "cells": izbndpos

We firstly introduce some notation. $\mathcal{B}_0$ is the set of faces that are at the full-slip boundary, $\mathcal{B}_1$ is the set of faces at the "waterlevel" boundary Γ_1 and so on.

There is no administration in D-Flow FM for $\mathcal{B}_0$. The default boundary conditions are satisfied by effectively setting the face-normal velocity component to zero, i.e.

$$u_j = 0, \quad j\in\mathcal{B}_0. \tag{6.105}$$

The non-default boundary conditions are imposed by using fictitious boundary cells, see Fig. 6.7. Note that the term "cell" is ambiguous as it is only defined by means of its circumcenter. For boundary conditions of the first category (1 to 7), we discriminate between boundaries where, roughly speaking, the water-level is imposed (1, 2, 5 and 7) and where velocities are imposed (6 and 7), i.e.

$$\Gamma_\zeta = \Gamma_1\cup\Gamma_2\cup\Gamma_5\cup\Gamma_6\cup\Gamma_7, \tag{6.106}$$
$$\Gamma_u = \Gamma_3\cup\Gamma_4, \tag{6.107}$$
$$\Gamma = \Gamma_\zeta\cup\Gamma_u \tag{6.108}$$

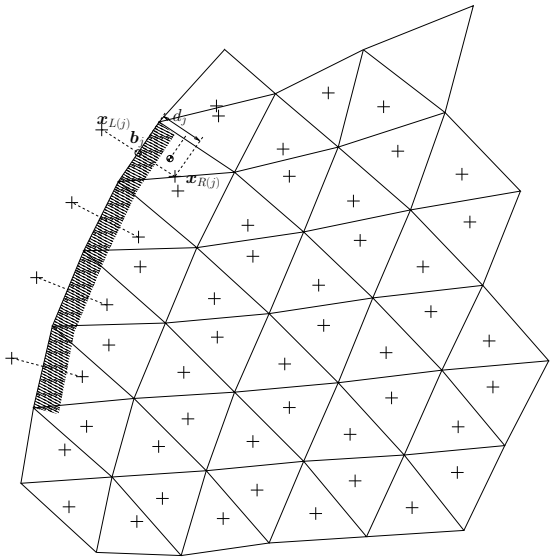


Figure 6.7: *Fictitious boundary "cells" near the shaded boundary; $\boldsymbol{x}_{Lj}$ is the fictitious "cell" center near boundary face j; $\boldsymbol{x}_{R(j)}$ is the inner-cell center; $\boldsymbol{b}_j$ is the point on face j that is nearest to the inner-cell center*

and similarly for the sets $\mathcal{B}_\zeta$, $\mathcal{B}_u$ and $\mathcal{B}$. The second category of boundaries are supplemental and are a subset of the first. Hence, for the definition of the fictitious boundary "cell" centers we only need to consider water-level and velocity boundaries, Γ_ζ and Γ_u respectively.

Let d_j measure the shortest distance from the cell circumcenter to the boundary face, see Fig. 6.7, and let $\boldsymbol{b}_j$ be the corresponding nearest point on the boundary face. Then the fictitious "cell" centers are computed with Algorithm (23). Note that $\boldsymbol{x}_n$ are mesh node coordinates and remember that the face normal $\boldsymbol{n}_j$ is inward positive.

Algorithm 23 addexternalboundarypoints: compute centers of fictitious boundary "cells"

$$\boldsymbol{x}_{L(j)} = \begin{cases} \boldsymbol{b}_j - \max(d_j, \frac{1}{2}\sqrt{b_{AR(j)}})\boldsymbol{n}_j, & j \in \mathcal{B}_\zeta \wedge \text{izbndpos} = 0 \\ \frac{1}{2}(\boldsymbol{x}_{ni_L(j)} + \boldsymbol{x}_{ni_R(j)}), & j \in \mathcal{B}_\zeta \wedge \text{izbndpos} = 1, \\ \boldsymbol{b}_j - \max(d_j, \frac{1}{2}\sqrt{b_{AR(j)}})\boldsymbol{n}_j, & j \in \mathcal{B}_u. \end{cases} \tag{6.109}$$

Remark 6.3.3. Option `izbndpos` $= 2$ is not documented here.

Algorithm (23) shows that the fictitious cell centers are *on* the boundary for `izbndpos=1` and at a distance d_j (or $\frac{1}{2}\sqrt{b_{AR(j)}}$) from the boundary otherwise.

Besides a center, the fictitious boundary "cells" also have a bed area b_A and bed level bl defined as

$$\left.\begin{array}{rcl} b_{AL(j)} & = & b_{AL(j)}, \\ bl_{L(j)} & = & bl_{R(j)}. \end{array}\right\} \quad j \in \mathcal{B}. \tag{6.110}$$

6.3.2 Discretization of the boundary conditions

The boundary conditions are accounted for by modification of the discretization near the boundaries. Assume that we are at time-level n and advance to time-level $n+1$, then the discretization of Eqns. (6.93) to (6.101) is:

$$u_j^{n+1} = 0, \qquad j \in \mathcal{B}_0\text{,default,} \tag{6.111}$$

$$\zeta_{L(j)}^{n+1} = \zeta_b(\boldsymbol{b}_j, \hat{t}^{n+1}), \qquad j \in \mathcal{B}_1\text{,"waterlevel",} \tag{6.112}$$

$$\frac{\zeta_{R(j)}^{n+1} - \zeta_{L(j)}^{n+1}}{\Delta x_j} = s_b(\boldsymbol{b}_j, \hat{t}^{n+1}), \qquad j \in \mathcal{B}_2\text{,"Neumann",} \tag{6.113}$$

$$u_j^{n+1} = u_b(\boldsymbol{b}_j, \hat{t}^{n+1}), \qquad j \in \mathcal{B}_3\text{,"velocity",} \tag{6.114}$$

$$u_j^{n+1} = \frac{Q_b(\hat{t}^{n+1})({h_{u_j}}^n)^{2/3}}{\sum\limits_{l\in\mathcal{B}_4} {A_{u_l}}^n({h_{u_l}}^n)^{2/3}}, \qquad j \in \mathcal{B}_4 \text{ "discharge",} \tag{6.115}$$

$$\begin{aligned}\zeta_{L(j)}^{n+1} &= 2\zeta_b(\boldsymbol{b}_j, \hat{t}^{n+1}) - \zeta_{R(j)}^0 + \dots \\ &\dots - \sqrt{\frac{\frac{1}{2}({h_s}_{L(j)}^n + {h_s}_{R(j)}^n)}{g}}u_j^n, \quad j \in \mathcal{B}_5\text{,"Riemann",}\end{aligned} \tag{6.116}$$

$$\zeta_{L(j)}^{n+1} = \zeta_{R(j)}^n, u_j^n > 0 \qquad j \in \mathcal{B}_6\text{,"outflow",} \tag{6.117}$$

$$\begin{aligned}\frac{\zeta_{L(j)}^{n+1} - \zeta_{L(j)}^n}{\Delta t^n} &= \sqrt{g\frac{1}{2}({h_s}_{L(j)}^n + {h_s}_{R(j)}^n)}\dots \\ \dots(\frac{\zeta_{R(j)}^n - \zeta_{L(j)}^n}{\Delta x_j} &+ s_b(\boldsymbol{b}_j, \hat{t}^{n+1})), u_j \le 0, \qquad j \in \mathcal{B}_6\text{,"outflow",}\end{aligned} \tag{6.118}$$

$$\zeta_{L(j)}^{n+1} = h_b(\sum_{l\in\mathcal{B}_7} q_l^n), \qquad j \in \mathcal{B}_7\text{,"Qh",} \tag{6.119}$$

where h_s is the cell-centered water depth, i.e.

$$h_{sk} = \zeta_k - bl_k, \tag{6.120}$$

$\zeta_b(\boldsymbol{x}, t)$ is a user-prescribed time-varying water level at boundary Γ_1, similar for normal slope $s_b(\boldsymbol{x}, t)$ and normal velocity $u_b(\boldsymbol{b}_j, \hat{t}^{n+1})$, and $Q_b(t)$ is a user-prescribed time-varying discharge at boundary Γ_4. Furthermore, $\hat{t}^{n+1}$ is an estimate of the next time level t^{n+1}.

Note that at "Qh" boundaries the discharge q_j^n is used, which is according to Algorithm (19)

$$q_j^n = {A_{u_j}}^{n-1}\left(\theta_j u_j^n + (1-\theta_j)u_j^{n-1}\right). \tag{6.121}$$

The function $h_b(Q)$ is user-provided by means of a table. It will not be discussed further.

We do neither mention the threshold on ${h_{u_j}}^n$ for the discharge boundaries under outflow conditions $Q_b(t) < 0$, nor a threshold on $\frac{1}{2}({h_s}_{L(j)}^n + {h_s}_{R(j)}^n)$ at the Neumann boundaries.

The second category boundary conditions only affect the reconstruction of the cell-centered full velocity vectors $\boldsymbol{u}_c$ near the boundary:

$${\boldsymbol{u_c}}_{L(j)}^n = u_j^n\boldsymbol{n}_j + v_b(\boldsymbol{b}_j, \hat{t}^{n+1})\boldsymbol{t}_j, \qquad j \in \mathcal{B}_8, \qquad \text{"tangentialvelocity",} \tag{6.122}$$

$${\boldsymbol{u_c}}_{L(j)}^n = \boldsymbol{u}_b(\boldsymbol{b}_j, \hat{t}^{n+1}), \qquad j \in \mathcal{B}_9, \qquad \text{"ucxucyadvectionvelocity",} \tag{6.123}$$

$${\boldsymbol{u_c}}_{L(j)}^n = u_b(\boldsymbol{b}_j, \hat{t}^{n+1})\boldsymbol{n}_j, \qquad j \in \mathcal{B}_{10}, \qquad \text{"normalvelocity",} \tag{6.124}$$

where $v_b(\boldsymbol{x}, t)$, $\boldsymbol{u}_b(\boldsymbol{x}, t)$ and $u_b(\boldsymbol{x}, t)$ are user-prescribed and time-varying at the boundary.

Remark 6.3.4. During the time-step from t^n to t^{n+1}, the cell center reconstructed is based on u_j^n (fully explicit). The boundary conditions are at the new time-level, on the other hand. This seems inconsistent.

Note that the "ucxucyadvectionvelocity" and "normalvelocity" conditions allow a supercritical inflow at the boundary. All three boundary conditions types are only relevant for inflow conditions.

Discharge boundaries: jbasqbnddownwindhs, qbndhutrs

For simplicity we only consider one discharge boundary and mention that there may be more than one. The face-based water depth in the evaluation of the flow area can optionally be set to a downwind approximation (for an inflowing discharge boundary) with the option `jbasqbnddownwindhs`, i.e.

$$h_{uj} = \zeta_{R(j)} - bl_{R(j)}, \quad j \in \mathcal{B}_4 \wedge \text{jbasqbnddownwindhs} = 1, \quad \text{"discharge"}. \tag{6.125}$$

Compare with Algorithm (2) where the face-based water depths h_{uj} are computed. They are overwritten in Algorithm (24) at the discharge boundary.

Algorithm 24 setau|discharge boundaries: adjustment to Algorithm (3) to overwrite water depths at the discharge boundaries

if jbasqbnddownwindhs=0 **then**
$\mathcal{B}^* = \{l \in \mathcal{B}_5 | h_{ul} > 0\}$
$h_{uj} = \max(0, \frac{\sum_{j \in \mathcal{B}^*} \zeta_{R(j)} w_{uj}}{\sum_{j \in \mathcal{B}^*} w_{uj}}), \quad j \in \mathcal{B}^*$
end if
if jbasqbnddownwindhs=1 **then**
$h_{uj} = \zeta_{R(j)} - bl_{R(j)}, \quad j \in \mathcal{B}_5$
compute $A_{uj}, j \in \mathcal{B}_5$ as in Algorithm (3)
end if
$\hat{\mathcal{B}} = \{l \in \mathcal{B}_5 | h_{ul} \geq \text{qbndhutrs} \vee Q_b \geq 0\}$
$h_{uj} = 0, \quad j \in \mathcal{B}_5 \setminus \hat{\mathcal{B}}$
$A_{uj} = 0, \quad j \in \mathcal{B}_5 \setminus \hat{\mathcal{B}}$
$zu_j = \frac{Q_b (h_{uj})^{2/3}}{\sum_{l \in \hat{\mathcal{B}}} (h_{ul})^{2/3} A_{ul}}, \; j \in \mathcal{B}_5$

Remark 6.3.5. Since for some cases the water depth at the discharge boundaries are modified *after* the volumes V_k and cross-sectional wetted areas A_{uj} are computed, the water depth now seems inconsistent with the aforementioned quantities.

Riemann boundaries

At a Riemann boundary we do not allow any outgoing perturbation with respect to some *reference* boundary state to reflect back from the boundary. This is achieved by prescribing the incoming Riemann invariant. Note that we disregard directional effects. Using the D-Flow FM convention of a positive inward normal at the boundary, this can be put as

$$\boldsymbol{u} \cdot \boldsymbol{n} + 2\sqrt{gh} = u_b + 2\sqrt{gh_b} \tag{6.126}$$

where we take boundary values (ζ_b, u_b) as the reference boundary state. By using $h_b =$

$h + \zeta_b - \zeta$, the term $\sqrt{gh_b}$ can be linearized in ζ around $\zeta = \zeta_b$ as

$$\sqrt{gh_b} = \sqrt{g(h + \zeta_b - \zeta)} \approx \sqrt{gh} + \frac{1}{2}\sqrt{\frac{g}{h}}(\zeta_b - \zeta). \tag{6.127}$$

Substitution yields

$$\sqrt{\frac{g}{h}}\zeta + \boldsymbol{u}\cdot\boldsymbol{n} = u_b + \sqrt{\frac{g}{h}}\zeta_b. \tag{6.128}$$

Instead of prescribing a combination of velocity and water level, we prefer to prescribe the water level at the boundary, i.e. ζ_b. For the necessary, but unknown velocity u_b we use linear theory with respect to the initial state $(\zeta, u) = (\zeta^0, u^0) = (\zeta^0, 0)$.

Remark 6.3.6. We assume that the initial velocity field is zero in any case.

By assuming small perturbations with respect to the initial conditions and considering conservation of mass at the boundary, we have:

$$u_b h = \sqrt{gh}(\zeta_b - \zeta^0), \tag{6.129}$$

or

$$u_b = \sqrt{\frac{g}{h}}(\zeta_b - \zeta^0). \tag{6.130}$$

Substitution of this expression in Eqn. (6.128) yields

$$\sqrt{\frac{g}{h}}\zeta + \boldsymbol{u}\cdot\boldsymbol{n} = \sqrt{\frac{g}{h}}(2\zeta_b - \zeta^0). \tag{6.131}$$

Note that a similar approach is taken in **?**.

The discretization in D-Flow FM is then as shown in Eqn. (6.116):

$$\zeta^{n+1}_{L(j)} = 2\zeta_b(\boldsymbol{b}_j, \hat{t}^{n+1}) - \zeta^0_{R(j)} - \sqrt{\frac{\frac{1}{2}(h_{s}{}^{n}_{L(j)} + h_{s}{}^{n}_{R(j)})}{g}}u^n_j, \quad j \in \mathcal{B}_5, \quad \text{"Riemann"}.$$

Qh boundaries: qhrelax

Again, for simplicity we will only consider one Qh boundary. The applied water level zb_j is relaxed with a user-specified parameter `qhrelax` that turns Eqn. (6.119) into

$$\zeta^{n+1}_j = \text{qhrelax } h_b(\sum_{l\in\mathcal{B}_7} q^n_l) + (1 - \text{qhrelax})\ \zeta^n_j, \quad j \in \mathcal{B}_7$$

By $h_b(\sum_{l\in\mathcal{B}_7} q^n_l)$ we in fact always refer to this relaxed expression and will not mention the relaxation explicitly.

6.3.3 Imposing the discrete boundary conditions: jacstbnd

During a time-step from t^n to t^{n+1}, the discrete boundary conditions, i.e. Eqns. (6.112) to (6.119) and Eqns. (6.122) to (6.124), are imposed in the following manner:

- the reconstruction of the full velocity vectors $\boldsymbol{u}_{c_j}^n$ is modified with Algorithm (25) to account for the boundary conditions at the new time level $\hat{t}^{n+1}$,
- the water-level boundary conditions at the *new* time level $\hat{t}^{n+1}$ are applied to the water level at the *old* time level t^n with Algorithm (26),
- the velocity boundary conditions are imposed on $u_j^{n+1}, j \in \mathcal{B}_u$ in Algorithm (28),
- the system of equations, referred to as the "water-level equations", to obtain ζ^{n+1} is adjusted in Algorithm (27) near the boundaries,
- having computed the water levels ζ^{n+1} from the "water-level equation" with Algorithm (21), the water levels in the fictitious boundary "cells" $\zeta_{Lj}^{n+1}, j \in \mathcal{B}_\zeta$ are computed with Algorithm (26),
- the velocities at the new time level u_j^{n+1} are computed with Algorithm (19) which requires no modifications since f_u and r_u were properly adjusted in Algorithm (28).

Remark 6.3.7. The boundary conditions at the *new* time level $\hat{t}^{n+1}$ are applied to the cell-center reconstruction of the full velocity vectors $\boldsymbol{u}_c^n$ at the old time level t^n.

Remark 6.3.8. It is unclear why boundary conditions need to be applied again to the water level at the old time level t^n at the beginning of the time step. They where applied at the end of the previous time step. Furthermore, conditions from the *new* time level $\hat{t}^{n+1}$ are now applied to the water level at the previous time level t^n.

Remark 6.3.9. It is unclear why boundary conditions need to be applied to the water level at the new time level $\hat{t}^{n+1}$ right after solving the "water-level equation" with Algorithm (26), since the fictitious boundary "cells" are included in the solution vector $\boldsymbol{s} = (\zeta_1, \zeta_2, \dots)^t$ and the discrete system of Eqn. (6.81) is augmented with the discrete boundary conditions of Eqns. (6.94) to (6.101) in Algorithm (27).

Algorithm 25 setucxucyucxuucyu|boundary conditions: adjustment to Algorithm (6) to satisfy the boundary conditions

$$\boldsymbol{u}_{cL(j)}^n = \begin{cases} \boldsymbol{u}_{cR(j)}^n, & j \in \mathcal{B}_2 \vee (j \in \mathcal{B} \wedge \text{jacstbnd} = 1) \\ (\boldsymbol{u}_{cR(j)}^n \cdot \boldsymbol{n}_j)\boldsymbol{n}_j, & j \in \mathcal{B} \setminus \mathcal{B}_2 \wedge \text{jacstbnd} = 0 \\ u_j^n \boldsymbol{n}_j + v_b(\boldsymbol{b}_j, \hat{t}^{n+1})\boldsymbol{t}_j, & j \in \mathcal{B}_8 \\ \boldsymbol{u}_b(\boldsymbol{b}_j, \hat{t}^{n+1}) & j \in \mathcal{B}_9 \\ u_b(\boldsymbol{b}_j, \hat{t}^{n+1})\boldsymbol{n}_j, & j \in \mathcal{B}_{10} \end{cases}$$

Remark 6.3.10. The discretization of the "outflowboundary" condition, Γ_6, in Algorithm (26) is different from the one in Algorithm (27) and seems incomplete. The condition for $u_j \leq 0$ is missing.

Remark 6.3.11. The "Qh" boundary condition is ineffective in Algorithm (26), since it is missing from Eqns. (6.132) and (6.133).

The water-level boundary conditions are inserted into the system of equations as follows. Firstly, Eqns. (6.112) to (6.119) show that for some z_b the boundary conditions can be put as

$$\zeta_{L(j)}^{n+1} = z_b, \qquad j \in \mathcal{B}_\zeta \setminus \mathcal{B}_2, \tag{6.134}$$

$$\frac{\zeta_{R(j)}^{n+1} - \zeta_{L(j)}^{n+1}}{\Delta x_j} = s_b(\boldsymbol{b}, \hat{t}^{n+1}), \quad j \in \mathcal{B}_2. \tag{6.135}$$

Algorithm 26 sets01zbnd: apply boundary conditions to water levels ζ^n or ζ^{n+1}

$$z_b = \begin{cases} (1-\alpha_{smo})\,\zeta_j^0 + \alpha_{smo}\zeta_b(\boldsymbol{b}_j, \hat{t}^{n+1}), & j \in \mathcal{B}_1 \\ \zeta_j^{n+1}, & j \in \mathcal{B}_2 \\ 2\zeta_b(\boldsymbol{b}_j, \hat{t}^{n+1}) - \sqrt{\frac{\max(\frac{1}{2}(h_s{}^n_{L(j)} + h_s{}^n_{R(j)}), \epsilon_{hs})}{g}} u_j^n, & j \in \mathcal{B}_5 \\ (1-\alpha_{smo})\zeta^0_{L(j)} + \alpha_{smo}\, h_b(\sum_{l\in\mathcal{B}_7} q_l^n), & j \in \mathcal{B}_7 \end{cases}$$

$$z_b = \max(zb - \tfrac{p_{atm\,L(j)} - p_{av}}{\rho_{mean}\, g}, bl_{L(j)} + 10^{-3}), \quad j \in \mathcal{B}_\zeta \setminus \mathcal{B}_6$$

if aply to ζ^n **then**

$$\zeta^n_{L(j)} = \begin{cases} z_b, & j \in \mathcal{B}_1 \cup \mathcal{B}_2 \cup \mathcal{B}_3 \cup \mathcal{B}_5 \\ \max(\zeta^n_{R(j)}, bl_{R(j)}), & u_j^n > 0, \quad j \in \mathcal{B}_6 \end{cases} \tag{6.132}$$

else {apply to ζ^{n+1}}

$$\zeta^{n+1}_{L(j)} = \begin{cases} z_b, & j \in \mathcal{B}_1 \cup \mathcal{B}_2 \cup \mathcal{B}_3 \cup \mathcal{B}_5 \\ \max(\zeta^n_{R(j)}, bl_{R(j)}), & u_j^n > 0, \quad j \in \mathcal{B}_6 \end{cases} \tag{6.133}$$

end if

The rows in the system that are affected by these boundary conditions are the rows that correspond to the fictitious boundary "cell" $L(j)$ and the neighboring internal cell $R(j)$. The latter may come as a surprise, but is due to our constraint that the system should remain symmetric. The general form of the system for these rows is obtained by substituting $k = L(j$ and $k = R(j), j \in \mathcal{B}_\zeta$in Eqn. (6.81) respectively, and using $O(L(j), j) = R(j)$ and $O(R(j), j) = L(j)$, i.e.

$$\left.\begin{array}{llll} B_r{}^{n+1(i)}_{L(j)}\ \zeta^{n+1(i+1)}_{L(j)} & & +C_r{}^n_j\ \zeta^{n+1(i+1)}_{R(j)} & = d_r{}^{n+1(i)}_{L(j)}, \\ B_r{}^{n+1(i)}_{R(j)}\ \zeta^{n+1(i+1)}_{R(j)} & + \sum_{l\in\mathcal{J}(R(l))\setminus j} C_r{}^n_l\ \zeta^{n+1(i+1)}_{O(R(j),l)} & +C_r{}^n_j\ \zeta^{n+1(i+1)}_{L(j)} & = d_r{}^{n+1(i)}_{R(j)}, \end{array}\right\} j \in \mathcal{B}_\zeta.$$

Combining these expressions yields for the non-Neumann boundary conditions

$$\left.\begin{array}{lll} & & = z_b, \\ B_r{}^{n+1(i)}_{R(j)}\ \zeta^{n+1(i+1)}_{L(j)} & + \sum_{l\in\mathcal{J}(R(l))\setminus j} C_r{}^n_l\ \zeta^{n+1(i+1)}_{O(R(j),l)} & = d_r{}^{n+1(i)}_{R(j)} - C_r{}^n_j\ z_b, \end{array}\right\} j \in \mathcal{B}_\zeta \setminus \mathcal{B}_2$$

and for the Neumann boundary condition

$$\left.\begin{array}{lll} -C_r{}^n_j\ \zeta^{n+1(i+1)}_{L(j)} & & +C_r{}^n_j\ \zeta^{n+1(i+1)}_{R(j)} = \\ & & -C_r{}^n_j \Delta x_j s_b(\boldsymbol{b}_j, \hat{t}^{n+1}), \\ B_r{}^{n+1(i)}_{R(j)}\ \zeta^{n+1(i+1)}_{R(j)} & + \sum_{l\in\mathcal{J}(R(l))\setminus j} C_r{}^n_l\ \zeta^{n+1(i+1)}_{O(R(j),l)} & +C_r{}^n_j\ \zeta^{n+1(i+1)}_{L(j)} =, \\ & & d_r{}^{n+1(i)}_{R(j)}, \end{array}\right\} j \in \mathcal{B}_2.$$

The consequences for the matrix elements are shown in Algorithm (27).

The velocity boundary conditions appear in the system in the following manner. The condition for the face-normal velocity components can be expressed as

$$u_j^{n+1} = z u_j, \quad j \in \mathcal{B}_u = \mathcal{B}_3 \cup \mathcal{B}_5, \tag{6.136}$$

Algorithm 27 s1nod|boundary conditions: adjustments to Algorithm (16) to satisfy the boundary conditions in the water-level equation

$B_{r_k}^{\,n+1(i)}\ \zeta_k^{n+1(i+1)} + \sum_{j\in\mathcal{J}(k)} C_{r_j}^{\,n}\ \zeta_{O(k,j)}^{n+1(i+1)} = d_{r_k}^{\,n+1(i)}$, Eqn. (6.81)

$$B_{r_{L(j)}}^{\,n+1(i)} = 1, \quad j \in \mathcal{B}_\zeta$$

$$z_b = \begin{cases} (1-\alpha_{smo})\,\zeta_j^0 + \alpha_{smo}\zeta_b(\boldsymbol{b}_j, \hat{t}^{n+1}), & j \in \mathcal{B}_1 \\ -s_b(\boldsymbol{b}_j, \hat{t}^{n+1})\Delta x_j C_{r_j}^{\,n}, & j \in \mathcal{B}_2 \\ 2\zeta_b(\boldsymbol{b}_j, \hat{t}^{n+1}) - \sqrt{\frac{\max(\frac{1}{2}(h_s{}_{L(j)}^{\,n} + h_s{}_{R(j)}^{\,n}), \epsilon_{hs})}{g}}\, u_j^n, & j \in \mathcal{B}_5 \\ \zeta_{R(j)}^n,\ u_j^n > 0, & j \in \mathcal{B}_6 \\ \zeta_{L(j)}^n + \Delta t^n \sqrt{g\tfrac{1}{2}(h_s{}_{L(j)}^{\,n} + h_s{}_{R(j)}^{\,n})}(\zeta_{R(j)}^n - \zeta_{L(j)}^n + s_b(\boldsymbol{b}_j, \hat{t}^{n+1})),\ u_j \le 0, & j \in \mathcal{B}_6 \\ (1-\alpha_{smo})\zeta_{L(j)}^0 + \alpha_{smo}\, h_b(\sum_{l\in\mathcal{B}_7} q_l^n), & j \in \mathcal{B}_7 \end{cases}$$

$$z_b = \max(zb - \tfrac{p_{atm\,L(j)} - p_{av}}{\rho_{mean}\, g}, bl_{L(j)} + 10^{-3}), \quad j \in \mathcal{B}_\zeta$$

$$\left.\begin{array}{ll} d_{r_{R(j)}}^{\,n+1(i)} & = d_{r_{R(j)}}^{\,n+1(i)} - C_{r_j}^{\,n} z_b, \\ B_{r_{L(j)}}^{\,n+1(i)} & = 1, \\ C_{r_j}^{\,n} & = 0, \\ d_{r_{L(j)}}^{\,n+1(i)} & = z_b, \end{array}\right\} \quad j \in \mathcal{B}_\zeta \setminus \mathcal{B}_2$$

$$\left.\begin{array}{ll} B_{r_{L(j)}}^{\,n+1(i)} & = -C_{r_j}^{\,n}, \\ d_{r_{L(j)}}^{\,n+1(i)} & = -C_{r_j}^{\,n}\Delta x_j s_b(\boldsymbol{b}_j, \hat{t}^{n+1}), \end{array}\right\} \quad j \in \mathcal{B}_2$$

$$\left.\begin{array}{lcl} C_{r_j}^{\,n} & = & -B_{r_{R(j)}}^{\,n+1(i)}, \\ B_{r_{L(j)}}^{\,n+1(i)} & = & -C_{r_j}^{\,n}, \\ d_{r_{L(j)}}^{\,n+1(i)} & = & 0, \\ B_{r_{R(j)}}^{\,n+1(i)} & = & B_{r_{R(j)}}^{\,n+1(i)} - C_{r_j}^{\,n}, \end{array}\right\} \quad j \in \mathcal{B}_u$$

where according to Eqn. (6.114)

$$zu_j = u_b(\boldsymbol{b}_j, \hat{t}^{n+1}), \quad j \in \mathcal{B}_3 \tag{6.137}$$

and zu_j is computed with Algorithm (24) for the discharge boundaries $j \in \mathcal{B}_5$. Since the velocity at the next time level with Eqn. (6.72) in Algorithm (12)

$$u_j^{n+1} = -f_{u_j}{}^n(\zeta_{R(j)}^{n+1} - \zeta_{L(j)}^{n+1}) + r_{u_j}{}^n,$$

the adjustments to Algorithm (12) are obvious and presented in Algorithm (28). Note that the velocity boundary conditions are relaxed with a parameter α_{smo} from the initial conditions, assumed zero. This will be explained in the next section.

Algorithm 28 furu|boundary conditions: adjustments to Algorithm (12) to satisfy the boundary conditions of the form $u_j^{n+1} = zu_j, j \in \mathcal{B}_u$ in $u_j^{n+1} = -f_{u_j}{}^n(\zeta_{R(j)}^{n+1} - \zeta_{L(j)}^{n+1}) + r_{u_j}{}^n$

$$\left.\begin{array}{rcl} f_{u_j}{}^n & = & 0, \\ r_{u_j}{}^n & = & \alpha_{smo} zu_j, \end{array}\right\} \quad j \in \mathcal{B}_u$$

Returning to the system of water-level equations, the consequence of the velocity boundary conditions for the matrix element $C_{r_j}{}^n$ can be seen in Algorithm (13), i.e.

$$C_{r_j}{}^n = 0, \quad j \in \mathcal{B}_u. \tag{6.138}$$

However, we want to apply a homogeneous Neumann condition to the water level at the velocity boundary:

$$\zeta_{R(j)}^{n+1} - \zeta_{L(j)}^{n+1} = 0 \in \mathcal{B}_u. \tag{6.139}$$

This can for arbitrary non-zero $C_{r_j}{}^n$ be formulated as

$$\left.\begin{array}{lll} -C_{r_j}{}^n\ \zeta_{L(j)}^{n+1(i+1)} & & +C_{r_j}{}^n\ \zeta_{R(j)}^{n+1(i+1)} = \\ (B_{r_{R(j)}}{}^{n+1(i)} - C_{r_j}{}^n)\ \zeta_{R(j)}^{n+1(i+1)} & + \sum_{l \in \mathcal{J}(R(l)) \setminus j} C_{rl}{}^n\ \zeta_{O(R(j),l)}^{n+1(i+1)} & +C_{r_j}{}^n\ \zeta_{L(j)}^{n+1(i+1)} = \\ & & d_{r_{R(j)}}{}^{n+1(i)}, \end{array}\right\} j \in \mathcal{B}_u.$$

To obtain a sensible order of magnitude, we set, as shown in Algorithm (27),

$$C_{r_j}{}^n = -B_{r_{R(j)}}{}^{n+1(i)}, \quad j \in \mathcal{B}_u. \tag{6.140}$$

6.3.4 Relaxation of the boundary conditions: Tlfsmo

In Algorithms (26), (28) and (27) the boundary conditions are relaxed from the initial values with a parameter α_{smo} that turns the discrete boundary conditions into

$$\zeta_{L(j)}^{n+1} = (1 - \alpha_{smo})\zeta_{L(j)}^0 + \alpha_{smo}\, \zeta_b(\boldsymbol{b}_j, \hat{t}^{n+1}), \quad j \in \mathcal{B}_1, \text{"waterlevel"}, \tag{6.141}$$

$$u_j^{n+1} = \alpha_{smo}\, u_b(\boldsymbol{b}_j, \hat{t}^{n+1}), \quad j \in \mathcal{B}_3, \text{"velocity"}, \tag{6.142}$$

$$u_j^{n+1} = \alpha_{smo}\, \frac{Q_b(\hat{t}^{n+1})(h_{u_j}{}^n)^{2/3}}{\sum_{l \in \mathcal{B}_4} A_{u_l}{}^n (h_{u_l}{}^n)^{2/3}}, \quad j \in \mathcal{B}_4 \text{ "discharge"}, \tag{6.143}$$

$$\zeta_{L(j)}^{n+1} = (1 - \alpha_{smo})\zeta_{L(j)}^0 + \alpha_{smo}\, h_b(\sum_{l \in \mathcal{B}_7} q_l^n), \quad j \in \mathcal{B}_7, \text{"Qh"} \tag{6.144}$$

and similar for the continuous formulation. The parameter α_{smo} is computed from the user-prescribed parameter `Tlfsmo` as

$$\alpha_{smo} = \min(\frac{\hat{t}^{n+1} - t^0}{\text{Tlfsmo}}, 1). \tag{6.145}$$

Remark 6.3.12. The second category velocity boundary conditions are *not* being relaxed in the same way as the first category, but maybe they should.

Remark 6.3.13. A zero initial velocity field is assumed in the relaxation of the boundary conditions.

6.3.5 Atmospheric pressure: PavBnd, rhomean

Local changes in atmospheric pressure at the boundaries, except for the outflow boundary, are accounted for by correcting the water level with

$$-\frac{p_{atm_{L(j)}} - p_{av}}{\rho_{mean}\, g} \tag{6.146}$$

as shown in Algorithms (26) and (27), where p_{atm_k}, p_{av} (called `PavBnd` in D-Flow FM) and ρ_{mean} (called `rhomean` in D-Flow FM) are the user-supplied atmospheric pressure in cell k, average pressure and average density, respectively.

6.3.6 Adjustments of numerical parameters at and near the boundary

At and *near* the boundary, the advection scheme and time-integration method are adjusted with Algorithm (29). The time-integration parameter θ_j is set to 1 and the advection scheme is set to '6' at and near the water-level boundary. See Algorithm (4) for an overview of the advection schemes. These settings are not only applied to the water-level boundary faces, but also to all faces of internal cells that are adjacent to the water-level boundary.

For the velocity boundary conditions, the time integration parameter θ_j is set to 1 at and near the boundary. Advection is turned off by setting the advection scheme to -1, but only for the faces at the boundary that is.

Algorithm 29 flow_initexternalforcings: adjust numerical settings near the boundaries

$$\left.\begin{array}{rcll} \theta_l & = & 1, & l \in \mathcal{J}(R(j)) \\ iadv_l & = & 6, & l \in \mathcal{J}(R(j)) \end{array}\right\} \quad j \in \mathcal{B}_\zeta$$

$$\left.\begin{array}{rcll} \theta_l & = & 1, & l \in \mathcal{J}(R(j)) \\ iadv_j & = & -1 & \end{array}\right\} \quad j \in \mathcal{B}_u$$

6.3.7 Viscous fluxes: irov

Momentum diffusion is elaborated in Section 6.1.4 and in particular Algorithm (11). It will be clear that we can not evaluate the viscous stresses $\boldsymbol{t}_{uj}$ at *closed boundaries* as in Eqn. (6.61). Instead, boundary conditions need to be imposed. These are:

$$\boldsymbol{t}_{uj} = \begin{cases} \boldsymbol{0}, & \text{irov=0, free slip,} \\ -u_j^*|u_j^*|\boldsymbol{s}_j, & \text{irov=1, partial slip,} \\ -\nu_j \frac{U_j}{\Delta y_j}\boldsymbol{s}_j, & \text{irov=2, no slip,} \end{cases} \tag{6.147}$$

where u_j^* is the friction velocity, $\boldsymbol{s}_j = \boldsymbol{n}_j^{\perp}$ a unit tangential boundary vector whose orientation we will not discuss and, if $R(j)$ is the boundary cell (note that $L(j)$ does not exist),

$$\Delta y_j = \frac{1}{2} b_{AR(j)} / w_{uj}. \tag{6.148}$$

The friction velocity is computed as

$$u_j^* = \frac{U_j \kappa}{C + d/z_0}, \tag{6.149}$$

with

$$U_j = \boldsymbol{u}_{cR(j)} \cdot \boldsymbol{s}_j, \tag{6.150}$$

z_0 the user specified roughness height, κ the Von Karman contanst, d a distance from the cell centroid perpendicular to the boundary face and C either 1 or 9.

The boundary cell-based momentum diffusion term then becomes

$$\nabla \cdot \left(\nu h^p (\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^t) \right) \big|_{\Omega_{R(j)}} \approx \boldsymbol{d}_{R(j)} + \frac{\sum_{l \in \{m \in \mathcal{B}_0 | R(m) = R(j)\}} \boldsymbol{t}_{ul} w_{ul} 1_{l,R(j)}}{b_{AR(j)}}, \quad j \in \mathcal{J}_0, \tag{6.151}$$

where $\boldsymbol{d}_{R(j)}$ represents the contribution from the non-boundary faces of boundary cell $R(j)$ as given by Eqn. (6.60) for $k = R(j)$.

Remark 6.3.14. Comparing the contribution of the viscous boundary stress with the expression for the contribution of the internal faces $\boldsymbol{d}_k$ in Eqn. (6.60) reveals that the `istresstype` does not apply to the contribution of the boundary stresses. Obviously $p = 0$ is applied here. See Remark 6.1.17 in this respect.

6.4 Summing up: the whole computational time step

With the discretization explained in the previous sections, we are now able to sum up the computational time step. It is shown in Algorithms (30), (31) and (32). The data being computed and updated are shown in Table 6.2.

Algorithm 30 flow_single_timestep: perform a computational time step from t^n to t^{n+1} and obtain ζ_k^{n+1}, u_j^{n+1}, $q_j^{n_1}$, $q_{a_j}^{n+1}$ and V_k^{n+1}, $\forall k$ and $\forall j$

flow_initimestep: compute derived data $h_{u_j}^n$, $A_{u_j}^n$, et cetera and perform the predictor phase of the fractional time step to obtain $\mathcal{A}_{\mathrm{i}j}$ and $\mathcal{A}_{\mathrm{e}j}$, $\forall j$ with Algorithm (31)

step_reduce: compose system of water-level equations, solve the system to obtain ζ_k^{n+1}, compute volumes and wetted areas V_k^{n+1}, A_k^{n+1}, set $h_{u_j}^n$ to zero (old time-level) for disabled faces during solve and perform the corrector phase to obtain u_j^{n+1}, q_j^{n+1} and $q_{a_j}^{n+1}$ with Algorithm (20)

Table 6.2: *data during a computational time step from t^n to t^{n+1} with Algorithm (30); the translation to D-Flow FM nomenclature is shown in the last column*

input		
time instant	t^n	`time0`
water level	ζ_k^n	`s0`
face-normal velocity components	u_j^n	`u0`
fluxes	q_j^n, ${q_a}_j^n$	`q1`, `qa`
water column volumes	V_k^n	`vol1`
wet bed areas	A_k^n	`A1`
time step	Δt^{n-1}	`dts`

during the time step (a selection)		
estimate for next time instant	$\hat{t}^{n+1}$	`time0+dts`
the face based water height	${h_u}_j^n$	`hu`
wetted cross-sectional areas	${A_u}_j^n$	`Au`
the water column heights	${h_s}_k^n$	`hs`
the cell-centered full velocity vectors	$\boldsymbol{u}_{c_k}^n$, $\boldsymbol{u}_{q_k}^n$	`ucx`, `ucy`, `ucxq`, `ucyq`
node-based full velocity vectors	$\boldsymbol{u}_{n_i}^n$	`ucnx`, `ucny`
momentum equation terms	$\mathcal{A}_{\mathrm{e}j}$, $\mathcal{A}_{\mathrm{i}j}$	`adve`, `advi`

output		
time instant	t^{n+1}	`time1`
water level	ζ_k^{n+1}	`s1`
face-normal velocity components	u_j^{n+1}	`u1`
fluxes	q_j^{n+1}, ${q_a}_j^{n+1}$	`q1`, `qa`
water column volumes	V_k^{n+1}	`vol1`
wet bed areas	A_k^{n+1}	`A1`
time step	Δt^n	`dts`

Algorithm 31 flow_initimestep: compute derived data and perform the predictor phase of the fractional time step

${h_s}_k^n = \zeta_k^n - bl_k, \quad \forall k$
$\hat{t}^{n+1} = t^n + \Delta t^n$
flow_setexternalboundaries: update boundary values at time instant $\hat{t}^{n+1}$
sethu: compute face-based water heights ${h_u}_j^n$ with Algorithm (2) and Algorithm (39) explained later
setau: compute the flow area ${A_u}_j^n$ with Algorithms (3) and (24)
setumod: compute cell-based full velocity vectors $\boldsymbol{u}_{c_k}^n$, $\boldsymbol{u}_{q_k}^n$, first-order upwind velocity $\boldsymbol{u}_{u_j}^{Ln}$ and add Coriolis forces and viscous fluxes to $\mathcal{A}_{\mathrm{e}j}$ with Algorithm (32)
compute bed friction coefficients
compute time step Δt^{n+1}
advec: add advection terms to $\mathcal{A}_{\mathrm{i}j}$ and $\mathcal{A}_{\mathrm{e}j}$ with Algorithms (4) and (40)
setextforcechkadvec: add external forces to $\mathcal{A}_{\mathrm{i}j}$ and $\mathcal{A}_{\mathrm{e}j}$ and make adjustments for small water depths, Algorithm (35) explained later

Algorithm 32 setumod: compute cell-based full velocity vectors $\boldsymbol{u}_{ck}$, $\boldsymbol{u}_{q_k}$, first-order upwind velocity $\boldsymbol{u}^L_{u\,j}$ and add Coriolis forces and viscous fluxes to $\mathcal{A}_{\mathsf{e}j}$

setucxucyucxuucyu: reconstruct cell centered velocity vectors $\boldsymbol{u}_{ck}$ and $\boldsymbol{u}_{q_k}$, and set first-order upwind fluxes $\boldsymbol{u}^L_{u\,j}$ with Algorithms (6) and (25)
compute tangential velocities v_j from the cell-centered velocities $\boldsymbol{u}_{q_k}$ with Eqn. (6.37)
compute Coriolis forces
setcornervelocities: interpolate nodal velocity vectors $\boldsymbol{u}_n$ from cell-centered velocity vectors $\boldsymbol{u}_c$ with Algorithm (7)
compute viscous momentum fluxes, except at the default boundaries, i.e. $j \notin \mathcal{J}_0$, and add to $\mathcal{A}_{\mathsf{e}j}$ with Algorithm (11)
compute viscous momentum fluxes near the default boundaries $j \in \mathcal{J}_0$ and add to $\mathcal{A}_{\mathsf{e}j}$

6.5 Flooding and drying

The governing equations, Eqns. (6.1) and (6.2), can only be formulated for positive water heights, i.e. the "wet" part of the whole domain where $h > 0$. However, our domain may also contain areas that are dry. If we let Ω denote the *whole* domain, then we can define the *wet* part $\bar{\Omega}(t)$ as

$$\bar{\Omega}(t) = \{\boldsymbol{x} \in \Omega | h(\boldsymbol{x}, t) > 0\}. \tag{6.152}$$

In other words, when we are faced with flooding and drying we are actually attempting to solve a moving boundary problem, where $\partial\bar{\Omega}(\boldsymbol{x}, t)$ is the moving boundary.

In D-Flow FM the governing equations are discretized on a stationary mesh (in two dimensions). Looking at the governing equations, Eqns. (6.1) and (6.2), we can immediately identify two difficulties in the numerical treatment of the wet/dry boundary:

- the spatial discretization near the moving wet/dry boundary, and
- the temporal discretization at cells that become wet or dry during a time-step.

One may think of two possible approaches to overcome the difficulties with the spatial operators near the moving boundary: the stencil could be adapted such that it does not extend to the dry part of the domain, or, alternatively, the spatial operators could be left unmodified and the flow variables in the dry part could be given values that comply with the (moving) boundary conditions. We leave it up to the reader to decide which approach is taken in D-Flow FM and restrict ourselves by describing the measures taken in D-Flow FM to account for the wet/dry boundary.

6.5.1 Wet cells and faces: epshu

We can distinguish between wet (or dry) cells and wet (or dry) faces, which are the discrete counterpart of $\bar{\Omega}$ (or $\Omega \setminus \bar{\Omega}$). Dry faces are identified by setting their face-based water height to zero, i.e. $h_{uj} = 0$. In D-Flow FM this occurs at two occasions during a time-step:

- at the beginning of the time step, and based on the water level ζ^n_k, faces for which $h_u \leq \epsilon_{hu}$ are disabled by setting them to zero with Algorithm (33). Note that ϵ_{hu} is a threshold which is called `epshu` in D-Flow FM and is user specified,
- during the time step, the water-level $\zeta^{n+1(i)}_k$ may have dropped below the bedlevel. Depending on `poshcheck`, the time-step is repeated with a smaller time step (type 1) or all faces of the cell are deactivated by setting their $h^n_{u_j}$ to zero, see Algorithm (17), and the water-level equation is solved again.

Algorithm 33 sethu|drying and wetting: adjustment to Algorithm (2) to account for drying and wetting

compute $h_{u_j}^{\,n}$ with Algorithm (2)
disable dry faces by setting $h_{u_j}^{\,n} = 0$ if $h_{u_j}^{\,n} \leq \epsilon_{hu}$

6.5.2 Spatial discretization near the wet/dry boundary

In D-Flow FM, dry faces affect the discretization of:

- the wet bed areas A_k^n and water-column volumes V_k^n,
- momentum advection: set to zero in Algorithm (4) (not mentioned there),
- bed friction forces: set to zero, and
- viscous fluxes: set to zero for $h_{u_j}^{\,n}$ in Algorithm (32).

The computation of the wet bed areas and water-columns was already presented in Algorithm (18). As can been seen in Algorithm (18), the bed is assumed constant in case of no non-linear iterations. That is, as far as the water-column volumes and wet bed areas are concerned. See Remark 6.1.4 in that respect. With a constant bed level, no modifications are necessary for the computation of the wet bed area. The bed is either completely wet, or it isn't. In case of non-linear computations, however, the bed is assumed non-constant in a cell. The expressions for V_k and A_k are then

$$V_k = \int_{\Omega_k \cap \bar{\Omega}} h \, \mathrm{d}\Omega \tag{6.153}$$

and

$$A_k = \int_{\Omega_k \cap \bar{\Omega}} \mathrm{d}\Omega \tag{6.154}$$

respectively, where we have used that $\Omega_k \cap \bar{\Omega}$ indicates the wet part op the cell. These integrals are discretized as indicated in Algorithm (34).

Remark 6.5.1. Applying Gauss's theorem to Eqn. (6.154) yields

$$A_k = \int_{\partial\Omega_k \cap \bar{\Omega}} \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_k) \cdot \boldsymbol{n} \, \mathrm{d}l + \int_{\Omega_k \cap \partial\bar{\Omega}} \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_k) \cdot \boldsymbol{n} \, \mathrm{d}l, \tag{6.155}$$

where we have assumed outward positive normal vectors $\boldsymbol{n}$. It shows that we do not only need to integrate along (a part of) the edges of Ω_k, but also along the wet/dry boundary in cell $\partial\bar{\Omega} \cap \Omega_k$. Since this term is missing in the expression for A_k, the wet bed area of a partially wet cell is incorrectly computed.

6.5.3 Spatial discretization of the momentum equation for small water depths: chkadv, trshcorio

Recall that Eqn. (6.14) summarizes the spatial discretization of the momentum equation:

$$\frac{\mathrm{d}u_j}{\mathrm{d}t} = -\frac{g}{\Delta x_j}\left(\zeta_{R(j)} - \zeta_{L(j)}\right) - \mathcal{A}_{\mathrm{i}j} u_j - \mathcal{A}_{\mathrm{e}j} - \frac{g\|\boldsymbol{u}_j\|}{C^2 h} u_j,$$

Algorithm 34 volsur|non-linear iterations: compute water-column volume $V_k^{n+1(i+1)}$ and wet bed area $A_k^{n+1(i+1)}$

if no non-linear iterations **then**
use Algorithm (18)
else
compute $\Delta b_j = \max(bob_{1j}, bob_{2j}) - \min(bob_{1j}, bob_{2j})$ and wet cross-sectional area A_{uj} as in Algorithm (3)

$$\begin{aligned} A_k &= \sum_{j\in\{l\in\mathcal{J}(k)|1_{k,l}=1\}} \tfrac{1}{2}\Delta x_j \alpha_j \min(\tfrac{h_{uj}}{\Delta b_j}, 1) w_{uj} + \\ &\quad \sum_{j\in\{l\in\mathcal{J}(k)|1_{k,l}=-1\}} \tfrac{1}{2}\Delta x_j (1-\alpha_j) \min(\tfrac{h_{uj}}{\Delta b_j}, 1) w_{uj} \\ V_k &= \sum_{j\in\{l\in\mathcal{J}(k)|1_{k,l}=1\}} \tfrac{1}{2}\Delta x_j \alpha_j A_{uj} + \\ &\quad \sum_{j\in\{l\in\mathcal{J}(k)|1_{k,l}=-1\}} \tfrac{1}{2}\Delta x_j (1-\alpha_j) A_{uj} \end{aligned}$$

end if

where $\mathcal{A}_{\mathrm{e}j}$ and $\mathcal{A}_{\mathrm{i}j}$ represent the contributions of momentum advection, diffusion, Coriolis forces and external forces not being bed friction. These contribution are computed with Algorithm (4) for advection and Algorithm (32) for diffusion and Coriolis forces, respectively. The contributions to $\mathcal{A}_{\mathrm{e}j}$ by external forces not being the bed friction are added by Algorithm (35). It shows that for small water depths h_{uj} the term $\mathcal{A}_{\mathrm{e}j}$ is limited to zero from a user-specified threshold h_{chkadv}, called `chkadv` in D-Flow FM.

Algorithm 35 setextforcechkadvec: add external forces not being the bed friction forces to $\mathcal{A}_{\mathrm{e}j}$ and $\mathcal{A}_{\mathrm{i}j}$, and limit for vanishing water depths, in the expression:
$\frac{\mathrm{d}u_j}{\mathrm{d}t} = -\frac{g}{\Delta x_j}\left(\zeta_{R(j)} - \zeta_{L(j)}\right) - \mathcal{A}_{\mathrm{i}j} u_j - \mathcal{A}_{\mathrm{e}j} - \frac{g\|\boldsymbol{u}_j\|}{C^2 h} u_j$

add external forces to $\mathcal{A}_{\mathrm{e}j}$
if $h_{uj} > 0$ **then**
if $h_{sL(j)} < \frac{1}{2} h_{sR(j)} \ \wedge \ \mathcal{A}_{\mathrm{e}j} < 0 \ \wedge \ h_{sR(j)} < h_{\mathrm{chkadv}}$ **then**
$\mathcal{A}_{\mathrm{e}j} = \min(\frac{h_{sL(j)}}{h_{\mathrm{chkadv}}}, 1)\mathcal{A}_{\mathrm{e}j}$
else if $h_{sR(j)} < \frac{1}{2} h_{sL(j)} \ \wedge \ \mathcal{A}_{\mathrm{e}j} > 0 \ \wedge \ h_{sL(j)} < h_{\mathrm{chkadv}}$ **then**
$\mathcal{A}_{\mathrm{e}j} = \min(\frac{h_{sR(j)}}{h_{\mathrm{chkadv}}}, 1)\mathcal{A}_{\mathrm{e}j}$
end if
end if

Remark 6.5.2. It is unclear why the term $\mathcal{A}_{\mathrm{e}j}$ needs to be limited to zero for vanishing water depths.

Remark 6.5.3. It is unclear why only the term $\mathcal{A}_{\mathrm{e}j}$ is limited, and not the other terms. In the first place $\mathcal{A}_{\mathrm{i}j}$, but also the remaining terms in Eqn. (6.14).

Coriolis forces are computed and added to $\mathcal{A}_{\mathrm{e}j}$ in Algorithm (32). Besides the limitation described above, an additional limitation is performed. If f_{cj} is the Coriolis normal force at face j, then it is limited as indicated in Algorithm (36) with a threshold $h_{\mathrm{trshcorio}}$ called `trshcorio` in D-Flow FM.

Remark 6.5.4. The Coriolis forces are limited by Algorithm (36) *and* by Algorithm (35).

Algorithm 36 setumod|limitation of Coriolis forces: adjustment to Algorithm (32) to account for vanishing water depths

$h_{\text{trshcorio}} = 1$
$h_{min_j} = \min(h_{sL(j)}, h_{sR(j)})$
limit Coriolis forces f_{cj} to $f_{cj} \min(\frac{h_{min}}{h_{\text{trshcorio}}}, 1)$

6.5.4 Temporal discretization of the momentum equation near the wet/dry boundary

If the face is *dry at the beginning* of the time step, then it is assumed that during a time-step from t^n to t^{n+1} no water is fluxed through it. The face-normal velocity u_j^n is set to zero in such circumstances.

Remark 6.5.5. The assumption that during a time step no water is fluxed through a dry face that is dry at the old time instant imposes a time step limitation.

Remark 6.5.6. Setting the face-normal velocities in the dry area to zero does not seem to obey a moving wet/dry boundary condition. Hence, the spatial operators and temporal discretization are not allowed to be applied without modification. However, they are.

Setting ${h_u}_j^n = 0$ *during* the time step does *not* affect the computation of momentum advection and diffusion et cetera, as the terms $\mathcal{A}_\text{e}$ and $\mathcal{A}_\text{e}$ remain untouched. It only affects the water-column volumes V_k^{n+1} and wet bed areas A_k^{n+1}, face-normal velocities u_j^{n+1} and fluxes q_j^{n+1} and ${q_a}_j^{n+1}$ at the new time instant as can be seen from Algorithm (20), which do not appear in the discretization of the momentum equation.

Recall that the temporal discretization of the momentum equation is expressed by Eqn. (6.72) for $h_{uj} > 0$, or

$$u_j^{n+1} = -{f_u}_j^n(\zeta_{R(j)}^{n+1} - \zeta_{L(j)}^{n+1}) + {r_u}_j^n, \quad {h_u}_j^n > 0.$$

Extended for the situation when the face becomes wet or dry, it becomes

$$u_j^{n+1} = \begin{cases} 0, & \text{face is dry at beginning of the time step,} \\ 0, & \text{face is wet and becomes dry during the time step,} \\ -{f_u}_j^n(\zeta_{R(j)}^{n+1} - \zeta_{L(j)}^{n+1}) + {r_u}_j^n, & \text{face remains wet.} \end{cases} \tag{6.156}$$

If the face is still wet at the end of the time step from t^n to t^{n+1}, but becomes dry at the beginning of the new time step from t^{n+1} to t^{n+2} due to Algorithm (33), then its normal velocity component is left unmodified, since the velocities are only set at the end of the time step by Algorithm (19).

Remark 6.5.7. A face that was still wet at the end of the previous time-step, but *becomes* dry during the current time-step can have a non-zero normal-velocity at the old time level. It is being used in the evaluation of advection terms, diffusion terms et cetera at the beginning of the next time step, although the face is dry. In contrast, faces that were already dry from the end of the previous time step have zero normal-velocity.

6.6 Fixed Weirs

The flow over a fixed weir can not be modeled in D-Flow FM as is, but requires additional modeling, referred to as a "subgrid" model, see **?** and **?** for further details. This section elaborates on the numerical treatment of the fixed weirs.

In D-Flow FM weirs are discretely represented along mesh lines. In such a manner, faces can be identified that are located exactly *on top* of the weirs, and no computational cells are cut by a weir. A cell is either on one side of a fixed weir, or on the other side. Provided that the cross-sectional wet areas of the faces A_{u_j} have been properly modified to account for the fixed weir (if present), no modifications have to be made to the discretization of the continuity equation. The momentum equation, on the other hand, has to be modified such that we obtain our desired subgrid model.

6.6.1 Adjustments to the geometry: oblique weirs and FixedWeirContraction

Recall that the bed geometry is represented by the face-based bed-levels bob_1 and bob_2, see Section 6.1.2 and Algorithm (1). The wet cross-sectional area A_u is derived from it with Algorithm (3). In other words, the fixed weirs are properly represented by adjusting bob_1 and bob_2. Also appearing in the expression for A_u is the face width w_u. Weirs that are not aligned with the mesh, called oblique weirs for shortness, are projected to the (non-aligned) weir, i.e.

$$w_{uj} = \|\boldsymbol{x}_{i_R(j)} - \boldsymbol{x}_{i_L(j)}\| \, |\boldsymbol{n}_j \cdot \boldsymbol{n}_{wj}| c, \tag{6.157}$$

where $\boldsymbol{n}_{wj}$ is a unit vector normal to the part of the fixed weir that is associated with face j. The cross-sectional wetted area is decreased by the same amount as w_u by means of Algorithm (3).

Remark 6.6.1. Oblique weirs are not fully understood at this moment. We do not attempt to explain Eqn. (6.157) further.

In Eqn. (6.157) c is a user-specified contraction coefficient that accounts for obstacles in the flow that accompanied with the weir, such as pillars. It is called `FixedWeirContraction` in D-Flow FM.

The adjustments of bob_1, bob_2 and w_u are performed with Algorithm (37).

Algorithm 37 setfixedweirs: change geometry bob_1, bob_2 and w_u and advection type for fixed weirs

$bob_{1j} = \max(z_{cj}, bob_{1j})$
$bob_{2j} = \max(z_{cj}, bob_{2j})$
if left and right weir sill heights are prescribed and conveyance type > 0 **then**
  set bob_1 and bob_2 of adjacent faces, not described further
end if
adjust advection type of adjacent faces with Algorithm (38)
$w_{uj} = \|\boldsymbol{x}_{i_R(j)} - \boldsymbol{x}_{i_L(j)}\| \, |\boldsymbol{n}_j \cdot \boldsymbol{n}_{wj}| c$

6.6.2 Adjustment to momentum advection near, but not on the weir

Due to our subgrid modeling,the flow over a weir is discontinuous. In principle, this has is consequences for the discretization of all spatial operators near the weir and to this end the advection type near the weir is also adjusted by Algorithm (37). More precisely, *all* faces belonging to cells that are adjacent to weirs, except for the faces that are associated with a weir themselves, with Algorithm (38) have their advection type set to 4. As can be seen in Algorithm (4), only inflowing cell-centered velocities are used in the expression for momentum advection for scheme 4. Doing so, the advection at faces adjacent to and upstream of the weir are not affected by the cell-centered velocities near the weir.

Remark 6.6.2. Since the flow over a weir is discontinuous due to our subgrid modeling, one may need to discretize the spatial operators near the weir more rigorously.

Algorithm 38 setfixedweirscheme3onlink: set advection type to 4 of faces near the weir

if face j is associated with a fixed weir **then**
  $iadv_j = 21$
  $\theta_j = 1$
  for $l \in \{m \in \mathcal{J}(L(j)) \cup \mathcal{J}(R(j)) \,|\, iadv_m \neq 21\}$ **do**
    $iadv_l = 4$
    $\theta_l = 1$
  end for
end if

6.6.3 Adjustments to the momentum advection on the weir: FixedWeirScheme

We assume that a face j is located exactly *on top* of a fixed weir or not at all. Upstream of a fixed weir, a contraction zone exists. The cell-centered water-level upstream of the face ($L(j)$ if $u_j > 0$) represents the far-field water-level before the contraction zone. The water-level at the cell-centered water-level downstream of the face ($R(j)$ if $u_j > 0$) is used as a downwind approximation of the water-level *on top* of the fixed weir. In other words, the discretization at a face j represents the flow upstream of the weir at face j. This is the contraction zone, which is governed by energy conservation. The expansion zone downstream of the weir, is governed by momentum conservation and is directly resolved in the mesh without, in principle, further adjustments.

Assume that at face j is on top of a weir and that the flow is from the left $L(j)$ to the right neighboring cell $R(j)$. We require that:

- energy is conserved from cell $L(j)$ to $R(j)$, and
- the downstream water level $\zeta_{R(j)}$ should have no effect on the fixed weir in supercritical conditions. In this case the water height on top of the weir reached its minimum value of $\frac{2}{3}E_j$, where E_j is the far-field energy head above crest. Its computation will be discussed later.

Energy conservation is expressed by means of Bernoulli's equation, i.e.

$$\frac{1}{2}{u_{in}}_j^2 + g\zeta_{L(j)} = \frac{1}{2}u_j^2 + g(z_{Cj} + h_{uj}), \tag{6.158}$$

where u_{in_j} is a far-field velocity component in face-normal direction $\boldsymbol{n}_j$ and z_{Cj} is the crest level. For the water height at the weir h_{uj} a downwind approximation is used that obeys our second requirement:

$$h_{uj} = \max(\zeta_{R(j)} - z_{Cj}, \frac{2}{3}E_j), \tag{6.159}$$

where E_j is the far-field energy head above the crest. It is computed as

$$E_j = \zeta_{L(j)} - z_{Cj} + \frac{1}{2g}{u_{in}}_j^2. \tag{6.160}$$

Remark 6.6.3. Eqn. (6.160) is only valid along streamlines and consequently we may only consider flows that are perpendicular to the weir (1D flows) or are uniform along both sides of the weir.

Substitution of Eqn. (6.162) in Eqn. (6.158), some rearrangement of terms and division by Δx_j yields

$$\frac{1}{2\Delta x_j}\left(u_j^2 - {u_{in}}_j^2\right) = -\frac{g}{\Delta x}\left(\zeta_{R(j)} - \zeta_{L(j)}\right) - \frac{g}{\Delta x}\max(0, \frac{2}{3}E_j - (\zeta_{R(j)} - z_{Cj})) \tag{6.161}$$

This equation if brought into the form that is solved in D-Flow FM by adding the acceleration term, which only serves to relax to our stationary subgrid expression of Eqn. (6.161)

$$\frac{\mathrm{d}u_j}{\mathrm{d}t}+\frac{1}{2\Delta x_j}\left(u_j^2-{u_{in}}_j^2\right)=-\frac{g}{\Delta x}\left(\zeta_{R(j)}-\zeta_{L(j)}\right)-\frac{g}{\Delta x}\max(0,\frac{2}{3}E_j-(\zeta_{R(j)}-z_{Cj})) \tag{6.162}$$

Remark 6.6.4. Although momentum diffusion over the wear is missing in Eqn. (6.162), it is actually included in D-Flow FM.

Recall that the spatial discretization is summarized as shown in Eqn. (6.14):

$$\frac{\mathrm{d}u_j}{\mathrm{d}t}=-\frac{g}{\Delta x_j}\left(\zeta_{R(j)}-\zeta_{L(j)}\right)-\mathcal{A}_{\mathrm{i}j}u_j-\mathcal{A}_{\mathrm{e}j}-\frac{g\|\boldsymbol{u}_j\|}{C^2h}u_j.$$

The terms for fixed weirs can then be put as, assuming $u_j>0$:

$$\mathcal{A}_{\mathrm{i}j} = \frac{1}{2\Delta x_j}u_j, \tag{6.163}$$

$$\mathcal{A}_{\mathrm{e}j} = -\frac{1}{2\Delta x_j}{u_{in}}_j^2+\frac{g}{\Delta x}\max(0,\frac{2}{3}E_j-(\zeta_{R(j)}-z_{Cj})). \tag{6.164}$$

Remark 6.6.5. Although bed friction is not included in Eqn. (6.161), it is included in D-Flow FM by means of Eqn. (6.14). Its actual computation will not be discussed at this occasion.

The terms of Eqn. (6.159) and Eqns. (6.163) and (6.164) are prescribed *partly* with Algorithm (39) and *partly* with Algorithm (40). Note that the latter also adjusts the cell-centered velocity vectors $\boldsymbol{u}_c$.

Algorithm 39 sethu|fixed weir: adjustment to Algorithm (2); set h_u and part (one of two) to $\mathcal{A}_{\mathrm{e}}$

if $u_j>0$ **then**
  compute far-field velocity $\hat{\boldsymbol{u}}_{cL(j)}$ with Algorithm (41)
  $$u_{inj}=\begin{cases}\hat{\boldsymbol{u}}_{cL(j)}\cdot\boldsymbol{n}_{wj}, & \text{fixed weir scheme } 4\\ 0 & \text{fixed weir scheme } 5\\ \hat{\boldsymbol{u}}_{cL(j)}\cdot\boldsymbol{n}_j, & \text{otherwise}\end{cases}$$
  $E_j=\zeta_{L(j)}-z_{cj}+\frac{1}{2g}{u_{in}}_j^2$
  if $\zeta_{R(j)}<\zeta_{L(j)}$ **then**
    $h_{uj}=\max(\zeta_{R(j)}-z_{cj},\frac{2}{3}E_j)$
    $\mathcal{A}_{\mathrm{e}j}=-\frac{g}{\Delta x_j}\min(0,\zeta_{R(j)}-z_{cj}-\frac{2}{3}E_j)$
  end if
else
  as above by interchanging $L(j)$ and $R(j)$ and taking reversed orientation into account
end if

In Algorithms (39) and (40) a far-field velocity u_{in} is computed. For FixedWeirScheme 4, the far-field velocity is projected in weir-normal, instead of face-normal direction. For FixedWeirScheme 5 the kinetic energy is not used in the determination of the far-field energy head.

Remark 6.6.6. Starting from Bernoulli's equation to derive the weir subgrid model, it seems inconsistent to project the far-field velocity in *weir-normal* direction for scheme 4, while using the *face-normal* velocity as the weir crest velocity.

Algorithm 40 advec|fixed weir: adjustment to Algorithm (4) for fixed weir; set $\mathcal{A}_\mathsf{i}$ and add part (two of two) to $\mathcal{A}_\mathsf{e}$; overwrite cell-center velocity vectors $\boldsymbol{u}_c$ of adjacent cells

if fixed weir scheme $\in \{3, 4, 5\}$ **then**
  compute and overwrite cell-centered weir-velocities $\boldsymbol{u}_{cL(j)}$ and $\boldsymbol{u}_{cR(j)}$ with Algorithm (42)
end if
if $u_j > 0$ **then**
  compute far-field velocity $\hat{\boldsymbol{u}}_{cL(j)}$ with Algorithm (41)
  $u_{in_j} = \begin{cases} \hat{\boldsymbol{u}}_{cL(j)} \cdot \boldsymbol{n}_{wj}, & \text{fixed weir scheme } 4 \\ \hat{\boldsymbol{u}}_{cL(j)} \cdot \boldsymbol{n}_j, & \text{otherwise} \end{cases}$
  $\mathcal{A}_{\mathsf{i}j} = \frac{u_j}{2\Delta x_j}$
  $\mathcal{A}_{\mathsf{e}j} = \mathcal{A}_{\mathsf{e}j} - \frac{{u_{in}}_j^2}{2\Delta x_j}$
else
  as above by interchanging $L(j)$ and $R(j)$ and taking reversed orientation into account
end if

Remark 6.6.7. Starting from Bernoulli's equation to derive the weir subgrid model, it seems inconsistent not to include the far-field velocity for scheme 5.

The far-field velocity is based on the cell-centered velocity vector in the adjacent, upstream cell. The cell-centered velocity vectors are reconstructed from the face-normal velocity components with Algorithm (6). However, the upstream cell-centered velocity is reconstructed from a set of face-normal velocity components that includes the weir itself. Since we are seeking for a far-field velocity, we have to exclude the increased crest velocity u_j from the reconstruction. See Remark 6.6.2 in this respect. This is achieved by estimating an unperturbed velocity $\hat{u}_j$ as if no weir was present and using that velocity in the reconstruction instead. The unperturbed velocity is estimated by using continuity, i.e.

$$\hat{u}_j = \frac{h_{uj}}{\hat{h}_j} u_j, \tag{6.165}$$

where $\hat{h}_j$ is a typical water depth for the upstream cell ($L(j)$ if $u_j > 0$). The reconstruction of the upstream cell-centered velocity vector is then performed as shown in Algorithm (41). Note that the faces that are associated with a weir are identified with $iadv_j = 21$, see Algorithm (38), so $\hat{\mathcal{J}}(k)$ is the set of faces of cell k without the faces that are associated to a fixed weir.

For the advection downstream of the wear the cell-centered velocity vectors get the opposite treatment with Algorithm (42).

6.6.4 Supercritical discharge

Supercritical conditions are defined by

$$\zeta^*_{R(j)} \leq z_c + \frac{2}{3} E_j, \tag{6.172}$$

where we let superscript * indicate supercritical and stationary conditions. The water height at the crest is then according to Eqn. (6.159)

$$h^*_{u_j} = \frac{2}{3} E_j. \tag{6.173}$$

Algorithm 41 getucxucynoweirs: reconstruct a cell-centered velocity vector near a fixed wear without the weir itself

$$\hat{\mathcal{J}}(k) = \{j \in \mathcal{J}(k) | iadv_j \neq 21\} \tag{6.166}$$

$$\hat{h}_k = \begin{cases} \frac{1}{\sum_{j \in \hat{\mathcal{J}}(k)} w_{uj}} \sum_{j \in \mathcal{J}(k)} w_{uj} h_{uj}, & \hat{\mathcal{J}}(k) \neq \emptyset \\ 0, & \text{otherwise} \end{cases} \tag{6.167}$$

$$\begin{aligned} \boldsymbol{u}_{ck} = \frac{1}{b_{Ak}} (& \sum_{j \in \{l \in \hat{\mathcal{J}}(k) | 1_{l,k} = 1\}} \alpha_j \Delta x_j w_{uj} \boldsymbol{n}_j u_j + \\ & \sum_{j \in \{l \in \mathcal{J}(k) \setminus \hat{\mathcal{J}}(k) | 1_{l,k} = 1\}} \alpha_j \Delta x_j w_{uj} \boldsymbol{n}_j u_j \min(1, \frac{h_{uj}}{\hat{h}_k}) + \\ & \sum_{j \in \{l \in \hat{\mathcal{J}}(k) | 1_{l,k} = -1\}} (1 - \alpha_j) \Delta x_j w_{uj} \boldsymbol{n}_j u_j + \\ & \sum_{j \in \{l \in \mathcal{J}(k) \setminus \hat{\mathcal{J}}(k) | 1_{l,k} = -1\}} (1 - \alpha_j) \Delta x_j w_{uj} \boldsymbol{n}_j u_j \min(1, \frac{h_{uj}}{\hat{h}_k})) \end{aligned} \tag{6.168}$$

Algorithm 42 getucxucyweironly: reconstruct a cell-centered velocity vector near a fixed wear with the weir itself

$$\bar{\mathcal{J}}(k) = \{j \in \mathcal{J}(k) | iadv_j = 21\} \tag{6.169}$$

$$\bar{h}_k = \begin{cases} \frac{1}{\sum_{j\in\bar{\mathcal{J}}(k)} w_{uj}} \sum_{j\in\bar{\mathcal{J}}(k)} w_{uj} h_{uj}, & \bar{\mathcal{J}}(k) \neq \emptyset \\ 0, & \text{otherwise} \end{cases} \tag{6.170}$$

$$\begin{aligned} \boldsymbol{u}_{ck} \;=\; \frac{1}{b_{Ak}} (& \sum_{j\in\{l\in\bar{\mathcal{J}}(k)|1_{l,k}=1\}} \alpha_j \Delta x_j w_{uj} \boldsymbol{n}_j u_j + \\ & \sum_{j\in\{l\in\mathcal{J}(k)\setminus\bar{\mathcal{J}}(k)|1_{l,k}=1\}} \alpha_j \Delta x_j w_{uj} \boldsymbol{n}_j u_j \max(1, \frac{h_{uj}}{\bar{h}_k}) + \\ & \sum_{j\in\{l\in\bar{\mathcal{J}}(k)|1_{l,k}=-1\}} (1-\alpha_j) \Delta x_j w_{uj} \boldsymbol{n}_j u_j + \\ & \sum_{j\in\{l\in\mathcal{J}(k)\setminus\bar{\mathcal{J}}(k)|1_{l,k}=-1\}} (1-\alpha_j) \Delta x_j w_{uj} \boldsymbol{n}_j u_j \max(1, \frac{h_{uj}}{\bar{h}_k})) \end{aligned} \tag{6.171}$$

and the crest velocity according to Eqn. (6.161)

$$u_j^* = \sqrt{\frac{2}{3} g E_j}. \tag{6.174}$$

The face-based discharge under stationary and supercritical conditions is then by definition

$$q_j^* := A_{u_j}^* u_j^* = w_{uj} h_{u_j}^* u_j^* = w_{uj} \frac{2}{3} E_j \sqrt{\frac{2}{3} g E_j}, \tag{6.175}$$

where we have used Eqn. (6.19) and Algorithm (3).

7 Numerical schemes for three-dimensional flows

7.1 Artificial mixing due to sigma-coordinates

The fluxes of the transport equations consist of both advective and diffusive fluxes. In sigma co-ordinates the approximation of the advective fluxes does not introduce large truncation errors. Therefore in this section we consider only diffusive fluxes given by

$$F_i = D_H \frac{\partial c}{\partial x_i},\ i = 1, 2 \qquad F_3 = D_V \frac{\partial c}{\partial x_3} \tag{7.1}$$

where D_H, denotes the horizontal eddy diffusion coefficient and D_V denotes the vertical eddy diffusion coefficient.

It is difficult to find a numerical approximation that is stable and positive. Near steep bottom slopes or near tidal flats where the total depth becomes very small, truncations errors in the approximation of the horizontal diffusive fluxes in σ-co-ordinates are likely to become very large, similarly to the horizontal pressure gradient. Thus a complete transformation must be included. However, in that case numerical problems are encountered concerning accuracy, stability and monotonicity. In D-Flow FM a method is applied which gives a consistent, stable and monotonic approximation of the horizontal diffusion terms even when the hydrostatic consistency condition is violated. For details we refer the user to Stelling and Van Kester (1994)

7.1.1 A finite volume method for a σ-grid

Applying the Gauss theorem to the transport equation yields

$$\frac{\partial}{\partial t} \int_v c\, dv + \oint_s \boldsymbol{F} \cdot \boldsymbol{n}\, ds = 0 \tag{7.2}$$

Instead of transforming the transport equation to σ-co-ordinates, we generate a sigma grid by choosing a distribution of the vertical co-ordinate sigma. The vertical diffusive fluxes are straightforward to implement. The only difficulty is the approximation of the horizontal diffusive fluxes. To explain this method, it is sufficient to consider a simplified two-dimensional transport equation

$$\frac{\partial c\,(x, z, t)}{\partial t} - \frac{\partial}{\partial x} D_H \frac{\partial c\,(x, z, t)}{\partial x} = 0 \tag{7.3}$$

For this equation a finite volume method has to be constructed that meets the following requirements:

1 consistent approximation of the horizontal diffusive fluxes
2 fulfilment of the min-max principle
3 minimal artificial vertical diffusion

A non-linear approach is chosen which consists of the following steps.

◇ Step 1
First, diffusive fluxes $f_{i+1/2,l+1/2}, l = 0, ..., 2K$ (K is the $sigma$-layers number), are defined according to

$$\begin{cases} D_H \min\left(\Delta_m c, \Delta_n c\right) \frac{z_{i+1/2,l+1} - z_{i+1/2,l}}{x_{i+1} - x_i}, & \Delta_m c > 0 \wedge \Delta_n c > 0 \\ D_H \max\left(\Delta_m c, \Delta_n c\right) \frac{z_{i+1/2,l+1} - z_{i+1/2,l}}{x_{i+1} - x_i}, & \Delta_m c \leq 0 \wedge \Delta_n c \leq 0 \\ 0, & \Delta_m c \Delta_n c < 0 \end{cases} \tag{7.4}$$

The differences $\Delta_{m/n}c = \Delta_{m/n}c_{i+1/2,l+1/2}$ are given by

$$f_{i+1/2,l+1/2} = \left.\begin{array}{l} \Delta_m c_{i+1/2,l+1/2} = c_{i+1}\left(z_{i,m(l)} - c_{i,m(l)}\right) \\ \Delta_n c_{i+1/2,l+1/2} = c_{i+1,n(l)} - c_i\left(z_{i+1,n(l)}\right) \end{array}\right\} \quad l = 0, ..., 2K \tag{7.5}$$

where $c_i\left(z\right)$ is a simple linear interpolation formula given by

$$c_i\left(z\right) = \begin{cases} c_{i,1}, & z \leq z_{i,1} \\ \frac{z - z_{i,k}}{z_{i,k+1} - z_{i,k}} c_{i,k+1} + \frac{z_{i,k+1} - z}{z_{i,k+1} - z_{i,k}} c_{i,k}, & z_{i,k} < z \leq z_{i,k+1} \\ c_{i,K}, & z \geq z_{i,K} \end{cases} \tag{7.6}$$

◇ Step 2
In this step the diffusive fluxes are added to the appropriate control volumes according to

$$V_{i,k}^{\tau+1} c_{i,k}^{\tau+1} = V_{i,k}^{\tau} c_{i,k}^{\tau} - \Delta t \sum_{\forall l | m(l)=k} f_{i+1/2,l+1/2}^{\tau} + \Delta t \sum_{\forall l | n(l)=k} f_{i-1/2,l+1/2}^{\tau} \tag{7.7}$$

where τ is the time index, $t = \tau \Delta t$ and V^{τ} denotes the size of the control volume. The absence of advection implies $V^{\tau} = V^{\tau+1}$

7.1.2 Approximation of the pressure term

The horizontal gradients of the pressure must be approximated for the horizontal momentum equations. The pressure gradient must be computed along the same verticals as the horizontal concentration gradients. The pressure p in Cartesian co-ordinates is given by

$$p\left(x, z\right) = \int_z^{\zeta} \rho\left(x, z', t\right) g dz' \tag{7.8}$$

From the Leibnitz rule it follows that $\partial p / \partial x$ is given by

$$\frac{\partial}{\partial x} = \frac{\partial}{\partial x} \int_z^{\zeta(x)} \rho\left(x, z'\right) g dz' = \int_z^{\zeta(x)} g \frac{\partial}{\partial x} \rho\left(x, z'\right) dz' + g\rho\left(\zeta\right) \frac{\partial \zeta}{\partial x} \tag{7.9}$$

The relation between the density p and the salinity s and temperature T is given by the equation of state, namely $\rho = \rho\left(s(x,t), T(x,t)\right)$. The integral in Equation 7.9 is replaced by a summation over the intervals which are in the water column above the velocity point with vertical co-ordinate z.

$$\begin{array}{rl} \left(\frac{\partial p}{\partial x}\right)\left(x_{i+1/2,z}\right) = & g \sum_{l=K+1}^{2K+1} \left[\left(\frac{\partial \rho}{\partial s}\right) \frac{f(s)}{D_H} + \left(\frac{\partial \rho}{\partial T}\right) \frac{f(T)}{D_H}\right]_{i+1/2,l+1/2} \\ & + g \frac{z_{i+1/2,k+1} - z}{z_{i+1/2,k+1} - z_{i+1/2,k}} \left[\left(\frac{\partial \rho}{\partial s}\right) \frac{f(s)}{D_H} + \left(\frac{\partial \rho}{\partial T}\right) \frac{f(T)}{D_H}\right]_{i+1/2,k+1/2} \\ & + g\rho \frac{\zeta_{i+1} - \zeta i}{\Delta x} \end{array} \tag{7.10}$$

where $k = \max\left(l | z_{i+1/2,l} < z\right)$

8 Parallelization

This chapter elaborates on the parallelization of D-Flow FM. The sequential time loop is described in Section 6.2. This chapter emphasises on the modifications needed for parallelization.

8.1 Parallel implementation

The goal of parallelization of D-Flow FM is twofold. We aim for faster computations on shared- or distributed-memory machines and the ability to model problems that do not fit on a single machine. To this end we decompose the computational domain into subdomains and apply the "single program, multiple data" (SPDM) technique for parallelization. Since we apply SPDM, we want each subdomain (process) to be as autonomous as can be and require that

- ◇ each subdomain has its own unique computational mesh,
- ◇ the subdomain interfaces act as boundaries where data is communicated,
- ◇ only primitive variables u and ζ are communicated,
- ◇ the parallel and sequential algorithm yield the same results, except for round-off errors that is,
- ◇ the modeling in all subdomains is identical, has the same time-step, et cetera. Note this this requirement compromises our aim for autonomous subdomain modeling.

8.1.1 Ghost cells

Keeping our design choices in mind, it is apparent from Eqns. (6.13) and (6.71) that during a time-step we need to compute advection, diffusion and the water-level gradient in the momentum equation *anywhere* in the subdomain. Similarly, we need to compute the discharge divergence in the continuity equation, Eqn. (6.73) *anywhere* in our subdomains. The stencil used for computing momentum advection and diffusion is depicted in Fig. 8.12.

It will be clear that the stencil can not be applied near the subdomain interfaces. Since we choose not to modify the stencil as explained in the foregoing, the subdomains need to be augmented with *ghost cells* that only serve to compute the time-step update for the "internal" water-levels and velocities. No valid velocity- and water-level update are computed for the "external", ghost water-levels and velocities. Instead, values at the next time-level are copied from the corresponding neighboring subdomains, where valid time-step updates were computed, see Fig. 8.2.

The question remains how many ghost cells need to be supplied. The stencil for the momentum advection and diffusion is depicted in Fig. 8.12. To be able to count the number of cells in the stencil, we firstly define the level of a neighboring cell. Cells adjacent to a face are level 1. Their neighboring cells, i.e. cells that share at least one common face, are level 2, et cetera, see Fig. 8.12. We say that a cell is in the stencil, if at least one of it's face-normal velocity components is required in the stencil, since in D-Flow FM a face-normal velocity can only exist if both its neighboring cells exist.

It will not be hard to see that one level of ghost cells suffices for the water-level gradients in the momentum equation and divergence in the continuity equation. The spatial discretization of momentum advection and diffusion will not be explained in detail here, but it is important to understand that advection and diffusion are in fact computed at cell centers, based on reconstructed cell-centered data, and interpolated back to the faces. So we have

- ◇ one level of neighbors for the interpolation from cell-centered to face-normal advection and diffusion,
- ◇ one additional level for a higher order cell-centered (collocated) discretization, and

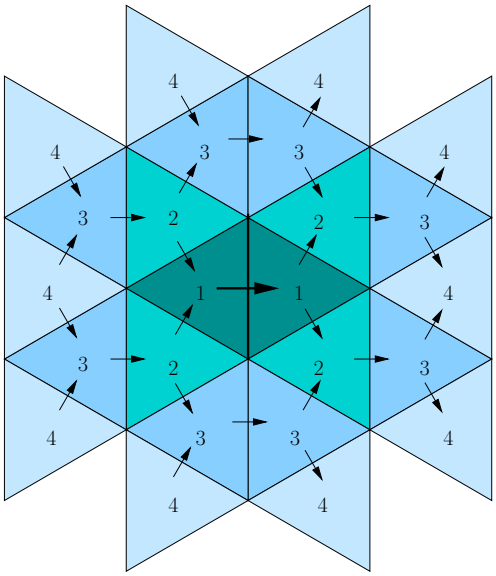


Figure 8.1: *Stencil for momentum advection and diffusion; the numbers indicate the level of the neighboring cells*

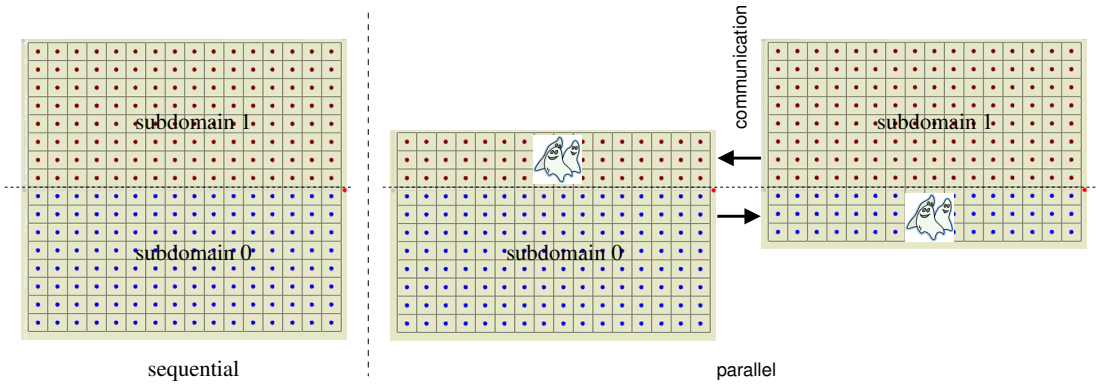


Figure 8.2: *Ghost cells*

Table 8.1: *METIS settings*

parameter/option	value	meaning
routine	METIS_PARTMESHDUAL	mesh partitioning method
PTYPE	PTYPE_KWAY	multilevel k-way partitioning algorithm
NITER	100	Number of iterations for the refinement algorithms at each stage of the uncoarsening process
CONTIG	0 or 1	enforce contiguous subdomains (1) or not (0)

◇ one additional level for the reconstruction of the cell-centered velocity vector from the edge-normal data.

This sums up to four levels of neighbors, as can be seen in Fig. 8.12. Consequently, four levels of ghosts cells are required for momentum advection and diffusion.

The ghost level of cell k is called $g_s(k)$ and of face j $g_u(j)$. They are related by

$$g_u(j) = \begin{cases} \min(g_s(L(j)), g_s(R(j))), & \text{face } j \text{ is a ghost face, not on the subdomain interface,} \\ \max(g_s(L(j)), g_s(R(j))), & \text{face } j \text{ is a ghost face, on the subdomain interface,} \\ 0 & \text{face } j \text{ is not a ghost face.} \end{cases} \tag{8.1}$$

If face j is on the subdomain interface, it can only be a ghost face of subdomain id if the neighboring ghost cell has a lower subdomain number than id, since we say that it is owned by the subdomain with the lowest number, explained hereafter, see Eqn. (8.2). If face j is not on the subdomain interface, it can only be a ghost face of subdomain id if both adjacent cells have subdomain numbers other than id.

8.1.2 Mesh partitioning

The mesh is partitioned with the METIS software package,see Karypis (2011). METIS produces a cell coloring of the unpartitioned mesh, that we will refer to as the cell subdomain number $id_s(k)$, where k is the cell number. The non-default METIS settings employed are listed in Table 8.1. We use METIS's multilevel k-way partitioning algorithm, since it enables us to enforce contiguous subdomains. We have observed that this comes at the cost of a reduced homogeneous distribution of cells over the subdomains, however.

Any cell in the unpartitioned mesh uniquely belongs to a subdomain, so the water-level unknowns can uniquely be assigned a subdomain number. The face-normal velocity unknowns, on the other hand, can not, since the velocities on the subdomain interfaces can belong to either of the two adjacent subdomains. We choose to uniquely assign a subdomain number $id_u(j)$ to face j by taking the minimum subdomain number of its two adjacent cells:

$$id_u(j) = \min\left(id_s(L(j)), id_s(R(j))\right). \tag{8.2}$$

Data will be communicated from the subdomain that owns the data to the subdomains that require the data.

With the cell coloring available, the subdomain meshes are augmented with four layers of ghost cells and written to file.

Remark 8.1.1. Level 5 ghost cells are not included in the subdomain meshes, except when all its neighboring cells have level 4 or lower, or are non-ghost cells. In that case all faces of

the level 5 ghost cell are present in the subdomain mesh. Since in D-Flow FM cells with all its faces being defined in the mesh can not be disregarded, the level 5 ghost cells itself are included in the subdomain mesh.

The cell coloring itself is not written to file either. Instead, the subdomain interfaces are saved as polygons to file and read by all parallel processes.

8.1.3 Communication

The whole domain mesh was partitioned as described in the foregoing. It is not used during the parallel computations and we will only consider the subdomains from now on.

During the computations we need to update the ghost values from the other subdomains. However, we do not need to communicate all variables at all instances in the time step. It depends on the operator under consideration. To this end, three sets of ghost values are defined:

$$\mathcal{G}_s = \{k : g_s(k) = 1\}, \tag{8.3}$$
$$\mathcal{G}_{s_{all}} = \{k : 1 \leq g_s(k) \leq N+1\}, \tag{8.4}$$
$$\mathcal{G}_u = \{j : 1 \leq g_u(j) \leq N+1\}, \tag{8.5}$$

where $N = 4$ is the number of ghost levels. It may come as a surprise that we include ghost levels up to $N+1$, however see Remark 8.1.1 in this respect. $\mathcal{G}_s$ refers to an update of the first level of ghost water-levels, needed in the continuity equation, $\mathcal{G}_{s_{all}}$ to all ghost water-levels and $\mathcal{G}_u$ to all face-normal velocity components respectively.

Communication information is not stored to any subdomain specific file. Instead, coordinates of the ghost cells and ghost faces are communicated with the other subdomains in the initialization phase of the computations and, doing so, *send* lists $\mathcal{S}_s$, $\mathcal{S}_{s_{all}}$ and $\mathcal{S}_u$ are constructed, see Algorithm (43).

Remark 8.1.2. We have a one-to-one mapping of task (or ranks) to subdomain number, i.e. task i will correspond to subdomain i, $i \in \{0, N-1\}$, with N being the number of subdomains.

Algorithm 43 partition_init: initialize the parallel communication

generate subdomain numbers based on the partitioning polygon (cells with subdomain numbers other than the own subdomain number will correspond to ghost cells)
set the ghost levels
make the ghost lists $\mathcal{G}_s$, $\mathcal{G}_{s_{all}}$ and $\mathcal{G}_u$
make the send lists $\mathcal{S}_s$, $\mathcal{S}_{s_{all}}$ and $\mathcal{S}_u$

The update of the ghost water-levels is performed by MPI communication, see Algorithm (44). The other ghost types are updated similarly.

Algorithm 44 update_ghosts: update the ghost water-levels by means of MPI communication

non-blocking MPI-send ζ_k, $k \in \mathcal{S}_s$ to other subdomains
MPI-receive ζ_k, $k \in \mathcal{G}_s$ from other subdomains
wait for send to terminate

8.1.4 Parallel computations

In the parallel run, each task will

- ◇ prepare the subdomain model, i.e.
  - ▫ read the subdomain mesh,
  - ▫ read the boundary conditions,
  - ▫ read external forcings,
  - ▫ et cetera,
- ◇ initialize the parallel communication, Algorithm (43),
- ◇ perform the time stepping, as in Algorithm (20),
- ◇ update ghost values during the time stepping,
- ◇ output flow variables.

Note that the boundary conditions, external forcing files, et cetera are shared by the subdomains. In fact, they are just the sequential files and do not need to be partitioned. Only the mesh and the model definition file need to be partitioned. The partitioned model definition files will contain references to the subdomain mesh and the partitioning polygon file, all other information equals its sequential counterpart.

The parallel time-step is shown in Algorithm (45). The parallel extension of Algorithm (19) is trivial and listed in Algorithm (46). The parallel solver for the water-level equation Algorithm (21) is described in the next section.

Remark 8.1.3. It is sufficient for the parallel water-level solver to update only level-1 ghost water-levels $\mathcal{G}_s$. It is therefore necessary that all ghost water-levels $\mathcal{G}_{s_{all}}$ are updated right after the solve, as shown in Algorithm (45).

The parallel extensions of the following will remain unmentioned:

- ◇ discharge boundary conditions,
- ◇ 1D-network branches,
- ◇ cross sections and observation stations (for post-processing).

8.1.5 Parallel Krylov solver

The unknown water-levels $k \in \mathcal{K}$ in Eqn. (6.81) are solved in the same manner as in case of the sequential computations, Algorithm (21), expecpt for the solver itself, which now is a parallel Krylov solver. We have two solvers available:

- ◇ the parallelized version of the sequential algorithm (Algorithm (22)), and
- ◇ a solver from the Portable, Extensible Toolkit for Scientific Computation (PETSc), Balay *et al.* (2013).

8.1.5.1 parallelized Krylov solver

The (reduced) global system to be solved has the form of

$$\begin{pmatrix} A^{[0,0]} & \cdots & A^{[0,N-1]} \\ \vdots & \ddots & \vdots \\ A^{[N-1,0]} & \cdots & A^{[N-1,N-1]} \end{pmatrix} \begin{pmatrix} \boldsymbol{s}^{[0]} \\ \vdots \\ \boldsymbol{s}^{[N-1]} \end{pmatrix} = \begin{pmatrix} \boldsymbol{d}^{[0]} \\ \vdots \\ \boldsymbol{d}^{[N-1]} \end{pmatrix}, \tag{8.10}$$

where the superscript $[id]$ indicates the domain number. A matrix-vector multiplication can be

Algorithm 45 parallel step_reduce: perform a time step; parallel-specific statements are outlined

while first iteration or repeat time-step (type 1) **do**
$t^{n+1} = t^n + \Delta t$
compute ${f_u}_j^n$ and ${r_u}_j^n$ with Algorithm (12)
while first iteration or repeat time-step (type 2) **do**
compute the matrix entries B_k^n, C_j^n and right-hand side d_k^n in the water-level equation with Algorithm (13)
determine the set of water-levels that need to be solved, Algorithm (15)
$i = 0$
$\zeta_k^{n+1(0)} = \zeta_k^n$
while $\left(\max_k \left|\zeta_k^{n+1(i)} - \zeta_k^{n+1(i-1)}\right| > \epsilon \ \wedge$ not repeat time-step$\right) \ \vee \ i = 0$ **do**
$i = i + 1$
compute the matrix entries ${B_r}_k^n$, ${C_r}_j^n$ and right-hand side ${d_r}_k^n$ in the water-level equation with Algorithm (16)
parallel solve the unknown water-levels and obtain $\zeta_k^{n+1(i+1)}$, see Section 8.1.5
update all ghost water-levels $\zeta_k^{n+1(i-1)}$, $k \in \mathcal{G}_{s_{all}}$
check positivity of water height with Algorithm (17) and repeat time-step if necessary with modified Δt (type 1) or ${h_u}_j^n$ (type 2, default)
reduce 'repeat time-step'
if not repeat time-step **then**
compute water-column volume $V_k^{n+1(i+1)}$ and wet bed area $A_k^{n+1(i+1)}$ with Algorithm (18)
reduce $\max_k \left|\zeta_k^{n+1(i)} - \zeta_k^{n+1(i-1)}\right|$
end if
end while
end while
end while
$\zeta_k^{n+1} = \zeta_k^{n+1(i+1)}$
compute velocities u_j^{n+1}, update ghost velocities u_j^{n+1}, $k \in \mathcal{G}_u$ and compute discharges q_j^{n+1} and ${q_a}_j^{n+1}$ at the next time level, Algorithm (46)

Algorithm 46 parallel u1q1: update velocity u_j^{n+1}, update ghost velocities u_j^{n+1}, and compute discharges q_j^{n+1} and $q_{a_j}^{n+1}$; parallel-specific statements are outlined

if $h_{u_j}^{\ n} > 0$ **then**
  $u_j^{n+1} = -f_{u_j}^{\ n}(\zeta_{R(j)}^{n+1} - \zeta_{L(j)}^{n+1}) + r_{u_j}^{\ n}$
else
  $u_j^{n+1} = 0$
end if
update ghost velocities u_j^{n+1}, $j \in \mathcal{G}_u$
if $h_{u_j}^{\ n} > 0$ **then**

$$q_j^{n+1} = A_{u_j}^{\ n}\left(\theta_j u_j^{n+1} + (1-\theta_j)u_j^n\right) \tag{8.6}$$
$$q_{a_j}^{\ n+1} = A_{u_j}^{\ n} u^{n+1} \tag{8.7}$$

else

$$q_j^{n+1} = 0 \tag{8.8}$$
$$q_{a_j}^{\ n+1} = 0 \tag{8.9}$$

end if

written as

$$\begin{pmatrix} \vdots \\ A^{[id,id]}\boldsymbol{s}^{[id]} + \sum_{jd \neq id} A^{[id,jd]}\boldsymbol{s}^{[jd]} \\ \vdots \end{pmatrix},$$

where the diagonal diagonal contribution is computed as in the sequential case, see Eqn. (6.92), however for the internal unknowns only

$$A^{[id,id]}\boldsymbol{s}^{[id]} = \begin{pmatrix} \vdots \\ B_{r\,k}^{[id]}\ s_k^{[id]} + \sum_{j \in \mathcal{J}^{[id]}(k) \setminus G_s^{[id]}} C_{r\,j}^{[id]}\ s_{O(k,j)}^{[id]} \\ \vdots \end{pmatrix} \tag{8.11}$$

and the off-diagonal contributions are computed by means of the ghost values $\mathcal{G}_s^{[id]}$

$$\sum_{jd \neq id} A^{[id,jd]}\boldsymbol{s}^{[jd]} = \begin{pmatrix} \vdots \\ \sum_{j \in \mathcal{J}^{[id]}(k) \cap G_s^{[id]}} C_{r\,j}^{[id]}\ s_{O(k,j)}^{[id]} \\ \vdots \end{pmatrix}, \tag{8.12}$$

provided that the ghost values $G_s^{[id]}$ are up-to-date .

Remark 8.1.4. Eqn. (8.10) shows that water-level unknowns in $\mathcal{S}_s$ are required for the global matrix-vector multiplication. For that reason, they are disregarded in the Maximum Degree algorithm and will never be eliminated from the solution vector $\boldsymbol{s}$.

The system is solved by a parallelized preconditioned Conjugate Gradient method of Algorithm (22), as shown in Algorithm (47), where we consider one subdomain id only and have dropped the superscript $[id]$. The parallel extensions are trivial, except for the preconditioner P that is. We apply a non-overlapping Additive Schwarz MILU factorization and preconditioning $P\boldsymbol{z}_r^{(i+1)} = \boldsymbol{r}^{(i+1)}$ can then be expressed as

$$P^{[id]}\boldsymbol{z}_r^{(i+1)[id]} = \boldsymbol{r}^{[id]} - \sum_{jd \neq id} A^{[id,jd]}\boldsymbol{z}_r^{(i)[id]}, \tag{8.13}$$

where $P^{[id]}$ approximates $A^{[id,id]}$. We use a MILU factorization available from SPARSKIT, Saad (1994).

8.1.5.2 PETSc solver

As an alternative to the parallelized sequential Krylov solver, as explained in the foregoing, we can apply a solver from the Portable, Extensible Toolkit for Scientific Computation (PETSc), Balay *et al.* (2013). We use default settings.

8.2 Test-cases

In this section two test-cases are considered. To assess the scalability of the parallel implementation, the computing time is measured for decompositions with varying number of subdomains.

We measure the wall-clock times spent in the time-steps. This does not include file output for post-processing. At prescribed modeling-time instances, computing times are measured and summed (in time) by each subdomain. The *maximum* computing times over all the subdomains are used to determine a time-step average during a measurement interval, i.e.

$$T_{\text{time-step}_k} = \frac{\max_d \sum_{i=1}^{n_k} \Delta T_i^d - \max_d \sum_{i=1}^{n_k-1} \Delta T_i^d}{n_k - n_{k-1}}, \tag{8.14}$$

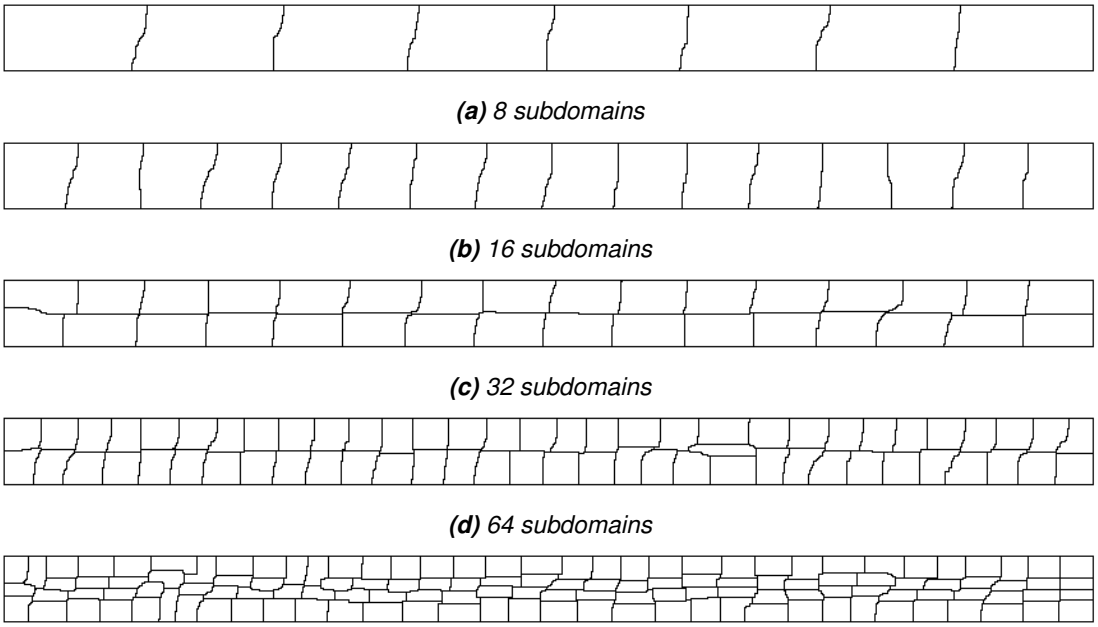
where ΔT_i^d is the wall-clock computing time of time-step i and subdomain d, k is a measurement index and n is the number of time steps. We will refer to this time as the time-step averaged wall-clock time of a "time-step". The time-step wall-clock time is further divided into

- ◇ $\text{MPI}_{\text{non-sol}}$: MPI-communication time not related to the Krylov solver. These are the update of the ghost-values $\mathcal{G}_{s_{all}}$ and $\mathcal{G}_u$, respectively and the reduction of the variables as indicated in Algorithm (45),
- ◇ solver: total Krylov solve time. This is the time spent by the Krylov solver, including MPI-communication,
- ◇ MPI_{sol}: MPI-communication time in the Krylov solver. Unfortunately, no such times are available for the PETSc solver,
- ◇ #Krylov iterations: the time-step averaged number of iterations needed for the Krylov solver to converge.

We expect that the non-communication times will show nearly linear scalability and foresee that the communication times behave much worse. Furthermore, if the precondition becomes less effective when the number of subdomains increases, the number of iterations will increase.

Algorithm 47 conjugategradient_MPI: solve water-level equation with a preconditioned Conjugate Gradient method; parallel specific statements are outlined

compute preconditioner P
update ghost values ζ_k, $k \in \mathcal{G}_s$
compute initial residual $\boldsymbol{r}^{(0)} = \boldsymbol{d} - A\boldsymbol{s}^{(0)}$
compute maximum error $\epsilon = \|\boldsymbol{r}^{(0)}\|_\infty$
update ghost residuals r_k, $k \in \mathcal{G}_s$
apply preconditioner $P\boldsymbol{z}_r^{(0)} = \boldsymbol{r}^{(0)}$
set $\boldsymbol{p}^{(0)} = \boldsymbol{z}_r^{(0)}$
compute inner product $(\boldsymbol{r}^{(0)}, \boldsymbol{z}_r^{(0)})$
reduce inner product $(\boldsymbol{r}^{(0)}, \boldsymbol{z}_r^{(0)})$
reduce maximum error $\epsilon = \|\boldsymbol{r}^{(0)}\|_\infty$
$i = 0$
while $\epsilon >$ tol **do**
  update ghost values p_k, $k \in \mathcal{G}_s$
  compute $A\boldsymbol{p}^{(i)}$
  compute $(\boldsymbol{p}^{(i)}, A\boldsymbol{p}^{(i)})$
  reduce $(\boldsymbol{p}^{(i)}, A\boldsymbol{p}^{(i)})$
  $\alpha^{(i)} = \frac{(\boldsymbol{r}^{(i)}, \boldsymbol{z}_r^{(i)})}{(\boldsymbol{p}^{(i)}, A\boldsymbol{p}^{(i)})}$
  $\boldsymbol{s}^{(i+1)} = \boldsymbol{s}^{(i)} + \alpha^{(i)}\boldsymbol{p}^{(i)}$
  $\boldsymbol{r}^{(i+1)} = \boldsymbol{r}^{(i)} - \alpha^{(i)}A\boldsymbol{p}^{(i)}$
  compute maximum error $\epsilon = \|\boldsymbol{r}^{(i+1)}\|_\infty$ $\epsilon = \|\boldsymbol{r}^{(0)}\|_\infty$
  reduce maximum error $\epsilon = \|\boldsymbol{r}^{(0)}\|_\infty$
  update ghost values $\boldsymbol{r}^{(i+1)}$, $k \in \mathcal{G}_s$
  apply preconditioner $P\boldsymbol{z}_r^{(i+1)} = \boldsymbol{r}^{(i+1)}$
  if $\epsilon >$ tol **then**
    compute $(\boldsymbol{r}^{(i+1)}, \boldsymbol{z}_r^{(i+1)})$
    reduce $(\boldsymbol{r}^{(i+1)}, \boldsymbol{z}_r^{(i+1)})$
    $\beta^{(i+1)} = \frac{(\boldsymbol{r}^{(i+1)}, \boldsymbol{z}_r^{(i+1)})}{(\boldsymbol{r}^{(i)}, \boldsymbol{z}_r^{(i)})}$
    $\boldsymbol{p}^{(i+1)} = \boldsymbol{z}_r^{(i+1)} + \beta^{(i+1)}\boldsymbol{p}^{(i)}$
    $i = i + 1$
  end if
end while



***(a)** 8 subdomains*

***(b)** 16 subdomains*

***(c)** 32 subdomains*

***(d)** 64 subdomains*

***(e)** 128 subdomains*

***Figure 8.3:** Partitioning of the schematic Waal model with METIS*

The speed-up factor f can now be defined as:

$$f_k(N) = \frac{T_{\text{time-step}_k}\big|_N}{T_{\text{time-step}_k}\big|_{ref}} N, \tag{8.15}$$

where N is the number of subdomains and ref refers to a reference domain decomposition, for which we take the decomposition with the smallest number of subdomains available. Note that we do not compare with the single-domain, sequential simulations.

The simulations were conducted on the Deltares h4 cluster and the Lisa cluster at SURFsara, see Lisa. For all our simulations, we took four cores per node.

Remark 8.2.1. Wall-clock times on Lisa were limited to 2.5 hours, so, depending on the number of subdomains, some simulations advanced further in modeling time than others.

For our comparison, we will always compare time-step averaged computing times at the same modeling times.

8.2.1 Schematic Waal model

The first test-case under considerations is the schematic Waal model, see Yossef and Zagonjoli (2010). The model has a rectangular domain of length 30 km and width 1800 m. It has a deep center section of width 600 m and bottom levels varying from 0.795 (left) to -2.205 m (right). The shallow outer part has a bottom level varying from 6.988 (left) to 3.988 m (right).

The mesh size in the deep, center part is 2×2 m^2 and in the shallow outer part 2×4 m^2. The total number of cells is $9\,000\,000$. The maximum time step is 0.45 sec. The domain is decomposed in 8, 16, 32, 64 and 128 subdomains, respectively. The partitioning is depicted in Fig. 8.3.

Table 8.2: *time-step averaged wall-clock times of the Schematic Waal model; Lisa; note: MPI communication times are not measured for the PETSc solver*

#dmns	t [h]	time step [s]	$MPI_{non\text{-}sol}$ [s]	solver [s]	MPI_{sol} [s]	#Krylov-iters
8	0.33	3.24200	0.07037	1.29400	0.00000	15.06000
	0.65					
	1.25					
	2.57					
	3.00					
16	0.33	1.64950	0.03644	0.64200	0.00000	16.01500
	0.65	1.62100	0.02498	0.62600	0.00000	15.00000
	1.25					
	2.57					
	3.00					
32	0.33	0.87400	0.03441	0.34975	0.00000	16.84500
	0.65	0.87000	0.03687	0.34800	0.00000	16.00000
	1.25	0.84851	0.03394	0.32426	0.00000	14.14356
	2.57					
	3.00					
64	0.33	0.44050	0.01904	0.18345	0.00000	17.22000
	0.65	0.44950	0.01983	0.17100	0.00000	16.00000
	1.25	0.42673	0.01962	0.16139	0.00000	14.25248
	2.57	0.41782	0.01906	0.15347	0.00000	13.20792
	3.00					
128	0.33	0.23870	0.01792	0.09415	0.00000	17.03000
	0.65	0.22450	0.01344	0.09075	0.00000	16.00000
	1.25	0.21733	0.01334	0.08361	0.00000	14.17327
	2.57	0.21436	0.01332	0.08069	0.00000	13.09901
	3.00	0.21733	0.01352	0.08020	0.00000	13.00000

Timing results on the SURFsara Lisa cluster are presented in Table 8.2 and the corresponding speed-up factor in Fig. 8.4. The results on the Deltares h4 cluster are shown in Table 8.3 and Fig. 8.5 respectively. Recall that the wall-clock computing time on Lisa was limited to 2.5 h, see Remark 8.2.1.

The results show that the speed-up factor with 128 subdomains is 108.66 on the Lisa cluster and 85.1 on the Deltares h4 cluster. This is a factor of 0.84, respectively 0.67 away from their theoretical maximum. The reduced scaling on the h4 may be attributed to the poorer scaling of the Krylov solver on the h4, due to communication overhead. It is interesting to see that the number of iterations of the Krylov solver does not increase significantly when the number of subdomain is increased. We do therefore not expect the preconditioner to have lost its effectiveness. On the other hand, a more advanced preconditioner should reduce the number of iterations in all cases and consequently the communication overhead, especially for large numbers of subdomains.

8.2.2 esk-model

Table 8.3: *time-step averaged wall-clock times of the Schematic Waal model; h4; note: MPI communication times are not measured for the PETSc solver*

#dmns	t [h]	time step [s]	$MPI_{non\text{-}sol}$ [s]	solver [s]	MPI_{sol} [s]	#Krylov-iters
8	0.33	3.74800	0.05160	1.31350	0.00000	15.06000
	0.65	3.74000	0.05333	1.31050	0.00000	15.00000
	1.25	3.62376	0.05163	1.19802	0.00000	13.00000
	2.57	3.56436	0.05029	1.13861	0.00000	12.00990
	3.00	3.56931	0.05019	1.13861	0.00000	12.06436
16	0.33	1.91550	0.04459	0.69650	0.00000	16.01500
	0.65	1.88450	0.04294	0.66700	0.00000	15.00000
	1.25	1.84158	0.04365	0.61881	0.00000	13.43564
	2.57	1.81188	0.04255	0.59406	0.00000	12.63861
	3.00	1.79703	0.04414	0.57426	0.00000	12.03960
32	0.33	1.01650	0.05025	0.39900	0.00000	16.84500
	0.65	1.01300	0.06046	0.38300	0.00000	16.00000
	1.25	0.96535	0.04963	0.34703	0.00000	14.10396
	2.57	0.95050	0.04990	0.33020	0.00000	13.00990
	3.00	0.95050	0.04960	0.33416	0.00000	13.00000
64	0.33	0.56200	0.03892	0.23580	0.00000	17.41000
	0.65	0.54450	0.03161	0.22550	0.00000	16.61000
	1.25	0.53465	0.03145	0.21634	0.00000	14.88614
	2.57	0.50990	0.03226	0.19158	0.00000	13.14851
	3.00	0.51485	0.03446	0.19307	0.00000	13.20297
128	0.33	0.35225	0.03553	0.17490	0.00000	17.03000
	0.65	0.34550	0.03375	0.17005	0.00000	16.00000
	1.25	0.32228	0.03249	0.14703	0.00000	14.18317
	2.57	0.30941	0.03199	0.13465	0.00000	13.04950
	3.00	0.31040	0.03188	0.13812	0.00000	13.0099

Table 8.4: *time-step averaged wall-clock times of the Schematic Waal model; SDSC's Gordon; note: MPI communication times are not measured for the PETSc solver*

#dmns	t [h]	time step [s]	$MPI_{non\text{-}sol}$ [s]	solver [s]	MPI_{sol} [s]	#Krylov-iters
8	0.33	2.85100	0.03012	1.15250	0.00000	15.06000
	0.65	2.85000	0.03273	1.14950	0.00000	15.00000
	1.25	2.73762	0.02946	1.04455	0.00000	13.00000
	2.57	2.68317	0.03151	0.99505	0.00000	12.00000
	3.00	2.68812	0.03118	0.99505	0.00000	12.03960
16	0.33	1.49700	0.01458	0.63950	0.00000	16.00500
	0.65	1.47000	0.01882	0.61200	0.00000	15.00000
	1.25	1.41584	0.01454	0.56238	0.00000	13.20792
	2.57	1.39604	0.01439	0.54455	0.00000	12.49505
	3.00	1.38614	0.01494	0.52970	0.00000	12.04455
32	0.33	0.75100	0.02350	0.32120	0.00000	16.84500
	0.65	0.73850	0.02326	0.31000	0.00000	16.00000
	1.25	0.71535	0.02308	0.28564	0.00000	14.11386
	2.57	0.69802	0.02249	0.27129	0.00000	13.00000
	3.00	0.70297	0.02239	0.27129	0.00000	13.00000
64	0.33	0.38245	0.01719	0.16115	0.00000	17.41000
	0.65	0.37900	0.01764	0.15610	0.00000	16.59500
	1.25	0.35941	0.01667	0.14059	0.00000	14.90594
	2.57	0.36386	0.02118	0.13960	0.00000	13.28218
	3.00	0.34851	0.01580	0.13020	0.00000	13.22277
128	0.33	0.19340	0.01688	0.07790	0.00000	17.03000
	0.65	0.18550	0.01454	0.07245	0.00000	16.00000
	1.25	0.18119	0.01455	0.06812	0.00000	14.16832
	2.57	0.17723	0.01499	0.06436	0.00000	13.05446
	3.00	0.17673	0.01435	0.06337	0.00000	13.00000

Table 8.5: *time-step averaged wall-clock times of the Schematic Waal model; SDSC's Gordon; CG+MILUD*

#dmns	t [h]	time step [s]	$MPI_{non\text{-}sol}$ [s]	solver [s]	MPI_{sol} [s]	#Krylov-iters
8	0.33	5.73000	0.02977	4.04400	0.06205	22.86000
	0.65	5.64500	0.03029	3.96000	0.07000	22.24500
	1.25	5.44554	0.02777	3.76238	0.03337	20.89109
	2.57					
	3.00					
16	0.33	3.13550	0.02596	2.27150	0.05955	24.54500
	0.65	3.02000	0.02771	2.16000	0.06665	23.00000
	1.25	2.89604	0.02547	2.03465	0.06297	21.40594
	2.57					
	3.00					
32	0.33	1.65000	0.02941	1.21550	0.08840	25.00000
	0.65	1.61900	0.03393	1.18350	0.08590	24.00000
	1.25	1.53960	0.02714	1.10396	0.07139	22.19802
	2.57					
	3.00					
64	0.33	0.88850	0.02410	0.66200	0.07745	26.20000
	0.65	0.85700	0.02364	0.62950	0.06110	24.99000
	1.25	0.82970	0.01814	0.60792	0.06282	23.75248
	2.57					
	3.00					
128	0.33	0.46550	0.01583	0.35070	0.06740	25.48500
	0.65	0.44900	0.01584	0.33450	0.06505	24.01500
	1.25	0.43465	0.01655	0.31980	0.05683	22.79703
	2.57	0.42327	0.01655	0.30891	0.05990	21.89604
	3.00	0.42079	0.01568	0.30446	0.05743	21.77228

Table 8.6: *time-step averaged wall-clock times of the 'esk-model'; Lisa; note: MPI communication times are not measured for the PETSc solver*

#dmns	t [h]	time step [s]	$MPI_{non\text{-}sol}$ [s]	solver [s]	MPI_{sol} [s]	#Krylov-iters
4	0.35	0.33152	0.00314	0.09406	0.00000	3.11984
	0.45					
	0.63					
	0.75					
	0.85					
	1.10					
8	0.35	0.16979	0.00355	0.05513	0.00000	3.30041
	0.45	0.17875	0.00451	0.05752	0.00000	4.06024
	0.63					
	0.75					
	0.85					
	1.10					
16	0.35	0.09284	0.00271	0.03237	0.00000	3.09976
	0.45	0.10073	0.00637	0.03337	0.00000	3.86932
	0.63	0.10140	0.00341	0.03565	0.00000	3.75257
	0.75					
	0.85					
	1.10					
32	0.35	0.04625	0.00171	0.01493	0.00000	3.36052
	0.45	0.04792	0.00200	0.01844	0.00000	4.34007
	0.63	0.04908	0.00202	0.01608	0.00000	3.97418
	0.75	0.04865	0.00209	0.01487	0.00000	3.09526
	0.85					
	1.10					
64	0.35	0.02323	0.00122	0.00999	0.00000	3.22176
	0.45	0.02508	0.00148	0.01139	0.00000	4.30026
	0.63	0.02567	0.00111	0.00915	0.00000	3.74654
	0.75	0.02543	0.00120	0.00894	0.00000	3.00078
	0.85	0.02545	0.00182	0.00846	0.00000	2.99741
	1.10					
128	0.35	0.01341	0.00087	0.00681	0.00000	3.89344
	0.45	0.01514	0.00119	0.00782	0.00000	3.00000
	0.63	0.01489	0.00125	0.00803	0.00000	4.00273
	0.75	0.01496	0.00101	0.00614	0.00000	3.06459
	0.85	0.01415	0.00090	0.00566	0.00000	3.00109
	1.10	0.01414	0.00100	0.00554	0.00000	3.00715

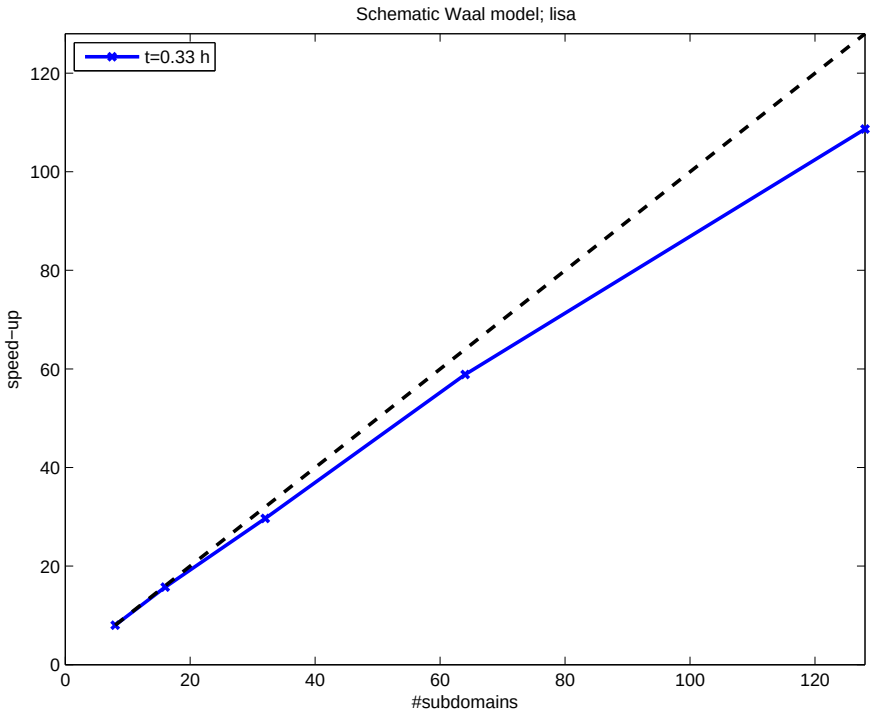


Figure 8.4: *Speed-up of the schematic Waal model; Lisa*

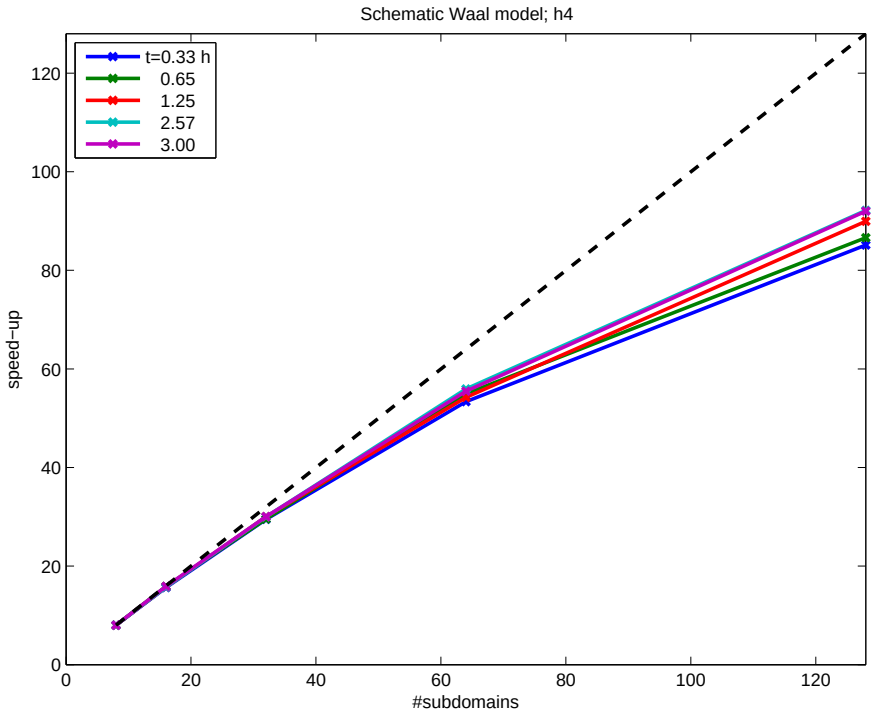


Figure 8.5: *Speed-up of the schematic Waal model; h4*

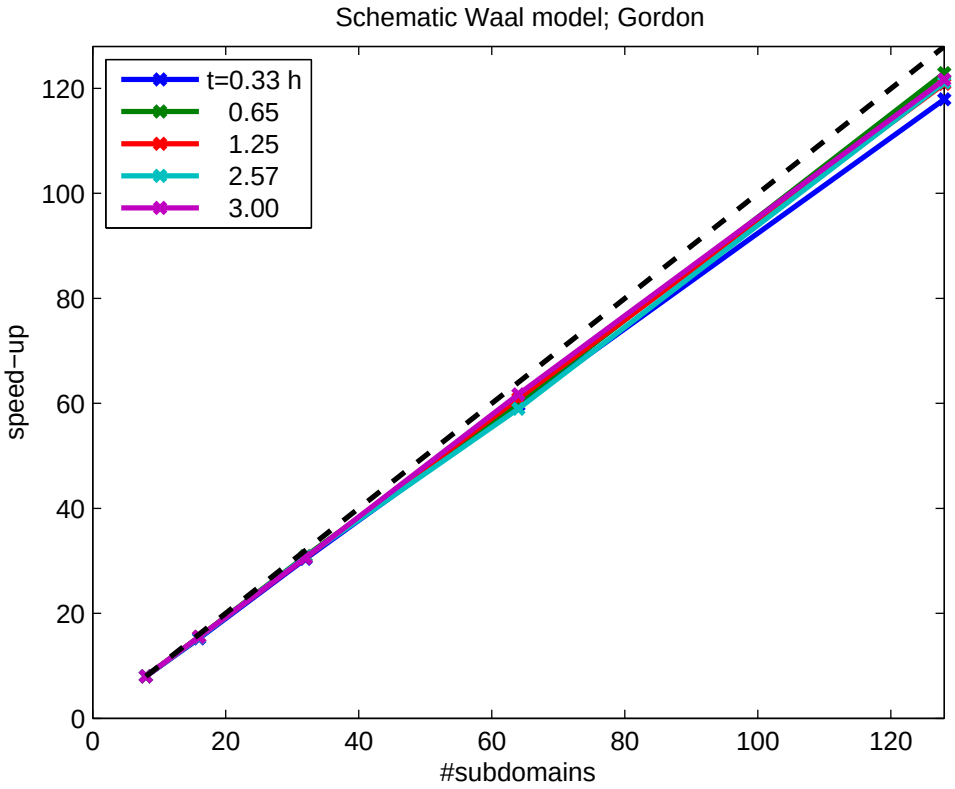


Figure 8.6: *Speed-up of the schematic Waal model; SDSC's Gordon; PETSc*

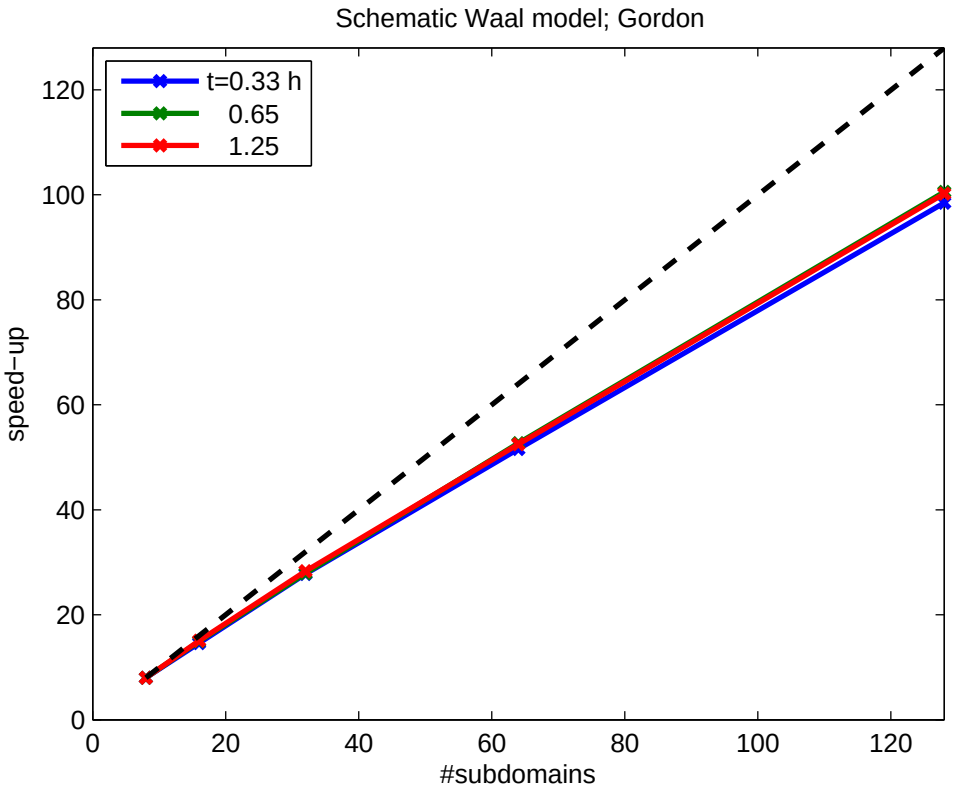
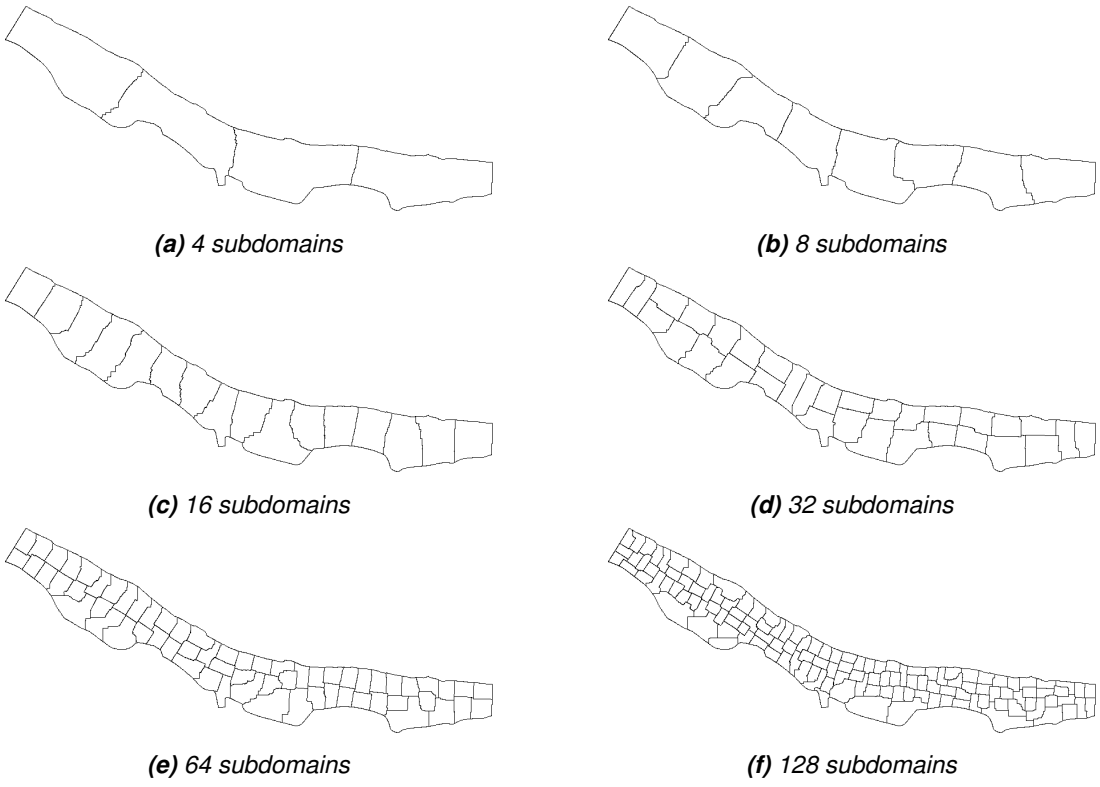


Figure 8.7: *Speed-up of the schematic Waal model; SDSC's Gordon; CG+MILUD*



(a) *4 subdomains*

(b) *8 subdomains*

(c) *16 subdomains*

(d) *32 subdomains*

(e) *64 subdomains*

(f) *128 subdomains*

Figure 8.8: *Partitioning of the 'esk-model' with METIS*

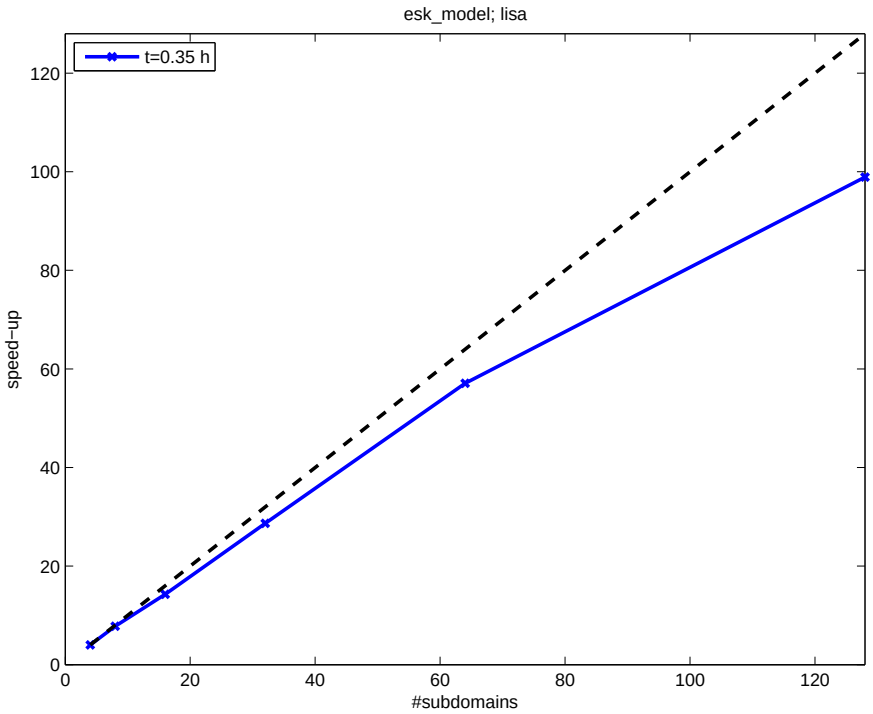


Figure 8.9: *Speed-up of the 'esk-model'; Lisa*

Table 8.7: *time-step averaged wall-clock times of the Schematic Waal model; Gordon; note: MPI communication times are not measured for the PETSc solver*

#dmns	t [h]	time step [s]	$MPI_{non\text{-}sol}$ [s]	solver [s]	MPI_{sol} [s]	#Krylov-iters
8	0.33	2.85100	0.03012	1.15250	0.00000	15.06000
	0.65	2.85000	0.03273	1.14950	0.00000	15.00000
	1.25					
	2.57					
	3.00					
16	0.33	1.02900	0.02056	0.20595	0.00000	0.00000
	0.65	1.01950	0.01502	0.20350	0.00000	0.00000
	1.25	1.02277	0.01663	0.20297	0.00000	0.00000
	2.57	1.01980	0.01624	0.20396	0.00000	0.00000
	3.00					
32	0.33	0.74950	0.02224	0.32080	0.00000	16.84500
	0.65	0.73900	0.02293	0.31000	0.00000	16.00000
	1.25	0.71634	0.02423	0.28614	0.00000	14.11386
	2.57					
	3.00					
64	0.33	0.37265	0.01499	0.15550	0.00000	17.41000
	0.65	0.36750	0.01483	0.15075	0.00000	16.59500
	1.25	0.35792	0.01431	0.14059	0.00000	14.90594
	2.57	0.34802	0.01390	0.13119	0.00000	13.28218
	3.00	0.34802	0.01353	0.13069	0.00000	13.22277
128	0.33	0.23665	0.03332	0.10325	0.00000	17.03000
	0.65	0.23350	0.03431	0.09945	0.00000	16.00000
	1.25	0.22624	0.03432	0.09287	0.00000	14.16832
	2.57	0.22228	0.03386	0.08861	0.00000	13.05446
	3.00	0.22129	0.03327	0.08812	0.00000	13.00000

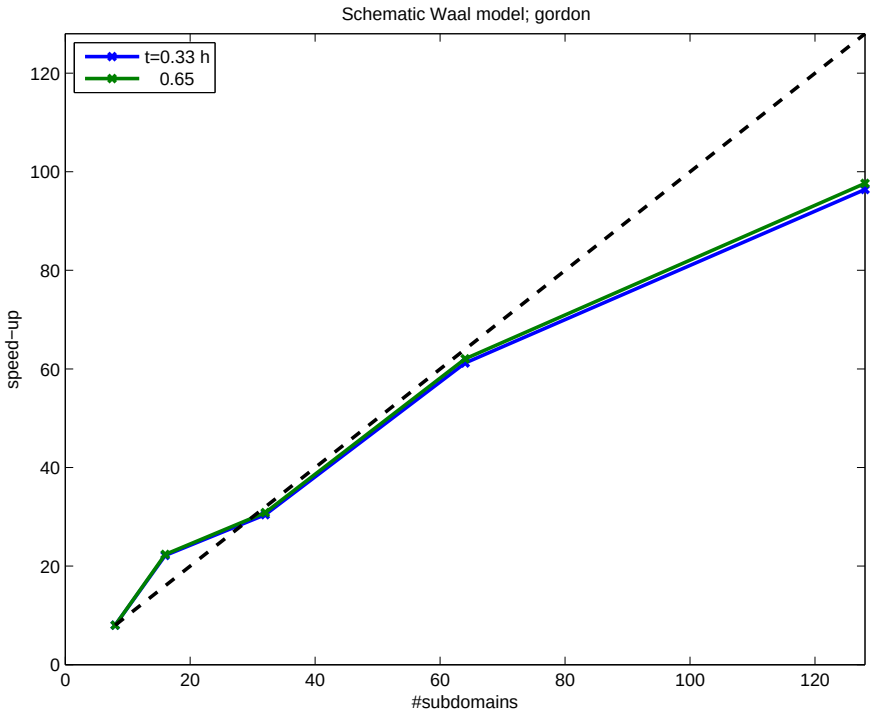


Figure 8.10: *Speed-up of the schematic Waal model; Gordon*

8.2.3 San Fransisco Delta-Bay model

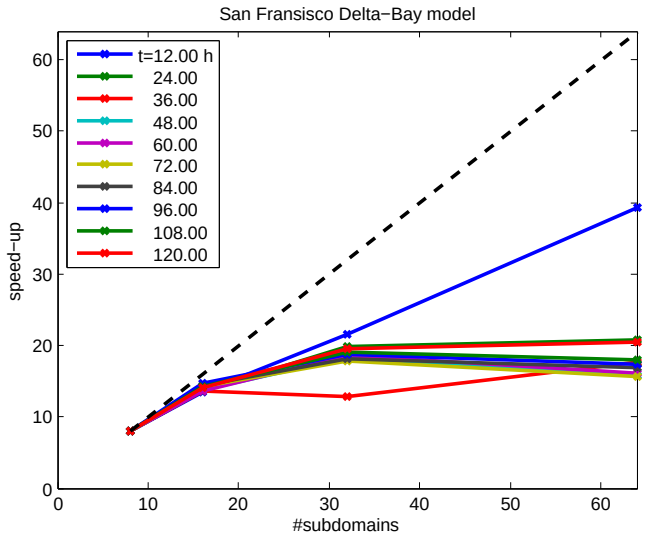


Figure 8.11: *Speed-up of the San Fransisco Delta-Bay model; Gordon*

Table 8.8: *time-step averaged wall-clock times of the San Fransisco Delta-Bay model; Gordon; note: MPI communication times are not measured for the PETSc solver*

#dmns	t [h]	time step [s]	$MPI_{non\text{-}sol}$ [s]	solver [s]	MPI_{sol} [s]	#Krylov-iters
8	12.00	0.14902	0.03099	0.01477	0.00000	16.25754
	24.00	0.14889	0.02928	0.01528	0.00000	17.40124
	36.00	0.14595	0.02592	0.01490	0.00000	15.49879
	48.00	0.14759	0.02715	0.01527	0.00000	16.83857
	60.00	0.14408	0.02398	0.01498	0.00000	15.99481
	72.00	0.14509	0.02441	0.01519	0.00000	16.30267
	84.00	0.14399	0.02353	0.01504	0.00000	16.12262
	96.00	0.14880	0.02593	0.01518	0.00000	16.15620
	108.00	0.14632	0.02445	0.01512	0.00000	16.01746
	120.00	0.15038	0.02918	0.01510	0.00000	15.54151
16	12.00	0.08892	0.02771	0.00958	0.00000	13.72882
	24.00	0.08726	0.02380	0.00981	0.00000	15.03835
	36.00	0.08581	0.02293	0.00973	0.00000	12.97420
	48.00	0.08607	0.02290	0.00991	0.00000	14.47656
	60.00	0.08461	0.02147	0.00993	0.00000	13.58980
	72.00	0.08183	0.01859	0.01005	0.00000	14.13277
	84.00	0.08015	0.01716	0.00999	0.00000	13.80952
	96.00	0.08123	0.01806	0.00998	0.00000	13.95845
	108.00	0.08180	0.01852	0.00989	0.00000	13.77271
	120.00	0.08547	0.02227	0.00985	0.00000	13.07582
32	12.00	0.05542	0.02316	0.00776	0.00000	18.51036
	24.00	0.06021	0.02928	0.00791	0.00000	20.00691
	36.00	0.09062	0.05868	0.00775	0.00000	17.71882
	48.00	0.06466	0.03252	0.00790	0.00000	19.38103
	60.00	0.06179	0.02975	0.00779	0.00000	18.33746
	72.00	0.06537	0.03326	0.00785	0.00000	18.83925
	84.00	0.06325	0.03124	0.00780	0.00000	18.64772
	96.00	0.06352	0.03145	0.00782	0.00000	18.54788
	108.00	0.06137	0.02929	0.00787	0.00000	18.57109
	120.00	0.06140	0.02937	0.00775	0.00000	17.73242
64	12.00	0.03032	0.01249	0.00593	0.00000	19.85593
	24.00	0.05749	0.03969	0.00612	0.00000	22.02080
	36.00	0.06610	0.04823	0.00595	0.00000	18.88561
	48.00	0.07463	0.05664	0.00613	0.00000	21.21491
	60.00	0.07151	0.05354	0.00605	0.00000	19.72320
	72.00	0.07442	0.05639	0.00611	0.00000	20.56026
	84.00	0.06798	0.05006	0.00607	0.00000	20.07916
	96.00	0.06844	0.05042	0.00611	0.00000	20.40540
	108.00	0.06524	0.04724	0.00609	0.00000	20.24993
	120.00	0.05862	0.04063	0.00606	0.00000	19.43446

Table 8.9: *time-step averaged wall-clock times of the San Fransisco Delta-Bay model; Gordon; non-solver MPI communication times*

#dmns	t [h]	MPI_u [s]	$MPI_{s_{all}}$ [s]	MPI_{reduce}
8	12.00	0.00028	0.02238	0.00833
	24.00	0.00029	0.01985	0.00914
	36.00	0.00028	0.01609	0.00956
	48.00	0.00028	0.01600	0.01086
	60.00	0.00029	0.01268	0.01101
	72.00	0.00027	0.01284	0.01130
	84.00	0.00028	0.01227	0.01098
	96.00	0.00027	0.01346	0.01220
	108.00	0.00028	0.01249	0.01168
	120.00	0.00028	0.01756	0.01133
16	12.00	0.00030	0.02184	0.00557
	24.00	0.00030	0.01594	0.00756
	36.00	0.00031	0.01250	0.01012
	48.00	0.00030	0.01202	0.01058
	60.00	0.00031	0.00982	0.01134
	72.00	0.00030	0.00866	0.00963
	84.00	0.00031	0.00795	0.00890
	96.00	0.00031	0.00869	0.00907
	108.00	0.00032	0.00807	0.01013
	120.00	0.00030	0.01173	0.01024
32	12.00	0.00032	0.01859	0.00424
	24.00	0.00033	0.01228	0.01667
	36.00	0.00033	0.00884	0.04951
	48.00	0.00032	0.00736	0.02484
	60.00	0.00034	0.00595	0.02347
	72.00	0.00031	0.00582	0.02713
	84.00	0.00033	0.00574	0.02517
	96.00	0.00033	0.00669	0.02442
	108.00	0.00033	0.00588	0.02307
	120.00	0.00032	0.00809	0.02096
64	12.00	0.00021	0.00902	0.00326
	24.00	0.00022	0.00886	0.03060
	36.00	0.00023	0.00696	0.04103
	48.00	0.00023	0.00676	0.04966
	60.00	0.00022	0.00437	0.04895
	72.00	0.00024	0.00395	0.05220
	84.00	0.00024	0.00375	0.04607
	96.00	0.00023	0.00423	0.04597
	108.00	0.00021	0.00374	0.04329
	120.00	0.00020	0.00517	0.03526

8.3 Governing equations

D-Flow FM solves the two- and three-dimensional shallow-water equations. We will focus on two dimensions first. The shallow-water equations express conservation of mass and momentum and can be put into the following form:

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_{\Omega} h \,\mathrm{d}\Omega + \int_{\partial\Omega} h\boldsymbol{u}\cdot\boldsymbol{n}\,\mathrm{d}\Gamma = 0, \tag{8.16}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_{\Omega} h\boldsymbol{u}\,\mathrm{d}\Omega + \int_{\partial\Omega} h\boldsymbol{u}\boldsymbol{u}\cdot\boldsymbol{n}\,\mathrm{d}\Gamma = -\int_{\partial\Omega}\frac{1}{2}h^2\boldsymbol{n}\,\mathrm{d}\Gamma - \int_{\Omega} h\nabla d\,\mathrm{d}\Omega \tag{8.17}$$

$$+\int_{\partial\Omega}(\nu h(\nabla\boldsymbol{u}+\nabla\boldsymbol{u}^T))\cdot\boldsymbol{n}\,\mathrm{d}\Gamma + \int_{\Omega}\tau\,\mathrm{d}\Omega,$$

(8.18)

where ζ is the water level, h the water height, $d=\zeta-h$ the bed level, $\boldsymbol{u}$ the velocity vector, g the gravitational acceleration, ν the viscosity and $\boldsymbol{\tau}$ is the bottom friction:

$$\boldsymbol{\tau} = \frac{g}{C^2}\|\boldsymbol{u}\|\boldsymbol{u}, \tag{8.19}$$

with C being the Chézy coefficient.

8.4 Spatial discretization

The spatial discretization is performed in a staggered manner, i.e. velocity normal components u_j are defined at the cell faces j, with face normal vector $\boldsymbol{n}_j$, and the water levels s_k at cell centers k.

We define volume V_k associated with cell Ω_k as

$$V_k = \int_{\Omega} h\,\mathrm{d}\Omega \tag{8.20}$$

and the discharge through face j as

$$q_j = \int_{\Gamma_j} h\boldsymbol{u}\cdot\boldsymbol{n}\,\mathrm{d}\Gamma, \tag{8.21}$$

which is discretized as

$$q_j = h_{\mathrm{upwind}(j)}u_j. \tag{8.22}$$

The upwind cell associated with face j is

$$\mathrm{upwind}(j) = \begin{cases} L(j), & u_j \geq 0, \\ R(j), & u_j < 0. \end{cases} \tag{8.23}$$

We define the water-column volume V_k as

$$V_k = \int_{\Omega_k} h\,\mathrm{d}\Omega \tag{8.24}$$

and for simplicity assume that it can be expressed as

$$V_k = b_{Ak} h_k, \tag{8.25}$$

where Ω_k is the (two-dimensional) grid cell and b_{Ak} is the area of its horizontal projection. Note that this relation does not hold in case of (partially) dry cells.

Borsboom et al. show that Eqns. (8.16) and (8.18) can be discretized conservatively as:

$$\frac{\mathrm{d}}{\mathrm{d}t} h_k = \frac{1}{b_{Ak}} \sum_{j \in \mathcal{J}(k)} q_j 1_{j,k}, \tag{8.26}$$

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}(\tilde{h}_j u_j) = &- g \bar{h}_j \frac{(s_{R(j)} - s_{L(j)})}{\Delta x_j} - \left(\alpha_{Lj} \boldsymbol{\mathcal{A}}_{L(j)} + \alpha_{Rj} \boldsymbol{\mathcal{A}}_{R(j)}\right) \cdot \boldsymbol{n}_j \\ &+ \left(\alpha_{Lj} \boldsymbol{\mathcal{D}}_{L(j)} + \alpha_{Rj} \boldsymbol{\mathcal{D}}_{R(j)}\right) \cdot \boldsymbol{n}_j + \tau_j, \end{aligned} \tag{8.27}$$

where $\Delta x_j = \|\boldsymbol{x}_{R(j)} - \boldsymbol{x}_{L(j)}\|$, $\tilde{h}_j$ is the weighted average face water height

$$\tilde{h}_j = \alpha_{L(j)} h_{L(j)} + \alpha_{R(j)} h_{R(j)}, \tag{8.28}$$

$\bar{h}_j$ is the average face water height

$$\bar{h}_j = \frac{1}{2} h_{L(j)} + \frac{1}{2} h_{R(j)}, \tag{8.29}$$

$\boldsymbol{\mathcal{A}}_k$ is the cell-centered conservative advection of $h\boldsymbol{u}$, discretized as

$$\boldsymbol{\mathcal{A}}_k = \frac{1}{b_{Ak}} \sum_{l \in \mathcal{J}(k)} \boldsymbol{u}_{\text{cupwind}(l)} q_l 1_{l,k}. \tag{8.30}$$

and $\boldsymbol{\mathcal{D}}_k$ is the cell-centered diffusion, not discussed further. The cell-center based velocity vectors are reconstructed from the face-normal velocity components with

$$\boldsymbol{u}_{ck} = \frac{1}{b_{Ak}} \sum_{j \in \mathcal{J}(k)} (\boldsymbol{x}_{uj} - \boldsymbol{x}_{ck}) u_j \Delta\Gamma_j 1_{j,k}. \tag{8.31}$$

Using

$$\frac{\mathrm{d}}{\mathrm{d}t}(\tilde{h}_j u_j) = \tilde{h}_j \frac{\mathrm{d}u_j}{\mathrm{d}t} + u_j \frac{\mathrm{d}\tilde{h}_j}{\mathrm{d}t} = \tilde{h}_j \frac{\mathrm{d}u_j}{\mathrm{d}t} + \left(\alpha_L \frac{\mathrm{d}h_{L(j)}}{\mathrm{d}t} + \alpha_R \frac{\mathrm{d}h_{R(j)}}{\mathrm{d}t}\right) \tag{8.32}$$

and substituting Eqn. (8.26) we obtain

$$\begin{aligned} \frac{\mathrm{d}u_j}{\mathrm{d}t} = &- g \frac{\bar{h}_j}{\tilde{h}_j} \frac{(s_{R(j)} - s_{L(j)})}{\Delta x_j} \\ &- \frac{1}{\tilde{h}_j} \left(\alpha_{Lj} \frac{1}{b_{AL(j)}} \sum_{l \in \mathcal{J}(L(j))} (\boldsymbol{u}_{\text{cupwind}(l)} \cdot \boldsymbol{n}_j - u_j) q_l 1_{l,k} + \right. \\ &\qquad \left. \alpha_{Rj} \frac{1}{b_{AR(j)}} \sum_{l \in \mathcal{J}(R(j))} (\boldsymbol{u}_{\text{cupwind}(l)} \cdot \boldsymbol{n}_j - u_j) q_l 1_{l,k} \right) \\ &+ \frac{1}{\tilde{h}_j} \left(\alpha_{Lj} \boldsymbol{\mathcal{D}}_{L(j)} + \alpha_{Rj} \boldsymbol{\mathcal{D}}_{R(j)}\right) \cdot \boldsymbol{n}_j + \frac{\tau_j}{\tilde{h}_j}. \end{aligned} \tag{8.33}$$

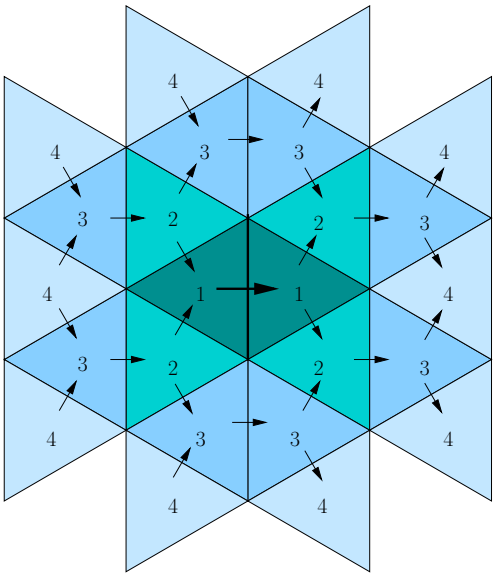


Figure 8.12: *Stencil for momentum advection and diffusion; the numbers indicate the level of the neighboring cells*

The advection and pressure-gradient terms are conform Kramer and Stelling. In D-Flow FM, however, the following form is implemented:

$$\begin{aligned}\frac{\mathrm{d}u_j}{\mathrm{d}t} =& -g\frac{(s_{R(j)}-s_{L(j)})}{\Delta x_j}\\ &-\frac{1}{V_{uj}}\Bigg(\alpha_{Lj}\sum_{l\in\mathcal{J}(L(j))}(\boldsymbol{u}_{\text{cupwind}(l)}\cdot\boldsymbol{n}_j-u_j)q_l 1_{l,k}+\\ &\qquad\alpha_{Rj}\sum_{l\in\mathcal{J}(R(j))}(\boldsymbol{u}_{\text{cupwind}(l)}\cdot\boldsymbol{n}_j-u_j)q_l 1_{l,k}\Bigg)\\ &+\frac{1}{\bar{h}_j}\left(\alpha_{Lj}\boldsymbol{\mathcal{D}}_{L(j)}+\alpha_{Rj}\boldsymbol{\mathcal{D}}_{R(j)}\right)\cdot\boldsymbol{n}_j+\frac{\tau_j}{h_{Rj}},\end{aligned} \tag{8.34}$$

where V_{uj} is a face-based volume

$$V_{uj}=\alpha_{Lj}V_{L(j)}+\alpha_{Rj}V_{R(j)} \tag{8.35}$$

and h_{Rj} is the hydraulic radius of face j.

The stencil used for computing momentum advection and diffusion is depicted in Fig. 8.12.

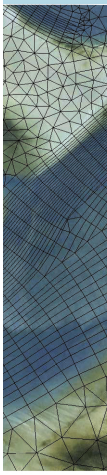
Issues:

- ◇ orthogonal meshes hard to achieve, compromises mesh smoothness,
- ◇ non-conservative advection and different pressure gradient term implemented, but gives best results for shock problems,
- ◇ higher-order implementation gives satisfactory results for swirling flows,
- ◇ shear-dominated flow suffers from wide advection stencil, see Poiseuille test-case.

9 Trachytopes

9.1 Introduction

This functionality allows you to specify the bed roughness and flow resistance on a sub-grid level by defining and using various land use or roughness/resistance classes, further referred to as trachytopes after the Greek word $\tau\rho\alpha\chi\acute{\upsilon}\tau\eta\varsigma$ for roughness. The input parameters in the .mdu and .arl and .ttd files to use the trachytopes functionality are described in the file types appendix in the User Manual. (The following description is based on the description in the Delft3D-FLOW manual)



At every time step (or less frequent as requested by the user) the trachytopes are converted into a representative bed roughness C, k or n and optional linear flow resistance coefficient $\lambda_{m,n}$ per velocity point.

$$M_\xi = -\frac{1}{2}\lambda_{m,n} u \sqrt{u^2 + v^2} \tag{9.1}$$

$$M_\eta = -\frac{1}{2}\lambda_{m,n} v \sqrt{u^2 + v^2} \tag{9.2}$$

To save computational time the user may choose to update the computed bed roughness and resistance coefficients less frequently than every time step. See .mdu, .arl and .ttd file type descriptions for an overview of the keywords and input files associated with this feature.

The following two sections describe the various classes of trachytopes distinguished and the way in which they are combined, respectively.

9.2 Trachytope classes

Three base classes of trachytopes are distinguished: area classes, line classes and point classes. The area classes (type range 51–200) basically cover the whole area, therefore, they are generally the dominant roughness factor. The line classes (type range 201–250) may be used to represent hedges and similar flow resistance elements; it will add anisotropy to the roughness field. The point class (type range 251–300) represents a set of point flow resistance elements. The next six sections provide an overview of the various trachytope formulae implemented.

Special classes (1–50)

In addition to the three base classes two special trachytope classes have been defined: a flood protected area and a composite trachytope class. The first class represents a sub-grid area that is protected from flooding and thus does not contribute to the bed roughness; however, the effect on the flow resistance should be taken into account. The second class can be used to make derived trachytope classes that are a combination of two other trachytopes: an area fraction α of trachytope type T_1 and an area fraction β (often equal to $1 - \alpha$) of trachytope type T_2.

FormNr	Name	Formula
Special classes (1–50)		
1	flood protected area	area fraction shows up as f_b in Eqs. 9.54 and 9.57
2	composite trachytope	fraction α of type T_1 and fraction β (generally $\beta = 1 - \alpha$) of type T_2

Area trachytope classes (51–200)

The class of area trachytopes is subdivided into three types: simple (51–100), alluvial (101–150) and vegetation (151–200). Four simple area trachytopes have been implemented representing the four standard roughness types of flow module.

FormNr	Name	Formula
51	White-Colebrook value	k
52	Chézy value	C
53	Manning value	$C = \sqrt[6]{h}/n$
54	z_0 value	$k = 30 z_0$

Four alluvial trachytopes have been implemented.

FormNr	Name	Formula
101	simplified Van Rijn (1984)	Equation 9.3
102	power relation	Equation 9.4
103[1]	Van Rijn	Equations 9.5 to 9.13
104[1]	Struiksma	Equations 9.14 to 9.17
105[1]	bedforms quadratic	Equation 9.18
106[1]	bedforms linear	Equation 9.19

The first alluvial roughness formula is a simplified version of the Van Rijn (1984) alluvial roughness predictor

$$k = A h^{0.7} \left[1 - e^{-B h^{-0.3}} \right] \tag{9.3}$$

it is obtained from Equation 9.5 by noting that $h_b \propto h^{0.7}$ and $L_b \propto h$ and ignoring the grain related roughness. The parameters A and B can be calibrated by the user. The second formula implemented is a straightforward general power law

$$C = A h^B \tag{9.4}$$

where A and B are calibration coefficients. The Van Rijn (1984) alluvial roughness predictor reads

$$k = k_{90} + 1.1 h_b \left(1 - e^{-25 h_b / L_b} \right) \tag{9.5}$$

[1] The alluvial roughness predictors 103, 104, 105 and 106 are not yet supported, because the coupling with the morphology module is not yet available.

where the bedform height h_b and length L_b are given by

$$h_b = 0.11h \left(\frac{D_{50}}{h}\right)^{0.3} \left(1 - e^{-T/2}\right)(25 - T) \tag{9.6}$$

$$L_b = 7.3h \tag{9.7}$$

where h is the local water depth and the transport stage parameter T is given by

$$T = \frac{u'^2_* - u^2_{*,cr}}{u^2_{*,cr}} \tag{9.8}$$

where u'_* is the bed shear velocity given by

$$u'^2_* = gu^2/C^2_{g,90} \tag{9.9}$$

where

$$C_{g,90} = 18\,{}^{10}\log(12h/k_{90}) \text{ and } k_{90} = 3D_{90} \tag{9.10}$$

and $u_{*,cr}$ is the critical bed shear velocity according Shields given by

$$u^2_{*,cr} = g\Delta D_{50}\theta_c \tag{9.11}$$

given

$$\theta_c = \begin{cases} 0.24/D_* & \text{if } D_* \leq 4 \\ 0.14D_*^{-0.64} & \text{if } 4 < D_* \leq 10 \\ 0.04D_*^{-0.10} & \text{if } 10 < D_* \leq 20 \\ 0.013D_*^{0.29} & \text{if } 20 < D_* \leq 150 \\ 0.055 & \text{if } 150 < D_* \end{cases} \tag{9.12}$$

where

$$D_* = D_{50}\left(\frac{g\Delta}{\nu^2}\right)^{1/3} \tag{9.13}$$

This predictor does not contain any calibration coefficients but requires D_{50} and D_{90} data from the morphology module. The coupling with the morphology module is work in progress

The second alluvial roughness predictor proposed by (Struiksma, pers. comm.) allows for a lot of adjustments, it reads

$$\frac{1}{C^2} = (1 - \xi)\frac{1}{C^2_{90}} + \xi\frac{1}{C^2_{min}} \tag{9.14}$$

where

$$C_{90} = A_1\,{}^{10}\log(A_2h/D_{90}) \tag{9.15}$$

and

$$\xi = \frac{\max(0, \theta_g - \theta_c)}{\theta_m - \theta_c}\frac{\theta^2_m - \theta_c\theta_g}{(\theta_m - \theta_c)\theta_g} \tag{9.16}$$

which varies from 0 at $\theta_g \leq \theta_c$ to 1 at $\theta_g = \theta_m$ where

$$\theta_g = \frac{u^2}{C^2_{90}\Delta D_{50}} \tag{9.17}$$

and A_1, A_2, θ_c, θ_m, $C_{\min}$ are coefficients that the user needs to specify. This formula requires also D_{50} and D_{90} data from the morphology module. The fifth formula is based on Van Rijn (2007) and reads

$$k = \min(\sqrt{k_{s,r}^2 + k_{s,mr}^2 + k_{s,d}^2}, \frac{h}{2}) \tag{9.18}$$

It uses the roughness heights of ripples k_r, mega-ripples k_{mr} and dunes k_d. These roughness heights are based on Van Rijn (2007). These formulae depend on D_{50} and D_{90} data which may be either specified as part of the roughness type or obtained from the morphology module. The sixth formula is similar, but uses a linear addition

$$k = \min(k_{s,r} + k_{s,mr} + k_{s,d}, \frac{h}{2}) \tag{9.19}$$

Four vegetation based area trachytopes have been implemented. Two formulae (referred to as 'Barneveld') are based on the work by Klopstra *et al.* (1996, 1997) and two on the work by Baptist (2005).

FormNr	**Name**	**Formula**
151	Barneveld 1	Eqs. 9.20 – 9.29, $C_D = 1.65$
152	Barneveld 2	Eqs. 9.20 – 9.26, 9.30 – 9.32
153	Baptist 1	Eqs. 9.33 and 9.34
154	Baptist 2	Eqs. 9.35, 9.37 and 9.38

The formula by Klopstra *et al.* (1997) reads

$$C = \frac{1}{h^{3/2}} \left\{ \begin{array}{l} \frac{2}{\sqrt{2A}} \left(\sqrt{C_3 e^{h_v\sqrt{2A}} + u_{v0}^2} \right) + \\ \quad \frac{u_{v0}}{\sqrt{2A}} \ln \left(\frac{(\sqrt{C_3 e^{h_v\sqrt{2A}} + u_{v0}^2} - u_{v0})(\sqrt{C_3 + u_{v0}^2} + u_{v0})}{(\sqrt{C_3 e^{2\sqrt{2A}} + u_{v0}^2} + u_{v0})(\sqrt{C_3 + u_{v0}^2} - u_{v0})} \right) + \\ \quad \frac{\sqrt{g(h-(h_v-a))}}{\kappa} \left((h - (h_v - a)) \ln \left(\frac{h-(h_v-a)}{z_0} \right) - a \ln \left(\frac{a}{z_0} \right) - (h - h_v) \right) \end{array} \right\} \tag{9.20}$$

where

$$A = \frac{nC_D}{2\alpha} \tag{9.21}$$

$$C_3 = \frac{2g(h - h_v)}{\alpha\sqrt{2A}(e^{h_v\sqrt{2A}} + e^{-h_v\sqrt{2A}})} \tag{9.22}$$

$$a = \frac{1 + \sqrt{1 + \frac{4E_1^2\kappa^2(h-h_v)}{g}}}{\frac{2E_1^2\kappa^2}{g}} \tag{9.23}$$

and

$$z_0 = ae^{-F} \tag{9.24}$$

where

$$E_1 = \frac{\sqrt{2A}C_3 e^{h_v\sqrt{2A}}}{2\sqrt{C_3 e^{h_v\sqrt{2A}} + u_{v0}^2}} \tag{9.25}$$

and

$$F = \frac{\kappa\sqrt{C_3 e^{h_v\sqrt{2A}} + u_{v0}^2}}{\sqrt{g(h-(h_v-a))}} \tag{9.26}$$

Here, h is the water depth, h_v is the vegetation height, and $n = mD$ where m is the number of stems per square metre and D is the stem diameter. For the first implementation the parameter α in Equation 9.22 is given by

$$\alpha = \max(0.001, 0.01\sqrt{hh_v}) \tag{9.27}$$

and the velocity within the vegetation is approximated by $u_{v0}\sqrt{i}$ where

$$u_{v0}^2 = \frac{2g}{C_D n} \tag{9.28}$$

and i is the water level gradient. For emerged vegetation the first implementation reads

$$\frac{1}{C^2} = \frac{C_D n h}{2g} \tag{9.29}$$

The second implementation based on Klopstra *et al.* (1996) is identical except for the following modifications to Eqs. 9.27 – 9.29. The main difference between the two implementations is the inclusion of the roughness C_b of the bed itself (without vegetation). The parameter α in Equation 9.22 is now given by

$$\alpha = 0.0227 h_v^{0.7} \tag{9.30}$$

and the velocity within the vegetation is approximated by $u_{v0}\sqrt{i}$ where

$$u_{v0}^2 = \frac{h_v}{\frac{C_D h_v n}{2g} + \frac{1}{C_b^2}} \tag{9.31}$$

and i is the water level gradient. For emerged vegetation the second implementation reads

$$\frac{1}{C^2} = \frac{C_D n h}{2g} + \frac{1}{C_b^2} \tag{9.32}$$

For large values of C_b the latter two equations simplify to the corresponding equations of the first implementation. The first implementation requires vegetation height h_v and density n as input parameters (the drag coefficient C_D is equal to 1.65); for second implementation you'll also need to specify the drag coefficient C_D and the alluvial bed roughness k_b (C_b in Equation 9.32 is computed as $18\,^{10}\log(12h/k_b)$).

The first implementation of the roughness predictor by Baptist (Baptist, 2005) reads for the case of submerged vegetation

$$C = \frac{1}{\sqrt{\frac{1}{C_b^2} + \frac{C_D n h_v}{2g}}} + \frac{\sqrt{g}}{\kappa}\ln(\frac{h}{h_v}) \tag{9.33}$$

where n is the vegetation density ($n = mD$ where m is the number of stems per square metre and D is the stem diameter). The second term goes to zero at the transition from submerged to emerged vegetation. At that transition the formula changes into the formula for non-submerged vegetation which reads

$$C = \frac{1}{\sqrt{\frac{1}{C_b^2} + \frac{C_D n h}{2g}}} \tag{9.34}$$

which is identical to the non-submerged case of the second implementation of the work by Klopstra *et al.* (1996) (see Equation 9.32).

The drawback of the three vegetation based formulations above is that they parameterize the flow resistance by means of the bed roughness. Consequently, the presence of vegetation will lead to a higher bed roughness and thus to a higher bed shear stress and larger sediment transport rates in case of morphological computations. Therefore, we have included a $-\frac{\lambda}{2}u^2$ term in the momentum equation where λ represents the flow resistance of the vegetation. For the case of non-submerged vegetation $h < h_v$ the flow resistance and bed roughness are strictly separated

$$C = C_b \quad \text{and} \quad \lambda = C_D n \tag{9.35}$$

In the case of submerged vegetation $h > h_v$ the two terms can't be split in an equally clean manner. However, we can split the terms such that the bed shear stress computed using the depth averaged velocity u and the net bed roughness C equals the bed shear stress computed using the velocity u_v within the vegetation layer and the real bed roughness C_b.

$$\frac{u^2}{C^2} = \frac{u_v^2}{C_b^2} \tag{9.36}$$

With this additional requirement we can rewrite Equation 9.33 as

$$C = C_b + \frac{\sqrt{g}}{\kappa} \ln(\frac{h}{h_v}) \sqrt{1 + \frac{C_D n h_v C_b^2}{2g}} \tag{9.37}$$

and

$$\lambda = C_D n \frac{h_v}{h} \frac{C_b^2}{C^2} \tag{9.38}$$

which simplify to Equation 9.35 for $h = h_v$. Both formulae by Baptist require vegetation height h_v, density n, drag coefficient C_D and alluvial bed roughness C_b as input parameters.

Linear trachytope classes (201–250)

Two formulae have been implemented for linear trachytopes such as hedges or bridge piers.

FormNr	**Name**	**Formula**
201	hedges 1	Eqs. 9.39 to 9.41
202	hedges 2	Eqs. 9.42 to 9.44

The first implementation reads

$$\frac{1}{C^2} = \frac{h}{2g}\frac{L_{hedge}}{W_{cell}L_{cell}}\frac{1-\mu^2}{\mu^2} \tag{9.39}$$

where L_{hedge} is the projected length of the hedge, W_{cell} and L_{cell} are the width and length of the grid cell. The ratio L_{hedge}/W_{cell} may be interpreted as the number of hedges that the flow encounters per unit width. The second ratio is thus the inverse of the average distance between these hedges within the grid cell. The last term may be loosely referred to as the drag of the hedge, which is determined by the hedge pass factor μ given by

$$\mu = 1 + 0.175n\left(\frac{h}{h_v} - 2\right) \tag{9.40}$$

if the hedge extends above the water level ($h_v > h$) and is given by

$$\mu = 1 - 0.175n\left(\frac{h}{h_v}\right) \tag{9.41}$$

if the hedge is fully submerged ($h > h_v$) where n is a dimensionless hedge density. The second implementation reads

$$\frac{1}{C^2} = \frac{C_D n L_{hedge} h}{2gL_{cell}W_{cell}} \tag{9.42}$$

or equivalently

$$C = \sqrt{\frac{2gL_{cell}W_{cell}}{hL_{hedge}}}\left(\sqrt{\frac{1}{C_D n}}\right) \tag{9.43}$$

for non-submerged conditions and

$$C = \sqrt{\frac{2gL_{cell}W_{cell}}{hL_{hedge}}}\left(\frac{h_v}{h}\sqrt{\frac{1}{C_D n}} + m_0\sqrt{\frac{\left(\frac{h-h_v}{h}\right)^2}{1-\left(\frac{h-h_v}{h}\right)^2}}\right) \tag{9.44}$$

for submerged conditions. We recognize the same ratio $L_{cell}W_{cell}/L_{hedge}$ that represents the average distance between hedges. Equation 9.42 can be directly compared to similar equations for area trachytopes (Equation 9.29), point trachytopes (Equation 9.45) and bridge resistance (see Delft3D-FLOW manual). Note that the formula for computing the loss coefficient for a bridge explicitly includes the reduction in the flow area and the resulting increase in the effective flow velocity, whereas the above mentioned trachytope formulae don't.

Point trachytope classes: various (251–300)

One formula for point trachytopes has been implemented. It may be used to represent groups of individual trees or on a smaller scale plants.

FormNr	Name	Formula
251	trees	Eqn. 9.45

The implemented formula reads

$$C = \sqrt{\frac{2g}{C_D n \min(h_v, h)}} \tag{9.45}$$

where $n = mD$ with m the number of trees per unit area and D the characteristic tree diameter, h_v is the vegetation height and h is the local water depth. The formula is identical to Equation 9.34 except for the fact that the point trachytope formula has no bed roughness and area associated with it. The generalization of Equation 9.45 to the submerged case ($h > h_v$) lacks the extra term in Equation 9.33.

9.3 Averaging and accumulation of trachytopes

Point and linear roughnesses are accumulated by summing the inverse of the squared Chézy values C_i.

$$\frac{1}{C_{pnt}^2} = \sum_i \frac{1}{C_{pnt,i}^2} \tag{9.46}$$

$$\frac{1}{C_{lin}^2} = \sum_i \frac{1}{C_{lin,i}^2} \tag{9.47}$$

The area roughnesses are accumulated weighted by the surface area fraction f_i. These roughnesses are accumulated as White-Colebrook roughness values and as Chézy values; for the latter values both the linear sum ("parallel") and the sum of inverse of squared values ("serial") are computed. Roughness values are converted into each other as needed based on the local water depth using Eqs. **??–??**.

$$k_{area} = \sum_i f_i k_i \tag{9.48}$$

$$\frac{1}{C_{area,s}^2} = \sum_i f_i \frac{1}{C_i^2} \tag{9.49}$$

$$C_{area,p} = \sum_i f_i C_i \tag{9.50}$$

For the fraction of the grid cell area for which no roughness class is specified the default roughness is used.

The flow resistance coefficients are also accumulated proportionally to the surface area fraction f_i associated with the trachytope considered. For the fraction of the grid cell area for which no flow resistance is specified, obviously none is used.

$$\lambda = \sum_i f_i \lambda_i \tag{9.51}$$

The final effective bed roughness of the grid cell may be computed by either one of the following two methods.

Method 1

The total mean roughness is computed by summing the White-Colebrook values for the areas and line and point resistance features.

$$k_m = k_{area} + k_{lin} + k_{pnt} \tag{9.52}$$

where $k_{lin} = 12h10^{-C_{lin}/18}$ and $k_{pnt} = 12h10^{-C_{pnt}/18}$. The effect of the water free area fraction f_b is taken into account by means of the following empirical relation in which

$C_m = 18\ ^{10}\log(12h/k_m)$ is the mean Chézy value corresponding to the total mean White-Colebrook roughness value obtained from Equation 9.52.

$$f_b = \max(\min(0.843, f_b), 0.014) \tag{9.53}$$

$$C_{\text{total}} = C_m \left(1.12 - 0.25 f_b - 0.99\sqrt{f_b}\right) \tag{9.54}$$

The resulting C_{total} value is used in the computation. This method together with trachytope classes 1, 51, 101, 151 and 201 corresponds to the NIKURADSE option of the WAQUA/TRIWAQ flow solver.

Method 2

The total roughness is computed by first averaging over the serial and parallel averages of the Chézy values according

$$C_{area} = \alpha_s C_{area,s} + (1 - \alpha_s) C_{area,p} \tag{9.55}$$

where $\alpha_s = 0.6$ by default. Subsequenty the effect of the water free area fraction f_b is taken into account by means of the following empirical relation (identical to Equation 9.54 of method 1).

$$f_b = \max(\min(0.843, f_b), 0.014) \tag{9.56}$$

$$C_{area,corr} = C_{area} \left(1.12 - 0.25 f_b - 0.99\sqrt{f_b}\right) \tag{9.57}$$

Finally the Chézy value representing the total bed roughness is computed by accumulating the inverses of the squared Chézy values.

$$\frac{1}{C_{total}^2} = \frac{1}{C_{area,corr}^2} + \frac{1}{C_{lin}^2} + \frac{1}{C_{pnt}^2} \tag{9.58}$$

The resulting C_{total} value is used in the computation. This method together with trachytope classes 1, 51, 52, 53, 101, 152, 202 and 251 corresponds to the ROUGHCOMBINATION option of the WAQUA/TRIWAQ flow solver.

References

Balay, S., J. Brown, , K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith and H. Zhang, 2013. *PETSc Users Manual*. Tech. Rep. ANL-95/11 - Revision 3.4, Argonne National Laboratory.

Baptist, M. J., 2005. *Modelling floodplain biogeomorphology*. Ph.D. thesis, Delft University of Technology.

Borsboom, M., 2013. *Construction and analysis of D-FLOW FM-type discretizations*. Tech. rep., Deltares memorandum.

Huang, W., 2001. "Practical Aspects of Formulation and Solution of Moving Mesh Partial Differential Equations." *Journal of Computational Physics* 171: 753-775.

Huang, W., 2005. "Anisotropic Mesh Adaptation and Movement." lecture notes for the workshop on Adaptive Method, Theory and Application.

Karypis, G., 2011. *METIS - A Software Package for Partitioning Unstructured Graph, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices, Version 5.0*. Tech. rep., University of Minnesota.

Kleptsova, O., G. S. Stelling and J. D. Pietrzak, 2010. "An accurate momentum advection scheme for a z-level coordinate models." *Ocean Dyn.* 60 (6): 1447–1461. DOI 10.1007/s10236-010-0350-y.

Klopstra, D., H. J. Barneveld and J. M. Van Noortwijk, 1996. *Analytisch model hydraulische ruwheid van overstroomde moerasvegetatie*. Tech. Rep. PR051, HKV consultants, Lelystad, The Netherlands. Commissioned by Rijkswaterstaat/RIZA, The Netherlands.

Klopstra, D., H. J. Barneveld, J. M. Van Noortwijk and E. H. Van Velzen, 1997. "Analytical model for hydraulic roughness of submerged vegetation." In *The 27th IAHR Congress, San Francisco, 1997; Proceedings of Theme A, Managing Water: Coping with Scarcity and Abundance*, pages 775-780. American Society of Civil Engineers (ASCE), New York.

Kramer, S. C. and G. S. Stelling, 2008. "A conservative unstructured scheme for rapidly varied flows." *Int. J. Numer. Methods Fluids* 58 (2): 183–212. DOI 10.1002/fld.1722.

Lisa. https://www.surfsara.nl/systems/lisa.

Mavriplis, D. J., 2003. *Revisiting the least-squares procedure for gradient reconstruction on unstructured meshes*. Report NASA/CR-2003-212683, NASA.

Natarajan, G. and F. Sotiropoulos, 2011. "IDeC(k): A new velocity reconstruction algorithm on arbitrarily polygonal staggered meshes." *Journal of Computational Physics* 230: 6583–6604.

Perot, B., 2000. "Conservation properties of unstructured staggered mesh schemes." *J. Comput. Phys.* 159 (1): 58–89. DOI 10.1006/jcph.2000.6424.

Rijn, L. C. van, 1984. "Sediment transport, Part III: bed form and alluvial roughness." *Journal of Hydraulic Engineering* 110 (12): 1733-1754.

Rijn, L. C. van, 2007. "Unified View of Sediment Transport by Currents and Waves. I: Initiation of Motion, Bed Roughness, and Bed-Load Transport." *Journal of Hydraulic Engineering* 133 (6): 649-667.

Rodi, W., 1984. "Turbulence models and their application in Hydraulics, State-of-the-art paper article sur l'etat de connaissance." *IAHR* Paper presented by the IAHR-Section on Fundamentals of Division II: Experimental and Mathematical Fluid Dynamics, The Netherlands.

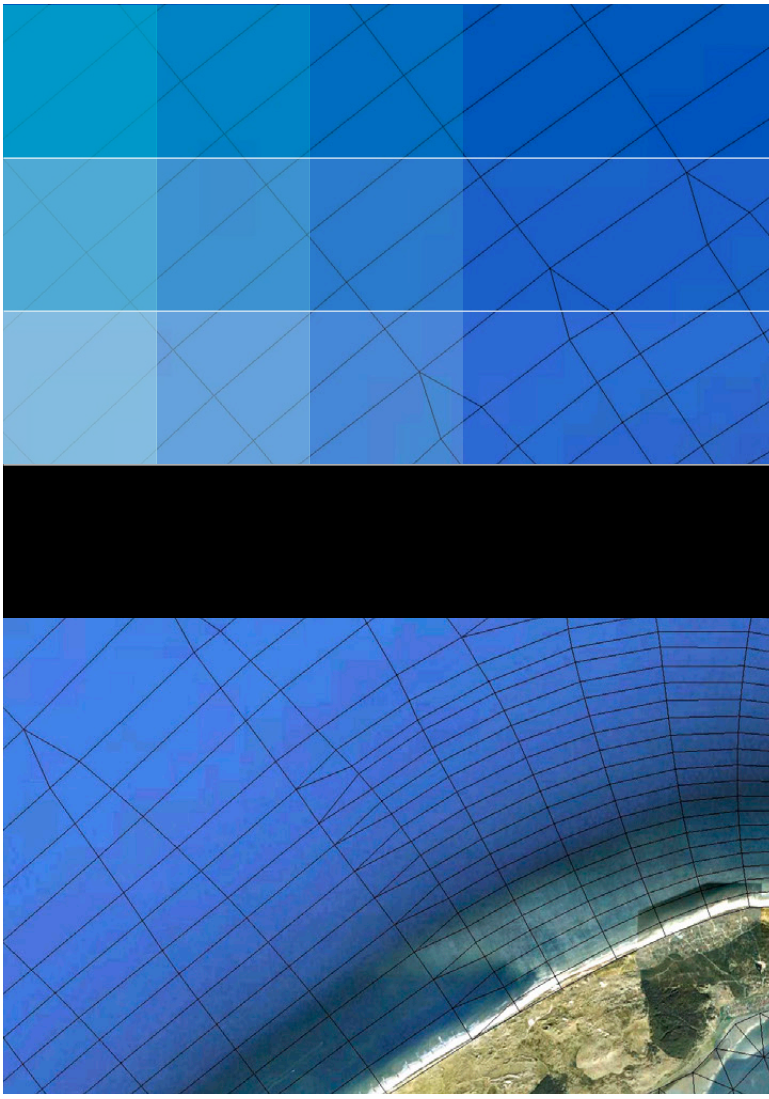
Saad, Y., 1994. “SPARSKIT: a basic tool kit for sparse matrix computations - Version 2.”

Shashkov, M., B. Swartz and W. B., 1998. “Local reconstruction of a vector field from its normal components on the faces of grid cells.” *Journal of Computational Physics* 139: 406–409.

Stelling, G. S. and J. A. T. M. van Kester, 1994. “On the approximation of horizontal gradients in sigma co-ordinates for bathymetry with steep bottom slopes.” *International Journal Numerical Methods In Fluids* 18: 915-955.

Uittenbogaard, R. E., J. A. T. M. van Kester and G. S. Stelling, 1992. *Implementation of three turbulence models in 3D-TRISULA for rectangular grids*. Tech. Rep. Z81, WL | Delft Hydraulics, Delft, The Netherlands.

Yossef, M. and M. Zagonjoli, 2010. *Modelling the hydraulic effect of lowering the groynes on design flood level*. Tech. Rep. 1002524-000, Deltares.



Deltares systems

PO Box 177
2600 MH Delft
Boussinesqweg 1
2629 HV Delft
The Netherlands

+31 (0)88 335 81 88
sales@deltaressystems.nl
www.deltaressystems.nl