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

Delft3D Flexible Mesh Suite

Deltares systems

DRAFT

D-Flow Flexible Mesh

Deltares

Enabling Delta Life

Technical Reference Manual

D-Flow Flexible Mesh

Technical Reference Manual

Released for:
Delft3D FM Suite 2020

Version: 1.1.0
SVN Revision: 66571

23 April 2020

D-Flow Flexible Mesh, Technical Reference Manual

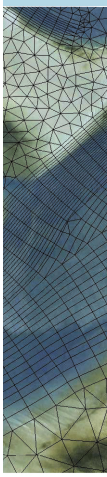
Published and printed by:
Deltares
Boussinesqweg 1
2629 HV Delft
P.O. 177
2600 MH Delft
The Netherlands

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: software@deltares.nl
www: https://www.deltares.nl/software

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

Contents

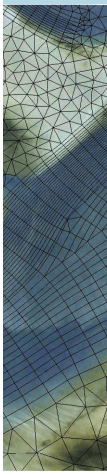


List of Figures

List of Tables

List of Symbols

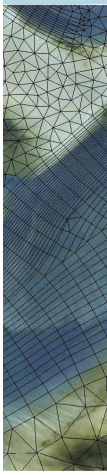
Symbol	Unit	Description
$\hat{h}_j$	m	Hydraulic radius at face j
$\mathcal{J}(k)$	$-$	Set of faces j of cell k
$\mathcal{N}(k)$	$-$	Set of nodes i of cell k
$\vec{n}_j$	$-$	Normal vector on face j of an cell, outward direction is positive
$\vec{u}_j$	m/s	Complete velocity vector at the velocity point on edge j
$\vec{x}_{\zeta_k}$	-	the coordinates of cell-center k
ζ_k	m	Water level at circumcenter of cell k
ζ_{u_j}	m	Water level at the velocity point u_j
A_{u_j}	m^2	Flow area at face j
bl_k	m	Bed level at cell k
h_k	m	Water depth at cell k ($h_k = \zeta_k - bl_k$)
i	$-$	Node counter
j	$-$	Face counter
k	$-$	Cell counter
$L(j)$	$-$	Left cell of face j, giving some orentation to the face
$l(j)$	$-$	Left node of face j, giving some orentation to the face
$R(j)$	$-$	Right cell of face j, giving some orentation to the face
$r(j)$	$-$	Right node of face j, giving some orentation to the face
$s_{j,k}$	$-$	Orientation of face j to cell k
u_j	m/s	Face-normal velocity
v_j	m/s	Tangential velocity component at cell face j
V_k	m^3	Volume of water column at cell k
w_{u_j}	m	Width of face j
z_i	m	Bed level at node i
$\mathcal{A}_{\mathrm{e}j}$	$-$	Explict part of the discretization of the advection and diffusion
$\mathcal{A}_{\mathrm{i}j}$	$-$	Implict part of the discretization of the advection and diffusion
bl_{1_j}	m	Bed level at left node of face j
bl_{2_j}	m	Bed level at right node of face j



1 Problem specification

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

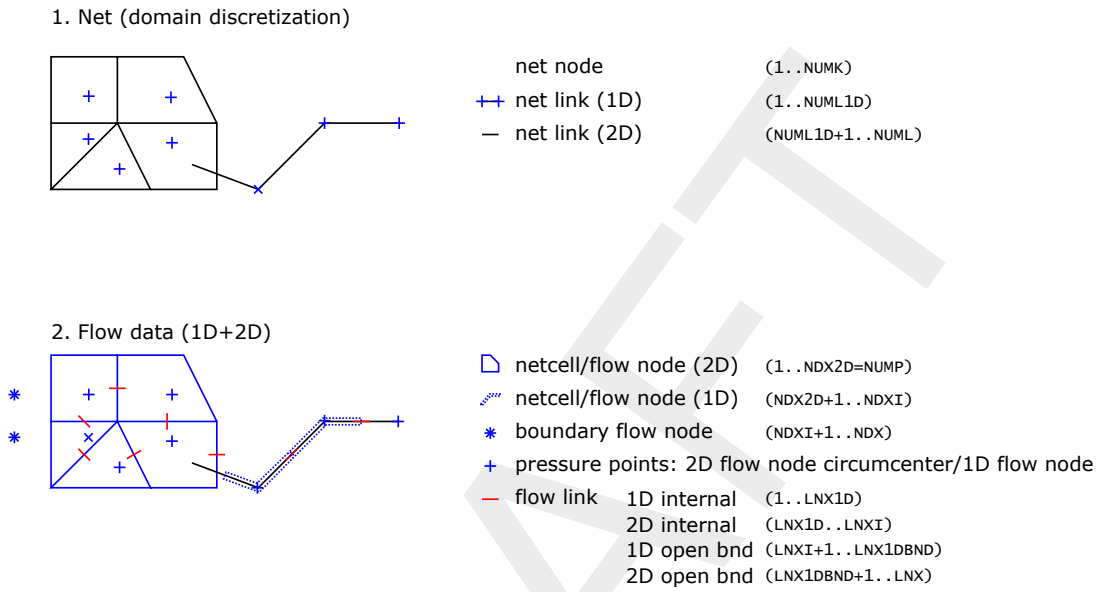
1.1 The master definition file



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



2.2 Implementation details of unstructured nets

2.3 Improve use of cache

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.

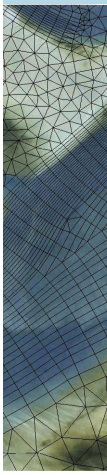
Net nodes can be renumbered, with the net links used as adjacency information. This is only done upon the user's request (*Operations* > *Renumber net nodes*), since net node ordering does not affect the flow simulation times very much. For flow nodes it is done automatically, as part of `flow_geominit()`. 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 i, 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

$$\frac{\partial \boldsymbol{x}}{\partial \xi} \cdot \frac{\partial \boldsymbol{x}}{\partial \eta} = 0, \tag{3.1}$$

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

$$\frac{\partial}{\partial \xi}\left(a \frac{\partial \boldsymbol{x}}{\partial \xi}\right) + \frac{\partial}{\partial \eta}\left(\frac{1}{a} \frac{\partial \boldsymbol{x}}{\partial \eta}\right) = 0, \tag{3.2}$$

$$\nabla \cdot \left(a \frac{\partial \boldsymbol{x}}{\partial \xi}, \frac{1}{a} \frac{\partial \boldsymbol{x}}{\partial \eta}\right)^{\mathrm{T}} = 0 \tag{3.3}$$

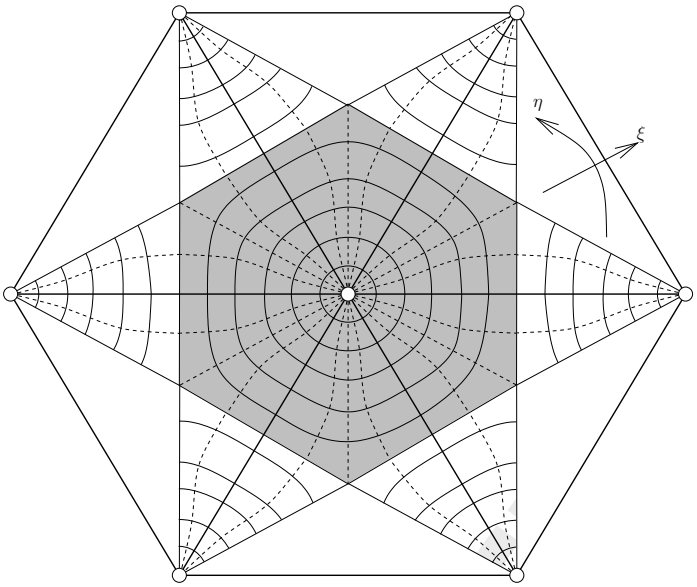


Figure 3.1: *Local grid mapping $\boldsymbol{x}(\xi,\eta)$ around a node for orthogonalization; ξ-lines are dashed; the dual cell is shaded*

where a is the aspect ratio defined by:

$$a = \frac{\left\| \dfrac{\partial \boldsymbol{x}}{\partial \eta} \right\|}{\left\| \dfrac{\partial \boldsymbol{x}}{\partial \xi} \right\|}. \tag{3.4}$$

Equation (3.2) can be written in the following form, after integration over the controle volume Ω and applying the Divergence theorem:

$$\oint_S \left(a\frac{\partial \boldsymbol{x}}{\partial \xi}, \frac{1}{a}\frac{\partial \boldsymbol{x}}{\partial \eta} \right)^{\mathrm{T}} \cdot \boldsymbol{n}\, \mathrm{d}S = \oint_S \left(a\frac{\partial \boldsymbol{x}}{\partial \xi} n_\xi + \frac{1}{a}\frac{\partial \boldsymbol{x}}{\partial \eta} n_\eta \right) \mathrm{d}S = 0, \tag{3.5}$$

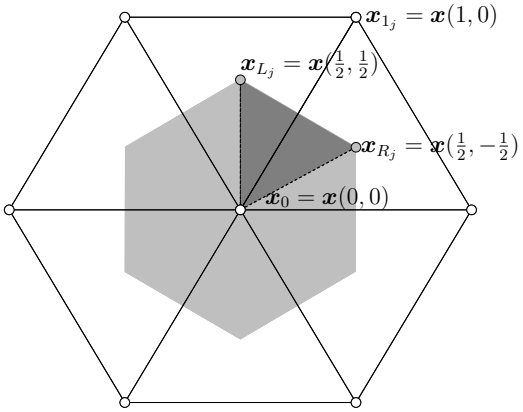
where S is the boundary of the control volume Ω in (ξ,η) space and $\boldsymbol{n} = (n_\xi, n_\eta)^{\mathrm{T}}$ is the outward orthonormal vector.

3.5.1 Discretization

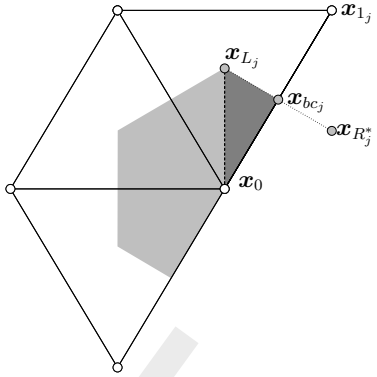
For the description of the discretization of Equation (3.4) and Equation (3.5) the following nomenclature is used:

Definition 3.5.2. *$\mathcal{J}_{int}$ is the set of* internal *primary edges connected to node i and $\boldsymbol{x}_0$ and $\boldsymbol{x}_{1_j}$ are the coordinates of that node and of the neighboring node connected through edge j, respectively. Furthermore, $\boldsymbol{x}_{L_j}$ and $\boldsymbol{x}_{R_j}$ are the left and right neighboring cell-circumcentre coordinates, see Figure 3.2a.*

Definition 3.5.3. *$\mathcal{J}_{bnd}$ is the set of* boundary *edges (nonempty if node i is on the grid boundary only), $\boldsymbol{x}_{R_j^*}$ are the coordinates of a virtual node and $\boldsymbol{x}_{bc_j}$ are boundary node coordinates, see Figure 3.2b.*



(a) *Internal edge*



(b) *Boundary edge; $\boldsymbol{x}_{bc_j}$ are the coordinates of a node projected onto the grid boundary, $\boldsymbol{x}_{R_j^*}$ are virtual node coordinates*

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

The discretizations of the aspect ratio for edge j, Equation (3.4), with $\Delta\xi = \Delta\eta = 1$ yields

$$a_j \approx \frac{\left\|\boldsymbol{x}_{L_j} - \boldsymbol{x}_{R_j}\right\|}{\Delta\eta}\frac{\Delta\xi}{\|\boldsymbol{x}_{1j} - \boldsymbol{x}_0\|} = \frac{\left\|\boldsymbol{x}_{L_j} - \boldsymbol{x}_{R_j}\right\|}{\|\boldsymbol{x}_{1j} - \boldsymbol{x}_0\|}, \qquad j \in \mathcal{J}_{int} \tag{3.6}$$

and the discretization of Equation (3.5) yields

$$\begin{aligned}\sum_{j\in\mathcal{J}_i} \frac{\|\boldsymbol{x}_{R_j} - \boldsymbol{x}_{L_j}\|}{\|\boldsymbol{x}_{1_j} - \boldsymbol{x}_0\|}(\boldsymbol{x}_{1_j} - \boldsymbol{x}_0) \quad + \\ \sum_{j\in\mathcal{J}_e}\left\{\frac{1}{2}\frac{\|\boldsymbol{x}_{R_j^*} - \boldsymbol{x}_{L_j}\|}{\|\boldsymbol{x}_{1_j} - \boldsymbol{x}_0\|}(\boldsymbol{x}_{1_j} - \boldsymbol{x}_0) \quad + \quad \frac{1}{2}\frac{\|\boldsymbol{x}_{1_j} - \boldsymbol{x}_0\|}{\|\boldsymbol{x}_{R_j^*} - \boldsymbol{x}_{L_j}\|}(\boldsymbol{x}_{R_j^*} - \boldsymbol{x}_{L_j})\right\} \quad = \quad 0,\end{aligned} \tag{3.7}$$

where the second summation in Equation (3.7) accounts for boundary edges.

The virtual node $\boldsymbol{x}_{R_j^*}$ is constructed by extrapolation from the circumcenter and boundary nodes, using Equation (3.8) to project the left cell-circumcenter orthogonally onto the grid boundary $\boldsymbol{x}_{bc_j}$:

$$\boldsymbol{x}_{bc_j} = \boldsymbol{x}_0 + \frac{(\boldsymbol{x}_{1_j} - \boldsymbol{x}_0)\bullet(\boldsymbol{x}_{L_j} - \boldsymbol{x}_0)}{\|\boldsymbol{x}_{1_j} - \boldsymbol{x}_0\|^2}(\boldsymbol{x}_{1_j} - \boldsymbol{x}_0). \tag{3.8}$$

$$\boldsymbol{x}_{R_j^*} = 2\boldsymbol{x}_{bc_j} - \boldsymbol{x}_{L_j}, \tag{3.9}$$

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

Finally, Equation (3.7) can be put in the following form

$$\sum_{j\in\mathcal{J}_i} a_j\left(\boldsymbol{x}_{1_j} - \boldsymbol{x}_0\right) \quad + \quad \sum_{l\in\mathcal{L}_e}\left\{\frac{1}{2}a_j\left(\boldsymbol{x}_{1_j} - \boldsymbol{x}_0\right) \quad + \quad \frac{1}{2}\|\boldsymbol{x}_{1_j} - \boldsymbol{x}_0\|\mathbf{n}_j,\right\} \quad = \quad 0,$$

(3.10)

where $\mathbf{n}_j = \frac{\boldsymbol{x}_{R_j^*} - \boldsymbol{x}_{L_j}}{\|\boldsymbol{x}_{R_j^*} - \boldsymbol{x}_{L_j}\|}$ is the outward normal at edge j and a_j is the aspect ratio of edge j, i.e.

$$a_j = \begin{cases} \dfrac{\|\boldsymbol{x}_{R_j} - \boldsymbol{x}_{L_j}\|}{\|\boldsymbol{x}_{1_j} - \boldsymbol{x}_0\|}, & j \in \mathcal{J}_i, \\ \dfrac{\|\boldsymbol{x}_{R_j^*} - \boldsymbol{x}_{L_j}\|}{\|\boldsymbol{x}_{1_j} - \boldsymbol{x}_0\|}, & j \in \mathcal{J}_e. \end{cases} \tag{3.11}$$

3.5.2 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 (3.10) the aspect ratio of Equation (3.11) is replaced by

$$a_j = \begin{cases} \dfrac{4\,\|\boldsymbol{x}_{R_j} - \boldsymbol{x}_{L_j}\|}{2\|\boldsymbol{x}_{1_j} - \boldsymbol{x}_0\| + \|\boldsymbol{x}_{2R_j} - \boldsymbol{x}_{1R_j}\| + \|\boldsymbol{x}_{2L_j} - \boldsymbol{x}_{1L_j}\|}, & j \in \mathcal{J}_{int}, \\ \dfrac{2\,\|\boldsymbol{x}_{R_j^*} - \boldsymbol{x}_{L_j}\|}{\|\boldsymbol{x}_{1_j} - \boldsymbol{x}_0\| + \|\boldsymbol{x}_{2L_j} - \boldsymbol{x}_{1L_j}\|}, & j \in \mathcal{J}_{bnd}, \end{cases} \tag{3.12}$$

where the nodes involved are depicted in Figure 3.3.

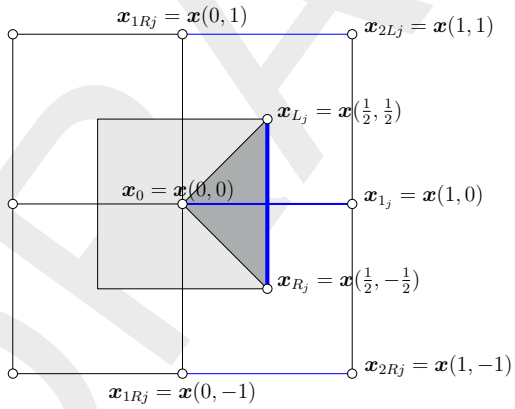


Figure 3.3: *Part of the control volume that surrounds edge j (dark shading) and the nodes involved; quadrilateral grid cells; edges used in Equation (3.12) are coloured blue*

3.6 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

$$\nabla \cdot \left(G^{-1} \nabla \xi^i\right) = 0, \quad i \in \{1, 2\}, \tag{3.13}$$

where G is the monitor function for grid adaptivity which will be explained later (section 3.6.3) and $\xi^1 \equiv \xi$, $\xi^2 \equiv \eta$.

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 $\boldsymbol{x}(\xi, \eta)$.

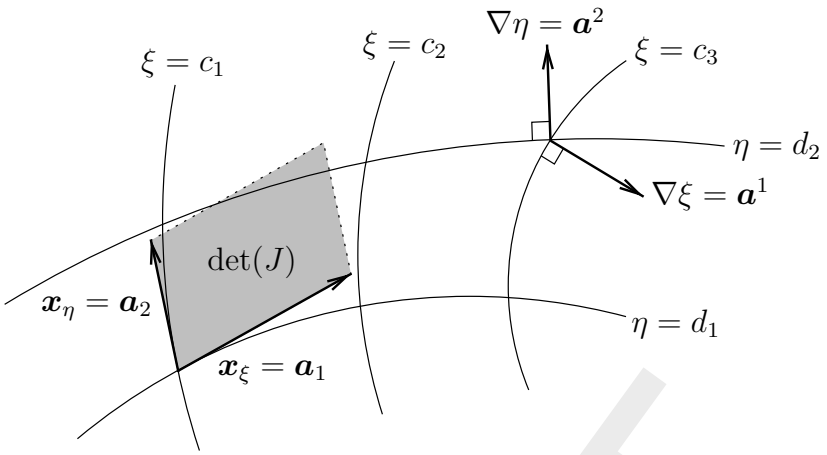


Figure 3.4: *Curvilinear coordinate mapping on a planar domain. The tangent and normal vectors are not necessarily up to scale (Van Dam, 2009).*

By interchanging the role of dependent and independent variables, Equation (3.13) can be transformed into an expression for the direct grid mapping $\boldsymbol{x}(\xi, \eta)$:

$$\begin{aligned}
&\frac{\partial \boldsymbol{x}}{\partial \xi}\left\langle \boldsymbol{a}^1, \frac{\partial\left(G^{-1}\right)}{\partial \xi}\boldsymbol{a}^1\right\rangle + \frac{\partial \boldsymbol{x}}{\partial \eta}\left\langle \boldsymbol{a}^1, \frac{\partial\left(G^{-1}\right)}{\partial \xi}\boldsymbol{a}^2\right\rangle + \\
&\frac{\partial \boldsymbol{x}}{\partial \xi}\left\langle \boldsymbol{a}^2, \frac{\partial\left(G^{-1}\right)}{\partial \eta}\boldsymbol{a}^1\right\rangle + \frac{\partial \boldsymbol{x}}{\partial \eta}\left\langle \boldsymbol{a}^2, \frac{\partial\left(G^{-1}\right)}{\partial \eta}\boldsymbol{a}^2\right\rangle \\
&- \left(\left\langle \boldsymbol{a}^1, G^{-1}\boldsymbol{a}^1\right\rangle \frac{\partial^2 \boldsymbol{x}}{\partial \xi^2} + \left\langle \boldsymbol{a}^1, G^{-1}\boldsymbol{a}^2\right\rangle \frac{\partial^2 \boldsymbol{x}}{\partial \xi \partial \eta} + \right. \\
&\qquad \left.\left\langle \boldsymbol{a}^2, G^{-1}\boldsymbol{a}^1\right\rangle \frac{\partial^2 \boldsymbol{x}}{\partial \eta \partial \xi} + \left\langle \boldsymbol{a}^2, G^{-1}\boldsymbol{a}^2\right\rangle \frac{\partial^2 \boldsymbol{x}}{\partial \eta^2}\right) = 0,
\end{aligned} \tag{3.14}$$

where by $\langle \bullet, \bullet \rangle$ an inner product is meant and $\boldsymbol{a}^1 = \nabla\xi$ and $\boldsymbol{a}^2 = \nabla\eta$ are the contravariant base vectors (Figure 3.4), by definition:

$$\boldsymbol{a}_\alpha \cdot \boldsymbol{a}^\beta = \delta_\alpha^\beta, \qquad \alpha, \beta \in \{1, 2\} \tag{3.15}$$

and thus

$$\|\boldsymbol{a}_\gamma\| = \frac{1}{\|\boldsymbol{a}^\gamma\|}, \qquad \gamma \in \{1, 2\} \tag{3.16}$$

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

3.6.1 Assigning the node coordinates in computational space

By assigning the node coordinates in computational (ξ, η)-space, 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:

$$\boldsymbol{x} = \boldsymbol{x}_0 + J(\boldsymbol{\xi} - \boldsymbol{\xi}_0) + \mathcal{O}\left(\|\boldsymbol{\xi}\|^2\right), \tag{3.17}$$

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):

$$J = U\Sigma V^{\mathrm{T}}, \tag{3.18}$$

where V^{T} is a rotation in (ξ, η)-space, Σ a compression/expansion and U a rotation in (x, y)-space, see Figure 3.5.

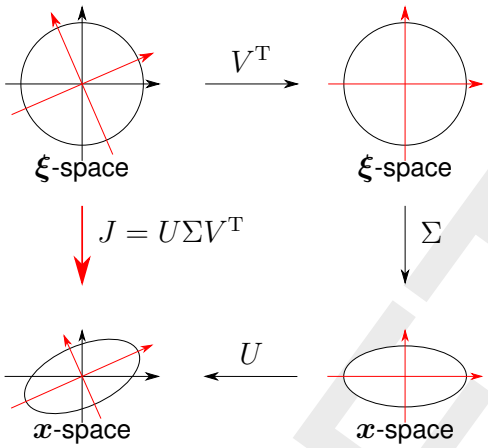


Figure 3.5: *Geometric meaning of the singular value decomposition of Jacobian matrix J (Huang, 2005, fig. 2.2)*

Since Equation (3.14) is invariant to rotation of the (ξ, η)-axis, rotation V is irrelevant and we may start by assigning $\boldsymbol{\xi} = (0, 0)^{\mathrm{T}}$ to the center node i and $\boldsymbol{\xi} = (1, 0)^{\mathrm{T}}$ to an arbitrary neighboring node.

We now proceed by considering a cell attached to a node i in coordinate frame (ξ', η'), see Figure 3.6, and define an *optimal* angle Φ^{opt} between two subsequent edges that are connected to node i.

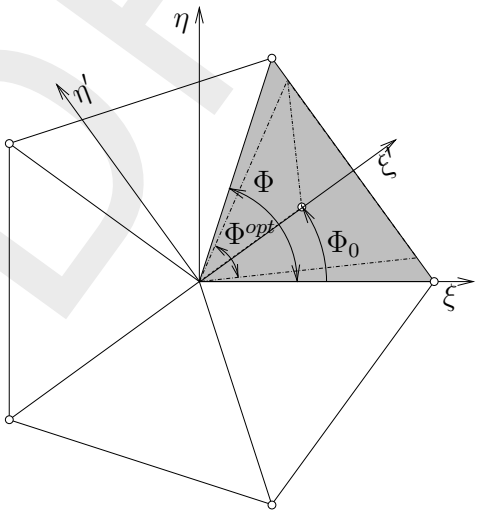


Figure 3.6: *non-rectangular triangular cell; the dashed cell is an optimal equiangular polygon, while the shaded cell is the resulting cell after scaling in η' direction; Φ_0 is the angle of the ξ'-axis in the (ξ, η)-frame*

Definition 3.6.2. *The optimal angle Φ^{opt} is the angle between two subsequent edges of a cell, both connected to node i, 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 for rectangular triangles. The optimal angle at a node of a rectangular triangle is either $\frac{1}{4}\pi$ or $\frac{1}{2}\pi$, depending on the grid connectivity.

For a non-rectangular triangle this optimal angle would be $\frac{1}{3}\pi$. However, by considering a node with five non-rectangular 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 Φ is the angle between two subsequent edges of a cell, both connected to node i, 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 (section 3.6.1.2).

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

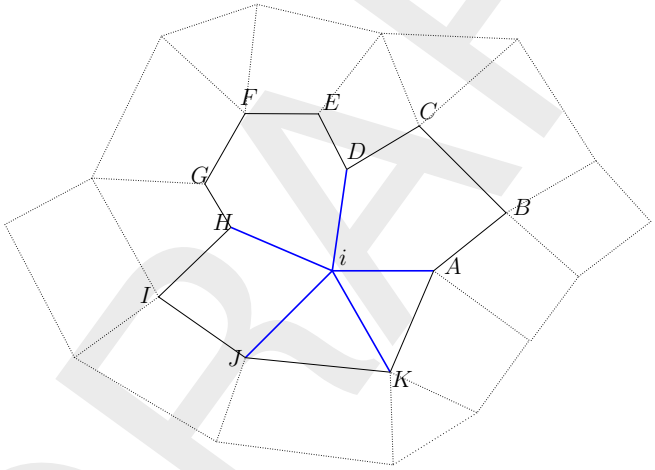


Figure 3.7: *The stencil for node i formed by the nodes $A, \ldots, K$. Node D and H are rectangular nodes. The node angle is between two subsequent blue edges.*

Definition 3.6.5. *The* stencil *is the set of cells that are connected to node i. A node angle is the angle between two subsequent stencil-boundary edges connected to node i. A rectangular node, not being node i itself, is a node that is connected to three or less non-stencil quadrilateral cells and no other non-stencil nodes. A rectangular cell is a quadrilateral cell or a triangular cell which contains at least one right angle.*

Note: Each node of a rectangular cell will be called a rectangular node. So one of the optimal angles Φ^{opt} of such a cell is rectangular. Rectangular nodes have optimal node angles as indicated in Figure 3.8, which can be $\frac{1}{4}\pi$, $\frac{1}{2}\pi$, π or $\frac{3}{2}\pi$. It will be indicated with the sub-script $rect$.

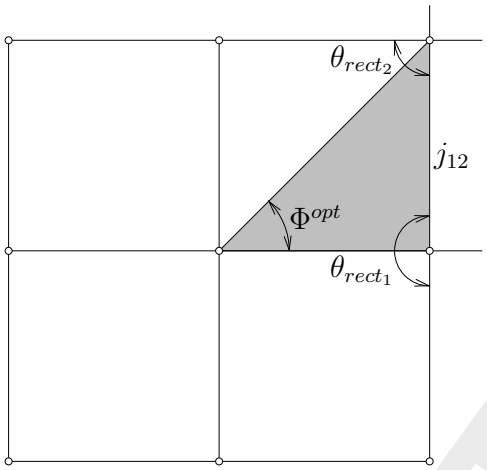


Figure 3.8: *Rectangular triangle cell; additional node angles θ_{rect1} and θ_{rect2} and edge j_{12} are used to determine optimal angle Φ^{opt}*

The rectangular node angles are computed by

$$\theta_{rect_i} = \begin{cases} (2 - \frac{1}{2}N_{nsq})\,\pi, & \text{node } i \text{ is a rectangular internal node,} \\ (1 - \frac{1}{2}N_{nsq})\,\pi, & \text{node } i \text{ is a rectangular boundary node,} \\ \frac{1}{2}\pi, & \text{node } i \text{ is a rectangular corner node,} \end{cases} \tag{3.19}$$

where N_{nsq} is the number of non-stencil quadrilaterals connected to node i. The optimal angle Φ^{opt} for rectangular nodes is finally determined by (see Figure 3.8)

$$\Phi^{opt} = \begin{cases} (1 - \frac{2}{N})\,\pi, & N \geq 4 \quad \vee \quad \text{non-rectangular cell,} \\ \frac{1}{2}\pi, & N = 3 \quad \wedge \quad \text{rectangular cell with two rectangular nodes '1' and '2'} \\ & \quad \wedge \quad \theta_{rect_1} + \theta_{rect_2} = \pi \\ & \quad \wedge \quad j_{12} \text{ is not a boundary edge,} \\ \frac{1}{4}\pi, & \text{other,} \end{cases} \tag{3.20}$$

where N is the number of nodes that comprise the cell. In the example of Figure 3.8, nodes 1 and 2 are rectangular nodes with angles of π and $\frac{1}{2}\pi$ respectively and the shaded cell is a rectangular 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 i.

Definition 3.6.6. *The sum of all cell-optimal and true angles are called $\Sigma\Phi^{opt}$ and $\Sigma\Phi$ respectively. Furthermore, the sum of all optimal and true angles of quadrilateral cells are called $\Sigma\Phi^{opt}_{quad}$ and $\Sigma\Phi_{quad}$ respectively. The number of quadrilateral cells is N_{quad}. The same definitions hold for the rectangular triangular cells: $\Sigma\Phi^{opt}_{tri_{rect}}$, $\Sigma\Phi_{tri_{rect}}$ and $N_{tri_{rect}}$ respectively and for the remaining cells: $\Sigma\Phi^{opt}_{rem}$, $\Sigma\Phi_{rem}$ and N_{rem} respectively.*

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

Of course holds

$$\Sigma\Phi^{opt} = \Sigma\Phi^{opt}_{quad} + \Sigma\Phi^{opt}_{tri_{rect}} + \Sigma\Phi^{opt}_{rem}, \tag{3.21}$$

$$N = N_{quad} + N_{tri_{rect}} + N_{rem}. \tag{3.22}$$

In a similar fashion, the sum of all true angles should sum up to $2\pi f$, where

$$f = \begin{cases} 1, & \text{internal node,} \\ \frac{1}{2}, & \text{boundary node,} \\ \frac{1}{4}, & \text{corner node.} \end{cases} \tag{3.23}$$

In other words, we seek true angles $\Sigma\Phi_{quad}$, $\Sigma\Phi_{tri_{rect}}$ and $\Sigma\Phi_{rem}$ such that:

$$\Sigma\Phi_{quad} + \Sigma\Phi_{tri_{rect}} + \Sigma\Phi_{rem} = 2\pi f. \tag{3.24}$$

This is achieved by setting

$$\Sigma\Phi_{quad} = \mu \quad \Sigma\Phi^{opt}_{quad}, \tag{3.25}$$

$$\Sigma\Phi_{tri_{rect}} = \mu \ \mu_{tri_{rect}} \Sigma\Phi^{opt}_{tri_{rect}}, \tag{3.26}$$

$$\Sigma\Phi_{rem} = \mu \ \mu_{rem} \Sigma\Phi^{opt}_{rem}. \tag{3.27}$$

We give highest precedence to the optimal angles of quadrilateral cells, followed by rectangular triangular cells and lowest precedence to the remaining cells. From the angle left for the remaining cells (non-rectangular triangles, pentagons and hexagons) the coefficient μ_{rem} can be determined (with a lower band):

$$\mu_{rem} = \max\left(\frac{2\pi\, f - \left(\Sigma\Phi^{opt}_{quad} + \Sigma\Phi^{opt}_{tri_{rect}}\right)}{\Sigma\Phi^{opt}_{rem}}, \frac{N_{tri}\Phi_{min}}{\Sigma\Phi^{opt}_{rem}} \right), \tag{3.28}$$

If there are remaining cells ($N_{rem} > 0$) then $\mu_{tri_{rect}} = 1$ and if there are no remaining cells $\mu_{rem} = 1$ and $\Sigma\Phi^{opt}_{rem} = 0$ and does not influence the angles available for quads and rectangular triangles. So:

$$\mu_{tri_{rect}} = \begin{cases} 1 & N_{rem} > 0 \\ \max\left(\dfrac{2\pi\, f - \Sigma\Phi^{opt}_{quad}}{\Sigma\Phi^{opt}_{tri_{rect}}}, \dfrac{N_{tri_{rect}}\Phi_{min}}{\Sigma\Phi^{opt}_{tri_{rect}}} \right), & N_{rem} = 0, \end{cases} \tag{3.29}$$

At last μ is determined by taken all cells into account

$$\mu = \frac{2\pi\, f}{\Sigma\Phi^{opt}_{quad} + \mu_{tri_{rect}}\Sigma\Phi^{opt}_{tri_{rect}} + \mu_{rem}\Sigma\Phi^{opt}_{rem}}. \tag{3.30}$$

$\Phi_{min} = \frac{1}{12}\pi$ is the minimum cell angle, determining a lower band for the factors $\mu_{tri_{rect}}$ and μ_{rem}.

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 i have unit length in computational (ξ, η)-space, which has its consequences for rectangular triangles.

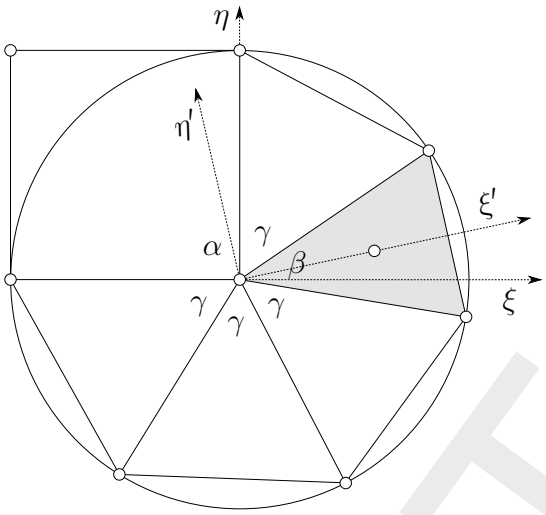
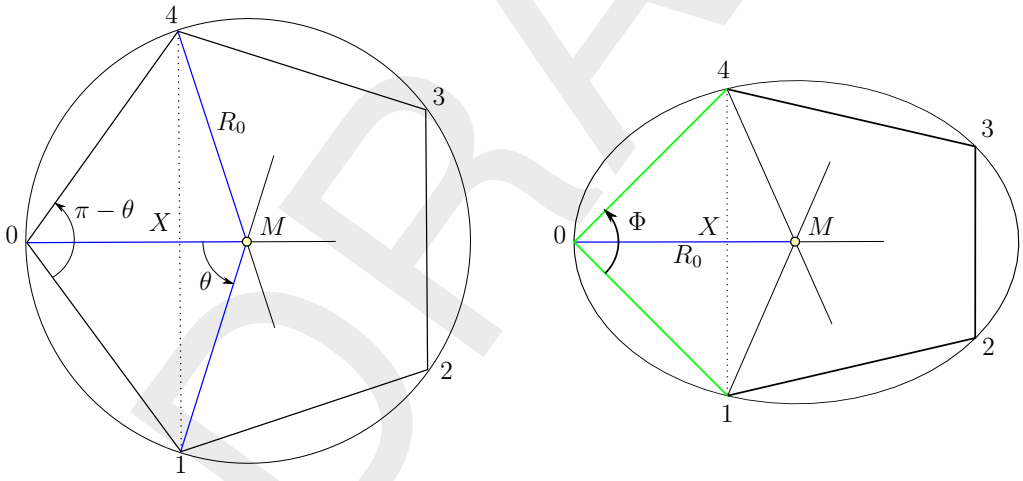


Figure 3.9: *Computational coordinates for one quadrilateral and five triangular cells, one of which is a rectangular (shaded) before transformation to (ξ, η)-coordinates. $\alpha = \frac{1}{2}\pi$, $\beta = \frac{1}{4}\pi$ and $\gamma = \frac{5}{4}\pi/4$.*

Remark 3.6.8. Since all edges connected to node i are required to have unit length, rectangular triangles may be transformed into non-rectangular triangles, but maintain their cell angle Φ^{opt}, see Figure 3.9 for an example.



(a) *Nodes on circle.* ***(b)*** *Nodes on ellipse.*

Figure 3.10: *The circle in Figure 3.10a is squeezed in vertical direction (i.e. $\perp OM$) to obtain the ellipse in Figure 3.10b. Blue: $d(M,0) = d(M,1) = d(M,4) = R_0$; Green: $d(0,1) = d(0,4) = 1$.*

Since the cell in (ξ', η') coordinates is an equiangular polygon in (ξ', η')-space (Figure 3.10a), the coordinates of the i^{th} node is

$$\xi' = \quad R_0\,(1 - \cos(i\,\theta)), \tag{3.31}$$

$$\eta' = -R_0\,\sin(i\,\theta), \tag{3.32}$$

$$\theta = \frac{2\pi}{N} \tag{3.33}$$

where $i = 0$ corresponds to the center node i and counting counterclockwise, N is the number of nodes that comprise the cell and R_0 the radius of the circumcircle, see Figure 3.10a (node $\boldsymbol{\xi}_{i-1} = i - 1$ and $i = 1, \ldots, N$). The circle in Figure 3.10a is squeezed in such a way that the edge from node 0 to node 1 and node 0 to node 4 has length 1 (distance: $d(0,1) = d(0,4) = 1$) and Φ is the true angle, also $d(0,X)$ remains the same $\left(R_0(1-\cos\theta) = \cos(\frac{1}{2}\Phi)\right)$ because the squeezing is perpendicular to the OM-axis (i.e. the ξ'-axis). The other edges of the polygon does not have, in general, a length of 1 after squeezing. The radius of the circumcircle read:

$$R_0 = \frac{\cos(\frac{1}{2}\Phi)}{(1-\cos\theta)} \tag{3.34}$$

The cell aspect ratio A is defined as the ratio between the distance $d(1,N)$ in Figure 3.10b and the distance $d(1,N)$ in Figure 3.10a, yielding:

$$A = \frac{(1-\cos\theta)\ \tan(\frac{1}{2}\Phi)}{\sin\theta}, \tag{3.35}$$

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

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 3.6. The transformation from (ξ', η') to (ξ, η) coordinates read:

$$\xi = \cos(\Phi_0)\ \xi' - A\ \sin(\Phi_0)\ \eta', \tag{3.36}$$
$$\eta = \sin(\Phi_0)\ \xi' + A\ \cos(\Phi_0)\ \eta', \tag{3.37}$$

3.6.2 Computing the operators

For the solution of Equation (3.14), we approximate $\dfrac{\partial^2 \boldsymbol{x}}{\partial\xi\partial\eta}$ at node $\boldsymbol{\xi}_0$

$$\left.\frac{\partial^2 \boldsymbol{x}}{\partial\xi\partial\eta}\right|_{\boldsymbol{\xi}_0} \approx \sum_{j\in\mathcal{J}} D_\xi \left(\sum_{i\in\mathcal{N}} G_\eta \boldsymbol{x}_i\right)_j, \tag{3.38}$$

and similar for the other derivatives $\dfrac{\partial^2 \boldsymbol{x}}{\partial\xi^2}$, $\dfrac{\partial^2 \boldsymbol{x}}{\partial\eta^2}$ and $\dfrac{\partial^2 \boldsymbol{x}}{\partial\eta\partial\xi}$, where:

Definition 3.6.9. *$\mathcal{J}$ is the set of edges attached to node $\boldsymbol{\xi}_0$ and $\mathcal{N}$ is the set of nodes in the stencil of node $\boldsymbol{\xi}_0$. 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

$$\nabla_\xi \Phi \approx \frac{1}{\text{vol}(\Omega_\xi)} \oint_{\partial\Omega_\xi} \Phi\, \boldsymbol{n}_\xi \,\mathrm{d}S_\xi. \tag{3.39}$$

3.6.2.1 Node-to-edge operator

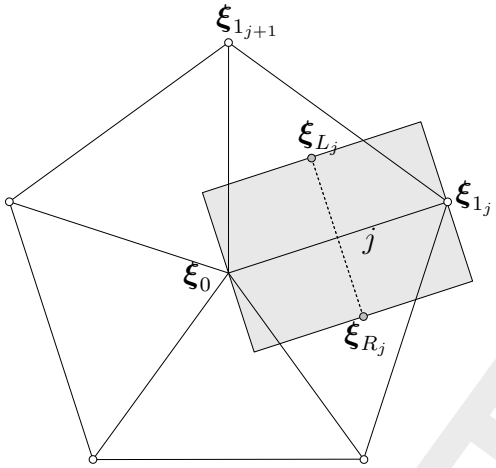


Figure 3.11: *Control volume for computing the* node-to-edge *gradient at edge j discrete for the discrete operators G_ξ, G_ξ*

For the *node-to-edge* gradient $(G_\xi, G_\eta)^{\mathrm{T}}$ we take the control volume as indicated in Figure 3.11 and obtain for some node-based quantity Φ

$$(G_\xi, G_\eta)^{\mathrm{T}}\,\Phi\Big|_j = \begin{cases} \dfrac{(\boldsymbol{\xi}_{R_j} - \boldsymbol{\xi}_{L_j})^\perp(\Phi_{1_j} - \Phi_0) \;-\; (\boldsymbol{\xi}_{1_j} - \boldsymbol{\xi}_0)^\perp(\Phi_{R_j} - \Phi_{L_j})}{\|(\boldsymbol{\xi}_{1_j} - \boldsymbol{\xi}_0) \times (\boldsymbol{\xi}_{R_j} - \boldsymbol{\xi}_{L_j})\|}, & j \in \mathcal{J}_{int}, \\ \dfrac{(\boldsymbol{\xi}_{R_j^*} - \boldsymbol{\xi}_{L_j})^\perp(\Phi_{1_j} - \Phi_0) \;-\; (\boldsymbol{\xi}_{1_j} - \boldsymbol{\xi}_0)^\perp(\Phi_{R_j^*} - \Phi_{L_j})}{\|(\boldsymbol{\xi}_{1_j} - \boldsymbol{\xi}_0) \times (\boldsymbol{\xi}_{R_j} - \boldsymbol{\xi}_{L_j})\|}, & j \in \mathcal{J}_{bnd}, \end{cases} \tag{3.40}$$

where we use similar definitions as Definitions 3.5.2 and 3.5.3, and Remark 3.5.4 also holds. Furthermore, $\boldsymbol{\xi} \cdot \boldsymbol{\xi}^\perp = 0 \Rightarrow \boldsymbol{\xi}^\perp = (-\eta, \xi)^{\mathrm{T}}$, so $\boldsymbol{\xi}^\perp$ is parallel to the contravariant vector $\boldsymbol{a}^2$ ($\boldsymbol{\xi} = \boldsymbol{a}_1$ and $\boldsymbol{a_1} \cdot \boldsymbol{a}^2 = 0$) and $\boldsymbol{\xi}_0 = \boldsymbol{0}$ by construction. Because the values $\boldsymbol{\xi}_{L_j}$ and $\boldsymbol{\xi}_{R_j}$ are not node based, the value at the circumcentres need to be determined from the node values of that cell.

Determine the value at circumcenters

The cell circumcenters $\boldsymbol{\xi}_{L_j}$ and $\boldsymbol{\xi}_{R_j}$ can be expressed in general form as

$$\boldsymbol{\xi}_{L_j} = \sum_{i \in \mathcal{N}} A^i_{L_j}\, \boldsymbol{\xi}_i, \tag{3.41}$$

$$\boldsymbol{\xi}_{R_j} = \boldsymbol{\xi}_{L_{j-1}}, \tag{3.42}$$

Definition 3.6.10. *A_{L_j} is the left* node-to-cell *mapping for the cell left from edge j.*

The above summation is over all nodes, the coefficient $A^i_{L_j} = 0$ if the node i does not belong to the left cell of edge j.

Circumcenter of non triangle

For a non-triangle cell k the centroid is taken as an approximation of the circumcenter. So:

$$A_{L_j} = \frac{1}{N}, \qquad L(j) = k \tag{3.43}$$

where N is the number of vertices of the cell k.

Circumcenter of a triangle

For triangular cells on the other hand, the circumcenter is used and computed as follows:

$$\boldsymbol{\xi}_{L_j} = \boldsymbol{\xi}_0 + \alpha(\boldsymbol{\xi}_{1_j} - \boldsymbol{\xi}_0) + \beta(\boldsymbol{\xi}_{1_{j+1}} - \boldsymbol{\xi}_0), \tag{3.44}$$

$$\boldsymbol{\xi}_{R_j} = \boldsymbol{\xi}_{L_{j-1}}, \tag{3.45}$$

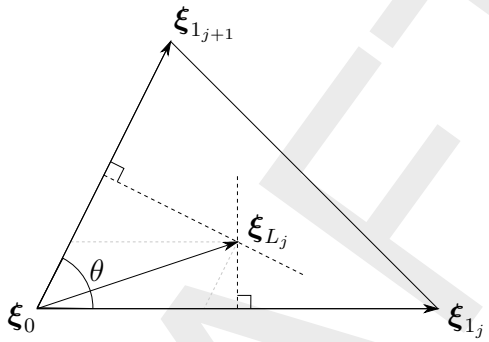


Figure 3.12: *Sketch for the computation of the cirumcentre of a triangle*

where

$$\alpha = \frac{1 - \frac{1}{\gamma}c}{2(1-c^2)}, \tag{3.46}$$

$$\beta = \frac{1 - \gamma c}{2(1-c^2)}, \tag{3.47}$$

and

$$\gamma = \frac{\|\boldsymbol{\xi}_{1_j} - \boldsymbol{\xi}_0\|}{\|\boldsymbol{\xi}_{1_{j+1}} - \boldsymbol{\xi}_0\|}, \tag{3.48}$$

$$c = \frac{(\boldsymbol{\xi}_{1_j} - \boldsymbol{\xi}_0) \cdot (\boldsymbol{\xi}_{1_{j+1}} - \boldsymbol{\xi}_0)}{\|\boldsymbol{\xi}_{1_j} - \boldsymbol{\xi}_0\| \, \|\boldsymbol{\xi}_{1_{j+1}} - \boldsymbol{\xi}_0\|} \qquad (= \cos\theta). \tag{3.49}$$

Remark 3.6.11. The edges j around node i are arranged in counterclockwise order.

The circumcenter of a triangle expressed in the vertex coordinates (Equation (3.41)) read:

$$\boldsymbol{\xi}_{L_j} = (1 - \alpha - \beta)\boldsymbol{\xi}_0 + \alpha\boldsymbol{\xi}_{1_j} + \beta\boldsymbol{\xi}_{1_{j+1}}, \tag{3.50}$$

$$\boldsymbol{\xi}_{R_j} = \boldsymbol{\xi}_{L_{j-1}}, \tag{3.51}$$

The cell center values Φ in Equation (3.40) are computed in the same manner, i.e.:

$$\Phi_{L_j} = \sum_{i \in \mathcal{N}} A^i_{L_j} \Phi_i, \tag{3.52}$$

$$\Phi_{R_j} = \Phi_{L_{j-1}}. \tag{3.53}$$

Operator G_ξ ***and*** G_η

Combining Equation (3.40), Equation (3.52) and Equation (3.53) yields for each internal edges j

$$G_\xi \Phi|_j = \frac{-(\eta_{R_j} - \eta_{L_j})(\Phi_{1_j} - \Phi_0) + (\eta_{1_j} - \eta_0) \sum_{i \in \mathcal{N}} (A^i_{L_{j-1}} \Phi_i - A^i_{L_j} \Phi_i)}{\|(\boldsymbol{\xi}_{1_j} - \boldsymbol{\xi}_0) \times (\boldsymbol{\xi}_{R_j} - \boldsymbol{\xi}_{L_j})\|}, \quad j \in \mathcal{J}_{int}, \tag{3.54}$$

and

$$G_\eta \Phi|_j = \frac{(\xi_{R_j} - \xi_{L_j})(\Phi_{1_j} - \Phi_0) - (\xi_{1_j} - \xi_0) \sum_{i \in \mathcal{N}} (A^i_{L_{j-1}} \Phi_i - A^i_{L_j} \Phi_i)}{\|(\boldsymbol{\xi}_{1_j} - \boldsymbol{\xi}_0) \times (\boldsymbol{\xi}_{R_j} - \boldsymbol{\xi}_{L_j})\|}, \quad j \in \mathcal{J}_{int}, \tag{3.55}$$

Boundary edges are treated in a similar fashion as before, see Equation (3.9), by creating a virtual node:

$$\boldsymbol{\xi}_{R^*_j} = 2\boldsymbol{\xi}_{bc_j} - \boldsymbol{\xi}_{L_j}, \tag{3.56}$$
$$\Phi_{R^*_j} = 2\Phi_{bc_j} - \Phi_{L_j}, \tag{3.57}$$

and

$$\boldsymbol{\xi}_{bc_j} = \boldsymbol{\xi}_0 + \alpha_\xi (\boldsymbol{\xi}_{1_j} - \boldsymbol{\xi}_0), \tag{3.58}$$
$$\Phi_{bc_j} = \Phi_0 + \alpha_x (\Phi_{1_j} - \Phi_0), \tag{3.59}$$

with

$$\alpha_\xi = (\boldsymbol{\xi}_{L_j} - \boldsymbol{\xi}_0) \cdot \frac{\boldsymbol{\xi}_{1_j} - \boldsymbol{\xi}_0}{\|\boldsymbol{\xi}_{1_j} - \boldsymbol{\xi}_0\|}, \tag{3.60}$$
$$\alpha_x = \alpha_\xi. \tag{3.61}$$

Remark 3.6.12. Note that $\alpha_\xi = \frac{1}{2}$ for triangular and quadrilateral cells. The boundary conditions are non-orthogonal, in contrast to Equation (3.8). This maintains the linearity of operators G_ξ and G_η.

Substitution in Equation (3.40) yields for each boundary edge j

$$G_\xi \Phi|_j = \frac{-(\eta_{R^*_j} - \xi_{L_j})(\Phi_{1_j} - \Phi_0) + (\eta_{1_j} - \eta_0)(\Phi_{R^*_j} - \sum_{i \in \mathcal{N}} A^i_{L_j} \Phi_i)}{\|(\boldsymbol{\xi}_{1_j} - \boldsymbol{\xi}_0) \times (\boldsymbol{\xi}_{R^*_j} - \boldsymbol{\xi}_{L_j})\|}, \quad j \in \mathcal{J}_{bnd}, \tag{3.62}$$

and

$$G_\eta \Phi|_j = \frac{(\xi_{R^*_j} - \xi_{L_j})(\Phi_{1_j} - \Phi_0) - (\xi_{1_j} - \xi_0)(\Phi_{R^*_j} - \sum_{i \in \mathcal{N}} A^i_{L_j} \Phi_i)}{\|(\boldsymbol{\xi}_{1_j} - \boldsymbol{\xi}_0) \times (\boldsymbol{\xi}_{R^*_j} - \boldsymbol{\xi}_{L_j})\|}, \quad j \in \mathcal{J}_{bnd}, \tag{3.63}$$

3.6.2.2 Edge-to-node operator

For the *edge-to-node* gradient we take the control volume as indicated in Figure 3.13

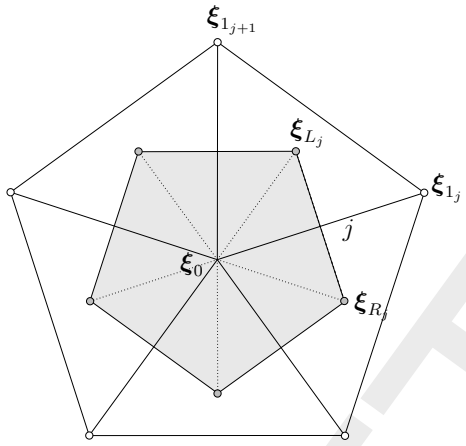


Figure 3.13: *Control volume for computing the* edge-to-node *gradient at the central node for the discrete operators D_ξ and D_η, where $\boldsymbol{\xi} = \boldsymbol{\xi}_0 = \mathbf{0}$*

and obtain

$$(D_\xi, D_\eta)^{\mathrm{T}} = \frac{1}{V}\boldsymbol{d}_j, \tag{3.64}$$

where

$$\boldsymbol{d}_j = \begin{cases} (\boldsymbol{\xi}_{R_j} - \boldsymbol{\xi}_{L_j})^{\perp}, & j \in \mathcal{J}_{int}, \\ (\boldsymbol{\xi}_{bc_j} - \boldsymbol{\xi}_{L_j})^{\perp} - (\boldsymbol{\xi}_{bc_j} - \boldsymbol{\xi}_0)^{\perp}, & j \in \mathcal{J}_{bnd}, \end{cases} \tag{3.65}$$

and with $\boldsymbol{\xi} \in I\!R^2$

$$V = \int_\Omega d\Omega = \frac{1}{2}\int_\Omega \nabla \cdot \boldsymbol{\xi}\, d\Omega = \frac{1}{2}\oint_{\partial\Omega} \boldsymbol{\xi} \cdot \boldsymbol{n}\, d\Gamma \Rightarrow \tag{3.66}$$

$$V = \frac{1}{2}\sum_{j\in\mathcal{J}_{int}} \frac{\boldsymbol{\xi}_{L_j} + \boldsymbol{\xi}_{R_j}}{2} \cdot \boldsymbol{d}_j + \frac{1}{2}\sum_{j\in\mathcal{J}_{bnd}} \frac{\boldsymbol{\xi}_{L_j} + \boldsymbol{\xi}_{R_j^*}}{2} \cdot \boldsymbol{d}_j. \tag{3.67}$$

3.6.2.3 Node-to-node operator

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

Definition 3.6.13. *J_ξ and J_η are the* node-to-node *approximations of the ξ and η derivatives respectively.*

They can be constructed as

$$J_\xi|_i = \sum_{j\in\mathcal{J}_{int}} D_\xi \left(\frac{1}{2}\sum_{i\in\mathcal{I}_{int}} (A^i_{L_j} J_i + A^i_{L_{j-1}} J_i)\right)_j + \sum_{j\in\mathcal{J}_{bnd}} D_\xi \left(\frac{1}{2}(J_{0_j} + J_{1_j})\right)_j, \tag{3.68}$$

and

$$J_\eta|_i = \sum_{j\in\mathcal{J}_{int}} D_\eta \left(\frac{1}{2} \sum_{i\in\mathcal{I}_{int}} (A^i_{L_j} J_i + A^i_{L_{j-1}} J_i) \right)_j + \sum_{j\in\mathcal{J}_{bnd}} D_\eta \left(\frac{1}{2}(J_{0_j} + J_{1_j}) \right)_j . \tag{3.69}$$

3.6.3 Computing the mesh monitor matrix

In the discretization of Equation (3.14), we approximate the contravariant base vectors by firstly computing the Jacobian by applying Equation (3.68) and Equation (3.69), and using $\boldsymbol{a}^1 = \nabla\xi$ and $\boldsymbol{a}^2 = \nabla\eta$:

$$\boldsymbol{a}^1 = (\quad J_{22}, -J_{12})^{\mathrm{T}} / \det J, \tag{3.70}$$

$$\boldsymbol{a}^2 = (-J_{21}, \quad J_{11})^{\mathrm{T}} / \det J. \tag{3.71}$$

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

$$\boldsymbol{v} = \nabla u, \tag{3.72}$$

which is approximated by firstly smoothing u, and computing

$$\boldsymbol{v} = \sum_{i\in\mathcal{N}} \boldsymbol{a}^1 J_\xi u_i + \boldsymbol{a}^2 J_\eta u_i. \tag{3.73}$$

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 (3.14) is a straightforward task. We obtain

$$\sum_{i\in\mathcal{N}} w_i \boldsymbol{x}_i = \boldsymbol{0}, \tag{3.74}$$

where

$$\begin{aligned} w_i = &\left\langle \boldsymbol{a}^1, \frac{\partial\,(G^{-1})}{\partial\xi}\boldsymbol{a}^1 \right\rangle J_\xi + \left\langle \boldsymbol{a}^1, \frac{\partial\,(G^{-1})}{\partial\xi}\boldsymbol{a}^2 \right\rangle J_\eta + \\ &\left\langle \boldsymbol{a}^2, \frac{\partial\,(G^{-1})}{\partial\eta}\boldsymbol{a}^1 \right\rangle J_\xi + \left\langle \boldsymbol{a}^2, \frac{\partial\,(G^{-1})}{\partial\eta}\boldsymbol{a}^2 \right\rangle J_\eta \\ &- \Bigg(\left\langle \boldsymbol{a}^1, G^{-1}\boldsymbol{a}^1 \right\rangle \sum_{j\in\mathcal{J}} D_\xi\, G_\xi|_j + \left\langle \boldsymbol{a}^1, G^{-1}\boldsymbol{a}^2 \right\rangle \sum_{j\in\mathcal{J}} D_\xi\, G_\eta|_j + \\ &\quad \left\langle \boldsymbol{a}^2, G^{-1}\boldsymbol{a}^1 \right\rangle \sum_{j\in\mathcal{J}} D_\eta\, G_\xi|_j + \left\langle \boldsymbol{a}^2, G^{-1}\boldsymbol{a}^2 \right\rangle \sum_{j\in\mathcal{J}} D_\eta\, G_\eta|_j \Bigg) = 0, \end{aligned} \tag{3.75}$$

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)

4.2.2 Solving the matrix

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

[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.

`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

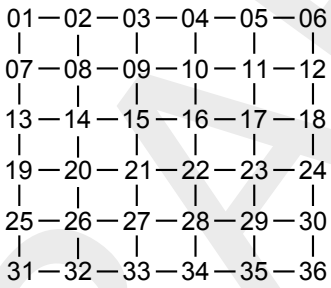
The subroutine does the following steps:

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 Reynolds stresses in the horizontal momentum equation 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 effect is not simulated as the vertical profile of the horizontal velocity is not resolved. This dispersive effect may be modelled as the product of a viscosity coefficient and a velocity gradient. The dispersive viscosity coefficient 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 instead of 2D modelling with Elder approximation.

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 viscosity ν_V. D-Flow FM simulates the larger scale horizontal turbulent motions through a

sub-grid scale method (SGS), eg. Elder. The user may add a background horizontal viscosity, ν_H^{back}, as a constant or spatially dependent. Consequently, in D-Flow FM 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 D-Flow FM a constant (space and time) background mixing coefficient may be specified by the user, which is a background value for the vertical eddy viscosity in the momentum equations. Consequently, the vertical eddy viscosity coefficient is defined by:

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

with ν_{mol} the kinematic viscosity of water. The 3D part ν_{3D} is computed by a 3D-turbulence closure model, see section 7.7.

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 clockwise, then $1/R > 0$ and if it rotates counterclockwise, then $1/R < 0$. Following this convention, the spiral flow intensity will be negative for bends with flows from left to right, and positive for bends with flows from right to the left.

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

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

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

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

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

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

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

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

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

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

Here g is the gravity, κ is the von Kármán constant and C is the Chézy 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 (without the Coriolis force)

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

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

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

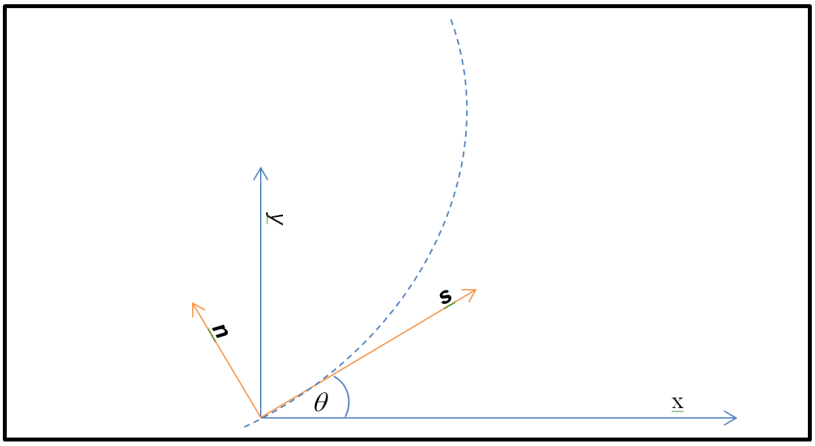


Figure 5.1: *The flow streamline path and the direction of dispersion stresses.*

averaged velocity can be written as:

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

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

The depth varying component can subsequently be written as:

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

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

Which can subsequently be rewritten as:

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

$$v^* = \frac{u}{|\boldsymbol{u}|} I f_n + v f_s \tag{5.26}$$

The dispersion terms can be evaluated as:

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

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

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

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, p. 9)
- $I^2 \langle f_n^2 \rangle$ is small for mildly curving, shallow water flow
- $\langle f_s f_n \rangle = 5\alpha - 15.6\alpha^2 + 37.5\alpha^3$ (cf. Delft3D-FLOW UM (2013, eq. 9.155))

Under these assumptions the dispersion stresses can be simplified to:

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

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

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

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 previous 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 $\Delta x_{ik} = x_k - x_i$, $\Delta y_{ik} = y_k - y_i$ and $\Delta u_{ik} = u_k - u_i$, it yields

$$E_{ik}^2 = \left(-\Delta u_{ik} + \left.\frac{\partial u}{\partial x}\right|_i \Delta x_{ik} + \left.\frac{\partial u}{\partial y}\right|_i \Delta y_{ik}\right)^2 \tag{5.35}$$

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

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

$$b_i = \sum_{k=1}^{N} w_{ik}^2 \Delta x_{ik} \Delta y_{ik} \tag{5.41}$$

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

$$d_i = \sum_{k=1}^{N} w_{ik}^2 \Delta u_{ik} \Delta x_{ik} \tag{5.43}$$

$$e_i = \sum_{k=1}^{N} w_{ik}^2 \Delta u_{ik} \Delta y_{ik} \tag{5.44}$$

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 $\mathcal{O}\left(\Delta x^4\right)$, 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 Haselbacher and Blazek (1999) and Anderson and Bonhaus (1994). 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}, $T_{xy}(= T_{yx})$ and T_{yy} are calculated parametrically by Equation (5.27) to Equation (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]

5.12 Overview of research keywords

In Table A.1 in Appendix A of the D-Flow Flexible Mesh User Manual an overview of keywords in the master definition file is given that can be specified by the user. These keywords can be changed in the DeltaShell Graphical User Interface as well. Next, there are keywords that cannot be specified yet by the Graphical User Interface of D-Flow FM. This mainly involves keywords for 3D modelling, because the GUI isn't ready yet for 3D modelling. These keywords are listed in Table A.2 in Appendix A of the D-Flow Flexible Mesh User Manual.

In addition there are several research keywords in the computational kernel of D-Flow Flexible Mesh that should, in principle, not be changed by the user. These keywords haven't been documented yet, aren't tested in the D-Flow FM testbenches and should be seen as tests. Therefore, the use of default setting of these research keywords is strongly recommended. Because these keywords are listed in the diagnostic file, they are listed in the table below.

Table 5.1: *Overview of numerical model parameters that should not be changed.*

Keyword	Default setting	Description
`[numerics]`		
`TransportMethod`	`1`	Time integation method in transport module, 1 = standard (default) , 2 = alternative
`TransportTimestepping`	`1`	Timestepping method in Transport module, 0 = global (default) , 1 = local
`Vertadvtypmom`	`6`	Vertical advection type in momentum equation; 6=default
`Vertadvtypsal`	`6`	Vertical advection type for salinity (0: none, 1: upwind explicit, 2: central explicit, 3: upwind implicit, 4: central implicit, 5: central implicit but upwind for neg. stratif., 6: higher order explicit, no Forester)
`Vertadvtyptem`	`6`	See description of keyword Vertadvtypsal
`Filterorder`	`2`	First-order or second order filter to suppress checkerboarding
`Icoriolistype`	`5`	0=No, 1=yes, if jsferic then spatially varying, 2-5: ..., if icoriolistype==6 then constant (anglat)
`Newcorio`	`0`	Temporary keyword for Coriolis improvement on open boundary; 0=No, 1=yes
`Corioadamsbashfordfac`	`0.`	Adams Bashford factor for Coriolis term
`Cffacver`	`0.`	Factor for including (1-CFL) in HO term vertical (0d0: no, 1d0: yes)

(continued on next page)

Keyword	Preferred setting	Description
		(continued from previous page)
Jarhoxu	0	Include density gradient in momentum equation (0: no, 1: yes)
Barocterm	2	1 or 2 for original or revised term (only documented)
BaroctimInt	1	Time integration baroclinic pressure, 1 = explicit, abs() = 2; adams bashford , 3 = ab3, 5 = adv rho
Horadvtypzlayer	0	Horizontal advection treatment of z-layers (0: default, 2: sigma-like)
Logprofatubndin	0	ubnds inflow: 0=uniform U1, 1 = log U1, 2 = user3D
Logprofkepsbndin	0	inflow: 0=0 keps, 1 = log keps, 2 = user3D
Turbulenceadvection	3	Turbulence advection (0: none, 1=UpwexpL, 2=Central explicit, 3: horizontally explicit and vertically implicit, vert. adv. keps
Tspinupturblogprof	0	0=original hu on qbnd, 1=downwind hs on qbnd
Keepstbndonoutflow	1	Keep salinity and temperature signals on boundary also at outflow, 1=yes, 0=no=default=copy inside value on outflow
Windhuorzwsbased	0	Switch for computation of wind stress at flow link: 0=hu based, 1=finite volume based

Keywords like Newcorio and Corioadamsbashfordfac are recent improvements. Because testing and documentation is lacking these keywords are in the table shown above. It is likely that in future these keywords will become operational.

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

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

$$\frac{\partial h\boldsymbol{u}}{\partial t} + \nabla \bullet (h\boldsymbol{u}\boldsymbol{u}) = -gh\nabla\zeta + \nabla \bullet (\nu h(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^{\mathrm{T}})) + \frac{\boldsymbol{\tau_b} + \boldsymbol{\tau_w}}{\rho}. \tag{6.2}$$

where $\nabla = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right)^{\mathrm{T}}$ (i.e. two dimensional), ζ is the water level, h the water depth, $\boldsymbol{u}$ the velocity vector, g the gravitational acceleration, ν the viscosity and ρ the water mass density.

$\boldsymbol{\tau_b}$ is the bottom friction:

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

with C being the Chézy coefficient.

Similarly, $\boldsymbol{\tau_w}$ is the wind friction acting at the free surface:

$$\boldsymbol{\tau_w} = C_d \rho_a \|\tilde{\boldsymbol{u}}_{\mathbf{10}}\|\tilde{\boldsymbol{u}}_{\mathbf{10}}, \tag{6.4}$$

ρ_a is the air density and C_d is air-water (or wind) friction coefficient. The wind velocity vector at 10 m above the free surface $\tilde{\boldsymbol{u}}_{\mathbf{10}}$ can either be the absolute wind velocity $\tilde{\boldsymbol{u}}_{\mathbf{10}} = \boldsymbol{u}_{\mathbf{10}}$ (which is the default option), or it can be the wind velocity relative to the flow velocity $\tilde{\boldsymbol{u}}_{\mathbf{10}} = \boldsymbol{u}_{\mathbf{10}} - \boldsymbol{u}$. This second option can be chosen by the user using the keyword `Relativewind`. The wind friction coefficient C_d is either prescribed as a constant or computed based on a relation depending on the actual wind velocity. Several such formulations are available. These are described in Section section 6.2.2.

Note that $\boldsymbol{\tau_w}$ and $\boldsymbol{\tau_b}$ have different signs.

Now we rewrite the time derivative of the momentum equation (Equation (6.2)) as:

$$\frac{\partial h\boldsymbol{u}}{\partial t} = h\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u}\frac{\partial h}{\partial t} \tag{6.5}$$

The shallow water equations can then be written as:

$$\frac{\partial h}{\partial t} + \nabla \bullet (h\boldsymbol{u}) = 0, \tag{6.6}$$

$$\frac{\partial \boldsymbol{u}}{\partial t} + \frac{1}{h}(\nabla \bullet (h\boldsymbol{u}\boldsymbol{u}) - \boldsymbol{u}\nabla \bullet (h\boldsymbol{u})) = -g\nabla\zeta + \frac{1}{h}\nabla \bullet (\nu h(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^{\mathrm{T}})) + \frac{1}{h}\frac{\boldsymbol{\tau_b} + \boldsymbol{\tau_w}}{\rho}, \tag{6.7}$$

The equations are complemented with appropriate initial conditions and water level and/or velocity boundary conditions. The boundary conditions are explained in section 6.4. The initial conditions will not be discussed further.

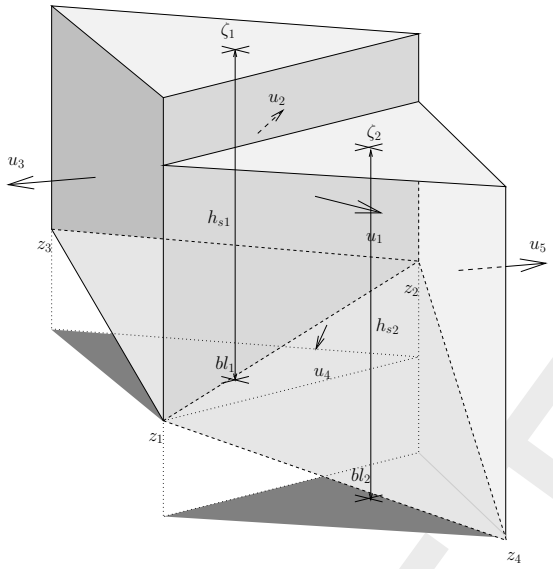
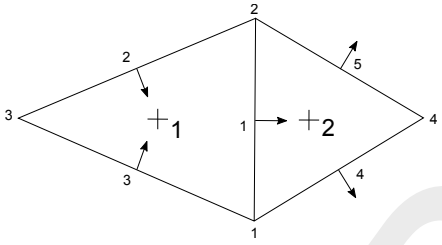
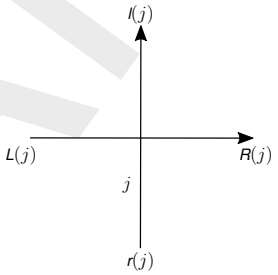


Figure 6.1: *Discretization of the water level ζ_k (at cell circumcenter), bed-levels z_i (at nodes) and bl_k (at cell circumcenter), water depth h_k ($= \zeta_k - bl_k$; at cell circumcenter) and face-normal velocities u_j (at faces).*



(a) *Top-view on Figure 6.1. Numbering of cells, faces and nodes. The flow direction through the face is positive from the left to the right cell as defined by $\boldsymbol{n}$.*



(b) *Orientation of face j to the neighboring cells and nodes.*

Figure 6.2: *Numbering of cells, faces and nodes, with their orientation to each other.*

6.1 Topology of the mesh

In this section the connectivity between cells, faces and nodes is defined (topology) and how the bed level is interpreted.

6.1.1 Connectivity

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

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 $l(j)$ and $r(j)$, given some orientation of face j,
- ◇ face j contains neighbors cells $L(j)$ and $R(j)$, given some orientation of face j.

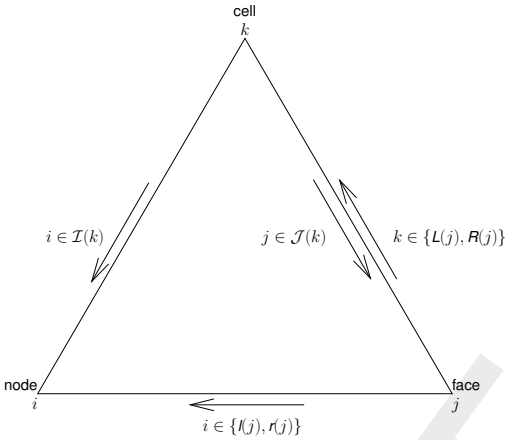


Figure 6.3: *Connectivity of cells, faces and nodes*

Thus, in the example of Figure 6.2:

$$
\begin{aligned}
&\mathcal{J}(1) = \{1, 2, 3\},\\
&\mathcal{J}(2) = \{4, 1, 5\},\\
&\mathcal{I}(1) = \{1, 2, 3\},\\
&\mathcal{I}(2) = \{2, 4, 1\},\\
&l(1) = 2 \text{ and } r(1) = 1,\\
&l(2) = 2 \text{ and } r(2) = 3,\\
&\ldots,\\
&l(5) = 2 \text{ and } r(5) = 4,\\
&L(1) = 1 \text{ and } R(1) = 2,\\
&L(2) = * \text{ and } R(2) = 1,\\
&\ldots,\\
&L(5) = 2 \text{ and } R(5) = *
\end{aligned}
$$

The orientation of face j with respect to cell $k \in \{L(j), R(j)\}$ is accounted for by $s_{j,k}$ in the following manner:

$$
s_{j,k} = \begin{cases} 1, & L(j) = k \quad (\boldsymbol{n}_j \text{ is outward normal of cell } k),\\ -1, & R(j) = k \quad (\boldsymbol{n}_j \text{ is inward normal of cell } k), \end{cases} \tag{6.8}
$$

where $\boldsymbol{n}_j$ is the normal vector of face j, defined positive in the direction from cell $L(j)$ to $R(j)$. In the example of Figure 6.2a $s_{1,1} = 1$ and $s_{1,2} = -1$.

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`,
$l(j)$: `lncn(2,j)`, $r(j)$: `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_k and its face-based values bl_{1_j} and bl_{2_j}. Given some orientation, bl_{1_j} represents the left-hand side bed level at face j and bl_{2_j} the right-hand side bed level, respectively. In such a manner a linear representation of the bed from bl_{1_j} to bl_{2_j} is obtained at face j. It is used, for example, in the computation of the flow area A_{u_j} as we will see in Section Face-based water depth h_{u_j}.

Note: that for the sake of clarity we will not discuss one-dimensional modelling 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 bl_{u_j}. 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 of nodes $\mathcal{I}(k)$ and faces $\mathcal{J}(k)$, respectively.

How the face-based bed levels bl_{1_j} and bl_{2_j} are determined is shown in Algorithm (1). Refer to section 6.1.1 for the definitions of $L(j)$, $R(j)$, $r(j)$ and $l(j)$ respectively.

The notation translates directly to administration in the D-Flow FM code as follows:

bl_{1_j}: bob(1,j),

bl_{2_j}: bob(2,j).

6.2 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 Figure 6.2 for an example. Note that in this example $k \in \{1, 2\}$ and $j \in \{1, 2, \ldots, 5\}$.

6.2.1 Continuity equation

The continuity equation reads:

$$\frac{\partial h}{\partial t} + \nabla \bullet (h\boldsymbol{u}) = 0, \tag{6.12}$$

and is spatially discretized as, see (Equation (6.6)):

$$\frac{\mathrm{d}V_k}{\mathrm{d}t} = -\sum_{j \in \mathcal{J}(k)} A_{u_j} u_j s_{j,k}, \tag{6.13}$$

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

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

$$bl_k = \begin{cases} \text{user specified}, & \text{bed level type} = 1, \\ \min\limits_{j \in \mathcal{J}(k)} (bl_{u_j}), & \text{bed level type} = 2, \\ \min\limits_{j \in \mathcal{J}(k)} \left[\min(bl_{1j}, bl_{2j})\right], & \text{bed level type} \in \{3,4,5\}, \\ \sum\limits_{i \in \mathcal{I}(k)} z_i / \sum\limits_{i \in \mathcal{I}(k)} 1, & \text{bed level type} = 6 \\ z_{k\text{uni}}, & \text{otherwise (this value is supplied to cells without specified bed level).} \end{cases} \tag{6.9}$$

$$bl_{1j} = \begin{cases} \max(bl_{L(j)}, bl_{R(j)}), & \text{bed level type} = 1, \\ bl_{u_j}, & \text{bed level type} = 2, \\ z_{l(j)}, & \text{bed level type} \in \{3,4,5\} \quad \wedge \quad \text{conveyance type} \geq 1, \\ \frac{1}{2}(z_{l(j)} + z_{r(j)}), & \text{bed level type} = 3 \quad \wedge \quad \text{conveyance type} < 1, \\ \min(z_{l(j)}, z_{r(j)}), & \text{bed level type} = 4 \quad \wedge \quad \text{conveyance type} < 1, \\ \max(z_{l(j)}, z_{r(j)}), & \text{bed level type} = 5 \quad \wedge \quad \text{conveyance type} < 1, \\ \max(bl_{L(j)}, bl_{R(j)}), & \text{bed level type} = 6. \end{cases} \tag{6.10}$$

$$bl_{2j} = \begin{cases} z_{r(j)}, & \text{bed level type} \in \{3,4,5\} \quad \wedge \quad \text{conveyance type} \geq 1, \\ bl_{1j}, & \text{otherwise.} \end{cases} \tag{6.11}$$

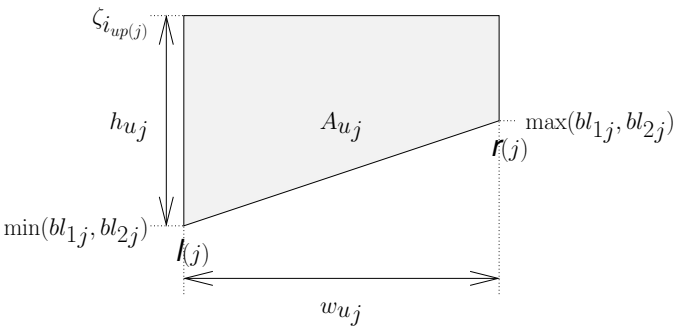


Figure 6.4: *Flow area A_{u_j} and face-based water depth h_{u_j}*

Furthermore, V_k is the volume of the water column at cell k computed with Algorithm (20), not discussed here, A_{u_j} approximates the flow area of face j, computed with Algorithm (3), and h_{u_j} is the water depth at face j, computed with Algorithm (2). Algorithms (3) and (2) will be discussed momentarily.

Face-based water depth h_{u_j}

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_{u_j} is then defined as the maximum water depth in face j, see Figure 6.4. It is computed with Algorithm (2).

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

$$h_{u_j} = \begin{cases} \zeta_{L(j)} - \min(bl_{1j}, bl_{2j}), & u_j > 0 \quad \vee \quad u_j = 0 \quad \wedge \quad \zeta_{L(j)} > \zeta_{R(j)}, \\ \zeta_{R(j)} - \min(bl_{1j}, bl_{2j}), & u_j < 0 \quad \vee \quad u_j = 0 \quad \wedge \quad \zeta_{L(j)} \leq \zeta_{R(j)}. \end{cases} \tag{6.14}$$

Example

In the example of Figure 6.1, the water level at face j, assumed constant in the face as indicated in the figure, is approximated by:

$$\zeta_{u_j} = \begin{cases} \zeta_2 & \text{if } u_1 < 0 \\ \max(\zeta_1, \zeta_2) & \text{if } u_1 = 0 \\ \zeta_1 & \text{if } u_1 > 0 \end{cases} \tag{6.15}$$

Remark 6.2.1. We will see later in Equation (6.112) that the time-integration of the continuity equation is implicit/explicit. Nonetheless, the upwind direction of h_{u_j} is based on the velocity at the beginning of the time-step only.

Remark 6.2.2. The *upwind* reconstruction of h_{u_j} 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_{u_j}

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

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

$$\Delta bl_j = \max(bl_{1j}, bl_{2j}) - \min(bl_{1j}, bl_{2j}), \tag{6.16}$$

$$A_{u_j} = \begin{cases} w_{uj} h_{u_j}, & \text{if } \Delta bl_j < \delta w_{uj}, \\ w_{uj} h_{u_j} \min(\frac{h_{u_j}}{\Delta bl_j}, 1) \left(1 - \frac{1}{2}\min(\frac{\Delta bl_j}{h_{u_j}}, 1)\right), & \text{otherwise.} \end{cases} \tag{6.17}$$

is the width of face j, see Figure 6.4, and Δbl_j is the cross-sectional bed variation.

Remark 6.2.3. The exception for the case $\Delta bl_j < \delta w_{uj}$ with $\delta = 10^{-3}$ in Equation (6.17) should maybe be reconsidered.

Remark 6.2.4. 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 linearly varying bed, on the other hand, is *not* accounted for in the computation of the water column volumes V_k in Algorithm (20), without non-linear iterations (explained later). This seems *inconsistent* when we are employing a finite volume approximation as in e.g. Equation (6.13).

Total area of a cell

The definition of variables to determined the total area of a computational cell are depicted in Figure 6.5.

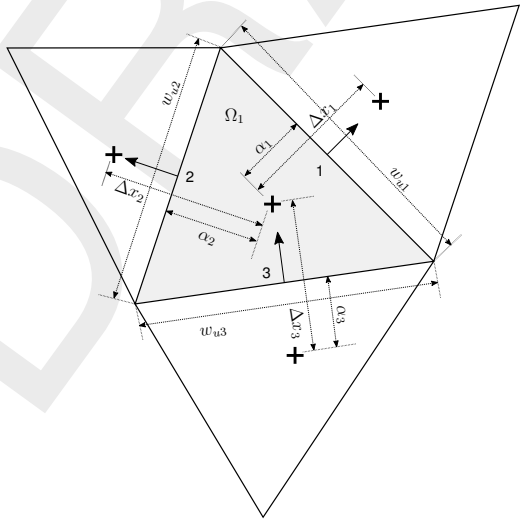


Figure 6.5: *Area computation for cell* Ω_1

The total area A of cell Ω_k is determined as follows:

$$A(\Omega_k) = \sum_{j \in \mathcal{J}(\Omega_k) | L(j) = k} \alpha_j \Delta x_j w_{uj} + \sum_{j \in \mathcal{J}(\Omega_k) | L(j) \neq k} (1 - \alpha_j) \Delta x_j w_{uj} \tag{6.18}$$

An example for area Ω_1 is:

$$A(\Omega_1) = \frac{1}{2}\alpha_1 \Delta x_1 w_{u1} + \frac{1}{2}\alpha_2 \Delta x_2 w_{u2} + \frac{1}{2}(1-\alpha_3)\Delta x_3 w_{u3} \tag{6.19}$$

6.2.2 Momentum equation

The momentum equation reads

$$\begin{aligned}\frac{\partial \boldsymbol{u}}{\partial t} + \frac{1}{h}(\nabla \bullet (h\boldsymbol{u}\boldsymbol{u}) - \boldsymbol{u}\nabla \bullet (h\boldsymbol{u})) = &- g\nabla\zeta + \frac{1}{h}\nabla \bullet (\nu h(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^{\mathrm{T}})) \\ &+ \frac{1}{h}\frac{\boldsymbol{\tau_b} + \boldsymbol{\tau_w}}{\rho},\end{aligned} \tag{6.20}$$

and is discretized at the *faces* and in face-normal direction, see Figure 6.6.

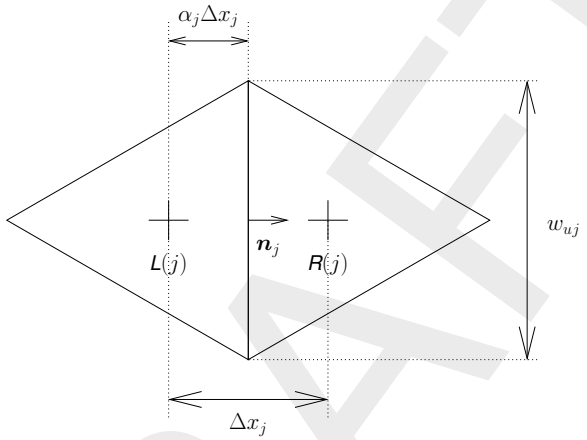


Figure 6.6: *Computational cells $L(j)$ and $R(j)$ neighboring face j; water levels are stored at the cell circumcenters, indicated with the $+$-sign*

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_{u_j} 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

$$g\nabla\zeta|_j \bullet \boldsymbol{n}_j \approx \frac{g}{\Delta x_j}(\zeta_{R(j)} - \zeta_{L(j)}). \tag{6.21}$$

The bed friction term is discretized as

$$\left.\frac{1}{h}\frac{\boldsymbol{\tau_b}}{\rho}\right|_j \bullet \boldsymbol{n}_j \approx -\frac{g\|\boldsymbol{u}_j\|}{C^2 \hat{h}_j} u_j, \tag{6.22}$$

where $\hat{h}_j$ acts as a hydraulic radius for which the computation depends on the "conveyance type". Its precise discretization is discussed later (see Algorithm (12) and Algorithm (13)). The magnitude of the velocity is computed by: $\|\boldsymbol{u}_j\| = \sqrt{u_j^2 + v_j^2}$.

The wind friction is computed as:

$$\left.\frac{1}{h}\frac{\boldsymbol{\tau_w}}{\rho}\right|_j \bullet \boldsymbol{n}_j \approx C_d \frac{\rho_a}{\rho}\frac{\|\tilde{\boldsymbol{u}}_{\mathbf{10}}\|\tilde{\boldsymbol{u}}_{\mathbf{10}} \bullet \boldsymbol{n}_j}{h_{uvj}}. \tag{6.23}$$

where h_{uvj} is the distance-weighted water depth at the face, determined using the depths at the adjacent cell centres $h_{\zeta_{L(j)}}$ and $h_{\zeta_{R(j)}}$, and the non-dimensional distance α_j of the cell centres to the face, see Figure 6.6:

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

where ε_{h_ζ} is a threshold.

The wind velocity vector at 10 m above the free surface $\tilde{\boldsymbol{u}}_{\mathbf{10}}$ can either be the absolute wind velocity $\tilde{\boldsymbol{u}}_{\mathbf{10}} = \boldsymbol{u}_{\mathbf{10}}$ (which is the default option), or it can be the wind velocity relative to the flow velocity $\tilde{\boldsymbol{u}}_{\mathbf{10}} = \boldsymbol{u}_{\mathbf{10}} - \boldsymbol{u}_j$, where $\boldsymbol{u}_j$ is the full velocity vector at the face, which depends on u_j itself and the tangential velocity v_j, determined using (6.50).

The definition of the wind friction coefficient C_d is described further below in the section on *Wind friction*.

Furthermore, advection and diffusion are discretized as

$$\left[\frac{1}{h}(\nabla\bullet(h\boldsymbol{u}\boldsymbol{u}) - \boldsymbol{u}\nabla\bullet(h\boldsymbol{u})) - \frac{1}{h}\nabla\bullet(\nu h(\nabla\boldsymbol{u} + \nabla\boldsymbol{u}^{\mathrm{T}}))\right]_j \bullet \boldsymbol{n}_j \approx \mathcal{A}_{\mathrm{i}j} u_j + \mathcal{A}_{\mathrm{e}j}. \tag{6.25}$$

The terms $\mathcal{A}_{\mathrm{i}j}$ (implicit part) and $\mathcal{A}_{\mathrm{e}j}$ (explicit part) 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 Equation (6.7) reads

$$\frac{\mathrm{d}u_j}{\mathrm{d}t} = -\frac{g}{\Delta x_j}\left(\zeta_{R(j)} - \zeta_{L(j)}\right) - \mathcal{A}_{\mathrm{i}j} u_j - \mathcal{A}_{\mathrm{e}j} - \frac{g\|\boldsymbol{u}_j\|}{C^2\hat{h}_j} u_j + C_d \frac{\rho_a}{\rho}\frac{\|\tilde{\boldsymbol{u}}_{\mathbf{10}}\|\tilde{\boldsymbol{u}}_{\mathbf{10}}\bullet\boldsymbol{n}_j}{\hat{h}_j}. \tag{6.26}$$

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 6.2.5. 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 6.2.6. 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

$$\nabla \cdot (h\boldsymbol{u}\boldsymbol{u})|_{\Omega_k} \approx \boldsymbol{a}_k. \tag{6.27}$$

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

$$\nabla \cdot (h\boldsymbol{u}\boldsymbol{u})|_{\Gamma_j} \cdot \boldsymbol{n}_j \approx \left(\alpha_j \boldsymbol{a}_{L(j)} + (1-\alpha_j)\boldsymbol{a}_{R(j)}\right) \cdot \boldsymbol{n}_j, \tag{6.28}$$

where α_j is the non-dimensional distance from the left cell center to the face, see Figure 6.6. Note that the terms $-\boldsymbol{u}\nabla \cdot (h\boldsymbol{u})$ and $\frac{1}{h}$ in Equation (6.25) are due to our non-conservative formulation and do not appear in Equation (6.28). In the non-conservative formulation of Equation (6.25), 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

$$\boldsymbol{a}_k = \frac{1}{A(\Omega_k)} \sum_{j \in \mathcal{J}(k)} \boldsymbol{u}_{u_j} q_j s_{j,k}, \tag{6.29}$$

where $\boldsymbol{u}_{u_j}$ 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.

$$\boldsymbol{u}_{u_j} = \begin{cases} \boldsymbol{u}_{L(j)}, & u_j \geq 0, \\ \boldsymbol{u}_{R(j)}, & u_j < 0. \end{cases}, \tag{6.30}$$

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

$$q_j = A_{u_j} u_j, \tag{6.31}$$

see also Algorithm (21), 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 Equation (6.123) (explained later). This means that we require the fluxes in the storage term to be identical to the fluxes in the discrete continuity equation, Equation (6.112), i.e. q_j^{n+1}, where n denotes the time level:

$$-\boldsymbol{u}\nabla \cdot (h\boldsymbol{u})|_k^n \approx -\frac{\boldsymbol{u}}{A(\Omega_k)} \sum_{j \in \mathcal{J}(k)} q_j^{n+1} s_{j,k}, \tag{6.32}$$

where we do not mention at which time level term $\frac{\boldsymbol{u}}{A(\Omega_k)}$ is to be evaluated. Equation (6.32) 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 6.2.7. 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. ((6.28)) and ((6.29)) and the storage term of Equation (6.32), the advection can now be composed in general form as

$$\begin{aligned}\mathcal{A}_{\mathrm{e}j} \;=\; & A_{Lj} \sum_{l\in\mathcal{J}^*(L(j))} q_l^* s_{l,L(j)} \boldsymbol{u}_{ul}\cdot\boldsymbol{n}_j - q_l^{**} s_{l,L(j)}(1-\theta_{l,L(j)})u^*_{Lj} \;+ \\ & A_{Rj} \sum_{l\in\mathcal{J}^*(R(j))} q_l^* s_{l,R(j)} \boldsymbol{u}_{ul}\cdot\boldsymbol{n}_j - q_l^{**} s_{l,R(j)}(1-\theta_{l,R(j)})u^*_{Rj},\end{aligned} \tag{6.33}$$

and

$$\mathcal{A}_{\mathrm{i}j} = -A_{Lj} \sum_{l\in\mathcal{J}^*(L(j))} q_l^{**} s_{l,L(j)}\theta_{l,L(j)} \;-\; A_{Rj} \sum_{l\in\mathcal{J}^*(R(j))} q_l^{**} s_{l,R(j)}\theta_{l,R(j)}, \tag{6.34}$$

where $\mathcal{J}^*$, q_l^*, q_l^{**}, $\theta_{l,\{L,R\}(j)}$, $u^*_{\{L,R\}_j}$, A_{L_j}, A_{R_j} 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.

$$\mathcal{J}^i(k) = \{j \in \mathcal{J}(k) | u_j s_{j,k} < 0\} \tag{6.35}$$

and

$$h_{uvj} = \max(\alpha_j h_{\zeta_{L(j)}} + (1-\alpha_j) h_{\zeta_{R(j)}}, \varepsilon_{h_\zeta}), \tag{6.36}$$

where ε_{h_ζ} is a threshold. $\theta_{l,L(j)}$ and $\theta_{l,R(j)}$ are determined by their face-based Courant numbers $\sigma_{j,L(j)}$ and $\sigma_{j,R(j)}$ as follows

$$\theta_{l,\{L,R\}(j)} = \frac{1}{1-\sigma_{j,\{L,R\}(j)}} \tag{6.37}$$

where $\sigma_{j,L(j)}$ and $\sigma_{j,R(j)}$ are computed as:

$$\sigma_{j,L(j)} = \begin{cases} \frac{1.4\Delta t|q_{a_j}|}{\alpha_j V_{L(j)} + (1-\alpha_j)V_{R(j)}}, & \sum\limits_{j\in\mathcal{J}(L(j))} 1 = 3, \\ \frac{\Delta t|q_{a_j}|}{\alpha_j V_{L(j)} + (1-\alpha_j)V_{R(j)}}, & \text{other}, \end{cases} \tag{6.38}$$

and

$$\sigma_{j,R(j)} = \begin{cases} \frac{1.4\Delta t|q_{a_j}|}{\alpha_j V_{L(j)} + (1-\alpha_j)V_{R(j)}}, & \sum\limits_{j\in\mathcal{J}(R(j))} 1 = 3, \\ \frac{\Delta t|q_{a_j}|}{\alpha_j V_{L(j)} + (1-\alpha_j)V_{R(j)}}, & \text{other}, \end{cases} \tag{6.39}$$

respectively.

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

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

$$\begin{aligned}\mathcal{A}_{\mathrm{e}j} = A_{Lj} \sum_{l\in\mathcal{J}^*(L(j))} q_l^* s_{l,L(j)} \boldsymbol{u}_{ul} \cdot \boldsymbol{n}_j - q_l^{**} s_{l,L(j)}(1-\theta_{l,L(j)}) u^*_{Lj} + \\ A_{Rj} \sum_{l\in\mathcal{J}^*(R(j))} q_l^* s_{l,R(j)} \boldsymbol{u}_{ul} \cdot \boldsymbol{n}_j - q_l^{**} s_{l,R(j)}(1-\theta_{l,R(j)}) u^*_{Rj}\end{aligned} \tag{6.40}$$

$$\mathcal{A}_{\mathrm{i}j} = -A_{Lj} \sum_{l\in\mathcal{J}^*(L(j))} q_l^{**} s_{l,L(j)} \theta_{l,L(j)} \quad - \quad A_{Rj} \sum_{l\in\mathcal{J}^*(R(j))} q_l^{**} s_{l,R(j)} \theta_{l,R(j)} \tag{6.41}$$

See Table 6.1 for the definition of the variables used in this algorithm.

Where 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

$$\boldsymbol{u}_{ck} \approx \frac{1}{A(\Omega_k)} \int_{\Omega_k} \nabla \cdot (\boldsymbol{u}(\boldsymbol{x} - \boldsymbol{x}_{ck}))\, \mathrm{d}\Omega = \frac{1}{A(\Omega_k)} \int_{\partial\Omega_k} (\boldsymbol{x} - \boldsymbol{x}_{ck}) \boldsymbol{u} \cdot \boldsymbol{n}\, \mathrm{d}\Gamma, \tag{6.42}$$

where Ω_k is the control volume, i.e. the cell k, $\partial\Omega_k$ the boundary of the control volume, $A(\Omega_k)$ its area and $\boldsymbol{n}$ is an outward orthonormal vector.

Remark 6.2.8. We will *not* discuss whether $\boldsymbol{u}_{ck}$ represents a cell-averaged or nodal value. Nevertheless, Equation (6.42) 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 Equation (6.42) in cell k is

$$\boldsymbol{u}_{ck} = \sum_{j\in\{l\in\mathcal{J}(k)|s_{l,k}=1\}} \boldsymbol{w}_{cLj} u_j \quad + \sum_{j\in\{l\in\mathcal{J}(k)|s_{l,k}=-1\}} \boldsymbol{w}_{cRj} u_j \tag{6.43}$$

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.

Remark 6.2.9. 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'

Algorithm 5 setlinktocenterweights: compute weight vectors $\boldsymbol{w}_{cLj}$ and $\boldsymbol{w}_{cRj}$ in the cell-center reconstruction of Equation (6.43)

$$\boldsymbol{w}_{cLj} = \frac{\alpha_j \Delta x_j \boldsymbol{n}_j w_{uj}}{A(\Omega_{L(j)})} \tag{6.44}$$

$$\boldsymbol{w}_{cRj} = \frac{(1-\alpha_j) \Delta x_j \boldsymbol{n}_j w_{uj}}{A(\Omega_{R(j)})} \tag{6.45}$$

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

$$\boldsymbol{u}_{q_k} = \frac{1}{h_{\zeta_k}} \left(\sum_{j \in \mathcal{J}(k) | L(j)=k} h_{uj} \boldsymbol{w}_{cL_j} u_j \quad + \sum_{j \in \mathcal{J}(k) | R(j)=k} h_{uj} \boldsymbol{w}_{cR_j} u_j \right) \tag{6.46}$$

if iPerot $= 2$ **then**

$$\boldsymbol{u}_{ck} = \boldsymbol{u}_{q_k} \tag{6.47}$$

else

$$\boldsymbol{u}_{ck} = \sum_{j \in \mathcal{J}(k) | L(j)=k} \boldsymbol{w}_{cL_j} u_j \quad + \sum_{j \in \mathcal{J}(k) | R(j)=k} \boldsymbol{w}_{cR_j} u_j \tag{6.48}$$

end if

$$\boldsymbol{u}_{uj} = \begin{cases} \boldsymbol{u}_{cL(j)}, & q_{aj} > 0 \\ \boldsymbol{u}_{cR(j)}, & q_{aj} < 0 \\ 0, & q_{aj} = 0 \end{cases} \tag{6.49}$$

Table 6.1: *Definition of the variables used in Algorithm (4)*

iadv	$\mathcal{J}^*$	q_l^*	q_l^{**}	$\theta_{l,\{L,R\}(j)}$	$u^*_{\{L,R\}j}$	A_{Lj}	A_{Rj}
0	-	-	-	-	0	0	
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	$\boldsymbol{u}_{c\{L,R\}(j)} \bullet \boldsymbol{n}_j$	$\frac{\alpha_j}{V_{L(j)}}$	$\frac{1-\alpha_j}{V_{R(j)}}$
34	$\mathcal{J}$	q_{al}	q_{al}	0	u_j	$\frac{\alpha_j}{h_{wj} b_{AL(j)}}$	$\frac{1-\alpha_j}{h_{wj} b_{AR(j)}}$
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	$\frac{\alpha_j}{h_{wj} b_{AL(j)}}$	$\frac{1-\alpha_j}{h_{wj} b_{AR(j)}}$
38	$\mathcal{J}$	q_l	q_l	0	$\boldsymbol{u}_{c\{L,R\}(j)} \bullet \boldsymbol{n}_j$	$\frac{\alpha_j}{V_{Au\,L(j)}}$	$\frac{1-\alpha_j}{V_{Au\,R(j)}}$
333	$\mathcal{J}$	q_{al}	q_{al}	0	u_j	$\frac{\alpha_j}{V_{Au\,L(j)}}$	$\frac{1-\alpha_j}{V_{Au\,R(j)}}$

reconstructed velocity vector. It is used for the tangential velocity v_j component at the faces:

$$v_j = \left(\alpha_j \boldsymbol{u}_{q_{L(j)}} + (1-\alpha_j)\boldsymbol{u}_{q_{R(j)}}\right) \times \boldsymbol{n}_j \tag{6.50}$$

Remark 6.2.10. 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 Equation (6.29). Higher order corrections are added in Algorithm (10), explained later.

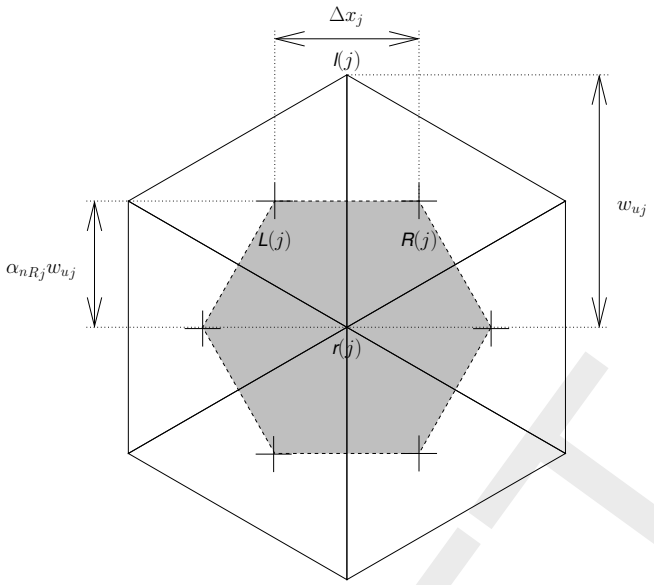


Figure 6.7: *Nodal interpolation from cell-centered values; contribution from face j to node $r(j)$; the shaded area indicates the control volume Ω_n.*

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 $\boldsymbol{u} \cdot \boldsymbol{n}$, 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 Figure 6.7 and interpolate to the nodes, say Φ_n, as follows:

$$\Phi_n \approx \frac{1}{2A(\Omega_n)} \int_{\Omega_n} \nabla \cdot (\Phi(\boldsymbol{x} - \boldsymbol{x}_n))\, \mathrm{d}\Omega = \frac{1}{2A(\Omega_n)} \int_{\partial\Omega_n} \Phi(\boldsymbol{x} - \boldsymbol{x}_n) \cdot \boldsymbol{n}\, \mathrm{d}\Gamma, \quad (6.51)$$

where

Ω_n	is the node-based control volume,
$\partial\Omega_n$	the boundary of the control volume,
$\mathrm{d}\Gamma$	its boundary,
$A(\Omega_n)$	the area of the control volume,
$\boldsymbol{n}$	the outward normal vector and
$\boldsymbol{x}_n$	are the node coordinates.

Remark 6.2.11. Equation (6.51) is a second order approximation if the node is located sufficiently close to the mass center of the control volume. In the example of Figure 6.7 this is indeed the case, but not necessarily for general meshes.

The discretization of Equation (6.51) at node i is

$$\Phi_i = \sum_{j \in \{l | l(l) = i\}} w_{Lj} \frac{1}{2} (\Phi_{cL(j)} + \Phi_{R(j)}) + \sum_{j \in \{l | r(l) = i\}} w_{Rj} \frac{1}{2} (\Phi_{L(j)} + \Phi_{R(j)}), \quad (6.52)$$

with weights w_{iLj} and w_{iRj} computed as

$$w_{iLj} = \frac{\frac{1}{2}\alpha_{iLj}\Delta x_j w_{uj}}{\sum_{l\in\{m|l(m)=l(j)\}} \frac{1}{2}\alpha_{iLj}\Delta x_l w_{ul} + \sum_{l\in\{l|r(m)=l(j)\}} \frac{1}{2}\alpha_{iRl}\Delta x_l w_{ul}} \tag{6.53}$$

$$w_{iRj} = \frac{\frac{1}{2}\alpha_{nRj}\Delta x_j w_{uj}}{\sum_{l\in\{m|l(m)=r(j)\}} \frac{1}{2}\alpha_{iLl}\Delta x_l w_{ul} + \sum_{l\in\{m|r(m)=r(j)\}} \frac{1}{2}\alpha_{iRl}\Delta x_l w_{ul}} \tag{6.54}$$

Note that $\alpha_{iLj}\Delta x_j$ and $\alpha_{iRj}\Delta x_j$ approximate the components of $(\boldsymbol{x} - \boldsymbol{x}_i) \cdot \boldsymbol{n}$ in Equation (6.53) and Equation (6.54) for node i and face j, which in D-Flow FM are computed as

$$\alpha_{iLj} = \frac{\|\frac{1}{2}(\boldsymbol{x}_{\zeta_{L(j)}} + \boldsymbol{x}_{\zeta_{R(j)}}) - \boldsymbol{x}_{il(j)}\|}{\|\frac{1}{2}(\boldsymbol{x}_{\zeta_{L(j)}} + \boldsymbol{x}_{\zeta_{R(j)}}) - \boldsymbol{x}_{il(j)}\| + \|\frac{1}{2}(\boldsymbol{x}_{\zeta_{L(j)}} + \boldsymbol{x}_{\zeta_{R(j)}}) - \boldsymbol{x}_{ir(j)}\|}, \tag{6.55}$$

$$\alpha_{iRj} = 1 - \alpha_{nLj}, \tag{6.56}$$

where

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

Remark 6.2.12. A more straightforward approach, employing the properties of an orthogonal mesh and using $w_{uj} := \|\boldsymbol{x}_{ir(j)} - \boldsymbol{x}_{il(j)}\|$, would be:

$$\alpha_{iLj} = \frac{\left(\frac{1}{2}(\boldsymbol{x}_{\zeta_{L(j)}} + \boldsymbol{x}_{\zeta_{R(j)}}) - \boldsymbol{x}_{il(j)}\right) \cdot (\boldsymbol{x}_{ir(j)} - \boldsymbol{x}_{il(j)})}{w_{uj}^2}. \tag{6.57}$$

In D-Flow FM the interpolation of cell-centered velocity vectors to nodal velocity vectors is as in Equation (6.51). 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.

$$\boldsymbol{u}_i = \sum_{j\in\{l|l(l)=i\}} w_{iLj}\frac{1}{2}(\boldsymbol{u}_{cL(j)} + \boldsymbol{u}_{cR(j)}) + \sum_{j\in\{l|r(l)=i\}} w_{iRj}\frac{1}{2}(\boldsymbol{u}_{cL(j)} + \boldsymbol{u}_{cR(j)}) \tag{6.58}$$

When "jacomp" $= 1$, however, the two components of the velocity vector (u_x, u_y), get different weights. If we say $\boldsymbol{u}_c =: (u_{cx}, u_{cy})^{\mathrm{T}}$ and $\boldsymbol{u}_n =: (u_{nx}, u_{ny})^{\mathrm{T}}$, 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.

Remark 6.2.13. For nodes *not* on the mesh boundary, it is unclear why the weights in Algorithm (7) for vector interpolation should differ from Equation (6.53) and Equation (6.54) for scalar interpolation, which is the case if "jacomp" $= 1$ in Algorithm (8). The option "jacomp" $= 1$ may need to be reconsidered.

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

$$u_{nxi} = \sum_{j\in\{l|l(l)=i\}} w_{nxLj}\frac{1}{2}(u_{cxL(j)} + u_{cxR(j)}) + \sum_{j\in\{l|r(l)=i\}} w_{nxRj}\frac{1}{2}(u_{cxL(j)} + u_{cxR(j)}) \tag{6.59}$$

$$u_{nyi} = \sum_{j\in\{l|l(l)=i\}} w_{nyLj}\frac{1}{2}(u_{cyL(j)} + u_{cyR(j)}) + \sum_{j\in\{l|r(l)=i\}} w_{nyRj}\frac{1}{2}(u_{cyL(j)} + u_{cyR(j)}) \tag{6.60}$$

Algorithm 8 setlinktocornerweights: compute weights w_{nxLj}, w_{nxRj}, w_{nyLj} and w_{nyRj} in the nodal interpolation of Algorithm (7), Equation (6.59) and Equation (6.60)

$$\chi_{xj} = \begin{cases} 2\max(10^{-6}, |\boldsymbol{n}_j \cdot \boldsymbol{e}_x|), & \text{jacomp} = 1 \\ 1, & \text{otherwise} \end{cases} \tag{6.61}$$

$$\chi_{y_j} = \begin{cases} 2\max(10^{-6}, |\boldsymbol{n}_j \cdot \boldsymbol{e}_y|), & \text{jacomp} = 1 \\ 1, & \text{otherwise} \end{cases} \tag{6.62}$$

if $r(j)$ and $l(j)$ are not boundary nodes **then**

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

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

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

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

else

unmentioned, at least one of the nodes is a boundary node

end if

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

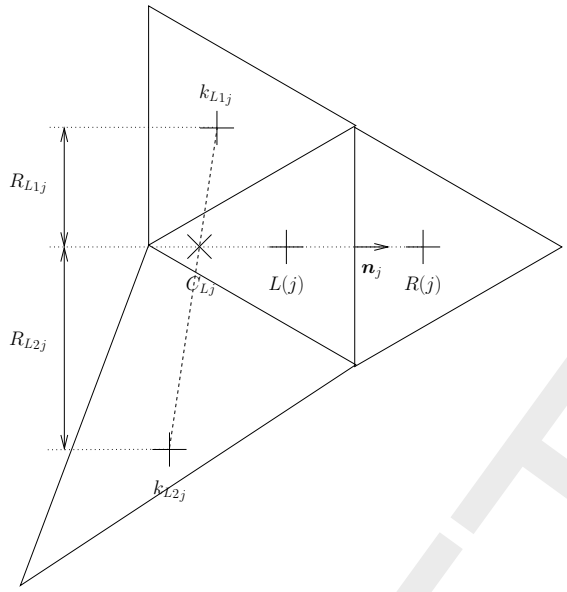


Figure 6.8: *Higher-order reconstruction of face-based velocity $\boldsymbol{u}_{uj}$, from the left*

$\boldsymbol{x}_{\zeta_k} \cdot \boldsymbol{e}_x$:	`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_{nxi}:	`ucnx(i),`	u_{ny_i}:	`ucny(i),`
α_{nLj}:	`acn(2,j),`	α_{nRj}:	`acn(1,j),`
w_{nxLj}:	`wcnx4(j),`	$w_{ny_{Lj}}$:	`wcny4(j),`
w_{nxRj}:	`wcnx3(j),`	$w_{ny_{Rj}}$:	`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 Equation (6.29). A first-order approximation is already available from Algorithm (6), Equation (6.49). 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 6.2.14. 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 $\{R(j), L(j), k_{L1j}, k_{L2j}\}$. An example is shown in Figure 6.8. 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_{L2j} 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. $\Phi_{C_{Lj}}$ (being one of the two cell-centered velocity vector components as we will see later), is interpolated as follows:

$$\Phi_{C_{Lj}} = s_{L1j}\Phi_{k_{L1j}} + s_{L2j}\Phi_{k_{L2j}}. \tag{6.63}$$

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

Algorithm 9 setupwslopes: determine the cells k_{L1} and k_{L2}, and compute weights s_{L1} and s_{L2} in Equation (6.63) 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_{L2j} according to the following rules

- ◇ cells k_{L1j} and k_{L2j} 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 $(\boldsymbol{x}_{C_{Lj}} - \boldsymbol{x}_{\zeta_{R(j)}}) \cdot (\boldsymbol{x}_{\zeta_{L(j)}} - \boldsymbol{x}_{\zeta_{R(j)}}) > 1.2$

if $R_{L_{1j}} < 0.1\Delta x_j$ and the "advection type" of face between k_{L1j} and $L(j) \notin \{6, 8\}$ **then**
$s_{L1j} = 1$, $s_{L2j} = 0$, $\gamma_{Lj} = \frac{\Delta x_j}{\|\boldsymbol{x}_{\zeta_{L(j)}} - \boldsymbol{x}_{\zeta_{k_{L1j}}}\|}$
else if cell k_{L2j} found and the "advection type" of faces between k_{L1j} and $L(j)$, and between k_{L2j} and $L(j) \notin \{6, 8\}$ **then**
$s_{L1j} = \frac{\|\boldsymbol{x}_{C_{Lj}} - \boldsymbol{x}_{\zeta_{k_{L2j}}}\|}{\|\boldsymbol{x}_{\zeta_{k_{L1j}}} - \boldsymbol{x}_{\zeta_{k_{L2j}}}\|}$, $s_{L2j} = 1 - s_{L1j}$, $\gamma_{Lj} = \frac{\Delta x_j}{\|\boldsymbol{x}_{\zeta_{L(j)}} - \boldsymbol{x}_{C_{Lj}}\|}$
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 Equation (6.63) is applied to the Cartesian components of the velocity vector. In such a manner values at $u_{xC_{Lj}}$ and $u_{yC_{Lj}}$ are obtained. The reconstruction is then performed with Algorithm (10). Note that in Algorithm (10), Equation (6.66) and Equation (6.67), γ_{Lj} accounts for the non-uniform spacing along the line through cell centers $L(j)$

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:

$$\boldsymbol{u}_{C_{L_j}} = s_{L1j}\boldsymbol{u}_{ck_{L1_j}} + s_{L2j}\boldsymbol{u}_{ck_{L2_j}} \tag{6.64}$$

$$\boldsymbol{u}_{C_{R_j}} = s_{R1j}\boldsymbol{u}_{ck_{R1_j}} + s_{R2j}\boldsymbol{u}_{ck_{R2_j}} \tag{6.65}$$

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

compute slope ratio in limiter, for each velocity component

$$r_x = \frac{u_{cR(j)_x} - u_{cL(j)_x}}{u_{cL(j)_x} - u_{C_{L_j x}}} \frac{1}{\gamma_{L_j}} \tag{6.66}$$

$$r_y = \frac{u_{cR(j)_y} - u_{cL(j)_y}}{u_{cL(j)_y} - u_{C_{L_j y}}} \frac{1}{\gamma_{L_j}} \tag{6.67}$$

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

$$\boldsymbol{u}_{uj} = \boldsymbol{u}_{cL(j)} + \alpha_j \max(1 - \frac{\Delta t|u_j|}{\Delta x_j}, 0) \begin{pmatrix} \Psi(r_x) & 0 \\ 0 & \Psi(r_y) \end{pmatrix} (\boldsymbol{u}_{cR(j)} - \boldsymbol{u}_{cL(j)}) \tag{6.68}$$

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

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 6.2.

Table 6.2: *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 6.2.15. 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 6.2.16. 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_{L1j}:	`klnup(1,j)`,	k_{R1j}:	`klnup(4,j)`,
k_{L2j}:	`klnup(2,j)`,	k_{R2j}:	`klnup(5,j)`,
s_{L1j}:	`slnup(1,j)`,	s_{R1j}:	`slnup(4,j)`,
s_{L2j}:	`slnup(2,j)`,	s_{R2j}:	`slnup(5,j)`,
γ_{Lj}:	`slnup(3,j)`,	γ_{Rj}:	`slnup(6,j)`.

Momentum diffusion

The momentum diffusion term in Equation (6.7) is

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

In D-Flow FM, this term is modified as

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

where

$$p = \begin{cases} 1, & \texttt{istresstype} = 3, \\ 1, & \texttt{istresstype} = 5, \\ 0, & \text{otherwise.} \end{cases} \tag{6.69}$$

Remark 6.2.17. 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 $\nabla \cdot (\nu h(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^{\mathrm{T}}))$ 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

$$\nabla \cdot \left(\nu h^p(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^{\mathrm{T}})\right)\big|_{\Omega_k} \approx \boldsymbol{d}_k, \tag{6.70}$$

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

$$\nabla \cdot (\nu h^p(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^{\mathrm{T}}))\big|_{\Gamma_j} \cdot \boldsymbol{n}_j \approx \left(\alpha_j \boldsymbol{d}_{L(j)} + (1-\alpha_j)\boldsymbol{d}_{R(j)}\right) \cdot \boldsymbol{n}_j, \tag{6.71}$$

where again α_j is the non-dimensional distance from the left cell center to the face, see Figure 6.6.

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

$$\nabla \cdot \left(\nu h^p(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^{\mathrm{T}})\right)\big|_{\Omega_k} = \frac{1}{A(\Omega)} \int\limits_{\partial\Omega_k} \nu h^p \left(\frac{\partial \boldsymbol{u}}{\partial n} + \nabla \boldsymbol{u} \cdot \boldsymbol{n}\right) \mathrm{d}\Gamma, \tag{6.72}$$

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

$$\nabla \cdot \left(\nu h^p (\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^{\mathrm{T}})\right)\big|_{\Omega_k} \approx \boldsymbol{d}_k =$$
$$\begin{cases} \frac{1}{b_{Ak}} \sum\limits_{j \in \mathcal{J}(k)} \boldsymbol{t}_{uj} w_{uj} s_{j,k}, & p = 0, \\ \frac{1}{b_{Ak}} \sum\limits_{j \in \mathcal{J}(k)} \boldsymbol{t}_{uj} w_{uj} \min(h_{\zeta_{L(j)}}, h_{\zeta_{R(j)}}) s_{j,k}, & p = 1, \quad \texttt{istresstype} = 3, \\ \frac{1}{b_{Ak}} \sum\limits_{j \in \mathcal{J}(k)} \boldsymbol{t}_{uj} A_{uj} s_{j,k}, & p = 1, \quad \texttt{istresstype} = 5. \end{cases} \tag{6.73}$$

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

$$\boldsymbol{t}_{uj} = \nu_j \left(\begin{pmatrix} 1 + n_{x_j}^2 & n_{xj} n_{y_j} \\ n_{xj} n_{y_j} & 1 + n_{y_j}^2 \end{pmatrix} \frac{\boldsymbol{u}_{cR(j)} - \boldsymbol{u}_{cL(j)}}{\Delta x_j} + \begin{pmatrix} -n_{xj} n_{y_j} & -n_{y_j}^2 \\ n_{x_j}^2 & n_{xj} n_{y_j} \end{pmatrix} \frac{\boldsymbol{u}_{nl(j)} - \boldsymbol{u}_{nr(j)}}{w_{uj}} \right). \tag{6.74}$$

For $\texttt{istresstype}$ = 6 the viscous stresses are completely expressed in normal and tangential components and essentially the same expression as Equation (6.74) is obtained.

Note that $\boldsymbol{u}_{ck}$ (here with $k \in \{L(j), R(j)\}$) and $\boldsymbol{u}_{ni}$ (here with $i \in \{l(j), r(j)\}$) 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 Equation (6.26) is finally obtained by bringing it in non-conservative form and interpolation at the faces

$$\mathcal{A}_{\mathrm{e}j} = -\left(\frac{\alpha_j \boldsymbol{d}_{L(j)}}{H_{L_j}^p} + \frac{(1 - \alpha_j) \boldsymbol{d}_{R(j)}}{H_{R_j}^p} \right) \cdot \boldsymbol{n}_j, \tag{6.75}$$

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

Remark 6.2.18. 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 6.2.19. In the discretization of the diffusive fluxes, the area of face j is approximated by $w_{uj} \min(h_{\zeta_{L(j)}}, h_{\zeta_{R(j)}})$ for $\texttt{istresstype}$ '3'. It is unclear why the actual cross-sectional area A_{uj} does not suffice. For the other $\texttt{istresstype}$s, see Remark 6.2.17.

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

compute viscous stresses

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

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

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

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

Turbulence modelling: Smagorinsky, Elder

For `istresstype` 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

$$\nu_j = E \, \frac{\kappa}{6} h_{uj} \frac{\sqrt{g}}{C} \sqrt{u_j^2 + v_j^2}, \tag{6.76}$$

where E is the user-specified Elder coefficient and C is the (time- and space varying) Chézy coefficient. And in case of the Smagorinsky model

$$\nu_j = \left(C_S \sqrt{\Delta x_j w_{uj}}\right)^2 \left| \sqrt{2\frac{\partial u_n}{\partial n}^2 + \left(\frac{\partial u_n}{\partial t} + \frac{\partial u_t}{\partial n}\right)^2 + 2\frac{\partial u_t}{\partial t}^2} \right|_j, \tag{6.77}$$

where C_S is a user-specified Smagorinsky coefficient and the velocity derivatives at face j are approximated with finite differences similarly to Equation (6.74).

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

$$\frac{\partial \boldsymbol{u}}{\partial t} = \nabla \cdot (\nu \nabla \boldsymbol{u}) \tag{6.78}$$

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

$$\frac{\boldsymbol{u}_{ck}^{n+1} - \boldsymbol{u}_{ck}^{n}}{\Delta t} = \frac{1}{V_k^n} \sum_{j \in \mathcal{J}(k)} \nu_j A_{uj}^{n} \frac{\boldsymbol{u}_{R(j)}^n - \boldsymbol{u}_{L(j)}^n}{\Delta x_j} s_{j,k}. \tag{6.79}$$

Remark 6.2.20. If we disregard the differences due the interpolation to the faces, and the missing terms $\nabla \cdot (h\nu \nabla \boldsymbol{u}^{\mathrm{T}})$, the discretization of the model equation only conforms to the form of the momentum diffusion term if `istresstype`=5, and if $V_k = b_{Ak} h_{\zeta_k}$ (no non-linear iterations, see Algorithm (20)), as can be seen by comparing with Algorithm (11).

Equation (6.79) can be rewritten as

$$\boldsymbol{u}_{c_k}^{n+1} = \left(1 - \frac{\Delta t}{V_k^n} \sum_{j \in \mathcal{J}(k)} \frac{\nu_j {A_{u_j}}^n}{\Delta x_j}\right) \boldsymbol{u}_{c_k}^{n} + \frac{\Delta t}{V_k^n} \sum_{j \in \mathcal{J}(k)} \frac{\nu_j {A_{u_j}}^n}{\Delta x_j} \boldsymbol{u}_{O(k,j)}^n, \tag{6.80}$$

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

$$O(k,j) = \mathit{L}(j) + \mathit{R}(j) - k. \tag{6.81}$$

We require that

$$0 \leq \frac{\Delta t}{V_k^n} \sum_{j \in \mathcal{J}(k)} \frac{\nu_j {A_{u_j}}^n}{\Delta x_j} \leq 1, \tag{6.82}$$

which is satisfied if we limit the viscosity coefficient by

$$\nu_j \leq \frac{1}{N} \frac{\Delta x_j}{{A_{u_j}}^n \Delta t} \min(V_{\mathit{L}(j)}^n, V_{\mathit{R}(j)}^n), \tag{6.83}$$

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 Equation (6.73) 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 6.4.7.

Perot's reconstruction of the cell-center velocity:

$$\boldsymbol{u}_c = \frac{1}{V_k} \sum_{j \in \mathcal{J}(k)} u_j 1_{j,k} \left(\boldsymbol{x}_j - \boldsymbol{x}_c\right) \tag{6.84}$$

Bed friction

The bottom friction can be expressed on the flow links as follows,

$$\frac{1}{h}\frac{\boldsymbol{\tau_b}}{\rho}\bigg|_j \cdot \boldsymbol{n}_j \approx -\frac{g\|\boldsymbol{u}_j\|}{C^2\hat{h}_j}u_j, \tag{6.85}$$

where $\hat{h}_j = A_j/P_j$ and $A_j = h_j\, dy_j$, as shown in Figure 6.9 The Chézy formula for determining the velocity is:

$$u_j = C\sqrt{\hat{h}_j i} \tag{6.86}$$

$$q_j = A_j C\sqrt{\hat{h}_j i} = \frac{A_j \hat{h}_j^{2/3}}{n}\sqrt{i} \tag{6.87}$$

$$Q = \sum_j q_j = \frac{\sum A_j \hat{h}_j^{2/3}}{n}\sqrt{i} = K\sqrt{i} \tag{6.88}$$

$$U = \frac{Q}{A} = \frac{K}{A}\sqrt{i} = \frac{\frac{1}{n}\sum A_j \hat{h}_j^{2/3}}{\sum A_j}\sqrt{i} \tag{6.89}$$

$$C_{fu} = \frac{g}{C^2\hat{h}_j} = g\left(\frac{\sum A_j}{\frac{1}{n}\sum A_j \hat{h}_j^{2/3}}\right)^2 = g\left(\frac{A}{K}\right)^2 \tag{6.90}$$

where $\hat{h}_j$ and K vary for different conveyance schemes. The differences in the various schemes are in the way of defining the hydraulic radius $\hat{h}_j$ and K in Equation (6.90).

In D-Flow FM we have as a default setting of `Ibedlevtype=3`. This is applied for estimation of the bedlevel at flow links. It assumes variation in cross flow direction of the local waterdepth, flow velocity and friction coefficient. This is called the analytic 2D conveyance method (type 3). This method shows good grid convergence. A more simple variant is 1D conveyance (type 2). Another method is only by taking the variation of the waterdepth into account across flow links, which leads to a hydraulic radius equal to the cross-sectional area divided by the wet perimeter (type 1). These methods are described in Algorithm (12). For derivation of formulations in Algorithm (12) we refer the reader to Appendix A.

When assuming a constant bedlevel at a flow link, one can either average between bedlevels at waterlevel points (type 0), or between the levels of two cornerpoints (type -1). The former one leads to lower bed friction in general, because the bedlevel at a waterlevelpoint is taken as the lowest connect link level. Formulations for conveyance types -1 and 0 are shown in Algorithm (13).

Wind friction

The wind friction can be expressed on the flow links as follows,

$$\frac{1}{h}\frac{\boldsymbol{\tau_w}}{\rho}\bigg|_j \cdot \boldsymbol{n}_j \approx C_d\frac{\rho_a}{\rho}\frac{\|\tilde{\boldsymbol{u}}_{\mathbf{10}}\|\tilde{\boldsymbol{u}}_{\mathbf{10}} \cdot \boldsymbol{n}_j}{h_{uvj}}. \tag{6.99}$$

where h_{uvj} is the distance-weighted water depth at the face, determined using the depths at the adjacent cell centres $h_{\zeta_{L(j)}}$ and $h_{\zeta_{R(j)}}$, and the non-dimensional distance α_j of the cell centres to the face, see Figure 6.6:

$$h_{uvj} = \max(\alpha_j h_{\zeta_{L(j)}} + (1-\alpha_j)h_{\zeta_{R(j)}}, \varepsilon_{h_\zeta}), \tag{6.100}$$

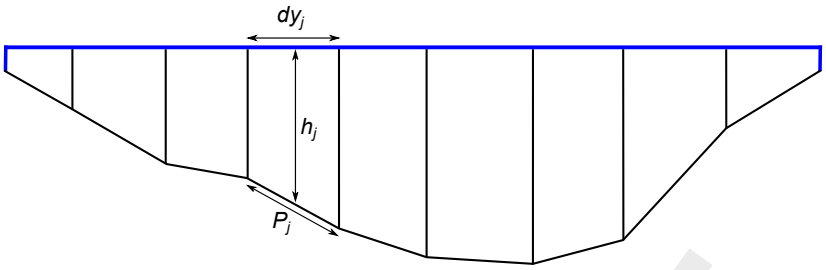


Figure 6.9: *Cross sectional bed bathemetry perpendicular to the flow direction.*

Algorithm 12 getprof2D: compute conveyance types 1, 2 and 3.

$$\hat{h}_j = A_j/P_j \tag{6.91}$$

$$\gamma_i = n\alpha_i \left(1+\alpha_i^2\right)^{1/4} \tag{6.92}$$

$$\gamma_i' = n\alpha_i \left(1+\alpha_i^2+\alpha_i'^2\right)^{1/4} \tag{6.93}$$

$$K_2 = \frac{3}{8}\left(h_i^{8/3} - h_{i+1}^{8/3}\right)/\gamma_i \tag{6.94}$$

$$K_3 = \left(\beta_i - h_i\frac{\delta}{\alpha_i}\right)K_2 + \frac{3}{11}\frac{\delta}{\alpha_i\gamma_i'}\left(h_i^{11/3} - h_{i+1}^{11/3}\right) \tag{6.95}$$

$$C_{fu} = \begin{cases} g/C^2\hat{h}_j, & \text{conveyance type} = 1, \\ g\left(A_j/K_2\right)^2, & \text{conveyance type} = 2, \\ g\left(A_j/K_3\right)^2, & \text{conveyance type} = 3, \end{cases} \tag{6.96}$$

Algorithm 13 setcfuhi: compute conveyance types 0 and -1.

$$C_{fu} = \frac{g}{C^2\hat{h}_j} \tag{6.97}$$

where

$$\hat{h}_j = \begin{cases} \max\left(\varepsilon_h, h_{uj}\right), & \text{conveyance type} = -1, \\ \max\left(\varepsilon_h, h_{uvj}\right), & \text{conveyance type} = 0, \end{cases} \tag{6.98}$$

where ε_{h_ζ} is a threshold.

By default, the effective wind velocity $\tilde{\boldsymbol{u}}_{\mathbf{10}}$ is simply chosen equal to the wind velocity at 10 m above the free surface $\boldsymbol{u}_{\mathbf{10}}$ and the wind shear stress is computed as:

$$\left.\frac{1}{h}\frac{\boldsymbol{\tau_w}}{\rho}\right|_j \cdot \boldsymbol{n}_j \approx C_d \frac{\rho_a}{\rho} \frac{\|\boldsymbol{u}_{10}\|}{h_{uvj}} \boldsymbol{u}_{\mathbf{10}} \cdot \boldsymbol{n}_j, \tag{6.101}$$

In D-Flow FM, there is the possibility to compute wind shear stress based on the *relative* wind velocity, i.e. relative to the flow velocity. This becomes important when the flow is predominantly forced by the wind and the flow velocity (in 3D, at the free surface) approaches the wind velocity. In this way, one can avoid that the wind still forces the flow, despite a zero (or small) difference between flow and wind speed. For the relative wind formulation, the equations above are modified to be:

$$\left.\frac{1}{h}\frac{\boldsymbol{\tau_w}}{\rho}\right|_j \cdot \boldsymbol{n}_j \approx C_d \frac{\rho_a}{\rho} \frac{\|\boldsymbol{u}_{10} - \boldsymbol{u}_j\|}{h_{uvj}} (\boldsymbol{u}_{\mathbf{10}} - \boldsymbol{u}_j) \cdot \boldsymbol{n}_j, \tag{6.102}$$

This second option can be chosen by the user using the keyword `Relativewind`.

The wind shear stress coefficient C_d can either be specified directly by the user or computed using a number of wind drag formulations. The following options are available:

- ◇ a constant drag coefficient,
- ◇ a linearly varying drag coefficient according to Smith and Banke (1975),
- ◇ a piecewise linearly varying drag coefficient according to Smith and Banke (1975),
- ◇ a dependency according to Charnock (1955),
- ◇ a dependency according to Hwang (2005a) and Hwang (2005b).

For a Smith & Banke type formulation, the additional entries for the `Cdbreakpoints` and `Windspeedbreakpoints` need to be prescribed by the user.

In the following sections, the implementations of these different options are described.

Smith & Banke type formulation

When specifying a Smith & Banke type dependency, the definition as sketched in Figure 6.10 should be kept in mind.

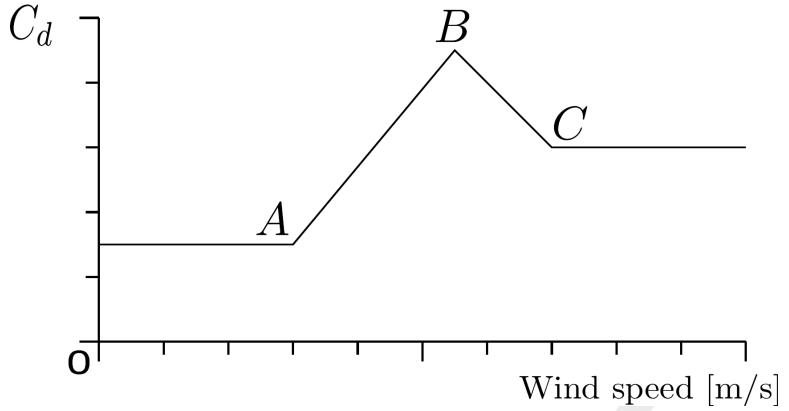


Figure 6.10: *Prescription of the dependency of the wind drag coefficient C_d on the wind speed is achieved by means of at least 1 point, with a maximum of 3 points.*

From this sketch, it can be seen that the wind drag is considered as dependent on the wind speed in a piecewise linear way. The options, that are facilitated in this respect, are:

- ◇ define *one* set of coordinates (breakpoint A), specifying a constant drag coefficient, valid for all wind speeds,
- ◇ define *two* sets of coordinates (breakpoints A and B), specifying a linearly varying dependency for one range of wind speeds,
- ◇ define *three* sets of coordinates (breakpoints A, B and C), specifying a piecewise linear dependency for two ranges of wind speeds.

Remark that for the latter two options, the drag coefficient is taken constant for wind speeds lower/higher than the lowest/highest specified wind speed, with a drag coefficient equal to the drag coefficient associated with the lowest/highest specified lowest/highest wind speed. In case of three breakpoints, the expression reads:

$$C_d\left(U_{10}\right) = \begin{cases} C_d^A, & U_{10} \leq U_{10}^A, \\ C_d^A + \left(C_d^B - C_d^A\right) \dfrac{U_{10} - U_{10}^A}{U_{10}^B - U_{10}^A}, & U_{10}^A \leq U_{10} \leq U_{10}^B, \\ C_d^B + \left(C_d^C - C_d^B\right) \dfrac{U_{10} - U_{10}^B}{U_{10}^C - U_{10}^B}, & U_{10}^B \leq U_{10} \leq U_{10}^C, \\ C_d^C, & U_{10}^C \leq U_{10}, \end{cases} \tag{6.103}$$

By means of the entries `Cdbreakpoints` and `Windspeedbreakpoints`, the coordinates of the breakpoints (see Figure 6.10) can be specified. Typical values associated with the Smith and Banke (1975) formulation are $C_d = 6.3 \times 10^{-4}$ for $U = 0$ m/s and $C_d = 7.23 \times 10^{-3}$ for $U = 100$ m/s.

Charnock formulation

The Charnock formulation (see Charnock (1955)) is based on the assumption of a fully developed turbulent boundary layer of the wind flow over the water surface. The associated wind speed profile follows a logithmic shape. In the Charnock formulation, the wind speed is considered at 10 meters above the free water surface, hence yielding the following expression:

$$\frac{U_{10}}{u_*} = \frac{1}{\kappa} \ln\left(\frac{z_{10}}{z_0}\right) \tag{6.104}$$

with κ the Von Kármán constant, z_{10} the distance to the water surface (equal to 10 m), u_* the friction velocity and U_{10} the wind speed at 10 m above the water surface. The drag coefficient C_d is defined as:

$$C_d = \frac{u_*^2}{U_{10}^2}. \tag{6.105}$$

Charnock (1955) has proposed to represent the friction of the water surface as z_0 according to:

$$z_0 = \frac{b\, u_*^2}{g}, \tag{6.106}$$

with g the gravitation acceleration and b a specific constant. Charnock (1955) has proposed $b = 0.012$. The value of the constant b can be specified in the MDU-file by the user by means of one single value for `Cdbreakpoints`. Since the above relation yields an implicit relation for u_*, the system is solved for iteratively.

Hwang formulation

The dynamic roughness could also be related to the steady state wave conditions of the flow field under consideration. The connection of the wave parameters with the drag coefficient as elaborated by Hwang (2005a) is available within D-Flow FM through `ICdtyp = 5`, given a wave field. The Hwang-formulation interprets the user defined wind speed as the wind speed at 10 m above the water surface.

The drag coefficient is computed as:

$$C_d = \left[\frac{1}{\kappa} \ln\left(\frac{k_p z_{10}}{k_p z_0}\right)\right]^{-2} \tag{6.107}$$

with $z_{10} = 10$ m, κ the Von Kármán constant. With wavelength scaling, $k_p z_0$ is the natural expression of the dimensionless roughness, where k_p is the wave number of the spectral peak, computed on the basis of the actual water depth and the provided peak period T_p as wave field. Further following Hwang (2005a),

$$k_p z_0 = \pi \exp\left(-\kappa C_{\lambda/2}^{-0.5}\right) \tag{6.108}$$

in which $C_{\lambda/2}$ is the drag coefficient at half the wavelength above surface. This parameter $C_{\lambda/2}$ is computed as:

$$C_{\lambda/2} = A_{10} \left(\frac{\omega_p U_{10}}{g}\right)^{a_{10}} \tag{6.109}$$

in which $A_{10} = 1.289 \times 10^{-3}$, $a_{10} = 0.815$, U_{10} the wind speed at 10 m above the water surface and ω_p the wave peak frequency ($\omega_p = 2\pi/T_p$). Thus, the drag coefficient C_d is defined.

Coriolis forces

[yet empty]

6.3 Temporal discretization

The spatial discretization is, as explained in section 6.2, 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 Equation (6.25)

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

then the temporal discretization of Equation (6.7) is

$$\begin{aligned}\left(\frac{1}{\Delta t}+\mathcal{A}_{\mathrm{i}j}^{n}+\frac{g\|\hat{\boldsymbol{u}}_j\|}{C^2h}\right)u_j^{n+1}=&\frac{1}{\Delta t}u_j^n-\frac{g\theta_j}{\Delta x_j}\left(\zeta_{R(j)}^{n+1}-\zeta_{L(j)}^{n+1}\right)\\&-\mathcal{A}_{\mathrm{e}j}^{n}-\frac{g(1-\theta_j)}{\Delta x_j}\left(\zeta_{R(j)}^{n}-\zeta_{L(j)}^{n}\right),\end{aligned}\tag{6.110}$$

where superscript n denotes the time level, $\hat{\boldsymbol{u}}_j$ is obtained by substituting $\hat{\boldsymbol{u}}_j=(\hat{u}_j,\boldsymbol{u}_j^n\cdot\boldsymbol{n}_j^{\perp})^{\mathrm{T}}$, $u_j^{n+1}=\hat{u}_j$, $\theta_j=0$ in Equation (6.110) 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 Equation (6.110) is summarized as

$$u_j^{n+1}=-f_{uj}^{\ n}(\zeta_{R(j)}^{n+1}-\zeta_{L(j)}^{n+1})+r_{uj}^{\ n},\tag{6.111}$$

where $f_{uj}^{\ n}$ and $r_{uj}^{\ n}$ are determined iteratively by Algorithm (14).

Algorithm 14 furu: compute $f_{uj}^{\ n}$ and $r_{uj}^{\ n}$ in $u_j^{n+1}=-f_{uj}^{\ n}(\zeta_{R(j)}^{n+1}-\zeta_{L(j)}^{n+1})+r_{uj}^{\ n}$

$$\hat{u}_j^{(0)}=u_j^{\ n}$$
$$p=0$$

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

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

end while

Here MAXITER $= 4$, $\varepsilon = 10^{-2}$ 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

$$\frac{V_k^{n+1} - V_k^n}{\Delta t} = -\sum_{j \in \mathcal{J}(k)} {A_u}_j{}^n \left(\theta_j {u_j}^{n+1} + (1-\theta_j){u_j}^n\right) s_{j,k}, \tag{6.112}$$

where $\mathcal{J}(k)$ is the set of faces that bound cell k and $s_{j,k}$ accounts for the orientation of face j with respect to cell k, i.e.

$$s_{j,k} = \begin{cases} 1, & L(j) = k \quad (\boldsymbol{n}_j \text{ is outward normal of cell } k), \\ -1, & R(j) = k \quad (\boldsymbol{n}_j \text{ is inward normal of cell } k). \end{cases} \tag{6.113}$$

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.

$$A_{uj} = h_{uj} w_j, \tag{6.114}$$

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

Substitution of Equation (6.111) in Equation (6.112) yields the following system for the water column volume at the next time instant:

$$\begin{aligned} \frac{V_k^{n+1} - V_k^n}{\Delta t} + \sum_{j \in \mathcal{J}(k)} {A_u}_j^n \theta_j {f_u}_j^n \, \zeta_k^{n+1} - \sum_{j \in \mathcal{J}(k)} {A_u}_j^n \theta_j {f_u}_j^n \, \zeta_{O(k,j)}^{n+1} = \\ - \sum_{j \in \mathcal{J}(k)} {A_u}_j^n \left[(1-\theta_j) u_j^n + \theta_j {r_u}_j^n\right] s_{j,k}, \end{aligned} \tag{6.115}$$

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

$$O(k,j) = L(j) + R(j) - k. \tag{6.116}$$

Remark 6.3.1. The flow area of face j, A_{uj}, always appears explicitly in the continuity equation, Equation (6.112).

The water level equation, Equation (6.115), is summarized as

$$\frac{V_k^{n+1} - V_k^n}{\Delta t} + B_k^n \, \zeta_k^{n+1} + \sum_{j \in \mathcal{J}(k)} C_j^n \, \zeta_{O(k,j)}^{n+1} = d_k^n, \tag{6.117}$$

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 (15).

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 (16).

The resulting set of water level cells is called $\mathcal{K}$, see Algorithm (17). The continuity equation is only applied at cells k for $k \in \mathcal{K}$.

In order to solve Equation (6.117), 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

Algorithm 15 s1ini: compute the matrix entries and right-hand side in the water level equation $\frac{V_k^{n+1}-V_k^n}{\Delta t} + B_k^n\ \zeta_k^{n+1} + \sum_{j\in\mathcal{J}(k)} C_j^n\ \zeta_{O(k,j)}^{n+1} = d_k^n$, Equation (6.117)

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

Algorithm 16 setkfs: mark the water level cells that are or may become (partially) wet with $k_{fs}(k) = 1$

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

in general, Equation (6.117) 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 p.

$$
V_k^{n+1(p+1)} = V_k^{n+1(p)} + A_k^{n+1(p)} \left(\zeta_k^{n+1(p+1)} - \zeta_k^{n+1(p)}\right), \tag{6.118}
$$

where $A_k^{n+1(p)}$ is the wet bed area of cell k at (iterative) time level $n+1(p)$. Substitution in Equation (6.117) yields

$$
\begin{aligned}
\left(\frac{1}{\Delta t} A_k^{n+1(p)} + B_k^n\right)\ \zeta_k^{n+1(p+1)} + \sum_{j\in\mathcal{J}(k)} C_j^n\ \zeta_{O(k,j)}^{n+1(p+1)} = \\
d_k^n - \frac{1}{\Delta t}\left(V_k^{n+1(p)} - V_k^n - A_k^{n+1(p)}\zeta_k^{n+1(p)}\right),
\end{aligned} \tag{6.119}
$$

which is summarized as

$$
{B_{rk}}^{n+1(p)}\ \zeta_k^{n+1(p+1)} + \sum_{j\in\mathcal{J}(k)} {C_{rj}}^n\ \zeta_{O(k,j)}^{n+1(p+1)} = {d_{rk}}^{n+1(p)}, \tag{6.120}
$$

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 (18).

Algorithm 17 pack_matrix: determine the set $\mathcal{K}$ of water level cells for which the continuity equation is solved

mark wet/dry cells, Algorithm (16)
$\mathcal{K} = \{k : k_{fs}(k) = 1\}$

Algorithm 18 s1nod: compute the matrix entries and right-hand side in the water level equation ${B_r}_k^{n+1(p)}\ \zeta_k^{n+1(p+1)} + \sum_{j\in\mathcal{J}(k)} {C_r}_j^{\ n}\ \zeta_{O(k,j)}^{n+1(p+1)} = {d_r}_k^{n+1(p)}$, Equation (6.120)

$$
\begin{aligned}
{B_r}_k^{n+1(p)} &= B_k^n + \frac{1}{\Delta t} A_k^{n+1(p)} \\
{C_r}_j^{\ n} &= C_j^n \\
{d_r}_k^{n+1(p)} &= d_k^n - \frac{1}{\Delta t}\left(V_k^{n+1(p)} - V_k^n - A_k^{n+1(p)}\zeta_k^{n+1(p)}\right).
\end{aligned}
$$

Note that we did not describe the water level boundary conditions in Algorithm (18).

The unknown water levels $k \in \mathcal{K}$ in Equation (6.120) are solved with a Krylov solver as will be explained in section 6.3.1.

During the iterative process, the water level $\zeta_k^{n+1(p+1)}$ 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 is eliminated from the system by setting the water levels of its bounding faces to zero (type 2, default), see Algorithm (19).

Algorithm 19 poshcheck: check positivity of water height

if $\zeta_k^{n+1(p+1)} < bl_k$ **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 $V_k^{n+1(p+1)}$ and wet bed area $A_k^{n+1(p+1)}$ of cell k are computed with Algorithm (20). Note that if no non-linear iterations are performed, the wet bed area is set equal to the cell bed area b_{Ak}.

Algorithm 20 volsur: compute water-column volume $V_k^{n+1(p+1)}$ and wet bed area $A_k^{n+1(p+1)}$

if no non-linear iterations **then**

$$
\begin{aligned}
V_k^{n+1(p+1)} &= b_{Ak}\max(\zeta_k^{n+1(p+1)} - bl_k, 0) \\
A_k^{n+1(p+1)} &= b_{Ak}
\end{aligned}
$$

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 Equation (6.111), see Algorithm (21). Two discharges

are computed at the next time level, namely

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

$$q_{a_j}{}^{n+1} = A_{u_j}{}^n u_j^{n+1}. \tag{6.122}$$

Discharge q_j^{n+1} satisfies the continuity equation Equation (6.112)

$$\frac{V_k^{n+1} - V_k^n}{\Delta t} = -\sum_{j \in \mathcal{J}(k)} q_j^{n+1} s_{j,k}, \tag{6.123}$$

and appears for example in the discretization of advection in Equation (6.7). 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 21 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**

$$u_j^{n+1} = -f_{u_j}{}^n (\zeta_{R(j)}^{n+1} - \zeta_{L(j)}^{n+1}) + r_{u_j}{}^n \tag{6.124}$$

$$q_j^{n+1} = A_{u_j}{}^n \left(\theta_j u_j^{n+1} + (1-\theta_j) u_j^n\right) \tag{6.125}$$

$$q_{a_j}{}^{n+1} = A_{u_j}{}^n u^{n+1} \tag{6.126}$$

else

$$u_j^{n+1} = 0 \tag{6.127}$$

$$q_j^{n+1} = 0 \tag{6.128}$$

$$q_{a_j}{}^{n+1} = 0 \tag{6.129}$$

end if

The time-step is summed up in Algorithm (22).

Algorithm 22 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 (14)
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 (15)
determine the set of water levels that need to be solved, Algorithm (17)
$p = 0$
$\zeta_k^{n+1(0)} = \zeta_k^n$
while $\left(\max_k \left|\zeta_k^{n+1(p)} - \zeta_k^{n+1(p-1)}\right| > \varepsilon \ \wedge$ not repeat time-step$\right) \ \vee \ p = 0$ **do**
$p = p + 1$
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 (18)
solve the unknown water levels and obtain $\zeta_k^{n+1(p+1)}$, Algorithm (23)
check positivity of water height with Algorithm (19) 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 $V_k^{n+1(p+1)}$ and wet bed area $A_k^{n+1(p+1)}$ with Algorithm (20)
end if
end while
end while
end while
$\zeta_k^{n+1} = \zeta_k^{n+1(p+1)}$
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 (21)

6.3.1 Solving the water level equation

The unknown water levels $k \in \mathcal{K}$ in Equation (6.120) are solved with a Krylov solver, Algorithm (23).

Algorithm 23 solve_matrix: solve the unknown water levels in Equation (6.120)

perform Gauss elimination to reduce the number of unknowns in the system
solve system with Algorithm (24)
perform the Gauss substitution and obtain $\zeta_k^{n+1(p+1)}, \quad k \in \mathcal{K}$
set $\zeta_k^{n+1(p+1)} = \zeta_k^{n+1(p)}, \quad k \notin \mathcal{K}$

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 (23) 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

$$A\boldsymbol{s} = \boldsymbol{d}, \tag{6.130}$$

where according to Equation (6.120)

$$A\boldsymbol{s} = \begin{pmatrix} B_{r1}\ \zeta_1 + \sum_{j\in\mathcal{J}(1)} C_{rj}\ \zeta_{O(1,j)} \\ B_{r2}\ \zeta_2 + \sum_{j\in\mathcal{J}(2)} C_{rj}\ \zeta_{O(2,j)} \\ \vdots \end{pmatrix} \tag{6.131}$$

and $\boldsymbol{d} = (d_{r1}, d_{r2}, \dots)^{\mathrm{T}}$. 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 (24). The preconditioner P can either be diagonal scaling, or an incomplete Cholesky decomposition.

Algorithm 24 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 $\varepsilon = \|\boldsymbol{r}^{(0)}\|_\infty$
apply preconditioner $P\boldsymbol{z}_r^{(0)} = \boldsymbol{r}^{(0)}$
set $\boldsymbol{p}^{(0)} = \boldsymbol{z}_r^{(0)}$
compute inner product $\left\langle \boldsymbol{r}^{(0)}, \boldsymbol{z}_r^{(0)} \right\rangle$
$i = 0$
while $\varepsilon >$ tol **do**
  compute $A\boldsymbol{p}^{(p)}$
  compute $\left\langle \boldsymbol{p}^{(p)}, A\boldsymbol{p}^{(p)} \right\rangle$
  $\alpha^{(p)} = \frac{\left\langle \boldsymbol{r}^{(p)}, \boldsymbol{z}_r^{(p)} \right\rangle}{\left\langle \boldsymbol{p}^{(p)}, A\boldsymbol{p}^{(p)} \right\rangle}$
  $\boldsymbol{s}^{(p+1)} = \boldsymbol{s}^{(p)} + \alpha^{(p)}\boldsymbol{p}^{(p)}$
  $\boldsymbol{r}^{(p+1)} = \boldsymbol{r}^{(p)} - \alpha^{(p)}A\boldsymbol{p}^{(p)}$
  compute maximum error $\varepsilon = \|\boldsymbol{r}^{(p+1)}\|_\infty$
  apply preconditioner $P\boldsymbol{z}_r^{(p+1)} = \boldsymbol{r}^{(p+1)}$
  if $\varepsilon >$ tol **then**
    compute $\left\langle \boldsymbol{r}^{(p+1)}, \boldsymbol{z}_r^{(p+1)} \right\rangle$
    $\beta^{(p+1)} = \frac{\left\langle \boldsymbol{r}^{(p+1)}, \boldsymbol{z}_r^{(p+1)} \right\rangle}{\left\langle \boldsymbol{r}^{(p)}, \boldsymbol{z}_r^{(p)} \right\rangle}$
    $\boldsymbol{p}^{(p+1)} = \boldsymbol{z}_r^{(p+1)} + \beta^{(p+1)}\boldsymbol{p}^{(p)}$
    $p = p + 1$
  end if
end while

Remark 6.3.2. The iterations in Algorithm (24) could be stopped after the computation of the absolute error. However, we want the possibility to base our stopping criterion on the preconditioned residual $\|\boldsymbol{z}_r^{(p)}\|_\infty$. For now, we will base our stopping criterion only on the residual $\boldsymbol{r}^{(p)}$.

6.4 Boundary Conditions

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

1 Boundary conditions that complement the governing equations, Equation (6.6) and Equation (6.7).
2 Supplementary boundary conditions that impose additional constraints at the boundaries.
3 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:

$$\boldsymbol{u}\cdot\boldsymbol{n} = 0, \qquad \boldsymbol{x}\in\Gamma_0, \qquad \text{default}, \tag{6.132}$$

$$\zeta = \zeta_b, \qquad \boldsymbol{x}\in\Gamma_1, \qquad \text{"waterlevel"}, \tag{6.133}$$

$$\frac{\partial\zeta}{\partial n} = s_b, \qquad \boldsymbol{x}\in\Gamma_2, \qquad \text{"Neumann"}, \tag{6.134}$$

$$\boldsymbol{u}\cdot\boldsymbol{n} = u_b, \qquad \boldsymbol{x}\in\Gamma_3, \qquad \text{"velocity"}, \tag{6.135}$$

$$\int_{\Gamma_4} h\boldsymbol{u}\cdot\boldsymbol{n}\,\mathrm{d}\Gamma = Q_b, \qquad \boldsymbol{x}\in\Gamma_4, \qquad \text{"discharge"}, \tag{6.136}$$

$$\zeta + \sqrt{\frac{h}{g}}\boldsymbol{u}\cdot\boldsymbol{n} = 2\zeta_b - \zeta^0, \qquad \boldsymbol{x}\in\Gamma_5, \qquad \text{"Riemann"}, \tag{6.137}$$

$$\frac{\partial\zeta}{\partial n} = 0, \boldsymbol{u}\cdot\boldsymbol{n} > 0, \qquad \boldsymbol{x}\in\Gamma_6, \qquad \text{"outflow"}, \tag{6.138}$$

$$\frac{\partial\zeta}{\partial t} - \sqrt{gh}\left(\frac{\partial\zeta}{\partial n} + s_b\right) = 0, \boldsymbol{u}\cdot\boldsymbol{n} \le 0, \qquad \boldsymbol{x}\in\Gamma_6, \qquad \text{"outflow"}, \tag{6.139}$$

$$\zeta = h_b\left(\int_{\Gamma_7} h\boldsymbol{u}\cdot\boldsymbol{n}\,\mathrm{d}\Gamma\right), \qquad \boldsymbol{x}\in\Gamma_7, \qquad \text{"Qh"}, \tag{6.140}$$

and

$$\boldsymbol{u}\cdot\boldsymbol{t} = v_b, \quad \boldsymbol{x}\in\Gamma_8, \quad \text{"tangentialvelocity"}, \tag{6.141}$$

$$\boldsymbol{u} = \boldsymbol{u}_b, \quad \boldsymbol{x}\in\Gamma_9, \quad \text{"ucxucyadvectionvelocity"}, \tag{6.142}$$

$$\boldsymbol{u}\cdot\boldsymbol{n} = u_b, \quad \boldsymbol{x}\in\Gamma_{10}, \quad \text{"normalvelocity"}, \tag{6.143}$$

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 6.4.1. The condition $\boldsymbol{u}\cdot\boldsymbol{n} > 0$ in Equation (6.138) is satisfied at *inflow* only.

Remark 6.4.2. At the "Qh" boundary ζ is a function h_b of Q and $Q-\zeta$ condition is imposed.

6.4.1 Virtual 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.

$$u_j = 0, \quad j\in\mathcal{B}_0. \tag{6.144}$$

The non-default boundary conditions are imposed by using virtual boundary cells, see Figure 6.11. 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.

$$\Gamma_\zeta = \Gamma_1\cup\Gamma_2\cup\Gamma_5\cup\Gamma_6\cup\Gamma_7, \tag{6.145}$$

$$\Gamma_u = \Gamma_3\cup\Gamma_4, \tag{6.146}$$

$$\Gamma = \Gamma_\zeta\cup\Gamma_u \tag{6.147}$$

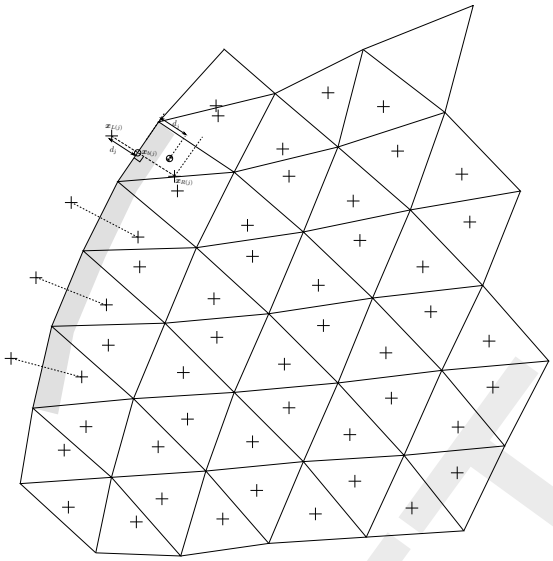


Figure 6.11: *Virtual boundary "cells" near the shaded boundary; $\boldsymbol{x}_{Lj}$ is the virtual "cell" center near boundary face j; $\boldsymbol{x}_{R(j)}$ is the inner-cell center; $\boldsymbol{b}_j$ is the point on face j that is nearest to the inner-cell center*

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 virtual 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 Figure 6.11, and let $\boldsymbol{b}_j$ be the corresponding nearest point on the boundary face. Then the virtual "cell" centers are computed with Algorithm (25). Note that $\boldsymbol{x}_n$ are mesh node coordinates and remember that the face normal $\boldsymbol{n}_j$ is inward positive.

Algorithm 25 addexternalboundarypoints: compute centers of virtual boundary "cells"

$$\boldsymbol{x}_{L(j)} = \begin{cases} \boldsymbol{b}_j - \max(d_j, \frac{1}{2}\sqrt{b_{AR(j)}})\boldsymbol{n}_j, & j \in \mathcal{B}_\zeta \wedge \text{izbndpos} = 0 \\ \frac{1}{2}(\boldsymbol{x}_{nl(j)} + \boldsymbol{x}_{nr(j)}), & j \in \mathcal{B}_\zeta \wedge \text{izbndpos} = 1, \\ \boldsymbol{b}_j - \max(d_j, \frac{1}{2}\sqrt{b_{AR(j)}})\boldsymbol{n}_j, & j \in \mathcal{B}_u. \end{cases} \tag{6.148}$$

Remark 6.4.3. Option `izbndpos` $= 2$ is not documented here.

Algorithm (25) shows that the virtual 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 virtual boundary "cells" also have a bed area b_A and bed level bl defined as

$$\left.\begin{array}{rcl} b_{AL(j)} & = & b_{AL(j)}, \\ bl_{L(j)} & = & bl_{R(j)}. \end{array}\right\} \quad j \in \mathcal{B}. \tag{6.149}$$

6.4.2 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. ((6.132)) to ((6.140)) is:

$$u_j^{n+1} = 0, \qquad j \in \mathcal{B}_0, \text{ default}, \tag{6.150}$$

$$\zeta_{L(j)}^{n+1} = \zeta_b(\boldsymbol{b}_j, \hat{t}^{n+1}), \qquad j \in \mathcal{B}_1, \text{ "waterlevel"}, \tag{6.151}$$

$$\frac{\zeta_{R(j)}^{n+1} - \zeta_{L(j)}^{n+1}}{\Delta x_j} = s_b(\boldsymbol{b}_j, \hat{t}^{n+1}), \qquad j \in \mathcal{B}_2, \text{ "Neumann"}, \tag{6.152}$$

$$u_j^{n+1} = u_b(\boldsymbol{b}_j, \hat{t}^{n+1}), \qquad j \in \mathcal{B}_3, \text{ "velocity"}, \tag{6.153}$$

$$u_j^{n+1} = \frac{Q_b(\hat{t}^{n+1})({h_{u_j}}^n)^{2/3}}{\sum\limits_{l \in \mathcal{B}_4} {A_{u_l}}^n ({h_{u_l}}^n)^{2/3}}, \qquad j \in \mathcal{B}_4 \quad \text{"discharge"}, \tag{6.154}$$

$$\begin{aligned} \zeta_{L(j)}^{n+1} &= 2\zeta_b(\boldsymbol{b}_j, \hat{t}^{n+1}) - \zeta_{R(j)}^0 + \dots \\ &\dots - \sqrt{\frac{\frac{1}{2}(h_{\zeta_{L(j)}}^n + h_{\zeta_{R(j)}}^n)}{g}} u_j^n, \quad j \in \mathcal{B}_5, \text{ "Riemann"}, \end{aligned} \tag{6.155}$$

$$\zeta_{L(j)}^{n+1} = \zeta_{R(j)}^n, u_j^n > 0 \qquad j \in \mathcal{B}_6, \text{ "outflow"}, \tag{6.156}$$

$$\begin{aligned} \frac{\zeta_{L(j)}^{n+1} - \zeta_{L(j)}^n}{\Delta t^n} &= \sqrt{g\frac{1}{2}(h_{\zeta_{L(j)}}^n + h_{\zeta_{R(j)}}^n)} \dots \\ \dots (\frac{\zeta_{R(j)}^n - \zeta_{L(j)}^n}{\Delta x_j} &+ s_b(\boldsymbol{b}_j, \hat{t}^{n+1})), u_j \le 0, \qquad j \in \mathcal{B}_6, \text{ "outflow"}, \end{aligned} \tag{6.157}$$

$$\zeta_{L(j)}^{n+1} = h_b(\sum_{l \in \mathcal{B}_7} q_l^n), \qquad j \in \mathcal{B}_7, \text{ "Qh"}, \tag{6.158}$$

where h_{ζ_k} is the cell-centered water depth, i.e.

$$h_{\zeta_k} = \zeta_k - bl_k, \tag{6.159}$$

$\zeta_b(\boldsymbol{x}, t)$ 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 (21)

$$q_j^n = {A_{u_j}}^{n-1} \left(\theta_j u_j^n + (1-\theta_j) u_j^{n-1}\right). \tag{6.160}$$

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 $\frac{1}{2}(h_{\zeta_{L(j)}}^n + h_{\zeta_{R(j)}}^n)$ 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:

$$\boldsymbol{u}_{\boldsymbol{c}L(j)}^n = u_j^n \boldsymbol{n}_j + v_b(\boldsymbol{b}_j, \hat{t}^{n+1})\boldsymbol{t}_j, \quad j \in \mathcal{B}_8, \text{ "tangentialvelocity"}, \tag{6.161}$$

$$\boldsymbol{u}_{\boldsymbol{c}L(j)}^n = \boldsymbol{u}_b(\boldsymbol{b}_j, \hat{t}^{n+1}), \qquad j \in \mathcal{B}_9, \text{ "ucxucyadvectionvelocity"}, \tag{6.162}$$

$$\boldsymbol{u}_{\boldsymbol{c}L(j)}^n = u_b(\boldsymbol{b}_j, \hat{t}^{n+1})\boldsymbol{n}_j, \qquad j \in \mathcal{B}_{10}, \text{"normalvelocity"}, \tag{6.163}$$

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 6.4.4. 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.

$$h_{uj} = \zeta_{R(j)} - bl_{R(j)}, \quad j \in \mathcal{B}_4 \quad \wedge \quad \text{jbasqbnddownwindhs} = 1, \quad \text{"discharge"}. \tag{6.164}$$

Compare with Algorithm (2) where the face-based water depths h_{uj} are computed. They are overwritten in Algorithm (26) at the discharge boundary.

Algorithm 26 setau | discharge boundaries: adjustment to Algorithm (3) to overwrite water depths at the discharge boundaries

if jbasqbnddownwindhs=0 **then**
$\mathcal{B}^* = \{l \in \mathcal{B}_4 | h_{ul} > 0\}$
$h_{uj} = \max(0, \frac{\sum_{l \in \mathcal{B}^*} \zeta_{R(l)} w_{ul}}{\sum_{l \in \mathcal{B}^*} w_{ul}}) - \min(bl_{1j}, bl_{2j}), \quad j \in \mathcal{B}^*$
end if
if jbasqbnddownwindhs=1 **then**
$h_{uj} = \zeta_{R(j)} - bl_{R(j)}, \quad j \in \mathcal{B}_4$
compute A_{uj}, $j \in \mathcal{B}_5$ as in Algorithm (3)
end if
$\hat{\mathcal{B}} = \{l \in \mathcal{B}_4 | h_{ul} \geq$ qbndhutrs $\vee\, Q_b \geq 0\}$
$h_{uj} = 0, \quad j \in \mathcal{B}_4 \setminus \hat{\mathcal{B}}$
$A_{uj} = 0, \quad j \in \mathcal{B}_4 \setminus \hat{\mathcal{B}}$
$zu_j = \frac{Q_b (h_{uj})^{2/3}}{\sum_{l \in \hat{\mathcal{B}}} (h_{ul})^{2/3} A_{ul}}$, $j \in \mathcal{B}_4$

Remark 6.4.5. 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

$$\boldsymbol{u} \cdot \boldsymbol{n} + 2\sqrt{gh} = u_b + 2\sqrt{gh_b} \tag{6.165}$$

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

$$\sqrt{gh_b} = \sqrt{g(h + \zeta_b - \zeta)} \approx \sqrt{gh} + \frac{1}{2}\sqrt{\frac{g}{h}}(\zeta_b - \zeta). \tag{6.166}$$

Substitution yields

$$\sqrt{\frac{g}{h}}\zeta + \boldsymbol{u} \cdot \boldsymbol{n} = u_b + \sqrt{\frac{g}{h}}\zeta_b. \tag{6.167}$$

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

Note: Assumed is that there is no residual flow in the model.

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

$$u_b h = \sqrt{gh}(\zeta_b - \zeta^0), \tag{6.168}$$

or

$$u_b = \sqrt{\frac{g}{h}}(\zeta_b - \zeta^0). \tag{6.169}$$

Substitution of this expression in Equation (6.167) yields

$$\sqrt{\frac{g}{h}}\zeta + \boldsymbol{u} \cdot \boldsymbol{n} = \sqrt{\frac{g}{h}}(2\zeta_b - \zeta^0). \tag{6.170}$$

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

The discretization in D-Flow FM is then as shown in Equation (6.155):

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

Qh boundaries: qhrelax

Again, for simplicity we will only consider one Qh boundary. The applied water level ζ_{b_j} is relaxed with a user-specified parameter α_{Qh} (`qhrelax`) that turns Equation (6.158) into

$$\zeta_j^{n+1} = \alpha_{Qh}\ h_b(\sum_{l \in \mathcal{B}_7} q_l^n) + (1 - \alpha_{Qh})\ \zeta_j^n, \quad j \in \mathcal{B}_7$$

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

6.4.3 Imposing the discrete boundary conditions: jacstbnd

During a time-step from t^n to t^{n+1}, the discrete boundary conditions, i.e. Equation (6.151) to Equation (6.158) and Equation (6.161) to Equation (6.163), are imposed in the following manner:

- ◇ the reconstruction of the full velocity vectors $\boldsymbol{u}_{c_j}^n$ is modified with Algorithm (27) 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 (28),
- ◇ the velocity boundary conditions are imposed on $u_j^{n+1}, j \in \mathcal{B}_u$ in Algorithm (30),
- ◇ the system of equations, referred to as the "water level equations", to obtain ζ^{n+1} is adjusted in Algorithm (29) near the boundaries,
- ◇ having computed the water levels ζ^{n+1} from the "water level equation" with Algorithm (23), the water levels in the virtual boundary "cells" ζ_{Lj}^{n+1}, $j \in \mathcal{B}_\zeta$ are computed with Algorithm (28),
- ◇ the velocities at the new time level u_j^{n+1} are computed with Algorithm (21) which requires no modifications since f_u and r_u were properly adjusted in Algorithm (30).

Remark 6.4.7. 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 6.4.8. 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 6.4.9. 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 (28), since the virtual boundary "cells" are included in the solution vector $\boldsymbol{s} = (\zeta_1, \zeta_2, \dots)^{\mathrm{T}}$ and the discrete system of Equation (6.120) is augmented with the discrete boundary conditions of Equation (6.133) to Equation (6.140) in Algorithm (29).

Algorithm 27 setucxucyucxuucyu | boundary conditions: adjustment to Algorithm (6) to satisfy the boundary conditions

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

Remark 6.4.10. The discretization of the "outflowboundary" condition, Γ_6, in Algorithm (28) is different from the one in Algorithm (29) and seems incomplete. The condition for $u_j \leq 0$ is missing.

Remark 6.4.11. The "Qh" boundary condition is ineffective in Algorithm (28), since it is missing from Equation (6.171) and Equation (6.172).

The water level boundary conditions are inserted into the system of equations as follows. Firstly, Equation (6.151) to Equation (6.158) show that for some z_b the boundary conditions can be put as

$$\zeta_{L(j)}^{n+1} = z_b, \qquad j \in \mathcal{B}_\zeta \setminus \mathcal{B}_2, \tag{6.173}$$

$$\frac{\zeta_{R(j)}^{n+1} - \zeta_{L(j)}^{n+1}}{\Delta x_j} = s_b(\boldsymbol{b}, \hat{t}^{n+1}), \quad j \in \mathcal{B}_2. \tag{6.174}$$

Algorithm 28 sets01zbnd: apply boundary conditions to water levels ζ^n or ζ^{n+1}

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

$z_b = \max(zb - \frac{p_{atm_{L(j)}} - p_{av}}{\rho_{mean}\, g}, bl_{L(j)} + \delta), \quad j \in \mathcal{B}_\zeta \setminus \mathcal{B}_6$

$\delta = 10^{-3}$

if apply to ζ^n **then**

$$\zeta^n_{L(j)} = \begin{cases} z_b, & j \in \mathcal{B}_1 \cup \mathcal{B}_2 \cup \mathcal{B}_5 \\ \max(\zeta^n_{R(j)}, bl_{R(j)}), \quad u_j^n > 0, & j \in \mathcal{B}_6 \end{cases} \tag{6.171}$$

else {apply to ζ^{n+1}}

$$\zeta^{n+1}_{L(j)} = \begin{cases} z_b, & j \in \mathcal{B}_1 \cup \mathcal{B}_2 \cup \mathcal{B}_5 \\ \max(\zeta^n_{R(j)}, bl_{R(j)}), \quad u_j^n > 0, & j \in \mathcal{B}_6 \end{cases} \tag{6.172}$$

end if

The rows in the system that are affected by these boundary conditions are the rows that correspond to the virtual 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 Equation (6.120) respectively, and using $O(L(j), j) = R(j)$ and $O(R(j), j) = L(j)$, i.e.

$$\left.\begin{array}{llll} B_{r\,L(j)}^{\;n+1(p)} & \zeta^{n+1(p+1)}_{L(j)} & & +C_{rj}^{\;n}\;\zeta^{n+1(p+1)}_{R(j)} = d_{r\,L(j)}^{\;n+1(p)}, \\ B_{r\,R(j)}^{\;n+1(p)} & \zeta^{n+1(p+1)}_{R(j)} + \sum_{l\in\mathcal{J}(R(l))\setminus j} C_{rl}^{\;n}\;\zeta^{n+1(p+1)}_{O(R(j),l)} & & +C_{rj}^{\;n}\;\zeta^{n+1(p+1)}_{L(j)} = d_{r\,R(j)}^{\;n+1(p)}, \end{array}\right\} j \in \mathcal{B}_\zeta.$$

Combining these expressions yields for the non-Neumann boundary conditions

$$\left.\begin{array}{llll} & \zeta^{n+1(p+1)}_{L(j)} & & = z_b, \\ B_{r\,R(j)}^{\;n+1(p)} & \zeta^{n+1(p+1)}_{R(j)} + \sum_{l\in\mathcal{J}(R(l))\setminus j} C_{rl}^{\;n}\;\zeta^{n+1(p+1)}_{O(R(j),l)} & & = d_{r\,R(j)}^{\;n+1(p)} - C_{rj}^{\;n}\;z_b, \end{array}\right\} j \in \mathcal{B}_\zeta \setminus \mathcal{B}_2$$

and for the Neumann boundary condition

$$\left.\begin{array}{llll} -C_{rj}^{\;n}\;\zeta^{n+1(p+1)}_{L(j)} & & & +C_{rj}^{\;n}\;\zeta^{n+1(p+1)}_{R(j)} = \\ & & & -C_{rj}^{\;n}\Delta x_j s_b(\boldsymbol{b}_j, \hat{t}^{n+1}), \\ B_{r\,R(j)}^{\;n+1(p)}\;\zeta^{n+1(p+1)}_{R(j)} & + \sum_{l\in\mathcal{J}(R(l))\setminus j} C_{rl}^{\;n}\;\zeta^{n+1(p+1)}_{O(R(j),l)} & & +C_{rj}^{\;n}\;\zeta^{n+1(p+1)}_{L(j)} =, \\ & & & d_{r\,R(j)}^{\;n+1(p)}, \end{array}\right\} j \in \mathcal{B}_2.$$

The consequences for the matrix elements are shown in Algorithm (29).

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

$$u_j^{n+1} = zu_j, \quad j \in \mathcal{B}_u = \mathcal{B}_3 \cup \mathcal{B}_4, \tag{6.175}$$

Algorithm 29 s1nod | boundary conditions: adjustments to Algorithm (18) to satisfy the boundary conditions in the water level equation

$B_{r\,k}^{\,n+1(p)}\ \zeta_k^{n+1(p+1)} + \sum_{j\in\mathcal{J}(k)} C_{r\,j}^{\,n}\ \zeta_{O(k,j)}^{n+1(p+1)} = d_{r\,k}^{\,n+1(p)}$, Equation (6.120)

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

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

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

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

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

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

where according to Equation (6.153)

$$zu_j = u_b(\boldsymbol{b}_j, \hat{t}^{n+1}), \quad j \in \mathcal{B}_3 \tag{6.176}$$

and zu_j is computed with Algorithm (26) for the discharge boundaries $j \in \mathcal{B}_4$. Since the velocity at the next time level with Equation (6.111) in Algorithm (14)

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

the adjustments to Algorithm (14) are obvious and presented in Algorithm (30). 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 30 furu | boundary conditions: adjustments to Algorithm (14) 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 (15), i.e.

$$C_{r_j}^{\ n} = 0, \quad j \in \mathcal{B}_u. \tag{6.177}$$

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

$$\zeta_{R(j)}^{n+1} - \zeta_{L(j)}^{n+1} = 0 \in \mathcal{B}_u. \tag{6.178}$$

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

$$\left.\begin{array}{lll} -C_{r_j}^{\ n}\ \zeta_{L(j)}^{n+1(p+1)} & & +C_{r_j}^{\ n}\ \zeta_{R(j)}^{n+1(p+1)} = \\ & & 0, \\ (B_{r_{R(j)}}^{\ n+1(p)} - C_{r_j}^{\ n})\ \zeta_{R(j)}^{n+1(p+1)} & + \sum_{l \in \mathcal{J}(R(l)) \setminus j} C_{r_l}^{\ n}\ \zeta_{O(R(j),l)}^{n+1(p+1)} & +C_{r_j}^{\ n}\ \zeta_{L(j)}^{n+1(p+1)} =, \\ & & d_{r_{R(j)}}^{\ n+1(p)}, \end{array}\right\} j \in \mathcal{B}_u.$$

To obtain a sensible order of magnitude, we set, as shown in Algorithm (29),

$$C_{r_j}^{\ n} = -B_{r_{R(j)}}^{\ n+1(p)}, \quad j \in \mathcal{B}_u. \tag{6.179}$$

6.4.4 Relaxation of the boundary conditions: Tlfsmo

In Algorithms (28), (30) and (29) the boundary conditions are relaxed from the initial values with a parameter α_{smo} that turns the discrete boundary conditions into

$$\zeta_{L(j)}^{n+1} = (1 - \alpha_{smo})\zeta_{L(j)}^{0} + \alpha_{smo}\, \zeta_b(\boldsymbol{b}_j, \hat{t}^{n+1}), \quad j \in \mathcal{B}_1, \text{"waterlevel"}, \tag{6.180}$$

$$u_j^{n+1} = \alpha_{smo}\, u_b(\boldsymbol{b}_j, \hat{t}^{n+1}), \quad j \in \mathcal{B}_3, \text{"velocity"}, \tag{6.181}$$

$$u_j^{n+1} = \alpha_{smo}\, \frac{Q_b(\hat{t}^{n+1})(h_{u_j}^{\ n})^{2/3}}{\sum_{l \in \mathcal{B}_4} A_{u_l}^{\ n}(h_{u_l}^{\ n})^{2/3}}, \quad j \in \mathcal{B}_4 \text{ "discharge"}, \tag{6.182}$$

$$\zeta_{L(j)}^{n+1} = (1 - \alpha_{smo})\zeta_{L(j)}^{0} + \alpha_{smo}\, h_b(\sum_{l \in \mathcal{B}_7} q_l^n), \quad j \in \mathcal{B}_7, \text{"Qh"} \tag{6.183}$$

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

$$\alpha_{smo} = \min\left(\frac{\hat{t}^{n+1} - t^0}{T_{smo}}, 1\right). \tag{6.184}$$

Remark 6.4.12. The second category velocity boundary conditions are *not* being relaxed in the same way as the first category, but maybe they should.

Remark 6.4.13. A zero initial velocity field is assumed in the relaxation of the boundary conditions.

6.4.5 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

$$-\frac{p_{atm_{L(j)}} - p_{av}}{\rho_{mean}\, g} \tag{6.185}$$

as shown in Algorithms (28) and (29), 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.

6.4.6 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 (31). 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 31 flow_initexternalforcings: adjust numerical settings near the boundaries

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

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

6.4.7 Viscous fluxes: irov

Momentum diffusion is elaborated in Theorem 6.2.2 and in particular Algorithm (11). It will be clear that we can not evaluate the viscous stresses $\boldsymbol{t}_{u_j}$ at *closed boundaries* as in Equation (6.74). Instead, boundary conditions need to be imposed. These are:

$$\boldsymbol{t}_{u_j} = \begin{cases} \boldsymbol{0}, & \text{irov=0, free slip,} \\ -u_j^*|u_j^*|\boldsymbol{s}_j, & \text{irov=1, partial slip,} \\ -\nu_j \frac{U_j}{\Delta y_j}\boldsymbol{s}_j, & \text{irov=2, no slip,} \end{cases} \tag{6.186}$$

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),

$$\Delta y_j = \frac{1}{2}\frac{b_{AR(j)}}{w_{uj}}. \tag{6.187}$$

The friction velocity is computed as

$$u_j^* = \frac{U_j \kappa}{C + d/z_0}, \tag{6.188}$$

with

$$U_j = \boldsymbol{u}_{cR(j)} \cdot \boldsymbol{s}_j, \tag{6.189}$$

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

$$\begin{aligned}
&\frac{1}{h^p}\,\nabla\cdot\left(\nu h^p(\nabla\boldsymbol{u}+\nabla\boldsymbol{u}^{\mathrm{T}})\right)\big|_{\Omega_{R(j)}} \\
&\approx \frac{1}{H^p_{R(j)}}\boldsymbol{d}_{R(j)} + \frac{\sum_{l\in\{m\in\mathcal{B}_0|R(m)=R(j)\}} \boldsymbol{t}_{ul} w_{ul} s_{l,R(j)}}{b_{AR(j)}}, \quad j\in\mathcal{J}_0,
\end{aligned} \tag{6.190}$$

where $\boldsymbol{d}_{R(j)}$ represents the contribution from the non-boundary faces of boundary cell $R(j)$ as given by Equation (6.73) for $k = R(j)$.

Remark 6.4.14. Comparing the contribution of the viscous boundary stress with the expression for the contribution of the internal faces $\boldsymbol{d}_k$ in Equation (6.73) reveals that the `istresstype` does not apply to the contribution of the boundary stresses. Apparently $p = 0$ is applied here. See Remark 6.2.17 in this respect.

6.5 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 (32), (33) and (34). The data being computed and updated are shown in Table 6.3.

Algorithm 32 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 $h_{uj}{}^n$, $A_{uj}{}^n$, et cetera and perform the predictor phase of the fractional time step to obtain $\mathcal{A}_{\mathsf{i}j}$ and $\mathcal{A}_{\mathsf{e}j}$, $\forall j$ with Algorithm (33)

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 $h_{uj}{}^n$ 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 (22)

Table 6.3: *Data during a computational time step from t^n to t^{n+1} with Algorithm (32); the translation to D-Flow FM nomenclature is shown in the last column*

input		D-Flow FM
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 33 flow_initimestep: compute derived data and perform the predictor phase of the fractional time step

$h_{\zeta_k}^{\ n} = \zeta_k^n - bl_k, \quad \forall k$
$\hat{t}^{n+1} = t^n + \Delta t^n$
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 (41) explained later
setau: compute the flow area $A_{u_j}^{\ n}$ with Algorithms (3) and (26)
setumod: compute cell-based full velocity vectors $\boldsymbol{u}_{c_k}^{\ n}$, $\boldsymbol{u}_{q_k}^{\ n}$, first-order upwind velocity $\boldsymbol{u}_{u\,j}^{L\,n}$ and add Coriolis forces and viscous fluxes to $\mathcal{A}_{\mathrm{e}\,j}$ with Algorithm (34)
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 (42)
setextforcechkadvec: add external forces to $\mathcal{A}_{\mathrm{i}\,j}$ and $\mathcal{A}_{\mathrm{e}\,j}$ and make adjustments for small water depths, Algorithm (37) explained later

Algorithm 34 setumod: compute cell-based full velocity vectors $\boldsymbol{u}_{c_k}$, $\boldsymbol{u}_{q_k}$, first-order upwind velocity $\boldsymbol{u}_{u\,j}^{L}$ and add Coriolis forces and viscous fluxes to $\mathcal{A}_{\mathrm{e}\,j}$

setucxucyucxuucyu: reconstruct cell centered velocity vectors $\boldsymbol{u}_{c_k}$ and $\boldsymbol{u}_{q_k}$, and set first-order upwind fluxes $\boldsymbol{u}_{u\,j}^{L}$ with Algorithms (6) and (27)
compute tangential velocities v_j from the cell-centered velocities $\boldsymbol{u}_{q_k}$ with Eqn. ((6.50))
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 $\mathcal{A}_{\mathrm{e}\,j}$ with Algorithm (11)
compute viscous momentum fluxes near the default boundaries $j \in \mathcal{J}_0$ and add to $\mathcal{A}_{\mathrm{e}\,j}$

6.6 Flooding and drying

The governing equations, Equation (6.6) and Equation (6.7), 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

$$\bar{\Omega}(t) = \{\boldsymbol{x} \in \Omega | h(\boldsymbol{x}, t) > 0\}. \tag{6.191}$$

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, Equation (6.6) and Equation (6.7), 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.

6.6.1 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. $h_{u_j} = 0$. 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_k^n, faces for which $h_u \leq \varepsilon_{hu}$ are disabled by setting them to zero with Algorithm (35). 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 $\zeta_k^{n+1(p)}$ 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 ${h_u}_j^n$ to zero, see Algorithm (19), and the water level equation is solved again.

Algorithm 35 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 ${h_u}_j^n \leq \varepsilon_{hu}$

6.6.2 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 (34).

The computation of the wet bed areas and water-columns was already presented in Algorithm (20). As can been seen in Algorithm (20), 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 6.2.4 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

$$V_k = \int_{\Omega_k \cap \bar{\Omega}} h \, \mathrm{d}\Omega \tag{6.192}$$

and

$$A_k = \int_{\Omega_k \cap \bar{\Omega}} \mathrm{d}\Omega \tag{6.193}$$

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 (36).

Remark 6.6.1. Applying Gauss's theorem to Equation (6.193) yields

$$A_k = \int_{\partial\Omega_k \cap \bar{\Omega}} \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_k) \cdot \boldsymbol{n} \, \mathrm{d}l + \int_{\Omega_k \cap \partial\bar{\Omega}} \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_k) \cdot \boldsymbol{n} \, \mathrm{d}l, \tag{6.194}$$

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.

Algorithm 36 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 (20)
else
  compute $\Delta b_j = \max(bl_{1j}, bl_{2j}) - \min(bl_{1j}, bl_{2j})$ and wet cross-sectional area A_{uj} as in Algorithm (3)

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

end if

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

Recall that Eqn. ((6.26)) summarizes the spatial discretization of the momentum equation:

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

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 (34) 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 (37). 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 37 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 $h_{sL(j)} < \frac{1}{2}h_{\zeta R(j)} \ \wedge \ \mathcal{A}_{\mathrm{e}j} < 0 \ \wedge \ h_{\zeta R(j)} < h_{\mathrm{chkadv}}$ **then**
    $\mathcal{A}_{\mathrm{e}j} = \min(\frac{h_{\zeta L(j)}}{h_{\mathrm{chkadv}}}, 1)\mathcal{A}_{\mathrm{e}j}$
  else if $h_{\zeta R(j)} < \frac{1}{2}h_{\zeta L(j)} \ \wedge \ \mathcal{A}_{\mathrm{e}j} > 0 \ \wedge \ h_{\zeta L(j)} < h_{\mathrm{chkadv}}$ **then**
    $\mathcal{A}_{\mathrm{e}j} = \min(\frac{h_{\zeta R(j)}}{h_{\mathrm{chkadv}}}, 1)\mathcal{A}_{\mathrm{e}j}$
  end if
end if

Remark 6.6.2. It is unclear why the term $\mathcal{A}_{\mathrm{e}j}$ needs to be limited to zero for vanishing water depths.

Remark 6.6.3. 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. ((6.26)).

Coriolis forces are computed and added to $\mathcal{A}_{\mathrm{e}j}$ in Algorithm (34). 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 (38) with a threshold $h_{\mathrm{trshcorio}}$ called | trshcorio | in D-Flow FM.

Algorithm 38 setumod | limitation of Coriolis forces: adjustment to Algorithm (34) to account for vanishing water depths

$h_{\mathrm{trshcorio}} = 1$
$h_{minj} = \min(h_{\zeta L(j)}, h_{\zeta R(j)})$
limit Coriolis forces f_{cj} to $f_{cj}\min(\frac{h_{min}}{h_{\mathrm{trshcorio}}}, 1)$

Remark 6.6.4. The Coriolis forces are limited by Algorithm (38) *and* by Algorithm (37).

6.6.4 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 6.6.5. 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 6.6.6. 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 $\mathcal{A}_\mathsf{e}$ and $\mathcal{A}_\mathsf{e}$ 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 (22), which do not appear in the discretization of the momentum equation.

Recall that the temporal discretization of the momentum equation is expressed by Eqn. ((6.111)) 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

$$u_j^{n+1} = \begin{cases} 0, & \text{face is dry at beginning of the time step,} \\ 0, & \text{face is wet and becomes dry during the time step,} \\ -{f_{u_j}}^n(\zeta_{R(j)}^{n+1} - \zeta_{L(j)}^{n+1}) + {r_{u_j}}^n, & \text{face remains wet.} \end{cases} \tag{6.195}$$

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 (35), then its normal velocity component is left unmodified, since the velocities are only set at the end of the time step by Algorithm (21).

Remark 6.6.7. 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.

6.7 Fixed Weirs

This section elaborates on the numerical treatment of the fixed weirs. They are commonly used to model sudden changes in depth (roads, summer dikes) and groynes in numerical simulations of rivers. In D-Flow FM, a fixed weir is a fixed non-movable construction generating energy losses due to constriction of the flow. 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_{uj} 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. The flow over a fixed weir can not be modeled in D-Flow FM as is, but is based on an alternative approach which is called 'subgrid' modelling, which means that a weir is not 'modelled on the grid', but that a parametrization is applied.

Two different subgrid approaches are available to simulate the energy losses by fixed weirs. First of all, a numerical approach has been implemented. Then, a special discretization of

the advective terms before and after the fixed weir is applied. This option is switched on via keyword `fixedweirtype=6`. This numerical approach is described in detail in section 6.7.1 to section 6.7.4. Next to the numerical approach, there is an empirical approach to determine the energy losses by weirs, for which two options are availabe in D-Flow FM, namely the so-called 'Tabellenboek' and 'Villemonte' approaches. The Tabellenboek option is switched on via keyword `fixedweirtype=8`, while the Villemonte approach coincides with keyword `fixedweirtype=9`. The two corresponding empirical formulas have been taken from the Simona software, see the website `http://www.helpdeskwater.nl/onderwerpen/applicaties-model/applicaties-per/watermanagement/watermanagemer simona`. Based on many flume measurements formulas have been derived to fit the measurements as well as possible. This empirical approach is described in section 6.7.5.

6.7.1 Adjustments to the geometry: oblique weirs and FixedWeirContraction

Recall that the bed geometry is represented by the face-based bed-levels bl_1 and bl_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 bl_1 and bl_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.

$$w_{uj} = c \, \|\boldsymbol{x}_{r(j)} - \boldsymbol{x}_{l(j)}\| \, |\boldsymbol{n}_j \cdot \boldsymbol{n}_{wj}|, \tag{6.196}$$

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 6.7.1. Oblique weirs are not fully understood at this moment. We do not attempt to explain Equation (6.196) further.

In Equation (6.196) 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 bl_1, bl_2 and w_u are performed with Algorithm (39).

Algorithm 39 setfixedweirs: change geometry bl_1, bl_2 and w_u and advection type for fixed weirs

$bl_{1j} = \max(z_{cj}, bl_{1j})$
$bl_{2j} = \max(z_{cj}, bl_{2j})$
if left and right weir sill heights are prescribed and conveyance type > 0 **then**
  set bl_{1j} and bl_{2j} of adjacent faces, not described further
end if
adjust advection type of adjacent faces with Algorithm (40)
$w_{uj} = c \, \|\boldsymbol{x}_{r(j)} - \boldsymbol{x}_{l(j)}\| \, |\boldsymbol{n}_j \cdot \boldsymbol{n}_{wj}|$

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

Due to our subgrid modelling,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 (39). 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 (40) 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 6.7.2. Since the flow over a weir is discontinuous due to our subgrid modelling, one may need to discretize the spatial operators near the weir more rigorously.

Algorithm 40 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

6.7.3 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:

1 Energy is conserved from cell $L(j)$ to $R(j)$.
2 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.

$$\frac{1}{2}{u_{in}}_j^2 + g\zeta_{L(j)} = \frac{1}{2}u_j^2 + g(z_{Cj} + h_{uj}), \tag{6.197}$$

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_{u_j} a downwind approximation is used that obeys our second requirement:

$$h_{uj} = \max(\zeta_{R(j)} - z_{Cj}, \frac{2}{3}E_j), \tag{6.198}$$

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

$$E_j = \zeta_{L(j)} - z_{Cj} + \frac{1}{2g}{u_{in}}_j^2. \tag{6.199}$$

Remark 6.7.3. Equation (6.199) 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 Equation (6.201) in Equation (6.197), some rearrangement of terms and division by Δx_j yields

$$\frac{1}{2\Delta x_j}\left(u_j^2 - {u_{in}}_j^2\right) = -\frac{g}{\Delta x}\left(\zeta_{R(j)} - \zeta_{L(j)}\right) - \frac{g}{\Delta x}\max(0, \frac{2}{3}E_j - (\zeta_{R(j)} - z_{Cj})) \quad (6.200)$$

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 Equation (6.200)

$$\frac{\mathrm{d}u_j}{\mathrm{d}t} + \frac{1}{2\Delta x_j}\left(u_j^2 - {u_{in}}_j^2\right) = -\frac{g}{\Delta x}\left(\zeta_{R(j)} - \zeta_{L(j)}\right) - \frac{g}{\Delta x}\max(0, \frac{2}{3}E_j - (\zeta_{R(j)} - z_{Cj})) \quad (6.201)$$

Remark 6.7.4. Although momentum diffusion over the wear is missing in Equation (6.201), it is actually included in D-Flow FM.

Recall that the spatial discretization is summarized as shown in Equation (6.26):

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

For the temporal discretization, see Equation (6.110). The terms for fixed weirs can then be put as, assuming $u_j > 0$:

$$\mathcal{A}_{\mathrm{i}j} = \frac{1}{2\Delta x_j}u_j, \quad (6.202)$$

$$\mathcal{A}_{\mathrm{e}j} = -\frac{1}{2\Delta x_j}{u_{in}}_j^2 + \frac{g}{\Delta x}\max(0, \frac{2}{3}E_j - (\zeta_{R(j)} - z_{Cj})). \quad (6.203)$$

Remark 6.7.5. Although bed friction is not included in Equation (6.200), it is included in D-Flow FM by means of Equation (6.26). Its actual computation will not be discussed at this occasion.

The terms of Equation (6.198), and Equation (6.202) and Equation (6.203) are prescribed *partly* with Algorithm (41) and *partly* with Algorithm (42). Note that the latter also adjusts the cell-centered velocity vectors $\boldsymbol{u}_c$.

In Algorithms (41) and (42) 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 6.7.6. 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.

Remark 6.7.7. 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 6.7.2 in this respect. This is achieved by estimating an unperturbed

Algorithm 41 sethu | fixed weir: adjustment to Algorithm (2); set h_u and part (one of two) to $\mathcal{A}_\mathsf{e}$

if $u_j > 0$ **then**
compute far-field velocity $\hat{\boldsymbol{u}}_{cL(j)}$ with Algorithm (43)

$$u_{inj} = \begin{cases} \hat{\boldsymbol{u}}_{cL(j)} \cdot \boldsymbol{n}_{wj}, & \text{fixed weir scheme } 4 \\ 0 & \text{fixed weir scheme } 5 \\ \hat{\boldsymbol{u}}_{cL(j)} \cdot \boldsymbol{n}_j, & \text{otherwise} \end{cases}$$

$E_j = \zeta_{L(j)} - z_{cj} + \frac{1}{2g} u_{in_j}^2$
if $\zeta_{R(j)} < \zeta_{L(j)}$ **then**
$h_{uj} = \max(\zeta_{R(j)} - z_{cj}, \frac{2}{3} E_j)$
$\mathcal{A}_{\mathsf{e}j} = -\frac{g}{\Delta x_j} \min(0, \zeta_{R(j)} - z_{cj} - \frac{2}{3} E_j)$
end if
else
as above by interchanging $L(j)$ and $R(j)$ and taking reversed orientation into account
end if

Algorithm 42 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 (44)
end if
if $u_j > 0$ **then**
compute far-field velocity $\hat{\boldsymbol{u}}_{cL(j)}$ with Algorithm (43)

$$u_{inj} = \begin{cases} \hat{\boldsymbol{u}}_{cL(j)} \cdot \boldsymbol{n}_{wj}, & \text{fixed weir scheme } 4 \\ \hat{\boldsymbol{u}}_{cL(j)} \cdot \boldsymbol{n}_j, & \text{otherwise} \end{cases}$$

$\mathcal{A}_{\mathsf{i}j} = \frac{u_j}{2\Delta x_j}$
$\mathcal{A}_{\mathsf{e}j} = \mathcal{A}_{\mathsf{e}j} - \frac{u_{in_j}^2}{2\Delta x_j}$
else
as above by interchanging $L(j)$ and $R(j)$ and taking reversed orientation into account
end if

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.

$$\hat{u}_j = \frac{h_{uj}}{\hat{h}_j} u_j, \tag{6.204}$$

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 (43). Note that the faces that are associated with a weir are identified with $iadv_j = 21$, see Algorithm (40), so $\hat{\mathcal{J}}(k)$ is the set of faces of cell k without the faces that are associated to a fixed weir.

Algorithm 43 getucxucynoweirs: reconstruct a cell-centered velocity vector near a fixed wear without the weir itself

$$\hat{R}(k) = \{j \in R(k) | iadv_j \neq 21\} \tag{6.205}$$

$$\hat{h}_k = \begin{cases} \frac{1}{\sum_{j \in \hat{R}(k)} w_{uj}} \sum_{j \in R(k)} w_{uj} h_{uj}, & \hat{R}(k) \neq \emptyset \\ 0, & \text{otherwise} \end{cases} \tag{6.206}$$

$$\begin{aligned} \boldsymbol{u}_{ck} = \frac{1}{b_{Ak}} (& \sum_{j \in \{l \in \hat{R}(k) | s_{l,k} = 1\}} \alpha_j \Delta x_j w_{uj} \boldsymbol{n}_j u_j + \\ & \sum_{j \in \{l \in R(k) \setminus \hat{R}(k) | s_{l,k} = 1\}} \alpha_j \Delta x_j w_{uj} \boldsymbol{n}_j u_j \min(1, \frac{h_{uj}}{\hat{h}_k}) + \\ & \sum_{j \in \{l \in \hat{R}(k) | s_{l,k} = -1\}} (1 - \alpha_j) \Delta x_j w_{uj} \boldsymbol{n}_j u_j + \\ & \sum_{j \in \{l \in R(k) \setminus \hat{R}(k) | s_{l,k} = -1\}} (1 - \alpha_j) \Delta x_j w_{uj} \boldsymbol{n}_j u_j \min(1, \frac{h_{uj}}{\hat{h}_k})) \end{aligned} \tag{6.207}$$

For the advection downstream of the wear the cell-centered velocity vectors get the opposite treatment with Algorithm (44).

Algorithm 44 getucxucyweironly: reconstruct a cell-centered velocity vector near a fixed wear with the weir itself

$$\bar{\mathcal{J}}(k) = \{j \in \mathcal{J}(k) | iadv_j = 21\} \tag{6.208}$$

$$\bar{h}_k = \begin{cases} \frac{1}{\sum_{j\in\bar{\mathcal{J}}(k)} w_{uj}} \sum_{j\in\bar{\mathcal{J}}(k)} w_{uj} h_{uj}, & \bar{R}(k) \neq \emptyset \\ 0, & \text{otherwise} \end{cases} \tag{6.209}$$

$$\begin{aligned} \boldsymbol{u}_{ck} \;=\; \frac{1}{b_{Ak}}\Big(& \sum_{j\in\{l\in\bar{\mathcal{J}}(k)|s_{l,k}=1\}} \alpha_j \Delta x_j w_{uj} \boldsymbol{n}_j u_j + \\ & \sum_{j\in\{l\in\mathcal{J}(k)\setminus\bar{R}(k)|s_{l,k}=1\}} \alpha_j \Delta x_j w_{uj} \boldsymbol{n}_j u_j \max(1, \frac{h_{uj}}{\bar{h}_k}) + \\ & \sum_{j\in\{l\in\bar{\mathcal{J}}(k)|s_{l,k}=-1\}} (1-\alpha_j) \Delta x_j w_{uj} \boldsymbol{n}_j u_j + \\ & \sum_{j\in\{l\in\mathcal{J}(k)\setminus\bar{R}(k)|s_{l,k}=-1\}} (1-\alpha_j) \Delta x_j w_{uj} \boldsymbol{n}_j u_j \max(1, \frac{h_{uj}}{\bar{h}_k})\Big) \end{aligned} \tag{6.210}$$

6.7.4 Supercritical discharge

Supercritical conditions are defined by

$$\zeta^*_{R(j)} \leq z_c + \frac{2}{3} E_j, \tag{6.211}$$

where we let superscript * indicate supercritical and stationary conditions. The water height at the crest is then according to Equation (6.198)

$$h^*_{u_j} = \frac{2}{3} E_j. \tag{6.212}$$

and the crest velocity according to Equation (6.200)

$$u^*_j = \sqrt{\frac{2}{3} g E_j}. \tag{6.213}$$

The face-based discharge under stationary and supercritical conditions is then by definition

$$q^*_j := A^*_{u_j} u^*_j = w_{uj} h^*_{u_j} u^*_j = w_{uj} \frac{2}{3} E_j \sqrt{\frac{2}{3} g E_j}, \tag{6.214}$$

where we have used Equation (6.31) and Algorithm (3).

6.7.5 Empirical formulas for subgrid modelling of weirs

The energy loss due to a weir described by the loss of energy height ([m]). The energy loss in the direction perpendicular to the weir is denoted as ΔE. This energy loss is added as an opposing force in the momentum equation by adding a term $-g\Delta E/\Delta x$ to the right hand side of the momentum equation, resulting in a jump in the water levels by ΔE at the location of the weir.

The computation of the energy loss depends on the flow condition. There is a distinction between a subcritical flow condition and a supercritical flow condition. Furthermore, there are two different empirical formulations for the energy loss in use in D-Flow FM:

1 **Tabellenboek**
Then, the energy loss is computed according to the following principles, see Wijbenga (1990):

- ◇ **In critical flow conditions**
To match the theoretical critical flow condition on the crest (flow velocity and wave propagation speed are the same on the crest).
- ◇ **In subcritical flow conditions**
Based on the so-called "Tabellenboek"' approach, see Vermaas (1987) and the formula according to Carnot. The formulation was fitted to match laboratory measurements with hydraulically smooth weirs and with the ramp factors $m_1 = m_2$ both equal to 4.0: see (Equation (6.222)) for the definitions of m_1 and m_2.
Whether Carnot's formula or the Tabellenboek is used depends on the flow velocity above the weir: if the flow velocity at the weir is less than $0.25\ m/s$, the energy loss is calculated according to the Carnot equation for the energy loss in a sudden expansion. If the flow velocity above the weir is more than $0.50\ m/s$, the energy loss is determined by interpolation of measured data which are collected in the Tabellenboek. When the flow velocity is between 0.25 and $0.50\ m/s$, a weighted average is taken between the energy loss following Carnot and the measurements.
For velocities at the weir less than $0.25\,m/s$ the energy loss is calculated according to Carnot's law:

$$\Delta E = \frac{1}{2g}\left(U_{weir} - \frac{Q_{weir}}{\zeta_2 + d_2}\right)^2 \tag{6.215}$$

with U_{weir} the flow velocity on the weir and ζ_2 the downstream water level and U_2 the downstream flow velocity.

2 **Villemonte**
The second available formulation is the formulation proposed by Villemonte (1947). The formula has terms for different aspects of the weir's geometry and for the vegetation on it, more than the Tabellenboek-formulation. This formulation involves a number of parameters, for which realistic values need to be found.
The default values produce an energy loss which is very close to the energy loss found by the Tabellenboek formula. Alternative values for the tuning parameters were calculated by Sieben (2011).
Depending on the flow condition, the empirical discharge is processed into the model in one of the following two ways:

2.1 In critical flow, a loss of energy height is prescribed which causes the discharge to converge to the empirical discharge over a small period of time.
2.2 In subcritical flow, a loss of energy height is prescribed which is the same as the loss of energy height in a one-dimensional, steady flow with the given discharge.
Under the influence of (wind-)forces or two-dimensional effects, the discharge may not converge to the empirical value, even though the energy loss will be the same as in the one-dimensional, steady case.

3 In section 6.7.6 the Villemonte approach is described in detail.

6.7.6 Villemonte model for weirs

The Villemonte model is based on the analysis of a large number of measurements, which were fitted for a formula which expresses the discharge across the weir as a function of the

energy heights E_1 upstream and E_2 downstream of the weir:

$$Q = Q(E_1, E_2). \tag{6.216}$$

The energy heights E_1 and E_2 are given by:

$$E_1 = \zeta_1 + \frac{U_1^2}{2g}, \qquad E_2 = \zeta_2 + \frac{U_2^2}{2g}, \tag{6.217}$$

where the following notations are introduced:

ζ_1	upstream water level, measured from weir crest $[m]$
ζ_2	downstream water level, measured from weir crest $[m]$
g	gravitational acceleration $[m/s^2]$
U_1	upstream flow velocity component in direction towards the weir $[m/s]$
U_2	downstream flow velocity component in direction from the weir $[m/s]$

Apart from the energy heights, the discharge in the empirical formula ((6.216)) depends on the properties of the weir. The formula proposed by Villemonte is

$$Q = C_{d0}\, Q_c(E_1) \sqrt{1 - \max\left(0, \min\left(1, \left(\frac{E_2}{E_1}\right)^p\right)\right)}, \tag{6.218}$$

where the following notations are introduced:

Q	discharge per unit width across the weir $[m^2/s]$
C_{d0}	resistance coefficient of the weir $[-]$
$Q_c(E_1)$	theoretical value for discharge across the weir in case of critical flow $[m^2/s]$
p	power coefficient in discharge formula $[-]$

Determination of C_{d0}, Q_{th} and p

The determination of C_{d0}, Q_{th} and p is as described in the following three sections.

Theoretical critical discharge

The theoretical value Q_c for the discharge in case of critical flow is given by

$$Q_c = \frac{2}{3} E_1 \sqrt{\frac{2g}{3} E_1}. \tag{6.219}$$

Note: that, even in critical flow conditions, the discharge has the form ((6.218)), and therefore differs from the theoretical critical discharge Q_c.

Resistence coefficient of the weir

The resistance coefficient C_{d0} depends on the weir's vegetation in the following way:

$$C_{d0} = (1 + \xi_1/3)^{-3/2} C_{d0,ref}, \tag{6.220}$$

where $C_{d0,ref}$ is the resistance coefficient the weir would have if it had no vegetation, and where ξ_1 is the dimensionless vegetation coefficient, given by

$$\xi_1 = (1 - A_r \min(h_v, h_1))\, C_{drag} \tag{6.221}$$

where the following notations are introduced:

C_{drag}	user-specified drag coefficient $[-]$
A_r	user-specified vegetation density per linear meter $[1/m]$
h_v	user-specified vegetation height $[m]$
h_1	upstream water level measured from the crest $[m]$

The effects of the weir's geometry, vegetation and flow conditions on the resistance coefficient $C_{d0,ref}$ is modeled in the following way:

$$C_{d0,ref} = \quad c_1 \left(w \left(1 - \frac{1}{4} e^{-m_1/2} \right) + (1-w) \left(\frac{4}{5} + \frac{13}{20} e^{-m_2/10} \right) \right), \tag{6.222}$$

where the following notations were introduced:

c_1	user-specified calibration coefficient $[-]$, default $[.0$
	Note: the Tabellenboek measurements correspond to the default value $c_1 = 1.0$.
m_1	user-specified ramp of the upwind slope toward the weir (ratio of ramp length and height, $[-]$, default 4.0
m_2	user-specified ramp of the downwind slope from the weir $[-]$, default 4.0
w	interpolation weight $[-]$

The interpolation weight w is given by

$$w = e^{-E_1/L_{crest}}, \tag{6.223}$$

where L_{crest} is the length of the weir's crest $[m]$ in the direction across the weir.

Power coefficient p

The effect of the vegetation on the power-coefficient p is modeled in the following way:

$$p = \frac{(1+\xi_1/3)^3}{1+2\,\xi_1} p_{ref}, \tag{6.224}$$

where p_{ref} is the power coefficient found in absence of vegetation:

$$p_{ref} = \frac{27}{4\,C_{d0}^2} \left(\left(1 + \frac{d_1}{E_1} \left(1 - e^{-m_2/c_2} \right) \right)^{-2} - \left(1 + \frac{d_1}{E_1} \right)^{-2} \right)^{-1}. \tag{6.225}$$

The user-specified calibration coefficient c_2 has a default value $c_2 = 10$. This is the correct value for hydraulically smooth weirs. For hydraulically rough weirs, the value should be 50.

6.7.7 Grid snapping of fixed weirs and thin dams

All geographical features of a model that are described by x-, and y-coordinates, like fixed weirs, thin dams and cross-sections, have to be interpolated to the computational grid when running a model. The computational core of D-Flow FM automatically assigns these features to the corresponding net links of the grid. This is called grid snapping. In this section is explained how the grid snapping is implemented in the computational core of D-Flow FM. This can be checked with the graphical user interface via the grid snapping feature.

In Figure 6.12 eight examples are shown how grid snapping of fixed weirs and thin dams has been implemented. The upper four examples are for thin dams (in red), while the lower four examples involve fixed weirs (in blue). In all eight examples one computational cell is

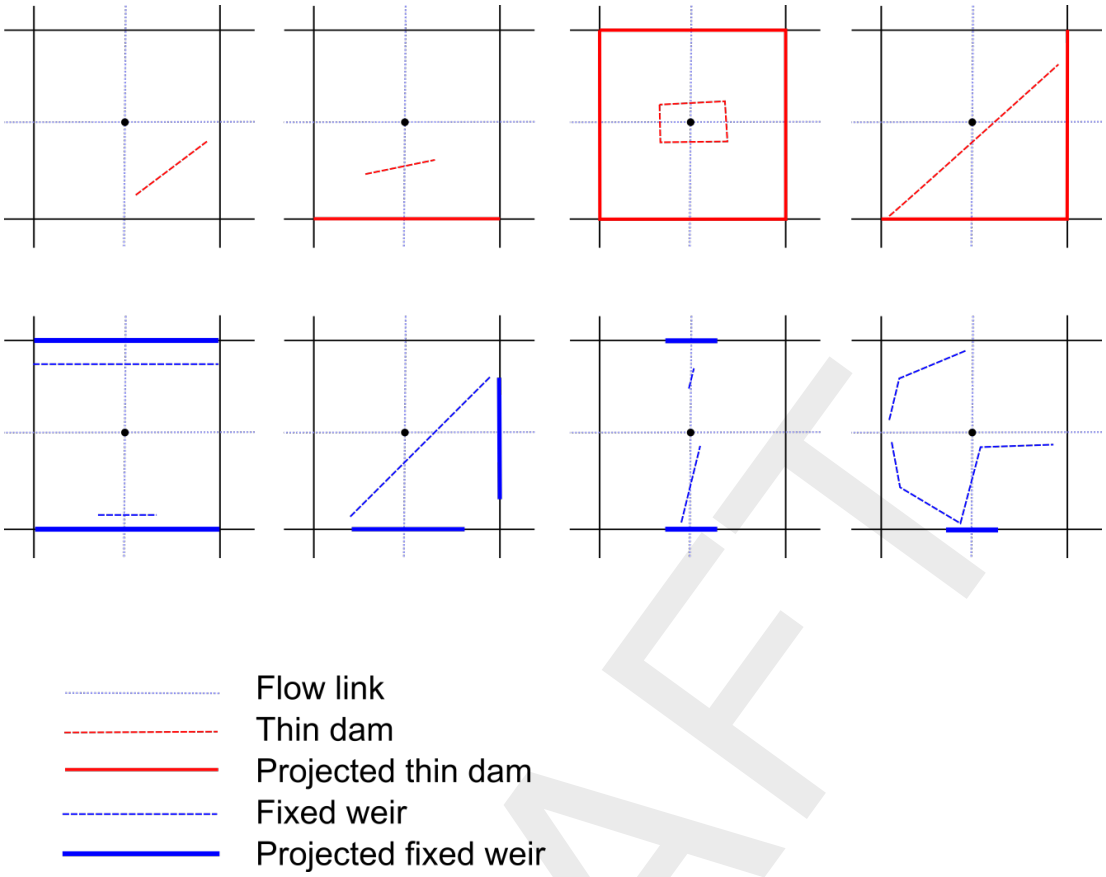


Figure 6.12: *Examples of grid snapping for fixed weirs and thin dams.*

shown with the water level point at the centre and four flow links that are connected to this water level point. If a thin dam or fixed weir intersects with a flow link, then this object will be snapped to the corresponding net link. In the left top example there is no intersection and thus no grid snapping to a net link. In the second example, there is one intersection with one flow link, in the third example four intersections and in the fourth example two intersections. This corresponds to the number of net links to which grid snapping have been applied. In the lower four examples of Figure 6.12 grid snapping of fixed weirs is explained. The algorithm for determining whether or not a fixed weir is snapped on a net link is the same as for thin dams and is explained above. The extra aspects for fixed weirs are its width and its crest height. The width of a fixed weir is illustrated in Figure 6.12, while the computation fo the crest height is illustrated in Figure 6.13. The following algorithm is applied for the computation of the crest height of a fixed weir:

- ◇ the crest level is the weighted average of the crest heights at the ends of the polyline of a fixed weir,
- ◇ if multiple fixed weirs intersect a flow link, then the maximum crest level of the interpolated values is taken,
- ◇ if multiple fixed weirs intersect a flow link, then the lowest toe level (in Dutch "teenhoogte") is taken. This is the case for both the left and the right side. Noted that the ground height is the distance between the crest level and the toe level. This is only relevant for toe levels of which the ground height is larger than zero. In case of a zero ground height, the corresponding toe level is neglected in the computation of the ground height.

Next, the input values of this fixed weir with the maximum crest height are used for the other quantities (ground height left, ground height right, talud left, talud right, crest length, vegetation

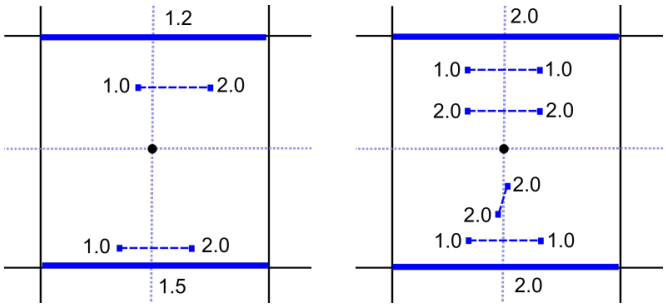


Figure 6.13: *Examples of computation of crest heights.*

coefficient)

The width (wu) of a fixed weir is determined by the corner (α) between the fixed weir and the net link and is computed according to

$$wu = \cos(\alpha) \times width, \tag{6.226}$$

with $width$ the width of a net link. If multiple fixed weirs intersect a flow link, then the maximum of the interpolated values is taken for the width.

6.8 Nested Newton non linear solver

In Algorithm (22) the method for performing a time step is with the Newton iteration presented. With the introduction of the Nested Newton iteration this algorithm is changed for 1d models.

Pressurized or partially pressurized flows are often modelled by introducing the so-called Preismann slot. (Preissmann A (1961)). This method has a number of disadvantages. The main disadvantage is the fact that the iteration process is not guaranteed to converge. As a result the time step must be limited from time to time.

In Casulli and Stelling (2013) the so-called Nested Newton method is presented. This method should counteract the disadvantages of the Preismann slot. In the Nested Newton method a second iteration layer is added to the non-linear iteration, hence the term "nested". In this method the width of a cross section is split in a non-decreasing part $p(x_k, z)$ and a non-increasing part $q(x_k, z)$.

Let $w(x, z)$ be the width of the channel at postion (x, z). The wet cross sectional area A_T is determined by:

$$A_T(x) = \int_{-\infty}^{\zeta} w(x, z)dz \tag{6.227}$$

In applications for open channel flow in general $w(x, z)$ is a non-decreasing function in z. In urban drainage systems this is not the case. However in all cases it is possible to define two non-decreasing functions $p(x, z)$ and $q(x, z)$, in such a way that:

$$w(x, z) = p(x, z) - q(x, z) \tag{6.228}$$

As a result the cross sectional total area can be written as:

$$A_T(x) = \int_{-\infty}^{\zeta} p(x, z)dz - \int_{-\infty}^{\zeta} q(x, z)dz \tag{6.229}$$

The Nested Newton time integration method can be summarized by (an elaborate description can be found in Algorithm (45)):

1 The index n indicates the time step number.
2 For the outer iteration loop the index m is used and corresponds with $\zeta_k^{n+1,m}$.
3 For the inner iteration loop the index p is used and corresponds with $\zeta_k^{n+1,m(p)}$.
4 During the inner iteration $\zeta_k^{n+1,m(p)}$ is updated.
5 The area of the cross section is calculated by Algorithm (46). As a result during the p-iteration the cross sectional width is non-decreasing
6 Once this iteration is converged, $\zeta_k^{n+1,m}$ is updated.
7 Return to step 4, until also the m-iteration loop is converged.

The non-linear time-step integration is summed up in Algorithm (22). The algorithm for the Nested Newton time-step integration is summed up in Algorithm (45). The main difference for this algorithm is the extra iteration loop with index m.

Algorithm 45 Nested Newton: 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 (14)
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 (15)
determine the set of water levels that need to be solved, Algorithm (17)
$m = 0$
$p = 0$
$\zeta_k^{n+1,m} = -bl_k$
$\zeta_k^{n+1,m(p)} = \zeta_k^n$
$converged_m = false$
$converged_p = false$
while not $converged_m$ **do**
while not $converged_p$ **do**
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 (18)
solve the unknown water levels and obtain $\zeta_k^{n+1,m(p+1)}$, Algorithm (23)
check positivity of water level with Algorithm (19) and repeat time-step if necessary with modified Δt (type 1) or ${h_u}_j^n$ (type 2, default)
compute water-column volume $V_k^{n+1,m(p+1)}$ and wet surface area $A_k^{n+1,m(p+1)}$ with Algorithm (46)
$converged_p = \max_k \left|\zeta_k^{n+1,m(p+1)} - \zeta_k^{n+1,m(p)}\right| < \varepsilon$
$p = p + 1$
end while
$\zeta_k^{n+1,m+1} = \zeta_k^{n+1,m(p)}$
$converged = \max_k \left|\zeta_k^{n+1,m+1} - \zeta_k^{n+1,m}\right| < \varepsilon$
$m = m + 1$
end while
end while
end while
$\zeta_k^{n+1} = \zeta_k^{n+1,m}$
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 (21)

Algorithm 46 Compute cross sectional area

$$V_k^{n+1,m(p)} = 0.5 \sum_{j \in \mathcal{J}(k)} A_{T_{ij}} dx_j \tag{6.230}$$

.

$$A_{T_{ij}}^{n+1,m(p)} = \int_{-\infty}^{\zeta_k^{n+1,m(p)}} p(x_{ij}, z)dz - \int_{-\infty}^{\zeta_k^{n+1,m(0)}} q(x_{ij}, z)dz \tag{6.231}$$

$$A_k^{n+1,m(p)} = 0.5 \sum_{j \in \mathcal{J}(k)} p(x_{ij}, \zeta_k^{n+1,m(p)}) - q(x_{ij}, \zeta_k^{n+1,m(0)}) \tag{6.232}$$

.

where x_{ij} is located on the cell face j at the side of cell k.

Remark 6.8.1. In contradiction the claim of Casulli and Stelling (2013) an extra slot width was required in order to stabilize urban models in D-Flow FM.

@@@

Herman heeft hier het volgende over gezegd:

Met deze opmerking rijst twijfel over de correctheid van de DFM implementatie. (Twee proffen zeggen A, wij constateren niet A). Dus of we twijfelen zelf ook over de juistheid van de DFM implementatie, en dan moeten we dat oplossen. Of we hebben hier een verklaring voor en dan moeten we die geven. Ik heb geen keihard bewijs maar heb het vermoeden dat het ligt aan de soms beroerde conditie van de matrix. Want dit probleem treedt in mijn testen tot nu toe alleen op in 1D2D modellen, niet in uitsluitend 1D modellen.

Ik vind het wel van belang om ervoor te zorgen, dat we dit gaan uitzoeken. Als we deze opmerking weghalen, dan lopen we het gevaar dat het vergeten wordt. Maar nu staat het wel heel erg expliciet in de documentatie. En die moeten we niet vervuilen met issues, die verder uitgezocht moeten worden.

@@@

Remark 6.8.2. The standard settings for the stop criterium for the conjugate gradient solver is 10^{-14}. The stop criterium for the Newton iteration is 10^{-8}. These two values have to be different.

7 Numerical schemes for three-dimensional flows

7.1 Governing equations

In D-Flow FM transport is formulated as,

$$\frac{d}{dt}\int_{V(t)}\varphi\,dV+\int_{\partial V(t)}\varphi\,(\boldsymbol{u}-\boldsymbol{v})\cdot\boldsymbol{n}\,dS=\int_{\partial V(t)}\left(K\tilde{\nabla}\varphi\right)\cdot\boldsymbol{n}\,dS+\int_{V(t)}s\,dV \quad (7.1)$$

where $V\,(t)$ is a three-dimensional control volume, φ is a transport variable, $\boldsymbol{u}$ the flow velocity field, $\boldsymbol{v}$ the velocity of the (vertically) moving control volume, K is a diagonal matrix $K=diag\,(\nu_H,\nu_H,\nu_V)$, $\tilde{\nabla}$ is the gradient operator in 3D, with diffusion coefficients and s a source term. In case of three-dimensional (layer-averaged) flow, with Δz a layer thickness from $z_1\,(x,y,t)$ to $z_2\,(x,y,t)$, we obtain

$$\begin{aligned}&\frac{\partial\Delta z\varphi}{\partial t}+\nabla\cdot(\Delta z\boldsymbol{u}\varphi)+\omega_{z_2}\left[\varphi\right]_{z=z_2}-\omega_{z_1}\left[\varphi\right]_{z=z_1}=\nabla\cdot(\Delta z\nu_H\nabla\varphi)\\&+\left[\nu_V\frac{\partial\varphi}{\partial z}-\nu_H\nabla z_2\cdot\nabla\varphi\right]_{z=z_2}-\left[\nu_V\frac{\partial\varphi}{\partial z}-\nu_H\nabla z_1\cdot\nabla\varphi\right]_{z=z_1}+\Delta zs\end{aligned} \quad (7.2)$$

where $\boldsymbol{u}$ and ∇ still the horizontal components are meant, i.e. $\boldsymbol{u}=(u,v)^T$ and $\nabla=\left(\frac{\partial}{\partial x},\frac{\partial}{\partial y}\right)^T$ and ν_V is the vertical diffusion coefficient. Furthermore, ω_{z1} and ω_{z2} are the velocity components normal, relative to the moving $z=z_1$ and $z=z_2$ layer interfaces, respectively.

The continuity equation is derived by setting $\varphi=1$ and $s=0$,

$$\frac{\partial\Delta z}{\partial t}+\nabla\cdot(\Delta z\boldsymbol{u})+\omega_{z_2}-\omega_{z_1}=0 \quad (7.3)$$

Summing up all equations along the layers, and setting zero flux condition at the bed and the free surface, it yields:

$$\frac{\partial h}{\partial t}+\nabla\cdot(h\boldsymbol{u})=hs \quad (7.4)$$

In a similar way, the horizontal momentum equation can be obtained by setting $\varphi=u$. Unlike the continuity equation, the momentum equation is not integrated over the depth.

$$\begin{aligned}&\frac{\partial\Delta zu}{\partial t}+\nabla\cdot(\Delta z\boldsymbol{u}u)+\omega_{z_2}u_{z_2}-\omega_{z_1}u_{z_1}=\nabla\cdot(\Delta z\nu_H\nabla u)\\&+\left[\nu_V\frac{\partial u}{\partial z}-\nu_H\nabla z_2\cdot\nabla u\right]_{z_2}-\left[\nu_V\frac{\partial u}{\partial z}-\nu_H\nabla z_1\cdot\nabla u\right]_{z_1}+\Delta zs\end{aligned} \quad (7.5)$$

subtracting Equation (7.3) from Equation (7.5) yields,

$$\begin{aligned}\frac{\partial u}{\partial t}+&\quad\tfrac{1}{\Delta z}\left[\nabla\cdot(\Delta z\boldsymbol{u}u)-\nabla\cdot(\Delta z\boldsymbol{u})-\nabla\cdot(\Delta z\nu_H\nabla\boldsymbol{u})\right]\\+&\quad\tfrac{1}{\Delta z}\left[\omega_{z_2}\left(u_{z_2}-u\right)-\omega_{z_1}\left(u_{z_1}-u\right)\right]=\\+&\quad\tfrac{1}{\Delta z}\left[\nu_V\tfrac{\partial u}{\partial z}-\nu_H\nabla z_2\cdot\nabla u\right]_{z_2}-\tfrac{1}{\Delta z}\left[\nu_V\tfrac{\partial u}{\partial z}-\nu_H\nabla z_1\cdot\nabla u\right]_{z_1}+s\end{aligned} \quad (7.6)$$

The last term in the right-hand side of Equation (7.6) includes the source terms, namely pressure. It can be described by $s=s_p$. The pressure term is imposed on all flow cells as

$$s_p=-g\frac{\partial\zeta}{\partial x} \quad (7.7)$$

The bed friction acts as surface force and it affects the flow in the first layer close to the bed.

$$\nu_V \left.\frac{\partial u}{\partial z}\right|_{z=0} = \frac{\tau_b}{\rho} \tag{7.8}$$

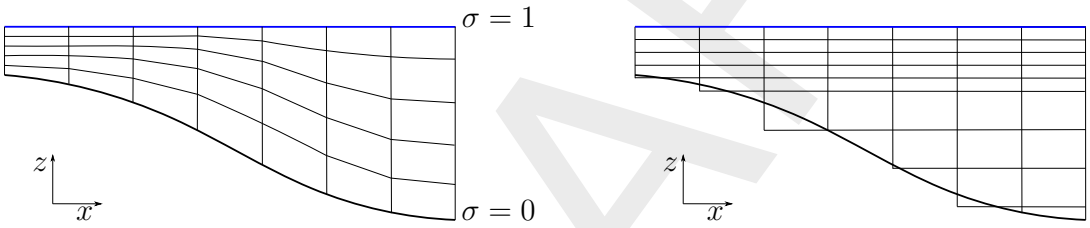
The wind force also acts as a surface force on the free surface, and hence it affects the top layer of the flow.

$$\nu_V \left.\frac{\partial u}{\partial z}\right|_{z=z_{max}} = C_d \frac{\rho_a}{\rho_w} W^2 \tag{7.9}$$

where W is the speed of the wind, ρ_a is the air density, ρ_w is the density of water and C_d is air-water friction coefficient.

7.2 Three-dimensional layers

In D-Flow FM two type of grid topologies in the vertical direction are applied, σ and z grids. σ-layers are layers which divide the computational regions between the bed and the free surface. These layers are adaptive and the interfaces of the layers may change in time associates with the deformation of the bed and free surface, see Figure 7.1a.



(a) σ-layers **(b)** z-layers

Figure 7.1: *A schematic view of σ- and z-layers.*

Unlike the σ-layers, z-layers are strictly horizontal and they don't adapt the temporal variation of the bed and free surface, see Figure 7.1b.

7.2.1 σ-layers

In σ-grid the vertical distribution of the grid form layers which can adapt the geometry of the bed and free surface. In D-Flow FM the thickness of sigma-layers can be uniform (equidistant in the vertical direction), user specified, or stretched around a user defined level γ with stretching factor of α (Algorithm (47)). For stretching around a user defined level, Equation (7.10) and Equation (7.11) are applied for both bottom and top parts.

Algorithm 47 flow_allocflow:

$$\Delta z_{i,1} = \begin{cases} \text{user specified}, & \text{stretching type} = 1, \quad i = 1 \\ \frac{1-\alpha_i}{1-\alpha_i^m} \times \gamma, & \text{stretching type} = 2, \quad i = 1,2 \\ \frac{1}{m}, & \text{otherwise} \qquad\quad i = 1 \end{cases} \tag{7.10}$$

$$\Delta z_{i,k} = \begin{cases} \text{user specified}, & \text{stretching type} = 1, \quad i = 1 \\ \Delta z_{i,k-1} \times \alpha_i, & \text{stretching type} = 2, \quad i = 1,2 \\ \Delta z_{i,k-1}, & \text{otherwise}, \qquad\quad i = 1 \end{cases} \tag{7.11}$$

7.2.2 z-layers

7.3 Connectivity

In D-Flow FM the horizontal connectivity of the computational cells in three-dimensional are defined identical with that of 2D case.

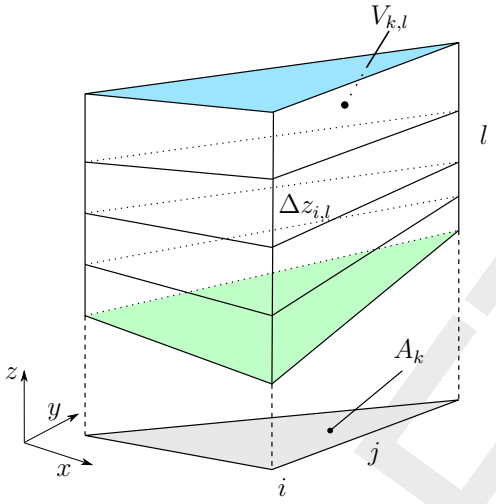


Figure 7.2: *Layer distribution in 3D.*
A_k, projected area of cell k.
$V_{k,l}$, volume in layer l of cell k.
$\Delta z_{i,l}$ thickness of layer l above node i.

However, in order to take the vertical distribution into account, a structured type of bookkeeping is applied. The data is stored in an one-dimensional array, and for each base node (2D flow node on the bed), two pointers are defined associated to the bottom and top cells in the vertical direction, defined by $\mathcal{K}_b(k)$ and $\mathcal{K}_t(k)$, respectively. Similar type of connectivity is also applied for the flow links in the vertical direction. Each base link (2D flow link on the bed), similar to flow nodes, consists of two pointers associated to the links in the bottom and top levels in the vertical direction, defined by $\mathcal{L}_b(j)$ and $\mathcal{L}_t(j)$.

The connectivity translates directly to administration in the D-Flow FM code as follows:

$\mathcal{K}_b(k)$: `kbot(k)`,
$\mathcal{K}_t(k)$: `ktop(k)`,
$\mathcal{L}_b(j)$: `Lbot(j)`,
$\mathcal{L}_t(j)$: `Ltop(j)`,

where j is the link index for the base flow nodes k.

Note: In D-Flow FM the layers are defined by one-dimensional array. For each cell column the layers change from $\mathcal{K}_b(k)$ to $\mathcal{K}_t(k)$ and for each flow-link column they change from $\mathcal{L}_b(j)$ to $\mathcal{L}_t(j)$. However for the reason of clarity, we define it by two-dimensional indices at which the first index specifies the base cell k (or flow link j) and the second index specifies the layer number l.

7.4 Spatial discretization

Algorithm 48 sethu: compute the layer depths at flow-link location j

$$
\begin{aligned}
z_{uj,l} &= \begin{cases} z_{L(j),l}, & u_{j,l} > 0 \quad \vee \quad u_{j,l} = 0 \quad \wedge \quad \zeta_{L(j)} > \zeta_{R(j)} \\ z_{R(j),l}, & u_{j,l} < 0 \quad \vee \quad u_{j,l} = 0 \quad \wedge \quad \zeta_{L(j)} \leq \zeta_{R(j)} \end{cases} \\
\sigma &= \frac{z_{uj,l} - z_{uj,0}}{z_{uj,\mathcal{M}(j)} - z_{uj,0}} \\
\tilde{h}_{u_{j,l}} &= \sigma h_{u_j} \\
A_{uj,l} &= \left(\tilde{h}_{u_{j,l}} - \tilde{h}_{u_{j,l-1}} \right) w_{uj}
\end{aligned}
$$

7.4.1 Continuity equation

The continuity equation Equation (7.4) is spatially discretized on the water column as

$$
\frac{dV_k}{dt} = - \sum_{j \in \mathcal{J}(k)} \sum_{l=1}^{\mathcal{M}(j)} A_{uj,l} u_{j,l} s_{j,k} \tag{7.12}
$$

where $\mathcal{M}(j)$ is the maximum layers number at the location of link j, V_k is the volume of water column for base cell k computed with Algorithm (20), $A_{uj,l}$ approximates the flow area of face j at layer l computed with Algorithm (3), $u_{j,l}$ is the normal velocity component associated to face (j,l) and $s_{j,k}$ accounts for the orientation of face j with respect to cell k.

Face based layer depth

The face-based layer depths reconstructed from the cell-centered layer depths with an upwind approximation. The face-based layer depth, $\tilde{h}_u$, is then applied to calculate the cross-sectional area, $A_{uj,l}$ at layer j (see Algorithm (48)).

7.4.2 Momentum equation

Unlike the two-dimensional case, the discretization of the momentum equation in 3D occurs on all flow faces associated with the layers (Horizontally). However, the discretization of the advection and diffusion terms are identical with that of the two-dimensional case, except it is applied along the layers. Hence, the advection and diffusion terms can be expressed by

$$
\begin{aligned}
advec &= \frac{1}{\Delta z} \left[\nabla \bullet (\Delta z \boldsymbol{u} \boldsymbol{u}) - \boldsymbol{u} \nabla \bullet (\Delta z \boldsymbol{u}) - \nabla \bullet (\nu \Delta z (\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^{\mathrm{T}})) \right]_{j,l} \bullet \boldsymbol{n}_j \\
&\approx \mathcal{A}_{\mathrm{i}j,l} u_{j,l} + \mathcal{A}_{\mathrm{e}j,l}
\end{aligned} \tag{7.13}
$$

where $\mathcal{A}_{\mathrm{i}j,l}$ and $\mathcal{A}_{\mathrm{e}j,l}$ are the implicit and explicit parts, respectively. Because of similarity with the two-dimensional case, the derivation of the equations is not discussed here. The advection term is discretized in Algorithm (4) and the diffusion term in Algorithm (11) for each layer. Similar to the two-dimensional case, the limiters, used for the advection, is described in Algorithm (9) and Algorithm (10) on each layer.

The water level-gradient term projected in the face-normal direction is discretized along the water column as

$$
\nabla \zeta|_j \bullet n_j \approx \frac{g}{\Delta x_j} \left(\zeta_{R(j)} - \zeta_{L(j)} \right) \tag{7.14}
$$

The contribution of the bed friction term into the momentum equation occurs only on the flow faces associated with the first σ-layer. However, unlike the 2D case, this term is not discretized

Algorithm 49 getustbcfuhi: compute the bed shear stress for the first σ-layer

$$
\begin{aligned}
S &= \ln\left(\frac{\Delta z_{uj,1}/2 + \mu z_0}{z_0}\right) \quad , \quad m = 0 \\
S &= \ln\left(\frac{\Delta z_{uj,1} + \mu z_0}{e z_0}\right) \quad , \quad m = 1 \\
S &= \ln\left(\frac{\Delta z_{uj,1}/e + \mu z_0}{z_0}\right) \quad , \quad m = 2 \\
S &= \ln\left(\frac{\Delta z_{uj,1}/2 + z_0}{z_0}\right) \quad , \quad m = 3 \\
\sqrt{c_f} &= \frac{\kappa}{S} \\
u_j^* &= u_{j,1}^{n+1}\sqrt{c_f}
\end{aligned}
$$

in its original form. The discretization of this term is done by fitting a logarithmic-law for rough beds. The merely of this approach is in finding the shear velocity by means of integration of the log-law wall function. The log-law wall-function for rough beds reads

$$\frac{u}{u^*} \approx \frac{1}{\kappa}\ln\left(\frac{z + \mu z_0}{z_0}\right) \tag{7.15}$$

where κ is the Von Kármán constant, z_0 is the roughness height and μ is a constant equal to 1 or 9. Integration of Equation (7.15) for the first layer at flow link j gives

$$u_{j,1} = \frac{u_j^*}{\kappa}\left[\left(1 + \mu\frac{z_0}{\Delta z_{uj,1}}\right)\ln\left(\frac{\Delta z_{uj,1} + \mu z_0}{z_0}\right) - \frac{z_0}{\Delta z_{uj,1}}\mu\ln(\mu) - 1\right] \tag{7.16}$$

Considering $z_0/\Delta z_{uj,1}$ to be small, the bed shear velocity will be

$$u_{j,1} \approx \frac{u^*}{\kappa}\ln\left(\frac{\Delta z_{uj,1} + \mu z_0}{e z_0}\right) \tag{7.17}$$

Then the shear velocity is derived as

$$u_j^* = \frac{\kappa}{\ln\left(\frac{\Delta z_{uj,1} + \mu z_0}{e z_0}\right)} u_{j,1} = u_{j,1}\sqrt{c_f} \tag{7.18}$$

The bed shear stress is defined as $\tau_b = \rho u^{*2}$. However, different forms of log-law functions are used in D-Flow FM. The calculation of the bed shear stress is given in Algorithm (49).

Remark 7.4.1. The log-law wall function is valid for fully developed flow. The bed shear stress under fully developed flow is lower than that of non-developed flow. Therefore, Equation (7.18) underestimates the bed friction for unsteady flows.

7.5 Temporal discretization

Similar to the 2D part, the spatial discretization in 3D is also performed in a staggered manner. The velocity normal components $u_{j,l}$ are defined at the cell faces (j, l), with face normal vector $\boldsymbol{n}_{j,l}$, and the water levels ζ_k at cell centers k. If advection and diffusion are spatially discretized as in Equation (7.13) then the temporal discretization of Equation (7.5) is

$$
\begin{aligned}
&\frac{\partial u}{\partial t} + Advec + \frac{1}{\Delta z}\left[\omega_{z_2}\left(u_{z_2} - u\right) - \omega_{z_1}\left(u_{z_1} - u\right)\right] \\
&\quad = -g\frac{\partial \zeta}{\partial x} + \frac{1}{\Delta z}\left[\nu_V\frac{\partial u}{\partial z}\right]_{z_2} - \frac{1}{\Delta z}\left[\nu_V\frac{\partial u}{\partial z}\right]_{z_1}
\end{aligned} \tag{7.19}
$$

Remark 7.5.1. Note that in D-Flow FM the second term in the vertical diffusion term, namely $\nu_H \delta z \bullet \nabla u$ is neglected in. It is done for simplification of the discretization. However, this term becomes important in when the layer-level gradient is large, and neglecting this term may cause large error.

After substitution of $Advec$ from Equation (7.13) and discretize the other terms implicitly, it yields the following discretized form of equation

$$\begin{aligned}\frac{1}{\Delta t}u_{j,l}^{n+1} + \mathcal{A}_{\mathrm{i}j,l}u_{j,l}^{n+1} + \mathcal{A}_{\mathrm{e}j,l} + \omega_2' u_{z_2}^{n+1} - \omega_1' u_{z_1}^{n+1} - (\omega_2' - \omega_1')\, u_{j,l}^{n+1} \\ = \frac{1}{\Delta t}u_{j,l}^{n} - \frac{g\theta_j}{\Delta x_j}\left(\zeta_{R(j)}^{n+1} - \zeta_{L(j)}^{n+1}\right) - \frac{g\,(1-\theta_j)}{\Delta x_j}\left(\zeta_{R(j)}^{n} - \zeta_{L(j)}^{n}\right) \\ +\nu_2'\left(u_{j,l+1}^{n+1} - u_{j,l}^{n+1}\right) - \nu_1'\left(u_{j,l}^{n+1} - u_{j,l-1}^{n+1}\right)\end{aligned} \tag{7.20}$$

where u_{z_1} and u_{z_2} are the upwinded velocities, $\omega_1' = \omega_{z_1}/\Delta z_{j,l}$, $\omega_2' = \omega_{z_2}/\Delta z_{j,l}$, $\nu_1' = \nu_1/z_{j,l}\Delta z_1$ and $\nu_2' = \nu_2/z_{j,l}\Delta z_2$, $\Delta z_1 = (z_{j,l} + z_{j,l-1})/2$ and $\Delta z_2 = (z_{j,l} + z_{j,l+1})/2$. ν_1 and ν_2 are the vertical eddy diffusivity at the top and the bottom of the velocity control volume, respectively. Applying a first order upwind scheme, Equation (7.20) can be reformed as follows

$$\frac{g\theta_j}{\Delta x_j}\left(\zeta_{R(j)}^{n+1} - \zeta_{L(j)}^{n+1}\right) + a_l u_{j,l-1}^{n+1} + b_l u_{j,l}^{n+1} + c_l u_{j,l+1}^{n+1} = d_l \tag{7.21}$$

where

$$a_l = -\max(\omega_1', 0) - \nu_1' \tag{7.22}$$

$$b_l = \frac{1}{\Delta t} + \mathcal{A}_{\mathrm{i}j,l} + \max(\omega_2', 0) - \min(\omega_1', 0) - (\omega_2' - \omega_1') + \nu_2' + \nu_1' \tag{7.23}$$

$$c_l = \min(\omega_2', 0) - \nu_2' \tag{7.24}$$

$$d_l = \frac{1}{\Delta t}u_{j,l}^{n} - \frac{g\,(1-\theta_j)}{\Delta x_j}\left(\zeta_{R(j)}^{n} - \zeta_{L(j)}^{n}\right) \tag{7.25}$$

Remark 7.5.2. in D-Flow FM the discretization of vertical convection is also done by central scheme (javau = 4). It is found that the central scheme may lead to instabilities.

Remark 7.5.3. In D-Flow FM the vertical advection can be switched off by using javau = 0. However, switching off the vertical advection may lead to non-physical results.

We write Equation (7.20) in a general form as

$$u_{j,l}^{n+1} = -f_{uj,l}^{\ n}\left(\zeta_{R(j)}^{n+1} - \zeta_{L(j)}^{n+1}\right) + r_{uj,l}^{\ n} \tag{7.26}$$

Substituting Equation (7.26) in Equation (7.21) leads to the following relation for f_u and r_u

$$\begin{aligned}\frac{g\theta_j}{\Delta x_j}\left(\zeta_{R(j)}^{n+1} - \zeta_{L(j)}^{n+1}\right) + a_l\left[r_{uj,l-1} - f_{uj,l-1}\left(\zeta_{R(j)}^{n+1} - \zeta_{L(j)}^{n+1}\right)\right] \\ +b_l\left[r_{uj,l} \quad - f_{uj,l} \quad \left(\zeta_{R(j)}^{n+1} - \zeta_{L(j)}^{n+1}\right)\right] \\ +c_l\left[r_{uj,l+1} - f_{uj,l+1}\left(\zeta_{R(j)}^{n+1} - \zeta_{L(j)}^{n+1}\right)\right] = d_l\end{aligned} \tag{7.27}$$

Equation (7.27) needs to be satisfied for any given initial water level (e.g. $\zeta_{R(j)} = \zeta_{L(j)}$) . Hence, this equation can be splitted to two equations.

◇ by making a homogeneous field and dropping the pressure terms,
◇ by substituting the derived equation from the previous step, and derive a second equation.

Dropping the pressure terms in Equation (7.27) gives

$$a_l r_{uj,l-1} + b_l r_{uj,l} + c_l r_{uj,l+1} = d_l \tag{7.28}$$

Substituting Equation (7.28) in Equation (7.27), leads to the following equation for f_u

$$a_l f_{uj,l-1} + b_l f_{uj,l} + c_l f_{uj,l+1} = d'_l \tag{7.29}$$

where

$$d'_l = -\frac{g\theta_j}{\Delta x_j}\left(\zeta^{n+1}_{R(j)} - \zeta^{n+1}_{L(j)}\right) \tag{7.30}$$

This procedure is presented in Algorithm (50).

Equation (7.28) and Equation (7.29) are tri-diagonal matrices for r_u and f_u, respectively. They are solved by Thomas Algorithm for tri-diagonal matrices (See Algorithm (51))

Algorithm 50 vertical_profile_u0: compute ${f_u}^n_{j,l}$ and ${r_u}^n_{j,l}$ in $u^{n+1}_{j,l} = -{f_u}^n_{j,l}(\zeta^{n+1}_{R(j,l)} - \zeta^{n+1}_{L(j)}) + {r_u}^n_{uj}$

For each j:
$$a = 0, \quad b = 1/\Delta t, \quad c = 0, \quad d_1 = u^n_{j,\mathcal{L}_b(j)}/\Delta t$$

for $l = 1$ to $\mathcal{M}(j) - 1$ **do**

$$adv_1 = \max(\omega'_l, 0)$$
$$adv = -\min(\omega'_l, 0)$$
$$T_1 = (\nu_t + adv_1)/\Delta z_{l+1}$$
$$T_2 = (\nu_t + adv)/\Delta z_l$$
$$b_{l+1} = b_{l+1} + T_1$$
$$a_{l+1} = a_{l+1} - T_1$$
$$b_l = b_l + T_2$$
$$c_l = c_l - T_2$$
$$d_{l+1} = u^n_{j,l+1}/\Delta t$$

end for
for $l = 1$ to $\mathcal{M}(j)$ **do**

$$b_l = b_l + \mathcal{A}_{\mathrm{i}j,k}$$
$$d_l = d_l - \mathcal{A}_{\mathrm{e}j,k} - \frac{g(1-\theta_j)}{\Delta x_j}\left(\zeta^n_{R(j)} - \zeta^n_{L(j)}\right)$$
$$d'_l = \frac{g\theta_j}{\Delta x_j}$$

end for
Solve $a_l r_{uj,l-1} + b_l r_{uj,l} + c_l r_{uj,l+1} = d_l$ by Algorithm (51)
Solve $a_l f_{uj,l-1} + b_l f_{uj,l} + c_l f_{uj,l+1} = d'_l$ by Algorithm (51)

Algorithm 51 tridag: compute tridiagonal matrix of $a_i u_{i-1} + b_i u_i + c_i u_{i+1} = d_i$

$$\beta = b_1$$
$$u_1 = d_1/\beta$$

for $i = 2$ to n step 1 **do**

$$e_i = \frac{c_{i-1}}{\beta}$$
$$\beta = b_i - a_i e_i$$
$$u_i = \frac{d_i - a_i u_{i-1}}{\beta}$$

end for

$$u_i = u_i - e_{i+1} u_{i+1} \quad , \quad i = n-1, n-2, ..., 1$$

The continuity equation is discretized as

$$\frac{V_k^{n+1} - V_k^n}{\Delta t} = - \sum_{j \in \mathcal{J}(k)} \sum_{l=1}^{\mathcal{M}(j)} A_{uj,l}^{\ n} \left[\theta_j u_{j,l}^{n+1} + (1-\theta_j)\, u_{j,l}^n \right] s_{j,k} \tag{7.31}$$

where V_k^{n+1} is the volume of the water column at base cell k and $A_{uj,l}$ approximates the flow area of face j, l

$$A_{uj,l} = \Delta z_{j,l} w_j \tag{7.32}$$

with $\Delta z_{j,l}$ is the distance between two layers l and $l-1$ related to base link j, w_j is the width of base link j. Subsitution of Equation (7.26) in Equation (7.31) yields the following equation.

$$\frac{V_k^{n+1} - V_k^n}{\Delta t} + \sum_{j \in \mathcal{J}(k)} \sum_{l=1}^{\mathcal{M}(j)} A_{uj,l}^{\ n} \theta_j f_{uj,l}^{\ n} \zeta_k^{n+1} - \sum_{j \in \mathcal{J}(k)} \sum_{l=1}^{\mathcal{M}(j)} A_{uj,l}^{\ n} \theta_j f_{uj,l}^{\ n} \zeta_{O(k,j)}^{n+1}$$
$$= - \sum_{j \in \mathcal{J}(k)} \sum_{l=1}^{\mathcal{M}(j)} A_{uj,l}^{\ n} \left[(1-\theta_j)\, u_{j,l}^n + \theta_j r_{uj,l}^{\ n} \right] s_{j,k} \tag{7.33}$$

Equation (7.31) can be summarized as

$$\frac{V_k^{n+1} - V_k^n}{\Delta t} + B_k^n \zeta_k^{n+1} + \sum_{j \in \mathcal{J}(k)} \sum_{l=1}^{\mathcal{M}(j)} C_j^n \zeta_{O(k,j)}^{n+1} = d_k^n \tag{7.34}$$

where B_k^n (diagonal entries), C_j^n (off-diagonal entry) and d_k^n (right-hand side) are computed by Algorithm (52).

In order to solve Equation (7.34), we need to express V_k^{n+1} in the terms of ζ_k^{n+1}. Since this relation is non-linear, Equation (7.34) is solved iteratively by means of Newton iterations. After linearizion of the volume, we have

$$V_k^{n+1(p+1)} = V_k^{n+1(p)} + A_k^{n+1(p)} \left(\zeta_k^{n+1(p+1)} - \zeta_k^{n+1(p)} \right) \tag{7.35}$$

Algorithm 52 s1ini: compute the matrix entries and right-hand side in the water level equation $\frac{V_k^{n+1}-V_k^n}{\Delta t} + B_k^n\ \zeta_k^{n+1} + \sum_{j\in\mathcal{J}(k)} C_j^n\ \zeta_{O(k,j)}^{n+1} = d_k^n$, Equation (7.34)

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

where $A_k^{n+1(p)}$ is the wet bed area of cell k at (iterative) time level $n+1(p)$. Substituting Equation (7.35) into Equation (7.34), yields

$$B_{rk}^{\ n+1(p)}\zeta_k^{n+1(p+1)} + \sum_{j\in\mathcal{J}(k)} C_{r_j}^n\zeta_{O(k,j)}^{n+1(p+1)} = d_{r_k}^{n+1(p)} \tag{7.36}$$

the form of Equation (7.36) is identical with that of Equation (6.120) in 2D. Hence, the coefficients $B_{r_k}^n, C_{r_k}^n$ and $d_{r_k}^n$ are computed at the same way of 2D, shown in Algorithm (18). Similar to 2D, the unknown water levels $k \in \mathcal{K}$ are solved with Krylov solver (see section 6.3.1).

The time step is finalized by employing Equation (7.26) back for u^{n+1} as it is shown in Algorithm (53). Moreover, the velocities and the discharge are integrated over depth.

Algorithm 53 u1q1: update velocity $u_{j,k}^{n+1}$ and discharges q_j^{n+1} and $q_{aj}^{\ n+1}$

if $h_{uj}^{\ n} > 0$ **then**

$$u_{j,l}^{n+1} = -f_{uj,l}^{\ n}(\zeta_{R(j)}^{n+1} - \zeta_{L(j)}^{n+1}) + r_{uj,l}^{\ n}$$
$$q_{j,l}^{n+1} = A_{uj,l}^{\ n}\left(\theta_j u_{j,l}^{n+1} + (1-\theta_j)u_{j,l}^n\right)$$
$$q_{aj,l}^{\ n+1} = A_{uj,l}^{\ n} u^{n+1}$$
$$q_{j,0}^{n+1} = \sum_{l=1}^{\mathcal{M}(j)} q_{j,l}^{n+1}$$
$$q_{aj,0}^{\ n+1} = \sum_{l=1}^{\mathcal{M}(j)} q_{aj,l}^{\ n+1}$$
$$A_{uj,0}^{\ n+1} = \sum_{l=1}^{\mathcal{M}(j)} A_{uj,l}^{\ n+1}$$
$$u_{j,0}^{n+1} = q_{j,0}^{n+1}/A_{uj,0}^{\ n+1}$$

end if

7.6 Vertical fluxes

In order to solve the vertical advection in Equation (7.19), the values of ω_{z1} and ω_{z2} needs to be calculated. In D-Flow FM the vertical fluxes are evaluated as the superposition of two effects.

1 Mass balance for each cell
2 The vertical motion of the water surface.

If q_H describes the sum of horizontal fluxes, then

$$\delta q_{Hk,l} = -\sum_{j\in\mathcal{J}(k)} q_{j,l}s_{j,k} + \delta q_{zk,l} \tag{7.37}$$

where $\delta q_{Hk,l} = q_{Hk,l} - q_{Hk,l-1}$ is the vertical flux passing through the interface (k,l), and $\delta q_{zk,l}$ is the vertical flux from cell (k,l) under effect of vertical motion of the free surface.

The difference in the horizontal fluxes has to be compensated via the vertical fluxes by $\delta q_{Hk,l} = -\delta q_{Vk,l}$ (first under assumption $\delta q_{zk,l} = 0$). As there is no vertical flux through the bed (zero flux), the calculation of the flux starts from the first layer, and is advanced to the upper layers.

The change of the water level (and the vertical location of layers) induces extra vertical fluxes through layer interfaces. This flux is equal to the rate of volume which passes through the interface, because of its motion. This flux is equal to

$$q_\sigma = A(\Omega_k)\frac{z_{k,l}^{n+1} - z_{k,l}^{n}}{\Delta t} \tag{7.38}$$

where $z_{k,l}$ is the vertical position of the interface (k,l). Note that this flux does not effect the value of the vertical velocity of the flow. The calculation of the vertical fluxes and velocities is described in Algorithm (54).

Algorithm 54 u1q1: calculate the vertical fluxes q_H

if ${h_u}_j^n > 0$ **then**
  $q_{Hk,0} = 0$
  for $l = 1$ **to** $\mathcal{N}(k)$ **do**

$$\begin{aligned}
\delta q_{Vk,l} &= -\sum_{j\in\mathcal{J}(k)} q_{j,l}s_{j,k} \\
\delta q_{Hk,l} &= -\delta q_{Vk,l} \\
q_{Hk,l} &= q_{Hk,l-1} + \delta q_{Hk,l} \\
\omega_{k,l} &= \omega_{k,l-1} + \delta q_{Hk,l}/A\left(\Omega_k\right) \\
q_{Hk,l} &= q_{Hk,l} - A(\Omega_k)\frac{z_{k,l}^{n+1} - z_{k,l}^{n}}{\Delta t}
\end{aligned}$$

  end for
end if

Remark 7.6.1. In the calculation of the fluxes, by marching, along the water column from the bottom to the top, the fluxes on the water surface will not be equal to zero. The error in the vertical flux on the water surface is supposed to be small, and it is neglected. Hence, the procedure is not fully mass conservative.

7.7 Turbulence closure models

In D-Flow FM four types of turbulence closure models are employed.

1 Constant coefficient model

2 Algebraic eddy viscosity closure model
3 k-ε turbulence model
4 k-τ turbulence model

At the first model, the eddy viscosity is a user defined constant. The three last models are based on the so-called eddy viscosity concept of Kolmogorov and Prandtl. The eddy viscosity is related to a characteristic length scale and velocity scale. The common target of this models is to find the eddy viscosity ν_V.

7.7.1 Constant coefficient model

This model is the simplest closure based on a constant value which has to be specified by the user. We remark that a constant eddy viscosity will lead to parabolic vertical velocity profile as laminar flow.

7.7.2 Algebraic eddy viscosity closure model

The algebraic eddy viscosity model does not involve transport equations for the turbulent quantities. This so-called zero order closure scheme consists of algebraic formulation. This model uses analytical formulas to determine L and ν_V.

$$L = z_{uj}\left(1 - \frac{z_{uj}}{h_{uj}}\right) \tag{7.39}$$

$$\nu_V = L u_{*_b} \kappa \tag{7.40}$$

with κ the von Kármán constant. For homogeneous flow this leads to a logarithmic velocity profile. The computation of ν_V is given in Algorithm (55).

Algorithm 55 update_verticalprofiles: compute the vertical turbulent viscosity ν_V

$L = \tilde{h}_{u_{j,l}}$
Compute $\sqrt{c_f}$ from Algorithm (49)
$u_{*_j} = \sqrt{c_f} u_{j,l}^{n+1}$
$\nu_V = \kappa L u_{*_j}$

7.7.3 k-ε turbulence model

In the k-ε turbulence model, transport equations have to be solved for both the turbulent kinetic energy k and for the energy dissipation ε. The mixing length L is then determined from ε and k according to:

$$L = c_D \frac{k\sqrt{k}}{\varepsilon}. \tag{7.41}$$

The transport equations for k and ε are non-linearly coupled by means of their eddy diffusivity D_k, D_ε and dissipation terms. The transport equations for k and ε are given by:

$$\frac{\partial k}{\partial t} + u\frac{\partial k}{\partial x} + v\frac{\partial k}{\partial y} + \omega\frac{\partial k}{\partial z} = \frac{\partial}{\partial z}\left(D_k \frac{\partial k}{\partial z}\right) + P_k + B_k - \varepsilon \tag{7.42}$$

$$\frac{\partial \varepsilon}{\partial t} + u\frac{\partial \varepsilon}{\partial x} + v\frac{\partial \varepsilon}{\partial y} + \omega\frac{\partial \varepsilon}{\partial z} = \frac{\partial}{\partial z}\left(D_\varepsilon \frac{\partial \varepsilon}{\partial z}\right) + P_\varepsilon + B_\varepsilon - c_{2\varepsilon}\frac{\varepsilon^2}{k} \tag{7.43}$$

with

$$D_k = \frac{\nu_{mol}}{\sigma_{mol}} + \frac{\nu_V}{\sigma_k} \tag{7.44}$$

$$D_\varepsilon = \frac{\nu_V}{\sigma_\varepsilon} \tag{7.45}$$

In the production term P_k of turbulent kinetic energy, the horizontal gradients of the horizontal velocity and all the gradients of the vertical velocities are neglected. The production term is given by:

$$P_k = \nu_V \left[\left(\frac{\partial u}{\partial z} \right)^2 + \left(\frac{\partial v}{\partial z} \right)^2 \right] \tag{7.46}$$

In stratified flows, turbulent kinetic energy is converted into potential energy. This is represented by a buoyancy flux B_k defined by:

$$B_k = g \frac{\nu_V}{\rho \sigma_\rho} \frac{\partial \rho}{\partial z} \tag{7.47}$$

with the Prandtl-Schmidt number $\sigma_\rho = 0.7$ for salinity and temperature and $\sigma_\rho = 1.0$ for suspended sediments. The production term P_ε and the buoyancy flux B_ε are defined by:

$$P_\varepsilon = c_{1\varepsilon} \frac{\varepsilon}{k} P_k \tag{7.48}$$

$$B_\varepsilon = c_{1\varepsilon} \frac{\varepsilon}{k} \left(1 - c_{3\varepsilon}\right) B_k \tag{7.49}$$

with the calibration constants given by (Rodi, 1984):

$$c_{1\varepsilon} = 1.44, \tag{7.50}$$

$$c_{2\varepsilon} = 1.92, \tag{7.51}$$

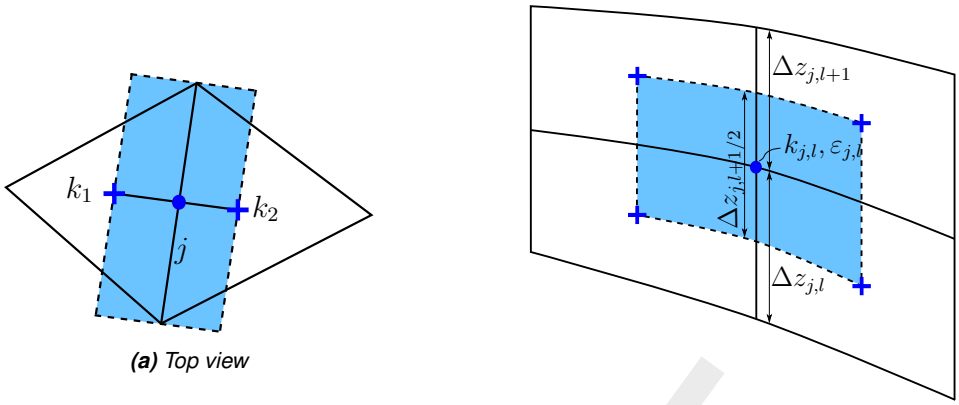
$$c_{3\varepsilon} = \begin{cases} 0.0 & \text{unstable stratification} \\ 1.0 & \text{stable stratification} \end{cases} \tag{7.52}$$

The vertical eddy viscosity ν_V is determined, with L prescribed by Equation (7.41), by:

$$\nu_V = c'_\mu L \sqrt{k} = c_\mu \frac{k^2}{\varepsilon} \tag{7.53}$$

with $c_\mu = c_D c'_\mu$.

The k-ε equations are discretized explicitly in the horizontal direction and implicitly in the vertical direction. The variables k and ε are located at the base flow-link and on the interface of the layers. Figure 7.3 shows an schematic view of the control volume for k and ε.



(a) *Top view*

(b) *Side view*

Figure 7.3: *The top view (a) and the side view (b) of the location of the turbulence variables on the computational grid.*

Advection

The horizontal advection term is discretized explicitly by means of first order upwind. The horizontal advection can be written in non-conservative form as

$$\boldsymbol{u} \cdot (\nabla k) = \nabla \cdot (\boldsymbol{u} k) - k \, (\nabla \cdot \boldsymbol{u}) \tag{7.54}$$

Integration of this term yields,

$$\int_V \boldsymbol{u} \cdot (\nabla k) \, dV = \int_V \nabla \cdot (\boldsymbol{u} k) \, dV - \int_V k \, (\nabla \cdot \boldsymbol{u}) \, dV \tag{7.55}$$

$$\int_V \boldsymbol{u} \cdot (\nabla k) \, dV = \int_A (\boldsymbol{u} k) \cdot \boldsymbol{n} dA - \int_A k \, (\boldsymbol{u} \cdot \boldsymbol{n}) \, dA \tag{7.56}$$

$$\boldsymbol{u} \cdot (\nabla k) = \frac{1}{V} \sum q_{j,l}^{n+1} k_{j,l}^n - \frac{k_{j,l}^{n+1}}{V} \sum q_{j,l}^{n+1} \tag{7.57}$$

Diffusion

The diffusion terms in equations Equation (7.42) and Equation (7.43) are discretized implicitly as,

$$\begin{aligned} \frac{\partial}{\partial z} \left(D_k \frac{\partial \tau}{\partial z} \right) \approx & \frac{\theta D_{k2}}{\Delta z_{j,l+1} \Delta z_{j,l+1/2}} \left(k_{j,l+1}^{n+1} - k_{j,l}^{n+1} \right) - \frac{\theta D_{k1}}{\Delta z_{j,l} \Delta z_{j,l+1/2}} \left(k_{j,l}^{n+1} - k_{j,l-1}^{n+1} \right) \\ & + \frac{(1-\theta) D_{k2}}{\Delta z_{j,l+1} \Delta z_{j,l+1/2}} \left(k_{j,l+1}^{n} - k_{j,l}^{n} \right) - \frac{(1-\theta) D_{k1}}{\Delta z_{j,l} \Delta z_{j,l+1/2}} \left(k_{j,l}^{n} - k_{j,l-1}^{n} \right) \end{aligned} \tag{7.58}$$

$$\begin{aligned} \frac{\partial}{\partial z} \left(D_\varepsilon \frac{\partial \varepsilon}{\partial z} \right) \approx & \frac{\theta D_{\varepsilon 2}}{\Delta z_{j,l+1} \Delta z_{j,l+1/2}} \left(\varepsilon_{j,l+1}^{n+1} - \varepsilon_{j,l}^{n+1} \right) - \frac{\theta D_{\varepsilon 1}}{\Delta z_{j,l} \Delta z_{j,l+1/2}} \left(\varepsilon_{j,l}^{n+1} - \varepsilon_{j,l-1}^{n+1} \right) \\ & + \frac{(1-\theta) D_{\varepsilon 2}}{\Delta z_{j,l+1} \Delta z_{j,l+1/2}} \left(\varepsilon_{j,l+1}^{n} - \varepsilon_{j,l}^{n} \right) - \frac{(1-\theta) D_{\varepsilon 1}}{\Delta z_{j,l} \Delta z_{j,l+1/2}} \left(\varepsilon_{j,l}^{n} - \varepsilon_{j,l-1}^{n} \right) \end{aligned} \tag{7.59}$$

where indices 1 and 2 refer to the values on the bottom and the top of control volume, respectively. The production terms for k equation is then discretized as

$$P_k \approx \nu_V \frac{\left(u^{n+1}_{j,l+1} - u^{n+1}_{j,l}\right)^2 + \left(v^{n+1}_{j,l+1} - v^{n+1}_{j,l}\right)^2}{\Delta z_{j,l+1/2}} \tag{7.60}$$

The dissipation term is a sink and the buoyancy term may act as a sink and can destabilize the system. In order to conserve a diagonal dominant matrix, a semi-implicit method (Patankar trick) is used to guarantee positivity of the numerical solutions. The discretized terms read

$$B_k \approx \begin{cases} 2B_k{}^n_{j,l} \frac{k^{n+1}_{j,l}}{k^n_{j,l}} - B_k{}^n_{j,l} & \text{if} \quad B < 0 \\ B_k{}^n_{j,l} & \text{if} \quad B \geq 0 \end{cases} \tag{7.61}$$

$$\varepsilon^n_{j,l} \approx \begin{cases} 2\varepsilon^n_{j,l} \frac{k^{n+1}_{j,l}}{k^n_{j,l}} - \varepsilon^n_{j,l} & \text{if} \quad k - \varepsilon \\ \frac{k^{n+1}_{j,l}}{\tau^n_{j,l}} & \text{if} \quad k - \tau \end{cases} \tag{7.62}$$

The production in the ε equation is reformed by substituting Equation (7.53) in Equation (7.48).

$$P_\varepsilon = \frac{c_{1\varepsilon} c_\mu}{\nu_V} k^n_{j,l} P_k \tag{7.63}$$

The discretization of the buoyancy term for the ε equation is similar to that of k equation: ensuring positivity by using Patankar trick as

$$B_\varepsilon = c_{3\varepsilon} c_\mu B_k k^n_{j,l} \tag{7.64}$$

collecting all above-mentioned terms, in their discretized forms, leads to tridiagonal matrices for k and ε in the form of

$$a_l k^{n+1}_{j,l-1} + b_l k^{n+1}_{j,l} + c_l k^{n+1}_{j,l+1} = d_l \tag{7.65}$$

and

$$a_l \varepsilon^{n+1}_{j,l-1} + b_l \varepsilon^{n+1}_{j,l} + c_l \varepsilon^{n+1}_{j,l+1} = d_l \tag{7.66}$$

which are solved by Thomas algorithm, with the following boundary conditions. The boundary conditions use the bed friction velocity, u_{*_b}, and surface friction velocity, u_{*_t} (caused by wind).

$$k_{j,0} = \frac{u^2_{*_b}}{\sqrt{c_\mu}}, \quad k_{j,\mathcal{M}(j)} = \frac{u^2_{*_t}}{\sqrt{c_\mu}} \tag{7.67}$$

$$\left.\frac{\partial \varepsilon}{\partial z}\right|_{j,1/2} = -\frac{|u_{*_b}|^3}{\kappa \left(\frac{1}{2}\Delta z_{j,1} + \mu z_0\right)^2}, \quad \left.\frac{\partial \varepsilon}{\partial z}\right|_{j,\mathcal{M}(j)-1/2} = \frac{|u_{*_t}|^3}{\kappa \left(\frac{1}{2}\Delta z_{j,\mathcal{M}(j)}\right)^2} \tag{7.68}$$

where μ takes values of 1 (based on theoretical analysis) or 9 (based on measurements).

Algorithm (56) and Algorithm (57) represent algorithms for k and ε equations, respectively.

Algorithm 56 update_verticalprofiles: compute the kinetic energy k

for $l = 1$ **to** $\mathcal{M}(j)$ **do**
$D_1 = D_{kj,l-1/2}/\left(\Delta z_{j,l+1/2}\Delta z_{j,l}\right)$, $D_2 = D_{kj,l+1/2}/\left(\Delta z_{j,l+1/2}\Delta z_{j,l+1}\right)$
$\nu'_V = \min\left(\nu_v, \nu_{min}\right)$
$\rho_{z1} = \frac{\rho_{R(j),l+1}-\rho_{R(j),l}}{\Delta z_{R(j),l+1/2}}$, $\rho_{z2} = \frac{\rho_{L(j),l+1}-\rho_{L(j),l}}{\Delta z_{L(j),l+1/2}}$
$\rho_z = \frac{1}{2}\left(\rho_{z1} + \rho_{z2}\right)$
$B_k = -g\frac{\nu'_V}{\rho\sigma_\rho}\rho_z$
$P_k = \nu'_V\left(\frac{u_{j,l+1}-u_{j,l}}{\Delta z_{j,l+1/2}}\right)^2 + \nu'_V\left(\frac{v_{j,l+1}-v_{j,l}}{\Delta z_{j,l+1/2}}\right)^2$
if $k-\varepsilon$ **then**
$S_k = \varepsilon^n_{j,l}/k^n_{l,j}$
$\beta = 2$
else if $k-\tau$ **then**
$S_k = 1/\tau^n_{j,l}$
$\beta = 1$
end if
$a_l = -D_1\theta - \max\left(\omega_1, 0\right)/\Delta z_{j,l+1/2}$
$b_l = 1/\Delta t + (D_1 + D_2)\theta + 2\max(B_k, 0)/k^n_{j,l} + \beta S_k$
$\quad + \max\left(\omega_2, 0\right)/\Delta z_{j,l+1/2} - \min\left(\omega_1, 0\right)/\Delta z_{j,l+1/2}$
$c_l = -D_2\theta + \min\left(\omega_2, 0\right)/\Delta z_{j,l+1/2}$
$d_l = k^n_{j,l}/\Delta t - D_2\left(k^n_{j,l} - k^n_{j,l+1}\right)(1-\theta) + D_1\left(k^n_{j,l-1} - k^n_{j,l}\right)(1-\theta)$
$\quad + \max(B_k, 0) - \min(B_k, 0) + P_k + (\beta - 1)S_k k^n_{j,l} - \mathcal{A}_{kj,l}$
end for
Calculate u_{*_b} by Algorithm (49)
$a_0 = 0, \quad b_0 = 1, \quad c_0 = 0, \quad d_0 = u^2_{*_b}/\sqrt{c_\mu}$
$a_{\mathcal{M}(j)} = 0, \quad b_{\mathcal{M}(j)} = 1, \quad c_{\mathcal{M}(j)} = 0, \quad d_{\mathcal{M}(j)} = u^2_{*_t}/\sqrt{c_\mu}$
Solve $a_l k_{l-1} + b_l k_l + c_l k_{l+1} = d_l$ by Algorithm (51)

Algorithm 57 update_verticalprofiles: compute the energy dissipation ε

for $l = 1$ **to** $\mathcal{M}(j)$ **do**
$D_1 = D_{\varepsilon j,l-1/2}/\left(\Delta z_{j,l+1/2}\Delta z_{j,l}\right)$, $D_2 = D_{\varepsilon j,l+1/2}/\left(\Delta z_{j,l+1/2}\Delta z_{j,l+1}\right)$
$\nu'_V = \min\left(\nu_v, \nu_{min}\right)$
$P_\varepsilon = c_{1\varepsilon}c_\mu k^n_{j,l}P_k/\nu'_V$
$B_\varepsilon = c_\mu c_{1\varepsilon}k^{n+1}_{j,l}\min(B_k, 0)$
$S_\varepsilon = c_{2\varepsilon}\varepsilon^n_{j,l}/k^{n+1}_{j,l}$
$a_l = -D_1\theta - \max\left(\omega_1, 0\right)/\Delta z_{j,l+1/2}$
$b_l = 1/\Delta t + (D_1 + D_2)\theta + 2S_\varepsilon + \max\left(\omega_2, 0\right)/\Delta z_{j,l+1/2} - \min\left(\omega_1, 0\right)/\Delta z_{j,l+1/2}$
$c_l = -D_2\theta + \min\left(\omega_2, 0\right)/\Delta z_{j,l+1/2}$
$d_l = \varepsilon^n_{j,l}/\Delta t - D_2\left(\varepsilon^n_{j,l} - \varepsilon^n_{j,l+1}\right)(1-\theta) + D_1\left(\varepsilon^n_{j,l-1} - \varepsilon^n_{j,l}\right)(1-\theta)$
$\quad - B_\varepsilon + P_\varepsilon + S_\varepsilon\varepsilon^n_{j,l} - \mathcal{A}_{\varepsilon j,l}$
end for
$a_0 = 0, \quad b_0 = 1, \quad c_0 = -1, \quad d_0 = \Delta z_{j,1}\max(u_{*_b}, 0)^3/(\kappa\Delta(\frac{1}{2}z_{j,1} + \mu z_0)^2)$
$a_{\mathcal{M}(j)} = -1, \quad b_{\mathcal{M}(j)} = 1, \quad c_{\mathcal{M}(j)} = 0, \quad d_{\mathcal{M}(j)} = 4\left|u_{*_t}\right|^3/(\kappa\Delta z_{j,\mathcal{M}(j)})$
Solve $a_l\varepsilon_{l-1} + b_l\varepsilon_l + c_l\varepsilon_{l+1} = d_l$ by Algorithm (51)

7.7.4 k-τ turbulence model

The time-scale of turbulence, τ, is defined by

$$\tau = \frac{k}{\varepsilon} \tag{7.69}$$

The eddy viscosity then equals

$$\nu_V = c_\mu \frac{k^2}{\varepsilon} = c_\mu k \tau \tag{7.70}$$

where $c_\mu = 0.09$. The variable τ models a typical time-scale of turbulent eddies. The $k - \tau$ equations read

$$\frac{\partial k}{\partial t} + u\frac{\partial k}{\partial x} + v\frac{\partial k}{\partial y} + \omega\frac{\partial k}{\partial z} = \frac{\partial}{\partial z}\left(D_k \frac{\partial k}{\partial z}\right) + P_k + B_k - \frac{k}{\tau} \tag{7.71}$$

$$\begin{aligned}\frac{\partial \tau}{\partial t} + u\frac{\partial \tau}{\partial x} + v\frac{\partial \tau}{\partial y} + \omega\frac{\partial \tau}{\partial z} = \frac{\partial}{\partial z}\left(D_\tau \frac{\partial \tau}{\partial z}\right) + P_\tau + B_\tau - (1 - c_{2\varepsilon}) \\ + D_{k\tau} + D_{\tau\tau} + D_{kk}\end{aligned} \tag{7.72}$$

where

$$D_\tau = \frac{\nu_{mol}}{\sigma_{mol}} + \frac{\nu_V}{\sigma_\tau} \tag{7.73}$$

$$P_\tau = \frac{\tau}{k}\left(1 - c_{1\varepsilon}\right) P_k \tag{7.74}$$

$$B_\tau = \frac{\tau}{k}\left(1 - c_{3\varepsilon}\right) B_k \tag{7.75}$$

$$D_{k\tau} = \frac{1}{h^2}\frac{2}{k} D_\tau \frac{\partial \tau}{\partial \sigma}\frac{\partial k}{\partial \sigma} \tag{7.76}$$

$$D_{\tau\tau} = -\frac{1}{h^2}\frac{2}{\tau} D_\tau \frac{\partial \tau}{\partial \sigma}\frac{\partial \tau}{\partial \sigma} \tag{7.77}$$

$$D_{kk} = -\frac{1}{h^2}\frac{\tau}{k}\frac{\partial}{\partial \sigma}\left[\left(\frac{1}{\sigma_\varepsilon} - \frac{1}{\sigma_k}\right)\nu_V \frac{\partial k}{\partial \sigma}\right] \tag{7.78}$$

The signs of the coefficients are so that the production term P_τ actually acts as sink of τ. This can be explained by realizing that production of dissipation ε results in a faster dissipation of turbulent eddies and therefore a smaller time-scale τ of turbulence. Even though P_τ acts as a sink, it will still be called a production term because of the parallels with the ε-equation. Likewise the dissipation term ε_τ is a source of τ.

By substituting Equation (7.70) in Equation (7.74) for k, it changes to the following form

$$P_\tau = \frac{c_\mu}{\nu_t} c_{1\tau} P_k \tau^2 \tag{7.79}$$

where $c_{1\tau} = 1 - c_{1\varepsilon}$ is a constant. This equation is linearized by Picard method as

$$P_\tau = \frac{c_\mu}{\nu_t} c_{1\tau} P_k \tau^n \tau^{n+1} \tag{7.80}$$

Similar to the above expression, by substituting Equation (7.70) in Equation (7.75), the buoyancy term is changed to the following form

$$B_\tau = c_{3\tau}\frac{c_\mu}{\nu_t} B_k \tau^2 \tag{7.81}$$

where $c_{3\tau} = 1$ is for stable stratification and $c = -0.44$ is for unstable stratification. After linearization by the Picard method, it yields

$$B_\tau = c_{3\tau}\frac{c_\mu}{\nu_t}B_k\tau_{j,l}^n\tau_{j,l}^{n+1} \tag{7.82}$$

or

$$B_\tau = c_{3\tau}\frac{c_\mu}{\sigma_\tau}N^2\tau_{j,l}^n\tau_{j,l}^{n+1} \tag{7.83}$$

and

$$N^2 = -\frac{g}{\rho_0}\frac{\partial\rho}{\partial z} \tag{7.84}$$

The currently used values of $c_{3\tau}$ for stable and unstable stratification guarantee that $c_{3\tau}N^2$ is positive under all conditions, hence $c_{3\tau} < 0$ if $N^2 < 0$ and $c_{3\tau} > 0$ if $N^2 > 0$.

The diffusion terms $D_{\tau\tau}$ and $D_{k\tau}$ are simplified by neglecting ν from the viscosity, as it is small compared with the eddy viscosity. By inserting Equation (7.70) into the equations for $D_{\tau\tau}$ and $D_{k\tau}$, it leads to the following form of equations

$$D_{\tau\tau} = -\left(\frac{2c_\mu}{\sigma_\tau}k\frac{\partial\tau}{\partial z}\right)\frac{\partial\tau}{\partial z} \tag{7.85}$$

$$D_{k\tau} = \left(\frac{2c_\mu}{\sigma_\tau}\tau\frac{\partial k}{\partial z}\right)\frac{\partial\tau}{\partial z} \tag{7.86}$$

By summing Equation (7.85) and Equation (7.85) we have,

$$D_{k\tau} + D_{\tau\tau} = \frac{2c_\mu}{\sigma_\tau}\left(\tau\frac{\partial k}{\partial z} - k\frac{\partial\tau}{\partial z}\right)\frac{\partial\tau}{\partial z} = \mathcal{A}\frac{\partial\tau}{\partial z} \tag{7.87}$$

Equation (7.87) is an advection equation. This equation is discretized by means of first order upwind as follows.

$$D_{k\tau} + D_{\tau\tau} = \max(\mathcal{A},0)\frac{\tau_{j,l}^{n+1} - \tau_{j,l-1}^{n+1}}{\Delta z_{j,l}} + \min(\mathcal{A},0)\frac{\tau_{j,l+1}^{n+1} - \tau_{j,l}^{n+1}}{\Delta z_{j,l+1}} \tag{7.88}$$

where $\mathcal{A}$ is discretized as,

$$\begin{aligned}\mathcal{A} \approx \frac{c_\mu}{\sigma_\tau}\Bigg(&\frac{\tau_{j,l-1}+\tau_{j,l}}{2}\frac{k_{j,l}-k_{j,l-1}}{\Delta z_{j,l}} + \frac{\tau_{j,l+1}+\tau_{j,l}}{2}\frac{k_{j,l+1}-k_{j,l}}{\Delta z_{j,l+1}}\\ &-\frac{k_{j,l-1}+k_{j,l}}{2}\frac{\tau_{j,l}-\tau_{j,l-1}}{\Delta z_{j,l}} - \frac{k_{j,l+1}+k_{j,l}}{2}\frac{\tau_{j,l+1}-\tau_{j,l}}{\Delta z_{j,l+1}}\Bigg)\end{aligned} \tag{7.89}$$

In this analysis, the term of D_{kk} in Equation (7.73) is neglected.

The vertical diffusion term (the first term in the right hand side of Equation (7.73)) is discretized implicitly by means of θ method.

$$\begin{aligned}\frac{\partial}{\partial z}\left(D_\tau\frac{\partial\tau}{\partial z}\right) \approx &\frac{\theta D_{\tau 2}}{\Delta z_{j,l+1}\Delta z_{j,l+1/2}}\left(\tau_{j,l+1}^{n+1} - \tau_{j,l}^{n+1}\right) - \frac{\theta D_{\tau 1}}{\Delta z_{j,l}\Delta z_{j,l+1/2}}\left(\tau_{j,l}^{n+1} - \tau_{j,l-1}^{n+1}\right)\\ &+\frac{(1-\theta)\,D_{\tau 2}}{\Delta z_{j,l+1}\Delta z_{j,l+1/2}}\left(\tau_{j,l+1}^{n} - \tau_{j,l}^{n}\right) - \frac{(1-\theta)\,D_{\tau 1}}{\Delta z_{j,l}\Delta z_{j,l+1/2}}\left(\tau_{j,l}^{n} - \tau_{j,l-1}^{n}\right)\end{aligned} \tag{7.90}$$

Combining all equations together, it leads to the following equation for each computational cell

$$a_l\tau_{j,l-1}^{n+1} + b_l\tau_{j,l}^{n+1} + c_l\tau_{l+1}^{n+1} = d_l \tag{7.91}$$

Marching in the vertical direction for each j location, it leads to a tridiagonal matrix (for each j) which is solved by Thomas algorithm (see Algorithm (58)). The boundary conditions at the bed and water surface are applied as,

$$a_0 = 0, \quad b_1 = 1, \quad c_0 = 0, \quad d_0 = \frac{9\kappa z_0}{\max(u_{*_b}, 10^{-6})\sqrt{c_\mu}}$$
$$a_{\mathcal{M}(j)} = 0, \quad b_{\mathcal{M}(j)} = 1, \quad c_{\mathcal{M}(j)} = 0, \quad d_{\mathcal{M}(j)} = 0$$

Remark 7.7.1. The D_{kk} term is neglected because it may cause numerical instabilities. If k goes to zero, then it follows from the positivity of k that $\partial k/\partial z$ also goes to zero. However, numerically $\partial k/\partial z$ is not necessarily small when k goes to zero, especially on coarse grids, and this term can grow to infinity. It is expected that this term is generally rather small because the factor $1/\sigma_\varepsilon - 1/\sigma_k$ in front of the term is only 0.2 with the current parameter settings.

Remark 7.7.2. The k-τ turbulence model is applied for the first time in D-Flow FM and it still needs intensive verifications.

Algorithm 58 update_verticalprofiles: compute the energy dissipation τ

for $l = 1$ **to** $\mathcal{M}(j)$ **do**
$D_1 = D_{\varepsilon j,l-1/2}/\left(\Delta z_{j,l+1/2}\Delta z_{j,l}\right)$, $D_2 = D_{\varepsilon j,l+1/2}/\left(\Delta z_{j,l+1/2}\Delta z_{j,l+1}\right)$
$\nu'_V = \min\left(\nu_v, \nu_{min}\right)$
$P_\tau = \left(1 - c_{1\varepsilon}\right) c_\mu \tau_{j,l}^n P_k/\nu'_V$
$B_\tau = c_{3\tau} c_\mu B_k \tau_{j,l}^n$
$\alpha_1 = \left(k_{j,l-1}^n + k_{j,l}^n\right)\left(\tau_{j,l}^n - \tau_{j,l-1}^n\right)/\left(2\Delta z_{j,l}\right)$
$\alpha_2 = \left(k_{j,l+1}^n + k_{j,l}^n\right)\left(\tau_{j,l+1}^n - \tau_{j,l}^n\right)/\left(2\Delta z_{j,l+1}\right)$
$D_{\tau\tau} = c_\mu\left(\alpha_1 + \alpha_2\right)/\sigma_\tau$
$\beta_1 = \left(\tau_{j,l-1}^n + \tau_{j,l}^n\right)\left(k_{j,l}^n - k_{j,l-1}^n\right)/\left(2\Delta z_{j,l}\right)$
$\beta_2 = \left(\tau_{j,l+1}^n + \tau_{j,l}^n\right)\left(k_{j,l+1}^n - k_{j,l}^n\right)/\left(2\Delta z_{j,l+1}\right)$
$D_{k\tau} = c_\mu\left(\beta_1 + \beta_2\right)/\sigma_\tau$
$D = \left(D_{\tau\tau} - D_{k\tau}\right)/\Delta z_{j,l+1/2}$
$a_l = -D_1\theta - \max(D, 0) - \max\left(\omega_1, 0\right)/\Delta z_{j,l+1/2}$
$b_l = 1/\Delta t + (D_1 + D_2)\theta - B_\tau - P_\tau + \max(D, 0) - \min(D, 0)$
$\quad + \max\left(\omega_2, 0\right)/\Delta z_{j,l+1/2} - \min\left(\omega_1, 0\right)/\Delta z_{j,l+1/2}$
$c_l = -D_2\theta + \min(D, 0) + \min\left(\omega_2, 0\right)/\Delta z_{j,l+1/2}$
$d_l = \tau_{j,l}^n/\Delta t - D_2\left(\tau_{j,l}^n - \tau_{j,l+1}^n\right)(1-\theta) + D_1\left(\tau_{j,l-1}^n - \tau_{j,l}^n\right)(1-\theta)$
$\quad -(1 - c_{2\varepsilon}) - \mathcal{A}_{\varepsilon j,l}$
end for
Calculate u_* by Algorithm (49)
$a_0 = 0, \quad b_1 = 1, \quad c_0 = 0, \quad d_0 = 9\kappa z_0/\left(\max(u_{*_b}, 10^{-6})\sqrt{c_\mu}\right)$
$a_{\mathcal{M}(j)} = 0, \quad b_{\mathcal{M}(j)} = 1, \quad c_{\mathcal{M}(j)} = 0, \quad d_{\mathcal{M}(j)} = 0$
Solve $a_l\tau_{l-1} + b_l\tau_l + c_l\tau_{l+1} = d_l$ by Algorithm (51)

7.8 Baroclinic pressure

Under the shallow-water assumption, the vertical momentum equation is reduced to a hydrostatic pressure equation. Vertical accelerations due to buoyancy effects and due to sudden variations in the bottom topography are not taken into account. So:

$$\frac{\partial P}{\partial z} = -g\rho \tag{7.92}$$

After integration between two successive layers, in the vertical direction, the hydrostatic pressure is

$$P\left(z\right) = P_2 + g\int_0^{\Delta z_l} \rho(z)dz \tag{7.93}$$

where P_2 is the pressure on the upper layer level. The local density is related to the values of temperature and salinity by the equation of state. The density is assumed to change linearly at each flow cell, along the layer in the vertical direction. The density can be described as $\rho(z) = \rho_2 + \alpha z$, with $\alpha = \left(\rho_1 - \rho_2\right)/\Delta z_l$, where ρ_1 and ρ_2 are the densities on the top and bottom of the flow cell, respectively. After substitution of this relation in Equation (7.93) and integration , it gives,

$$P(z) = P_2 + g\left(\rho_2 \Delta z_l + \frac{1}{2}\alpha \Delta z_l^2\right) \tag{7.94}$$

The force on the flow is derived by integration of the pressure in the vertical direction.

$$F\left(z\right) = \int_0^{\Delta z_l} P(z)dz \tag{7.95}$$

Subsituting Equation (7.95) in Equation (7.94) gives,

$$F\left(z\right) = P_2 \Delta z_l + \frac{1}{2}g\rho_2 \Delta z_l^2 + \frac{1}{6}g\alpha \Delta z_l^3 \tag{7.96}$$

Remark 7.8.1. For integration of Equation (7.94) and Equation (7.95), the vertical coordinate is locally set starting from the top of the cell (on the layer) toward downward.

The merely of this method is to calculate $F(z)$ at the cells between the layers, and integrate the force from the water surface to the cell levels. Once it is done, the forces around the control volume are integrate. This method is applied in Algorithm (59).

Algorithm 59 addbaroc2: compute the baroclinic pressure along σ-layers

First, extrapolate the density on the water surface

$$\begin{aligned}
\beta_L =& \frac{\Delta z_{L(j),\mathcal{M}(j)-1}}{\Delta z_{L(j),\mathcal{M}(j)} + \Delta z_{L(j),\mathcal{M}(j)-1}} \\
\beta_R =& \frac{\Delta z_{R(j),\mathcal{M}(j)-1}}{\Delta z_{R(j),\mathcal{M}(j)} + \Delta z_{R(j),\mathcal{M}(j)-1}} \\
\tilde{\rho}_{L,\mathcal{M}(j)} =& (1+\beta_L)\,\rho_{L(j),\mathcal{M}(j)} - \beta_L \rho_{L(j),\mathcal{M}(j)-1} \\
\tilde{\rho}_{R,\mathcal{M}(j)} =& (1+\beta_R)\,\rho_{R(j),\mathcal{M}(j)} - \beta_R \rho_{R(j),\mathcal{M}(j)-1}
\end{aligned}$$

for $l = \mathcal{M}(j) - 1$ to 1 step -1 **do**

$$\begin{aligned}
\beta_L =& \frac{\Delta z_{L(j),l}}{\Delta z_{L(j),l} + \Delta z_{L(j),l+1}} \\
\beta_R =& \frac{\Delta z_{R(j),l}}{\Delta z_{R(j),l} + \Delta z_{R(j),l+1}} \\
\tilde{\rho}_{L,l} =& \beta_L \rho_{L(j),l} + (1-\beta_L)\,\rho_{L(j),l+1} \\
\tilde{\rho}_{R,l} =& \beta_R \rho_{R(j),l} + (1-\beta_R)\,\rho_{R(j),l+1}
\end{aligned}$$

end for

for $l = \mathcal{M}(j)$ to 1 step -1 **do**

$$\begin{aligned}
\alpha_L =& \frac{\tilde{\rho}_{L,l-1} - \tilde{\rho}_{L,l}}{\Delta z_{L(j),l}} \quad , \quad \alpha_R = \frac{\tilde{\rho}_{R,l-1} - \tilde{\rho}_{R,l}}{\Delta z_{R(j),l}} \\
P_L =& g \sum_{l'=l}^{\mathcal{M}(j)} \frac{1}{2} \tilde{\rho}_{L,l'} \Delta z_{L(j),l'} + \frac{1}{6} \alpha_L \Delta z^2_{L(j),l'} \\
P_R =& g \sum_{l'=l}^{\mathcal{M}(j)} \frac{1}{2} \tilde{\rho}_{R,l'} \Delta z_{R(j),l'} + \frac{1}{6} \alpha_R \Delta z^2_{R(j),l'} \\
G_L =& P_L \Delta z_{L(j),l} + \frac{1}{2} \tilde{\rho}_{L,l} \Delta z^2_{L(j),l} + \frac{1}{6} \alpha_L \Delta z^3_{L(j),l} \\
G_R =& P_R \Delta z_{R(j),l} + \frac{1}{2} \tilde{\rho}_{R,l} \Delta z^2_{R(j),l} + \frac{1}{6} \alpha_R \Delta z^3_{R(j),l} \\
\tilde{\rho}_{j,l} =& \frac{1}{4} \left(\tilde{\rho}_{L(j),l} + \tilde{\rho}_{L(j),l-1} + \tilde{\rho}_{R(j),l} + \tilde{\rho}_{R(j),l-1} \right) \\
G_B =& \left(z_{L(j),l-1} - z_{R(j),l-1} \right) \sum_{l'=l}^{\mathcal{M}(j)} \tilde{\rho}_{j,l'} \Delta z_{j,l'} \\
G_T =& \left(z_{L(j),l} - z_{R(j),l} \right) \sum_{l'=l+1}^{\mathcal{M}(j)} \tilde{\rho}_{j,l'} \Delta z_{j,l'} \\
\delta P_l =& G_L - G_R + G_B - G_T \\
V_\rho =& \frac{1}{4} \left[\Delta z_{L(j),l} \left(\tilde{\rho}_{L,l} + \tilde{\rho}_{L,l-1} \right) + \Delta z_{R(j),l} \left(\tilde{\rho}_{R,l} + \tilde{\rho}_{R,l-1} \right) \right] \\
\Delta P =& \delta P_l / V_\rho
\end{aligned}$$

end for

7.9 Artificial mixing due to σ-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

$$F_i = D_H \frac{\partial c}{\partial x_i}, \quad i = 1, 2; \qquad F_3 = D_V \frac{\partial c}{\partial x_3} \tag{7.97}$$

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-coordinates 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)

7.9.1 A finite volume method for a σ-grid

Applying the Gauss theorem to the transport equation yields

$$\frac{\partial}{\partial t} \int_v c \, dv + \oint_s \boldsymbol{F} \cdot \boldsymbol{n} \, ds = 0 \tag{7.98}$$

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 one-dimensional heat equation (i.e. a transport equation without advection in one dimension)

$$\frac{\partial c\,(x, z, t)}{\partial t} - \frac{\partial}{\partial x} \left(D_H \frac{\partial c\,(x, z, t)}{\partial x} \right) = 0 \tag{7.99}$$

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 $f_{i+\frac{1}{2},l+\frac{1}{2}}, l = 0, \ldots, 2K$ (K is the σ-layer number), are defined according to

$$f_{i+\frac{1}{2},l+\frac{1}{2}} = \begin{cases} D_H \min\left(\Delta_m c, \Delta_n c\right) \frac{z_{i+\frac{1}{2},l+1} - z_{i+\frac{1}{2},l}}{x_{i+1} - x_i}, & \text{if } \Delta_m c > 0 \wedge \Delta_n c > 0 \\ D_H \max\left(\Delta_m c, \Delta_n c\right) \frac{z_{i+\frac{1}{2},l+1} - z_{i+\frac{1}{2},l}}{x_{i+1} - x_i}, & \text{if } \Delta_m c \leq 0 \wedge \Delta_n c \leq 0 \\ 0, & \text{if } \Delta_m c \Delta_n c < 0 \end{cases} \tag{7.100}$$

The differences $\Delta_{m/n}c = \Delta_{m/n}c_{i+\frac{1}{2},l+\frac{1}{2}}$ are given by ($l = 0, \ldots, 2K$)

$$f_{i+\frac{1}{2},l+\frac{1}{2}} = \begin{cases} \Delta_m c_{i+\frac{1}{2},l+\frac{1}{2}} = c_{i+1}\left(z_{i,m(l)} - c_{i,m(l)}\right) \\ \Delta_n c_{i+\frac{1}{2},l+\frac{1}{2}} = c_{i+1,n(l)} - c_i\left(z_{i+1,n(l)}\right) \end{cases} \tag{7.101}$$

where $c_i\left(z\right)$ is a simple linear interpolation formula given by

$$c_i\left(z\right) = \begin{cases} c_{i,1}, & \text{if } z \leq z_{i,1} \\ \frac{z-z_{i,k}}{z_{i,k+1}-z_{i,k}} c_{i,k+1} + \frac{z_{i,k+1}-z}{z_{i,k+1}-z_{i,k}} c_{i,k}, & \text{if } z_{i,k} < z \leq z_{i,k+1} \\ c_{i,K}, & \text{if } z \geq z_{i,K} \end{cases} \tag{7.102}$$

◇ Step 2
In this step the diffusive fluxes are added to the appropriate control volumes according to

$$V_{i,k}^{n+1} c_{i,k}^{n+1} = V_{i,k}^{n} c_{i,k}^{n} - \Delta t \sum_{\forall l | m(l)=k} f_{i+\frac{1}{2},l+\frac{1}{2}}^{n} + \Delta t \sum_{\forall l | n(l)=k} f_{i-\frac{1}{2},l+\frac{1}{2}}^{n} \tag{7.103}$$

where n is the time index, $t = n\Delta t$ and V^n denotes the size of the control volume. The absence of advection implies $V^n = V^{n+1}$

7.9.2 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 coordinates is given by

$$p\left(x,z\right) = \int_z^{\zeta} \rho\left(x,z',t\right) g dz' \tag{7.104}$$

From the Leibniz rule it follows that $\partial p/\partial x$ is given by

$$\frac{\partial p}{\partial x} = \frac{\partial}{\partial x} \int_z^{\zeta(x)} \rho\left(x,z'\right) g dz' = \int_z^{\zeta(x)} g \frac{\partial}{\partial x} \rho\left(x,z'\right) dz' + g\rho\left(\zeta\right) \frac{\partial \zeta}{\partial x} \tag{7.105}$$

The relation between the density ρ 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 (7.105) is replaced by a summation over the intervals which are in the water column above the velocity point with vertical co-ordinate z.

$$\begin{aligned} \left(\frac{\partial p}{\partial x}\right)\left(x_{i+\frac{1}{2},z}\right) =& g \sum_{l=K+1}^{2K+1} \left[\left(\frac{\partial \rho}{\partial s}\right)\frac{f\left(s\right)}{D_H} + \left(\frac{\partial \rho}{\partial T}\right)\frac{f\left(T\right)}{D_H}\right]_{i+\frac{1}{2},l+\frac{1}{2}} \\ &+ g\frac{z_{i+\frac{1}{2},k+1} - z}{z_{i+\frac{1}{2},k+1} - z_{i+\frac{1}{2},k}} \left[\left(\frac{\partial \rho}{\partial s}\right)\frac{f\left(s\right)}{D_H} + \left(\frac{\partial \rho}{\partial T}\right)\frac{f\left(T\right)}{D_H}\right]_{i+\frac{1}{2},k+\frac{1}{2}} \\ &+ g\rho\frac{\zeta_{i+1} - \zeta i}{\Delta x} \end{aligned} \tag{7.106}$$

where $k = \max\left(l | z_{i+\frac{1}{2},l} < z\right)$

8 Parallelization

This chapter elaborates on the parallelization of D-Flow FM, which enables the simulation on 1D and 2D network. The sequential time loop is described in section 6.3. 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 modelling in all subdomains is identical, has the same time-step, et cetera. Note this this requirement compromises our aim for autonomous subdomain modelling.

8.1.1 Ghost cells

Keeping our design choices in mind, it is apparent from Equation (6.25) to Equation (6.110) 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, Equation (6.112) *anywhere* in our subdomains. The stencil used for computing momentum advection and diffusion is depicted in Figure 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 Figure 8.2.

The question remains how many ghost cells need to be supplied. The stencil for the momentum advection and diffusion is depicted in Figure 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 Figure 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,

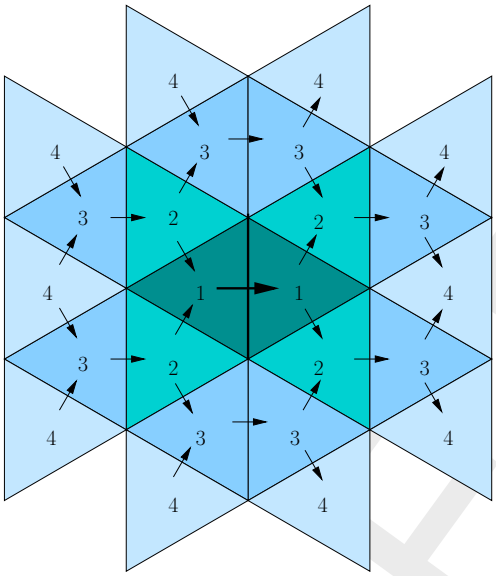


Figure 8.1: *Stencil for momentum advection and diffusion; the numbers indicate the level of the neighboring cells*

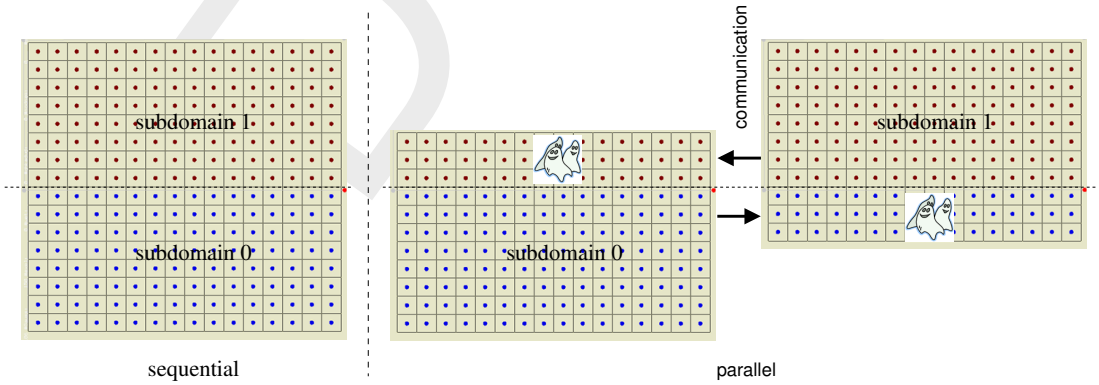


Figure 8.2: *Ghost cells*

Table 8.1: *METIS settings*

parameter/option	value	meaning
routine	METIS_PartGraphRecursive or METIS_PartGraphKway	mesh partitioning method
NITER	100	Number of iterations for the refinement algorithms at each stage of the uncoarsening process
UFACTOR	1.001	allowed load imbalance
CONTIG	0 or 1	enforce contiguous subdomains (1) or not (0), only available when using K-way method

- ◇ one additional level for a higher order cell-centered (collocated) discretization, and
- ◇ 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 Figure 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 Equation (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 with METIS

The METIS software package,see Karypis (2013), is used for partitioning the mesh. A dual graph of the mesh is firstly generated, and then partitioned. 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. Two partition methods are available in METIS: Recursive Bisection (default method in D-Flow FM) and multilevle K-way. The latter enables to enforce contiguous subdomains, and it is observed that this comes at the cost of a reduced homogeneous distribution of cells over the subdomains, however.

The non-default METIS settings employed are listed in Table 8.1.

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 partition mesh files. There, the cell coloring is also written and then read during the initialization of the parallel computation.

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.

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 (60).

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 60 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 (61). The other ghost types are updated similarly.

8.1.4 Parallel computations

In the parallel run, each task will

Algorithm 61 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

- ◇ prepare the subdomain model, i.e.
  - □ read the subdomain mesh,
  - □ read the boundary conditions,
  - □ read external forcings,
  - □ et cetera,
- ◇ initialize the parallel communication, Algorithm (60),
- ◇ perform the time stepping, as in Algorithm (22),
- ◇ 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, all other information equals its sequential counterpart.

The parallel time-step is shown in Algorithm (62). The parallel extension of Algorithm (21) is trivial and listed in Algorithm (63). The parallel solver for the water-level equation Algorithm (23) 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 (62).

The parallel extensions of the following will remain unmentioned:

- ◇ discharge boundary conditions,
- ◇ cross sections and observation stations (for post-processing).

8.1.5 Parallel Krylov solver

The unknown water levels $k \in \mathcal{K}$ in Equation (6.120) are solved in the same manner as in case of the sequential computations, Algorithm (23), except for the solver itself, which now is a parallel Krylov solver. We have two solvers available:

1. the parallelized version of the sequential algorithm (Algorithm (24)), and
2. 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

$$\begin{pmatrix} A^{[0,0]} & \dots & A^{[0,N-1]} \\ \vdots & \ddots & \vdots \\ A^{[N-1,0]} & \dots & A^{[N-1,N-1]} \end{pmatrix} \begin{pmatrix} \boldsymbol{s}^{[0]} \\ \vdots \\ \boldsymbol{s}^{[N-1]} \end{pmatrix} = \begin{pmatrix} \boldsymbol{d}^{[0]} \\ \vdots \\ \boldsymbol{d}^{[N-1]} \end{pmatrix}, \tag{8.10}$$

Algorithm 62 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 (14)
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 (15)
determine the set of water-levels that need to be solved, Algorithm (17)
$i = 0$
$\zeta_k^{n+1(0)} = \zeta_k^n$
while $\left(\max_k \left|\zeta_k^{n+1(i)} - \zeta_k^{n+1(i-1)}\right| > \varepsilon \ \wedge$ 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 (18)
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 (19) 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 (20)
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 (63)

Algorithm 63 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

where the superscript $[id]$ indicates the domain number. A matrix-vector multiplication can be 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 Equation (6.131), however for the internal unknowns only

$$A^{[id,id]}\boldsymbol{s}^{[id]} = \begin{pmatrix} \vdots \\ B_{rk}^{[id]}\ s_k^{[id]} + \sum_{j \in \mathcal{J}^{[id]}(k) \setminus G_s^{[id]}} C_{rj}^{[id]}\ s_{O(k,j)}^{[id]} \\ \vdots \end{pmatrix} \tag{8.11}$$

and the off-diagonal contributions are computed by means of the ghost values $\mathcal{G}_s^{[id]}$

$$\sum_{jd \neq id} A^{[id,jd]}\boldsymbol{s}^{[jd]} = \begin{pmatrix} \vdots \\ \sum_{j \in \mathcal{J}^{[id]}(k) \cap G_s^{[id]}} C_{rj}^{[id]}\ s_{O(k,j)}^{[id]} \\ \vdots \end{pmatrix}, \tag{8.12}$$

provided that the ghost values $G_s^{[id]}$ are up-to-date .

Remark 8.1.4. Equation (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 (24), as shown in Algorithm (64), 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 modelling-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 (62),
- ◇ 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.

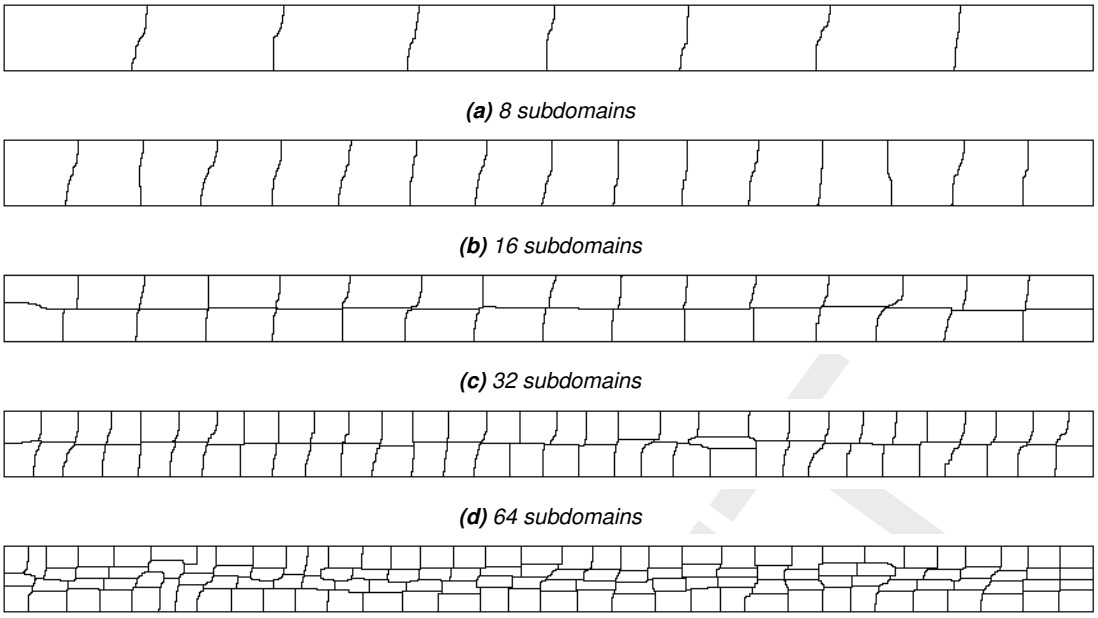
We expect that the non-communication times will show nearly linear scalability and foresee that the communication times behave much worse. Furthermore, if the precondition becomes less effective when the number of subdomains increases, the number of iterations will increase.

The speed-up factor f can now be defined as:

$$f_k(N) = \frac{T_{\text{time-step}_k}\big|_N}{T_{\text{time-step}_k}\big|_{ref}} N, \tag{8.15}$$

Algorithm 64 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 $\varepsilon = \|\boldsymbol{r}^{(0)}\|_\infty$
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 $\left\langle \boldsymbol{r}^{(0)}, \boldsymbol{z}_r^{(0)} \right\rangle$
reduce inner product $\left\langle \boldsymbol{r}^{(0)}, \boldsymbol{z}_r^{(0)} \right\rangle$
reduce maximum error $\varepsilon = \|\boldsymbol{r}^{(0)}\|_\infty$
$i = 0$
while $\varepsilon >$ tol **do**
  update ghost values p_k, $k \in \mathcal{G}_s$
  compute $A\boldsymbol{p}^{(i)}$
  compute $\left\langle \boldsymbol{p}^{(i)}, A\boldsymbol{p}^{(i)} \right\rangle$
  reduce $\left\langle \boldsymbol{p}^{(i)}, A\boldsymbol{p}^{(i)} \right\rangle$
  $\alpha^{(i)} = \dfrac{\left\langle \boldsymbol{r}^{(i)}, \boldsymbol{z}_r^{(i)} \right\rangle}{\left\langle \boldsymbol{p}^{(i)}, A\boldsymbol{p}^{(i)} \right\rangle}$
  $\boldsymbol{s}^{(i+1)} = \boldsymbol{s}^{(i)} + \alpha^{(i)}\boldsymbol{p}^{(i)}$
  $\boldsymbol{r}^{(i+1)} = \boldsymbol{r}^{(i)} - \alpha^{(i)}A\boldsymbol{p}^{(i)}$
  compute maximum error $\varepsilon = \|\boldsymbol{r}^{(i+1)}\|_\infty$ $\varepsilon = \|\boldsymbol{r}^{(0)}\|_\infty$
  reduce maximum error $\varepsilon = \|\boldsymbol{r}^{(0)}\|_\infty$
  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 $\varepsilon >$ tol **then**
    compute $\left\langle \boldsymbol{r}^{(i+1)}, \boldsymbol{z}_r^{(i+1)} \right\rangle$
    reduce $\left\langle \boldsymbol{r}^{(i+1)}, \boldsymbol{z}_r^{(i+1)} \right\rangle$
    $\beta^{(i+1)} = \dfrac{\left\langle \boldsymbol{r}^{(i+1)}, \boldsymbol{z}_r^{(i+1)} \right\rangle}{\left\langle \boldsymbol{r}^{(i)}, \boldsymbol{z}_r^{(i)} \right\rangle}$
    $\boldsymbol{p}^{(i+1)} = \boldsymbol{z}_r^{(i+1)} + \beta^{(i+1)}\boldsymbol{p}^{(i)}$
    $i = i + 1$
  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*

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 modelling time than others.

For our comparison, we will always compare time-step averaged computing times at the same modelling 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 Figure 8.3.

Timing results on the SURFsara Lisa cluster are presented in Table 8.2 and the corresponding speed-up factor in Figure 8.4. The results on the Deltares h4 cluster are shown in Table 8.3 and Figure 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

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

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]	$MPI_{non-sol}$ [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]	$MPI_{non-sol}$ [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

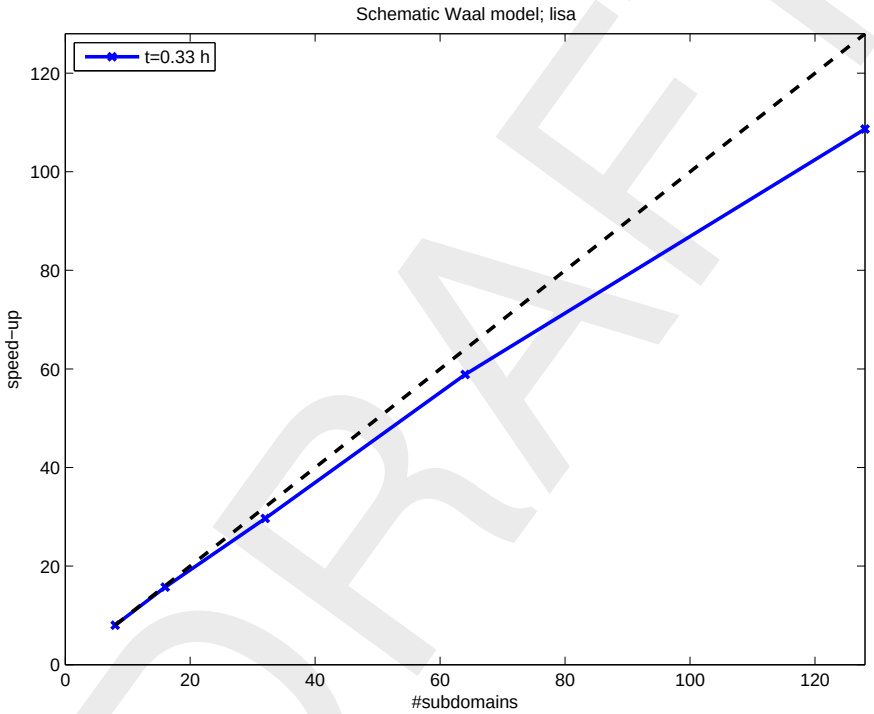


Figure 8.4: *Speed-up of the schematic Waal model; Lisa*

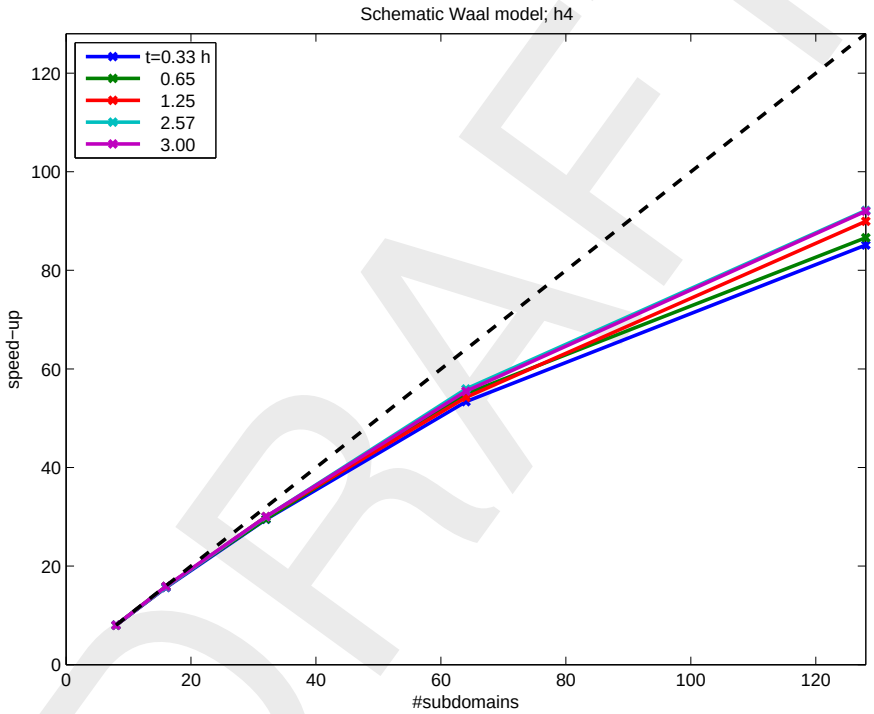


Figure 8.5: *Speed-up of the schematic Waal model; h4*

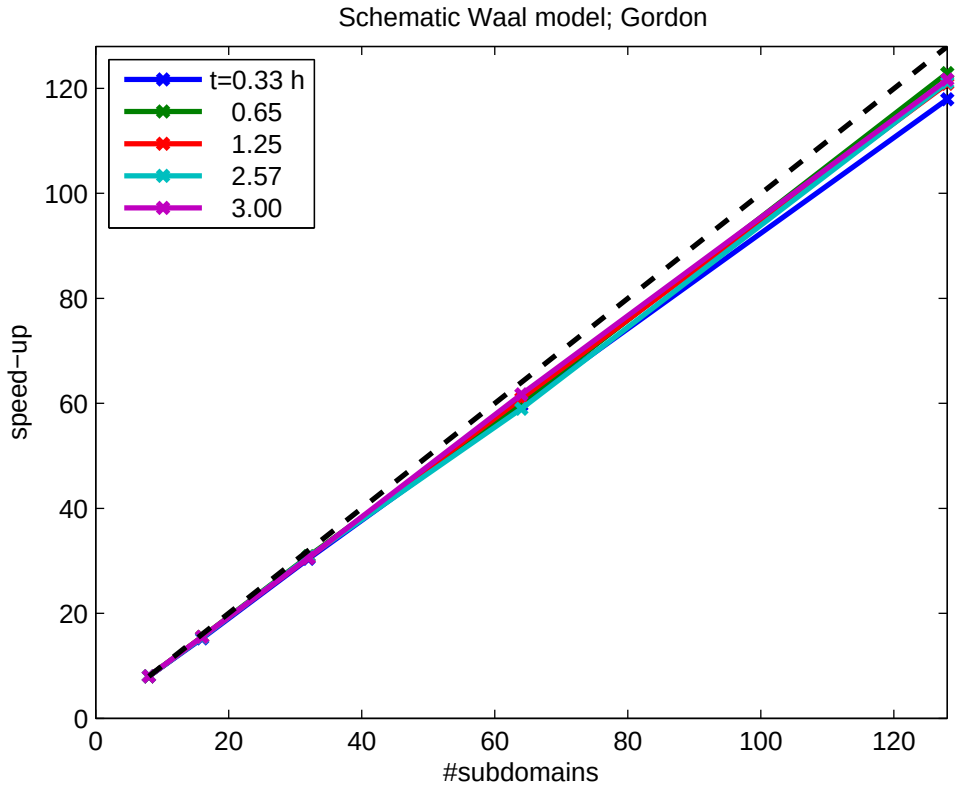


Figure 8.6: *Speed-up of the schematic Waal model; SDSC's Gordon; PETSc*

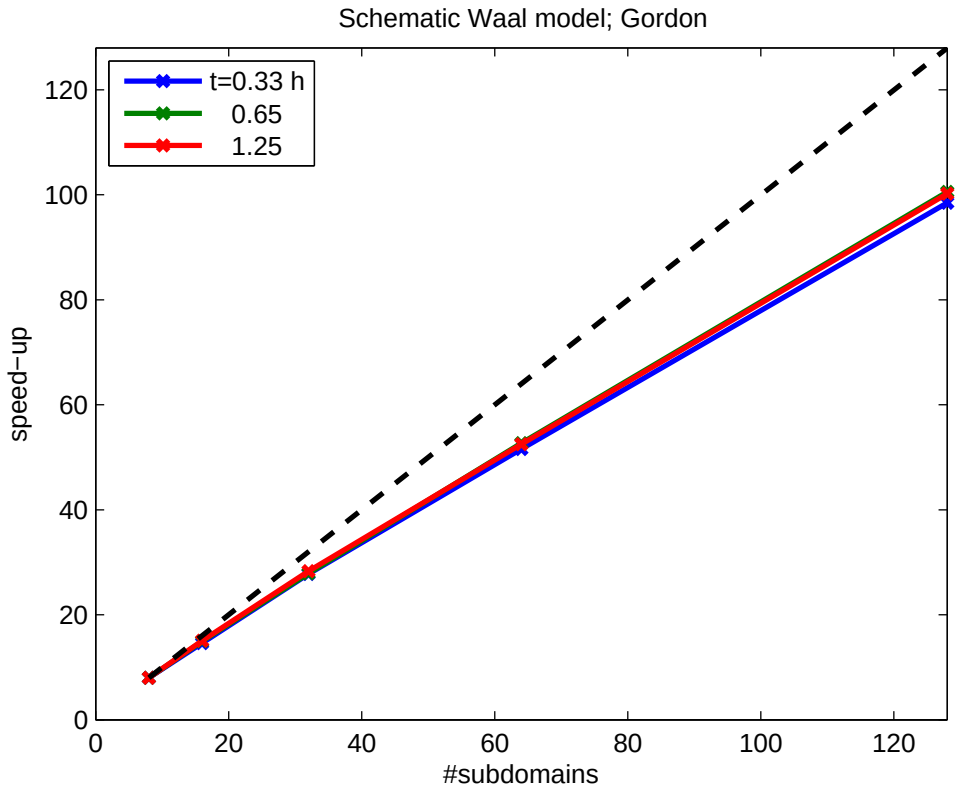
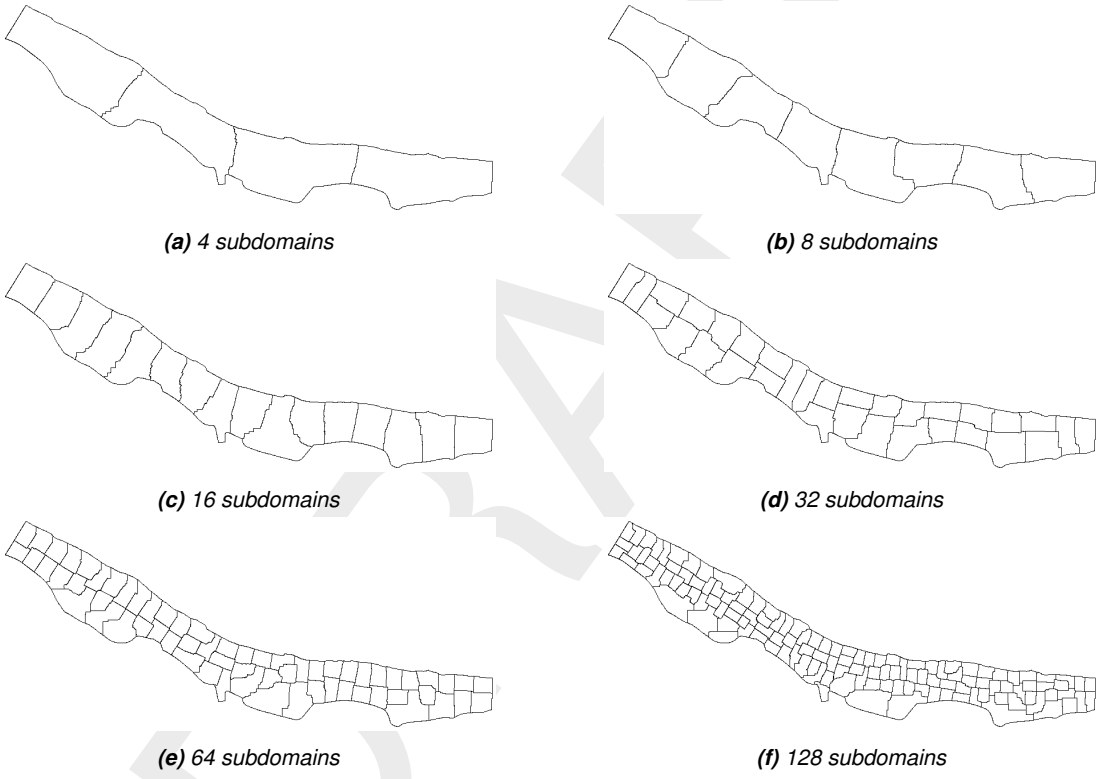


Figure 8.7: *Speed-up of the schematic Waal model; SDSC's Gordon; CG+MILUD*



(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*

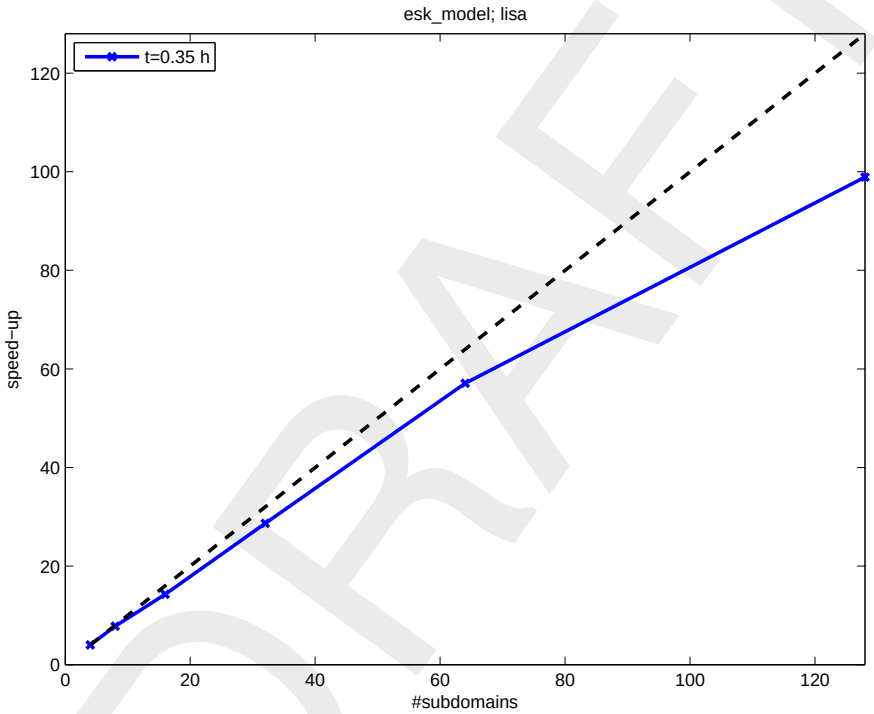


Figure 8.9: *Speed-up of the 'esk-model'; Lisa*

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

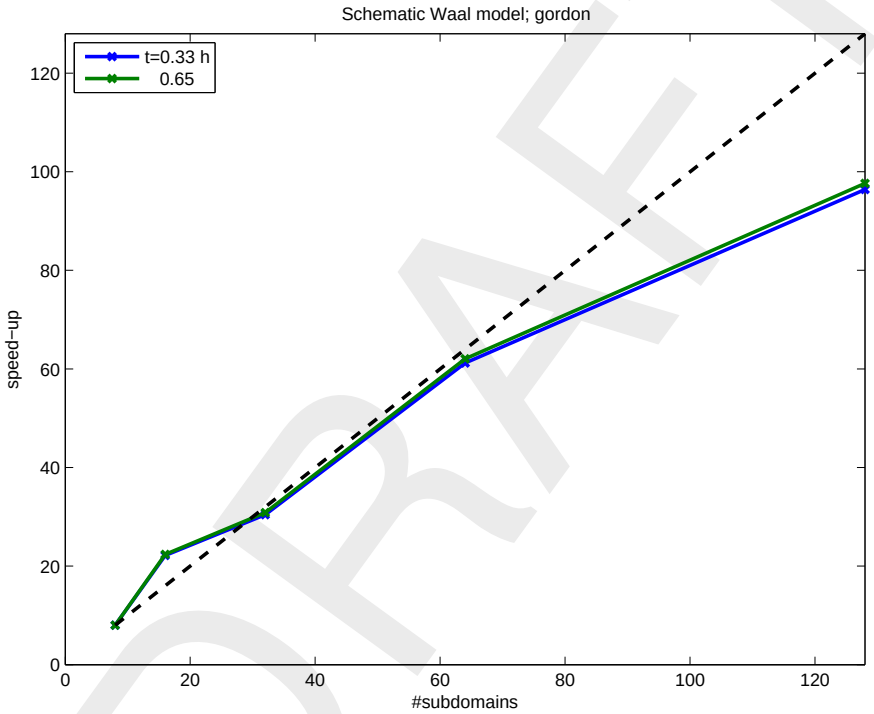


Figure 8.10: *Speed-up of the schematic Waal model; Gordon*

8.2.3 San Fransisco Delta-Bay model

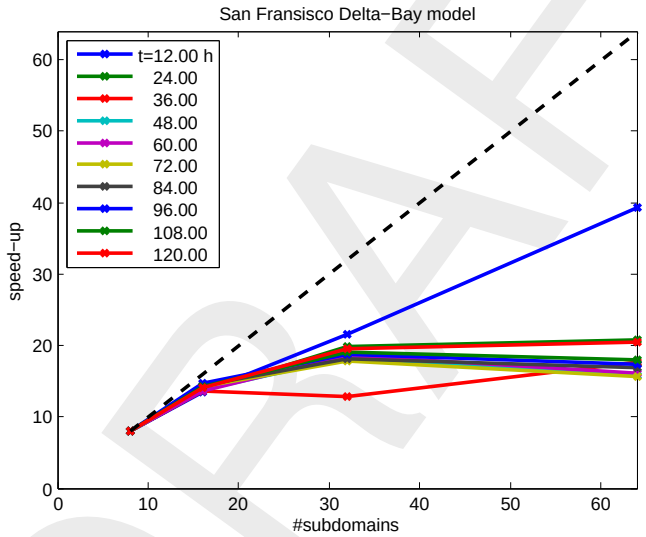


Figure 8.11: *Speed-up of the San Fransisco Delta-Bay model; Gordon*

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 Equation (8.16) and Equation (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 Equation (8.26) we obtain

$$\begin{aligned} \frac{\mathrm{d}u_j}{\mathrm{d}t} = &- g \frac{\bar{h}_j}{\tilde{h}_j} \frac{(s_{R(j)} - s_{L(j)})}{\Delta x_j} \\ &- \frac{1}{\tilde{h}_j} \left(\alpha_{Lj} \frac{1}{b_{AL(j)}} \sum_{l \in \mathcal{J}(L(j))} (\boldsymbol{u}_{\text{cupwind}(l)} \cdot \boldsymbol{n}_j - u_j) q_l 1_{l,k} + \right. \\ &\qquad \left. \alpha_{Rj} \frac{1}{b_{AR(j)}} \sum_{l \in \mathcal{J}(R(j))} (\boldsymbol{u}_{\text{cupwind}(l)} \cdot \boldsymbol{n}_j - u_j) q_l 1_{l,k} \right) \\ &+ \frac{1}{\tilde{h}_j} \left(\alpha_{Lj} \boldsymbol{\mathcal{D}}_{L(j)} + \alpha_{Rj} \boldsymbol{\mathcal{D}}_{R(j)}\right) \cdot \boldsymbol{n}_j + \frac{\tau_j}{\tilde{h}_j}. \end{aligned} \tag{8.33}$$

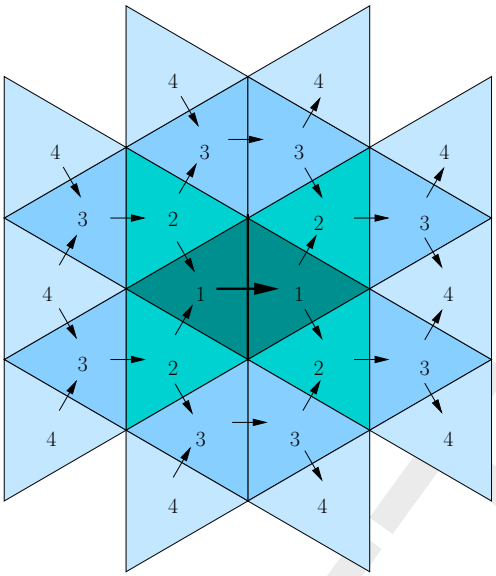


Figure 8.12: *Stencil for momentum advection and diffusion; the numbers indicate the level of the neighboring cells*

The advection and pressure-gradient terms are conform Kramer and Stelling. In D-Flow FM, however, the following form is implemented:

$$\begin{aligned}\frac{\mathrm{d}u_j}{\mathrm{d}t} =& -g\frac{(s_{R(j)}-s_{L(j)})}{\Delta x_j}\\ &-\frac{1}{V_{uj}}\Bigg(\alpha_{Lj}\sum_{l\in\mathcal{J}(L(j))}(\boldsymbol{u}_{\text{cupwind}(l)}\cdot\boldsymbol{n}_j-u_j)q_l 1_{l,k}+\\ &\qquad\alpha_{Rj}\sum_{l\in\mathcal{J}(R(j))}(\boldsymbol{u}_{\text{cupwind}(l)}\cdot\boldsymbol{n}_j-u_j)q_l 1_{l,k}\Bigg)\\ &+\frac{1}{h_j}\left(\alpha_{Lj}\boldsymbol{\mathcal{D}}_{L(j)}+\alpha_{Rj}\boldsymbol{\mathcal{D}}_{R(j)}\right)\cdot\boldsymbol{n}_j+\frac{\tau_j}{h_{Rj}},\end{aligned} \tag{8.34}$$

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.

A Analytical conveyance

A.1 Conveyance type 2

In conveyance type 2, we consider the flow is one dimensional, and we calculate the bed friction based on intersection perpendicular to the flow direction(Figure A.1). Parameter K_2 can be derived as follows.

$$\alpha_i = \frac{z_{i+1} - z_i}{y_{i+1} - y_i} = \frac{h_i - h_{i+1}}{y_{i+1} - y_i} \tag{A.1}$$

$$K_2 = \int_A C\sqrt{R}dA \ , \ or \ K_2 = \int_A C\sqrt{\frac{dA}{dP}}dA \tag{A.2}$$

Where R is hydraulic radius, C is Chézy coefficient, A is the cross sectional area and P is the wet area.

$$dA = h(y)dy \ \ , \ \ h(y) = h_i - \alpha_i \left(y - y_i\right) \tag{A.3}$$

$$dP = \sqrt{dy^2 + \left(\alpha_i dy\right)^2} = \sqrt{1 + \alpha_i^2}dy \tag{A.4}$$

$$K_2 = \int_{y_i}^{y_{i+1}} \frac{C}{(1+\alpha_i^2)^{\frac{1}{4}}} h(y)^{\frac{3}{2}} dy \ \ , \ \ (C = \frac{h(y)^{\frac{1}{6}}}{n}) \tag{A.5}$$

$$K_2 = \int_{y_i}^{y_{i+1}} \frac{1}{n(1+\alpha_i^2)^{\frac{1}{4}}} h(y)^{\frac{5}{3}} dy \tag{A.6}$$

$$K_2 = \int_{y_i}^{y_{i+1}} \frac{1}{n(1+\alpha_i^2)^{\frac{1}{4}}} (h_i - \alpha_i \left(y - y_i\right))^{\frac{5}{3}} dy \tag{A.7}$$

$$K_2 = \frac{-1}{n\alpha_i(1+\alpha_i^2)^{\frac{1}{4}}} \frac{3}{8} \left\{h_i - \alpha_i \left(y - y_i\right)\right\}^{\frac{8}{3}} \Big|_{y_i}^{y_{i+1}} \tag{A.8}$$

$$K_2 = \frac{1}{n\alpha_i(1+\alpha_i^2)^{\frac{1}{4}}} \frac{3}{8} \left(h_i^{\frac{8}{3}} - h_{i+1}^{\frac{8}{3}}\right) \tag{A.9}$$

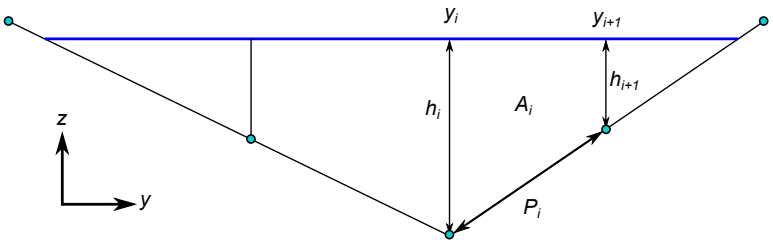


Figure A.1: *A schematic view of cross sectional bed bathemetry perpendicular to the flow direction.*

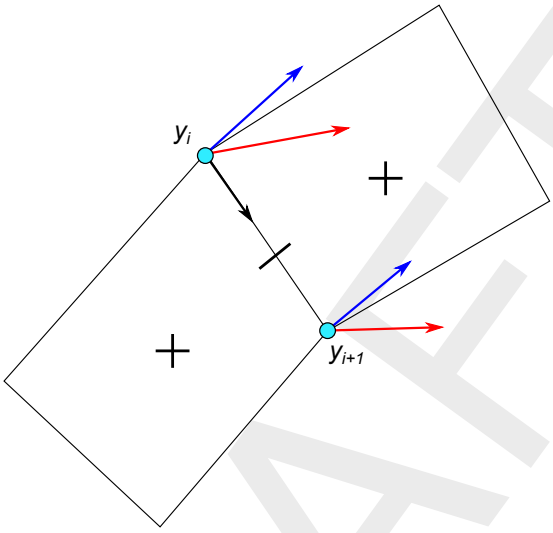


Figure A.2: *A schematic view of flow nodes and the velocity components in two-dimensional case.*

A.2 Conveyance type 3

Conveyance type 3 is similar to type 2, except it is extended to consider the second velocity component. Considering a two-dimensional case as illustrated in Figure A.2, we can derive K_3 as follows,

$$\frac{uU}{C^2R} = i,\ U = \frac{u}{\beta^2} \tag{A.10}$$

$$u_j = \beta C\sqrt{R_j}\sqrt{i},\ K_3 = \frac{\beta A_j R_j^{\frac{2}{3}}}{n} \tag{A.11}$$

$$\beta = \beta_i - \delta\left(y - y_i\right),\ \delta = \frac{\beta_i - \beta_{i+1}}{y_{i+1} - y_i} \tag{A.12}$$

If α_i and α_i' are the slopes in the streamwise and transverse directions respectively, we have,

$$K_3 = \int_{y_i}^{y_{i+1}} \frac{\beta_i - \delta\left(y - y_i\right)}{n\left(1 + \alpha_i^2 + \alpha_i'^2\right)^{\frac{1}{4}}}\left(h_i - \alpha_i\left(y - y_i\right)\right)^{\frac{5}{3}} dy \tag{A.13}$$

$$K_3 = \frac{T}{n\left(1 + \alpha_i^2 + \alpha_i'^2\right)^{\frac{1}{4}}} \tag{A.14}$$

$$T = \int_{y_i}^{y_{i+1}} (\beta_i - \delta (y - y_i)) (h_i - \alpha_i (y - y_i))^{\frac{5}{3}} dy \tag{A.15}$$

$$T = \int_{y_i}^{y_{i+1}} (\beta_i - \delta (y - y_i)) (h_i - \alpha_i (y - y_i))^{\frac{5}{3}} dy \tag{A.16}$$

$$T = \frac{\delta}{\alpha_i} \int_{y_i}^{y_{i+1}} \left(\beta_i \frac{\alpha_i}{\delta} - \alpha_i (y - y_i)\right) (h_i - \alpha_i (y - y_i))^{\frac{5}{3}} dy \tag{A.17}$$

$$T = \frac{\delta}{\alpha_i} \int_{y_i}^{y_{i+1}} \left(\beta_i \frac{\alpha_i}{\delta} - h_i + (h_i - \alpha_i (y - y_i))\right) (h_i - \alpha_i (y - y_i))^{\frac{5}{3}} dy \tag{A.18}$$

$$T = \frac{\delta}{\alpha_i} \int_{y_i}^{y_{i+1}} \left(\beta_i \frac{\alpha_i}{\delta} - h_i\right) (h_i - \alpha_i (y - y_i))^{\frac{5}{3}} + (h_i - \alpha_i (y - y_i))^{\frac{8}{3}} dy \tag{A.19}$$

$$T = \frac{-\delta}{\alpha_i \alpha_i} \left(\beta_i \frac{\alpha_i}{\delta} - h_i\right) \frac{3}{8} \left| (h_i - \alpha_i (y - y_i))^{\frac{8}{3}} \right|_{y_i}^{y_{i+1}} + \frac{-\delta}{\alpha_i \alpha_i} \frac{3}{11} \left| (h_i - \alpha_i (y - y_i))^{\frac{11}{3}} \right|_{y_i}^{y_{i+1}} \tag{A.20}$$

$$T = \frac{\delta}{\alpha_i \alpha_i} \left[\left(\beta_i \frac{\alpha_i}{\delta} - h_i\right) \frac{3}{8} \left(h_i^{\frac{8}{3}} - h_{i+1}^{\frac{8}{3}}\right) + \frac{3}{11} \left(h_i^{\frac{11}{3}} - h_{i+1}^{\frac{11}{3}}\right) \right] \tag{A.21}$$

$$T = \frac{1}{\alpha} \left[\left(\beta_i - h_i \frac{\delta}{\alpha_i}\right) \frac{3}{8} \left(h_i^{\frac{8}{3}} - h_{i+1}^{\frac{8}{3}}\right) + \frac{\delta}{\alpha_i} \frac{3}{11} \left(h_i^{\frac{11}{3}} - h_{i+1}^{\frac{11}{3}}\right) \right] \tag{A.22}$$

$$K_3 = \frac{T}{n\left(1 + \alpha_i^2 + \alpha_i'^2\right)^{\frac{1}{4}}} = \left(\beta_i - h_i \frac{\delta}{\alpha_i}\right) K_\alpha + \frac{1}{n(1 + \alpha_{si}^2 + \alpha_{ni}^2)^{\frac{1}{4}}} \frac{\delta}{\alpha_i \alpha_i} \frac{3}{11} \left(h_i^{\frac{11}{3}} - h_{i+1}^{\frac{11}{3}}\right) \tag{A.23}$$

References

Anderson, W. K. and D. L. Bonhaus, 1994. "An implicit upwind algorithm for computing turbulent flows on unstructured grids." *Computers Fluids* 23 (1): 1–21.

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.

Borsboom, M., 2013. *Construction and analysis of D-FLOW FM-type discretizations*. Tech. rep., Deltares memorandum.

Casulli and Stelling, 2013. "A semi-implicit numerical model for urban drainage systems." *International Journal for Numerical Methods in Fluids* .

Charnock, H., 1955. "Wind-stress on a water surface." *Q. J. Royal Meteorol. Soc.* 81: 639–640.

Dam, A. van, 2009. *Go with the Flow*. Ph.D. thesis, University of Utrecht. Moving meshes and solution monitoring for compressible flow simulation.

Delft3D-FLOW UM, 2013. *Delft3D-FLOW User Manual*. Deltares, 3.14 ed.

Haselbacher, A. and J. Blazek, 1999. "On the accurate and efficient discretization of the Navier-Stokes equations on mixed grids." In *Proceedings of the 14th AIAA CFD Conference*, AIAA Paper 99-3363-CP, pages 946–956. Snowmass,CO.

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.

Hwang, P., 2005a. "Comparison of the ocean surface wind stress computed with different parameterization functions of the drag coefficient." *J. Oceanogr.* 61: 91–107.

Hwang, P., 2005b. "Drag coefficient, dynamic roughness and reference wind speed." *J. Oceanogr.* 61: 399–413.

Karypis, G., 2013. *METIS - A Software Package for Partitioning Unstructured Graph, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices, Version 5.1.0*. Tech. rep., Department of Computer Science and Engineering, 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.

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.

Olesen, K. W., 1987. *Bed topography in shallow river bends*. Tech. rep., Delft University of Technology. Communications on Hydraulic and Geotechnical Engineering.

Perot, B., 2000. "Conservation properties of unstructured staggered mesh schemes." *J. Comput. Phys.* 159 (1): 58–89. DOI: 10.1006/jcph.2000.6424.

Preissmann A, C. J., ed., 1961. *Calcul des intumescences sur machinas electroniques.*, vol. Proceedengs of 9th Congress of International Association for Hydraulic Research (IAHR), Dubrovnik, Yugoslavia, 1961; 656-664. IAHR.

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.

Sieben, J., 2011. *Overzicht en synthese beschikbare data overlaatproeven, Update 2011*. Tech. rep., Rijkswaterstaat.

Smith, S. D. and E. G. Banke, 1975. "Variation of the sea surface drag coefficient with wind speed." *Quarterly Joournal of the Royal Meteorological Society* 101: 665–673.

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.

Vermaas, H., 1987. *Energylosses due to weirs*. Tech. Rep. Q92, WL | Delft Hydraulics, Delft, The Netherlands. In Dutch (Energieverliezen door overlaten: Een gewijzigde berekeningsprocedure voor WAQUA-rivieren versie).

Villemonte, J., 1947. "Submerged Weir Discharge Studies." *Engineering News Record* pages 866–869.

Wijbenga, J. H. A., 1990. *Representation of extra energy losses in RIVCUR*. Tech. Rep. Q910, WL | Delft Hydraulics, Delft, The Netherlands. In Dutch (Weergave van extra energieverlies in RIVCUR), research for Rijkswaterstaat, Dienst Binnenwateren/RIZA.

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 VH Delft
The Netehrlands

+31 (0)88 335 81 88
sales@deltaressystems.nl
www.deltaressystems.nl