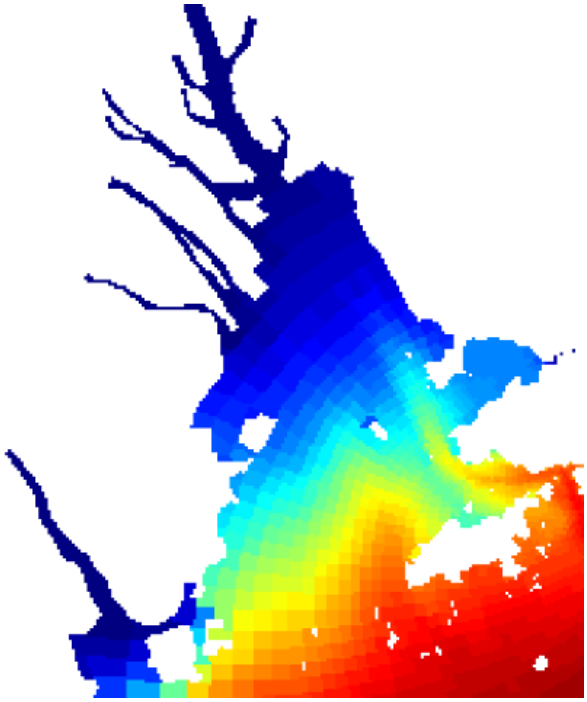
An accurate and robust finite volume method for the advection diffusion equation

Paulien van Slingerland

December 14, 2006



TU Delft
Delft University of Technology

WL | Delft Hydraulics

Contents

Chapter 1

Introduction

At present, plans are being made for the construction of Liquefied Natural Gas pipes in the sea bed off the coast of Hong Kong. To this end, dredging is necessary which causes plumes of silt in the water. The silt particles float in the water for a relatively long period of time, until, eventually, they settle on the sea bed. Unfortunately, both phenomena are in general harmful to coral reefs and Chinese white dolphins, two protected species that live in the sea near to Hong Kong. So, before the plans can be carried out, it is necessary to determine how much of the ocean may be affected by those plumes.

Water is indispensable for many organisms, especially for humans. People use it for drinking, fishing, bathing, irrigating, shipping, and so on. Accordingly, it is very important to maintain water quality. The quality of water is determined by the concentrations of the substances it contains, such as oxygen, salts, silt and bacteria. From the example above it is clear that it could easily be diminished. The question is: Could this be foreseen?

Fortunately, software is already available for this purpose. Delft3D-WAQ, a simulation program that has been developed by WL | Delft Hydraulics, is a useful tool in forecasting water quality. In particular, it is able to predict the size of silt plumes caused by dredging (see Figure 1.1). Basically, the software approximates the solution of the advection diffusion equation by means of the finite volume method. Since it is often necessary to predict one or two years ahead, large time steps are prefered in order to have limited computing time.

However, there are two aspects that need improvement. First of all, the current schemes are either expensive explicit higher order schemes or inaccurate implicit first order schemes. Moreover, the convergence speed of the present solver for linear systems is unsatisfactory for diffusion dominated problems.

In this literature study, answers to the following questions will be sought:

I. *What are the possibilities for an accurate finite volume scheme for the advection diffusion equation on an unstructured three-dimensional grid that has a high upper bound for the time step?*

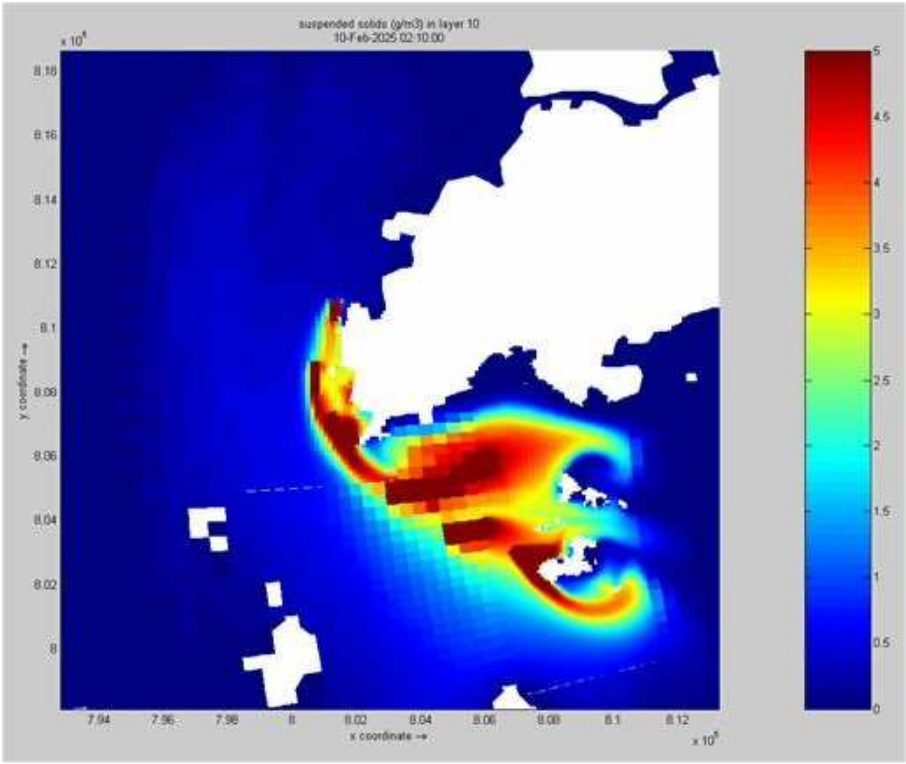
II. *How can the convergence speed of the current linear solver be increased for*

systems resulting from diffusion dominated problems?

In other words: Can the damage done to dolphins and coral reefs be estimated better and faster?

This report consists of seven chapters. First of all, a problem definition is given in Chapter 2. A physical water quality description is translated into a mathematical model that is based on the advection diffusion equation. The solution to this model can be approximated by means of the finite volume method, which is introduced in Chapter 3. Implicit variants of this method require the solution of many large linear systems. In order to solve these systems efficiently, iterative solvers are considered in Chapter 4. Useful tools in improving the performance of iterative schemes are preconditioning (Chapter 5) and reordering of the matrix (Chapter 6). Chapter 7 discusses storage schemes for sparse matrices that can save both memory and time.

Figure 1.1: WAQ's forecast of the silt plumes



Chapter 2

Physical and mathematical waterquality model

In this chapter, both a physical and a mathematical water quality model are discussed.

2.1 Physical model

The quality of water is determined by the concentrations of the substances it contains, such as oxygen, algae, salts, bacteria, viruses, toxic heavy metals, pesticides, and silt. These concentrations can be affected in two ways. Firstly, particles can be transported through the water in several ways. Moreover, water quality processes play an important role. Both phenomena will be discussed briefly below.

2.1.1 Transport

A substance can be transported by advection, diffusion, and by an own movement that is independent of the preceding types of transport.

Advection

Advection is transport of a substance due to the motion of the fluid. The flow carries the particles in the downstream direction.

Diffusion

Roughly speaking, *diffusion* is a mixing process. A distinction can be made between molecular diffusion and turbulent diffusion. *Molecular diffusion* is the spontaneous spreading of matter due to the random movement of molecules. It only applies to substances that are liquid, gaseous, or dissolved. A schematic visualisation is given by Figure 2.1. *Turbulent diffusion* is mixing due to turbulent flow. A practical example of turbulent water flow and an illustration of the blending process is shown in Figure 2.2.

Figure 2.1: Molecular diffusion

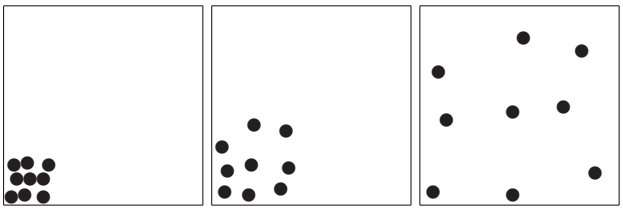
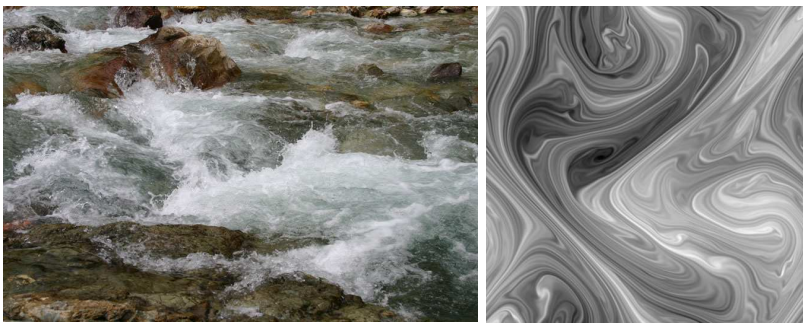


Figure 2.2: Turbulent diffusion



Own movement

Own movement is any movement that is not caused by advection or diffusion. This kind of movement could be forced by gravity, the substance itself, or the wind. *Gravitational movement* arises when there is a difference between the density of the substance and that of the water. Silt, for example, is heavier than water. Therefore, it will generally have an extra downward motion. *Active movement* only applies to organisms that can 'swim' in some sense. Examples are shrimps, fish, and certain algae that can propel themselves through the water. *Floating movement* is the motion that a floating substance obtains from the wind. As a result, its concentration is generally higher on the downwind water surface side of the area.

2.1.2 Water quality processes

Apart from transport, water quality processes can have a great effect on the concentration of a substance. Examples are photosynthesis, mineralisation, sedimentation, nitrification, and the mortality of bacteria. These processes will not be discussed in detail in this report, since all processes can be modeled by one nonlinear inhomogeneous term in the mathematical model, as will become clear in the next section.

2.2 Mathematical model

The mathematical water quality model, corresponding to the physical description above, is a special case of a conservation law.

Model 2.1 (Conservation law)**.** Let $c(\underline{\mathbf{x}}, t)$ be a conserved quantity with flux function $\underline{\mathbf{f}}(\underline{\mathbf{x}}, t)$ and source term $p(\underline{\mathbf{x}}, t)$. The behavior of c can be modeled according to a *conservation law*:

$$\begin{cases} \frac{\partial c}{\partial t}(\underline{\mathbf{x}}, t) + \nabla \cdot \underline{\mathbf{f}}\left(c(\underline{\mathbf{x}}, t)\right) &= p(\underline{\mathbf{x}}, t) \\ c(\underline{\mathbf{x}}, 0) &= \overset{\circ}{c}\left(\underline{\mathbf{x}}\right) \\ c|_{\underline{\mathbf{x}} \in \partial D_1} &= \breve{c}(\underline{\mathbf{x}}, t) \\ (\nabla c \cdot \underline{\mathbf{n}})|_{\underline{\mathbf{x}} \in \partial D_2} &= 0 \end{cases} \tag{2.1}$$

Here, $t \in [0, T] \subset [0, \infty)$ and $\underline{\mathbf{x}} \in D \subset \mathbb{R}^m$. $\overset{\circ}{c}\left(\underline{\mathbf{x}}\right)$ is the initial condition. The boundary of D is partitioned according to $\partial D = \partial D_1 \cup \partial D_2$. $\underline{\mathbf{n}}$ is the outward normal unit vector of D. On ∂D_1 a *Dirichlet boundary condition* and on ∂D_2 a *Neumann boundary condition* is imposed[1]. □

Model 2.2 (Water quality model)**.** Consider a fluid with velocity profile $\underline{\hat{\mathbf{u}}}(\underline{\mathbf{x}}, t)$ and diffusion coefficient $d(\underline{\mathbf{x}}, t)$. Suppose that the fluid contains a substance with concentration $c(\underline{\mathbf{x}}, t)$ and velocity due to own movement $\underline{\tilde{\mathbf{u}}}(\underline{\mathbf{x}}, t)$. Let $p(\underline{\mathbf{x}}, t)$ represent water quality processes. p may also depend on c or the concentration of other substances. The water quality model follows from Model 2.1 by using the following flux function:

$$\underline{\mathbf{f}}(c) = \underbrace{(\underline{\hat{\mathbf{u}}} + \underline{\tilde{\mathbf{u}}})}_{=:\underline{\mathbf{u}}} c - d\nabla c$$

□

2.3 Conclusion

The quality of water is determined bij the concentrations of the substances that it contains. These can be affected by transport and water quality processes. The corresponding mathematical model is the advection diffusion equation.

[1] Off course, there are other types of boundary conditions, but they will not be considered in this report.

Chapter 3

Finite volume method

In Chapter 2, a mathematical water quality model was formulated. In general, it is impossible to solve such a model analytically. However, a numerical approximation can be obtained by means of the finite volume method, which is discussed in this chapter.

3.1 Finite volume method for scalar conservation laws

A conservation law can be integrated to obtain an integral form, which forms the basis of the finite volume method.

Proposition 3.1 (Integral form)**.** *Consider Model 2.1. Let $V \subset D$ be a control volume. Then,*

$$\frac{d}{dt}\int_V c(\underline{\mathbf{x}},t)\,d\underline{\mathbf{x}} + \int_{\partial V} \underline{\mathbf{f}}(c(\underline{\mathbf{x}},t))\cdot\underline{\mathbf{n}}\,d\underline{\mathbf{x}} \quad = \quad \int_V p(\underline{\mathbf{x}},t)\,d\underline{\mathbf{x}} \tag{3.1}$$

Proof. Integrate $\frac{\partial c}{\partial t} + \nabla\cdot\underline{\mathbf{f}}(c) = p(\underline{\mathbf{x}},t)$ over V and apply the Gauss theorem. ■

Basically, the finite volume method subdivides the space domain into grid cells and approximates the integral form for each cell. In order to do this, the notion of numerical flux needs to be introduced. The grid is chosen to be cell-centered.

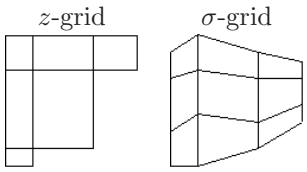
Definition 3.2 (Cell centered grid)**.** A *cell centered grid* of a spatial domain $D \subset \mathbb{R}^d$ consists of set of control volumes $V = \{V_i \subset D : i = 1, ..., I\}$ and a set of storage locations $X = \{\underline{\mathbf{x}}_i \in D : i = 1, ..., I\}$ such that $D = \cup_{i=1}^{I} V_i$ and $\underline{\mathbf{x}}_i$ is at the center of mass of V_i. Notation: $G = (V, X)$. □

Remark 3.3 (Grid in WAQ). WAQ often receives the velocity profile from Delft-3DFLOW, another simulation program. FLOW can only handle two types of structured grids, which are schematically displayed in Figure 3.1. Both types use a staggered grid in the horizontal direction. A *z-grid* is composed of columns with rectangular time independent volumes (more precisely: orthotopes), whereby the number of volumes per column may vary. A *σ-grid* is composed of columns with a fixed number of time dependent cells, whose edges

are not necessarily perpendicular.

WAQ's grid results from aggregating adjacent cells of the mesh used by FLOW. From the example in Figure 3.2 becomes clear that this generally leads to an unstructured grid. Some schemes in WAQ require some structure though. □

Figure 3.1: Structured grid types of FLOW



Definition 3.4 (Numerical flux function)**.** Consider a conservation law (Model 2.1) and a cell centered grid $G = (V, X)$ for D. Define $S_{ij} = \partial V_i \cap \partial V_j$. A *numerical flux function* is a (locally Lipschitz continuous) function $\phi_{ij} : \mathbb{R}^2 \to \mathbb{R}$ that has following two properties:

1. Consistency:

$$\phi_{ij}(u, u) = \frac{1}{|S_{ij}|} \int_{S_{ij}} \underline{\mathbf{f}}(u) \cdot \underline{\mathbf{n}} \, d\underline{\mathbf{x}}$$

2. Conservation:

$$\phi_{ij}(u, v) = -\phi_{ji}(v, u)$$

See Figure 3.3 for an illustration. □

Now, the finite volume method can be formulated.

Method 3.5 (Finite Volume Method (FVM))**.** A *finite volume* approximation for Model 2.1 can be obtained by applying the following steps:

1. Choose a time discretisation $t_0, t_1, ..., t_N \in [0, T]$ such that

$$0 = t_0 < t_1 < ... < t_N$$

   Furthermore, choose a cell-centered grid

$$G^n = (\{V_1^n, ..., V_I^n\}, \{\underline{\mathbf{x}}_1^n, ..., \underline{\mathbf{x}}_I^n\})$$

   for the time dependent space domain D at each time t_n $(n = 1, ..., N)$.

2. Additionally, introduce the following cell averages:

$$\begin{aligned} \overline{c}_i^n &= \frac{1}{|V_i^n|} \int_{V_i^n} c(\underline{\mathbf{x}}, t_n) \, d\underline{\mathbf{x}} \\ \overline{p}_i^n &= \frac{1}{|V_i^n|} \int_{V_i^n} p(\underline{\mathbf{x}}, t_n) \, d\underline{\mathbf{x}} \end{aligned}$$

Figure 3.2: An example of a grid in WAQ (bottom), resulting from gathering grid cells of the grid used by FLOW (top)

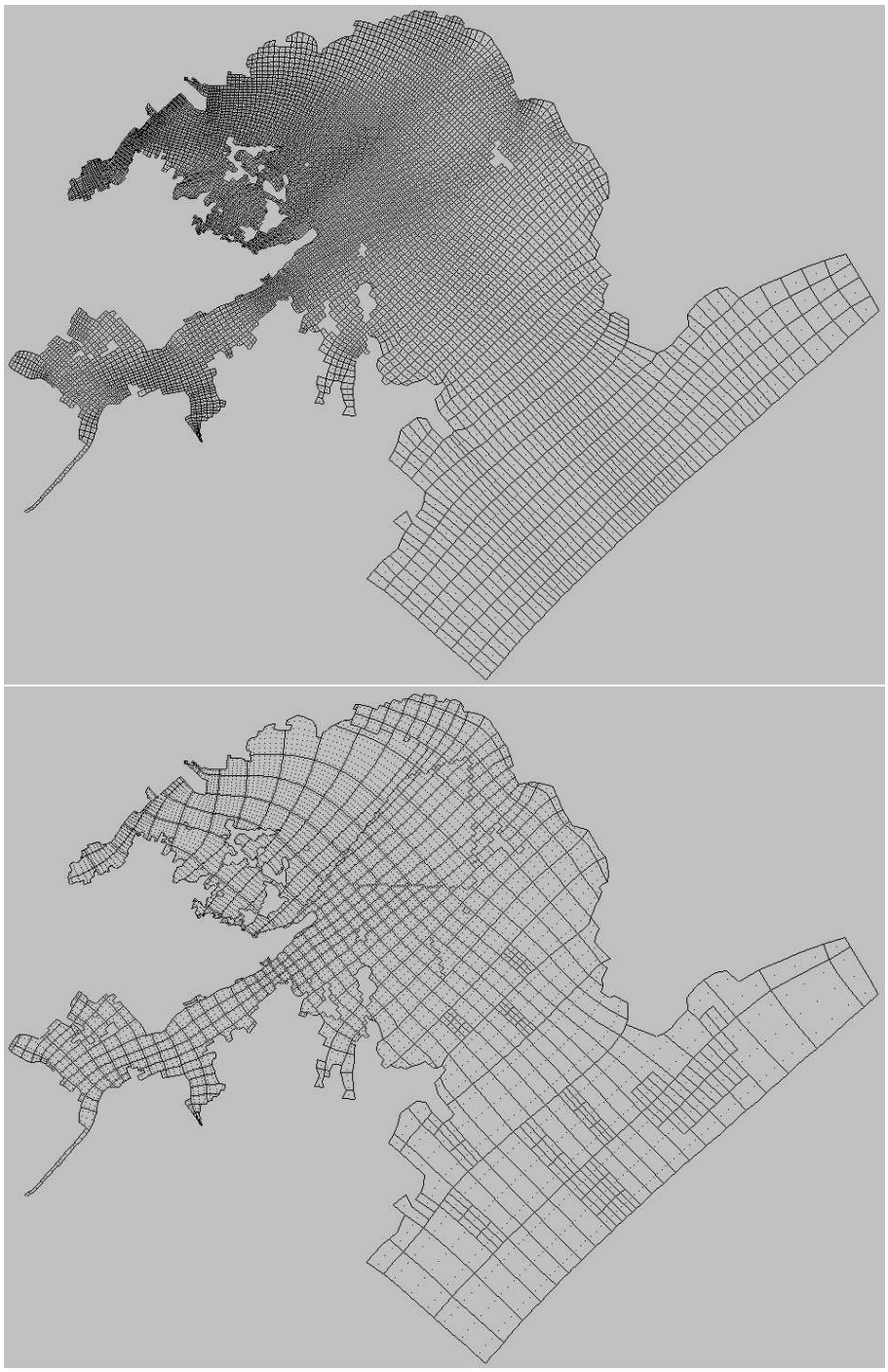
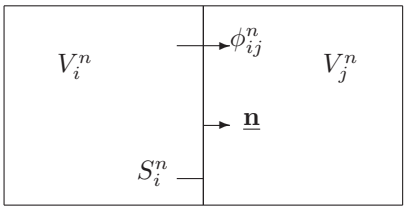


Figure 3.3: Schematic illustration of numerical flux



If K grid cells are adjacent to the boundary ∂D_1, then adjacent dummy volumes $\breve{V}_k^n$ $(k = 1, ..., K)$ are introduced adjacent to ∂D_1 with average value

$$\breve{\overline{c}}_k^n = \int_{\partial D \cap \partial \breve{V}_k^n} \breve{c}(\underline{\mathbf{x}}, t_n)\, d\underline{\mathbf{x}}$$

3. Let $A_i^n = \{j : V_j^n \text{ adjacent to } V_i^n\}$ contain the indices of the neighboring grid cells of V_i^n. Furthermore, let $S_{ij}^n = \partial V_i^n \cap \partial V_j^n$ denote the common boundary of neighbors V_i^n and V_j^n. Introduce analogue quantities for the virtual cells: $\breve{A}_i^n = \{k : \breve{V}_k^n \text{ adjacent to } V_i^n\}$ and $\breve{S}_{ik}^n = \partial V_i^n \cap \partial \cap V_k^n$.

   Approximate (3.1) for a grid cell V_i^n according to[1]:

$$g(\overline{c}_i^n) := \frac{|V_i^n|\overline{c}_i^n - |V_i^{n-1}|\overline{c}_i^{n-1}}{t_n - t_{n-1}} - \theta a_i^n - (1-\theta) a_i^{n-1} \quad = \quad 0 \qquad (3.2)$$

   with

$$a_i^n \quad = \quad |V_i^n|\overline{p}_i^n - \sum_{j \in A_i^n} |S_{ij}^n| \phi_{ij}(\overline{c}_i^n, \overline{c}_j^n) - \sum_{k \in \breve{A}_i^n} |\breve{S}_{ik}^n| \breve{\phi}_{ik}(\overline{c}_i^n, \breve{\overline{c}}_k^n) \qquad (3.3)$$

   ϕ_{ij} and $\breve{\phi}_{ik}$ are numerical flux functions (see Definition 3.4). $\theta \in [0, 1]$ is a parameter. Note that the method is fully *explicit* if $\theta = 0$, and fully *implicit* if $\theta = 1$.

4. Solve the resulting systems to obtain $\overline{c}_i^n$ $(i = 1, ..., I;\ n = 1, ..., N)$.

5. Approximate the solution of Model 2.1 according to $c(\underline{\mathbf{x}}_i, t_n) \approx \overline{c}_i^n$. □

More detailed information on the finite volume method can be found in [1], [7, Chapter 4], and [9, Chapter 3].

3.1.1 Convergence, stability, and related topics

In order to determine the quality of a finite volume scheme, several properties have to be introduced.

First of all, the method should yield the exact solution for infinitely small grid cells and time steps. In that case, the method is convergent.

[1] There are other approximations possible, but they will not be considered in this report.

Definition 3.6 (Global truncation error)**.** The *global truncation error* of the FVM (Method 3.5) at time t_n is defined as:

$$e_i^n \quad = \quad c(\underline{\mathbf{x}}_i, t_n) - \overline{c}_i^n \qquad \square$$

Definition 3.7 (Local truncation error)**.** The *local truncation error* of the FVM (Method 3.5) at time t_n is defined as:

$$\tilde{e}_i^n \quad = \quad g(c(\underline{\mathbf{x}}_i, t_n))$$

Here, g is as in (3.2). $\square$

Definition 3.8 (Convergence)**.** Consider the FVM (Method 3.5). Let the spatial mesh sizes and the time steps be decreasing functions of a parameter h. The FVM (Method 3.5) *converges* at time t_n with respect to some norm $\|.\|$ if

$$\lim_{h \downarrow 0} \|\underline{\mathbf{e}}^n\| \quad = \quad 0 \qquad t_n, \underline{\mathbf{x}}_1, ..., \underline{\mathbf{x}}_I \text{ fixed}$$

Here, $\underline{\mathbf{e}}^n$ is the vector containing the global truncation errors. $\square$

Intuitively, convergence can only occur if the local truncation error is small enough. This condition is called consistency.

Definition 3.9 (Consistency)**.** Consider the FVM (Method 3.5). Let the spatial mesh sizes and the time steps be decreasing functions of a parameter h. The FVM (Method 3.5) is *consistent* at time t_n with respect to some norm $\|.\|$ if

$$\lim_{h \downarrow 0} \|\underline{\tilde{\mathbf{e}}}^n\| \quad = \quad 0 \qquad t_n, \underline{\mathbf{x}}_1, ..., \underline{\mathbf{x}}_I \text{ fixed}$$

Here, $\underline{\tilde{\mathbf{e}}}^n$ is the vector containing the local truncation errors. $\square$

Secondly, a small perturbation in the initial condition should not lead to a completely different solution. This is what stability signifies.

Definition 3.10 (Absolute stability)**.** The FVM (Method 3.5) is called *absolutely stable*, if there exists constants $k, \tau > 0$ (τ may depend on the spatial mesh size) such that, if

$$t_n - t_{n-1} \leq \tau \qquad \text{for all } n = 1, ..., N$$

then, for any perturbation $\underline{\mathbf{q}}^0$ of the initial condition $\underline{\overline{\mathbf{c}}}^0$, resulting in a perturbation $\underline{\mathbf{q}}^n$ in $\underline{\overline{\mathbf{c}}}^n$:

$$\|\underline{\mathbf{q}}^n\| \leq k \|\underline{\mathbf{q}}^0\| \qquad \text{for all } n = 1, ..., N \qquad \square$$

Additionally, negative concentrations are unphysical. Therefore, the scheme should be positive.

Definition 3.11 (Positivity)**.** The FVM (Method 3.5) is *positive* if

$$\overline{c}_i^n \geq 0 \qquad \text{for all } i = 1, .., I, n = 1, ..., N \qquad \square$$

Theorem 3.12 (Positivity preserving)**.** *If Method 3.5 yields solutions* $\underline{\overline{\mathbf{c}}}^n = \{\overline{c}_i^n\}$ *so that* $\exists A, B \in \mathbb{R}^{I \times I}$ *such that:*

1. *A is an M-matrix*
2. *B has nonnnegative entries*
3. $\forall n = 1, ..., N-1$*:*

$$A\underline{\overline{c}}^{n+1} = B\underline{\overline{c}}^{n}$$

Then, the scheme is positivity preserving.
Proof. See [4, p. 533] ■

Finally, the method should not generate spurious wiggles. In the one-dimensional case this is ensured by monotonicity preserving schemes.

Definition 3.13 (Monotonicity preserving)**.** In the one-dimensional case, the FVM (Method 3.5) is *monotonicity preserving* if

$$\overline{c}_i^n \geq \overline{c}_{i+1}^n \quad \forall i = 1, ..., I \Rightarrow \overline{c}_i^{n+1} \geq \overline{c}_{i+1}^{n+1} \quad \forall i = 1, ..., I \qquad \square$$

The following property extends the concept of monotonicity preserving schemes to the multi-dimensional case.

Definition 3.14 (Local Extremum Diminishing (LED))**.** The FVM (Method 3.5) is *local extremum diminishing* if local maxima are non-increasing and local minima are nondecreasing. □

It seems reasonable that the numerical flux ϕ_{ij}^n, that is directed from grid cell V_i^n to V_j^n, should increase, if the concentration $\overline{c}_i^n$ in volume V_i^n increases. At the same time, the flux should decrease, if $\overline{c}_j^n$ increases. This leads to the following LED criterion.

Definition 3.15 (Monotone numerical flux)**.** A (partial differentiable) numerical flux function ϕ_{ij} is *monotone* if

1. $\frac{\partial}{\partial u}\phi_{ij}(u, v) \geq 0$
2. $\frac{\partial}{\partial v}\phi_{ij}(u, v) \leq 0$ □

Proposition 3.16. *Consider the FVM (Method 3.5) for* $p_i^n = 0$ *(*$i = 1, ..., I, n = 0, ..., N$*). The scheme is LED if the applied numerical flux functions are monotone.*
Proof. See [1, p.12]. ■

3.2 Finite volume method for the one-dimensional advection equation

In this section, several examples of numerical flux functions for the one-dimensional advection equation are discussed.

Model 3.17 (Advection equation (1D))**.** The one-dimensional advection equation follows from Model 2.1 for $m = 1$, $p = 0$ and flux function:

$$f(c(x,t)) = u(x,t)c(x,t) \qquad \square$$

Definition 3.18 (Order of accuracy)**.** Consider the one-dimensional variant ($m = 1$) of Method 3.5 with constant time step Δt and constant cell width Δx. The method is said to be s_1 *order accurate* in time and s_2 order accurate in space with respect to a norm $\|.\|$ if:

$$\|\underline{\mathbf{e}}^n\| = O(\Delta t^{s_1}) + O(\Delta x^{s_2})$$

Here, $\underline{\mathbf{e}}^n$ is the vector containing the global truncation errors. □

3.2.1 First order schemes

The following two schemes are both first order in space.

Method 3.19 (Central scheme)**.** The *central* scheme for Model 3.17 follows from Method 3.5 by using the following numerical flux:

$$\phi_{ij}(\overline{c}_i^n, \overline{c}_j^n) \quad = \quad \overline{\overline{u}}_{ij}^n \cdot n_{ij}^n \frac{\overline{c}_i^n + \overline{c}_j^n}{2}$$

Here, $\overline{\overline{u}}_{ij}^n$ denotes the value of u on the boundary of cells V_i and V_j. □

Method 3.20 (First order upwind scheme)**.** The *first order upwind* scheme for Model 3.17 follows from Method 3.5 by using the following numerical flux:

$$\phi_{ij}(\overline{c}_i^n, \overline{c}_j^n) \quad = \quad \max\{\overline{\overline{u}}_{ij}^n \cdot n_{ij}^n, 0\}\overline{c}_i^n + \min\{\overline{\overline{u}}_{ij}^n \cdot n_{ij}^n, 0\}\overline{c}_j^n$$

Here, $\overline{\overline{u}}_{ij}^n$ denotes the value of u on the boundary of cells V_i and V_j. □

A disadvantage of the first order upwind schme is that it introduces numerical diffusion, as becomes clear from the proposition below.

Proposition 3.21 (Modified equation for the first order upwind scheme)**.** *Consider Model 3.17 for contant $u > 0$ on a time independent space domain. Apply the FVM with first order upwind fluxes (Method 3.20), constant grid cell width Δx, and constant time step Δt:*

$$\frac{\overline{c}_i^{n+1} - \overline{c}_i^n}{\Delta t} + \theta u \frac{\overline{c}_i^{n+1} - \overline{c}_{i-1}^{n+1}}{\Delta x} + (1-\theta)u \frac{\overline{c}_i^n - \overline{c}_{i-1}^n}{\Delta x} \quad = \quad 0$$

Let $\eta(x,t)$ be a function such that $\eta(x_i, t_n) = \overline{c}_i^n$ (for all $i = 1, ..., I$, for all $n = 1, ..., N$). Then for all $i = 1, ..., I$ and for all $n = 1, ..., N$:

$$\frac{\partial \eta}{\partial t}(x_i, t_n) + u \frac{\partial \eta}{\partial x}(x_i, t_n) \approx \underbrace{\frac{u\Delta x}{2}\left(1 - (1-2\theta)\frac{u\Delta t}{\Delta x}\right)}_{\textit{Numerical diffusion coefficient}} \frac{\partial^2 \eta}{\partial x^2}(x_i, t_n)$$

Proof. Because $\eta(x_i, t_n) = \overline{c}_i^n$,

$$\begin{aligned} &\frac{\eta(x_i, t_{n+1}) - \eta(x_i, t_n)}{\Delta t} \\ +\theta u &\frac{\eta(x_i, t_{n+1}) - \eta(x_{i-1}, t_{n+1})}{\Delta x} \\ +(1-\theta)u &\frac{\eta(x_i, t_n) - \eta(x_{i-1}, t_n)}{\Delta x} \quad = \quad 0 \end{aligned}$$

Using a Taylor expansion around t_n results in (higher order terms that will be neglected later are colored):

$$\begin{aligned}
&\frac{\partial \eta}{\partial t}(x_i,t_n) + \frac{\Delta t}{2}\frac{\partial^2 \eta}{\partial t^2}(x_i,t_n) + \frac{\Delta t^2}{6}\frac{\partial^3 \eta}{\partial t^3}(x_i,\tau_1) \\
+\theta u\Bigg(&\frac{\eta(x_i,t_n) + \Delta t\frac{\partial \eta}{\partial t}(x_i,t_n) + \frac{\Delta t^2}{2}\frac{\partial^2 \eta}{\partial t^2}(x_i,t_n) + \frac{\Delta t^3}{6}\frac{\partial^3 \eta}{\partial t^3}(x_i,\tau_2)}{\Delta x} \\
-&\frac{\eta(x_{i-1},t_n) + \Delta t\frac{\partial \eta}{\partial t}(x_{i-1},t_n) + \frac{\Delta t^2}{2}\frac{\partial^2 \eta}{\partial t^2}(x_{i-1},t_n) + \frac{\Delta t^3}{6}\frac{\partial^3 \eta}{\partial t^3}(x_{i-1},\tau_3)}{\Delta x}\Bigg) \\
&+(1-\theta)u\frac{\eta(x_i,t_n) - \eta(x_{i-1},t_n)}{\Delta x} \quad = \quad 0
\end{aligned}$$

for certain $\tau_1, \tau_2, \tau_3 \in [t_n, t_{n+1}]$. Applying a Taylor expansion around x_i yields:

$$\begin{aligned}
&\frac{\partial \eta}{\partial t}(x_i,t_n) + \frac{\Delta t}{2}\frac{\partial^2 \eta}{\partial t^2}(x_i,t_n) + \frac{\Delta t^2}{6}\frac{\partial^3 \eta}{\partial t^3}(x_i,\tau_1) \\
+\theta u\Bigg(&\frac{\partial \eta}{\partial x}(x_i,t_n) - \frac{\Delta x}{2}\frac{\partial^2 \eta}{\partial x^2}(x_i,t_n) + \frac{\Delta x^2}{6}\frac{\partial^3 \eta}{\partial x^3}(\xi_1,t_n) \\
&+\Delta t\frac{\partial}{\partial x}\frac{\partial \eta}{\partial t}(x_i,t_n) - \frac{\Delta t\Delta x}{2}\frac{\partial^2}{\partial x^2}\frac{\partial \eta}{\partial t}(\xi_2,t_n) \\
&+\frac{\Delta t^3}{6\Delta x}\frac{\partial^3 \eta}{\partial t^3}(x_i,\tau_2) - \frac{\Delta t^3}{6\Delta x}\frac{\partial^3 \eta}{\partial t^3}(\xi_3,\tau_3)\Bigg) \\
+(1-\theta)u\left(\frac{\partial \eta}{\partial x}(x_i,t_n) - \frac{\Delta x}{2}\frac{\partial^2 \eta}{\partial x^2}(x_i,t_n) + \frac{\Delta x^2}{6}\frac{\partial^3 \eta}{\partial x^3}(\xi_4,t_n)\right) \quad &= \quad 0
\end{aligned}$$

for certain $\xi_1, \xi_2, \xi_3, \xi_4 \in [x_{i-1}, x_i]$. Rewriting gives:

$$\begin{aligned}
\frac{\partial \eta}{\partial t}(x_i,t_n) + u\frac{\partial \eta}{\partial x}(x_i,t_n) \quad = \quad & \frac{u\Delta x}{2}\frac{\partial^2 \eta}{\partial x^2}(x_i,t_n) \\
&-\frac{\Delta t}{2}\frac{\partial^2 \eta}{\partial t^2}(x_i,t_n) - \theta u\Delta t\frac{\partial}{\partial x}\frac{\partial \eta}{\partial t}(x_i,t_n) \\
&-\frac{\Delta t^2}{6}\frac{\partial^3 \eta}{\partial t^3}(x_i,\tau_1) \\
&-\theta u\Bigg(\frac{\Delta x^2}{6}\frac{\partial^3 \eta}{\partial x^3}(\xi_1,t_n) - \frac{\Delta t\Delta x}{2}\frac{\partial^2}{\partial x^2}\frac{\partial \eta}{\partial t}(\xi_2,t_n) \\
&+\frac{\Delta t^3}{6\Delta x}\frac{\partial^3 \eta}{\partial t^3}(x_i,\tau_2) - \frac{\Delta t^3}{6\Delta x}\frac{\partial^3 \eta}{\partial t^3}(\xi_3,\tau_3)\Bigg) \\
&-(1-\theta)u\frac{\Delta x^2}{6}\frac{\partial^3 \eta}{\partial x^3}(\xi_4,t_n)
\end{aligned}$$

Note that:

$$\begin{aligned}
\frac{\partial \eta}{\partial t}(x_i,t_n) \quad &= \quad -u\frac{\partial \eta}{\partial x}(x_i,t_n) + O(\Delta t) + O(\Delta x) \\
\frac{\partial^2 \eta}{\partial t^2}(x_i,t_n) \quad &= \quad u^2\frac{\partial^2 \eta}{\partial x^2}(x_i,t_n) + O(\Delta t) + O(\Delta x)
\end{aligned}$$

Substitution yields:

$$\begin{aligned}\frac{\partial \eta}{\partial t}(x_i,t_n)+u\frac{\partial \eta}{\partial x}(x_i,t_n) &= \frac{u\Delta x}{2}\frac{\partial^2\eta}{\partial x^2}(x_i,t_n)\\ &\quad -\frac{u^2\Delta t}{2}\frac{\partial^2\eta}{\partial x^2}(x_i,t_n)+\theta u^2\Delta t\frac{\partial^2\eta}{\partial x^2}(x_i,t_n)\\ &\quad +O\left(\Delta t^2\right)+O\left(\Delta x^2\right)+O(\Delta x\Delta t)\end{aligned}$$

Rewriting and neglecting second order terms ends the proof. ■

Ideally, the numerical diffusion coefficient is zero. On the one hand, it should be positive in order to have stability. On the other hand, it should be as small as possible to avoid too much smearing of the solution. Now, there are two interesting cases.

1. If $\theta \in [\frac{1}{2},1]$, the numerical diffusion coefficient of this scheme is positive for all $u,\Delta x,\Delta t>0$. This means that the scheme is stable for any time step!

2. If $\theta=0$, the numerical diffsuion coefficient is positive provided that

$$\frac{u\Delta t}{\Delta x}\leq 1$$

   This is also known as the *CFL condition*. Note that $\frac{u\Delta t}{\Delta x}=1$ implies zero numerical diffusion.

3.2.2 Higher order schemes

The following schemes are higher order accurate in space.

Method 3.22 (Second order upwind scheme)**.** The *second order upwind* scheme for Model 3.17 follows from Method 3.5 by using the following numerical flux:

$$\phi_{ij}(\overline{c}_i^n,\overline{c}_j^n) = \max\{\overline{\overline{u}}_{ij}^n\cdot n_{ij}^n,0\}\frac{3\overline{c}_i^n-1\overline{c}_{i-1}^n}{2}+\min\{\overline{\overline{u}}_{ij}^n\cdot n_{ij}^n,0\}\frac{3\overline{c}_j^n-\overline{c}_{j+1}^n}{2}$$

Here, $\overline{\overline{u}}_{ij}^n$ denotes the value of u on the boundary of cells V_i and V_j. V_{i-1} is the volume next to V_i, opposite to V_j. Analogously, V_{j+1} is the volume next to V_j, opposite to V_i. □

Method 3.23 (Third order upwind scheme)**.** The *third order upwind* scheme for Model 3.17 follows from Method 3.5 by using the following numerical flux:

$$\begin{aligned}\phi_{ij}(\overline{c}_i^n,\overline{c}_j^n) &= \max\{\overline{\overline{u}}_{ij}^n\cdot n_{ij}^n,0\}\frac{10\overline{c}_i^n-5\overline{c}_{i-1}^n+\overline{c}_{i-2}^n}{6}\\ &\quad +\min\{\overline{\overline{u}}_{ij}^n\cdot n_{ij}^n,0\}\frac{10\overline{c}_j^n-5\overline{c}_{j+1}^n+\overline{c}_{j+2}^n}{6}\end{aligned}$$

Here, $\overline{\overline{u}}_{ij}^n$ denotes the value of u on the boundary of cells V_i and V_j. V_{i-1} is the volume next to V_i, opposite to V_j. Analogously, V_{j+1} is the volume next to V_j, opposite to V_i. □

Intuititively, the higher order upwind schemes would not perform well on unstructured grids. The reason is that there are more than two grid points involved that are generally not in a straight line. A higher order scheme that does not have this disadvantage is Lax-Wendroff.

Method 3.24 (Explicit Lax-Wendroff scheme)**.** The *explicit Lax-Wendroff* scheme for Model 3.17 follows from Method 3.5 by using $\theta = 0$ and the following numerical flux:

$$\phi_{ij}(\overline{c}_i^n, \overline{c}_j^n) \quad = \quad \overline{\overline{u}}_{ij}^n \cdot n_{ij}^n \frac{\overline{c}_i^n + \overline{c}_j^n}{2} - (t_n - t_{n-1}) \frac{\left(\overline{\overline{u}}_{ij}^n \cdot n_{ij}^n\right)^2}{2} \frac{\overline{c}_j^n - \overline{c}_i^n}{|x_j^n - x_i^n|}$$

Here, $\overline{\overline{u}}_{ij}^n$ denotes the value of u on the boundary of cells V_i and V_j. □

Method 3.25 (Implicit Lax-Wendroff scheme)**.** The *implicit Lax-Wendroff* scheme for Model 3.17 follows from Method 3.5 by using $\theta = 0$ and the following numerical flux:

$$\phi_{ij}(\overline{c}_i^n, \overline{c}_j^n) \quad = \quad \overline{\overline{u}}_{ij}^n \cdot n_{ij}^n \frac{\overline{c}_i^n + \overline{c}_j^n}{2} + (t_n - t_{n-1}) \frac{\left(\overline{\overline{u}}_{ij}^n \cdot n_{ij}^n\right)^2}{2} \frac{\overline{c}_j^n - \overline{c}_i^n}{|x_j^n - x_i^n|}$$

Here, $\overline{\overline{u}}_{ij}^n$ denotes the value of u on the boundary of cells V_i and V_j. □

3.3 Finite volume method for the water quality model

Proposition 3.26 (Numerical flux for the water quality model)**.** *Consider the FVM (Method 3.5) for Model 2.2. So, the numerical flux should approximate:*

$$f_{ij}^n := \frac{1}{|S_{ij}^n|} \int_{S_{ij}^n} \Big(\underline{\mathbf{u}}(\underline{\mathbf{x}}, t_n) c(\underline{\mathbf{x}}, t_n) - d(\underline{\mathbf{x}}, t_n) \nabla c(\underline{\mathbf{x}}, t_n) \Big) \cdot \underline{\mathbf{n}} \, d\underline{\mathbf{x}} \tag{3.4}$$

Define the following boundary averages:

$$\begin{aligned} \overline{\overline{c}}_{ij}^n &= \frac{1}{|S_{ij}^n|} \int_{S_{ij}^n} c(\underline{\mathbf{x}}, t_n) \, d\underline{\mathbf{x}} \\ \overline{\overline{\underline{\mathbf{u}}}}_{ij}^n &= \frac{1}{|S_{ij}^n|} \int_{S_{ij}^n} \underline{\mathbf{u}}(\underline{\mathbf{x}}, t_n) \, d\underline{\mathbf{x}} \end{aligned}$$

Furthermore, introduce the corresponding deviations:

$$\begin{aligned} \tilde{c}_{ij}^n(\underline{\mathbf{x}}) &= c(\underline{\mathbf{x}}, t_n) - \overline{\overline{c}}_{ij}^n \\ \tilde{\underline{\mathbf{u}}}_{ij}^n(\underline{\mathbf{x}}) &= \underline{\mathbf{u}}(\underline{\mathbf{x}}, t_n) - \overline{\overline{\underline{\mathbf{u}}}}_{ij}^n \end{aligned}$$

Then, (3.4) is equivalent to:

$$f_{ij}^n = \overline{\overline{c}}_{ij}^n \overline{\overline{\underline{\mathbf{u}}}}_{ij}^n \cdot \underline{\mathbf{n}}_{ij}^n - \frac{1}{|S_{ij}^n|} \int_{S_{ij}^n} \Big(d(\underline{\mathbf{x}}, t_n) \nabla c(\underline{\mathbf{x}}, t_n) - \tilde{c}_{ij}^n \tilde{\underline{\mathbf{u}}}_{ij}^n(\underline{\mathbf{x}}) \Big) \cdot \underline{\mathbf{n}} \, d\underline{\mathbf{x}} \tag{3.5}$$

Proof. The advection term can be rewritten according to:

$$\begin{aligned}\int_{S_{ij}^n} c(\underline{\mathbf{x}},t_n)\underline{\mathbf{u}}(\underline{\mathbf{x}},t_n)\cdot\underline{\mathbf{n}}\,d\underline{\mathbf{x}} &= \int_{S_{ij}^n} \overline{\tilde{c}}_{ij}^n\overline{\underline{\tilde{\mathbf{u}}}}_{ij}^n\cdot\underline{\mathbf{n}}\,d\underline{\mathbf{x}}+\int_{S_{ij}^n}\tilde{c}_{ij}^n\underline{\tilde{\mathbf{u}}}_{ij}^n(\underline{\mathbf{x}})\cdot\underline{\mathbf{n}}\,d\underline{\mathbf{x}}\\ &\quad+\int_{S_{ij}^n}\overline{\tilde{c}}_{ij}^n\underline{\tilde{\mathbf{u}}}_{ij}^n(\underline{\mathbf{x}})\cdot\underline{\mathbf{n}}\,d\underline{\mathbf{x}}+\int_{S_{ij}^n}\tilde{c}_{ij}^n(\underline{\mathbf{x}})\overline{\underline{\tilde{\mathbf{u}}}}_{ij}^n(\underline{\mathbf{x}})\cdot\underline{\mathbf{n}}\,d\underline{\mathbf{x}}\end{aligned}$$

Since the average deviation of the average is zero, this reduces to:

$$\begin{aligned}\int_{S_{ij}^n} c(\underline{\mathbf{x}},t_n)\underline{\mathbf{u}}(\underline{\mathbf{x}},t_n)\cdot\underline{\mathbf{n}}\,d\underline{\mathbf{x}} &= \int_{S_{ij}^n}\overline{\tilde{c}}_{ij}^n\overline{\underline{\tilde{\mathbf{u}}}}_{ij}^n\cdot\underline{\mathbf{n}}\,d\underline{\mathbf{x}}+\int_{S_{ij}^n}\tilde{c}_{ij}^n\underline{\tilde{\mathbf{u}}}_{ij}^n(\underline{\mathbf{x}})\cdot\underline{\mathbf{n}}\,d\underline{\mathbf{x}}\\ &= |S_{ij}^n|\overline{\tilde{c}}_{ij}^n\overline{\underline{\tilde{\mathbf{u}}}}_{ij}^n\cdot\underline{\mathbf{n}}_{ij}^n+\int_{S_{ij}^n}\tilde{c}_{ij}^n\underline{\tilde{\mathbf{u}}}_{ij}^n(\underline{\mathbf{x}})\cdot\underline{\mathbf{n}}\,d\underline{\mathbf{x}}\end{aligned}$$

Substitution in (3.4) completes the proof. ■

Assumption 3.27. The term $\frac{1}{|S_{ij}^n|}\int_{S_{ij}^n}\tilde{c}_{ij}^n\underline{\tilde{\mathbf{u}}}_{ij}^n(\underline{\mathbf{x}})\cdot\underline{\mathbf{n}}\,d\underline{\mathbf{x}}$ in (3.5) represents the effect of turbulence on a sub-grid scale. It is assumed that this term can be modeled as a diffusion term, i.e. $\exists\tilde{D}: D\times[0,T]\to\mathbb{R}^{m\times m}$ such that (3.5) is equivalent to:

$$\phi_{ij}^n\approx f_{ij}^n=\overline{\tilde{c}}_{ij}^n\overline{\underline{\tilde{\mathbf{u}}}}_{ij}^n\cdot\underline{\mathbf{n}}_{ij}^n-\frac{1}{|S_{ij}^n|}\int_{S_{ij}^n}\Big(\tilde{D}(\underline{\mathbf{x}},t_n)\nabla c(\underline{\mathbf{x}},t_n)\Big)\cdot\underline{\mathbf{n}}\,d\underline{\mathbf{x}}\tag{3.6}$$

□

Remark 3.28 (Magnitude of $\tilde{D}$). The order of the magnitude of the elements of $\tilde{D}$ strongly depends on the dimension of the problem:

Dimension	Order of magnitude of extra diffusion
1	$1000\,\mathrm{m}^2\,\mathrm{s}^{-1}$
2	$10\,\mathrm{m}^2\,\mathrm{s}^{-1}$
3	$1\,\mathrm{m}^2\,\mathrm{s}^{-1}$

Normally, the original diffusion coefficient d lies between $0\,\mathrm{m}^2\,\mathrm{s}^{-1}$ and $1\,\mathrm{m}^2\,\mathrm{s}^{-1}$. Hence, diffusion dominated problems mainly result from one- and two-dimensional problems. □

Method 3.29 (FVM for the water quality model)**.** The FVM for the water quality model (Model 2.2) follows from Method 3.5 by using the following numerical flux:

$$\phi_{ij}(\overline{c}_i^n,\overline{c}_j^n) = \psi_{ij}^n-d_{ij}^n\frac{\overline{c}_j^n-\overline{c}_i^n}{|\underline{\mathbf{x}}_j^n-\underline{\mathbf{x}}_i^n|}\tag{3.7}$$

d_{ij}^n represents the total amount of diffusion form V_i^n to V_j^n. ψ_{ij}^n is a numerical flux function for the one-dimensional advection equation (see Section 3.2). The numerical flux $\breve{\phi}_{ik}$ for the boundary cells can be chosen analogously (and possibly different from ϕ_{ij}). □

An overview of the current schemes of WAQ can be found in Appendix A.

3.4 Flux correcting transport algorithm

Barth mentions in [1, p. 19] that there are no *linear* higher order methods that are monotonicity preserving. Thus, a linear method either is relatively inaccurate or generates spurious wiggles. The flux correcting transport algorithm attempts to combine the monotonicity preserving property of a first order scheme with the accuracy of a higher order scheme by means of a nonlinear limiter.

Method 3.30 (Flux Correcting Transport (FCT))**.** Consider a first order monotone numerical flux function $\hat{\phi}_{ij}$ and a higher order numerical flux function $\tilde{\phi}_{ij}$. The flux correcting transport method follows from Method 3.5 by using the following numerical flux function:

$$\phi_{ij}(\overline{c}_i^n, \overline{c}_j^n) \quad = \quad \hat{\phi}_{ij}(\overline{c}_i^n, \overline{c}_j^n) + l_{ij}^n \big(\underbrace{\tilde{\phi}_{ij}(\overline{c}_i^n, \overline{c}_j^n) - \hat{\phi}_{ij}(\overline{c}_i^n, \overline{c}_j^n)}_{=:\alpha_{ij}^n} \big)$$

α_{ij}^n can be interpreted as a correction term, which is limited by l_{ij}^n. □

Note that choosing $l_{ij}^n = 0$ corresponds to applying the first order method. On the other hand, setting $l_{ij}^n = 1$ is equivalent to using the higher order method. More sensible choices are described below.

Remark 3.31. Since α_{ij}^n often corrects the numerical diffusion of the first order flux, it is sometimes referred to as anti-diffusion. As a result, for the water quality model, it could be wiser to include the diffusion term in the correction term, so, to use:

$$\phi_{ij}(\overline{c}_i^n, \overline{c}_j^n) \quad = \quad \hat{\psi}_{ij}(\overline{c}_i^n, \overline{c}_j^n) + l_{ij}^n \big(\underbrace{\tilde{\psi}_{ij}(\overline{c}_i^n, \overline{c}_j^n) - \hat{\psi}_{ij}(\overline{c}_i^n, \overline{c}_j^n) - d_{ij}^n \frac{\overline{c}_j^n - \overline{c}_i^n}{|\underline{\mathbf{x}}_j^n - \underline{\mathbf{x}}_i^n|}}_{=:\alpha_{ij}^n} \big)$$

instead of:

$$\phi_{ij}(\overline{c}_i^n, \overline{c}_j^n) \quad = \quad \hat{\psi}_{ij}(\overline{c}_i^n, \overline{c}_j^n) - d_{ij}^n \frac{\overline{c}_j^n - \overline{c}_i^n}{|\underline{\mathbf{x}}_j^n - \underline{\mathbf{x}}_i^n|} + l_{ij}^n \big(\underbrace{\tilde{\psi}_{ij}(\overline{c}_i^n, \overline{c}_j^n) - \hat{\psi}_{ij}(\overline{c}_i^n, \overline{c}_j^n)}_{=:\alpha_{ij}^n} \big)$$

Here, $\hat{\psi}_{ij}$ is a monotone first order flux that corresponds to the advection term and $\hat{\psi}_{ij}$ is a higher order advection flux. (See Method 3.29 for other notational aspects.) Depending on the limiter, the strategy above can lead to a more lenient limiter and, as a result, to a scheme that is more accurate. □

The limiter that is used by WAQ is a generalised version of the one-dimensional limiter that was proposed by Boris & Book [2]. This limiter has been extended to the multi-dimensional case on a structured grid by Zalesak [13]. Below, it is further generalised for an unstructured grid. The limiter allows as much correction as possible, provided that is generates no new local extrema.

Method 3.32 (FCT a la Boris & Book)**.** This method results from Method 3.30 by using the following limiter:

1. Compute $\hat{\overline{c}}_i^n$ by means of Method 3.5 using $\phi_{ij} = \hat{\phi}_{ij}$.

2. Set $\alpha_{ij}^n = 0$ if[2]:

$$\alpha_{ij}^n(\hat{\overline{c}}_i^n - \hat{\overline{c}}_j^n) > 0 \quad \text{and} \quad \alpha_{ij}^n(\hat{\overline{c}}_j^n - \hat{\overline{c}}_{j+1}^n) > 0$$

or

$$\alpha_{ij}^n(\hat{\overline{c}}_i - \hat{\overline{c}}_j^n) > 0 \quad \text{and} \quad \alpha_{ij}^n(\hat{\overline{c}}_{i-1}^n - \hat{\overline{c}}_i^n) > 0$$

V_{i-1} is the volume next to V_i, opposite to V_j. Analogously, V_{j+1} is the volume next to V_j, opposite to V_i.

3. Construct an upper and a lower bound for $\overline{c}_i^n$ by using one of the following two options:

i. $\overline{c}_i^{\max} = \max_{j\in A_i\cup\{i\}}\{\hat{\overline{c}}_j^n\}$
$\overline{c}_i^{\min} = \min_{j\in A_i\cup\{i\}}\{\hat{\overline{c}}_j^n\}$

ii. $\overline{c}_i^{\max} = \max_{j\in A_i\cup\{i\}}\left\{\max\{\overline{c}_j^{n-1}, \hat{\overline{c}}_j^n\}\right\}$
$\overline{c}_i^{\min} = \min_{j\in A_i\cup\{i\}}\left\{\min\{\overline{c}_j^{n-1}, \hat{\overline{c}}_j^n\}\right\}$

4. The mass increase in cell V_i without the limiter reads:

$$\lambda_i^+ = \sum_{j\in A_i}(t_n - t_{n-1})|S_{ij}|\max\{0, -\alpha_{ij}^n\}$$

The allowed mass increase is, however:

$$\mu_i^+ = |V_i|\left(\overline{c}_i^{\max} - \hat{\overline{c}}_i^n\right)$$

Thus, the allowed fraction of mass increase is denoted by:

$$\nu_i^+ = \begin{cases} \min\{1, \frac{\mu_i^+}{\lambda_i^+}\} & \lambda_i^+ > 0 \\ 0 & \lambda_i^+ = 0 \end{cases}$$

Introduce analogue quantities for mass decrease:

$$\begin{aligned} \lambda_i^- &= \sum_{j\in A_i}(t_n - t_{n-1})|S_{ij}|\max\{0, \alpha_{ij}^n\} \\ \mu_i^- &= |V_i|\left(\hat{\overline{c}}_i^n - \overline{c}_i^{\min}\right) \\ \nu_i^- &= \begin{cases} \min\{1, \frac{\mu_i^-}{\lambda_i^-}\} & \lambda_i^- > 0 \\ 0 & \lambda_i^- = 0 \end{cases} \end{aligned}$$

5. The limiter now reads:

$$l_{ij}^n = \begin{cases} \min\{\nu_j^+, \nu_i^-\} & \alpha_{ij}^n \geq 0 \\ \min\{\nu_i^+, \nu_j^-\} & \alpha_{ij}^n < 0 \end{cases}$$ □

[2]This part is tricky in case of an unstructured grid. Nonetheless, storage of the neighbors of the neighbors is already implemented in WAQ. Moreover, in practice, the effect is minimal according to Zalesak [13, p. 342].

Another limiting strategy is based on smoothness. In smooth regions, the limiter is close to 1. In other regions, the limiter increases or decreases the slope.

Method 3.33. Let $h : \mathbb{R} \to \mathbb{R}$. This method results from Method 3.30 by using the following limiter l_{ij}^n:

$$\begin{aligned} \beta_{ij}^n &= \begin{cases} \frac{\bar{c}_i - \bar{c}_{i-1}}{\bar{c}_j - \bar{c}_i} & \text{if } \overline{\underline{\mathbf{u}}}_{ij}^n \cdot \mathbf{n}_{ij}^n \geq 0 \\ \frac{\bar{c}_{j+1} - \bar{c}_j}{\bar{c}_j - \bar{c}_i} & \text{if } \overline{\underline{\mathbf{u}}}_{ij}^n \cdot \mathbf{n}_{ij}^n < 0 \end{cases} \\ l_{ij}^n &= h\left(\beta_{ij}^n\right) \end{aligned}$$

V_{i-1} is the volume next to V_i, opposite to V_j. Analogously, V_{j+1} is the volume next to V_j, opposite to V_i. □

Example 3.34. The following choices for h lead to well-known examples of Method 3.33:

h	method
$h(\beta) = 0$	first order upwind
$h(\beta) = 1$	Lax-Wendroff
$h(\beta) = \beta$	Beam-Warming
$h(\beta) = \frac{1}{2}(1 + \beta)$	Fromm
$h(\beta) = \text{minmod}(\beta, 1)$	minmod
$h(\beta) = \max\{0, \min\{1, 2\beta\}, \min\{2, \beta\}\}$	superbee
$h(\beta) = \max\{0, \min\{\frac{1+\beta}{2}, 2, 2\beta\}\}$	MC
$h(\beta) = \frac{\beta + \lvert\beta\rvert}{1 + \lvert\beta\rvert}$	van Leer

in which the minmod function is defined as:

$$\text{minmod}(a, b) = \begin{cases} a & |a| < |b|, ab > 0 \\ b & |b| \leq |a|, ab > 0 \\ 0 & ab \leq 0 \end{cases}$$

□

More detailed information on the limiters above can be found in [10, Chapter 6]. Since they were originally designed for one-dimensional problems, it is not clear that they will also perform well for unstructured three-dimensional meshes.

3.5 Nonlinearity

In the implicit case, Method 3.5 generally leads to a coupled nonlinear system of equations. First of all, because p is a nonlinear term, which may depend nonlinearly on the concentrations of other substances. Furthermore, applying a limiter (Method 3.30) introduces another nonlinear term, which may be nondifferentiable. How is such a system to be dealt with in the implicit case?

Option one is to make use of a nonlinear solver (for more information, see, for instance, [3, Chapter 10]). There are two disadvantages of this strategy. First of all, the method is expensive, because a linear system has to be solved each iteration. Moreover, the Jacobian, which is ordinarily needed for nonlinear solvers,

is not defined if the function is nondifferentiable. An approximation of the Jacobian is hard to obtain on an unstructured grid. All together, using a nonlinear solver seems unattractive and will not be investigated any further in this report.

An alternative strategy is to approximate all implicit nonlinear terms explicitly, as described in the following method.

Method 3.35. This method consists of the following steps, involving linear systems only:

1. Compute $\hat{\overline{c}}_i^n$ by applying Method 3.5 with a first order monotone numerical flux function $\hat{\phi}_{ij}$, using $\hat{\overline{p}}_i^n \approx \overline{p}_i^{n-1}$.

2. Compute $\tilde{\overline{c}}_i^n$ by applying Method 3.5 with a higher order numerical flux function $\tilde{\phi}_{ij}$ using $\tilde{\overline{p}}_i^n \approx \overline{p}_i^{n-1}$.

3. Define:

$$\begin{aligned} \hat{\phi}_{ij}^n &= \hat{\phi}_{ij}(\hat{\overline{c}}_i^n, \hat{\overline{c}}_j^n) \\ \tilde{\phi}_{ij}^n &= \tilde{\phi}_{ij}(\tilde{\overline{c}}_i^n, \tilde{\overline{c}}_j^n) \end{aligned}$$

   and approximate the limiter l_{ij}^n explicitly by means of $\hat{\phi}_{ij}^n$, $\tilde{\phi}_{ij}^n$, $\hat{\overline{c}}_i^n$, and possibly $\tilde{\overline{c}}_i^n$. Then, the limited flux reads:

$$\phi_{ij}(\overline{c}_i^n, \overline{c}_j^n) = \hat{\phi}_{ij}^n + l_{ij}^n(\tilde{\phi}_{ij}^n - \hat{\phi}_{ij}^n)$$

4. Solve $\overline{c}_i^n$ by applying Method 3.5 with numerical flux $\phi_{ij}(\overline{c}_i^n, \overline{c}_j^n)$ using $\overline{p}_i^n \approx \overline{p}_i^{n-1}$. □

3.6 Conclusion

The solution to the water quality model can be approximated by the finite volume method. Because the grid is generally unstructured, the flux through a cell edge is approximated by a one-dimensional numerical flux function in the normal direction. Normally, a central difference approach is applied to the diffusion term. The time integration method is the θ-scheme. The nature of the total method is determined by the selected numerical flux function for the advection term and the applied value of θ. An implicit flux correcting transport scheme, in which nonlinear terms are approximated explicitly, might yield a good combination of speed and accuracy.

Chapter 4

Solution methods for linear systems

In Chapter 3, the finite volume method was discussed to solve Model 2.2. Since it is often needed to predict water quality several years ahead, large time steps are desirable. Therefore, implicit methods are preferable to explicit schemes, as the latter do not allow arbitrary time steps without becoming unstable. Implicit methods require the solution of many large sparse[1] linear systems. To obtain these solutions, efficient solvers are discussed in this chapter.

4.1 Direct methods

A direct method computes a theoretically exact solution using a finite number of operations. A useful measure for the amount of work is the number of floating point operations (*flops*).

Example 4.1. Consider a full matrix $A \in \mathbb{R}^{n\times n}$ and a vector $\underline{\mathbf{x}} \in \mathbb{R}^n$. Then the matrix vector product $A\underline{\mathbf{x}}$ needs $2n^2$ flops. □

4.1.1 Triangular matrices

Triangular systems are easily solved by means of backward or forward substitution.

Method 4.2 (Forward substitution)**.** Let $L \in \mathbb{R}^{n\times n}$ be a lower triangular matrix. *Forward substitution* solves the linear system $L\underline{\mathbf{x}} = \underline{\mathbf{b}}$ by means of the following algorithm:

1. for $i = 1, ..., n$:
2. $\quad x_i = b_i$
3. $\quad$ for $j = 1, ..., i-1$:
4. $\quad\quad x_i = x_i - l_{ij}x_j$
5. $\quad$ end
6. end

[1] A matrix is sparse if it contains 'many' zero elements

For a full matrix, the computational costs amount to n^2 flops. In case the matrix has at most q nonzero off-diagonal elements per row, approximately $2qn$ flops are needed. □

Method 4.3 (Backward substitution)**.** Let $U \in \mathbb{R}^{n\times n}$ be an upper triangular matrix. *Backward substitution* solves the linear system $U\underline{\mathbf{x}} = \underline{\mathbf{b}}$ by means of the following algorithm:

1. for $i = n, ..., 1$:
2. $\quad x_i = b_i$
3. $\quad$ for $j = i + 1, ..., n$:
4. $\quad\quad x_i = x_i - u_{ij}x_j$
5. $\quad$ end
6. $\quad x_i = \frac{x_i}{u_{ii}}$
7. end

For a full matrix, the computational costs amount to n^2 flops. In case the matrix has at most q nonzero off-diagonal elements per row, approximately $2qn$ flops are needed. □

4.1.2 General square matrices

A general square system can be solved by means of Gaussian elimination. This method reduces a linear system to two triangular systems, which can be solved by backward or forward substitution. The triangular systems are obtained by constructing an LU factorisation of the matrix.

Definition 4.4 (LU factorisation)**.** An *LU factorisation* of a matrix A consists of a unit lower triangular matrix L and an upper triangular matrix U such that:

$$A \quad = \quad LU$$ □

Method 4.5 (LU factorisation)**.** The following algorithm generates an LU factorisation for a matrix $A \in \mathbb{R}^{n\times n}$, provided that the pivots, u_{kk}, are nonzero.

1. for $i = 1, ..., n$:
2. $\quad w = a_{i*}$
3. $\quad$ for $k = 1, ..., i - 1$:
4. $\quad\quad w_k = \frac{w_k}{u_{kk}}$
5. $\quad\quad w = w - w_k u_{k*}$
6. $\quad$ end
7. $\quad l_{ij} = w_j$ for $j = 1, ..., i - 1$
8. $\quad u_{ij} = w_j$ for $j = i, ..., n$
9. end

a_{i*} denotes row i of A. For a full matrix, the computational costs of the factorisation amount to $\frac{2}{3}n^3$ flops. □

More detailed information about LU factorizations can be found in [8, Section 3.2].

Method 4.6 (Gaussian elimination)**.** *Gaussian elimination* solves an $n \times n$ linear system $A\underline{\mathbf{x}} = \underline{\mathbf{b}}$ in three steps:

1. Construct an LU factorisation of A. This can by done by means of Method 4.5.
2. Solve $\underline{\mathbf{y}}$ from $L\underline{\mathbf{y}} = \underline{\mathbf{b}}$ by means of forward substitution (Method 4.2)
3. Solve $\underline{\mathbf{x}}$ from $U\underline{\mathbf{x}} = \underline{\mathbf{y}}$ by applying backward substitution (Method 4.3) □

A disadvantage of Gaussian elimination for sparse matrices is that L and U are generally less sparse than A. This effect is called fill-in.

Definition 4.7 (Fill-in)**.** The fill-in of a matrix consists of those entries which change from an initial zero to a nonzero value during the execution of an algorithm. □

Theorem 4.8. *Consider a matrix A with lower bandwidth q_l and upper bandwidth q_u. Let $A = LU$ be an LU factorisation. Then, L has lower bandwidth q_l and U has upper bandwidth q_u. Moreover, if $n \gg q_l, q_u$, the costs of the LU factorisation are approximately $2nq_lq_u$ flops.*
Proof. See [8, Theorem 4.3.1 and p. 153]. ■

4.2 Iterative Methods

An alternative for Gaussian elimination is provided by iterative methods. These methods iteratively improve an initial solution estimation.

Method 4.9 (Iterative method: general form)**.** Consider an $n \times n$ linear system $A\underline{\mathbf{x}} = \underline{\mathbf{b}}$. An *iterative method* consists of the following steps:

1. Choose an initial guess of the solution, $\underline{\mathbf{x}}_0 \in \mathbb{R}^n$.
2. Set $k = 1$. Choose an improved approximation of the solution, $\underline{\mathbf{x}}_k$, and set $k = k + 1$, until a certain termination criterion is met.
3. Approximate the solution of $A\underline{\mathbf{x}} = \underline{\mathbf{b}}$ according to $\underline{\mathbf{x}} \approx \underline{\mathbf{x}}_k$. □

Definition 4.10 (Convergent iterative method)**.** Method 4.9 is *convergent* if

$$\|\underline{\mathbf{x}}_k - \underline{\mathbf{x}}\| \to 0$$ □

Remark 4.11 (A good termination criterion)*.* A good termination criterion has the following properties:

1. It is scaling invariant. This means that the number of iterations for $\alpha A\underline{\mathbf{x}} = \alpha\underline{\mathbf{b}}$ is independent of $\alpha \in \mathbb{R}$.
2. The number of iterations should not be independent of the initial estimation $\underline{\mathbf{x}}_0$; a better initial guess should lead to a smaller number of iterations.
3. It provides an upper bound for the relative error $\frac{\|\underline{\mathbf{x}} - \underline{\mathbf{x}}_k\|_2}{\|\underline{\mathbf{x}}\|_2}$.

An example of a good termination criterion, satisfying the properties above, is

$$\frac{\|\underline{\mathbf{b}} - A\underline{\mathbf{x}}_k\|_2}{\|\underline{\mathbf{b}}\|_2} \leq \epsilon$$ □

4.2.1 Linear fixed point iteration

Linear fixed point iteration determines the estimates of the solution by means of a matrix splitting of A.

Definition 4.12 (Matrix splitting)**.** A *matrix splitting* of a matrix A consists of matrices M and N such that

$$A = M - N$$ □

Method 4.13 (Linear fixed point iteration)**.** *Linear fixed point iteration* follows from Method 4.9 by updating $\underline{\mathbf{x}}_k$ according to a matrix splitting $A = M - N$, with M nonsingular. The following strategies are equivalent:

$$M\underline{\mathbf{x}}_k = N\underline{\mathbf{x}}_{k-1} + \underline{\mathbf{b}} \quad (4.1)$$
$$\underline{\mathbf{x}}_k = M^{-1}N\underline{\mathbf{x}}_{k-1} + M^{-1}\underline{\mathbf{b}} \quad (4.2)$$
$$\underline{\mathbf{x}}_k = \underline{\mathbf{x}}_{k-1} + M^{-1}(\underline{\mathbf{b}} - A\underline{\mathbf{x}}_{k-1}) \quad (4.3)$$

□

Example 4.14. Consider a matrix A. Define a diagonal matrix D, a strictly lower triangular matrix L, and a strictly upper triangular matrix U, such that $A = L + D + U$. Moreover, let $\omega \in \mathbb{R}$ be a constant. The following matrix splittings $A = M - N$ lead to well-known methods:

M	N	Method
D	$-L - U$	Gauss-Jacobi (GJ)
$D + L$	$-U$	Gauss-Seidel (GS)
$D + U$	$-L$	Backward Gauss-Seidel
$D + \omega L$	$(\omega - 1)L - U$	Successive Over-Relaxation (SOR)

□

Theorem 4.15 (Convergence of linear fixed point iteration)**.** *Method 4.13 converges for any starting vector* $\underline{\mathbf{x}}_0$*, if*

$$\max\{|\lambda| : \lambda \text{ eigenvalue of } M^{-1}N\} < 1$$

Proof. See [8, Theorem 10.1.1]. ■

Method 4.16. Method 4.13 can be applied twice, to obtain a new iterative scheme:

1. Perform Method 4.13 with a matrix splitting $A = M_1 - N_1$ to obtain $\underline{\mathbf{x}}_k^*$:

$$M_1\underline{\mathbf{x}}_k^* = N_1\underline{\mathbf{x}}_{k-1} + \underline{\mathbf{b}}$$

2. Apply Method 4.13 once more using another matrix splitting $A = M_2 - N_2$ to acquire $\underline{\mathbf{x}}_k$:

$$M_2\underline{\mathbf{x}}_k = N_2\underline{\mathbf{x}}_k^* + \underline{\mathbf{b}}$$ □

Proposition 4.17. *Method 4.16 is equivalent to Method 4.13 for the matrix splitting $A = M - (M - A)$, with*

$$M \quad = \quad M_1 \left(M_1 + N_2\right)^{-1} M_2$$

Proof. Since (4.1) and (4.3) are equivalent, Method 4.18 can be rewritten to obtain:

$$\begin{aligned} \underline{\mathbf{x}}_k^* &:= \underline{\mathbf{x}}_{k-1} + M_1^{-1}\left(\underline{\mathbf{b}} - A\underline{\mathbf{x}}_{k-1}\right) \\ \underline{\mathbf{x}}_k &= \underline{\mathbf{x}}_k^* + M_2^{-1}\left(\underline{\mathbf{b}} - A\underline{\mathbf{x}}_k^*\right) \end{aligned}$$

$\Rightarrow$

$$\begin{aligned} \underline{\mathbf{x}}_k &= \underline{\mathbf{x}}_{k-1} + M_1^{-1}\left(\underline{\mathbf{b}} - A\underline{\mathbf{x}}_{k-1}\right) \\ &\quad + M_2^{-1}\Big(\underline{\mathbf{b}} - A\big(\underline{\mathbf{x}}_{k-1} + M_1^{-1}(\underline{\mathbf{b}} - A\underline{\mathbf{x}}_{k-1})\big)\Big) \\ &= \underline{\mathbf{x}}_{k-1} + M_1^{-1}\left(\underline{\mathbf{b}} - A\underline{\mathbf{x}}_{k-1}\right) \\ &\quad + M_2^{-1}\big(\underline{\mathbf{b}} - A\underline{\mathbf{x}}_{k-1} - AM_1^{-1}(\underline{\mathbf{b}} - A\underline{\mathbf{x}}_{k-1})\big) \\ &= \underline{\mathbf{x}}_{k-1} + \left(M_1^{-1} + M_2^{-1}(I - AM_1^{-1})\right)\left(\underline{\mathbf{b}} - A\underline{\mathbf{x}}_{k-1}\right) \\ &= \underline{\mathbf{x}}_{k-1} + \Big(M_1^{-1} + M_2^{-1}\big(I - (M_2 - N_2)M_1^{-1}\big)\Big)\left(\underline{\mathbf{b}} - A\underline{\mathbf{x}}_{k-1}\right) \\ &= \underline{\mathbf{x}}_{k-1} + \Big(I + M_2^{-1}\big(M_1 - (M_2 - N_2)\big)\Big)M_1^{-1}\left(\underline{\mathbf{b}} - A\underline{\mathbf{x}}_{k-1}\right) \\ &= \underline{\mathbf{x}}_{k-1} + M_2^{-1}(M_1 + N_2)M_1^{-1}\left(\underline{\mathbf{b}} - A\underline{\mathbf{x}}_{k-1}\right) \\ &= \underline{\mathbf{x}}_{k-1} + \Big(\underbrace{M_1(M_1 + N_2)^{-1}M_2}_{=:M}\Big)^{-1}\left(\underline{\mathbf{b}} - A\underline{\mathbf{x}}_{k-1}\right) \end{aligned}$$

Applying the equivalence of (4.1) and (4.3) once more completes the proof. ■

An example of Method 4.16 is Symmetric Gauss-Seidel.

Method 4.18 (Symmetric Gauss-Seidel)**.** *Symmetric Gauss-Seidel* follows from method 4.16 by applying one step of Gauss-Seidel ($M_1 = D + L$, $N_1 = -U$), followed by one step of Backward Gauss-Seidel ($M_2 = D + U$, $N_2 = -L$). □

4.2.2 Krylov methods

A large category of iterative schemes is formed by the Krylov methods. These are based on a so-called Krylov space, which is defined below.

Definition 4.19 (Krylov space)**.** Let $A \in \mathbb{R}^{n\times n}$ and $\underline{\mathbf{r}} \in \mathbb{R}^n$. A *Krylov space* is of the form:

$$K^k(A, \underline{\mathbf{r}}) \quad = \quad \text{span}\left\{\underline{\mathbf{r}}, A\underline{\mathbf{r}}, ..., A^{k-1}\underline{\mathbf{r}}\right\} \qquad \square$$

The link between Krylov methods and linear fixed point iteration becomes clear from the following proposition.

Proposition 4.20. *Let $\underline{\mathbf{x}}_k$ result from linear fixed point iteration (Method 4.13):*

$$\underline{\mathbf{x}}_k \quad = \quad \underline{\mathbf{x}}_{k-1} + M^{-1}\underbrace{\left(\underline{\mathbf{b}} - A\underline{\mathbf{x}}_{k-1}\right)}_{\underline{\mathbf{r}}_{k-1}}$$

Then,

$$\underline{\mathbf{x}}_k \in \underline{\mathbf{x}}_0 + \underbrace{K^k(M^{-1}A, M^{-1}\underline{\mathbf{r}}_0)}_{=:K^k} \quad \textit{for all } k \geq 1$$

Proof. First of all, note that the statement is true for $k = 1$:

$$\underline{\mathbf{x}}_1 \;=\; \underline{\mathbf{x}}_0 + M^{-1}\underline{\mathbf{r}}_0 \in \underline{\mathbf{x}}_0 + K^1$$

Now, suppose that $\underline{\mathbf{x}}_k \in \underline{\mathbf{x}}_0 + K^k$ $(k \geq 1)$. The residual $\underline{\mathbf{r}}_k$ can be expressed in $\underline{\mathbf{r}}_0$, according to:

$$\begin{aligned}
\underline{\mathbf{r}}_k &= \underline{\mathbf{b}} - A\underline{\mathbf{x}}_k \\
&= \underline{\mathbf{b}} - A(\underline{\mathbf{x}}_{k-1} + M^{-1}\underline{\mathbf{r}}_{k-1}) \\
&= \underline{\mathbf{r}}_{k-1} - AM^{-1}\underline{\mathbf{r}}_{k-1} \\
&= (I - AM^{-1})\underline{\mathbf{r}}_{k-1} \\
&= (I - AM^{-1})^k\underline{\mathbf{r}}_0
\end{aligned}$$

So,

$$\underline{\mathbf{x}}_{k+1} = \underline{\mathbf{x}}_k + M^{-1}\underline{\mathbf{r}}_k = \underbrace{\underline{\mathbf{x}}_k}_{\in \underline{\mathbf{x}}_0 + K^k} + \underbrace{M^{-1}(I - AM^{-1})^k\underline{\mathbf{r}}_0}_{\in K^{k+1}} \in \underline{\mathbf{x}}_0 + K^{k+1}$$

■

Method 4.21 (Krylov method)**.** A *Krylov method* follows from Method 4.9 by choosing $\underline{\mathbf{x}}_k$ such that:

1. $\underline{\mathbf{x}}_k \in \underline{\mathbf{x}}_0 + K^k(A, \underline{\mathbf{r}}_0)$
2. $\underline{\mathbf{r}}_k = \underline{\mathbf{b}} - A\underline{\mathbf{x}}_k \perp L^k$

Here, L^k is a k-dimensional subspace of $\mathbb{R}^n$. □

The nature of a Krylov method is mainly determined by two aspects. Of course, the choice of L^k plays an important role. Additionally, there are several ways to construct a basis for $K^k(A, \underline{\mathbf{r}}_0)$. Three choices are listed below, including the methods resulting from them.

1. Arnoldi provides an orthonormal basis $V = \{\underline{\mathbf{v}}_1, ..., \underline{\mathbf{v}}_k\}$ for $K^k(A, \underline{\mathbf{v}}_1)$ (variants: Arnoldi-Modified Gram-Schmidt and Householder Arnoldi)
   - $L^k = K^k(A, \underline{\mathbf{r}}_0)$: Full Orthogonalization Method (FOM) (variants: Restarted FOM (FOM(k), Incomplete Orthogonalisation Method (IOM), Direct IOM (DIOM))
   - $L^k = AK^k(A, \underline{\mathbf{r}}_0)$: General Minimal RESidual (GMRES) (variants: Restarted GMRES (GMRES(k)), Quasi-GMRES (QGMRES), Direct QGMRES (DQGMRES))
2. Lanczos is like Arnoldi, but only applicable to symmetric matrices (variant: Direct Lanczos (D-Lanczos))
   - $L^k = K^k(A, \underline{\mathbf{r}}_0)$: Conjugate Gradient (GG) (for positive definite matrices) (variant: CG-Three-term recurrence variant (for positive definite matrices))

- $L^k = AK^k(A, \underline{\mathbf{r}}_0)$: Conjugate Residual (CR) (for positive definite hermitian matrices)

3. Lanczos Biothogonalisation (BiLanczos) computes a basis $V = \{\underline{\mathbf{v}}_1, ..., \underline{\mathbf{v}}_k\}$ for $K^k(A, \underline{\mathbf{v}}_1)$ and a basis $W = \{\underline{\mathbf{w}}_1, ..., \underline{\mathbf{w}}_k\}$ for $K^k(A^T, \underline{\mathbf{w}}_1)$, such that $\underline{\mathbf{v}}_i \perp \underline{\mathbf{w}}_j$ for all $i, j = 1, ..., k$

   - $L^k = K^k(A^T, \underline{\mathbf{r}}_0)$: BiConjugate Gradient (BCG) (variants: Conjugate Gradient Squared (CGS), Biconjugate Gradient Stabilized (BICGSTAB)) and Quasi Minimal Residual (QMR) (variant: Transpose Free QMR (TFQMR)

In the next sections, Arnoldi and GMRES will be considered in more detail. For more information about other Krylov methods, see [11, Chapter 6 and 7].

Arnoldi

The Arnoldi method constructs an orthonormal basis for a Krylov space with the help of Gram-Schmidt.

Method 4.22 (Arnoldi)**.** The *Arnoldi* method constructs an orthonormal basis $\{\underline{\mathbf{v}}_1, ..., \underline{\mathbf{v}}_k\}$ for the Krylov space $K^k(A, \underline{\mathbf{r}})$ by means of the following algorithm:

1. $\underline{\mathbf{v}}_1 = \frac{1}{\|r\|_2}\underline{\mathbf{r}}$
2. for $j = 1, .., k$:
3. for $i = 1, ..., j$: $h_{ij} = \left(A\underline{\mathbf{v}}_j\right)^T \underline{\mathbf{v}}_i$
4. $\underline{\mathbf{v}}_{j+1} = \underline{\mathbf{v}}_{j+1} - \sum_{i=1}^{j} h_{ij}\underline{\mathbf{v}}_i$
5. $h_{j+1,j} = \|\underline{\mathbf{v}}_{j+1}\|_2$
6. if $h_{j+1,j} = 0$: stop
7. $\underline{\mathbf{v}}_{j+1} = \frac{1}{h_{j+1,j}}\underline{\mathbf{v}}_{j+1}$
8. end □

The following variant of the algorithm above uses Modified Gram-Schmidt instead of Gram-Schmidt.

Method 4.23 (Arnoldi-Modified Gram-Schmidt)**.** The *Arnoldi-Modified Gram-Schmidt* method constructs an orthonormal basis $\{\underline{\mathbf{v}}_1, ..., \underline{\mathbf{v}}_k\}$ for the Krylov space $K^k(A, \underline{\mathbf{r}})$ by means of the following algorithm.

1. $\underline{\mathbf{v}}_1 = \frac{1}{\|r\|_2}\underline{\mathbf{r}}$
2. for $j = 1, .., k$:
3. $\underline{\mathbf{v}}_{j+1} = A\underline{\mathbf{v}}_j$
4. for $i = 1, ..., j$:
5. $h_{ij} = \underline{\mathbf{v}}_{j+1}^T\underline{\mathbf{v}}_i$
6. $\underline{\mathbf{v}}_{j+1} = \underline{\mathbf{v}}_{j+1} - h_{ij}\underline{\mathbf{v}}_i$
7. end
8. $h_{j+1,j} = \|\underline{\mathbf{v}}_{j+1}\|_2$
9. if $h_{j+1,j} = 0$: stop
10. $\underline{\mathbf{v}}_{j+1} = \frac{1}{h_{j+1,j}}\underline{\mathbf{v}}_{j+1}$
11. end □

In theory, the results of both algorithms are the same. In practice, Arnoldi-Modified Gram-Schmidt is less sensitive to round-off errors. Another variant is Householder Arnoldi [11, p. 149], which is even more reliable, but also more expensive.

General minimal residual method

The general minimal residual method [12] is a Krylov method that uses $L^k = AK^k(A, \underline{\mathbf{r}}_0)$. As its name already indicates, it is based on minimizing the residual $\underline{\mathbf{r}}_k = \underline{\mathbf{b}} - A\underline{\mathbf{x}}_k$. Each step, $\underline{\mathbf{x}}_k$ is chosen such that:

$$\begin{aligned} \underline{\mathbf{x}}_k &= \underline{\mathbf{x}}_0 + \arg \min_{\underline{\mathbf{z}} \in K^k(A, \underline{\mathbf{r}}_0)} \| \underline{\mathbf{b}} - A\left(\underline{\mathbf{x}}_0 + \underline{\mathbf{z}}\right) \|_2 \\ &= \underline{\mathbf{x}}_0 + \arg \min_{\underline{\mathbf{z}} \in K^k(A, \underline{\mathbf{r}}_0)} \| \underline{\mathbf{r}}_0 - A\underline{\mathbf{z}} \|_2 \end{aligned}$$

Since this involves a minimization problem that is not trivial to solve, it will be rewritten.

Proposition 4.24. *Let $\underline{\mathbf{v}}_j$ (for j=1,...,k+1) and h_{ij} (for $i = 1, ..., k$ and $j = 1, ..., k+1$) result from applying (a variant of) Arnoldi to $K^k(A, \underline{\mathbf{r}}_0)$. Define $V_k \in \mathbb{R}^{n \times k}$ and $H_k \in \mathbb{R}^{k+1 \times k}$ (also known as the* Hessenberg matrix*) such that:*

$$V_k = \begin{bmatrix} \underline{\mathbf{v}}_1 & \dots & \underline{\mathbf{v}}_k \end{bmatrix} \tag{4.4}$$

$$H_k = \begin{bmatrix} h_{11} & \dots & h_{1k} \\ h_{21} & \dots & h_{2k} \\ & \ddots & \vdots \\ & & h_{k+1,k} \end{bmatrix} \tag{4.5}$$

Define $\underline{\mathbf{y}}_k$ as the solution of the following linear least squares problem:

$$\underline{\mathbf{y}}_k = \arg \min_{\underline{\mathbf{y}} \in \mathbb{R}^k} \left\| \|\underline{\mathbf{r}}_0\|_2 \underline{\mathbf{e}}_1 - H_k \underline{\mathbf{y}} \right\|_2 \tag{4.6}$$

Then

$$\arg \min_{\underline{\mathbf{z}} \in K^k(A, \underline{\mathbf{r}}_0)} \|\underline{\mathbf{r}}_0 - A\underline{\mathbf{z}}\|_2 = V_k \underline{\mathbf{y}}_k$$

Proof. See [11, Section 6.5.1]. ■

In order to determine $\underline{\mathbf{y}}_k$ in (4.6) efficiently, H_k will be transformed by what are called *Givens rotations* in order to achieve the following structure:

$$H_k^{(i)} = \begin{bmatrix} h_{11}^{(i)} & \dots & \dots & \dots & h_{1k}^{(i)} \\ & \ddots & & & \vdots \\ & & h_{i+1,i+1}^{(i)} & & \vdots \\ & & h_{i+2,i+1}^{(i)} & \ddots & \vdots \\ & & & \ddots & h_{kk}^{(i)} \\ & & & & h_{k+1,k}^{(i)} \end{bmatrix}$$

Note that in particular $H_k^{(k)}$ will have a favorable structure.

Proposition 4.25. *Let H_k and $\underline{\mathbf{r}}_0$ be as in Proposition 4.24. Define:*

$$H_k^{(0)} = H_k \tag{4.7}$$

Let I_i be the $i \times i$ identity matrix. Introduce Givens rotations $\Omega_i \in \mathbb{R}^{k+1 \times k+1}$ $(i = 1, ..., k)$ according to:

$$\Omega_i = \begin{bmatrix} I_{i-1} & & & \\ & c_i & s_i & \\ & -s_i & c_i & \\ & & & I_{k-i} \end{bmatrix} \tag{4.8}$$

$$c_i = \frac{h_{i+1,i}}{\sqrt{\left(h_{i,i}^{(i-1)}\right)^2 + h_{i+1,i}^2}} \tag{4.9}$$

$$s_i = \frac{h_{i,i}^{(i-1)}}{\sqrt{\left(h_{i,i}^{(i-1)}\right)^2 + h_{i+1,i}^2}} \tag{4.10}$$

Apply the Givens rotations to H_k and $\|\underline{\mathbf{r}}_0\|_2 \underline{\mathbf{e}}_1$ to obtain:

$$H_k^{(i)} = \Omega_i ... \Omega_1 H_k \tag{4.11}$$

$$\underline{\mathbf{g}}^{(i)} = \Omega_i ... \Omega_1 \|\underline{\mathbf{r}}_0\|_2 \underline{\mathbf{e}}_1 \tag{4.12}$$

Then, the following two statements hold:

1. *The solution of the linear least square problem (4.6) satisfies:*

$$\hat{H}_k^{(k)} \underline{\mathbf{y}}_k = \hat{\underline{\mathbf{g}}}^{(k)}$$

   in which $\hat{H}_k^{(k)}$ and $\hat{\underline{\mathbf{g}}}^{(k)}$ result from $H_k^{(k)}$ and $\underline{\mathbf{g}}^{(k)}$ by deleting their last rows. Note that $\hat{H}_k^{(k)}$ is an upper triangular matrix, so the system is easily solved by means of backward substitution (Method 4.3).

2. *The 2-norm of the residual, $\|\underline{\mathbf{b}} - A\underline{\mathbf{x}}_k\|_2$, is given by the last element of $\underline{\mathbf{g}}^{(k)}$. This result is convenient for the implementation of a stopping criterion.*

Proof. See [11, Proposition 6.9] ■

Putting it all together:

Method 4.26 (General Minimal RESidual method (GMRES))**.** The *general minimal residual method* follows from Method 4.9 by using the following strategy to obtain $\underline{\mathbf{x}}_k$:

1. Compute H_k and V_k by applying (a variant of) Arnoldi to $K^k(A, \underline{\mathbf{r}}_0)$ and using equations (4.4) and (4.5).
2. Determine $H_k^{(k)}$ and $\underline{\mathbf{g}}^{(k)}$ using equations (4.7)-(4.12).
3. Calculate $\underline{\mathbf{y}}_k$ by means of the first statement of Proposition 4.25.
4. $\underline{\mathbf{x}}_k = \underline{\mathbf{x}}_0 + V_k \underline{\mathbf{y}}_k$.

Note that $\underline{\mathbf{y}}_k$ and $\underline{\mathbf{x}}_k$ only need to be computed after the termination criterion has been reached. □

Theorem 4.27 (Convergence of GMRES)**.** *Consider a diagonalizable matrix A with eigenvalues $\lambda_1, ..., \lambda_n$ and corresponding eigenvectors $\underline{\mathbf{v}}_1, ..., \underline{\mathbf{v}}_n$. So, $V = [\underline{\mathbf{v}}_1 \dots \underline{\mathbf{v}}_n]$ is an invertible matrix. Then, the relative error in step k of GMRES (Method 4.26) satisfies:*

$$\frac{\|\underline{\mathbf{b}} - A\underline{\mathbf{x}}_k\|_2}{\|\underline{\mathbf{b}} - A\underline{\mathbf{x}}_0\|_2} \leq \|V\|_2\|V^{-1}\|_2 \min_{p\in\mathbb{P}_k, p(0)=1} \max_{i\{1,\dots,n\}} |p(\lambda_i)|$$

If, moreover, all eigenvalues are contained in an ellipse $E \subset \mathbb{C}$, excluding the origin and having center $c \in \mathbb{C}$, focal distance $d \in \mathbb{C}$, and semi major axis $a \in \mathbb{C}$ (see Figure 4.1 for an example), then, the relative error satisfies:

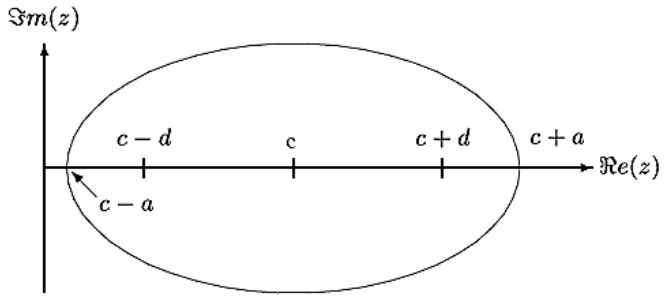
$$\begin{aligned}\frac{\|\underline{\mathbf{b}} - A\underline{\mathbf{x}}_k\|_2}{\|\underline{\mathbf{b}} - A\underline{\mathbf{x}}_0\|_2} &\leq \|V\|_2\|V^{-1}\|_2 \left|\frac{p_k\left(\frac{a}{d}\right)}{p_k\left(\frac{c}{d}\right)}\right| \\ &\approx \|V\|_2\|V^{-1}\|_2 \left|\frac{a+\sqrt{a^2-d^2}}{c+\sqrt{c^2-d^2}}\right|^k\end{aligned}$$

Here, $p_k : \mathbb{C} \to \mathbb{C}$ is the complex Chebychev polynomial*, which can be defined recursively according to:*

$$\begin{aligned}p_1(z) &= 1 \\ p_2(z) &= z \\ p_k(z) &= 2zp_k(z) - p_{k-1}(z) \qquad (k \geq 2)\end{aligned}$$

Proof. See [11, Proposition 6.15, Corollary 6.1, and (6.100)] ■

Figure 4.1: Example of an ellipse for $c, d, a \in \mathbb{R}$



4.3 Conclusion

If an implicit FVM is used to solve the water quality model, many large sparse linear systems need to be solved. For such systems, iterative methods are more suitable than direct methods. At present, GMRES is implemented in WAQ. This Krylov method converges faster as the eigenvalues are more clustered.

Chapter 5

Preconditioning

In practice, iterative methods often have an unsatisfactory convergence speed. A popular way to deal with this problem is enhancing the spectrum by means of preconditioning, which is discussed in this chapter.

5.1 Basic preconditioning

Preconditioning transforms a linear system into an equivalent system that can be handled better by an iterative method.

Method 5.1 (Preconditioning: general form)**.** Consider an $n \times n$ linear system $A\underline{\mathbf{x}} = \underline{\mathbf{b}}$. *Preconditioning* consists of the following steps:

1. Choose invertible matrices $P_l, P_r \in \mathbb{R}^{n \times n}$, the preconditioners.
2. Solve $\underline{\mathbf{y}}$ from:

$$P_l^{-1} A P_r^{-1} \underline{\mathbf{y}} \quad = \quad P_l^{-1} \underline{\mathbf{b}}$$

3. Determine $\underline{\mathbf{x}}$ according to:

$$\underline{\mathbf{x}} \quad = \quad P_r^{-1} \underline{\mathbf{y}}$$

□

Remark 5.2. If $P_r = I$, one speaks of *left preconditioning*. Similarly, choosing $P_l = I$ results in *right preconditioning*. Usually, the termination criterion of an iterative method is based on the residual norm $\|\underline{\mathbf{r}}_k\| = \|\underline{\mathbf{b}} - A\underline{\mathbf{x}}_k\|$. Preconditioning translates the residual norm to $\|P_l^{-1}\underline{\mathbf{r}}_k\|$. For this reason, right preconditioning is often preferred. □

Remark 5.3 (Recycling). In general, it is more costly to compute a preconditioner than to solve the preconditioned system. Therefore, it might be efficient to recycle the preconditioner, i.e. using the same preconditioner in multiple time steps. If the matrix does not vary much in time, the preconditioner hopefully remains effective. □

Good preconditioners at least guarantee that:

1. P_l and P_r can be determined inexpensively;

2. the spectrum[1] of $P_l^{-1}AP_r^{-1}$ is favorable with respect to convergence[2];
3. $P_l^{-1}\underline{\mathbf{v}}$ and $P_r^{-1}\underline{\mathbf{v}}$ can be computed at low cost ($\underline{\mathbf{v}} \in \mathbb{R}^n$).

What are suitable preconditioners? This will be treated in the following sections.

5.2 Preconditioners based on matrix splitting

Matrix splitting gives rise to a class of preconditioners that are relatively simple to construct.

Method 5.4 (Preconditioners based on matrix splitting)**.** There are two ways to derive preconditioners from a matrix splitting $A = M - N$:

1. $P_l = M$
   $P_r = I$
2. $P_l = I$
   $P_r = M$

The construction costs are 0 flops! □

Example 5.5 (Symmetric GS preconditioner). An example of the preconditioner above is the symmetric GS preconditioner, which is currently used in WAQ to speed up GMRES (schemes 15 and 16). Remembering Method 4.18 and Proposition 4.17 results in:

$$M = M_1 \left(M_1 + N_2\right)^{-1} M_2 = (D + L)D^{-1}(D + U) \qquad \square$$

The symmetric GS preconditioner seems to be inadequate for diffusion dominated problems. This is illustrated in Figure 5.1, in which the relative residual $\frac{\underline{\mathbf{b}} - A\underline{\mathbf{x}}_k}{\|\underline{\mathbf{b}}\|_2}$ is plotted for each iteration k for a two-dimensional problem. The blue line corresponds to the actual diffusion dominated case in which the diffusion coefficient has been increased by $10\,\mathrm{m}^2\,\mathrm{s}$ (see Remark 3.28). The red line is the result of neglecting this amount of extra diffusion. Indeed, the current preconditioning seems unsuitable for diffusion dominated problems.

5.3 Preconditioners based on an incomplete LU factorisation

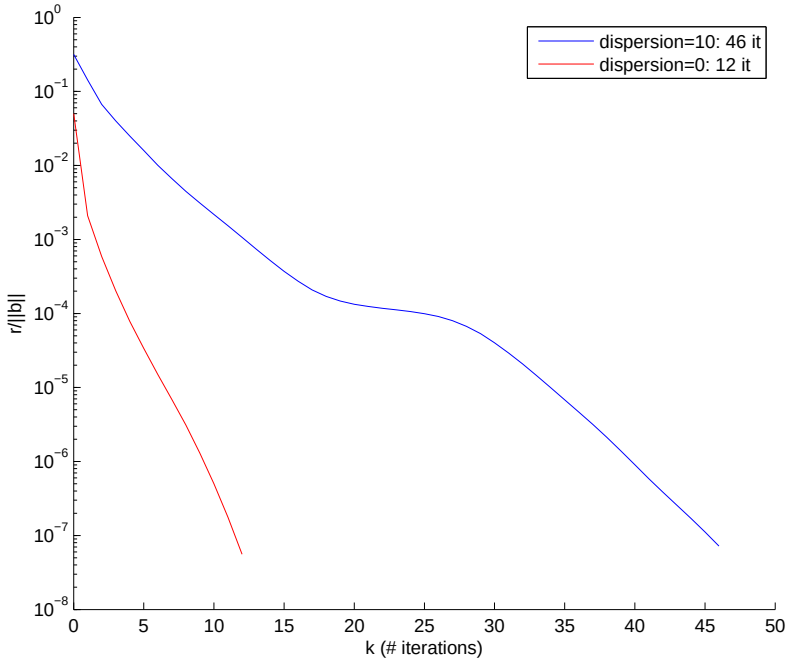
As mentioned before, LU factorisations are inefficient for sparse matrices (due to fill-in), which is why iterative methods are essential in the first place. However, approximate factorisations often result in powerful preconditioners.

Definition 5.6 (Incomplete LU factorisation)**.** An *incomplete LU factorisation* of a matrix A consists of a unit lower triangular matrix $\tilde{L}$, an upper triangular matrix $\tilde{U}$, and a matrix R such that:

$$A = \tilde{L}\tilde{U} - R \qquad \square$$

[1] set of eigenvalues
[2] Remember Theorem 4.27 and 4.15

Figure 5.1: The effect of diffusion dominance on the convergence of GMRES



Method 5.7 (Preconditioners based on an incomplete LU factorisation)**.** There are three ways to derive preconditioners from an incomplete LU factorisation:

1. $P_l = \tilde{L}\tilde{U}$
   $P_r = I$
2. $P_l = I$
   $P_r = \tilde{L}\tilde{U}$
3. $P_l = \tilde{L}$
   $P_r = \tilde{U}$ □

In the sections hereafter, several algorithms to compute an incomplete LU factorisation will be discussed.

5.3.1 Incomplete LU threshold

The incomplete LU threshold method provides a basic strategy to compute an incomplete LU factorisation. The algorithm results from adding two dropping rules to the algorithm that computes an ordinary LU factorisation (see Method 4.5). A dropping rule sets an element equal to zero if it satisfies certain criteria. To put it more bluntly: If you don't want to compute it, discard it.

Method 5.8 (Incomplete LU Threshold (ILUT))**.** The *incomplete LU threshold* algorithm computes an incomplete LU factorisation $A = \tilde{L}\tilde{U} - R$ by means of the following algorithm, provided that the pivots, $\tilde{u}_{kk}$, are nonzero:

1. for $i = 1, ..., n$:
2. $w := a_{i*}$
3. for $k = 1, ..., i-1$:
4. $w_k = \frac{w_k}{\tilde{u}_{kk}}$
5. Apply a dropping rule to w_k
6. $w := w - w_k \tilde{u}_{k*}$
7. end
8. Apply a dropping rule to w
9. $\tilde{l}_{ij} = w_j$ for $j = 1, ..., i-1$
10. $\tilde{u}_{ij} = w_j$ for $j = i, ..., n$
11. end □

An application of Method 5.8 is ILUT(p,τ), which drops elements that are small in some sense. Moreover, it limits the number of elements per row.

Method 5.9 (ILUT(p,τ))**.** *ILUT(p,τ)* follows from Method 5.8 by using the following dropping rules:

Line 5 is replaced by:

if $w_k < \tau \|a_{i*}\|_2$: $w_k = 0$

Line 8 is replaced by:

for $k = 1, ..., n$:
  if $w_k < \tau \|a_{i*}\|_2$: $w_k = 0$
end
Drop all elements in $\underline{\mathbf{w}}$, except w_i, the p largest elements in $\{w_1, ..., w_{i-1}\}$, and the p largest elements in $\{w_{i+1}, ..., w_n\}$.

If A contains at most q nonzero elements per row, the costs of this factorisation amount to approximately

$$n\Big(p\underbrace{\big(2(p+1)+1\big)}_{\text{row update}} + \underbrace{2q}_{\|a_{i*}\|_2}\Big)$$

flops. □

5.3.2 Incomplete LU

Incomplete LU preconditioners form a subcategory of ILUT preconditioners. Their dropping rules are based on a zero pattern.

Method 5.10 (Incomplete LU (ILU): general form)**.** Let

$$Z \subset \{(i,j) \in [1, ..., n] \times [1, ..., n] : i \neq j\}$$

be a zero pattern. The general *Incomplete LU* algorithm follows from Method 5.8 by letting both dropping rules set w_k equal to zero if $(i,k) \notin Z$. In other words:

Line 5 is replaced by:

if $(i,k) \in Z$: $w_k = 0$

Line 8 is replaced by:

for $k = 1, ..., n$:
  if $(i,k) \in Z$: $w_k = 0$
end

Note that R follows from:

$$r_{ij} = \begin{cases} a_{ij} & (i,j) \in Z \\ 0 & (i,j) \notin Z \end{cases}$$ □

An application of Method 5.10 is $ILU(0)$.

Method 5.11 (ILU(0))**.** *ILU(0)* follows from Method 5.10 by taking Z equal to the zero pattern of A. □

In practice, ILU(0) can have insufficient accuracy, resulting in inefficiency and unreliability. ILU(0) has no fill-in. ILU(p), the generalisation of ILU(0), attempts to improve ILU(0) by allowing some fill-in. The method drops elements that have a level of fill (see Method 5.12 below for a definition) larger than p.

Method 5.12 (ILU(p))**.** Let f_{ij} denote the level of fill. Initially, this quantity is defined as follows:

$$f_{ij} = \begin{cases} 0 & a_{ij} \neq 0 \text{ or } i = j \\ \infty & a_{ij} = 0 \end{cases}$$

ILU(p) follows from Method 5.10 by using the zero pattern:

$$Z = \{(i,j) \in [1, ..., n] \times [1, ..., n] : f_{ij} > p\}$$

Right before the end of the k-loop (so before line 7 in Method 5.8), f_{ij} is updated according to:

$$f_{ij} = \begin{cases} \min\{f_{ij}, f_{ik} + f_{kj} + 1\} & w_j \neq 0 \\ f_{ij} & w_j = 0 \end{cases} \qquad j = 1, ..., n$$

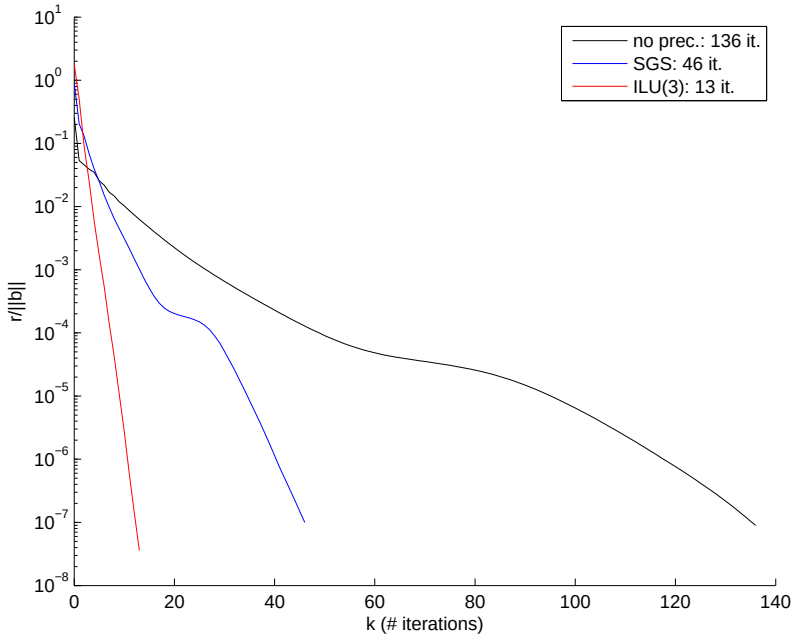
If $\tilde{L}$ has at most $\tilde{q}_l$ nonzero off-diagonal elements per row and $\tilde{U}$ has at most $\tilde{q}_u$ nonzero off-diagonal elements per row, then the costs of the factorization amount to approximately

$$n\tilde{q}_l\big(\underbrace{(2\tilde{q}_u + 1)}_{\text{row update}} + \underbrace{(\tilde{q}_l + \tilde{q}_u + 1)2}_{\text{level of fill update}}\big)$$

flops. □

ILU(p) might be a good alternative for the current preconditioner. This idea is based on Figure 5.2, in which the relative residual $\frac{\mathbf{b} - A\mathbf{x}_k}{\|\mathbf{b}\|_2}$ has been plotted for each iteration k for a two-dimensional diffusion dominated problem. The black line, which corresponds to the case without preconditioning, shows that preconditioning is indispensable. The blue line, which is the result of the symmetric Gauss-Seidel preconditioner, demonstrates that the current preconditioner is insufficient for this type of problems. The red line, which coincides with ILU(3)

Figure 5.2: The effect of preconditioning on the convergence of GMRES



preconditioning, illustrates that ILU(p) performs rather well for the diffusion dominates problems.

Nonetheless, according to Saad [11, p. 280], there "are a number of drawbacks to the above algorithm. First, the amount of fill-in and computational work for obtaining the ILU(p) factorization is not predictable for $p > 0$. Second, the cost of updating the levels can be quite high. Most importantly, the level of fill for indefinite matrices may not be a good indicator of the size of the elements that are being dropped. Thus, the algorithm may drop large elements and result in an inaccurate incomplete factorisation".

So far, the elements that were dropped were simply discarded. Modified ILU uses a different approach. It adds dropped elements to the diagonal of $\tilde{U}$.

Method 5.13 (Modified ILU (MILU))**.** *Modified ILU* follows from Method 5.10, by inserting the following diagonal update of $\tilde{U}$ right after line 10 in Method 5.8:

$$\tilde{u}_{ii} \quad := \quad \tilde{u}_{ii} + \sum_{m=1}^{n} r_{im} \qquad \square$$

MILU guarantees that A and $\tilde{L}\tilde{U}$ have the same row sums. Its results are especially good for matrices resulting from the discretisation of a PDE that has a more or less constant solution.

5.4 Conclusion

The convergence speed of an iterative method depends on the spectrum of the matrix. Preconditioning transforms a linear system into an equivalent system that has better spectral properties. At present, WAQ uses symmetric Gauss-Seidel preconditioning, which is costless to construct. Unfortunately, for diffusion dominated problems, this preconditioner is inadequate. An alternative preconditioning, such as ILU(p), could lead to a much smaller number of iterations. However, its construction comes with a price.

Chapter 6

Reordering

In Chapter 5, preconditioning was introduced to speed up the convergence of an iterative method. The construction of a good preconditioner is generally expensive. An important tool in facing this problem, which is especially useful for matrices arising from discretisation on an unstructured grid, is reordering of the matrix elements in advance. This is treated in this chapter.

In this chapter, level-set ordering, independent set ordering, and multi-color ordering are discussed. Other orderings can be found in [5, Chapter 8] and [6, Chapter 5].

6.1 Symmetric permutation

Reordering is actually is special case of preconditioning, in which the preconditioners are permutation matrices. A permutation matrix is the identity matrix with its rows or columns permuted.

Definition 6.1 (Interchange matrix)**.** An *interchange matrix* is the identity matrix with two of its rows interchanged. □

Definition 6.2 (Permutation matrix)**.** A *permutation matrix* $P \in \mathbb{R}^{n \times n}$ is a product of (at most n) interchange matrices. □

Proposition 6.3. *If P is a permutation matrix, then*

$$P^{-1} = P^T$$

Proof. See [11, p. 73] ■

Method 6.4 (Symmetric permutation)**.** A symmetric permutation follows from Method 5.1 by using

$$\begin{aligned} P_l &= P^T \\ P_r &= P \end{aligned}$$

□

Example 6.5. Consider a system

$$\underbrace{\begin{bmatrix} a_{11} & 0 & a_{13} & 0 \\ 0 & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & 0 \\ 0 & a_{42} & 0 & a_{44} \end{bmatrix}}_{A} \underline{\mathbf{x}} = \underline{\mathbf{b}}$$

Choose the following permutation matrix:

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Then

$$P^T AP = \begin{bmatrix} a_{11} & a_{13} & 0 & 0 \\ a_{31} & a_{33} & a_{32} & 0 \\ 0 & a_{23} & a_{22} & a_{24} \\ 0 & 0 & a_{42} & a_{44} \end{bmatrix}$$

Note that an (I)LU-factorisation of this matrix would have zero fill-in. □

6.2 Renumbering the adjacency graph

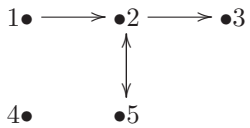
A symmetric permutation of a matrix is equivalent to renumbering the vertices of its adjacency graph [11, p.75].

Definition 6.6 (Graph)**.** A *graph* G consists of a set of vertices $V = \{v_1, ..., v_n\}$ and a set of edges $E \subset V \times V$. Notation: $G = (V, E)$. □

Definition 6.7 (Adjacency graph)**.** The *adjacency graph* of a matrix $A \in \mathbb{R}^{n \times n}$ is a graph $G = (V, E)$ such that:

- The vertices represent the unknowns: $v_1, ..., v_n \in V$.
- The edges represent the nonzero elements of the matrix: $(v_i, v_j) \in E$, if $a_{ij} \neq 0$ and $i \neq j$. □

Example 6.8 (Adjacency graph). The adjacency graph



corresponds to the matrix structure

$$\begin{bmatrix} * & * & & & \\ & * & * & & * \\ & & * & & \\ & & & * & \\ & * & & & * \end{bmatrix}$$

□

6.2.1 Level-set orderings

Level-set orderings are based on traversing the graph by level sets.

Definition 6.9 (Adjacent)**.** Two vertices in a graph are *adjacent* if they have a common edge. In other words: If $G = (V, E)$ is a graph, then $v, w \in V$ are adjacent, if $(v, w) \in E$ or $(w, v) \in E$. □

Definition 6.10 (Level set)**.** A *level set* of a graph $G = (V, E)$ is a recursively defined subset of V. The initial level set L_1 can be any subset of V. Each next level-set L_k ($k \geq 2$) contains the unmarked neighbors of the vertices of the previous level set:

$$L_k \quad = \quad \{v \in V \setminus (L_1 \cup ... \cup L_{k-1}) : \exists w \in L_1 \cup ... \cup L_{k-1} \text{ adjacent to } v\} \qquad \square$$

Method 6.11 (Level set ordering)**.** Consider a graph $G = (V, E)$. A basic *level set ordering* can be constructed by applying the following steps:

1. Choose an initial level set $L_1 = \{v_{m_1}, ..., v_{M_1}\} \subset V$. Mark all vertices $v \in L_1$.
2. While unmarked vertices are available: Determine the next level set $L_k = \{v_{m_k}, ..., v_{M_k}\}$ by traversing L_{k-1} in a certain way. Mark all vertices $v \in L_k$.
3. Order the vertices in the following manner:

$$v_{m_1}, ..., v_{M_1}, v_{m_2}, ..., v_{M_2}, ... \qquad \square$$

Level set orderings differ from one another in initial level set, way of traversing, and way of numbering. The Cuthill-McKee ordering, for example, is based on the degrees of the vertices.

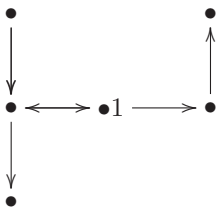
Definition 6.12 (Degree)**.** The *degree* of a vertex of a graph is the number of edges incident to it. Loops[1] are counted twice. □

Method 6.13 (Cuthill-McKee (CMK) ordering)**.** The *Cuthill-McKee ordering* follows from Method 6.11 by using the following strategies:

- The initial level set consists of a single node: $L_1 = \{v_{m_1}\}$.
- The elements of a level set are traversed from the nodes of lowest degree to those of highest degree.
- Nodes are numbered in the same order as they are traversed. □

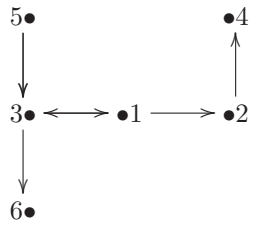
CMK ordering normally leads to a smaller bandwidth.

Example 6.14 (CMK ordering). Consider the following adjacency graph with initial level set {1}:



[1] Note that loops do not occur in adjacency graphs

Applying CMK ordering yields:



□

6.2.2 Independent set orderings

An independent set ordering isolates unknowns that are independent of one another. This results in a matrix of the form:

$$\begin{bmatrix} D & E \\ F & C \end{bmatrix}$$

in which D is a diagonal matrix. This structure is especially useful for parallel computing.

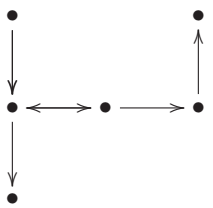
Definition 6.15 (Independent set)**.** An *independent set* of a graph $G = (V, E)$ is a set $S \subset V$ such that no two vertices are adjacent. More precisely, $\forall v \in S$:

$$(v, w) \in E \text{ or } (w, v) \in E \Rightarrow w \notin S$$ □

Method 6.16 (Independent Set Ordering (ISO))**.** An *independent set ordering* for a graph $G = (V, E)$ can be obtained by means of the following algorithm:

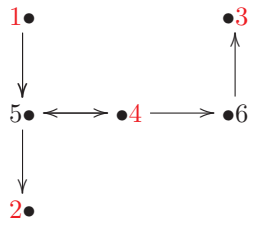
1. Initially, put $S = \varnothing$.
2. While unmarked vertices are available: Choose[2] an unmarked vertex v and add it to S. Mark v and all vertices adjacent to v.
3. Number the vertices that belong to the independent set S first. Then, number the other vertices. □

Example 6.17 (ISO). Consider the following adjacency graph:



An example of an ISO ordering is:

[2]Choose for instance the vertex of lowest degree. Heuristically, this yields a large independent set.



□

6.2.3 Multicolor orderings

Graph coloring is the process of coloring (labeling) vertices such that adjacent vertices do not have the same color. Moreover, this should be done with the least possible amount of colors.

A multicolor ordering is a color by color ordering after graph coloring has been executed. If k colors are used, k independent sets are obtained. This yields a $k \times k$ block matrix with diagonal matrices on the diagonal: Lovely for parallel computing.

In practice, the smallest possible number of colors to color the graph can rarely be easily determined. Therefore, this criterion is relaxed to obtain the following greedy algorithm.

Method 6.18 (Multicolor Ordering)**.** A *multicolor ordering* for a graph $G = (V, E)$ can be obtained by means of the following greedy algorithm:

1. Let S_i $(i = 1, ..., n)$ contain the vertices with color i. Initially, none of the vertices is colored, so put $S_i = \varnothing$ $(i = 1, ..., n)$.
2. While unmarked nodes are available:
   i. Choose an unmarked vertex $v \in V$.
   ii. Determine the 'smallest' color that none of its neighbors has:
   $$i = \min\{i \in \{1, ..., n\} | \forall w \in V \text{ adjacent to } v : w \notin S_i \}$$
   iii. Color v with color i: $S_i = S_i \cup \{v\}$
   iv. Mark v.
3. First number the vertices of S_1, then the nodes in S_2, and so on. □

6.3 Conclusion

The costs and the quality of a preconditioner partly depend on the structure of the original matrix. Therefore, it can be a good strategy to reorder the matrix elements in advance. This can be executed by renumbering the corresponding adjacency graph.

Chapter 7

Storage of sparse matrices

Since sparse matrices contain a large number of zero elements, an efficient way of storing them can save memory as well as computing time. Two popular storing formats are the coordinate format and the compressed sparse row format, which are both discussed in this chapter. Other formats can be found in [5, Chapter 2].

7.1 Coordinate format

The most basic format is the coordinate format[1], which only stores the nonzero elements and their row and column index.

Definition 7.1 (Coordinate format)**.** The *coordinate format* of a matrix $A \in \mathbb{R}^{m\times n}$ with k nonzero elements consists of three vectors. $\underline{\mathbf{e}} \in \mathbb{R}^k$ contains the nonzero elements of A, $\underline{\mathbf{r}} \in \mathbb{N}^k$ contains their row indices, and $\underline{\mathbf{c}} \in \mathbb{N}^k$ contains their column indices. The vectors can be filled in any order. If they are filled by row, the vectors are constructed according to:

1. $k = 0$
2. for $i = 1, ..., m$:
3. for $j = 1, ..., n$:
4. if $a_{ij} \neq 0$:
5. $k = k + 1$
6. $e_k = a_{ij}$
7. $r_k = i$
8. $c_k = j$
9. end
10. end
11. end □

Example 7.2 (Coordinate format). Consider the matrix

$$A = \begin{bmatrix} 1 & 0 & 0 & 2 & 0 \\ 3 & 4 & 0 & 5 & 0 \\ 6 & 0 & 7 & 8 & 9 \\ 0 & 0 & 10 & 11 & 0 \\ 0 & 0 & 0 & 0 & 12 \end{bmatrix} \tag{7.1}$$

[1] This format is used by MATLAB

The corresponding coordinate format reads:

$$\begin{array}{rcl} \underline{\mathbf{e}} & = & [\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ ]^T \\ \underline{\mathbf{r}} & = & [\ 1\ 1\ 2\ 2\ 2\ 3\ 3\ 3\ 3\ 4\ 4\ 5\ ]^T \\ \underline{\mathbf{c}} & = & [\ 1\ 4\ 1\ 2\ 4\ 1\ 3\ 4\ 5\ 3\ 4\ 5\ ]^T \end{array}$$

□

7.2 Compressed sparse row format

One of the most popular formats is the compressed sparse row format, which is comparable to the coordinate format. The difference is that the row indices are stored more efficiently.

Definition 7.3 (Compressed Sparse Row format (CSR))**.** The *compressed sparse row format* of a matrix $A \in \mathbb{R}^{m\times n}$ with k nonzero elements consists of three vectors. $\underline{\mathbf{e}} \in \mathbb{R}^k$ contains the nonzero elements of A, $\underline{\mathbf{c}} \in \mathbb{N}^k$ contains their column indices, and $\underline{\mathbf{r}} \in \mathbb{N}^{m+1}$ contains the pointers to the beginning of each row in the vectors $\underline{\mathbf{e}}$ and $\underline{\mathbf{c}}$. The vectors are filled by row, so according to:

1. $k = 0$
2. for $i = 1, ..., m$:
3. for $j = 1, ..., n$:
4. if $a_{ij} \neq 0$:
5. $k = k + 1$
6. $e_k = a_{ij}$
7. $c_k = j$
8. if the row pointer is not already set for row i: $r_i = k$
9. end
10. end
11. if the row pointer is not already set for row i: $r_i = k$
12. end
13. $r_{m+1} = k + 1$

□

Example 7.4 (CSR format). The *CSR format* for (7.1) reads:

$$\begin{array}{rcl} \underline{\mathbf{e}} & = & [\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ \ ]^T \\ \underline{\mathbf{r}} & = & [\ 1\ \ 3\ \ \ 6\ \ \ \ \ 10\ \ 12\ 13\ ]^T \\ \underline{\mathbf{c}} & = & [\ 1\ 4\ 1\ 2\ 4\ 1\ 3\ 4\ 5\ 3\ 4\ 5\ \ ]^T \end{array}$$

□

Method 7.5 (CSR-vector product)**.** A matrix stored in CSR format can be multiplied by a vector $\underline{\mathbf{x}}$ as follows:

1. for $i = 1 : n$
2. $k_1 = r_i$
3. $k_2 = r_{i+1} - 1$
4. $\left(\underline{\mathbf{e}}(k_1 : k_2)\right)^T \underline{\mathbf{x}}(c(k_1 : k_2))$
5. end

□

Remark 7.6 (Current format in WAQ). The current storage format in WAQ is comparable to the CSR format. The difference is that the diagonal elements are stored separately in a vector $\underline{\mathbf{d}}$. Furthermore, the row pointers point to the end of each row, instead of the beginning.

Some numerical schemes of WAQ treat the horizontal and the vertical direction separately. In that case, the format is slightly different. In the current implementation, for each grid cell V_i, there is exactly one coefficient t_i that represents the relation to grid cell right above it, and exactly one coefficient b_i that represents the relation to the grid cell below it. These two coefficients, that may be zero, are stored in $\underline{\mathbf{e}}$ in front of the (other) nonzero elements of the row.

For example, consider a matrix that corresponds to a three-dimensional $2 \times 2 \times 2$ rectangular grid:

$$\begin{bmatrix}
d_1 & h_{12} & h_{13} & & \mathbf{t_1} & & & \\
h_{21} & d_2 & h_{23} & h_{24} & & \mathbf{t_2} & & \\
h_{31} & h_{32} & d_3 & h_{34} & h_{35} & & \mathbf{t_3} & \\
 & h_{42} & h_{43} & d_4 & h_{45} & h_{46} & & \mathbf{t_4} \\
\mathbf{b_5} & & h_{53} & h_{54} & d_5 & h_{56} & h_{57} & \\
 & \mathbf{b_6} & & h_{64} & h_{65} & d_6 & h_{67} & h_{68} \\
 & & \mathbf{b_7} & & h_{75} & h_{76} & d_7 & h_{78} \\
 & & & \mathbf{b_8} & & h_{86} & h_{87} & d_8
\end{bmatrix}$$

This matrix would be stored as follows:

$$\begin{array}{lccccccccccccccc}
\underline{\mathbf{d}} = [& d_1 & & & & d_2 & & & & & & d_8 & & & &]^T \\
\underline{\mathbf{e}} = [& \mathbf{0} & \mathbf{t_1} & h_{12} & h_{13} & \mathbf{0} & \mathbf{t_2} & h_{21} & h_{23} & h_{24} & \dots & \mathbf{b_8} & \mathbf{0} & h_{86} & h_{87} &]^T \\
\underline{\mathbf{c}} = [& 0 & 5 & 2 & 3 & 0 & 6 & 1 & 3 & 4 & \dots & 4 & 0 & 6 & 7 &]^T \\
\underline{\mathbf{r}} = [& & & & 5 & & & & & 9 & \dots & & & & 42 &]^T
\end{array}$$

□

7.3 Conclusion

Since sparse matrices contain a large number of zero elements, an efficient way of storing them can save both memory and computing time. The main idea is to store only the nonzero elements. Additionally, information about their location in the original matrix is stored in some efficient manner. The current storage format is comparable to the compressed sparse row format.

Chapter 8

Conclusion and further investigation

There are several strategies that may tackle the two problems that were formulated in the introduction. These need to be further investigated.

8.1 An accurate and robust scheme

The current numerical schemes of WAQ are either expensive explicit higher order schemes or inaccurate implicit first order schemes. Therefore, an answer to the following question has been sought:

What are the possibilities for an accurate finite volume scheme for the advection diffusion equation on an unstructured three-dimensional grid that has a high upper bound for the time step?

An implicit flux correcting transport scheme, in which nonlinear terms are approximated explicitly, might yield a good combination of speed and accuracy. A detailed description of this scheme can be found in Method 3.35.

The following aspects need to be further investigated:

1. Which limiter and numerical fluxes could best be combined?
2. Does the corresponding stability criterion induce a sufficiently high upperbound for the time step?
3. Does FCT perform well for large time steps (the amount of admissable anti-diffusion might be inversely proportional to the time step)?

8.2 Convergence of GMRES

Another problem is the unsatisfactory convergence speed of the present solver resulting from linear systems for diffusion dominated problems. Therefore, an answer to the following question has been sought:

How can the convergence speed of the current solver for linear systems be enhanced for diffusion dominated problems?

First of all, a different solver could be implemented. Currently, GMRES (see Method 4.26) is used for solving linear systems. It is possible that an alternative solver (see Section 4.2.2) performs better for diffusion dominated problems.

Moreover, the preconditioning could be changed. At present, the solver is equipped with symmetric Gauss-Seidel preconditioning (see Example 5.5). Choosing a different preconditioner (see Chapter 5) could lead to a smaller number of iterations.

Finally, reordering (see Chapter 6) of the matrix elements could be applied before preconditioning. At the moment, this technique is not being used. However, it might result in cheaper and/or better preconditioners, especially in the case of an unstructured grid.

The following aspects need to be further investigated:

1. Is there a linear solver that performs better than GMRES for diffusion dominated problems?
2. Which combination of reordening and preconditioning is optimal?
3. Is it efficient to reuse a preconditioner, that is relatively expensive to construct, during multiple time steps?

Appendix A

Current schemes

At present, fifteen different finite volume schemes (see Method 3.29) can be used in WAQ. These are described briefly below.

Scheme 1 is an explicit ($\theta = 0$) first order upwind scheme (so ψ_{ij} is as in Method 3.20).

Scheme 2 is like scheme 1, except that it uses the predictor corrector method for time integration:

1. Compute $\overline{c}_i^{n+\frac{1}{2}}$ (for $i = 1, ..., I$) according to:

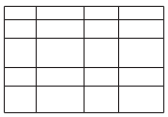
$$\frac{|V_i^{n+\frac{1}{2}}|\overline{c}_i^{n+\frac{1}{2}} - |V_i^n|\overline{c}_i^n}{\frac{1}{2}(t_{n+1} - t_n)} = a_i^n$$

2. Compute $\overline{c}_i^{n+1}$ (for $i = 1, ..., I$) according to:

$$\frac{|V_i^{n+1}|\overline{c}_i^{n+1} - |V_i^n|\overline{c}_i^n}{t_{n+1} - t_n} = a_i^{n+\frac{1}{2}}$$

Scheme 3 is an explicit Lax-Wendroff scheme (so ψ_{ij} is as in Method 3.24).

Scheme 4 is an Alternation Direction Implicit (ADI) method. It can only be applied in two dimensions on a structured grid:



This method calculates two successive timesteps in two different ways, using a semi-implicit scheme. In one time step the derivatives in the y-direction are evaluated explicitly instead of implicitly. In the other time step the derivatives in the x-direction are evaluated explicitly instead of implicitly. More precisely:

1. Let a_i^n be as in (3.3) and write $a_i^n = a_{i,x}^n + a_{i,y}^n$, where $a_{i,x}^n$ involves fluxes in the x-direction only and $a_{i,y}^n$ involves fluxes in the y-direction only.

2. Compute $\overline{c}_i^{n+1}$ according to:

$$\frac{|V_i^{n+1}|\overline{c}_i^{n+1} - |V_i^n|\overline{c}_i^n}{t_{n+1} - t_n} = (1-\theta)\left(a_{i,x}^n + a_{i,y}^n\right) + \theta\left(a_{i,x}^{n+1} + a_{i,y}^n\right)$$

3. Compute $\overline{c}_i^{n+2}$ according to:

$$\frac{|V_i^{n+2}|\overline{c}_i^{n+2} - |V_i^{n+1}|\overline{c}_i^{n+1}}{t_{n+2} - t_{n+1}} = (1-\theta)\left(a_{i,x}^{n+1} + a_{i,y}^{n+1}\right) + \theta\left(a_{i,x}^{n+1} + a_{i,y}^{n+2}\right)$$

This scheme uses $\theta = \frac{1}{2}$. Explicit fluxes are as in (3.7) with ψ_{ij} a higher order upwind scheme (see Method 3.22 and 3.23). Implicit fluxes are also as in (3.7), with ψ_{ij} a central flux (see Method 3.19). Since only one direction at the time is implicit, this results in a tridiagonal matrix, which is relatively easy to solve with a direct method.

Scheme 5 is an explicit ($\theta = 0$) FCT method a la Boris & Book (see Method 3.32 and Remark 3.31) which corrects the first order upwind scheme (so $\hat{\phi}_{ij}$ is as in Method 3.20) with the help of Lax-Wendroff (so $\tilde{\phi}_{ij}$ is as in Method 3.24).

Scheme 10 is an implicit ($\theta = 1$) first order upwind scheme (see Method 3.20).

Scheme 11 treats the horizontal and vertical direction seperately. In the horizontal direction, an explicit first order upwind scheme (scheme 1) is applied. In the vertical direction, a semi-implicit ($\theta = \frac{1}{2}$) central scheme (so ψ_{ij} is as in Method 3.19) is used. The resulting tridiagonal linear systems are solved by means of a direct method.

Scheme 12 is like scheme 11, except that it uses an explicit FCT scheme (scheme 5) in the horizontal direction.

Scheme 13 is like Scheme 11, except that it uses a first order upwind scheme (so ψ_{ij} is as in Method 3.20) in the vertical direction.

Scheme 14 is like scheme 12, except that it uses a first order upwind scheme (so ψ_{ij} is as in Method 3.20) in the vertical direction.

Scheme 15 is like scheme 10, except that, in the horizontal direction, the linear systems are solved by means of GMRES (see Method 4.26) with a symmetric GS preconditioner (see Example 5.5). In the vertical direction, a direct method is used.

Scheme 16 is like Scheme 15, except that it uses a central scheme (so ψ_{ij} is as in Method 3.19) in the vertical direction.

Scheme 19 treats the horizontal and vertical direction seperately. In the horizontal direction, an ADI method (scheme 4) is used. In the vertical direction, an implicit ($\theta = 1$) central (see Method 3.19) scheme is used.[1]

Scheme 20 is like scheme 19, except that it uses a first order upwind scheme (so ψ_{ij} is as in Method 3.20) in the vertical direction.

[1] Since the scheme is not positive, oscillations may occur. Therefor, an iterative procedure based on local diffusion and a non-linear smoothing operator (Forrester filter) are applied.

Bibliography

[1] T. Barth and M. Ohlberger. Finite volume methods: Foundation and analysis. In *Encyclopedia of Computational Mechanics*. John Wiley & Sons, 2004.

[2] J. P. Boris and D. L. Book. Flux corrected transport. 1. shasta, a fluid transport algorithm that works. *Journal of Computational Physics*, 11:38–69, 1973.

[3] R.L. Burden and J.D. Faires. *Numerical Analysis*. Brooks/Cole, Pacific Grove, 2001.

[4] Kuzmin D and S. Turek. Flux correction tools for finite elements. *Journal of Computational Physics*, 175:525–558, 2002.

[5] I.S. Duff, A.M. Erisman, and J.K. Reid. *Direct Methods for Sparse Matrices*. Clarendon Press, Oxford, 1986.

[6] A. George and J.W. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice Hall, Inc., New Jersey, 1981.

[7] E. Godlewski. *Numerical Approximation of Hyperbolic Systems of Conservation Laws*. Springer, New York, 1996.

[8] G.H. Golub and C.F. Van Loan. *Matrix computations*. The Johns Hopkins University Press, Baltimore and London, third edition, 1996.

[9] D. Kröner. *Numerical Schemes for Conservation Laws*. John Wiley & Sons Ltd, West Sussex and B.G. Teubner, Stuttgart, 1997.

[10] R.J. LeVeque. *Finitie Volume Methods for Hyperbolic Problems*. Cambridge University Press, New York, 2002.

[11] Y. Saad. Iterative methods for sparse linear systems. This is a revised version of the book published in 1996 by PWS Publishing, Boston. It can be downloaded from http://www-users.cs.umn.edu/~saad/books.html, 2000.

[12] Y. Saad and M.H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7:856–869, 1986.

[13] S.T. Zalesak. Fully multidimensional flux-corrected transport algorithms for fluids. *Journal of Computational Physics*, 31:335–362, 1979.

Index