Delft University of Technology

Faculty of Applied Sciences

Faculty of EEMCS

Bachelor Thesis

# Numerical Simulation of Dispersion in Stratified Porous Media

Steven Kortekaas

Bachelor Thesis,
BSc Applied Physics and Applied Mathematics,
TU Delft

Supervisors:
Dr.ir. J.E. Romate
Prof.dr.ir. C.R. Kleijn
Delft, December 17, 2019

**Abstract**

The transport of a solute dissolved in a fluid flowing trough porous media is, next to advection and diffusion, determined by hydrodynamic dispersion. This behaviour is commonly characterized using the longitudinal and transverse dispersion coefficients. Laboratory and field measurements of these coefficients tend to differ, which might be attributed to heterogeneities found in field porous media.

To investigate this, a stratified porous medium consisting of two layers is considered. Each layer has different physical properties, resulting in a different average fluid velocity. As a consequence of the difference in velocity, transport of the solute occurs between the two layers. Under certain circumstances the layers start to behave as one single layer, with one single effective dispersion coefficient, explaining the discrepancy between field and laboratory measurements.

The two-layer stratified porous medium is characterized using a dimensionless number. It is investigated for which values of this number the porous medium acts as one single layer, and for which values the medium behaves as two separate layers. This is done by introducing an index, which effectively measures the behaviour of the medium in terms of these two limit cases. The calculation of the index is done using a numerical simulation of flow and dispersion in the stratified porous medium.

It was found that the dimensionless number was in general a good predictor of the behaviour of the stratified porous medium. The system behaved as one single layer if the dimensionless number (after a correction with a certain factor) was much greater than unity. Similarly, the system behaved as two separate layers if the number was much less than unity. However, this number failed if the ratio of the two layer thicknesses was varied. A correction to the dimensionless number was suggested, taking the ratio into account.

# Contents

# 1 Introduction

Transport of a solute in porous media is, next to advection and diffusion, determined by hydrodynamic dispersion. Hydrodynamic dispersion is similar to molecular diffusion, but is unlike diffusion anisotropic and depends on the average fluid velocity.

Quantitative description of hydrodynamic dispersion is relevant to different fields like hydrology and oil reservoir engineering, [1, 17]. For example, in groundwater hydrology the flow of contaminated water could be modelled with hydrodynamic dispersion. Likewise, the transition zone between salt and fresh water in coastal regions can be better understood through hydrodynamic dispersion. In oil reservoir engineering, hydrodynamic dispersion is of interest regarding enhanced oil recovery precesses, [3, 10].

In the direction of the fluid velocity, dispersion is often modelled using the longitudinal dispersion coefficient, $D_l = \alpha_l u + D_m$, where $\alpha_l$ is the longitudinal dispersivity, $u$ is the average fluid velocity and $D_m$ the molecular diffusion coefficient, [1, 6, 15]. Field and laboratory measurements of $\alpha_l$ tend to differ, [8, 10]. Not only due to scale (in laboratory measurements, $\alpha_l$ is in the order of centimeters, while in field measurements it can be in the order of meters), but additional differences might be attributed to heterogeneities found in field porous media. Earlier work to explain this difference in measurements includes [3, 8, 10, 19].

To investigate this, a stratified porous medium is considered, consisting of two layers with different physical properties. In this system, a solute dissolved in a fluid is continuously injected in both layers. Due to the different physical properties, the average fluid velocities in each layer will be different, which will cause transversal transport of the solute between the two layers. Under certain circumstances, the two-layer stratified porous medium can be described as being one single layer, with one single effective dispersion coefficient. This effective dispersion coefficient can in turn explain the discrepancy between field and laboratory measurements.

It is the aim of this study to characterize the system using a dimensionless number, which is called the transverse dispersion number. It is then investigated for which values of the transverse dispersion number the system behaves as one single layer or as two separate layers. This is done using the transverse dispersion index, which effectively measures the behaviour of the system in terms of two limit cases. The transport of the solute in the two-layer stratified porous medium is modelled using numerical simulations, which are performed using MATLAB. From the numerical simulations, the transverse dispersion index can be calculated.

In chapter 2 a general description of porous media and transport therein is given. Then in chapter 3, the model of the two-layer stratified porous medium is given, as well as a way to characterize it and measure its behaviour. The numerical method is described in chapter 4. In chapter 5 the results will be presented, after which the conclusions will be given in chapter 6.

# 2 Transport Phenomena in Porous Media

This section gives an overview of transport phenomena occurring in porous media, and the way they are modelled. This background information is used later on in the modeling of the transport of a solute dissolved in a fluid flowing through a layered (stratified) porous medium, and the characterization of such a system. For more elaborate introductions on porous media and the transport of a solute in porous media, see for example [1, 17].

First, a description of porous media is given, after which single-phase fluid flow in porous media is described. Next, the transport of a solute dissolved in a fluid flowing in a porous medium is described, as well as the equations and boundary conditions to model the transport.

## 2.1 Description of porous media

In the most simple terms, porous media are solids which contain interconnected "holes" usually filled with fluid. The interconnected holes should also enable continuous paths across the medium in order for the fluid to flow. The part of the medium which is solid is called the *solid matrix* and the space that is not part of the solid matrix is called the *void* or *pore space*. A more proper definition of porous media is given in [1]. Naturally, porous media can occur in many forms, such as packed beds or fractured rock.

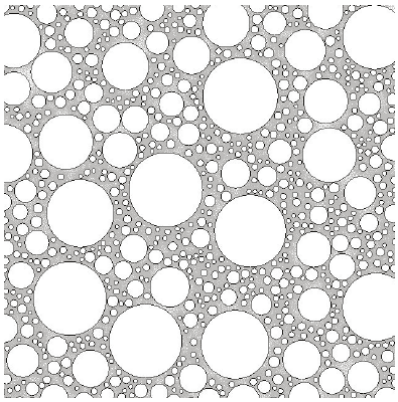


Figure 1: A randomly distributed porous medium, with random sized and constant shaped grains. The pore space is grey, and the solid matrix is white. This example contains no dead-end pores nor unconnected pores. This figure is from [5].

Porous media often are made up out of constituent particles, or grains. The distribution of the grains can be either structured of random, as well as their shape and size. In case of randomness, it is hard to describe phenomena, like fluid flow and transport, occurring in porous media, as will be explained in section 2.2. For regular porous media,

these difficulties might be possible to overcome, but are mostly still too difficult to solve (numerically). In either way, one usually resorts to statistical or continuum methods, or a combination of both. An example of a statistical method is a network of capillary tubes, and for continuum models often spatial-averaging is used, [1]. The models used in this thesis are based on the latter.

To quantify the pore space in a porous medium the porosity $\phi$ is used. This is defined as the ratio between the volume occupied by the pore space $V_p$ (pore volume) and the total volume $V$ of the medium, $\phi = \frac{V_p}{V}$. It could also be possible that a porous medium has a porosity which locally differs (for example porous media can be layered), in which case the $V_p$ and $V$ are taken in a representative elementary volume (the smallest volume for which the continuum approach holds).

Apart from interconnected pores, porous media can also contain *unconnected pores* and *dead-end pores*. From the standpoint of flow through porous media only interconnected pores are of interest, so that unconnected pores can be considered to be part of the solid matrix. This gives rise to an effective porosity $\phi_{\text{eff}}$, in which $V_p$ is taken only over interconnected pore space. The porosity in this thesis will be assumed to be the effective porosity.

## 2.2 Darcy and fluid flow in porous media

Fluid flow through the pores of a porous medium generally has a low Reynolds number (this will be defined below), so that the Stokes equation can be used to calculate the fluid velocity field $\mathbf{u}$ in the pore space. Also, the fluid is considered to be Newtonian, so that the Stokes equations take the form of ([9, 17])

$$\mu \boldsymbol{\nabla}^2 \mathbf{u} - \boldsymbol{\nabla} p + \rho \mathbf{g} = \mathbf{0} \tag{1}$$

where $\mu$ is the dynamic viscosity, $p$ is the pressure and $\mathbf{g}$ is the gravitational acceleration.

Furthermore, the fluid is considered to be incompressible, so its density $\rho$ is constant over time and space. From the continuity equation ([1]),

$$\frac{\partial \rho}{\partial t} + \boldsymbol{\nabla} \cdot (\rho \mathbf{u}) = 0 \tag{2}$$

a constant density gives that

$$\boldsymbol{\nabla} \cdot \mathbf{u} = 0. \tag{3}$$

Equations (1) and (3) can in principle be used to calculate the flow through the pores of a porous medium, given the pore geometry of the medium, pressure, dynamic viscosity and proper boundary conditions. However, the geometry of the pores of a medium is for all practical purposes unknown. To be able to approximate the flow velocity a continuum

approach can be used to get effective flow equations at a much larger scale. This yields Darcy's law ([9, 17]),

$$\mathbf{q} = -\frac{k}{\mu}\left(\boldsymbol{\nabla} p - \rho \mathbf{g}\right) \tag{4}$$

where $\mathbf{q}$ is the volume flux per unit area, also called Darcy velocity, and $k$ is the permeability of the porous medium. Multiple derivations of Darcy's law can be given (for example in [1], [13], [17] and [22]), and they agree with experimental results. Darcy's law in the form of (4) suffices for most applications in ground water hydrology and petroleum engineering as long as the permeability is isotropic. The Darcy velocity is not the same as the (average) fluid velocity[1], but is related to it by the (effective) porosity $\phi$ of the porous medium,

$$\bar{\mathbf{u}} = \frac{\mathbf{q}}{\phi} \tag{5}$$

This accounts for the fact that only a fraction of the porous medium is available for flow.

To define a range of validity for Darcy's law, the Reynolds number (Re) is used. It is defined as

$$\mathrm{Re} = \frac{\rho \bar{u} d}{\mu} \tag{6}$$

where $\bar{u}$ is the magnitude of the average fluid velocity and $d$ is a characteristic length. For the characteristic length multiple options are possible, but often the average size or diameter of the porous medium's constituents (or grains) is used. In practically all cases, with $d$ the average grain size, Darcy's law is valid if Re does not exceed some value between 1 and 10 (in other words, for laminar flow), [1].

In case the flow is in the transition zone from laminar to turbulent, or if it is turbulent, Darcy's law is not valid any more, since Stokes equation is not valid anymore. Some extensions to Darcy's law take this into account, but this is beyond the scope of this thesis.

In stratified porous media, Darcy's law is still valid. Consider a stratified porous medium consisting of $n$ layers, each with porosity $\phi_j$, permeability $k_j$ and thickness $h_j$ (see figure 2). The viscosity $\mu$ is considered constant, and it is assumed that gravity can be ignored. Then, if the pressure is constant in $y$, the average fluid velocity in layer $j$ is only in the $\hat{x}$-direction, and is given by

$$\bar{u}_j = -\frac{k_j}{\mu \phi_j}\frac{\partial p}{\partial x}$$

[1] For the average fluid velocity, also $\boldsymbol{\nabla} \cdot \bar{\mathbf{u}} = 0$ holds.

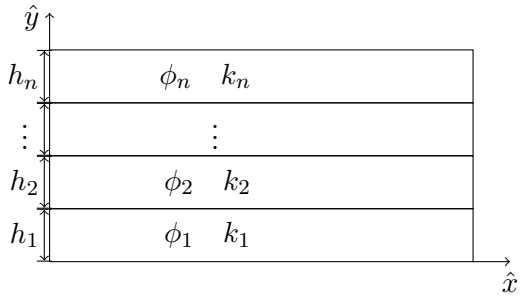



Figure 2: Stratified porous medium consisting of $n$ layers, each with porosity $\phi$, permeability $k$ and thickness $h$.

### 2.3 Transport of a solute in a porous medium

The transport of a solute dissolved in a fluid is generally described by the well-known advection-diffusion equation,

$$\frac{\partial c}{\partial t} = \boldsymbol{\nabla} \cdot (D_m \boldsymbol{\nabla} c) - \boldsymbol{\nabla} \cdot c\mathbf{u} \tag{7}$$

where $c$ is the solute concentration and $D_m$ is the molecular diffusion coefficient. The first term on the right-hand side of (7) accounts for molecular diffusion (as described by Fick's law), and the second term for advection.

Equation (7), together with (1), (3) and proper boundary conditions, can fully describe the transport of particles dissolved in a fluid flowing through a porous medium. (This is done, for example, in [3] and [7]) However, again the geometry of the pores of the medium is for all practical purposes unknown, and furthermore for porous media the formulation of boundary conditions is very hard. To deal with this, lots of different methods have been developed [1, 17].

Most methods, such as volume averaging methods, yield a generalization of (7), with $D_m$ replaced by a tensor $\mathbf{D}$, called the hydrodynamic dispersion tensor, and $\mathbf{u}$ replaced by $\bar{\mathbf{u}}$ from (5). This gives the advection-dispersion equation,

$$\frac{\partial c}{\partial t} = \boldsymbol{\nabla} \cdot (\mathbf{D} \boldsymbol{\nabla} c) - \boldsymbol{\nabla} \cdot c\bar{\mathbf{u}} \tag{8}$$

When the fluid flows in a porous medium, the flow lines are deviated in the geometry of the pore space. Because of this deviation, the length of the path each particle of the solute takes differs, causing an additional spread of the particles. (Dependent on the Stokes number, the particles may also change flow lines more often in a porous medium, which even further causes spreading.) Mostly (but not always) the spreading of the solute is diffusion like, justifying the use of the hydrodynamic dispersion tensor, [6]. Hydrodynamic dispersion is the spreading of a solute when it is transported in a porous medium.

## 2.4 Hydrodynamic dispersion

The dispersion tensor describes the anisotropic behaviour of hydrodynamic dispersion. To demonstrate this anisotropic behaviour, consider the case where the velocity field only has a component in the $\hat{x}$-direction. Then for most models the dispersion tensor takes the form of (in two dimensions)

$$\mathbf{D} = \begin{pmatrix} D_l & 0 \\ 0 & D_t \end{pmatrix} \tag{9}$$

where $D_l$ and $D_t$ are called the longitudinal and transverse coefficients of dispersion respectively.

Table 1 summarizes experimental results ([17]) of the longitudinal and transverse coefficients of dispersion in terms of the molecular diffusion coefficient $D_m$ and the dimensionless Peclet number Pe. The Peclet number is defined, for porous media, as $\mathrm{Pe} = \frac{\bar{u}d}{D_m}$ where $\bar{u}$ is the magnitude of the average fluid velocity, $d$ is a representative grain diameter and $D_m$ is the molecular diffusion constant. Like with Reynolds number, the average grain size is often used for $d$. Further in the table, $F$ is the formation factor (left undefined here), the $\alpha$'s and $\beta$'s are constants (which for different rows are not the same) and the $f$'s are some functions.

For $\mathrm{Pe} < 0.3$, diffusion becomes dominant, and the dispersion coefficients are not dependent on Pe. Then as Pe increases ($0.3 < \mathrm{Pe} < 5$) a transition zone is reached where it is hard to tell the relationship between $D_l$, $D_t$ and Pe. For $5 < \mathrm{Pe} < 300$ the so-called power-law regime is reached, where the influence of diffusion is noticeable, but the Peclet dependency is clear ($\beta_l$ and $\beta_t$ usually take a value close to one). If $300 < \mathrm{Pe} < 10^5$, then the diffusion becomes negligible, and the dispersion coefficients become linearly dependent on Pe. Next, if Pe further increases, the fluid flow becomes turbulent, and thus the dispersion coefficients become dependent on the Reynolds number. This hardly occurs in porous media. Lastly, there is hold-up dispersion, which is independent of Pe. This is caused by the dead-end pores (section 2.1), which trap the solute so that it can only escape through molecular diffusion.

Table 1: Experimental Peclet dependency of the longitudinal and transverse dispersion coefficients, [17].

| Pe | $\frac{D_l}{D_m} =$ | $\frac{D_l}{D_m} =$ | regime |
|---|---|---|---|
| $<0.3$ | $\frac{1}{F\phi}$ | $\frac{1}{F\phi}$ | diffusion |
| $0.3 < \mathrm{Pe} < 5$ | $f_l(\mathrm{Pe})$ | $f_t(\mathrm{Pe})$ | transition |
| $5 < \mathrm{Pe} < 300$ | $\frac{1}{F\phi} + \alpha_l \mathrm{Pe}^{\beta_l}$ | $\frac{1}{F\phi} + \alpha_t \mathrm{Pe}^{\beta_t}$ | power-law |
| $300 < \mathrm{Pe} < 10^5$ | $\alpha_l \mathrm{Pe}$ | $\alpha_t \mathrm{Pe}$ | pure advection |
| $\mathrm{Pe} > 10^5$ | $f_l(\mathrm{Pe}, \mathrm{Re})$ | $f_t(\mathrm{Pe}, \mathrm{Re})$ | turbulent |
| | $\alpha_l \mathrm{Pe}^2$ | $\alpha_t \mathrm{Pe}^2$ | holdup dispersion |

Table 2: Comparison of multiple methods to estimate the dispersion coefficient $D_l$

| Method | $\frac{D_l}{D_m} =$ | Author(s) |
|---|---|---|
| Capillary tube | $1 + \frac{1}{48}\mathrm{Pe}^2$ | Taylor |
| Volume-averaging | $1 + \frac{3}{4}\mathrm{Pe} + \alpha\mathrm{Pe}\ln\mathrm{Pe} + \beta\mathrm{Pe}^2$ | Koch and Brady |
| Fluid mechanical | $\alpha\mathrm{Pe}(\ln\mathrm{Pe})^\beta$ | Saffman |

Table 2 gives three examples of methods which estimate the longitudinal dispersion coefficient (with similar expressions for the transverse dispersion coefficients). This earliest and simplest work has been done by Taylor, [18]. He did not consider a porous medium, but a capillary tube instead. More advanced works include that of Koch and Brady ([9]), and Saffman ([16]). Each method uses different assumptions and simplifications, and are valid for different values of Pe, but the results contain similar terms. More methods can be found in [1], [9] and [17], for example.

More generally, the dispersion tensor $\mathbf{D}$ is usually taken to be ([1, 21])

$$\mathbf{D} = (\alpha_l - \alpha_t)\frac{\bar{\mathbf{u}}\bar{\mathbf{u}}^T}{||\bar{\mathbf{u}}||} + \alpha_t||\bar{\mathbf{u}}||\mathbf{I} + D_m\mathbf{I} \tag{10}$$

where $\alpha_l$ and $\alpha_t$ are the longitudinal and transverse dispersivities respectively, $\bar{\mathbf{u}}$ is the average fluid velocity and $\mathbf{I}$ is the identity matrix. In one dimension, the dispersion tensor becomes $D_l = D_m + \alpha_l\bar{u}$, so that there is no anisotropic behaviour. (This is known as the Perkins-Johnston relationship, [15]) In case of a constant velocity $\bar{\mathbf{u}} = (\bar{u}, 0)^T$ in the $\hat{x}$-direction (in two dimensions here), it follows from (10) that

$$\mathbf{D} = u\begin{pmatrix}\alpha_l & 0\\ 0 & \alpha_t\end{pmatrix} + \begin{pmatrix}D_m & 0\\ 0 & D_m\end{pmatrix} \tag{11}$$

(which explains the terms longitudinal and transverse dispersivity). As can be seen, (11) agrees with the experimental results from table 1 if $5 < Pe < 10^5$.

Values of the longitudinal dispersivities range between 0.13-0.51 cm in laboratory measurements, to even 2.5 m in field measurements. The value of the transverse dispersivities is typically $\alpha_t \approx \alpha_l/30$ [10].

# 3 Dispersion in Stratified Porous Media

In a paper by L.W. Lake and G.J. Hirasaki, *Taylor's Dispersion in Stratified Porous Media* ([10]), the transport of a solute through a two-dimensional stratified porous medium consisting of two layers is described. Under certain circumstances these two layers actually start behaving as one single layer. This aks for an investigation on how this behaviour can be characterized and predicted.

Furthermore, field measurement differ from laboratory measurements of the longitudinal dispersivity $\alpha_l$. If a stratified porous medium can be described as one single layer, then this can result in an effective longitudinal dispersivity, $\alpha_{l,\text{eff}}$. This might explain the discrepancy between the laboratory and field measurements, as porous media in field measurements tend to be (more) heterogeneous.

In this chapter, first the physical properties and the model of a two-layer stratified porous medium are described. Then a description is given of the behaviour of this system for various values of the physical parameters, and of the behaviour in their limit. Lastly a way to a priori characterize the system using a dimensionless number is presented, as well as an index to effectively measure the behaviour of a given system in terms of the limit cases.

## 3.1 Model

Consider a two-dimensional porous medium, with length $L_x$ and thickness $L_y$. The porous medium is assumed to consist of two layers, each with height $h_j$, porosity $\phi_j$ and longitudinal average fluid velocity[2] $u_j$, as shown in figure 3. The transverse average fluid velocity is assumed zero. Without loss of generality, throughout this thesis it is assumed that $u_1 > u_2$.

These average fluid velocities can be realized if there is the pressure is contant in $y$, see section 2.2. (Further the viscosity $\mu$ is assumed constant, so that any choice of $u_j$ and $\phi_j$ determines the permeability $k_j$ in each layer.)

The transport of a solute with concentration $c$ through a porous medium in general is described by (section 2.3)

$$\frac{\partial c}{\partial t} + \boldsymbol{\nabla} \cdot c\mathbf{u} = \boldsymbol{\nabla} \cdot \mathbf{D}\boldsymbol{\nabla} c \tag{12}$$

where $\mathbf{u}$ is the average fluid velocity and $\mathbf{D}$ is the dispersion tensor. Using the above assumption ($\mathbf{u} = (u, 0)^T$), $\mathbf{D}$ is given by (section 2.4)

$$\mathbf{D} = \begin{pmatrix} \alpha_l u + D_m & 0 \\ 0 & \alpha_t u + D_m \end{pmatrix} = \begin{pmatrix} D_l & 0 \\ 0 & D_t \end{pmatrix} \tag{13}$$

[2]From here on, the average fluid velocity will be denoted with $u$ or $\mathbf{u}$, instead of $\bar{u}$ or $\bar{\mathbf{u}}$ (see section 2.2)

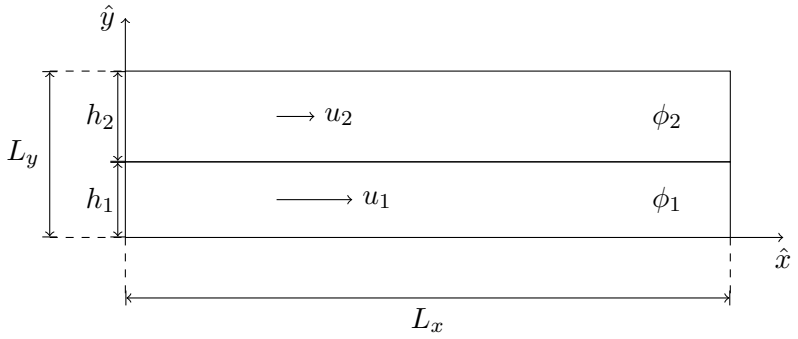



Figure 3: Two layer porous medium.

with $\alpha_l$ and $\alpha_t$ the longitudinal and transverse dispersivities, respectively, and $D_m$ the molecular diffusion coefficient. The dispersivities and diffusion coefficient are assumed homogeneous in the medium. Furthermore, the fluid is assumed incompressible, so that $\boldsymbol{\nabla} \cdot \mathbf{u} = 0$. With this and $\mathbf{D}$, and equation (12) becomes

$$\frac{\partial c}{\partial t} + u\frac{\partial c}{\partial x} = (\alpha_l u + D_m)\frac{\partial^2 c}{\partial x^2} + \frac{\partial}{\partial y}\left((\alpha_t u + D_m)\frac{\partial c}{\partial y}\right) \tag{14}$$

### 3.1.1 Initial and boundary conditions

The initial concentration is assumed zero on the domain, and at the left boundary ($x = 0$) the concentration is kept constant at unity, $c(0, y, t) = 1$.

The top and bottom boundaries ($y = 0$ and $y = L_y$) are solid boundaries so that the solute flux is zero,

$$(\mathbf{D}\boldsymbol{\nabla} c - c\mathbf{u}) \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} = 0 \quad y = 0, L_y$$

Since the transverse velocity is zero, there is no transverse advective transport, and the boundary condition for $y = 0$ and $y = L_y$ becomes $D_t \frac{\partial c}{\partial y} = 0$.

At the right boundary ($x = L_x$), ideally a free-flow boundary condition is imposed.[3] However, for the numerical method, this is not possible. Instead, an (artificial) boundary condition of no net dispersive transport is imposed, $D_l \frac{\partial c}{\partial x} = 0,\ x = L_x$.

Summarized the initial and boundary conditions are,

$$\begin{cases} c(x, y, 0) = 0, & (x, y) \in (0, L_x) \times (0, L_y) \\ c(0, y, t) = 1, & y \in (0, L_y), t > 0 \\ \frac{\partial c}{\partial y}(x, y, t) = 0, & y \in \{0, L_y\}, x \in (0, L_x), t > 0 \\ \frac{\partial c}{\partial x}(L_x, y, t) = 0, & y \in (0, L_y), t > 0 \end{cases} \tag{15}$$

[3] In other words, if the domain is infinitely extended in the $\hat{x}$-direction, then the boundary condition would become $\lim_{x\to\infty} c = 0$.

## 3.2 Qualitative behaviour

To understand the behaviour of the transport of a solute through the stratified porous medium, first the limit cases of two separate layers and one single layer need to be considered. This aids in finding a quantitative measure for transport behaviour through a stratified porous medium and to characterize it.

### 3.2.1 Single layer limit

In certain cases transport in the stratified porous medium can behave as through a single layer, and might be described by a one-dimensional advection-dispersion equation.

In the following, concentration fronts are defined as the moving fronts in case there is only advection, see figure 4. Furthermore, since the left boundary concentration is constant in $y$, the gradient in this direction is zero, $\frac{\partial c}{\partial y}(0, y, t) = 0$. Similarly, since the initial concentration is constant in $y$, the initial gradient in this direction is zero, $\frac{\partial c}{\partial y}(x, y, 0) = 0$. This means that initially there is no transverse dispersive transport. Transverse dispersive transport later arises because the concentration fronts moving at different velocities create a gradient in the $y$-direction. For the system to behave as one single layer, this gradient needs to remain small.

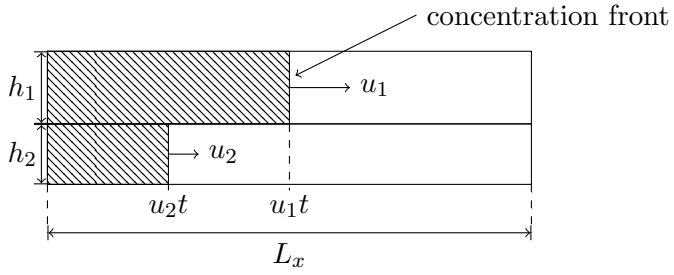



Figure 4: Concentration fronts at time $t$.

The cases to be distinguished, for the system to behave as a single layer, are (all other physical properties remain constant)

(a) $h_1$ or $h_2 \to L_y$. If the thickness of the porous medium is nearly the same as one of the layer thicknesses, then the time for the solute to transversely cross the small layer (by transverse dispersion) will be very short. The concentration in the small layer will be dominated by the concentration in the big layer.

(b) $L_y \to 0$ (with constant layer thickness ratio). If the medium thickness goes to zero, then the time for the concentration solute to transversely cross the medium will also go to zero, and any gradient will be damped.

(c) $\alpha_t \to \infty$. If $\alpha_t \to \infty$, then the transverse dispersion will go to infinity. Even though there will be a $y$-gradient caused by the concentration moving at different

velocities, the infinite transversal dispersive flux will instantaneously damp this out.

(d) $L_x \to \infty$. If the medium becomes very long, then a $y$-gradient may develop, but this will damp out, since the time to longitudinally cross the medium is much longer than to transversely cross the medium.

(e) $u_1 = u_2$. The case of equal velocities is trivial.

Thus, for any of the above cases the porous medium is expected to behave as a single layer, and can be described by

$$\frac{\partial c'}{\partial t} + u' \frac{\partial c'}{\partial x} = D_{\text{eff}} \frac{\partial^2 c'}{\partial x^2} \tag{16}$$

with $u'$ some average velocity and $D_{\text{eff}}$ the effective dispersion coefficient, and with initial and boundary conditions from (15) A solution is given by[4]

$$c'(x, y, t) = \text{erfc}\left(\frac{x - u't}{\sqrt{4 D_{\text{eff}} t}}\right) \tag{17}$$

It can be verified that this solution satisifies (16). However, it does not satisfy[5] the boundary conditions $c'(0, y, t) = 1$ and $\frac{\partial c'}{\partial x}(L_x, y, t) = 0$. The latter does not matter, as this boundary condition was imposed for numerical reasons, and former is nearly satisfied as $c'(0, y, t)$ is close to one at all times.

Now, $u'$ and $D_{\text{eff}}$ can be expressed in terms of the properties of the two layer medium (like the layer velocities and thicknesses). An expression for $u'$ and $D_{\text{eff}}$ is given in [10]. In this expression, $D_{\text{eff}}$ is larger than the thickness-weighed average longitudinal dispersion coefficients of the two layer (and in turn results in an increased effective longitudinal dispersivitiy $\alpha_{l,\text{eff}}$). Thus heterogeneities in the porous medium, under certain circumstances, tend to increase the effective longitudinal dispersion.

### 3.2.2 Double layer limit

Similar to the single layer limit, multiple cases can be distinguished for the system to behave as two separate layers.

(a) $D_t = 0$. In case there is no transverse dispersion (in any of the two layers), the porous medium will behave as two separate layers. (Naturally, the transverse dispersion cannot be zero, as there is always molecular diffusion.)

(b) $h_1$ and $h_2 \to \infty$. If the layers are very thick, then the medium will behave as two separate layers, as the time for the concentration solute to cross the medium will go to infinity, and any $y$-gradient will not be damped out.

[4] The complementary error function is defined here as $\text{erfc}(x) = 1 - \text{erf}(x) = \frac{1}{\sqrt{\pi}} \int_x^\infty \exp(-z^2) \text{d}z$

[5] It is the solution on an infinite domain, with the boundary conditions that $\lim_{x \to \pm\infty} c$ is finite. It does satisfy the initial condition.

(c) $L_x \to 0$. Similarly, if the medium is very short, then the two concentration fronts reach the end of the medium before the gradient in the $y$-direction increases, and no interaction occurs. This can also be interpreted a considering very short time scales.

(d) $u_1 >> u_2$. If the difference between the velocities is large, then the $y$-gradient will remain large throughout the medium.

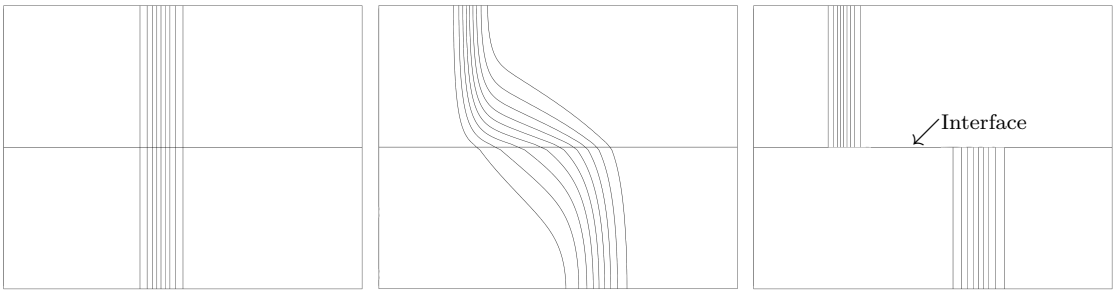
With the same arguments regarding the boundary conditions for (17), the solution to equation (14) in the double layer limit is approximately given by

$$\tilde{c}(x,y,t) = \begin{cases} \operatorname{erfc}\left(\frac{x-u_1 t}{\sqrt{4D_{l1}t}}\right), & y \in [0, h_1) \\ \operatorname{erfc}\left(\frac{x-u_2 t}{\sqrt{4D_{l2}t}}\right), & y \in (h_1, L_y] \end{cases} \tag{18}$$

where $D_{lj} = \alpha_l u_j + D_m$ (at $y = h_1$, $\tilde{c}$ is left undefined).

### 3.2.3 Intermediate case

Figure 5 show the calculated isoconcentration lines for the two limit cases and for a system with interacting layers using the numerical method presented later. In the single layer limit the isoconcentration lines are straight through both layers, as is expected from (17). In the double layer limit, the layers behave separately and there is no (net) transport of solute between the them. (Also the isoconcentration lines in the bottom layer are spaced further apart, since the dispersion coefficient is velocity dependent, unlike normal diffusion problems.)




(a) Single layer limit. (b) Interacting layers. (c) Double layer limit.

Figure 5: Isoconcentration lines for different cases. In (b) and (c) the bottom layer has a higher velocity.

With understanding of the limit cases, the behaviour of actual stratified porous media can be described, and how the layers are going to interact. Because of the difference in layer velocities an interface, as shown in figure 5c for the double layer limit, will form, causing a gradient in the concentration in the $\hat{y}$-direction. This can be seen in figure 5b, as the isoconcentration lines are not straight. The $y$-gradient causes a transverse

dispersive flux, transporting the solute form the layer with the higher velocity to the other layer. The transversal dispersive flux increases for a larger velocity difference.

Within each layer, the transversal dispersive flux is dependent on the layer velocity. This can be seen in figure 5b as the isoconcentration lines in the higher velocity layer are less curved. If $u_2$ is very low, then the curved iso-concentration lines will be more curved, and the medium behaves as if in the double layer limit. On the other hand, if $u_2$ is higher (but lower than $u_1$) then the curved iso-concentration lines will be less curved, and the medium behaves as if in the single layer limit. (But for higher $u_2$, the interaction between the layers will be less.)

### 3.3 Transverse dispersion number $N_{TD}$

With this understanding, a characterization of the stratified porous medium can be done using a dimensionless number. The transverse dispersion number is defined as

$$N_{TD} = \frac{t_l}{t_t} \tag{19}$$

with $t_l$ the time for the solute to cross the medium in the longitudinal direction and $t_t$ the time to cross the medium in the transverse direction [10].

These times can be approximately expressed in terms of the fluxes by

$$t_l = \frac{cL_x}{cu - D_l \frac{\partial c}{\partial x}}, \qquad t_t = \frac{cL_y}{D_t \frac{\partial c}{\partial y}}$$

Now define $y_D = \frac{y}{L_y}$, so that $\frac{\partial c}{\partial y_D} = L_y \frac{\partial c}{\partial y}$. Then $\frac{\partial c}{\partial y_D}$ is of the same order as $c$, and the longitudinal dispersion is negligible compared to the advection so that

$$N_{TD} \approx \frac{L_x}{L_y^2} \frac{D_t \frac{\partial c}{\partial y_D}}{cu} \approx \frac{L_x}{L_y^2} \frac{D_t}{u}$$

Lastly, to be able to calculate $N_{TD}$ from the parameters, $D_t = \alpha_t u + D_m$ is calculated using the lower velocity and for $u$ the higher velocity is taken (this can been seen as a lower bound on $N_{TD}$). Assuming furthermore that $D_m << \alpha_t u_2$, and that $u_2 < u_1$, this gives

$$N_{TD} \approx \alpha_t \frac{L_x}{L_y^2} \frac{u_2}{u_1} \tag{20}$$

This expression for $N_{TD}$ is used in [10], and will be referred to as Lake's $N_{TD}$.

#### 3.3.1 Discussion of $N_{TD}$

From definition (19) low values of $N_{TD}$ imply that the time for the species to cross the medium longitudinally is shorter than to do so transversely. If this is the case one might expect the stratified porous medium to behave like two separate layers.

Indeed, from (20) if $\alpha_t \to 0$ (from the above discussion the system then behaves as two layers), then $N_{TD} \to 0$. If the velocity difference between the layers increases, then $\frac{u_2}{u_1} \to 0$, and thus $N_{TD} \to 0$. Also if the medium is very thick, $L_y \to \infty$ or very short $L_x \to 0$, then $N_{TD} \to 0$.

In these limits, $N_{TD}$ behaves as expected. However, if $u_1 = u_2$, then $N_{TD}$ can be arbitrarily small, but the system still behaves as a single layer. Similarly, if $h_1$ or $h_2 \to L_y$, then $N_{TD}$ can take any value, but the system behaves as a single layer.

To solve the latter issue, the approximation $t_t$ can be replaced by

$$t_t = \frac{ch_1}{D_{t1}\frac{\partial c}{\partial y}} + \frac{ch_2}{D_{t2}\frac{\partial c}{\partial y}}$$

where $D_{tj} = \alpha_t u_j + D_m$. Further, if for $u$ the thickness-weighed average of the velocity is taken, then a similar derivation as above gives

$$\tilde{N}_{TD} = \frac{\alpha_t u_1 u_2}{h_1 u_2 + h_2 u_1} \frac{L_x}{h_1 u_1 + h_2 u_2} \tag{21}$$

This $\tilde{N}_{TD}$ is suggested to take the effect of varying thickness ratio into account. Suppose $h_1 = h_2 = L_y/2$ and $u_1 >> u_2$, then (21) becomes,

$$\tilde{N}_{TD} = 4\alpha_t \frac{L_x}{L_y^2} \frac{u_2}{u_1}$$

which differs from Lake's $N_{TD}$ by a factor of 4.

Lastly, assuming that $\alpha_l$ is in the order of $10^{-4}$ to $10^{-3}$ m, and $D_m$ is in the order of $10^{-9}$ $\mathrm{m^2\,s^{-1}}$, the velocity needs to be higher than roughly $10^{-3}$ $\mathrm{m\,s^{-1}}$ for the molecular diffusion to be neglected.

### 3.4 Transverse dispersion index $I_{TD}$

With the transverse dispersion number $N_{TD}$ (and $\tilde{N}_{TD}$) as a suggested dimensionless number to characterize the transverse dispersion in a two-layer stratified porous medium, this needs to be verified. For that, the transverse dispersion index $I_{TD}$ is introduced, with the desired property that for a system in the single-layer limit $I_{TD} = 1$ and in the double-layer limit $I_{TD} = 0$. $I_{TD}$ then effectively measures the behaviour of a system regarding the transverse dispersion.

To this end, for a given system a function $\rho$ is needed, of which the value lies between the value of $\rho$ for the two limits. Then $I_{TD}$ can be constructed from $\rho$ as

$$I_{TD} = \frac{\rho_{\text{system}} - \rho_{\text{double layer}}}{\rho_{\text{single layer}} - \rho_{\text{double layer}}} \tag{22}$$

Indeed, if the system behaves as a single layer, then $\rho_{\text{system}} = \rho_{\text{single layer}}$ and $I_{TD} = 1$. Similarly, if the system behaves as in the double layer limit, then $\rho_{\text{system}} = \rho_{\text{double layer}}$ and $I_{TD} = 0$. For any system, $I_{TD} \in [0, 1]$.

To find a well-defined $\rho$, first the amount of fluid volume injected at time $t$ is considered, which equals

$$\phi_1 h_1 u_1 t + \phi_2 h_2 u_2 t$$

This corresponds to the hatched area in figure 6. Expressed in units of pore volume, this is[6]

$$Q = \frac{\phi_1 h_1 u_1 + \phi_2 h_2 u_2}{L_x L_y \phi} t \tag{23}$$

which is called the pore volume injected (PVI). Since the concentration on the left boundary is constant, the PVI corresponds to the injected mass of the species. Although in figure 6 there is only transport due to advection, the PVI is also valid for media where there is longitudinal and/or transversal dispersion.

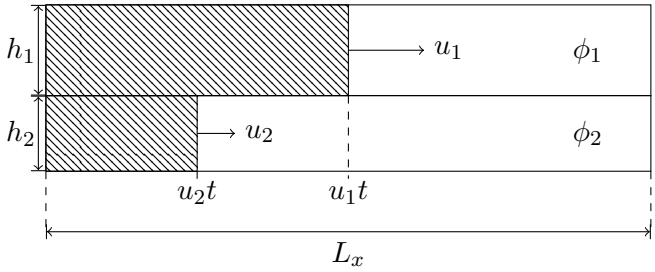



Figure 6: Pore Volume Injected.

Next, the effluent concentration $c_{\text{out}}$ of each layer is defined. For each layer, this is the $y$-averaged concentration at $x = L_x$,

$$c_{\text{out},1}(t) = \frac{1}{h_1} \int_0^{h_1} c(L_x, y, t)\ \mathrm{d}y, \quad c_{\text{out},2}(t) = \frac{1}{h_2} \int_{h_1}^{L_y} c(L_x, y, t)\ \mathrm{d}y \tag{24}$$

Then, a choice for $\rho$ is to take the PVI when the effluent concentration of the layer with largest fluid volume flux ($\phi h u$) equals 0.5. This can be calculated by determining time $t_{0.5}$ from

$$\begin{cases} c_{\text{out},1}(t_{0.5}) = 0.5, & \phi_1 h_1 u_1 > \phi_2 h_2 u_2 \\ c_{\text{out},2}(t_{0.5}) = 0.5, & \phi_1 h_1 u_1 < \phi_2 h_2 u_2 \end{cases} \tag{25}$$

So that

$$\rho = \frac{\phi_1 h_1 u_1 + \phi_2 h_2 u_2}{L_x L_y \phi} t_{0.5} \tag{26}$$

For the limit cases of single layer and double layer, expression can be given for $\rho$.

[6]The porosity is $\phi = \frac{V_p}{V} = \frac{V_p}{L_x L_y} = \frac{\phi_1 h_1 + \phi_2 h_2}{L_y}$, where $V_p$ is the pore volume.

### Single layer

For a system in the single-layer limit, the PVI simply becomes $Q = \frac{u't}{L_x}$. From (17)

$$c_{\text{out},1}(t) = c_{\text{out},2}(t) \approx \text{erfc}\left(\frac{L_x - u't}{\sqrt{4D_{\text{eff}}t}}\right) \tag{27}$$

so that the effluent concentration of both layers equals 0.5 when $t = t_{0,5} = \frac{L_x}{u'}$, following from the symmetry of the complementary error function. This yields that $\rho_{\text{single layer}} = 1$.

### Double layer

Similar for a system in the double-layer limit, the effluent concentrations can be determined from (18),

$$c_{\text{out},1}(t) \approx \text{erfc}\left(\frac{L_x - u_1 t}{\sqrt{4D_{l1}t}}\right), \quad c_{\text{out},2}(t) \approx \text{erfc}\left(\frac{L_x - u_2 t}{\sqrt{4D_{l2}t}}\right) \tag{28}$$

Again using the symmetry of the complementary error function, the time $t_{0.5}$ is given by,

$$\begin{cases} t_{0.5} = \frac{L_x}{u_1}, & \phi_1 h_1 u_1 > \phi_2 h_2 u_2 \\ t_{0.5} = \frac{L_x}{u_2}, & \phi_1 h_1 u_1 < \phi_2 h_2 u_2 \end{cases} \tag{29}$$

From this, $\rho_{\text{double layer}}$ can be calculated, and is given by

$$\rho_{\text{double layer}} = \begin{cases} \frac{\phi_1 h_1 u_1 + \phi_2 h_2 u_2}{u_1 L_y \phi}, & \phi_1 h_1 u_1 > \phi_2 h_2 u_2 \\ \frac{\phi_1 h_1 u_1 + \phi_2 h_2 u_2}{u_2 L_y \phi}, & \phi_1 h_1 u_1 < \phi_2 h_2 u_2 \end{cases} \tag{30}$$

Lastly, in the two limits, $\rho$ is not influenced by numerical diffusion (this will be explained in section 4.5). For example, writing $D_{\text{num}}$ for the numerical diffusion, the effluent concentration in the single layer would become

$$c_{\text{out},1}(t) = c_{\text{out},2}(t) \approx \text{erfc}\left(\frac{L_x - u't}{\sqrt{4(D_{\text{eff}} + D_{\text{num}})t}}\right)$$

However, because of the symmetry of the complementary error function, the time for the effluent concentration to become 0.5 remains $t_{0,5} = \frac{L_x}{u'}$. A similar argument goes for the double-layer limit effluent concentration.

# 4 Numerical Modeling of Dispersive Transport

In this chapter, the numerical method used to solve the transport of the solute through the two-layer stratified porous medium, as described by (12) in section 3.1, is presented.

First a general description of the finite volume method (FVM) will be given, followed by some definitions regarding the analysis of the numerical method. Then a discretization for the stratified porous medium is given, with an analysis of the method. A way to validate the numerical method using an analytical solution is then presented. Lastly, a description of the calculation of $I_{TD}$ (section 3.4) from the numerical solution is given.

## 4.1 Finite volume methods

In order to solve the advection-dispersion equation (12) the Finite Volume Method (FVM) is used. This section describes the application of the FVM to (12).

The domain $D$, on which the PDE is to be solved, is subdivided in in finite volumes $v_i \subset D$ called cells.[7], where $i$ denotes the index of the finite volume. Then in the FVM the partial differential equation is integrated over finite volume $v_i$. The FVM described in this section applies to any proper grid choice.

Equation (12) can be written as

$$\frac{\partial c}{\partial t} = \boldsymbol{\nabla} \cdot (\mathbf{D}\boldsymbol{\nabla} c - c\mathbf{u})$$

where $\mathbf{D}$ is the dispersion tensor. This is integrated over a volume $v_i$

$$\iint_{v_i} \frac{\partial c}{\partial t} \mathrm{d}\mathbf{x} = \iint_{v_i} (\boldsymbol{\nabla} \cdot (\mathbf{D}\boldsymbol{\nabla} c - c\mathbf{u}))\, \mathrm{d}\mathbf{x} = \oint_{\partial v_i} (\mathbf{D}\boldsymbol{\nabla} c - c\mathbf{u}) \cdot \mathbf{n} \mathrm{d}s$$

where the last step follows from the Gauss integral theorem, and $\mathbf{n}$ is the outward normal. If $c$ is continuously differentiable the order of integration and differentiation can be changed [8], and the entire equation is integrated over an interval $\Delta t$ from $t_n$ to $t_{n+1}$ to obtain

$$\int_{t_n}^{t_{n+1}} \frac{\partial}{\partial t} \iint_{v_i} c \, \mathrm{d}\mathbf{x}\mathrm{d}t = \int_{t_n}^{t_{n+1}} \oint_{\partial v_i} (\mathbf{D}\boldsymbol{\nabla} c - c\mathbf{u}) \cdot \mathbf{n} \mathrm{d}s \mathrm{d}t \tag{31}$$

Now, the volume average of concentration $c$ is

$$Q_i = \frac{1}{|v_i|} \iint_{v_i} c \, \mathrm{d}\mathbf{x} \tag{32}$$

[7] The finite volumes should be disjoint if they are not adjacent, $v_i \cap v_j = \emptyset,\ i \neq j$, and if they are adjacent then $v_i \cap v_j = \partial v_i \cap \partial v_j,\ i \neq j$. Moreover they should cover $D$, $\bigcup_j v_j = D$).

[8] Equation (12) is in fact derived from the mass conservation law in integral form, so that $c$ does not need to be continuously differentiable.

where $|v_i|$ is the volume of $v_i$. This can be substituted in equation (31) and after applying the fundamental theorem of calculus this yields

$$|v_i|\, Q_i^{n+1} - |v_i|\, Q_i^n = \int_{t_n}^{t_{n+1}} \oint_{\partial v_i} (\mathbf{D}\boldsymbol{\nabla} c - c\mathbf{u}) \cdot \mathbf{n} \mathrm{d}s \mathrm{d}t \tag{33}$$

where $Q_i^n$ is $Q_i$ at $t = t_n$. Now the time-averaged flux is introduced,

$$F_i^n = \frac{1}{\Delta t} \int_{t_n}^{t_{n+1}} \oint_{\partial v_i} (\mathbf{D}\boldsymbol{\nabla} c - c\mathbf{u}) \cdot \mathbf{n} \mathrm{d}s \mathrm{d}t \tag{34}$$

Substituting this in equation (33) and rearranging yields

$$Q_i^{n+1} = Q_i^n + \frac{\Delta t}{|v_i|} F_i^n \tag{35}$$

Note that this equation is still exact. The only information that is lost on $c$ is because of the averaging in (32). In the two-dimensional case, if $c$ is sufficiently smooth, then the (exact) cell average $Q_{ij}^n$ agrees with $c_{ij}^n = c(x_i, y_j, t_n)$ to $\mathcal{O}(\Delta x^2) + \mathcal{O}(\Delta y^2)$ [9] (with the index substituted by a double index $ij$). The challenge in numerically solving the advection-dispersion equation lies in properly approximating $F_i^n$.

For any approximation of $F_i^n$ equation (35) still is in conservation form since if the sum over all cells is taken, then all fluxes of adjacent cells cancel out and just the flux through the boundaries of the domain $D$ remains.

## 4.2 Two-dimensional discretization

In this section the numerical discretization of the model described in section 3.1 is given.

The domain, with length $L_x$ and height $L_y$, is divided in $N_x$ and $N_y$ intervals of length $\Delta x$ and $\Delta y$, respectively (see figure 7). The $x$-coordinate of each cell is denoted by the index $i$, and the $y$-coordinate by $j$.

As stated in the description of the model, the velocity field is given by $\mathbf{u} = (u(y), 0)^T$ (where the first component of the velocity is explicitly dependent on $y$). With this velocity field, the dispersion tensor becomes,

$$\mathbf{D} = \begin{pmatrix} \alpha_l u(y) + D_m & 0 \\ 0 & \alpha_t u(y) + D_m \end{pmatrix}$$

Substituting this in the time-averaged flux (34) gives

$$F_{ij}^n = \frac{1}{\Delta t} \int_{t_n}^{t_{n+1}} \underbrace{\oint_{\partial v_{ij}} \begin{pmatrix} (\alpha_l u(y) + D_m) \frac{\partial c}{\partial x} - c u(y) \\ (\alpha_t u(y) + D_m) \frac{\partial c}{\partial y} \end{pmatrix} \cdot \mathbf{n} \mathrm{d}s}_{\Phi_{i,j}} \mathrm{d}t$$

[9] $\frac{1}{\Delta x \Delta y} \int_{v_i} c(x, y, t) \mathrm{d}x \mathrm{d}y = c(x, y, t) + \mathcal{O}(\Delta x^2) + \mathcal{O}(\Delta y^2)$, with $v_i$ square with sides $\Delta x$ and $\Delta y$.

As mentioned in section 4.1, the aim in a finite-volume method is to properly discretize $F_{ij}^n$. The flux $\Phi_{i,j}$ through the boundary of $v_{i,j}$ can be divided into four parts as shown in figure 7,

$$\Phi_{i,j} = \Phi_{(i,j),1} + \Phi_{(i,j),2} + \Phi_{(i,j),3} + \Phi_{(i,j),4}$$

These partial fluxes, with the corresponding normal vectors, are given by

$$\begin{aligned}
\Phi_{(i,j),1} &= \int_{y_{j-\frac{1}{2}}}^{y_{j+\frac{1}{2}}} \left[ (\alpha_l u(y) + D_m) \frac{\partial c}{\partial x}(x_{i+\frac{1}{2}}, y) - c(x_{i+\frac{1}{2}}, y) u(y) \right] \mathrm{d}y \\
\Phi_{(i,j),2} &= \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \left[ (\alpha_t u(y_{j+\frac{1}{2}}) + D_m) \frac{\partial c}{\partial y}(x, y_{j+\frac{1}{2}}) \right] \mathrm{d}x \\
\Phi_{(i,j),3} &= -\int_{y_{j-\frac{1}{2}}}^{y_{j+\frac{1}{2}}} \left[ (\alpha_l u(y) + D_m) \frac{\partial c}{\partial x}(x_{i-\frac{1}{2}}, y) - c(x_{i-\frac{1}{2}}, y) u(y) \right] \mathrm{d}y \\
\Phi_{(i,j),4} &= -\int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \left[ (\alpha_t u(y_{j-\frac{1}{2}}) + D_m) \frac{\partial c}{\partial y}(x, y_{j-\frac{1}{2}}) \right] \mathrm{d}x
\end{aligned}$$

The discretization of the partial fluxes is elaborated in appendix A.1. Note that $u = u(y)$ is assumed, so that $D_l = D_l(y)$ and $D_t = D_t(y)$. If the velocity contains a discontinuity, then $D_l$ and $D_t$ are discontinuous as well. In that case, the grid is always chosen so that any discontinuity is on a cell interface. Then $D_l$ has no discontinuity on the cell boundary. And on the discontinuity, the harmonic average is taken of the values of $D_t$ above and below the discontinuity.

From the partial fluxes, the total flux can be written as a linear combination of all values of $v_{ij}$ and its neighbours,

$$\Phi_{i,j} = (a_1 + a_2 + a_3 + a_4)\, Q_{i,j} + a_5 Q_{i+1,j} - a_2 Q_{i,j+1} + a_6 Q_{i-1,j} - a_4 Q_{i,j-1} \tag{36}$$

(where the coefficients depend on $j$). Then the time average can be approximated by a time weighed average,

$$F_{i,j}^n \approx \beta \Phi_{i,j}^n + (1-\beta)\Phi_{i,j}^{n+1}$$

where $\beta \in [0,1]$. With the volume of each cell $|v_{ij}| = \Delta x \Delta y$, equation (35) becomes

$$Q_{ij}^{n+1} = Q_{ij}^n + \frac{\Delta t}{\Delta x \Delta y} F_{ij}^n$$

so that the full numerical scheme, with the coefficients from (36) and appendix A.1, is given by

$$Q_{ij}^{n+1} - \frac{\Delta t}{\Delta x \Delta y}(1-\beta)\Phi_{i,j}^{n+1} = Q_{ij}^n + \frac{\Delta t}{\Delta x \Delta y}\beta \Phi_{i,j}^n \tag{37}$$

For $\beta = 1$ the scheme is fully explicit.

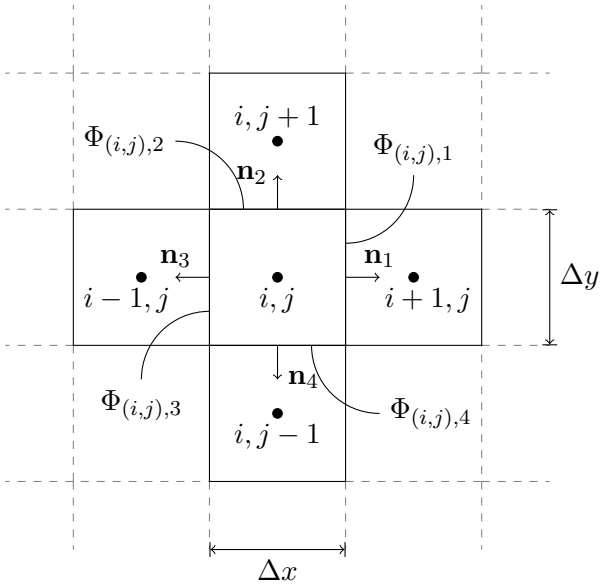



Figure 7: Two-dimensional grid at cell $i, j$.

To solve this linear system, note that it can be written as

$$B\mathbf{Q^{n+1}} = A\mathbf{Q^n} + \mathbf{a} \tag{38}$$

where $\mathbf{a}$ is a constant vector to account for the constant boundary conditions.

The linear system (38) is then solved in MATLAB using built in system solvers. The system is programmed as sparse matrices to improve computational efficiency. The MATLAB code which implements this system is included in appendix A.6.

### 4.3 Numerical boundary conditions

On the left boundary, $x = 0$, the concentration is at unity, $c(0, y, t) = 1$. Numerically, to implement this boundary condition, the average of the cells to the right of the left boundary of the domain, $Q^n_{1,j}$, and the left virtual cells[10] $Q^n_{0,j}$ outside of the domain is taken,

$$\frac{Q^n_{1,j} + Q^n_{0,j}}{2} = 1, \quad j \in \{1, ..., N_y\} \tag{39}$$

At all other boundaries, the gradient is zero, so that the difference between the cells on

[10]Virtual cells are cell which are not in the numerical domain, but are defined so that the boundary conditions can be satisfied.

the boundary of the domain and the virtual cells is zero,

$$Q^n_{N_x,j} - Q^n_{N_x+1,j} = 0, \quad j \in \{1, ..., N_y\} \tag{40}$$

$$Q^n_{i,1} - Q^n_{i,0} = Q^n_{i,N_y+1} - Q^n_{i,N_y} = 0, \quad i \in \{1, ..., N_x\} \tag{41}$$

These boundary conditions are substituted in (37) (for cells on the boundary). From this vectors **a** and **b** of (38) can be calculated.

Lastly, the right (artificial) boundary condition (imposed for numerical reasons), creates an additional error in the effluent concentration. The effect of artificial boundary conditions is for example considered in [2], [4] and [12]. To understand this, consider the analytical solution (17), of which the derivative at $x = L_x$ is not zero at all times. To reduce the effect of the artificial boundary condition, the domain is extended to $2L_x$, while the effluent concentration is still measured at $x = L_x$.

### 4.4 Definitions

In this section (which largely follows [11]) the definitions of convergence, consistency and stability of numerical methods are given. In the following only problems on an unbounded domain are considered, since the introduction of boundary conditions leads to additional difficulties in the analysis.

To analyze the convergence, consistency and stability, first the error needs to be quantified. Let $q^n_i$ be the exact value at point $x_i$ and time $t_n$ which is approximated by $Q^n_i$, and let $T = N_t\Delta t$ be the time over which the method is computed. The *global error* is then defined by

$$E^{N_t} = Q^{N_t} - q^{N_t}. \tag{42}$$

For finite volume methods, the exact value at $(\mathbf{x}_i, t_n)$ is given by $q^n_i = \frac{1}{v_i}\int_{v_i} c(\mathbf{x}, t_n)\ \mathrm{d}\mathbf{x}$, where $c$ is the exact solution. However, for numerical analysis it is easier to use $c^n_i$ as the exact value, as can been seen from the Taylor expansions used in explicitly calculating the local truncation error later on. Assuming $c$ is sufficiently smooth, this agrees with the cell average to $\mathcal{O}(\Delta x^2) + \mathcal{O}(\Delta y^2)$.

It is throughout this thesis assumed that $\Delta t$, $\Delta x$ and $\Delta y$ are related in some manner as the grid is refined. With the global error defined, the definition of convergence and accuracy can be given.

**Definition 1** (Convergence and accuracy)**.** Let $||\cdot||$ be some norm and $E^{N_t}$ the global error. A numerical method is convergent at time $T = N_t\Delta t$ in the norm $||\cdot||$ if

$$\lim_{\Delta t \to 0} ||E^{N_t}|| = 0 \tag{43}$$

The method is called accurate of order $s$ if

$$||E^{N_t}|| = \mathcal{O}(\Delta t^s) \quad \text{as } \Delta t \to 0 \tag{44}$$

Convergence depends on the norm being used, and the norms most commonly used are given by the $p$-norm (here in one-dimension)

$$||E||_p = \left( \Delta x \sum_{i=-\infty}^{\infty} |E_i|^p \right)^{1/p} \tag{45}$$

In this thesis the 2-norm is used.

Following from the fundamental theorem of numerical methods for differential equations, convergence is implied if the method is stable and consistent. Both definitions will be given below.

### 4.4.1 Local truncation error and consistency

A explicit one-step numerical method can be written in general as

$$Q^{n+1} = \mathcal{N}(Q^n) \tag{46}$$

Where $\mathcal{N}(\cdot)$ is the numerical operator mapping the numerical solution $Q^n$ at time $t_n$ to the numerical solution $Q^{n+1}$ at the next time step.
**Definition 2** (Local truncation error)**.** The local truncation error is defined as

$$\tau^n = \frac{1}{\Delta t} \left[ \mathcal{N}(q^n) - q^{n+1} \right]. \tag{47}$$

The local truncation error gives the error introduced when applying the numerical method to the exact solution. The local truncation error gives an indication of the magnitude of the global error, and in case it is stable, the order of accuracy. A more proper motivation of this definition can be found in Appendix A.2.
**Definition 3** (Consistency)**.** A numerical method written as (46) is consistent with the differential equation it tries to approximate if for the local truncation error $\tau^n$

$$\lim_{\Delta t \to 0} \tau^n = 0 \tag{48}$$

### 4.4.2 Stability

As can been seen from Appendix A.2, stability aims to give a bound on $\mathcal{N}(q^n + E^n) - \mathcal{N}(q^n)$, the effect of the numerical operator on a perturbation or error $E^n$.

There are multiple ways of proving stability, such as showing that a numerical operator is contractive in some norm, using Lax-Richtmyer stability or performing a von Neumann Analysis, or using eigenvalue expansion.

Is this thesis Von Neumann stability is used, which uses the 2-norm. The imaginary unit is denoted here with $\iota$ to avoid confusion with the grid index $i$. In the Von

Neumann stability analysis (here in one spatial dimension, multiple spatial dimensions is similar), $Q_i^n = \exp(\iota\xi i\Delta x)$ is substituted in the numerical method with $\beta = 1$. This yields an expression of the form $Q_i^{n+1} = g(\xi, \Delta x, \Delta t)Q_i^n$, where $g(\xi, \Delta x, \Delta t)$ is called the amplification factor of the numerical method. Then the method is stable if $|g(\xi, \Delta x, \Delta t)| \leq 1$.

### 4.4.3 TVD and Monotonicity-preserving

Stability of numerical schemes, especially in multiple dimension, can be hard to prove. Therefore, other concepts are introduced, TVD and monotonicity-preserving. First, the total variation of a numerical scheme is defined by

$$\mathrm{TV}(Q^n) = \sum_i \left|Q_i^n - Q_{i-1}^n\right| \tag{49}$$

Total Variation is here only defined for one spatial dimension.
**Definition 4** (Total Variation Diminishing)**.** A numerical scheme is called total variation diminishing (TVD) if

$$\mathrm{TV}(Q^{n+1}) \leq \mathrm{TV}(Q^n) \quad \forall n \tag{50}$$

If a scheme is TVD, the in each time step, the total variation does not grow.

Further, any linear numerical scheme can be written as[11]

$$Q_i^{n+1} = \sum_k \gamma_k Q_{i+k}^n \tag{51}$$

**Lemma 1.** *A linear numerical scheme, written in the form of* (51)*, is TVD if* $\sum_k |\gamma_k| \leq 1$*.*

*Proof.* Assume $\sum_k |\gamma_k| \leq 1$. Using the triangle inequality, the total variation at $n+1$ can be bounded by

$$\begin{aligned}
\mathrm{TV}(Q^{n+1}) &= \sum_i \left|Q_i^{n+1} - Q_{i-1}^{n+1}\right| = \sum_i \left|\sum_k \gamma_k Q_{i+k}^n - \sum_k \gamma_k Q_{i+k-1}^n\right| \\
&\leq \sum_i \sum_k |\gamma_k| \left|Q_{i+k}^n - Q_{i+k-1}^n\right| = \sum_k |\gamma_k| \sum_i \left|Q_{i+k}^n - Q_{i+k-1}^n\right| \\
&= \mathrm{TV}(Q^n) \sum_k |\gamma_k| \leq \mathrm{TV}(Q^n)
\end{aligned} \tag{52}$$

where the fact is used that the total variation is invariant under translation (over index $k$). □

[11] $\gamma_k$ can also depend on $i$.

Next to TVD, there is the concept of monotonicity-preserving.
**Definition 5** (Monotonicity-preserving ([11]))**.** A numerical scheme is called monotonicity-preserving if

$$Q_i^n \geq Q_{i+1}^n \quad \forall i \implies Q_i^{n+1} \geq Q_{i+1}^{n+1} \quad \forall i \tag{53}$$

If a scheme is monotonicity-preserving, then no oscillations can develop if the scheme was initially monotone. Unlike TVD, monotonicity-preserving can also be defined for multiple spatial dimensions [14].
**Theorem 2** (Godunov's Theorem ([20]))**.** *A linear numerical scheme in the form of* (51) *is monotonicity-preserving if and only if* $\gamma_k \geq 0, \ \forall k$.

Godunov's Theorem is also valid for multiple spatial dimensions [14].
**Theorem 3** ([11])**.** *A linear numerical scheme that is TVD is also monotonicity-preserving.*
**Lemma 4.** *A linear numerical scheme that is monotonicity-preserving and with* $\sum_k \gamma_k = 1$*, is TVD.*

*Proof.* From Godunov's Theorem, $\gamma_k \geq 0, \ \forall k$, so that

$$\sum_k |\gamma_k| = \sum_k \gamma_k = 1$$

Then lemma 1 completes the proof. □

### 4.5 Numerical analysis of the two-dimensional discretization

In this section the numerical analysis is given of the two-dimensional discretization (37), as described in section 4.2, using definitions from section 4.4. The analysis here is performed for $\beta = 1$ (explicit). Stability for this scheme will not be proved.

#### Local truncation error

The local truncation error of the numerical scheme is given by

$$\begin{aligned}\tau = u(y_j)\frac{\Delta x}{2}\frac{\partial^2 q}{\partial x^2}(x_i, y_j, t_m) \\ - \frac{\Delta t}{2}\frac{\partial^2 q}{\partial t^2}(x_i, y_j, t_m) + u(y_j)\frac{\Delta x^2}{6}\frac{\partial^3 q}{\partial x^3}(x_i, y_j, t_m) + \mathcal{O}\left(\Delta t^2, \Delta x^2, \Delta y^2\right)\end{aligned} \tag{54}$$

where the remainder depends on higher order derivatives of $q$. The derivation of this can be done using Taylor expansion (assuming $c$ is sufficiently smooth), and is shown in appendix A.3. This expression for the local truncation error is not valid on a discontinuity.

Under the assumption that $\Delta x \to 0$ and $\Delta y \to 0$, if $\Delta t \to 0$, then this numerical scheme is consistent. The scheme is first order in time and $x$, and second order in $y$.

By repeated differentiation of the differential equation with respect to $x$ and $t$, the second derivative with respect to $t$ in the local truncation error can be expressed as a second derivative with respect to $x$ (and some higher order derivatives)[12]. This yields

$$\tau = \underbrace{\left( u(y_j)\frac{\Delta x}{2} - u(y_j)^2\frac{\Delta t}{2} \right)}_{D_{\text{num}}} \frac{\partial^2 q}{\partial x^2}(x_i, y_j, t_m) + \mathcal{O}\left(\Delta t, \Delta x, \Delta y^2\right) \tag{55}$$

The underbraced term is the numerical diffusion, $D_{\text{num}}$. The biggest numerical diffusion occurs in the layer with the highest velocity. For accurate solutions of the advection-dispersion equation the numerical diffusion needs to be small compared to the dispersion coefficient. So

$$\frac{D_{\text{num}}}{D_{l1}} = \frac{u_1\Delta x - u_1^2\Delta t}{2\alpha_l u_1 + 2D_m} \approx \frac{\Delta x - u_1\Delta t}{2\alpha_l}$$

needs to be small. However, if $\Delta x = u_1\Delta t$, then there is no numerical diffusion, but the scheme might be unstable. In the $y$-direction, there is no numerical diffusion, as there is no second derivative with respect to $y$ in the local truncation error.[13]

Further, if $u\Delta t > \Delta x$, then the numerical diffusion becomes negative, and the scheme might become unstable. (This is the CFL-condition [11]).

**Monotonicity-preserving**

The numerical scheme monotonicity-preserving if all coefficients of (37) are positive. The only coefficient that can be negative, is that of $Q_{ij}^n$. This leads to the condition

$$1 + \frac{\Delta t}{\Delta x \Delta y}(a_1 + a_2 + a_3 + a_4) \geq 0$$

Filling out the coefficients yields

$$\begin{aligned} 1 \geq \frac{\Delta t}{\Delta x \Delta y}\Bigg( & \left[ 2(\alpha_l u(y_j) + D_m)\frac{\Delta y}{\Delta x} + \Delta y u(u_j) \right] \\ +2\frac{\Delta x}{\Delta y} & \left[ \frac{(\alpha_t u(y_{j-1}) + D_m)(\alpha_t u(y_j) + D_m)}{\alpha_t(u(y_{j-1}) + u(y_j)) + 2D_m} \right] + 2\frac{\Delta x}{\Delta y} \left[ \frac{(\alpha_t u(y_{j+1}) + D_m)(\alpha_t u(y_j) + D_m)}{\alpha_t(u(y_{j+1}) + u(y_j)) + 2D_m} \right] \Bigg) \end{aligned}$$

This has to be true for all $j$. Therefore, the maximum velocity $u_{\max}$ is used, so that

$$1 \geq 2(\alpha_l u_{\max} + D_m)\frac{\Delta t}{\Delta x^2} + u_{\max}\frac{\Delta t}{\Delta x} + 2(\alpha_t u_{\max} + D_m)\frac{\Delta t}{\Delta y^2} \tag{56}$$

This is a sufficient condition for the scheme to be monotonicity-preserving.

[12] Using a subscript to indicate the derivative and writing here $u_j = u(y_j)$, $q_{tt} = u_j^2 q_{xx} - u_j D_{lj} q_{xxx} - u_j D_{tj} q_{yyx} + D_{lj} q_{xxt} + D_{tj} q_{yyt} = u_j^2 q_{xx} - 2u_j D_{lj} D_{tj} q_{xxx} - 2u_j D_{tj} q_{yyy} + D_{lj}^2 q_{xxxx} + D_{tj}^2 q_{yyyy} + D_{lj} D_{tj} q_{xxyy}$

[13] In fact, every even derivative with respect to $x$ (or $y$) in the local truncation error causes numerical diffusion. However, the higher order derivatives are $\mathcal{O}(\Delta x^2)$ or higher (or $\mathcal{O}(\Delta y^2)$ or higher).

## 4.6 One-dimensional discretization

Since stability is hard to prove for the two-dimensional scheme (37), this will be investigated for advection-dispersion equation in one dimension

$$\frac{\partial c}{\partial t} + u\frac{\partial c}{\partial x} = (\alpha_l u + D_m)\frac{\partial^2 c}{\partial x^2}$$

(compare with equation (14)).

With a similar discretization as in section 4.2, the numerical scheme is given by

$$\begin{cases} Q_i^{n+1} - \frac{\Delta t}{\Delta x}(1-\beta)\Phi_i^{n+1} = Q_i^n + \frac{\Delta t}{\Delta x}\beta\Phi_i^n \\ \Phi_i^n = -\left[D_l\frac{Q_i^n - Q_{i-1}^n}{\Delta x} - u\left(\theta Q_{i-1}^n + (1-\theta)Q_i^n\right)\right] + \left[D_l\frac{Q_{i+1}^n - Q_i^n}{\Delta x} - u\left(\theta Q_i^n + (1-\theta)Q_{i+1}^n\right)\right] \end{cases} \tag{57}$$

with $\beta \in [0,1]$ and $\theta \in [0,1]$. For $\beta = 1$ the scheme is explicit, and for $\theta = 1$ the scheme is fully upwind. For $\beta = 1$ the scheme becomes

$$Q_i^{n+1} = \left(1 + \frac{\Delta t}{\Delta x}\left[u(1-2\theta) - \frac{2D_l}{\Delta x}\right]\right)Q_i^n + \frac{\Delta t}{\Delta x}\left(u\theta + \frac{D_l}{\Delta x}\right)Q_{i-1}^n + \frac{\Delta t}{\Delta x}\left(u(\theta-1) + \frac{D_l}{\Delta x}\right)Q_{i+1}^n$$

Writing $\sigma = \frac{u\Delta t}{\Delta x}$ (Courant number) and $\delta_l = \frac{D_l \Delta t}{\Delta x^2}$ (dispersion number),

$$Q_i^{n+1} = \underbrace{(1 + \sigma(1-2\theta) - 2\delta_l)}_{\gamma_0}Q_i^n + \underbrace{(\sigma\theta + \delta_l)}_{\gamma_{-1}}Q_{i-1}^n + \underbrace{(\sigma(\theta-1) + \delta_l)}_{\gamma_1}Q_{i+1}^n \tag{58}$$

### 4.6.1 Numerical analysis of the one-dimensional discretization

#### Monotonicity-preserving and TVD

The numerical scheme (58) is monotonicity-preserving if all coefficients $\gamma_k$ are positive. Assuming $u \geq 0$, $\gamma_{-1}$ is unconditionally larger than or equal to zero. Also $\gamma_0 \geq 0$ if

$$1 \geq \sigma(2\theta - 1) + 2\delta_l \tag{59}$$

and $\gamma_1 \geq 0$ if

$$\sigma(\theta - 1) + \delta_l \geq 0 \tag{60}$$

In case the method is fully upwind ($\theta = 1$), then $\sigma + 2\delta_l \leq 1$ is sufficient for the scheme to be monotonicity-preserving. This is similar to the monotonicity-preserving condition (56) of the two-dimensional scheme.

Further, since $\gamma_0 + \gamma_{-1} + \gamma_1 = 1$, the scheme is also TVD under the same conditions.

**Stability in the 2-norm**

For the von Neumann analysis, $Q_i^n = \exp(\iota\xi i\Delta x)$ is substituted in (58). Using the property that $Q_{i+m}^n = \exp(\iota\xi m\Delta x)Q_i^n,\ m \in \mathbb{N}$, this yields

$$\begin{aligned} g(\xi, \Delta x, \Delta t) &= 1 + \sigma(1-2\theta) - 2\delta_l + (\sigma\theta + \delta_l)\, e^{-\iota\xi\Delta x} + (\sigma(\theta-1) + \delta_l)\, e^{\iota\xi\Delta x} \\ &= 1 + (\sigma(2\theta-1) + 2\delta_l)\,(\cos(\iota\xi\Delta x) - 1) - \iota\sigma\sin(\iota\xi\Delta x) \end{aligned}$$

Then $|g(\xi, \Delta x, \Delta t)| \leq 1,\ \forall\xi$, leads to the condition (see appendix A.4)

$$\sigma^2 \leq \sigma(2\theta - 1) + 2\delta_l \leq 1 \tag{61}$$

Second inequality of this stability condition is the same as (59). In case the method is fully upwind ($\theta = 1$), then stability is a stronger condition than monotonicity-preserving. For $\theta \neq 1$, stability does not imply monotonicity-preserving, as (60) might not be satisfied.

Rewriting this using $\delta_l = \frac{\alpha_l}{\Delta x}\sigma + \frac{D_m\Delta t}{\Delta x^2} = \frac{\alpha_l}{\Delta x}\sigma + \delta_m$ (since $D_l = \alpha_l u + D_m$), the stability condition becomes

$$\sigma^2 \leq \sigma\left(2\theta - 1 + 2\frac{\alpha_l}{\Delta x}\right) + 2\delta_m \leq 1$$

From this it can be seen that larger $\alpha_l$ requires $\sigma$ to be smaller for the scheme to remain stable.

Lastly, consider

$$\frac{\partial c}{\partial t} = (\alpha_t u + D_m)\frac{\partial^2 c}{\partial y^2}$$

with a similar derivation of the numerical scheme and stability as above and introducing $\delta_t = \frac{D_t\Delta t}{\Delta y^2}$, the stability condition becomes

$$2\delta_t \leq 1 \tag{62}$$

(61) and (62) both give a stability condition in one direction of the two dimensional scheme.

**Local truncation error**

The local truncation error of scheme (58) is given by

$$\begin{aligned} \tau^n = \frac{1}{\Delta t}\Bigg[ \left(1 + \frac{\Delta t}{\Delta x}\left[u(1-2\theta) - \frac{2D_l}{\Delta x}\right]\right) q_i^n + \frac{\Delta t}{\Delta x}\left(u\theta + \frac{D_l}{\Delta x}\right) q_{i-1}^n \\ + \frac{\Delta t}{\Delta x}\left(u(\theta-1) + \frac{D_l}{\Delta x}\right) q_{i+1}^n - q_i^{n+1}\Bigg] \end{aligned} \tag{63}$$

where $q_i^n = (x_i, t_n)$ is the value of the exact solution at $(x_i, t_n)$. The values $q_i^{n+1},\ q_{i+1}^n$ and $q_{i-1}^n$ can be expanded in Taylor series about $(x_i, t_n)$ (see Appendix A.5), assuming

it is sufficiently smooth. After common terms are canceled, the local truncation error becomes[14]

$$\tau^n = D_l q_{xx} - (1+\theta) u q_x + \left(\theta - \frac{1}{2}\right) \Delta x \; u q_{xx} + \theta u q_x - q_t - \frac{\Delta t}{2} q_{tt} + \mathcal{O}(\Delta t^2) + \mathcal{O}(\Delta x^2) \quad (64)$$

Since $q$ is a solution to the equation ($D_l q_{xx} - u q_x = q_t$), it follows that

$$\tau^n = \left(\theta - \frac{1}{2}\right) \Delta x \; u q_{xx} - \frac{\Delta t}{2} q_{tt} + \mathcal{O}(\Delta t^2) + \mathcal{O}(\Delta x^2) \quad (65)$$

This local truncation error vanishes as $\Delta t \to 0$ ($\Delta t$ and $\Delta x$ are assumed related), for all values of $\theta$, so that this method is consistent.

Now, $q_{tt}$ can also be rewritten in term of higher order partial derivatives of $q$, by differentiating the equation $q_t = -u q_x + D_l q_{xx}$ with respect to $t$ and $x$ multiple times and assuming that $q$ is sufficiently smooth such that the order of differentiating can be interchanged. In this way, it can be found that

$$q_{tt} = u^2 q_{xx} - u D_l q_{xxx} + D_l q_{txx} = u^2 q_{xx} - 2u D_l q_{xxx} + D_l^2 q_{xxxx}$$

Substituting this in (65) gives after rearranging

$$\tau^n = \Delta x \; \left(\theta - \frac{1}{2} - \frac{u \Delta t}{2 \Delta x}\right) u q_{xx} \; + \frac{\Delta t}{2} \left(2 u D q_{xxx} - D^2 q_{xxxx}\right) + \mathcal{O}(\Delta t^2) + \mathcal{O}(\Delta x^2) \quad (66)$$

Now, if $\theta = \frac{1}{2} + \frac{u \Delta t}{2 \Delta x} = \frac{\sigma + 1}{2}$, then the first term cancels, and the method becomes second order in space. The monotonicity-preserving condition then becomes

$$\sigma \leq \sigma^2 + 2\delta_l \leq 1$$

and the stability condition becomes

$$\sigma^2 \leq \sigma^2 + 2\delta_l \leq 1$$

Since $\sigma \leq 1$, $\sigma^2 \leq \sigma$, so that if it is monotonicity-preserving, then it is stable (but the converse is not true).

For $\theta = 1$, comparing this to the two-dimensional local truncation error (55), these expressions are similar. The numerical diffusion is a result of the first-order discretization of the advection term.

### 4.7 Validation of the numerical method

To validate the numerical method, an analytical solution for specific boundary and initial conditions and constant velocity is used. Although this analytical solution does not say anything about a discontinuous velocity, it can be used to check whether the (implementation of the) numerical method is behaving as it should, and gives an indication of the global error.

[14] The sub- and superscripts $i$ and $n$ are omitted for clarity. $q_x$, $q_{xx}$, $q_t$ and $q_{tt}$ are evaluated in $(x_i, t_n)$.

**Analytical solution**

Suppose the spatial domain is $[0, \infty) \times \mathbb{R}$, $u$ is constant with respect to $x$ and $y$, and the initial and boundary conditions are given by

$$c_\infty(0, y, t) = \begin{cases} c_0, & |y| \leq a \\ 0, & |y| > a \end{cases} \tag{67}$$

$$\lim_{y \to \pm\infty} \frac{dc_\infty}{dy} = 0 \tag{68}$$

$$\lim_{x \to \infty} \frac{dc_\infty}{dx} = 0 \tag{69}$$

Then the analytical solution of (14) with these boundary conditions is given by ([3])

$$c_\infty(x, y, t) = \frac{c_0 x}{4\sqrt{\pi D_l}} \int_0^t \exp\left(\frac{-(x - \bar{u}\tau)^2}{4D_l\tau}\right) \tau^{-\frac{3}{2}} \left[\operatorname{erf}\left(\frac{a - y}{2\sqrt{D_t\tau}}\right) + \operatorname{erf}\left(\frac{a + y}{2\sqrt{D_t\tau}}\right)\right] \mathrm{d}\tau \tag{70}$$

where $D_l = \alpha_l u + D_m$ and $D_t = \alpha_t u + D_m$. The subscript $\infty$ is to denote that this solution is on an infinite domain in the $y$-direction.

Since $c$ is assumed to be sufficiently smooth, $c_\infty$ can be used to find a solution to the same problem on the spatial domain $\mathbb{R}^+ \times [-L_y/2, L_y/2]$ with boundary conditions

$$c(0, y, t) = \begin{cases} c_0, & |y| \leq a \\ 0, & |y| > a \end{cases} \tag{71}$$

$$\frac{dc}{dy}(x, \pm L_y/2, t) = 0 \tag{72}$$

$$\lim_{x \to \infty} \frac{dc}{dx} = 0 \tag{73}$$

assuming that $a \leq L_y/2$. This solution is given by

$$c(x, y, t) = \sum_{n=-\infty}^{\infty} c_\infty(x, y + nL_y, t) \tag{74}$$

It can be verified that this solution satisfies the boundary conditions at $y = \pm L_y$. This solution can be used to validate the numerical method and give an indication of the global error.

## 4.8 Numerical computation of $I_{TD}$

This section shows how $I_{TD}$ (see section 3.4) is calculated from the numerical solution. Suppose the discontinuity of the average fluid velocity is between the cells with indices

$j = N_b$ and $j = N_b + 1$ (so that $y_j$ is in layer 1 if $j \leq N_b$). Then for the average outflow concentration of layer 1 ($c_{\text{out},2}(t_n)$ is similar) is approximated by

$$\begin{aligned} c_{\text{out},1}(t_n) =& \frac{1}{h_1} \int_0^{h_1} c(L_x, y, t_n) \mathrm{d}y = \frac{1}{h_1} \sum_{j=1}^{N_b} \int_{(j-1)\Delta y}^{j\Delta y} c(L_x, y, t_n) \mathrm{d}y \approx \\ \approx& \frac{1}{h_1} \sum_{j=1}^{N_b} \frac{\Delta y}{2} \left( Q^n_{N_x,j} + Q^n_{N_x+1,j} \right) = \frac{1}{2N_b} \sum_{j=1}^{N_b} Q^n_{N_x,j} + Q^n_{N_x+1,j} \end{aligned}$$

Then at a certain time $t_m$, $c_{\text{out},1}(t_m) < 0.5 < c_{\text{out},1}(t_{m+1})$. From this a linear interpolation is done to determine $t_{0.5}$, and subsequently $\rho_{\text{system}}$ (from (26)). Further, $\rho_{\text{single layer}} = 1$, and $\rho_{\text{double layer}}$ is calculated from (30). Using equation (22), $I_{TD}$ follows follows from these values.

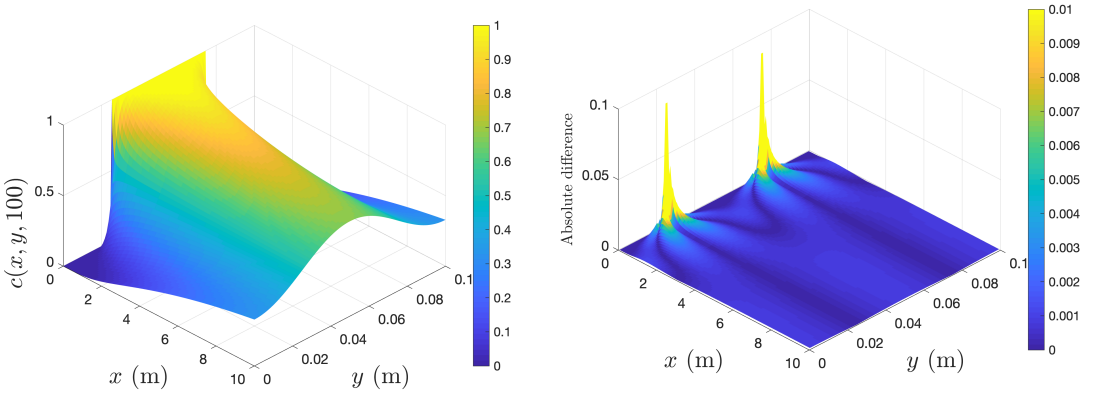
# 5 Results

This section gives results of the numerical simulations which were used to simulate the two-layered porous medium as described in section 3 using the numerical method from section 4. MATLAB code is included in appendix A.6, on which the scripts are based which give these results.

First the numerical method is validated using an analytical method to see if it behaves as expected. Next, the calculation of $I_{TD}$ using the numerical simulations is verified. Lastly, different sets of simulations are preformed, in which only one physical parameter varies. This is done to investigate for which values of $N_{TD}$ and $\tilde{N}_{TD}$ the system behaves as one single layer, as two separate layers, or is in the transition zone between the two limit cases.

## 5.1 Validation of the numerical method

The numerical method was validated using the analytical solution as described in section 4.7.[15] For example, figure 8a shows the numerical solution of the problem described in section 4.7, with certain physical parameters and $a = 0.5L_y$. Figure 8b shows the absolute difference between the numerical and the analytical solution for the same problem. The maximum absolute difference here is 0.09. This is, however, close to the discontinuity in the left boundary condition. Away from the left boundary, the absolute difference is below 0.5%. Different sets of physical parameters yielded similar results, and the numerical method behaved as expected.




(a) Numerical solution

(b) The absolute difference between the numerical solution and the analytical solution.

Figure 8: Example of the validation of the numerical method using an analytical solution.

[15] In the validation, the concentration on the left boundary ($x = 0$) was not constant at unity. Otherwise, there is no transverse dispersion, as the solution would be independent of $y$.

## 5.2 Verification of $I_{TD}$

In this section, $I_{TD}$ from section 3.4 is verified using numerical simulations. The calculation of $I_{TD}$ from the numerical simulation is described in section 4.8

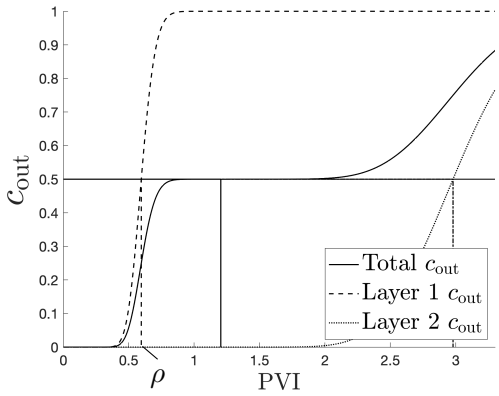
Figure 9 shows the effluent concentrations of three numerical simulations expressed in PVI. The effluent concentrations are plotted for both layers, and for the layers combined.[16]. All three runs have the same physical and numerical parameters, except for the transverse dispersion coefficient, $D_t$. In these runs, $\phi_1 h_1 u_1 > \phi_2 h_2 u_2$, so that the effluent concentration of layer 1 is used in the calculation of $\rho$ (equation (26)) and $I_{TD}$ (equation (22)). For every run in figure 9, $\rho = \rho_{\text{system}}$ is indicated. Further, for all three runs $\rho_{\text{single layer}} = 1$, and $\rho_{\text{double layer}} = 0.6$ (calculated using (30)).

Figure 9a shows the effluent concentrations of a numerical simulation where $D_t = 0$, corresponding to the double-layer limit. From the numerical simulation, the effluent concentration of layer 1 reaches the value of 0.5 when $\rho = \rho_{\text{system}} = 0.60$, as can be seen in the figure. The resulting transverse dispersion index (equation (22)) is $I_{TD} = 0.01$. This is close to zero, as was expected for the double-layer limit. The slight error might be caused by the numerical error, or by the fact that (28) used to calculate $\rho_{\text{double layer}}$ is an approximation.

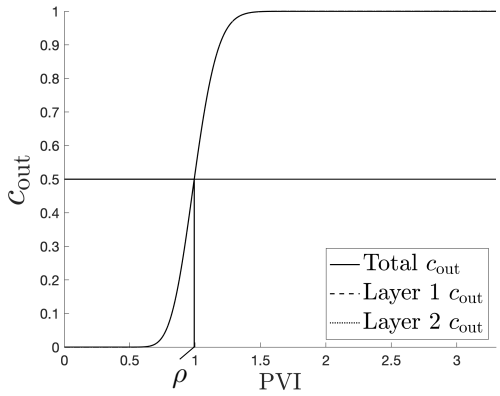
Similarly, figure 9b shows the effluent concentrations of a run where $D_t$ was large, corresponding to the single-layer limit. In this plot, the three different effluent concentrations cannot be distinguished, as both layer have the same concentration. From the figure, the effluent concentration of layer 1 reaches the value of 0.5 when $\rho = \rho_{\text{system}} = 0.99$. The resulting transverse dispersion index is $I_{TD} = 0.98$, which is close to one, as was expected for the double-layer limit.

Lastly, figure 9c shows the effluent concentrations of a run where there was some transverse dispersion. This is an example of what can be expected from an actual two-layer system. From the figure, the effluent concentration of layer 1 reaches the value of 0.5 when $\rho = \rho_{\text{system}} = 0.78$. The resulting transverse dispersion index is $I_{TD} = 0.44$.

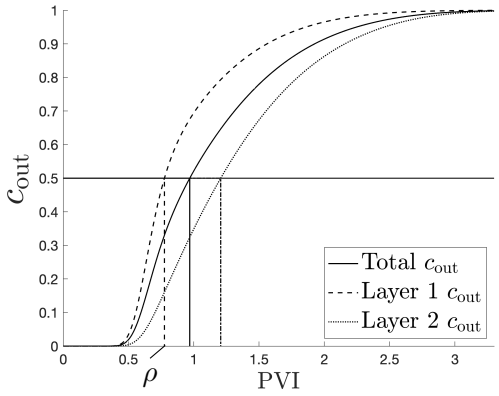
[16] The total effluent concetration is $c_{\text{out}} = \frac{1}{L_y} \int_0^{L_y} c(L_x, y, t) \mathrm{d}y = \frac{h_1}{L_y} c_{\text{out},1} + \frac{h_2}{L_y} c_{\text{out},2}$




(a) $D_t = 0$, $N_{TD} = 0$, $I_{TD} = 0.01$.




(b) Large $D_t$, $N_{TD} = 1000$, $I_{TD} = 0.98$. All three effluent concentrations are plotted but cannot be distinguished (as is expected in the single-layer limit).




(c) Some $D_t$, $N_{TD} = 0.1$, $I_{TD} = 0.44$.

Figure 9: The effluent concentrations of three numerical simulations, expressed in PVI. All three simulations have the same physical and numerical parameters, except for the transverse dispersion coefficient, $D_t$. The $N_{TD}$ used is Lake's.

## 5.3 $I_{TD}$ versus $N_{TD}$

Figure 10 shows an example of a numerical solution at a certain time. Different simulations were performed with varying physical and numerical parameters. From the numerical solutions, the $I_{TD}$ was calculated as described in section 4.8.

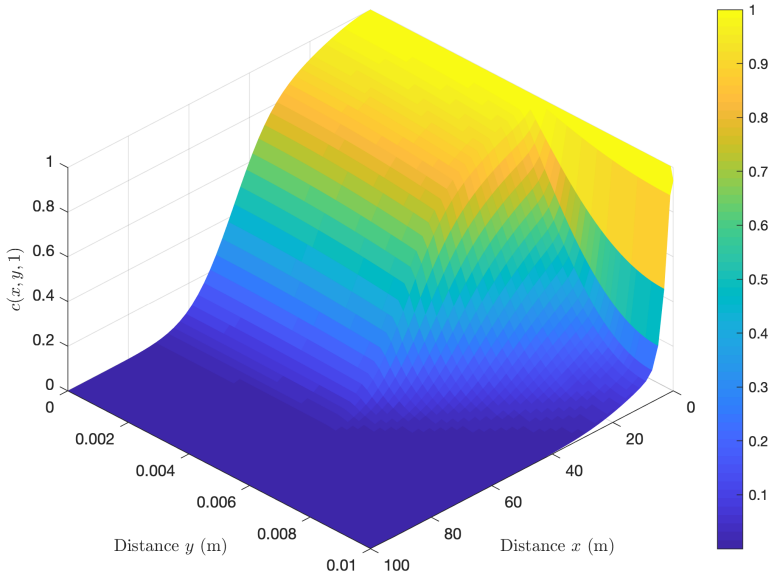



Figure 10: Example of numerical solution at $t = 1$ s. This solution has $N_{TD} = 0.07$, and is one of the simulations from figure 11

In figure 11 the value of $I_{TD}$ for multiple simulations is plotted against Lake's $N_{TD}$ for that simulation. In all the simulations the numerical and physical parameters were kept constant, expect for $\alpha_t$. $I_{TD}$ shows monotonic behaviour, as should be expected. However, at $I_{TD} = 1$, $N_{TD}$ does not equal 1, but 0.1, and is thus off by a factor 10. This is similar to the factor 14 found in [10].

To determine whether a system behaves as a single-layer system, as a double-layer system without transverse dispersion, or as a system of two interacting layers, a threshold needs to be defined for $I_{TD}$. To this end, if $I_{TD} < 0.1$, the system is considered to behave as a double-layer system, and for $I_{TD} > 0.9$ the system is considered to be a single-layer system. This threshold is arbitrary, but looking at figure 11, $I_{TD}$ is most sensitive to changes in $N_{TD}$ between these thresholds, justifying their choice.

From figure 11 it then also follows that the system behaves as a double-layer system if $10N_{TD} < 0.1$, and as a single-layer system for $10N_{TD} > 5$. The latter is in agreement with [10], but $10N_{TD} < 0.1$ is slightly lower than the value 0.2 in [10].

Similarly, two sets of simulations were performed in which only the thickness $L_y$ was varying, see figure 12. These simulations agree to the conclusion as for figure 11: the

system behaves as a double-layer system if $10N_{TD} < 0.1$, and as a single-layer system for $10N_{TD} > 5$.

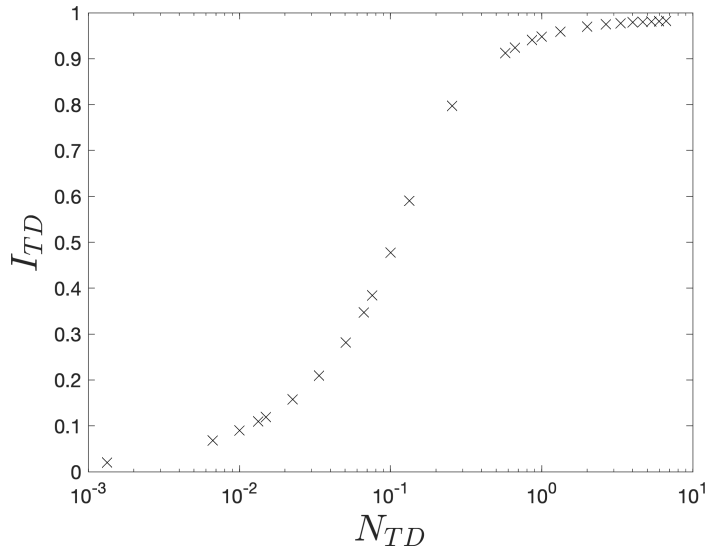



Figure 11: $I_{TD}$ plotted against Lake's $N_{TD}$ for different numerical solutions. The only varying parameter in these simulations is $\alpha_t$.

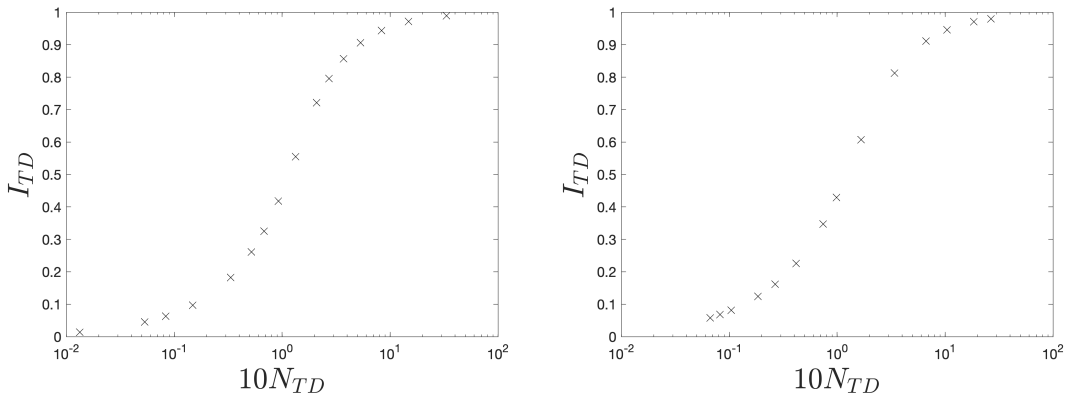



Figure 12: $I_{TD}$ plotted against Lake's $N_{TD}$ for two sets of simulations. The only varying parameter in these simulations is $L_y$.

Also, simulations were performed where the velocity $u_1$ was varied (but still $u_1 > u_2$). The $I_{TD}$ for these simulations is shown in 13, and results in a similar conclusion regarding $N_{TD}$.

Lake's $N_{TD}$ is mostly a good predictor of the behaviour of the two layer stratified porous medium, but needs to be corrected by a factor 10 in order for $I_{TD}$ to equal 0.5 if $N_{TD} = 1$.

If Lake's $N_{TD}$ is redefined as

$$N_{TD} = 10\alpha_t \frac{L_x}{L_y^2} \frac{u_2}{u_1} \tag{75}$$

then the system can be described as one single-layer if $N_{TD} > 5$ and as a double-layer system if $N_{TD} < 0.1$.

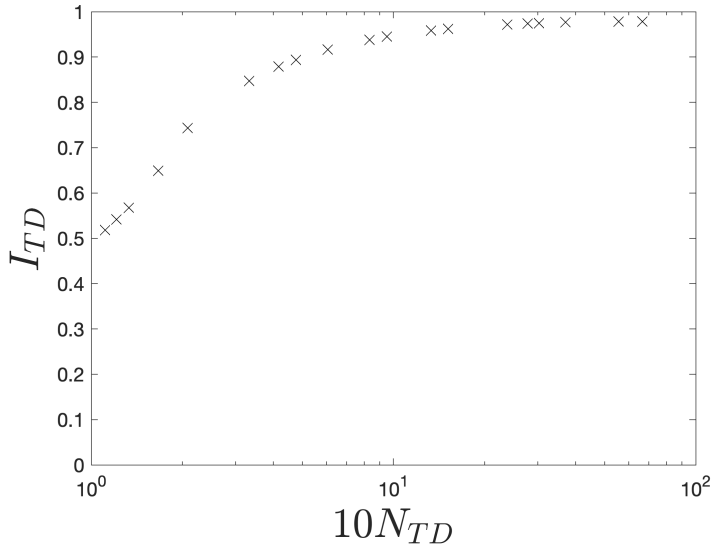



Figure 13: $I_{TD}$ plotted against Lake's $N_{TD}$ for different numerical solutions. The only varying parameter in these simulations is $u_1$.

### 5.3.1 $\tilde{N}_{TD}$

Lake's $N_{TD}$ does not take into account the layer-thickness ratio. To correct for this, $\tilde{N}_{TD}$ (equation (21)) is introduced. Figure 14 shows a set of simulations for which $\tilde{N}_{TD}$ was calculated. The only parameter that was varied was the ratio of the two layers thicknesses. As can be seen, $\tilde{N}_{TD}$ correlates well with $I_{TD}$. Similarly to the factor 10 used to restore Lake's $N_{TD}$, $\tilde{N}_{TD}$ needs to multiplied by approximately 3.5 so that $I_{TD} = 0.5$ when $\tilde{N}_{TD} = 1$. The threshold of $I_{TD} = 0.9$ is reached when $3.5\tilde{N}_{TD} \approx 1.55$, so that for larger $\tilde{N}_{TD}$ the system behaves as a single-layer system. By only varying the thickness ratio, no conclusion can be reached on for what values of $\tilde{N}_{TD}$ the system behaves as a double-layer system.

$\tilde{N}_{TD}$ was also calculated for the simulations from figures 11 and 12. For example, figure 15 shows the $I_{TD}$ plotted against $\tilde{N}_{TD}$ for the simulations where $\alpha_t$ was varying. The same factor 3.5 was necessary so that $I_{TD} = 0.5$ when $\tilde{N}_{TD} = 1$. Also in this case, the system behaves as a double-layer system if $3.5\tilde{N}_{TD} < 0.1$, and as a single-layer system for $3.5\tilde{N}_{TD} > 5$.

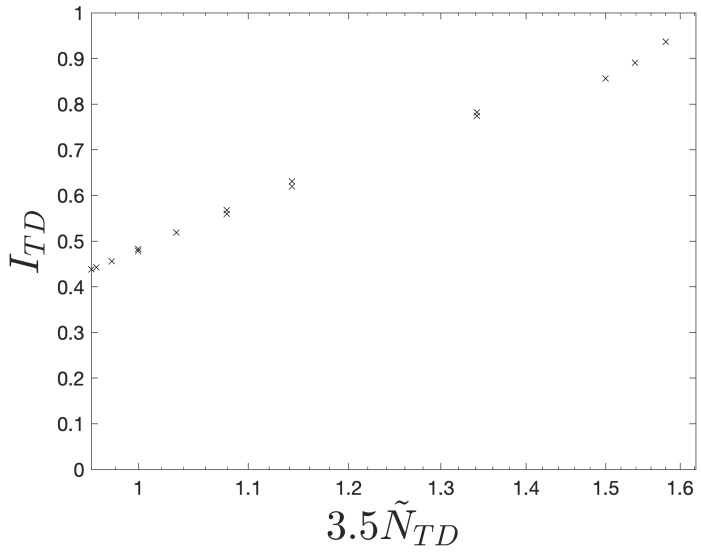



Figure 14: $I_{TD}$ plotted against $\tilde{N}_{TD}$ for different numerical solutions. The only varying parameter in these simulations is the ratio of the layer thickness ratio.

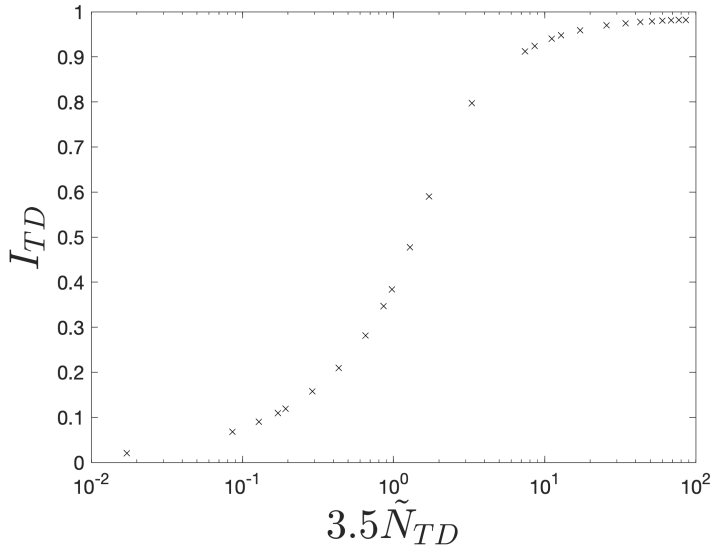



Figure 15: $I_{TD}$ plotted against $\tilde{N}_{TD}$ for different numerical solutions. The only varying parameter in these simulations is $\alpha_t$.

The transverse dispersion number $\tilde{N}_{TD}$ was in all cases a good predictor for the behaviour of the two-layer system. $\tilde{N}_{TD}$ needs to be corrected by a factor 3.5, so that it is redefined as

$$\tilde{N}_{TD} = 3.5 \frac{\alpha_t u_1 u_2 L_x}{(h_1 u_2 + h_2 u_1)(h_1 u_1 + h_2 u_2)} \tag{76}$$

The system behaves as one single layer if $\tilde{N}_{TD} > 5$ and as a two layers $\tilde{N}_{TD} < 0.1$.

### 5.4 Further work

The two-layer stratified porous medium suggests a generalization to multiple layers. Lake and Hirasaki [10] investigated how a multiple-layer porous medium can be reduced to less layers. They suggested to use a grouping procedure, where two adjacent layers are combined if their $N_{TD}$ is larger than 5 (starting with the layer pair with the highest $N_{TD}$). In this way, the behaviour of a multi-layer system can be described, and the effect of the heterogeneities. The effect of $\tilde{N}_{TD}$ in this grouping procedure needs to be investigated.

Furthermore, this grouping procedure has a greedy algorithm-like nature. It might be possible to find better grouping algorithms to reduce a multi-layer system. Verification of a multi-layer system also requires $I_{TD}$ to be generalized.

Lastly, this thesis and [10] only consider stratified porous media, where the heterogeneities only occur perpendicular to the direction of the flow. This choice was made since it simplifies Darcy's law. The effect on the (longitudinal) dispersion coefficient could als be investigated for more general heterogeneous porous media. Simulations of this then also require Darcy's law to be solved.

## 6 Conclusion

It was found that the transverse dispersion number, $N_{TD}$, was in general a good predictor of the behaviour of the stratified porous medium. The system behaved as one single layer if the transverse dispersion number (after a correction with a certain factor) was greater than 5. Similarly, the system behaved as two separate layers if the number was less than 0.1.

However, $N_{TD}$ failed if the ratio of the two layer thicknesses was varied. A new definition of the transverse dispersion number, $\tilde{N}_{TD}$, was suggested, taking the ratio of the layer thicknesses into account. Like $N_{TD}$, the system behaved as one single layer if $\tilde{N}_{TD}$ (after a correction with a certain factor) was greater than 5, and the system behaved as two separate layers if the number was less than 0.1. $\tilde{N}_{TD}$ was in all cases a good predictor of the behaviour of the stratified porous medium.

## Nomenclature

$\alpha_l$, $\alpha_{l,\text{eff}}$ (Effective) longitudinal dispersivity (m)

$\alpha_t$ Transverse dispersivity (m)

$\bar{u}$ Magnitude of the average fluid velocity field ($\mathrm{m\,s^{-1}}$)

$\beta$ Parameter in the approximation of the time-averaged flux

$\Delta t$ Time discretization step (s)

$\Delta x, \Delta y$ Spatial discretization steps (m)

$\delta_l$, $\delta_t$ Longitudinal and transverse dispersion number

$\hat{x}$, $\hat{y}$ Unit vectors

$\iota$ Imaginary number

$\mu$ Dynamic viscosity (Pa·s)

$\phi$, $\phi_{\text{eff}}$, $\phi_j$ (Effective) porosity (in layer $j$)

$\Phi_{ij}$ Flux through the boundary of cell $ij$

$\rho$ Fluid density ($\mathrm{kg\,m^{-3}}$)

$\rho$ PVI when the effluent concentration of the layer with largest fluid volume flux equals 0.5

$\sigma$ Courant number

$\theta$ Parameter in the one dimensional discretization

$\bar{\mathbf{u}}$ Average fluid velocity field ($\mathrm{m\,s^{-1}}$)

$\mathbf{D}$ Dispersion tensor

$\mathbf{g}$ Gravitational acceleration ($\mathrm{m\,s^{-2}}$)

$\mathbf{I}$ Identity matrix

$\mathbf{n}$ Outward normal

$\mathbf{q}$ Darcy velocity field ($\mathrm{m\,s^{-1}}$)

$\mathbf{u}$ Fluid velocity field ($\mathrm{m\,s^{-1}}$)

$c$ Concentration ($\mathrm{kg\,m^{-3}}$)

$c_{\text{out},j}$ Effluent concentration of layer $j$

$D$ Domain

$d$ Characteristic length (m)

$D_{\text{eff}}$ Effective diffusion coefficient ($\mathrm{m^2\,s^{-1}}$)

$D_{\text{num}}$ Numerical diffusion coefficient ($\mathrm{m^2\,s^{-1}}$)

$D_l, D_t$ Longitudinal and transversal dispersion coefficients ($\mathrm{m^2\,s^{-1}}$)

$D_m$ Molecular diffusion coefficient ($\mathrm{m^2\,s^{-1}}$)

$F$ Formation factor

$F_i^n$, $F_{ij}^n$ Time-averaged flux at time $t_n$ through the boundary of cell $i$ ($ij$)

$h_j$ Thickness of layer $j$ of the stratified porous medium (m)

$I_{TD}$ Transverse dispersion index

$k$ Porous medium permeability ($\mathrm{m^2}$)

$L_x$, $L_y$ Length and thickness of the two-layer porous medium (m)

$N_t$ Number of time points

$N_x$, $N_y$ Number of cells in the $x$- and $y$-direction

$N_{TD}$ Transverse dispersion number

$p$ Pressure ($\mathrm{N/m^2}$)

$Q$ Pore volume injected (PVI)

| | |
|---|---|
| $q$ | Exact solution of the advection-dispersion equation |
| $Q_i^n,\ Q_{ij}^n$ | (Approximate) volume average of the concentration at time $t_n$ in cell $i$ $(ij)$ |
| $T$ | Time over which the numerical method is computed (s) |
| $t$ | Time coordinate (s) |
| $t_l,\ t_t$ | Time for the solute to cross the medium longitudinally and transversely (s) |
| $u_j$ | Average fluid velocity field in layer $j$ ($\mathrm{m\,s^{-1}}$) |
| $V$ | Volume of the porous medium ($\mathrm{m^3}$) |
| $v_i,\ v_{ij}$ | Finite volume or cell |
| $V_p$ | Pore volume, volume of the pore space of the porous medium ($\mathrm{m^3}$) |
| $x,\ y$ | Spatial coordinates (m) |
| $x_i\ y_j$ | Spatial coordinates of cell $ij$ |
| $y_D$ | Dimensionless coordinate, $\frac{y}{L_y}$ |
| Pe | Peclet number |
| Re | Reynolds number |

# A Appendix

## A.1 Partial Fluxes

In the partial fluxes form section 4.2, the midpoint rule is used to approximate the integral. To discretize the partial derivatives, the central difference is applied. For the velocity, the upwind discretization is applied. The partial fluxes are then given by

$$\begin{aligned}
\Phi_{(i,j),1} &= \int_{y_{j-\frac{1}{2}}}^{y_{j+\frac{1}{2}}} \left[ (\alpha_l u(y) + D_m) \frac{\partial c}{\partial x}(x_{i+\frac{1}{2}}, y) - c(x_{i+\frac{1}{2}}, y) u(y) \right] \mathrm{d}y \\
&\approx \Delta y \left[ (\alpha_l u(y_j) + D_m) \frac{Q_{i+1,j} - Q_{i,j}}{\Delta x} - Q_{i,j} u(y_j) \right] \\
&= \underbrace{- \left[ (\alpha_l u(y_j) + D_m) \frac{\Delta y}{\Delta x} + \Delta y u(y_j) \right]}_{a_1} Q_{i,j} + \underbrace{(\alpha_l u(y_j) + D_m) \frac{\Delta y}{\Delta x}}_{a_5} Q_{i+1,j}
\end{aligned}$$

As stated in 4.2, if the velocity contains a discontinuity, then the harmonic average of the transverse dispersion coefficient above and below the discontinuity is taken,

$$D_{l,\mathrm{ave},j} = 2 \left[ \frac{(\alpha_t u(y_{j+1}) + D_m)(\alpha_t u(y_j) + D_m)}{\alpha_t (u(y_{j+1}) + u(y_j)) + 2D_m} \right] \tag{77}$$

The discretization is chosen so that any discontinuity is always on the cell interface. Note that if $u(y_{j+1}) = u(y_j)$, then $D_{l,\mathrm{ave}} = (\alpha_t u(y_j) + D_m)$. With this, $\Phi_{(i,j),2}$ is given by

$$\begin{aligned}
\Phi_{(i,j),2} &= \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \left[ (\alpha_t u(y_{j+\frac{1}{2}}) + D_m) \frac{\partial c}{\partial y}(x, y_{j+\frac{1}{2}}) \right] \mathrm{d}x \\
&\approx \Delta x \left[ 2 \frac{(\alpha_t u(y_{j+1}) + D_m)(\alpha_t u(y_j) + D_m)}{\alpha_t (u(y_{j+1}) + u(y_j)) + 2D_m} \frac{Q_{i,j+1} - Q_{i,j}}{\Delta y} \right] \\
&= \underbrace{2 \frac{\Delta x}{\Delta y} \left[ \frac{(\alpha_t u(y_{j+1}) + D_m)(\alpha_t u(y_j) + D_m)}{\alpha_t (u(y_{j+1}) + u(y_j)) + 2D_m} \right]}_{-a_2} Q_{i,j+1} \\
&\quad \underbrace{-2 \frac{\Delta x}{\Delta y} \left[ \frac{(\alpha_t u(y_{j+1}) + D_m)(\alpha_t u(y_j) + D_m)}{\alpha_t (u(y_{j+1}) + u(y_j)) + 2D_m} \right]}_{a_2} Q_{i,j}
\end{aligned}$$

Similarly, $\Phi_{(i,j),3}$ and $\Phi_{(i,j),4}$ are given by

$$\begin{aligned}
\Phi_{(i,j),3} &= -\int_{y_{j-\frac{1}{2}}}^{y_{j+\frac{1}{2}}} \left[(\alpha_l u(y) + D_m)\frac{\partial c}{\partial x}(x_{i-\frac{1}{2}}, y) - c(x_{i-\frac{1}{2}}, y)u(y)\right] \mathrm{d}y \\
&\approx -\Delta y \left[(\alpha_l u(y_j) + D_m)\frac{Q_{i,j} - Q_{i-1,j}}{\Delta x} - Q_{i-1,j}u(y_j)\right] \\
&= \underbrace{\left[(\alpha_l u(y_j) + D_m)\frac{\Delta y}{\Delta x} + \Delta y u(y_j)\right]}_{a_6} Q_{i-1,j} \underbrace{-(\alpha_l u(y_j) + D_m)\frac{\Delta y}{\Delta x}}_{a_3} Q_{i,j}
\end{aligned}$$

$$\begin{aligned}
\Phi_{(i,j),4} &= -\int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \left[(\alpha_t u(y_{j-\frac{1}{2}}) + D_m)\frac{\partial c}{\partial y}(x, y_{j-\frac{1}{2}})\right] \mathrm{d}x \\
&\approx -\Delta x \left[2\frac{(\alpha_t u(y_{j-1}) + D_m)(\alpha_t u(y_j) + D_m)}{\alpha_t(u(y_{j-1}) + u(y_j)) + 2D_m}\frac{Q_{i,j} - Q_{i,j-1}}{\Delta y}\right] \\
&= \underbrace{2\frac{\Delta x}{\Delta y}\left[\frac{(\alpha_t u(y_{j-1}) + D_m)(\alpha_t u(y_j) + D_m)}{\alpha_t(u(y_{j-1}) + u(y_j)) + 2D_m}\right]}_{-a_4} Q_{i,j-1} \\
&\quad \underbrace{-2\frac{\Delta x}{\Delta y}\left[\frac{(\alpha_t u(y_{j-1}) + D_m)(\alpha_t u(y_j) + D_m)}{\alpha_t(u(y_{j-1}) + u(y_j)) + 2D_m}\right]}_{a_4} Q_{i,j}
\end{aligned}$$

Note that if $u(y_{j-1}) = u(y_j)$, then $a_4 = -\frac{\Delta x}{\Delta y}(\alpha_t u(y_j) + D_m)$.

## A.2 Motivation of stability, consistency and local truncation error

The approximation $Q$ of a solution $q$ at time $t_n$ can be written by introducing the error $E$ such that

$$Q^n = q^n + E^n$$

If this is applied to a numerical operator $\mathcal{N}(\cdot)$, then

$$Q^{n+1} = \mathcal{N}(Q^n) = \mathcal{N}(q^n + E^n)$$

So that for the global truncation error at $t_{n+1}$

$$\begin{aligned}
E^{n+1} &= Q^{n+1} - q^{n+1} \\
&= \mathcal{N}(q^n + E^n) - q^{n+1}
\end{aligned}$$

With the local truncation error defined as

$$\tau^n = \frac{1}{\Delta t}\left[\mathcal{N}(q^n) - q^{n+1}\right],$$

$E^{n+1}$ can be rewritten as

$$E^{n+1} = \underbrace{\mathcal{N}(q^n + E^n) - \mathcal{N}(q^n)}_{\text{stability}} + \underbrace{\Delta t\ \tau^n}_{\text{consistency}}$$

In this way it can clearly be seen that the first underbraced terms determine the effect of the existing error after using the numerical method and the second underbraced term the effect of the error introduced by one time step. Stability is to give a bound on $\mathcal{N}(q^n + E^n) - \mathcal{N}(q^n)$ and consistency on the one-step error.

### A.3 Derivation of the local truncation error of numerical scheme (37)

For $\beta = 1$, inserting the exact solution in the numerical scheme (37) and expanding in Taylor series around $(x_i, y_j, t_m)$ gives

$$\begin{aligned}
q_{i,j}^{m+1} &= \alpha q_{i,j}^m + \beta q_{i+1,j}^m + \gamma q_{i,j+1}^m + \delta q_{i-1,j}^m + \epsilon q_{i,j-1}^m \\
&= \alpha q_{i,j}^m + \sum_{n=0}^{\infty} \beta \frac{\Delta x^n}{n!} \frac{\partial^n q}{\partial x^n}(x_i, y_j, t_m) + \sum_{n=0}^{\infty} \gamma \frac{\Delta y^n}{n!} \frac{\partial^n q}{\partial y^n}(x_i, y_j, t_m) \\
&\quad + \sum_{n=0}^{\infty} \delta \frac{(-\Delta x)^n}{n!} \frac{\partial^n q}{\partial x^n}(x_i, y_j, t_m) + \sum_{n=0}^{\infty} \epsilon \frac{(-\Delta y)^n}{n!} \frac{\partial^n q}{\partial y^n}(x_i, y_j, t_m) \\
&= \alpha q_{i,j}^m + \sum_{n=0}^{\infty} \left(\beta + \delta(-1)^n\right) \frac{\Delta x^n}{n!} \frac{\partial^n q}{\partial x^n}(x_i, y_j, t_m) + \left(\gamma + \epsilon(-1)^n\right) \frac{\Delta y^n}{n!} \frac{\partial^n q}{\partial y^n}(x_i, y_j, t_m)
\end{aligned}$$

Where the coefficients are given by

$$\begin{aligned}
\alpha &= 1 + \frac{\Delta t}{\Delta x \Delta y}(a_1 + a_2 + a_3 + a_4) \\
&= 1 - 2(\alpha_l u(y_j) + D_m)\frac{\Delta t}{\Delta x^2} - 2(\alpha_t u(y_j) + D_m)\frac{\Delta t}{\Delta y^2} - u(y_j)\frac{\Delta t}{\Delta x} \\
\beta &= \frac{\Delta t}{\Delta x \Delta y} a_5 = (\alpha_l u(y_j) + D_m)\frac{\Delta t}{\Delta x^2} \\
\gamma &= -\frac{\Delta t}{\Delta x \Delta y} a_2 = (\alpha_t u(y_j) + D_m)\frac{\Delta t}{\Delta y^2} \\
\delta &= \frac{\Delta t}{\Delta x \Delta y} a_6 = (\alpha_l u(y_j) + D_m)\frac{\Delta t}{\Delta x^2} + u(y_j)\frac{\Delta t}{\Delta x} \\
\epsilon &= -\frac{\Delta t}{\Delta x \Delta y} a_4 = (\alpha_t u(y_j) + D_m)\frac{\Delta t}{\Delta y^2}
\end{aligned}$$

The local truncation error $\tau$ is then given by

$$\begin{aligned}
\tau\Delta t =&\mathcal{N}(q_{i,j}^m) - q_{i,j}^{m+1} = \\
=&\alpha q_{i,j}^m + \sum_{n=0}^{\infty} \left(\beta + \delta(-1)^n\right) \frac{\Delta x^n}{n!} \frac{\partial^n q}{\partial x^n}(x_i, y_j, t_m) + \left(\gamma + \epsilon(-1)^n\right) \frac{\Delta y^n}{n!} \frac{\partial^n q}{\partial y^n}(x_i, y_j, t_m) - \\
&\sum_{n=0}^{\infty} \frac{\Delta t^n}{n!} \frac{\partial^n q}{\partial t^n}(x_i, y_j, t_m) \\
=&u(y_j) \frac{\Delta x \Delta t}{2} \frac{\partial^2 q}{\partial x^2}(x_i, y_j, t_m) - \frac{\Delta t^2}{2} \frac{\partial^2 q}{\partial t^2}(x_i, y_j, t_m) + \\
&\sum_{n=3}^{\infty} (\alpha_l u(y_j) + D_m) \Delta t \left(1 + (-1)^n\right) \frac{\Delta x^{n-2}}{n!} \frac{\partial^n q}{\partial x^n}(x_i, y_j, t_m) + u(y_j) \Delta t \frac{\Delta x^{n-1}}{n!} \frac{\partial^n q}{\partial x^n}(x_i, y_j, t_m) + \\
&(\alpha_t u(y_j) + D_m) \Delta t \left(1 + (-1)^n\right) \frac{\Delta y^{n-2}}{n!} \frac{\partial^n q}{\partial y^n}(x_i, y_j, t_m) - \frac{\Delta t^n}{n!} \frac{\partial^n q}{\partial t^n}(x_i, y_j, t_m)
\end{aligned}$$

### A.4 Derivation of the stability condition of numerical scheme (58)

The amplification factor of scheme (58) is given by

$$g(\xi, \Delta x, \Delta t) = 1 + \underbrace{(\sigma(2\theta - 1) + 2\delta_l)}_{d}(\cos(\iota \underbrace{\xi \Delta x}_{\omega}) - 1) - \iota\sigma \sin(\iota \xi \Delta x)$$

Writing $\sigma(2\theta - 1) + 2\delta_l = d$ and $\xi \Delta x = \omega$, then $|g(\xi, \Delta x, \Delta t)| \leq 1$ leads to

$$[1 + d(\cos(\omega) - 1)]^2 + \sigma^2 \sin^2(\omega) \leq 1$$

Using $\sin^2(\omega) + \cos^2(\omega) = 1$, this can be written as

$$(\cos(\omega) - 1)(d^2(\cos(\omega) - 1) + 2d - \sigma^2(\cos(\omega) + 1)) \leq 0$$

Since $\cos(\omega) - 1 \leq 0, \ \forall \omega$, this yields that

$$d^2(\cos(\omega) - 1) + 2d - \sigma^2(\cos(\omega) + 1) \geq 0$$

Rewriting this gives

$$-\sigma^2 - d^2 + 2d + (d^2 - \sigma^2)\cos(\omega) \geq 0$$

This has to hold for all $\omega$ (or $\xi$ actually). Now, $\cos(\omega)$ attains its extrema in $\omega = 0$ or $\omega = \pi$. The conditions for stability therefore become

$$\sigma^2 \leq d \text{ and } 0 \leq d \leq 1$$

And combining the gives the condition from (61).

## A.5 Taylor series for the one-dimensional local truncation error

The Taylor serries of $q(x,t)$ around $(x_i, t_n)$ is given by

$$q(x,t) = q(x_i,t_n) + (x-x_i)q_x(x_i,t_n) + (t-t_n)q_t(x_i,t_n) + \\ \frac{1}{2!}\left((x-x_i)^2 q_{xx}(x_i,t_n) + 2(x-x_i)(t-t_n)q_{xt}(x_i,t_n) + (t-t_n)^2 q_{tt}(x_i,t_n)\right) + ...$$

From this it can be easily found that $q(x_{i+1},t_n)$, $q(x_{i-1},t_n)$, $q(x_{i-2},t_n)$, and $q(x_i,t_{n+1})$ are given by

$$q_{i+1}^n = q_i^n + \Delta x q_x(x_i,t_n) + \frac{1}{2!}(\Delta x)^2 q_{xx}(x_i,t_n) + \frac{1}{3!}(\Delta x)^3 q_{xxx}(x_i,t_n) + \mathcal{O}\left(\Delta x^4\right)$$
$$q_{i-1}^n = q_i^n - \Delta x q_x(x_i,t_n) + \frac{1}{2!}(\Delta x)^2 q_{xx}(x_i,t_n) - \frac{1}{3!}(\Delta x)^3 q_{xxx}(x_i,t_n) + \mathcal{O}\left(\Delta x^4\right)$$
$$q_i^{n+1} = q_i^n + \Delta t q_t(x_i,t_n) + \frac{1}{2!}(\Delta t)^2 q_{tt}(x_i,t_n) + \mathcal{O}\left(\Delta t^3\right)$$

## A.6 MATLAB Code

In this appendix the code is shown used for the numerical method. The code is based on a class FasterNumericalSystem2D. First an example script is shown, which implements the class. The class is shown below the example script. Other scripts are based on this example script, such as the script that runs multiple simulations with only one parameter varying (used to plot $I_{TD}$ versus $N_{TD}$) or the script which implements the analytical solution.

```
%% Example of the implementation of FasterNumericalSystem.m
clear all; close all; clc;
format compact

fprintf('SetParameters.\n');
date = string(datetime);
fprintf('%s\n',date);

% Enable animation, saving and printing the figures
ParameterRelativeFilePath = './parameters';
relativeFilePath = './saved/example';
filenamePrefix = 'example_run';

% Physical parameters
Lx = 10; % m, length in x-direction
Ly = 0.01; % m, length in y-direction
T = 2.5;

layerVelocities = [5 1];
layerPorosities = [0.9 0.9];
Fh = 0.5;
layerBoundaries = [0 Fh 1]*Ly;

Dm = 1e-9;
alpha_l = 1e-2;
alpha_t = alpha_l/30;

% Numerical parameters
Nx = 50;
Ny = 200;
Nt = 2500;
Nextension = 2;

dx = Lx/Nx;
dy = Ly/Ny;
```

```
T = Lx/min(layerVelocities)*1.1;
dt = T/(Nt-1);

beta = 1; % in [0,1], 'explicivity'

% Initial condition and boundary conditions
ic = @(x,y) heaviside(y-Ly/2)+0*x;
initialConditionIsZero = true;

constantBoundaryConditions = true;
leftBC = @(y) 1+0*y; % BC at x = 0
rightBC = @(y) 0*(y); % BC at x = Lx
bottomBC = @(x) 0*(x); % BC at y = 0
topBC = @(x) 0*(x); % BC at y = Ly

N_TD = (Lx/Ly)^2*(alpha_t*min(layerVelocities)/(Lx*max(layerVelocities)));

fprintf('N_TD = %f.\n',N_TD);

if mod(Fh*Ny,1) ~= 0
    error('The velocity discontinuity is not on a cell interface.')
end

%% Saving the workspace

if ~isfolder(relativeFilePath)
    mkdir(relativeFilePath);
end

if N_TD>0.01
    filePath = sprintf('%s/%s_N_TD=%.3f',relativeFilePath,filenamePrefix,N_TD);
else
    filePath = sprintf('%s/%s_N_TD=%.2e',relativeFilePath,filenamePrefix,N_TD);
end

% Checking if the folder exists, otherwise create it.
index = 0;
while true
    if isfolder(filePath)
        index = index + 1;
        if N_TD>0.01
            filePath = sprintf('%s/%s_N_TD=%.3f_%u',relativeFilePath,filenamePrefix,N_TD,
                ↪ index);
```

```
        else
            filePath = sprintf('%s/%s_N_TD=%.2e_%u',relativeFilePath,filenamePrefix,N_TD,
                ↪ index);
        end
    else
        mkdir(filePath);
        break
    end
end
clear index

try
    save(ParameterFilename,'-v7.3')
    save(sprintf('%s/%s_two_layer_dispersion_parameters.mat',filePath,filenamePrefix),'-v7
        ↪ .3')
catch
    warning('File might not be saved correctly.')
end

try
    system = FasterNumericalSystem2D(Lx,Ly,T,Nx,Ny,Nt,Nextension);
catch
    warning('Object could not be created.');

end

try
    system.setNumericalParameters(beta);
    system.setPhysicalParameters(layerVelocities,layerBoundaries,layerPorosities,alpha_l,
        ↪ alpha_t,Dm);
catch
    warning('Physical or numerical parameters could not be set.');

end

try
    system.setInitialCondition(ic,initialConditionIsZero);
    system.setBoundaryConditions(leftBC,1,rightBC,2,bottomBC,2,topBC,2,
        ↪ constantBoundaryConditions);
catch
    warning('Initial or boundary conditions could not be set.');

end
```

```
try
    system.checkStability;
catch
    warning('Stability check failed.');
end

try
    system.setLinearSytem;
catch
    warning('System could not be set.');

end

try
    elapsedTime = system.solveLinearSytem;
catch
    warning('System could not be solved.');

end

%% Calculating the breakthrough curve
try
    [N_TD,I_TD,tm] = system.breakthrough(true,true,true,filePath,filenamePrefix);
catch
    warning('Breakthrough curve failed.')
end

try
    filename = sprintf('%s/%s_solution.mat',filePath,filenamePrefix);
    save(filename,'-v7.3')
catch
    warning('File might not be saved correctly.')
end

%% Visualizing the solution.
try
    system.plotSolution(T/8,45,45,true,filePath,filenamePrefix);

    system.contourPlotSolution(T/4 ,true,filePath,filenamePrefix);
catch
    warning('Plots failed.')
end

```

```
try
    system.animateSolution([5 45],[45 45],true,filePath,filenamePrefix);
    system.animateContourPlot(true,filePath,filenamePrefix);
catch
    warning('Animation Failed.')
end
```

### Class FasterNumericalSystem2D

```
classdef FasterNumericalSystem2D < handle

    properties
        Lx
        Lxout
        Ly
        T
        Nx
        Nout
        Ny
        Nt
        dx
        dy
        dt

        x
        y
        t

        Qfull
        Qlfull
        Qrfull
        Qbfull
        Qtfull

        A
        B
        a
        b

        xStep
        yStep
        tStep
```

```
        Nxmax
        Nymax
        Ntmax
        xIndex
        yIndex
        tIndex
        storeIndex

        Q
        Ql
        Qr
        Qb
        Qt
        outflowLayer1
        outflowLayer2

        leftBoundaryType = 0;
        rightBoundaryType = 0;
        bottomBoundaryType = 0;
        topBoundaryType = 0;

        layerVelocities
        layerBoundaries
        layerPorosity

        alpha_l = 0;
        alpha_t = 0;
        Dm = 0;

        beta = 1;

        boundariesAreSet = false;
        initialConditionIsSet = false;
        physicalParametersAreSet = false;
        numericalParametersAreSet = false;
        linearSystemIsSet = false;
        solutionIsReady = false;
    end

    methods
        function obj = FasterNumericalSystem2D(Lx,Ly,T,Nx,Ny,Nt,Nextension)

            obj.Lx = Nextension*Lx;
```

```
77              obj.Ly = Ly;
78              obj.T = T;
79              obj.Nx = Nextension*Nx;
80              obj.Nout = Nx;
81              obj.Lxout = Lx;
82              obj.Ny = Ny;
83              obj.Nt = Nt;
84              obj.dx = Lx/Nx;
85              obj.dy = Ly/Ny;
86              obj.dt = T/(Nt-1);
87
88              obj.x = linspace(obj.dx/2,obj.Lx-obj.dx/2,obj.Nx);
89              obj.y = linspace(obj.dy/2,obj.Ly-obj.dy/2,obj.Ny);
90              obj.t = linspace(0,obj.T,obj.Nt);
91
92              obj.Qfull = sparse(obj.Nx*obj.Ny,1);
93              obj.Qlfull = sparse(obj.Ny,1);
94              obj.Qrfull = sparse(obj.Ny,1);
95              obj.Qbfull = sparse(obj.Nx,1);
96              obj.Qtfull = sparse(obj.Nx,1);
97
98              obj.Nxmax = 200;
99              obj.Nymax = 200;
100             obj.Ntmax = 1000;
101
102             if obj.Nx > obj.Nxmax
103                 obj.xStep = floor(obj.Nx/obj.Nxmax);
104                 obj.xIndex = 1:obj.xStep:obj.Nx;
105                 if obj.xIndex(end) ~= obj.Nx
106                     obj.xIndex = [obj.xIndex obj.Nx];
107                 end
108                 obj.Nxmax = length(obj.xIndex);
109             else
110                 obj.xStep = 1;
111                 obj.xIndex = 1:obj.Nx;
112                 obj.Nxmax = obj.Nx;
113             end
114             if obj.Ny > obj.Nymax
115                 obj.yStep = floor(obj.Ny/obj.Nymax);
116                 obj.yIndex = 1:obj.yStep:obj.Ny;
117                 if obj.yIndex(end) ~= obj.Ny
118                     obj.yIndex = [obj.yIndex obj.Ny];
119                 end
```

```
120             obj.Nymax = length(obj.yIndex);
121         else
122             obj.yStep = 1;
123             obj.yIndex = 1:obj.Ny;
124             obj.Nymax = obj.Ny;
125         end
126         if obj.Nt > obj.Ntmax
127             obj.tStep = floor(obj.Nt/obj.Ntmax);
128             obj.tIndex = 1:obj.tStep:obj.Nt;
129             if obj.tIndex(end) ~= obj.Nt
130                 obj.tIndex = [obj.tIndex obj.Nt];
131             end
132             obj.Ntmax = length(obj.tIndex);
133         else
134             obj.tStep = 1;
135             obj.tIndex = 1:obj.Nt;
136             obj.Ntmax = obj.Nt;
137         end
138 
139         for ii = obj.xIndex
140             obj.storeIndex = [obj.storeIndex (ii-1)*obj.Ny+obj.yIndex];
141         end
142 
143         obj.Q = zeros(obj.Nxmax*obj.Nymax,obj.Ntmax);
144         obj.Ql = zeros(obj.Nymax,obj.Ntmax);
145         obj.Qr = zeros(obj.Nymax,obj.Ntmax);
146         obj.Qb = zeros(obj.Nxmax,obj.Ntmax);
147         obj.Qt = zeros(obj.Nxmax,obj.Ntmax);
148 
149     end
150 
151     function setInitialCondition(obj,ic,isZero)
152         fprintf('Setting initial condition... ');
153 
154         if isZero
155             obj.Qfull = sparse(obj.Nx*obj.Ny,1);
156         else
157             for i = 1:obj.Nx
158                 for j = 1:obj.Ny
159                     obj.Qfull((i-1)*obj.Ny+j,1) = integral2(ic,(i-1)*obj.dx,i*obj.dx,(j
                            ↪ -1)*obj.dy,j*obj.dy)/(obj.dx*obj.dy);
160                 end
161             end
```

```
162             end
163
164
165             obj.initialConditionIsSet = true;
166             fprintf('Initial condition correctly set.\n');
167         end
168
169         function setBoundaryConditions(obj,leftBoundaryCondition,leftBoundaryType,
                ↪ rightBoundaryCondition,rightBoundaryType,bottomBoundaryCondition,
                ↪ bottomBoundaryType,topBoundaryCondition,topBoundaryType,areConstant)
170             % Currently only handles stationary boundary conditions.
171             fprintf('Setting boundary conditions... ');
172
173             obj.leftBoundaryType = leftBoundaryType;
174             obj.rightBoundaryType = rightBoundaryType;
175             obj.bottomBoundaryType = bottomBoundaryType;
176             obj.topBoundaryType = topBoundaryType;
177
178             if leftBoundaryType == 1 % Dirichlet boundary condition.
179                 if areConstant
180                     obj.Qlfull(:) = leftBoundaryCondition(0);
181                     obj.Ql(:,:) = leftBoundaryCondition(0);
182                 else
183                     for i = 1:obj.Ny
184                         obj.Qlfull(i) = integral(leftBoundaryCondition,(i-1)*obj.dy,i*obj.dy
                            ↪ )/(obj.dy);
185                     end
186                     for i = 1:obj.Ntmax
187                         obj.Ql(:,i) = obj.Qlfull(obj.yIndex);
188                     end
189                 end
190             elseif leftBoundaryType == 2 % Von Neumann boundary condition.
191
192             else
193                 fprintf('Wrong left boundary type!\n');
194                 return
195             end
196
197             if rightBoundaryType == 1
198                 if areConstant
199                     obj.Qrfull(:) = rightBoundaryCondition(0);
200                     obj.Qr(:,:) = rightBoundaryCondition(0);
201                 else
```

```
202             for i = 1:obj.Ny
203                 obj.Qrfull(i) = integral(rightBoundaryCondition,(i-1)*obj.dy,i*obj.
                        ↪ dy)/(obj.dy);
204             end
205             for i = 1:obj.Ntmax
206                 obj.Qr(:,i) = obj.Qrfull(obj.yIndex);
207             end
208         end
209     elseif rightBoundaryType == 2
210
211     else
212         fprintf('Wrong right boundary type!\n');
213         return
214     end
215
216     if bottomBoundaryType == 1
217         if areConstant
218             obj.Qbfull(:) = bottomBoundaryCondition(0);
219             obj.Qb(:,:) = bottomBoundaryCondition(0);
220         else
221             for i = 1:obj.Nx
222                 obj.Qbfull(i) = integral(bottomBoundaryCondition,(i-1)*obj.dx,i*obj.
                        ↪ dx)/(obj.dx);
223             end
224             for i = 1:obj.Ntmax
225                 obj.Qb(:,i) = obj.Qbfull(obj.xIndex);
226             end
227         end
228     elseif bottomBoundaryType == 2
229
230     else
231         fprintf('Wrong bottom boundary type!\n');
232         return
233     end
234
235     if topBoundaryType == 1
236         if areConstant
237             obj.Qtfull(:) = topBoundaryCondition(0);
238             obj.Qt(:,:) = topBoundaryCondition(0);
239         else
240             for i = 1:obj.Nx
241                 obj.Qtfull(i) = integral(topBoundaryCondition,(i-1)*obj.dx,i*obj.dx)
                        ↪ /(obj.dx);
```

```
242                 end
243                 for i = 1:obj.Ntmax
244                     obj.Qt(:,i) = obj.Qtfull(obj.xIndex);
245                 end
246             end
247         elseif topBoundaryType == 2
248 
249         else
250             fprintf('Wrong top boundary type!\n');
251             return
252         end
253 
254         obj.boundariesAreSet = true;
255         fprintf('Boundaries correctly set.\n');
256     end
257 
258     function setPhysicalParameters(obj,layerVelocities,layerBoundaries,layerPorosity,
        ↪ alpha_l,alpha_t,Dm)
259         fprintf('Setting physical parameters... ');
260 
261         obj.layerVelocities = layerVelocities;
262         obj.layerBoundaries = layerBoundaries;
263         obj.layerPorosity = layerPorosity;
264 
265         obj.alpha_l = alpha_l;
266         obj.alpha_t = alpha_t;
267         obj.Dm = Dm;
268 
269         obj.physicalParametersAreSet = true;
270         fprintf('Physical parameters correctly set.\n');
271     end
272 
273     function setNumericalParameters(obj,beta)
274         fprintf('Setting numerical parameters... ');
275         obj.beta = beta;
276 
277         obj.numericalParametersAreSet = true;
278         fprintf('Numerical parameters correctly set.\n');
279     end
280 
281     function setLinearSytem(obj)
282         fprintf('Setting system... ');
283 
```

```
284             if ~obj.boundariesAreSet
285                 fprintf('Error in setLinearSytem: Boundaries are not set!\n');
286                 return
287             end
288             if ~obj.physicalParametersAreSet
289                 fprintf('Warning in setLinearSytem: Physical parameters are not set!\n');
290             end
291             if ~obj.numericalParametersAreSet
292                 fprintf('Warning in setLinearSytem: Numerical parameters are not set!\n');
293             end
294
295             obj.A = sparse(obj.Nx*obj.Ny,obj.Nx*obj.Ny);
296             obj.B = sparse(obj.Nx*obj.Ny,obj.Nx*obj.Ny);
297             obj.a = sparse(obj.Nx*obj.Ny,1);
298             obj.b = sparse(obj.Nx*obj.Ny,1);
299
300             f = waitbar(0,sprintf('Iteration 1 out of %i\nEstimated time to completion: --
                    ↪ min -- s',obj.Nx),'Name','Setting the linear system...');
301             iterationTime = zeros(obj.Nx-1,1);
302
303             for ii = 1:obj.Nx
304                 tic;
305                 for jj = 1:obj.Ny
306                     k = (ii-1)*obj.Ny+jj;
307
308                     y_cell = obj.y(jj);
309                     %x_cell = 0; % Used for the velocity calculation, which is constant in x
                            ↪ .
310                     %t_cell = 0; % Used for the velocity calculation, which is constant in t
                            ↪ .
311
312                     % Calculate the transeverse dispersion coefficient in
313                     % the cell centers
314                     Dt_below = obj.alpha_t*obj.ux(y_cell-obj.dy)+obj.Dm;
315                     Dt = obj.alpha_t*obj.ux(y_cell )+obj.Dm;
316                     Dt_above = obj.alpha_t*obj.ux(y_cell+obj.dy)+obj.Dm;
317
318                     % Calculate the dispersion coefficient on the
319                     % interfaces.
320                     Dtb = 2*Dt*Dt_below/(Dt+Dt_below); % transverse bottom
321                     Dtt = 2*Dt*Dt_above/(Dt+Dt_above); % transverse top
322                     Dl = obj.alpha_l*obj.ux(y_cell)+obj.Dm; % longitudinal
323
```

```
324
325             a1 = -(Dl)*obj.dy/obj.dx - obj.dy*obj.ux(y_cell);
326             a2 = -(Dtt)*obj.dx/obj.dy;
327             a3 = -(Dl)*obj.dy/obj.dx;
328             a4 = -(Dtb)*obj.dx/obj.dy;
329             a5 = (Dl)*obj.dy/obj.dx;
330             a6 = (Dl)*obj.dy/obj.dx + obj.dy*obj.ux(y_cell);
331
332             obj.A(k,k) = obj.A(k,k) + 1-obj.dt/(obj.dx*obj.dy)*(1-obj.beta)*(a1+a2+
                    ↪ a3+a4);
333             obj.B(k,k) = obj.B(k,k) + 1+obj.dt/(obj.dx*obj.dy)*obj.beta *(a1+a2+a3+
                    ↪ a4);
334
335
336             % Check if cell is on the right boundary
337             if ii == obj.Nx
338                 if obj.rightBoundaryType == 1
339                     obj.A(k,k) = obj.A(k,k) + -obj.dt/(obj.dx*obj.dy)*(1-obj.beta)
                            ↪ *(-a5);
340                     obj.B(k,k) = obj.B(k,k) + obj.dt/(obj.dx*obj.dy)*obj.beta *(-a5)
                            ↪ ;
341                     obj.a(k) = obj.a(k) + -obj.dt/(obj.dx*obj.dy)*(1-obj.beta)*(a5)
                            ↪ *2*obj.Qrfull(jj);
342                     obj.b(k) = obj.b(k) + obj.dt/(obj.dx*obj.dy)*obj.beta *(a5)*2*
                            ↪ obj.Qrfull(jj);
343                 elseif obj.rightBoundaryType == 2
344                     obj.A(k,k) = obj.A(k,k) + -obj.dt/(obj.dx*obj.dy)*(1-obj.beta)*(
                            ↪ a5);
345                     obj.B(k,k) = obj.B(k,k) + obj.dt/(obj.dx*obj.dy)*obj.beta *(a5);
346                 end
347             else
348                 obj.A(k,k+obj.Ny) = obj.A(k,k+obj.Ny) + -obj.dt/(obj.dx*obj.dy)*(1-
                        ↪ obj.beta)*(a5);
349                 obj.B(k,k+obj.Ny) = obj.B(k,k+obj.Ny) + obj.dt/(obj.dx*obj.dy)*obj.
                        ↪ beta *(a5);
350             end
351
352             % Check if cell is on the top boundary
353             if jj == obj.Ny
354                 if obj.topBoundaryType == 1
355                     obj.A(k,k) = obj.A(k,k) + -obj.dt/(obj.dx*obj.dy)*(1-obj.beta)*(
                            ↪ a2);
356                     obj.B(k,k) = obj.B(k,k) + obj.dt/(obj.dx*obj.dy)*obj.beta *(a2);
```

```
357                 obj.a(k) = obj.a(k) + -obj.dt/(obj.dx*obj.dy)*(1-obj.beta)*(-a2)
                        ↪ *2*obj.Qtfull(ii);
358                 obj.b(k) = obj.b(k) + obj.dt/(obj.dx*obj.dy)*obj.beta *(-a2)*2*
                        ↪ obj.Qtfull(ii);
359             elseif obj.topBoundaryType == 2
360                 obj.A(k,k) = obj.A(k,k) + -obj.dt/(obj.dx*obj.dy)*(1-obj.beta)
                        ↪ *(-a2);
361                 obj.B(k,k) = obj.B(k,k) + obj.dt/(obj.dx*obj.dy)*obj.beta *(-a2)
                        ↪ ;
362             end
363         else
364             obj.A(k,k+1) = obj.A(k,k+1) + -obj.dt/(obj.dx*obj.dy)*(1-obj.beta)
                    ↪ *(-a2);
365             obj.B(k,k+1) = obj.B(k,k+1) + obj.dt/(obj.dx*obj.dy)*obj.beta *(-a2)
                    ↪ ;
366         end
367
368         % Check if cell is on the left boundary
369         if ii == 1
370             if obj.leftBoundaryType == 1
371                 obj.A(k,k) = obj.A(k,k) + -obj.dt/(obj.dx*obj.dy)*(1-obj.beta)
                        ↪ *(-a6);
372                 obj.B(k,k) = obj.B(k,k) + obj.dt/(obj.dx*obj.dy)*obj.beta *(-a6)
                        ↪ ;
373                 obj.a(k) = obj.a(k) + -obj.dt/(obj.dx*obj.dy)*(1-obj.beta)*(a6)
                        ↪ *2*obj.Qlfull(jj);
374                 obj.b(k) = obj.b(k) + obj.dt/(obj.dx*obj.dy)*obj.beta *(a6)*2*
                        ↪ obj.Qlfull(jj);
375             elseif obj.leftBoundaryType == 2
376                 obj.A(k,k) = obj.A(k,k) + -obj.dt/(obj.dx*obj.dy)*(1-obj.beta)*(
                        ↪ a6);
377                 obj.B(k,k) = obj.B(k,k) + obj.dt/(obj.dx*obj.dy)*obj.beta *(a6);
378             end
379         else
380             obj.A(k,k-obj.Ny) = obj.A(k,k-obj.Ny) + -obj.dt/(obj.dx*obj.dy)*(1-
                    ↪ obj.beta)*(a6);
381             obj.B(k,k-obj.Ny) = obj.B(k,k-obj.Ny) + obj.dt/(obj.dx*obj.dy)*obj.
                    ↪ beta *(a6);
382         end
383
384         % Check if cell is on the bottom boundary
385         if jj == 1
386             if obj.bottomBoundaryType == 1
```

```
387                     obj.A(k,k) = obj.A(k,k) + -obj.dt/(obj.dx*obj.dy)*(1-obj.beta)*(
                            ↪ a4);
388                     obj.B(k,k) = obj.B(k,k) + obj.dt/(obj.dx*obj.dy)*obj.beta *(a4);
389                     obj.a(k) = obj.a(k) + -obj.dt/(obj.dx*obj.dy)*(1-obj.beta)*(-a4)
                            ↪ *2*obj.Qbfull(ii);
390                     obj.b(k) = obj.b(k) + obj.dt/(obj.dx*obj.dy)*obj.beta *(-a4)*2*
                            ↪ obj.Qbfull(ii);
391                 elseif obj.bottomBoundaryType == 2
392                     obj.A(k,k) = obj.A(k,k) + -obj.dt/(obj.dx*obj.dy)*(1-obj.beta)
                            ↪ *(-a4);
393                     obj.B(k,k) = obj.B(k,k) + obj.dt/(obj.dx*obj.dy)*obj.beta *(-a4)
                            ↪ ;
394                 end
395             else
396                 obj.A(k,k-1) = obj.A(k,k-1) + -obj.dt/(obj.dx*obj.dy)*(1-obj.beta)
                        ↪ *(-a4);
397                 obj.B(k,k-1) = obj.B(k,k-1) + obj.dt/(obj.dx*obj.dy)*obj.beta *(-a4)
                        ↪ ;
398             end
399
400
401
402         end
403
404         iterationTime(ii) = toc;
405
406         % Estimate remaining time
407         estimatedTime = mean(iterationTime(1:ii))*(obj.Nx-1-ii)*1.4;
408
409         % Update waitbar every 20 iterations (every iteration slows down the
                ↪ process a lot).
410         if mod(ii,50) == 0
411             waitbar(ii/(obj.Nx-1),f,sprintf('Iteration %i out of %i\nEstimated time
                    ↪ to completion: %i min %3.2f s',ii,obj.Nx,floor(estimatedTime/60),
                    ↪ mod(estimatedTime,60)));
412         end
413     end
414
415     waitbar(1,f,sprintf('Iteration %i out of %i\nEstimated time to completion: %i
            ↪ min %3.2f s',obj.Nx,obj.Nx,0,0));
416     delete(f)
417
418     obj.linearSystemIsSet = true;
```

```
419             fprintf('System correctly set.\n');
420         end
421
422         function elapsedTime = solveLinearSytem(obj)
423             fprintf('Solving system... ');
424             readyToSolve = true;
425
426             if ~obj.boundariesAreSet && ~obj.initialConditionIsSet
427                 fprintf('Error in solveLinearSytem: Boundaries and inital condition are not
                        ↪  set!\n');
428                 readyToSolve = false;
429             elseif ~obj.boundariesAreSet
430                 fprintf('Error in solveLinearSytem: Boundaries are not set!\n');
431                 readyToSolve = false;
432             elseif ~obj.initialConditionIsSet
433                 fprintf('Error in solveLinearSytem: Initial condition is not set!\n');
434                 readyToSolve = false;
435             end
436             if ~obj.physicalParametersAreSet
437                 fprintf('Warning in solveLinearSytem: Default physical parameters are used
                        ↪ !\n');
438             end
439             if ~obj.numericalParametersAreSet
440                 fprintf('Warning in solveLinearSytem: Default numerical parameters are used
                        ↪ !\n');
441             end
442             if ~obj.linearSystemIsSet
443                 fprintf('Error in solveLinearSytem: Linear system is not set!\n');
444                 readyToSolve = false;
445             end
446
447             if ~readyToSolve
448                 fprintf('Error in solveLinearSytem: System could not be solved.\n');
449                 return
450             end
451
452             Nboundary = obj.layerBoundaries(2)/obj.dy;
453
454             obj.outflowLayer1 = zeros(1,obj.Nt);
455             obj.outflowLayer2 = zeros(1,obj.Nt);
456             % A waitbar to show the progress and estimate time to
457             % completion.
```

```
458             f = waitbar(0,sprintf('Iteration 1 out of %i\nEstimated time to completion: --
                    ↪ min -- s',obj.Nt),'Name','Solving the linear system...');
459             iterationTime = zeros(obj.Nt-1,1);
460             % Record the time to solve.
461             timerValue = tic;
462
463             % Actually solving of the linear system.
464             for i=1:obj.Nt-1
465
466                 tic;
467                 % Store Qfull in Q
468                 if ~isempty(find(obj.tIndex == i,1))
469                     j = find(obj.tIndex == i,1);
470                     obj.Q(:,j) = obj.Qfull(obj.storeIndex);
471                 end
472
473                 obj.outflowLayer1(i) = mean(0.5*(obj.Qfull(((obj.Nout-1)*obj.Ny+1):((obj.
                        ↪ Nout-1)*obj.Ny+Nboundary))+obj.Qfull((obj.Nout*obj.Ny+1):(obj.Nout*
                        ↪ obj.Ny+Nboundary))));
474                 obj.outflowLayer2(i) = mean(0.5*(obj.Qfull(((obj.Nout-1)*obj.Ny+1+Nboundary
                        ↪ ):(obj.Nout*obj.Ny)) +obj.Qfull((obj.Nout*obj.Ny+1+Nboundary):((obj.
                        ↪ Nout+1)*obj.Ny))));
475
476                 % Calculating the solution at the next time step.
477                 if obj.beta == 1
478                     obj.Qfull = (obj.B*obj.Qfull+obj.b);
479                 else
480                     obj.Qfull = obj.A\(obj.B*obj.Qfull+obj.b-obj.a);
481                 end
482
483                 iterationTime(i) = toc;
484
485                 % Estimate remaining time
486                 estimatedTime = mean(iterationTime(1:i))*(obj.Nt-1-i)*1.4;
487
488                 % Update waitbar every 20 iterations (every iteration slows down the
                        ↪ process a lot).
489                 if mod(i,20) == 0
490                     waitbar(i/(obj.Nt-1),f,sprintf('Iteration %i out of %i\nEstimated time
                            ↪ to completion: %i min %3.2f s',i,obj.Nt,floor(estimatedTime/60),
                            ↪ mod(estimatedTime,60)));
491                 end
492             end
```

```
493             obj.Q(:,end) = obj.Qfull(obj.storeIndex);
494             obj.outflowLayer1(end) = mean(0.5*(obj.Qfull(((obj.Nout-1)*obj.Ny+1):((obj.Nout
                    ↪ -1)*obj.Ny+Nboundary))+obj.Qfull((obj.Nout*obj.Ny+1):(obj.Nout*obj.Ny+
                    ↪ Nboundary))));
495             obj.outflowLayer2(end) = mean(0.5*(obj.Qfull(((obj.Nout-1)*obj.Ny+1+Nboundary)
                    ↪ :(obj.Nout*obj.Ny)) +obj.Qfull((obj.Nout*obj.Ny+1+Nboundary):((obj.Nout
                    ↪ +1)*obj.Ny))));
496
497
498
499             elapsedTime = toc(timerValue);
500
501             waitbar(1,f,sprintf('Iteration %i out of %i\nEstimated time to completion: %i
                    ↪ min %3.2f s',obj.Nt,obj.Nt,0,0));
502             delete(f)
503
504             % Set the values of the ghosts cells if they are of the second
505             % type.
506             if obj.leftBoundaryType == 2
507                 obj.Ql = obj.Q(1:obj.Nymax,:);
508             end
509             if obj.rightBoundaryType == 2
510                 obj.Qr = obj.Q(((obj.Nxmax-1)*obj.Nymax+1):(obj.Nxmax*obj.Nymax),:);
511             end
512             if obj.bottomBoundaryType == 2
513                 obj.Qb = obj.Q(1:obj.Nymax:((obj.Nxmax-1)*obj.Nymax+1),:);
514             end
515             if obj.topBoundaryType == 2
516                 obj.Qt = obj.Q(obj.Nymax:obj.Nymax:obj.Nxmax*obj.Nymax,:);
517             end
518
519             obj.solutionIsReady = true;
520             fprintf('System succesfully solved.\n');
521         end
522
523         %% Stability analysis
524
525         function [nu, di_x] = checkStability(obj)
526             if ~obj.physicalParametersAreSet
527                 fprintf('Error in checkStability: Physical parameters are not set!\n');
528                 return
529             end
530             if ~obj.numericalParametersAreSet
```

```
531             fprintf('Error in checkStability: Numerical parameters are not set!\n');
532             return
533         end
534 
535         nu = max(obj.layerVelocities)*obj.dt/obj.dx; % Courant number
536         di_x = (obj.alpha_l*max(obj.layerVelocities)+obj.Dm)*obj.dt/obj.dx^2; %
                ↪ Diffusion number
537         di_y = (obj.alpha_t*max(obj.layerVelocities)+obj.Dm)*obj.dt/obj.dy^2; %
                ↪ Diffusion number
538 
539         if 2*di_x+nu > 1
540             fprintf('The scheme might be unstable: 2*di_x+nu = %2.3f > 1.\n',2*di_x+nu)
                    ↪ ;
541         end
542         if 2*di_x+nu < nu^2
543             fprintf('The scheme might be unstable: 2*di_x+nu = %2.3f < %2.3f = nu^2.\n'
                    ↪ ,2*di_x+nu,nu^2);
544         end
545         if 2*di_y > 1
546             fprintf('The scheme might be unstable: 2*di_y = %2.3f > 1.\n',2*di_y);
547         end
548     end
549 
550     %% Plotting
551 
552     function [Qgrid, Xgrid, Ygrid] = indexVectorToGrid(obj)
553         Qgrid = zeros(obj.Nxmax+2,obj.Nymax+2,obj.Ntmax);
554 
555         for i = 1:obj.Nxmax
556             Qgrid(i+1,2:end-1,:) = obj.Q((1:obj.Nymax)+(i-1)*obj.Nymax,:);
557         end
558 
559         Qgrid(1,2:(end-1),:) = obj.Ql;
560         Qgrid(end,2:(end-1),:) = obj.Qr;
561         Qgrid(2:(end-1),1,:) = obj.Qb;
562         Qgrid(2:(end-1),end,:) = obj.Qt;
563         Qgrid(1,1,:) = 0.5*(obj.Ql(1,:) +obj.Qb(1,:));
564         Qgrid(1,end,:) = 0.5*(obj.Ql(end,:)+obj.Qt(1,:));
565         Qgrid(end,1,:) = 0.5*(obj.Qr(1,:) +obj.Qb(end,:));
566         Qgrid(end,end,:) = 0.5*(obj.Qr(end,:)+obj.Qt(end,:));
567 
568         Qgrid = permute(Qgrid(:,:,:),[2 1 3]);
569         xPlot = [0 obj.x(obj.xIndex) obj.Lx];
```

```
570             yPlot = [0 obj.y(obj.yIndex) obj.Ly];
571             [Xgrid,Ygrid] = meshgrid(xPlot,yPlot);
572         end
573 
574         function animateSolution(obj,azimuth,elevation,saveToFile,filePath,filenamePrefix)
575             if ~obj.solutionIsReady
576                 fprintf('Error in animateSolution: System is not solved.\n');
577                 return
578             end
579             fprintf('Animation running...')
580 
581             f = figure('Name','Animation','Color','white');
582             ax = axes;
583             [Qgrid,Xgrid,Ygrid] = obj.indexVectorToGrid;
584             s = surf(Xgrid,Ygrid,Qgrid(:,:,1),'EdgeColor','none');
585             Qmax = max(Qgrid,[],'all');
586             axis(ax,[0,obj.Lxout,0,obj.Ly,0,Qmax*1.05]);
587             colorbar
588 
589             view(ax,azimuth(1),elevation(1));
590 
591             title('Animation of numerical solution, $t$ = 0.00 s','Interpreter','Latex');
592             xlabel('Distance $x$ (m)','Interpreter','Latex');
593             ylabel('Distance $y$ (m)','Interpreter','Latex');
594             zlabel('$c(x,y,0)$','Interpreter','Latex');
595 
596             if saveToFile
597                 v = VideoWriter(sprintf('%s/%s_movie.mp4',filePath,filenamePrefix),'MPEG-4'
                        ↪ );
598                 open(v);
599 
600                 for i = 1:obj.Ntmax
601                     s.ZData = Qgrid(:,:,i);
602                     title(ax,sprintf('Animation of numerical solution, $t$ = %.2f s',(i-1)*
                            ↪ obj.dt*obj.tStep),'Interpreter','Latex');
603                     zlabel(ax,['$c(x,y,' num2str((i-1)*obj.dt*obj.tStep) ')$'],'Interpreter'
                            ↪ ,'Latex');
604                     view(ax,(azimuth(2)-azimuth(1))/(obj.Ntmax-1)*(i-1)+azimuth(1),(
                            ↪ elevation(2)-elevation(1))/(obj.Ntmax-1)*(i-1)+elevation(1));
605                     frame = getframe(f);
606                     writeVideo(v,frame);
607                 end
```

```
608             title(ax,sprintf('Animation of numerical solution, $t$ = %.2f s',(i-1)*obj.
                    ↪ dt*obj.tStep),'Interpreter','Latex');
609             zlabel(ax,['$c(x,y,' num2str(obj.T) ')$'],'Interpreter','Latex');
610             view(ax,azimuth(2),elevation(2));
611             frame = getframe(f);
612             writeVideo(v,frame);
613             close(v);
614         else
615             for i = 1:obj.Ntmax
616                 s.ZData = Qgrid(:,:,i);
617                 title(ax,sprintf('Animation of numerical solution, $t$ = %.2f s',(i-1)*
                        ↪ obj.dt*obj.tStep),'Interpreter','Latex');
618                 zlabel(ax,['$c(x,y,' num2str((i-1)*obj.dt*obj.tStep) ')$'],'Interpreter'
                        ↪ ,'Latex');
619                 view(ax,(azimuth(2)-azimuth(1))/(obj.Ntmax-1)*(i-1)+azimuth(1),(
                        ↪ elevation(2)-elevation(1))/(obj.Ntmax-1)*(i-1)+elevation(1));
620                 pause(obj.dt*obj.tStep);
621             end
622         end
623         fprintf('Ready.\n')
624     end
625
626     function animateContourPlot(obj,saveToFile,filePath,filenamePrefix)
627         if ~obj.solutionIsReady
628             fprintf('Error in animateSolution: System is not solved.\n');
629             return
630         end
631         fprintf('Animation running...')
632
633         f = figure('Name','Animation','Color','white');
634         ax = axes;
635         [Qgrid,Xgrid,Ygrid] = obj.indexVectorToGrid;
636         [~,s] = contour(Xgrid,Ygrid,Qgrid(:,:,1));
637         hold(ax,'on');
638         plot([0 obj.Lx],[obj.layerBoundaries(2) obj.layerBoundaries(2)],'k')
639         axis(ax,[0,obj.Lxout,0,obj.Ly]);
640
641         title('Animation of contour plot, $t$ = 0.00 s','Interpreter','Latex');
642         xlabel('Distance $x$ (m)','Interpreter','Latex');
643         ylabel('Distance $y$ (m)','Interpreter','Latex');
644
645         if saveToFile
```

```
646             v = VideoWriter(sprintf('%s/%s_contour_movie.mp4',filePath,filenamePrefix),
                    ↪ 'MPEG-4');
647             open(v);
648 
649             for i = 1:obj.Ntmax
650                 s.ZData = Qgrid(:,:,i);
651                 title(ax,sprintf('Animation of contour plot, $t$ = %.2f s',(i-1)*obj.dt*
                        ↪ obj.tStep),'Interpreter','Latex');
652                 frame = getframe(f);
653                 writeVideo(v,frame);
654             end
655             title(ax,sprintf('Animation of contour plot, $t$ = %.2f s',obj.T),'
                    ↪ Interpreter','Latex');
656             frame = getframe(f);
657             writeVideo(v,frame);
658             close(v);
659         else
660             for i = 1:obj.Ntmax
661                 s.ZData = Qgrid(:,:,i);
662                 title(ax,sprintf('Animation of contour plot, $t$ = %.2f s',(i-1)*obj.dt*
                        ↪ obj.tStep),'Interpreter','Latex');
663                 pause(obj.dt*obj.tStep);
664             end
665         end
666         fprintf('Ready.\n')
667     end
668 
669     function plotSolution(obj,time,azimuth,elevation,printFigures,filePath,
            ↪ filenamePrefix)
670         if ~obj.solutionIsReady
671             fprintf('Error in plotSolution: System is not solved.\n');
672             return
673         end
674 
675         if time > obj.T
676             time = obj.T;
677             fprintf('Warning in plotSolution: Time out of bounds. Plot is at t=%4.2f.\n
                    ↪ ',obj.T);
678         elseif time < 0
679             time = 0;
680             fprintf('Warning in plotSolution: Time out of bounds. Plot is at t=0.\n');
681         end
682 
```

```
683             timeIndex = round(time/obj.T*(obj.Ntmax-1)+1);
684
685             figure('Name',['Plot at t=' num2str(time)],'Color','white');
686             ax = axes;
687             [Qgrid,Xgrid,Ygrid] = obj.indexVectorToGrid;
688             surf(Xgrid,Ygrid,Qgrid(:,:,timeIndex),'EdgeColor','none');
689             ax.FontSize = 14;
690             Qmax = max(Qgrid,[],'all');
691             axis(ax,[0,obj.Lxout,0,obj.Ly,0,Qmax]);
692             colorbar
693
694             view(ax,azimuth,elevation);
695             fontsize = 16;
696
697             xlabel('$x$ (m)','Interpreter','Latex','Fontsize',1.5*fontsize);
698             ylabel('$y$ (m)','Interpreter','Latex','Fontsize',1.5*fontsize);
699             zlabel(['$c(x,y,' num2str(time) ')$ kg/m$^3$'],'Interpreter','Latex','Fontsize'
                    ↪ ,1.5*fontsize);
700
701             if printFigures
702                 filename = sprintf('%s/%s_solution_plot_t=%.0fms',filePath,filenamePrefix,
                        ↪ time*1000);
703                 print(filename,'-dpng','-r300')
704             end
705         end
706
707         function contourPlotSolution(obj,time,printFigures,filePath,filenamePrefix)
708             if ~obj.solutionIsReady
709                 fprintf('Error in plotSolution: System is not solved.\n');
710                 return
711             end
712
713             if time > obj.T
714                 time = obj.T;
715                 fprintf('Warning in plotSolution: Time out of bounds. Plot is at t=%4.2f.\n
                        ↪ ',obj.T);
716             elseif time < 0
717                 time = 0;
718                 fprintf('Warning in plotSolution: Time out of bounds. Plot is at t=0.\n');
719             end
720
721             timeIndex = round(time/obj.T*(obj.Ntmax-1)+1);
722
```

```
723             figure('Name',['Contour Plot at t=' num2str(time)],'Color','white');
724             [Qgrid,Xgrid,Ygrid] = obj.indexVectorToGrid;
725             contour(Xgrid,Ygrid,Qgrid(:,:,timeIndex),'k');
726             hold('on');
727             plot([0 obj.Lx],[obj.layerBoundaries(2) obj.layerBoundaries(2)],'k')
728             axis([0,obj.Lxout,0,obj.Ly]);
729 
730             xlabel('Distance $x$ (m)','Interpreter','Latex');
731             ylabel('Distance $y$ (m)','Interpreter','Latex');
732 
733             if printFigures
734                 filename = sprintf('%s/%s_contour_plot_t=%.0fms',filePath,filenamePrefix,
                        ↪ time*1000);
735                 print(filename,'-dpng','-r300')
736             end
737         end
738 
739         function [N_TD,I_TD,tm] = breakthrough(obj,plotDoubleLayerLimit,inPVI,printFigures
            ↪ ,filePath,filenamePrefix)
740             if ~obj.solutionIsReady
741                 fprintf('Error in breakthrough: System is not solved.\n');
742                 return
743             end
744             fontSize = 24;
745 
746             time = obj.t;
747 
748 
749             if plotDoubleLayerLimit
750                 Dl = obj.alpha_l*obj.layerVelocities+obj.Dm;
751                 doubleLayer = @(x,t,u,h,Dl) ((h(2)-h(1))*(1-erf((x-u(1)*t)./(2*sqrt(Dl(1)*t
                        ↪ ))))+(h(3)-h(2))*(1-erf((x-u(2)*t)./(2*sqrt(Dl(2)*t)))))/(2*h(3));
752             end
753 
754             outflow = 0.5*(obj.outflowLayer1 + obj.outflowLayer2);
755 
756             tm = 0;
757             try
758                 for i = 1:length(time)
759                     if outflow(i) < 0.5 && outflow(i+1) > 0.5
760                         tm = interp1([outflow(i) outflow(i+1)],[time(i) time(i+1)],0.5);
761                         break
762                     end
```

```
763             end
764         catch
765             warning('tm could not be found.');
766         end
767         tm1 = 0;
768         try
769             for i = 1:length(time)
770                 if obj.outflowLayer1(i) < 0.5 && obj.outflowLayer1(i+1) > 0.5
771                     tm1 = interp1([obj.outflowLayer1(i) obj.outflowLayer1(i+1)],[time(i)
                            ↪  time(i+1)],0.5);
772                     break
773                 end
774             end
775         catch
776             warning('tm1 could not be found.');
777         end
778         tm2 = 0;
779         try
780             for i = 1:length(time)
781                 if obj.outflowLayer2(i) < 0.5 && obj.outflowLayer2(i+1) > 0.5
782                     tm2 = interp1([obj.outflowLayer2(i) obj.outflowLayer2(i+1)],[time(i)
                            ↪  time(i+1)],0.5);
783                     break
784                 end
785             end
786         catch
787             warning('tm2 could not be found.');
788         end
789
790         if (obj.layerBoundaries(2)-obj.layerBoundaries(1))*obj.layerPorosity(1)*obj.
                ↪ layerVelocities(1)>=(obj.layerBoundaries(3)-obj.layerBoundaries(2))*obj.
                ↪ layerPorosity(2)*obj.layerVelocities(2)
791             QD = obj.multiLayerPVI(tm1);
792             QDi = obj.multiLayerPVI(obj.Lxout/obj.layerVelocities(1));
793             I_TD = (QD-QDi)/(1-QDi);
794         elseif (obj.layerBoundaries(2)-obj.layerBoundaries(1))*obj.layerPorosity(1)*obj
                ↪ .layerVelocities(1)<(obj.layerBoundaries(3)-obj.layerBoundaries(2))*obj.
                ↪ layerPorosity(2)*obj.layerVelocities(2)
795             QD = obj.multiLayerPVI(tm2);
796             QDi = obj.multiLayerPVI(obj.Lxout/obj.layerVelocities(2));
797             I_TD = (QD-QDi)/(1-QDi);
798         else
799             error('I_TD could not be calculated.')
```

```
800             end
801
802             N_TD = (obj.Lxout/obj.Ly)^2*(obj.alpha_t*min(obj.layerVelocities)/(obj.Lxout*
                    ↪ max(obj.layerVelocities)));
803
804             figure('Name','Breakthrough','Color','white');
805             ax = axes;
806             hold(ax,'on');
807             if inPVI
808                 if plotDoubleLayerLimit
809                     plot(ax,obj.multiLayerPVI(obj.t),doubleLayer(obj.Lxout,obj.t,obj.
                            ↪ layerVelocities,obj.layerBoundaries,Dl),'k-.','LineWidth',1);
810                 end
811
812                 plot(ax,obj.multiLayerPVI(time),outflow,'k-','LineWidth',1.2);
813                 plot(ax,obj.multiLayerPVI(time),obj.outflowLayer1,'k--','LineWidth',1.2);
814                 plot(ax,obj.multiLayerPVI(time),obj.outflowLayer2,'k:','LineWidth',1.2);
815                 plot(ax,[0 obj.multiLayerPVI(obj.T)],[0.5 0.5],'k','LineWidth',1);
816                 plot(ax,[0 obj.multiLayerPVI(tm) obj.multiLayerPVI(tm)], [0.5 0.5 0], 'k-',
                        ↪ 'LineWidth',1.2)
817                 plot(ax,[0 obj.multiLayerPVI(tm1) obj.multiLayerPVI(tm1)], [0.5 0.5 0], 'k
                        ↪ --','LineWidth',1.2)
818                 plot(ax,[0 obj.multiLayerPVI(tm2) obj.multiLayerPVI(tm2)], [0.5 0.5 0], 'k
                        ↪ -.','LineWidth',1.2)
819                 ax = gca;
820                 ax.FontSize = 14;
821                 legend('Total $c_\mathrm{out}$','Layer 1 $c_\mathrm{out}$','Layer 2 $c_\
                        ↪ mathrm{out}$','Interpreter','LaTeX','FontSize',fontSize,'location','
                        ↪ southeast')
822
823                 axis(ax,[0 obj.multiLayerPVI(obj.T) 0 1]);
824                 xlabel(ax,'PVI','Interpreter','LaTeX','FontSize', fontSize);
825
826                 if N_TD > 0.02
827                     text(obj.multiLayerPVI(obj.T)/10,0.9,sprintf('$N_{TD}$=%.2e\n$I_{TD}$
                            ↪ =%.3f\n$(Q_{Di})_{c=0.5}$=%.3f\n$(Q_{D})_{c=0.5}$=%.3f',N_TD,I_TD
                            ↪ ,QDi,QD),'Interpreter','LaTeX','FontSize', fontSize)
828                 else
829                     text(obj.multiLayerPVI(obj.T)/10,0.9,sprintf('$N_{TD}$=%.2e\n$I_{TD}$
                            ↪ =%.3f\n$(Q_{Di})_{c=0.5}$=%.3f\n$(Q_{D})_{c=0.5}$=%.3f',N_TD,I_TD
                            ↪ ,QDi,QD),'Interpreter','LaTeX','FontSize', fontSize)
830                 end
831             else
```

```
832             if plotDoubleLayerLimit
833                 plot(ax,obj.t,doubleLayer(obj.Lxout,obj.t,obj.layerVelocities,obj.
                        ↪ layerBoundaries,Dl),'k-.','LineWidth',1);
834             end
835             plot(ax,time,outflow,'k');
836             plot(ax,time,obj.layerBoundaries(2)*obj.outflowLayer1/obj.layerBoundaries
                    ↪ (3),'b');
837             plot(ax,time,((obj.layerBoundaries(3)-obj.layerBoundaries(2))*obj.
                    ↪ outflowLayer2+obj.layerBoundaries(2))/obj.layerBoundaries(3),'r');
838             plot(ax,[0 obj.T],[0.5 0.5],'k','LineWidth',1);
839             plot(ax,[0 tm tm], [0.5 0.5 0], 'k','LineWidth',1)
840             plot(ax,[0 tm1 tm1], [(obj.layerBoundaries(2)-obj.layerBoundaries(1))*0.5/
                    ↪ obj.layerBoundaries(3) (obj.layerBoundaries(2)-obj.layerBoundaries(1)
                    ↪ )*0.5/obj.layerBoundaries(3) 0], 'b','LineWidth',1)
841             plot(ax,[0 tm2 tm2], [(obj.layerBoundaries(3)-obj.layerBoundaries(2))*0.5/
                    ↪ obj.layerBoundaries(3)+obj.layerBoundaries(2)/obj.layerBoundaries(3)
                    ↪ (obj.layerBoundaries(3)-obj.layerBoundaries(2))*0.5/obj.
                    ↪ layerBoundaries(3)+obj.layerBoundaries(2)/obj.layerBoundaries(3) 0],
                    ↪ 'r','LineWidth',1)
842             axis(ax,[0 obj.T 0 1]);
843             xlabel(ax,'$t$','Interpreter','LaTeX','FontSize', fontSize);
844
845             if N_TD > 0.02
846                 text(obj.T/10,0.9,sprintf('$N_{TD}$=%.2e\n$I_{TD}$=%.3f\n$(Q_{Di})_{c
                        ↪ =0.5}$=%.3f\n$(Q_{D})_{c=0.5}$=%.3f',N_TD,I_TD,QDi,QD),'
                        ↪ Interpreter','LaTeX','FontSize', fontSize)
847             else
848                 text(obj.T/10,0.9,sprintf('$N_{TD}$=%.2e\n$I_{TD}$=%.3f\n$(Q_{Di})_{c
                        ↪ =0.5}$=%.3f\n$(Q_{D})_{c=0.5}$=%.3f',N_TD,I_TD,QDi,QD),'
                        ↪ Interpreter','LaTeX','FontSize', fontSize)
849             end
850         end
851
852         ylabel(ax,'$c_\mathrm{out}$','Interpreter','LaTeX','FontSize', 1.5*fontSize);
853
854         if plotDoubleLayerLimit
855             legend(ax,'Double layer limit','Numerical solution','Layer 1','Layer 2','
                    ↪ Location','southeast','Interpreter','LaTeX','FontSize', fontSize);
856         end
857
858         if printFigures
859             filename = sprintf('%s/%s_breakthrough.png',filePath,filenamePrefix);
860             print(filename,'-dpng','-r300')
```

```
            end

        end

        function PVI = multiLayerPVI(obj,t)
            coef = 0;
            for i = 1:length(obj.layerVelocities)
                coef = coef + (obj.layerBoundaries(i+1)-obj.layerBoundaries(i))*obj.
                    ↪ layerVelocities(i)*obj.layerPorosity(i);
            end

            phi = 0;
            for i = 1:length(obj.layerPorosity)
                phi = phi + (obj.layerBoundaries(i+1)-obj.layerBoundaries(i))*obj.
                    ↪ layerPorosity(i);
            end


            PVI = coef*t/(obj.Lxout*phi);
        end

        function velocity = ux(obj,y)
            velocity = 0.*y;
            for i = 1:length(obj.layerVelocities)
                velocity = velocity + obj.layerVelocities(i)*heaviside(y-obj.
                    ↪ layerBoundaries(i)).*heaviside(-y+obj.layerBoundaries(i+1));
            end
        end

    end
end
```

# References


[1] Jacob Bear. *Dynamics of fluids in porous media.* American Elsevier Publishing Company, Inc., New York, 1972.

[2] Eric Dubach. Artificial boundary conditions for diffusion equations: Numerical study. *Journal of Computational and Applied Mathematics*, 70:127–144, 06 1996.

[3] Gholamreza Garmeh, Russell Johns, and Larry Lake. Pore-scale simulation of dispersion in porous media. *SPE Journal*, 14:559–567, 12 2009.

[4] David Griffiths. The 'no boundary condition' outflow boundary condition. *International Journal for Numerical Methods in Fluids*, 24:393–411, 02 1997.

[5] Ekkehard Holzbecher and S Oehlmann. Comparison of heat and mass transport at the micro-scale. 12 2019.

[6] Raman Jha, Steven Bryant, and Larry Lake. Effect of diffusion on dispersion. *SPE Journal*, 16:65–77, 03 2011.

[7] Raman Jha, Abraham John, Steven Bryant, and Larry Lake. Flow reversal and mixing. *SPE Journal*, 14, 09 2006.

[8] Abraham John, Larry Lake, Steven Bryant, and James Jennings. Investigation of field scale dispersion. *Proceedings - SPE Symposium on Improved Oil Recovery*, 2, 01 2008.

[9] Donald Koch and John Brady . Dispersion in fixed beds. *Journal of Fluid Mechanics*, 154:399 – 427, 05 1985.

[10] Larry Lake and G. Hirasaki. Taylor's dispersion in stratified porous media. *Society of Petroleum Engineers Journal*, 21:459–468, 08 1981.

[11] Randall J. Leveque. *Finite Volume Methods for Hyperbolic Problems.* Cambridge University Press, New York, 2002.

[12] Jean-Pierre Lohéac. An artificial boundary condition for an advection-diffusion equation. *Mathematical Methods in the Applied Sciences*, 14:155–175, 03 1991.

[13] Morris Muskat and Milan Meres. The flow of heterogeneous fluid through porous media. *Physics*, 7:346 – 363, 10 1936.

[14] Vladimir Ostapenko. On the monotonicity of multidimensional difference schemes. *Doklady Mathematics*, 86, 11 2012.

[15] T. Perkins and O. Johnston. A review of diffusion and dispersion in porous media. *Society of Petroleum Engineers Journal*, 3:70–84, 03 1963.

[16] P. Saffman . A theory of dispersion in a porous medium. *Journal of Fluid Mechanics*, 6:321 – 349, 10 1959.

[17] Muhammad Sahimi. *Flow and Transport in porous media and fractured rock.* Weinheim, 1995.

[18] Geoffrey Taylor. The dispersion of soluble matter in solvent flowing slowly through a tube. *Proceedings of The Royal Society A: Mathematical, Physical and Engineering Sciences*, 219:186–203, 08 1953.

[19] Joseph Warren and Francis Skiba. Macroscopic dispersion. *Society of Petroleum Engineers Journal*, 4:215–231, 09 1964.

[20] P. Wesseling. *Principles of Computational Fluid Dynamics.* Springer-Verlag, Berlin, 2001.

[21] Stephen Whitaker. Diffusion and dispersion in porous media. *AIChE Journal*, 13:420 – 427, 05 1967.

[22] Stephen Whitaker. Flow in porous media i: A theoretical derivation of darcy's law. *Transport in Porous Media*, 1:3–25, 03 1986.