Numerical and analytical model for dipole tracer tests in heterogeneous aquifers

Oscar Zeijlmans van Emmichoven
6481108

Master Thesis
Final version
Supervisors: Dr. Alraune Zech, Prof.Dr. Ruud Schotting

Department of Earth Science
Utrecht University
The Netherlands
2024

Contents

Abstract

Heterogeneity of hydraulic conductivity in aquifers is a topic touched upon by many researchers, but very few analyses were reported on the characteristic parameters of solute transport using stochastic methods. We study he effects of aquifer heterogeneity by running numerical simulations of dipole tracer tests in the software *MODFLOW 6*, instigated with Python scripts using the *FloPy* package. Random hydraulic conductivity fields are drawn by the Python package *GStools* from a log-normal distribution, characterized by a mean, variance and correlation length. A series of ensembles was run with varying variances and correlation lengths. Normalised mean breakthrough curves of the ensembles were compared for homogeneous, stratified and heterogeneous aquifers. Outcomes show that an increase in variance leads to an earlier arrival of the tracer, due to the occurrence of preferential flow paths, but will also lead to more tailing. An increase in correlation length enhances the effects of an increase in variance. For a short correlation length (i.e. much smaller than the distance between the wells) the heterogeneous solution tends towards the solution for a homogeneous medium, while for a longer correlation length, the solution tends towards the solution for a stratified medium. Based on the ensemble results we conclude that a fully heterogeneous aquifer shows a different breakthrough curve than stratified or a homogeneous structured aquifers. Therefore it cannot be treated the same way.

1 Introduction

The monitoring and modelling of groundwater is of utmost importance as groundwater is a vital source for clean drinking water supply and irrigation in agriculture. Globally, over 2.5 billion people (32% of the world population) depend solely on groundwater for their daily needs of drinking water [*Grönwall and Danert*, 2020]. In the Netherlands up to 60% of drinking water comes from groundwater [*Versteegh and Dik*, 2015]. The composition of groundwater differs, both, in space and time, and depends on a number of factors including, but not limited to parent rock, intensity of weathering, residence time and external factors, such as precipitation and temperature, the presence of microorganisms, and land use [*Brindha et al.*, 2014]. The combination of all these factors leads to whether groundwater is suitable for drinking water and industrial use. Unfortunately, groundwater is very susceptible to contamination coming from the land surface. Leaching nitrate from fertilisers [*Nolan and Hitt*, 2006], leaking septic tanks [*Burchart-Korol and Zawartka*, 2019] or discharge of industrial wastewater [*Hussain et al.*, 2021] are only a few examples of possible sources threatening the quality of groundwater. Once a contaminant has reached the groundwater it is important to determine where the pollutant will end up to make a proper risk assessment.

The movement of a conservative solute with the groundwater is determined by advection and dispersion. Advection is the passive transport of the solute with the flow of the groundwater and dispersion is the spreading of solute in directions other than the flow velocity. Dispersion can be separated in mechanical dispersion and molecular diffusion, where mechanical dispersion is caused by differences in flow velocity (whether that is due to differences in hydraulic conductivity, or flow paths differing in length, or resistance within the pore space) and molecular diffusion is caused by the random movement of the molecules themselves. In practise they cannot be distinguished. Consequently they are combined in a single dispersion coefficient [*Shi et al.*, 2016].

To accurately model flow through the subsurface, a great sum of data must be acquired. Specifically, the hydraulic conductivity and dispersion of an aquifer are of interest as they determine the flow velocity through the subsurface and spreading of solutes in the groundwater. This can be done through tracer tests. In a tracer test a tracer is introduced into the subsurface. This tracer is then followed through observation points and finally measured at some extraction point. A wide variety of tracer tests exists, each with their own advantages and disadvantages. One kind of tracer tests are dipole tracer tests (DTT), also named two-well test or doublet test. Here two wells are installed in the subsurface; an injection well (or recharge well) and an extraction (or pumping) well. By injecting and extracting similar amounts of water simultaneously, a dipole flow pattern arises between the two wells, hence the name "dipole test" (Figure 1, *Zech et al.* [2018]). After establishing the flow pattern a tracer with a known concentration is added to the injection well. The tracer particles move over the streamlines from the injection to the extraction well. At the extraction well the arrival time of the tracer is measured and a breakthrough curve (BTC) can be constructed [*Gelhar and Leonhart*, 1982].

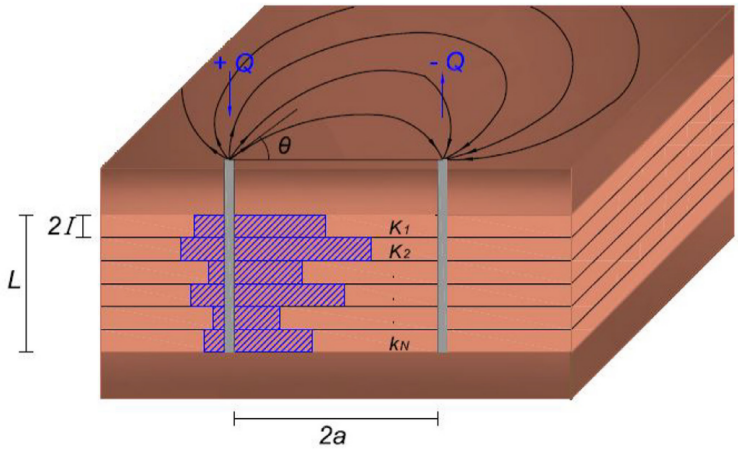

Figure 1: Conceptual diagram of a two-well tracer test setup (Figure from *Zech et al.* [2018]).

The most common issue when modelling groundwater flow is the availability of accurate parameters that apply to the scales of concern in the field [*Gelhar*, 1993]. This issue arises from the fact that hydraulic properties of a subsurface are extremely variable in space. Determining the right parameters for a model has proven to be challenging. The equations and parameters used in these models are often tested on the scale of laboratory experiments or small scale field experiments, while, for example, regional scale groundwater models apply the equations on the scale of several kilometres, thus assuming a certain degree of homogeneity that in reality

does not exist. *Gelhar* [1993] in his book *Stochastic Subsurface Hydrology* and brings up three suggestions to circumvent, or at least minimise, the issue:

- The first suggestion states is to measure the actual three-dimensional distribution of, for example, hydraulic conductivity in complete detail for a field site. The data is then used in a numerical model that captures all of the effects of the variability. However, this gives rise to a new problem for the scale of the model and the variability may differ several orders of magnitude (which is almost always the case). In practice, this solution is not feasible for two reasons: one being the computational burden. The other being the task to obtain the detailed measurements. Such a program would be impractical, both, financially and time wise; ignoring the possibility that this may even significantly alter the hydraulic properties of the subsurface.
- The second suggestion is to ignore variability in the subsurface. This assumes that the flow and transport equations are valid in some average sense for the large-scale field problems. Since the model no longer considers the variability, that raises the question whether such a model is still representative of the real situation.
- The third suggestion is to express the variability of the parameters in the subsurface in the form of a random variable. Instead of identifying all variability in a system, the heterogeneity of the system is assumed to be, to a certain extend, random. This is done by describing the variations in the hydraulic properties using stochastic processes or random fields. When solved, these stochastic equations produce probabilistic results. This approach still requires an adequate amount of field data before being applied to specific field situations and the results will also include a level of uncertainty to estimate reliability.

A lot of research has been conducted in the field of heterogeneous conductivity in dipole flow setting, but very few analyses were reported on the characteristic parameters of solute transport using stochastic test methods in dipole flow settings. In this study we explore the effects of heterogeneity in hydraulic conductivity on the shape of the break through curve (BTC) in a dipole tracer test setting, through a stochastic approach. We compare numerical dipole tracer tests (DTT) for a homogeneous, stratified and heterogeneous setting. In addition, we will compare the numerical results to the analytical solution for a DTT in a stratified subsurface derived by *Zech et al.* [2018]. Specifically, we address the research questions:

- What is the impact of the log-normal conductivity heterogeneity parameters correlation length and variance, on the shape of the BTC?
- How does the BTC for transport in fully heterogeneous conductivity compare to that of stratified and homogeneous subsurface structures?
- How do the numerical simulation results compare to the analytical solution derived by *Zech et al.* [2018]?

All data and Python code for the numerical models used in this research will be available on GitHub with documentation.

2 Methods

2.1 Groundwater Flow and Mass Transport Equation

Solute transport in saturated media is described by the advection-dispersion equation (ADE):

$$\frac{\partial}{\partial t}(\theta c) + \frac{\partial}{\partial x_i}(q_i c) = \frac{\partial}{\partial x_i}\left(\theta D_{ij}\frac{\partial c}{\partial x_j}\right) + \theta R: \qquad i,j = 1,2,3 \tag{1}$$

Where c is the species concentration $[M/L^3]$, q_i the groundwater flux along the specified axis, D_{ij} is a dispersion coefficient tensor $[L^2/T]$, R is (source of species)/(time and volume of fluid) $[M/L^3T]$ and θ is porosity. Depending on the complexity of the field situation, this equation can be solved either analytically or numerically [*Gelhar*, 1993]. Nowadays there are many methods to approximate complex three-dimensional situations for both flow and transport, including a wide variety of software packages such as *STANMOD* (analytical solver of advection-dispersion equation, *Van Genuchten et al.* [2012]) or *MODFLOW* (framework for numerical modelling of groundwater flow and groundwater transport, *Langevin et al.* [2017]).

The interest in hydraulic conductivity as decisive parameter originates from Darcy's law, the equation describing groundwater flow and flow velocity (eq. 2):

$$v = \frac{-K}{\theta}\nabla h \tag{2}$$

where v is the flow velocity, ∇h the gradient in hydraulic head, θ the porosity and K the hydraulic conductivity. Differences in porosity are within one order of magnitude and differences in the hydraulic gradient are in a typical field setting also relatively small. However, the hydraulic conductivity can vary over 13 orders of magnitude, making it in most cases the decisive parameter for flow velocity [*Kohlbecker et al.*, 2006].

2.2 Numerical Model

2.2.1 MODFLOW 6

We solve the ADE numerically using the software MODFLOW, the USGS's open-source modular hydrologic model[*Langevin et al.*, 2017]. It provides a framework for modellers to simulate simple and complex groundwater scenarios within the same model. We use the most recent version MODFLOW 6 as it allows for both, groundwater flow and transport modelling, using irregular spaced grids. The groundwater flow model for MODFLOW 6 is based on a generalized control-volume finite-difference (CVFD) approach in which a cell can be hydraulically connected to any number of surrounding cells [*Langevin et al.*, 2017].

MODFLOW does not come with a standard graphical user interface (GUI), but instead reads and returns a series of text files containing all information of the model. To generate the input files and visualise/process the output files, we use the Python package *FloPy* [*Bakker et al.*, 2023]. FloPy enables us to setup Python scripts to write and read MODFLOW files and thus run multiple simulations in a quick and organised manner.

2.2.2 Model Setup

The model scenario is similar to the schematic setup in figure 1. Two wells (an injection and extraction well) are placed in a square domain. Both wells are fully penetrating and pump at an equal but opposite rate throughout the whole simulation time. The boundaries of the domain are fixed at a constant head of $0m$. At the start of a run ($t = t_0$), a non-reactive, non-absorbing tracer is introduced to the injection well. A concentration of 1000 g/m^3 is injected for 6 hours at a rate of 10^{-4} m^3/s. After these 6 hours, the injection well injects clean water. For 200 days, every hour (=4800 hours) the concentration in the system is determined. Since there is no way for the tracer to leave the system other than the extraction well, all mass leaving the system must be extracted through the well and a BTC can be constructed.

Parameters used for runs with a heterogeneous subsurface are listed in Table 1. In the case of a non-homogeneous subsurface, the mean hydraulic conductivity refers to the underlying Gaussian distribution, which will be discussed in more depth in section 2.3

2.2.3 Irregular Grid

An advantage of using MODFLOW 6 is the use of irregular shaped grids. This enables us to increase the resolution in areas of interest i.e. the area directly around the wells, and lower the resolution in areas we are

Parameter	Symbol	Model settings	Default settings
Porosity	θ, n	0.3	0.3
Dist. between wells	-	10 m	1
Domain size	-	4000m x 4000 m	400 x 400 m
Aquifer depth	-	8 m	8 m
Number of layers	-	20	1
Mean hydr. cond.*	K_{mean}	10^{-4} m/s	$10^{-4} m/s$
Longitudinal dispersivity	α_L	0.5 m	0.5 m
Injection/extraction rate	Q	$(-)10^{-4} m^3/s$	$(-)10^{-4} m^3/s$
Injection concentration	C	1000 g/m^3	1000 g/m^3
Injection time	-	6 hours	6 hours
Post injection time	-	200 days = 4800 hours	10 days = 240 hours

Table 1: Settings used for simulating a dipole tracer test in a heterogeneous subsurface. *Mean and variance of underlying Gaussian distribution

not interested in i.e. far away from the wells.

The grid is roughly into three parts (Figure 2). First, a coarse grid is created of 25 by 25 cells. The centre cell of the coarse grid is divided into 64 by 64 cells. All solute transport is expected to take place in the refined area. Finally, a square around the centre of 2 by 2 times the distance between the wells is again refined by dividing every cell into 16 cells. All cell sizes are chosen dependent on the distance between the wells. The coarse cells are 16 by 16 times the distance between the wells, the middle grid, where the transport takes place, has cells with sides 0.25 times the well distance and the inner refined grid has cells with sides of 0.0625 times the well distance. An example grid for a distance of 10 meter between the wells can be seen in Figure 2. The grid is equally spaced over depth for the number of layers.

2.3 Heterogeneous Hydraulic Conductivity

2.3.1 Log-normal Fields

A Gaussian distribution is characterised by all values being close to a mean, with small and large numbers being the exception. In a field situation, however, researchers have found the distribution is more skewed towards the lower side, which is captured better by a log-normal distribution as it also favours lower values [*Muñoz-Carpena et al.*, 2002]. In addition, using a log-normal distribution has the advantage that negative values don't occur, which, again, matches with what is observed in the field. For these reasons the hydraulic conductivity is classically expressed by a log-normal probability density function (PDF) given by the equation:

$$Y = \ln X \tag{3}$$

where Y is Gaussian (normal) distributed and X is random and log-normal distributed. In a field situation the hydraulic conductivity is usually not fully random, but follows a certain spatial structure. To capture this spatial distribution, we use a 4 parameter log-normal distribution, which depends not only on variance σ^2 and mean μ, but also on a horizontal and vertical correlation length ℓ. The Gaussian spatial correlation function we use is defined as:

$$\gamma(r) = \sigma^2 \cdot \left(1 - \exp\left(-\left(s \cdot \frac{r}{\ell}\right)^2\right)\right) + n \tag{4}$$

where σ^2 is the variance, s is a scaling factor for unit conversion or normalisation ($= \frac{\sqrt{\pi}}{8}$), r is the lag distance, ℓ is the spatial correlation length and n is the nugget (subscale variance) [*Müller and Schüler*, 2019].

The log-normal hydraulic conductivity fields are produced using the Python package *GSTools*. Random fields are generated with the randomization method. Here, a spatial random field is represented by a stochastic Fourier integral and its discretised modes are evaluated at random frequencies [*Müller and Schüler*, 2019]. First, a random Gaussian distributed field is generated, given a mean, variance and correlation length (Eq. 4). Then, the value at the centre of each grid cell is log-transformed ($x \to \exp x$). An example of a fully heterogeneous field is shown in Figure 3. In case of a stratified structure, the correlation length is ignored and each layer is assigned a random, independent hydraulic conductivity (Figure 1), drawn from a log-normal distribution (Eq.3).

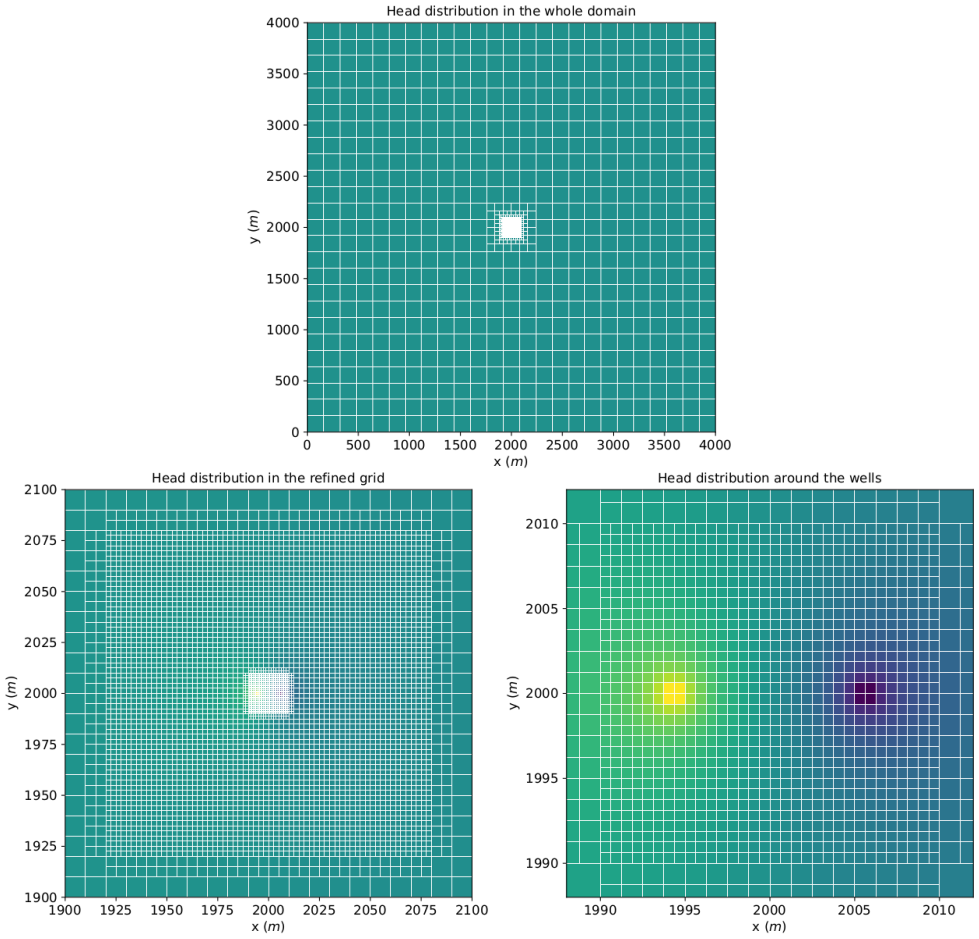

Figure 2: Model grid at different resolutions: Coarse outer grid of entire domain (top), medium refined transport grid (left) and extra refined grid in direct vicinity of the wells (right)

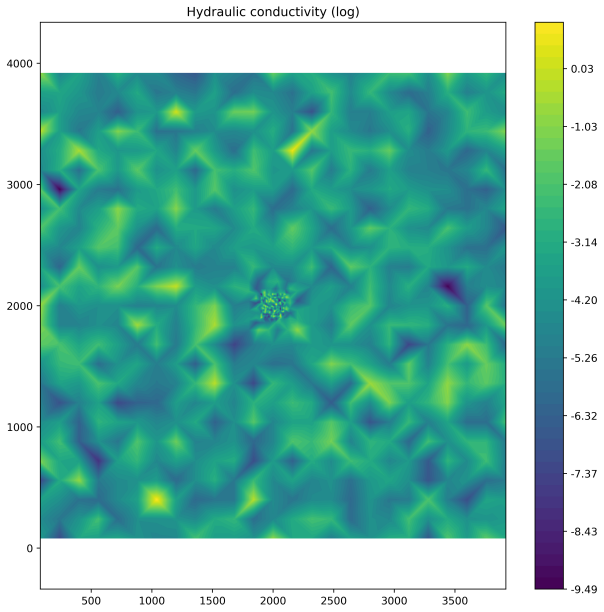

Figure 3: Example of a random normal $y = log_{10}K$ field with $\mu = 10^{-4}$, $\sigma^2 = 2$, $L = 10m$

2.3.2 Ensembles of Heterogeneous Fields

A multitude of ensembles is created to gain insight into the effect of the stochastic parameters. Within an ensemble the stochastic parameters (variance, mean and correlation length) are kept the same, but between ensembles the variance and correlation length differ. The stochastic parameters used for fully heterogeneous K-fields is shown in Table 2. We test for variances in log-hydraulic conductivity of 0.1, 0.5, 1, 2 and 4. These

Parameter	Symbol	Model Settings	Default Settings
Variance	σ^2	$0.1, 0.5, 1, 2, 4$	1
Horizontal Correlation Length	ℓ	$2.5m, 5m, 10m$	10

Table 2: Variance and horizontal correlation length for the ensembles for fully heterogeneous K-fields.

values align with field situations, where a value of 0.1 is almost homogeneous (= variance of 0) and a value of 4 represents a very heterogeneous aquifer. For the correlation length, we choose the values 2.5 m, 5.0 m, and 10.0m. These values of the correlation length do not exceed the distance between the wells (=10m). They are also large enough to be effectively captured by the model resolution (for a distance between the wells of 10m). The vertical correlation length is usually much smaller in the field than the horizontal correlation length. Therefore we set the vertical correlation length to 20% of the horizontal correlation length. For the stratified

Parameter	Symbol	Model Settings	Default Settings
Variance	σ^2	$0.1, 0.5, 1, 2, 4$	1
Number of layers	ℓ	10, 20, 40	1

Table 3: Variance and number of layers for the ensembles for stratified K-fields.

cases we apply the same variances of 0.1, 0.5, 1, 2 and 4. Stratified aquifers cannot have different horizontal correlation lengths. Instead, we consider the number of layers the aquifer is divided in. We created ensembles for 10, 20 and 40 layers. The upper limit of 40 layers has been chosen due to computation times rapidly increasing with additional layers as displayed in Figure 4. Where a simulation of a single layer takes a couple of minutes, a simulation with 40 layers takes 6 to 9 hours.

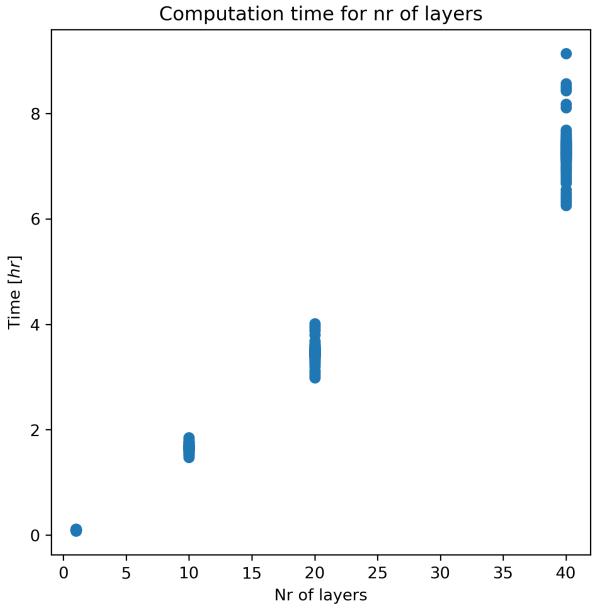

Figure 4: Computation time increases with the number of layers. Every dot represents a unique realisation.

2.3.3 Convergence Test

We tested ensemble converges by taking the sum of squares error (SSE). A sample containing the BTC's of x runs is created for which the mean BTC is determined by taking the arithmetic mean. We then take the total difference between the sample mean BTC and the ensemble mean for all 192 runs:

$$SSE = \sum_{t=0}^{t=t_{end}} \left(\bar{Y}_x(t) - \mu(t)\right)^2 \tag{5}$$

where $\bar{Y}_x(t)$ is the arithmetic mean BTC after x runs and $\mu(t)$ is the total ensemble mean BTC. The closer the SSE is to 0, the better the match between the sample mean and ensemble mean.

2.4 Analytical Model for Stratified Subsurface

The model supports two injection conditions: resident concentration (depth averaged), where the discharge is equally divided between all layers and flux proportional (in the model referred to as flux-averaged), where the discharge at the injection and extraction well is averaged based on the layers hydraulic conductivity. *Zech et al.* [2018] reported the semi-analytical solution for the probability density function (PDF), which is identical to the BTC. Assuming a stratified subsurface where each layer is assigned to a random hydraulic conductivity taken from a log-normal distribution, the BTC for resident concentration mode is given by the equation:

$$f_\tau^F(\tau) = \frac{1}{\pi\tau\sqrt{2\pi\sigma_Y^2}} \int_0^\pi exp\left[-\frac{\left(\sigma_Y^2/2 + \ln g\left(\theta\right) - \ln\tau\right)^2}{2\sigma_Y^2}\right] d\theta \tag{6}$$

and for flux proportional mode:

$$f_\tau^F(\tau) = \frac{1}{\pi\tau^2\sqrt{2\pi\sigma_Y^2}} \times \int_0^\pi g\left(\theta\right) exp\left[-\frac{\left(\sigma_Y^2/2 + \ln g\left(\theta\right) - \ln\tau\right)^2}{2\sigma_Y^2}\right] d\theta \tag{7}$$

where σ_Y^2 is the variance in log-hydraulic conductivity, $g(\theta)$ is given as

$$g(\theta) = \frac{1}{\sin^2\left(\theta\right)}\left(1 - \theta\cot\theta\right) \tag{8}$$

where θ is the angle of the flow lines from/towards the wells, and τ is the dimensionless travel time, defined as:

$$\tau = \frac{qt}{4\pi\theta a^2} \tag{9}$$

where q is the depth averaged flow velocity, t is time, θ the porosity and $2a$ the distance between the wells.

We created a Python implementation of the semi-analytical solution for both injection conditions (Figure 2.4). Figure 2.4 shows that for a low variance in log-hydraulic conductivity the difference in BTC between the two injection conditions is small, but increases for a larger variance. From the two injection conditions, flux proportional (Eq. 7) is the situation normally encountered in the field, hence forward we work with this condition, unless stated otherwise.

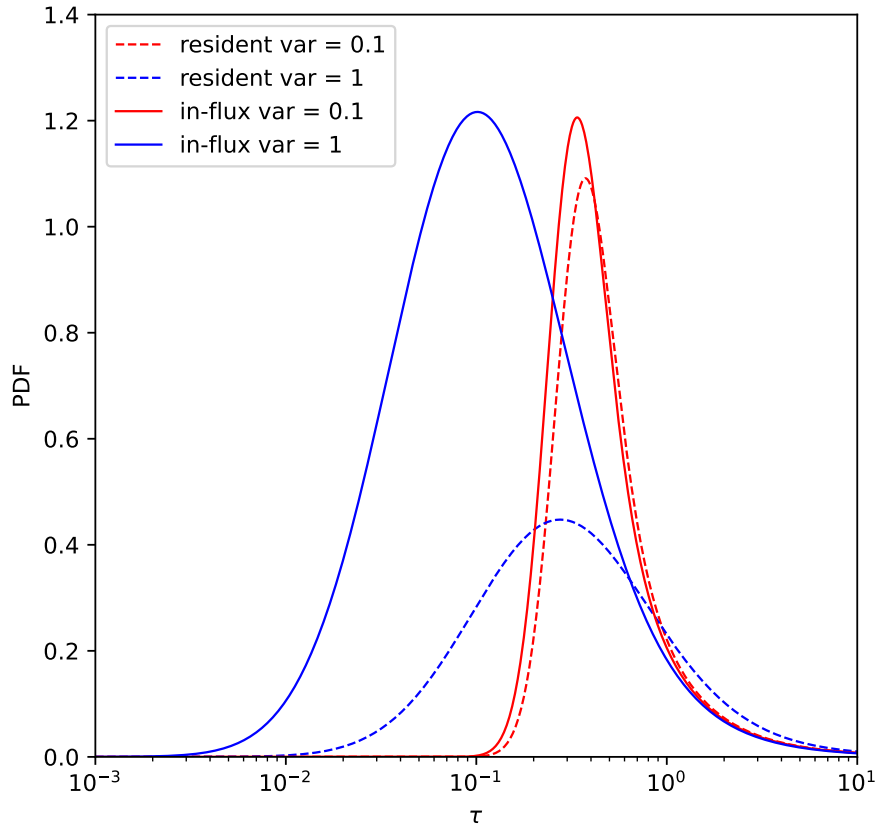

Figure 5: Implementation of the equations presented by *Zech et al.* [2018], showing the probability density function (=BTC) against travel time $\tau(= \frac{qt}{4\pi\theta a^2})$ for flux proportional ("in-flux"; solid lines) and resident modes ("resident", dashed lines) injection conditions, for $\sigma_Y^2 = 0.1$ (red lines) and $\sigma_Y^2 = 1$ (blue lines).

3 Results & Discussion

3.1 Heterogeneous Aquifers

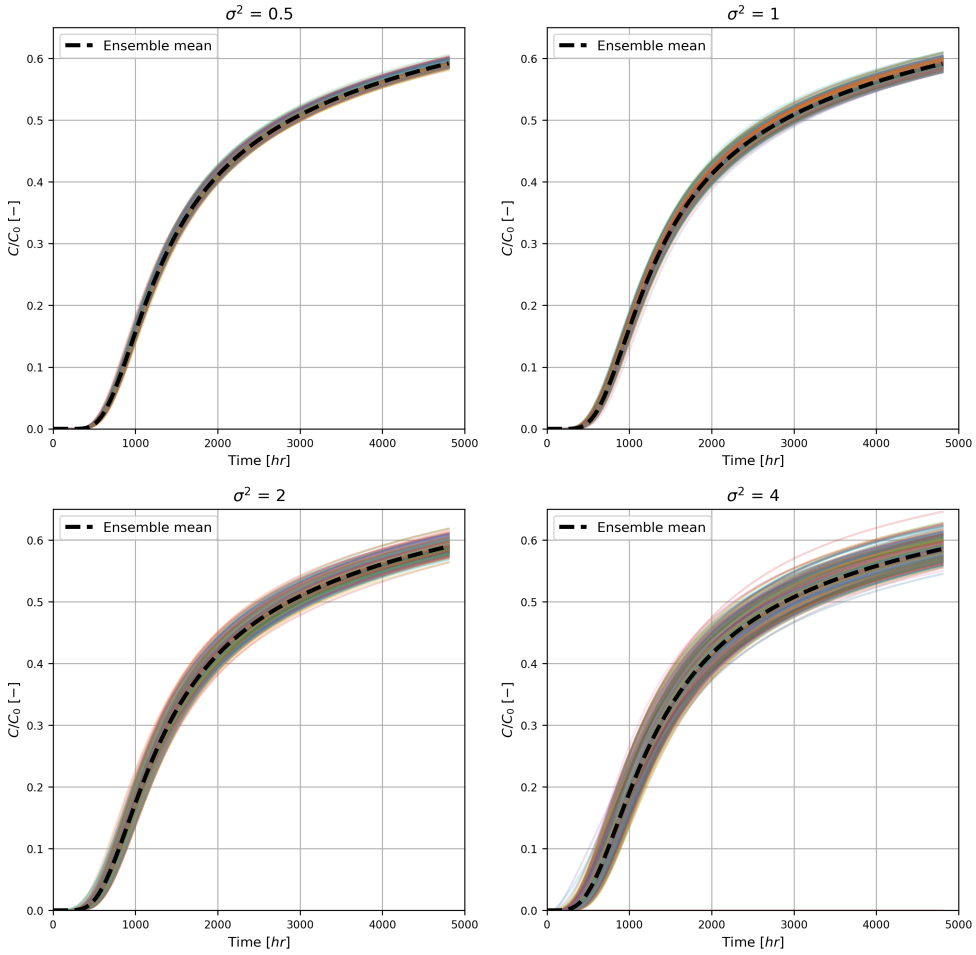

Figure 6: Breakthrough curves of all realisations fir each of the ensembles with ensemble mean in dashed black (sorted by variance)

Figure 6 shows the impact of the variance in hydraulic conductivity on the BTC. With an increase in variance, the spread of breakthrough curves increases for individual realisations. Especially for a variance of 2 and 4, the individual runs partially strongly differ from the ensemble mean. For lower variances (i.e. $\sigma^2 \leq 1$) the difference between individual runs only becomes noticeable in the tail i.e. at late times. When the variance increases, the deviation from the ensemble mean becomes noticeable much earlier, for $\sigma^2 = 4$ immediately when the tracer arrives at the extraction well.

The increasing spread in breakthrough curves with increasing heterogeneity is to be expected. With increasing variance in hydraulic conductivity, the differences in flow resistance between the flow paths increase. The groundwater will follow the path of least resistance. Thus, a small increase in hydraulic conductivity will lead to an increase in flow, a common phenomenon referred to as preferential flow. These preferential flow paths allow the groundwater to bypass parts of the aquifer with low hydraulic conductivity. Consequently, the residence time of solutes is shorter than expected based on the mean hydraulic conductivity [*Hagedorn and Bundt*, 2002]. The effect of preferential flow is also visible in Figure 7, where we directly compare the ensemble means for different variances (but same correlation length) for all 3 selected correlation lengths. On average, the tracer arrives earlier in aquifers with a high variances compared to aquifers with a low variance.

With a higher variance, not only does the tracer arrive earlier, it also lingers around for longer. Both of the extremes, high and low conductivity areas, increase. This means that, while the bulk of the tracer may arrive earlier, it will take longer for the slower flow paths to arrive at the extraction well. Transport at the slowest velocities becomes visible in the upper part, or tail, of the BTC, hence this is referred to as tailing. The effect of tailing is not that apparent in Figure 7, but the first signs of extended tailing at higher variances can already be seen. The final slope of the breakthrough curves at higher variances is less steep than for lower variances.

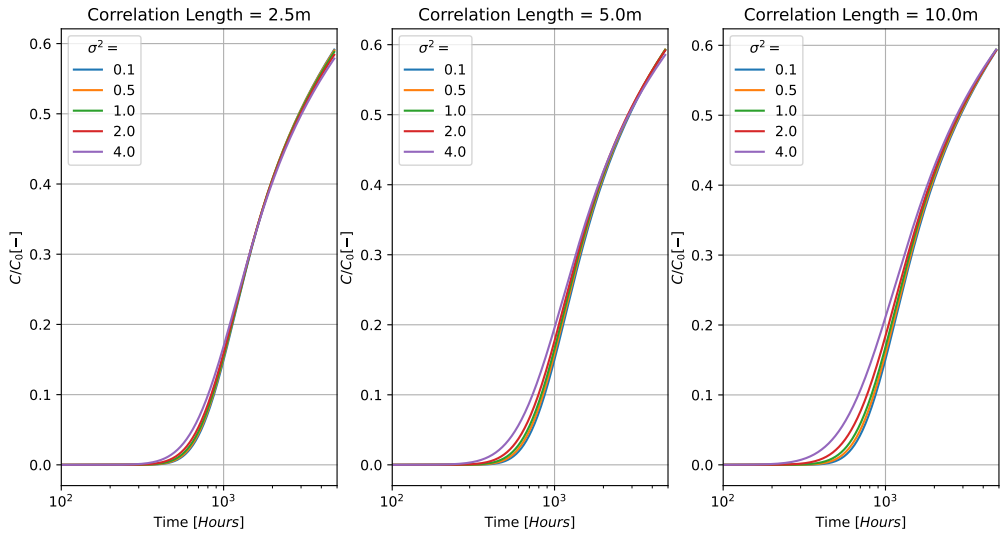

Figure 7: Ensemble mean breakthrough curves for the three correlation lengths 2.5m (left), 5m (centre), 10m (right). Note log-scale on time axis.

Figure 8 shows the ensemble mean BTCs for a heterogeneous structure for the different correlation lengths ℓ and variances $\sigma^2 = 0.1$, $\sigma^2 = 1$ and $\sigma^2 = 4$. We see that the tracer arrives a little faster for a larger correlation length, but this difference only becomes apparent at very high variances. In combination with Figure 7, we see that the effects of a large variance increase with the correlation length. If the correlation length is smaller than the distance between the wells, preferential flow is less strong since the tracer encounters different patterns of high and low conductivity on each flow path, either directly between wells or along a longer (dipole) flow path. When the correlation length becomes larger, being in the same range as the distance between the wells or even bigger, the difference in hydraulic conductivity between flow paths will be more pronounced, resulting in stronger preferential flow.

Figure 9 shows the results of the convergence test for fully heterogeneous structured K-fields for increasing variance σ^2 and different correlation lengths ℓ. This confirms that the total difference from the ensemble mean increases with increasing variance. For a low variance, the total difference is fairly low, but when more variance is introduced, the total difference increases multiple orders of magnitude. Comparing the ensembles for correlation length, we do not observe a clear trend. The deviations from the ensemble mean observed in Figure 9 are dominated by the difference in variance, but hardly affected by correlation length.

3.2 Stratified Aquifers

Figure 10 shows the ensemble mean BTC's for a stratified structure with 10 and 40 layers compared to a homogeneous and heterogeneous structure ($\ell = 10m$) for variances $\sigma^2 = 0.1$, $\sigma^2 = 1$ and $\sigma^2 = 4$. We observe similar behaviour for the stratified aquifers as for the heterogeneous aquifers. For an increasing number of layers, the variance makes a difference for both, the arrival time and tailing, of the tracer. Figure 10 also shows that the ensemble mean BTC is only noticeably affected for higher variances. Differences between BTCs are present mainly at the early arrival stage. This can again be ascribed to the appearance of preferential flow paths. With multiple layers, there will be high(er) conductivity layers and low(er) conductivity layers. Layers with higher conductivity will process much more than the lower conductivity layers, on average resulting in a faster arrival time.

Figure 10 shows that for a low variance (i.e. $\sigma^2 = 0.1$) the heterogeneous and stratified solution match very closely. This is to be expected as a variance of 0.1 corresponds to very little heterogeneity. When the variance increases, however, a divergence can be seen between the heterogeneous and stratified case. For a variance of 1 and higher, the heterogeneous case matches with neither the homogeneous case nor the stratified case with many layers, but ends up in between the two. For a variance of 4 the heterogeneous solution matches with the stratified solution for 10 layers during the early arrival of the tracer, but diverges for later times.

Figure 11 shows the mean ensemble breakthrough curves for both, stratified and heterogeneous conductivity, but now the variance is fixed to 4 and results for different correlation lengths are displayed. For a correlation

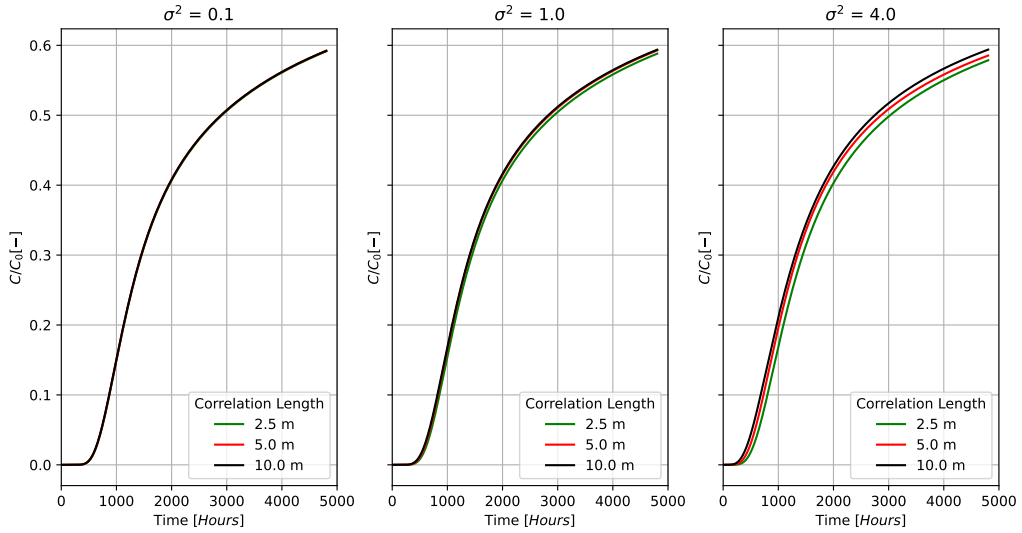

Figure 8: Ensemble mean breakthrough curves for three variances σ^2 of 0.1 (left), 1 (middle), 4 (right).

length of 2.5m, the heterogeneous case is closer to the homogeneous case, while with an increasing correlation length the BTC gets closer to the BTCs for stratified media with a very good match to the stratified case for 10 layers.

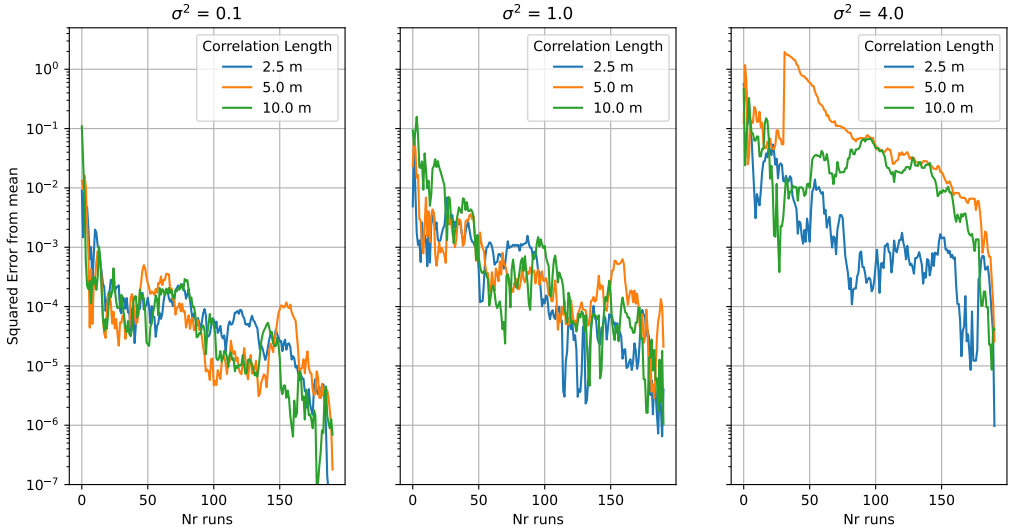

Figure 9: Convergence test results for ensembles with $\sigma^2 = 0.1$ (left), $\sigma^2 = 1$ (middle) and $\sigma^2 = 4$ (right) for the three tested correlation lengths. Shown is the total squared error from the ensemble mean for increasing number of runs (Eq. 5)

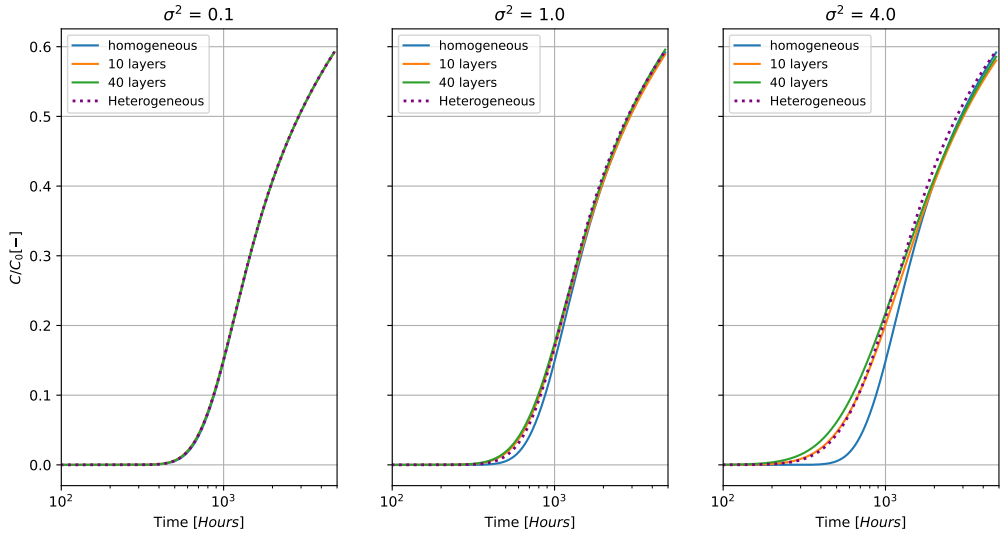

Figure 10: Mean breakthrough curves for a homogeneous structure, stratified structure divided into 10 and 40 layers and for a heterogeneous structure for correlation length of 10m.

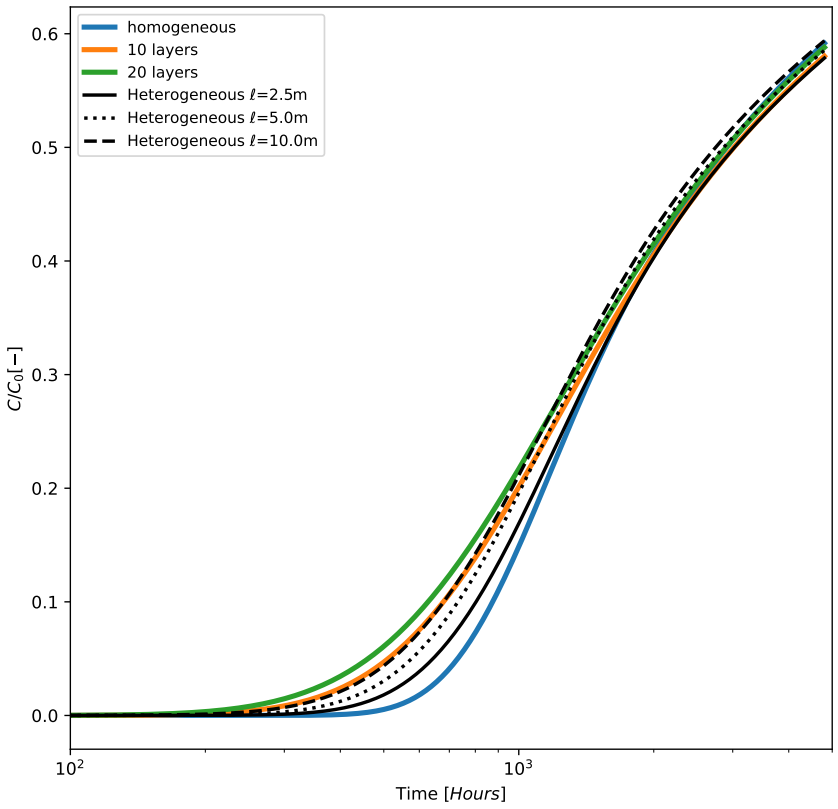

Figure 11: Mean breakthrough curves for a homogeneous aquifer and stratified aquifer with 10 and 20 layers compared to BTC for heterogeneous aquifer with similar variance and correlation lengths, 2.5m, 5.0m and 10m ($\sigma^2 = 4$) .

3.3 Analytical Solution for Stratified Aquifers

Figure 12 shows the BTCs of a stratified medium for different variances from eq. (7), being derived by *Zech et al.* [2018] and reproduced here through an independent Python implementation. The analytical solution differs in behaviour compared to the numerical solution shown in Figure 10. The analytical solution shows that (under ergodic conditions) the BTC shifts to the left with an increase in variance. While the numerical solution shows a similar behaviour for the early arrival of the tracer, for later times a higher variance in the model results in more tailing, as opposed to the analytical model, where this does not happen. The difference between the

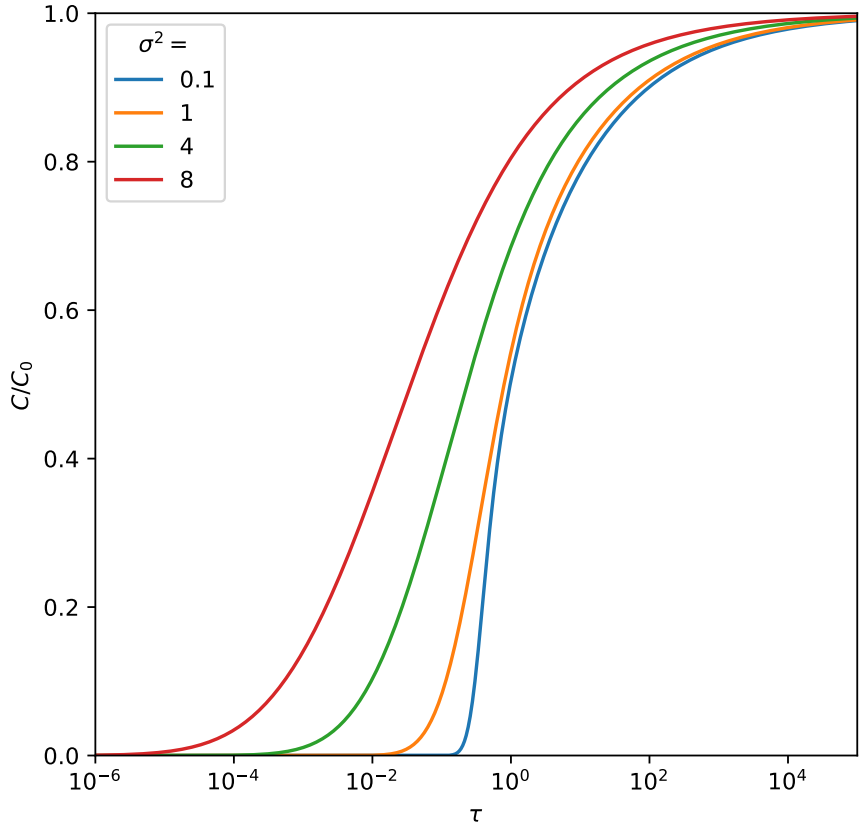

Figure 12: Analytical solution for stratified medium, derived by *Zech et al.* [2018], for variances σ^2 of 0.1, 0.5, 1, 2 and 4 ((7)) as a function of travel time $\tau(= \frac{qt}{4\pi\theta a^2})$.

numerical and analytical model becomes more apparent when both approaches are plotted next to each other. Figure 13 shows the analytical solution for a homogeneous K-structure plotted against the numerical solution for a homogeneous K-structure and ensemble means for a stratified and heterogeneous aquifer with a variance of 0.1 (=almost homogeneous). All three numerical solutions follow the same line. The analytical solution, however, arrives much earlier than the numerical solutions. Conceptually all lines need to be identical. We discuss potential causes for the discrepancies later in this section.

The analytical solutions arrives faster than the numerical solution by about a factor of $\sqrt{2}$. Figure13 (right) shows that the analytical and numerical solutions do match when shifted by a factor of $\sqrt{2}$, but still not a perfect match. The factor of $\sqrt{2}$ is also specific for these settings as the fitting factor seems to depend on the distance between the wells and also differs for higher variances.

The discrepancy between the analytical and numerical model can result from different causes:

- **Typographical error:** To rule out any mistakes in the Python implementation, both, the analytical and numerical solution have been thoroughly checked on typographical errors. It is highly unlikely this is the cause for the discrepancy.
- **Dispersion:** The numerical model includes dispersion while the analytical solution does not. The presence

of dispersion affects the shape of the BTC. However, dispersion alone is not enough to make up for the discrepancy. Results from simulations done without dispersion ($\alpha_L = 0$) show the same discrepancy observed in Figure 13.

- **Numerical issue Modflow 6:** The version of Modflow used for this study is fairly new and is still in active development. During this study, numerical inconsistencies where encountered when creating the model. The current version of the model, however, shows consistent results for the different K-field structures. This indicates the discrepancy comes not from within Modflow, but is rather due to a conceptual difference between the numerical and analytical solution.

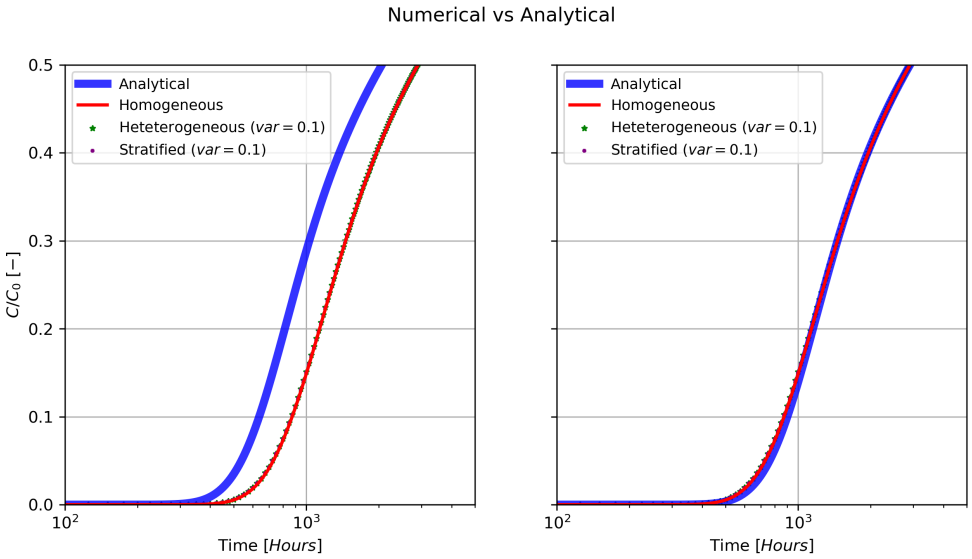

Figure 13: Analytical and numerical solutions for homogeneous, stratified and heterogeneous K-structure for original (left) and fitted (right) analytical solution.

3.4 Model Remarks and Recommendations

Researchers showed the hydraulic conductivity distribution for very heterogeneous structures is not always successfully captured by a log-normal distribution. Instead, other distributions, for example, a heavy-tailed Levy α stable distribution [*Muñoz-Carpena et al.*, 2002; *Kohlbecker et al.*, 2006], perform better in these cases at describing heterogeneity in hydraulic conductivity. Adapting the model to support different distributions increases the variety in field situations the model can be applied to.

A serious drawback of the model is the computation time for a single run. A single run for a stratified structure with 40 layers and the current settings takes 6 to 9 hours to finish (runtime for one realisation on the university cluster and personal laptop is the same). The computation time increases exponentially with an increase in layers, making this model not viable to run for highly refined aquifers. According to *Zech et al.* [2018], a stratified aquifer with a variance of 4 in log-hydraulic conductivity requires over a 100 layers to reach ergodic conditions. Running an ensemble for such a structure is estimated to take at least 20 hours per realisation. An adaptation of the model to support the use of multiple cores and graphical processing units (GPU) could speed up the calculations done by MODFLOW by several orders of magnitude [*Hughes and White*, 2013].

We did not study the effects of the spatial correlation length being different in the horizontal and vertical direction. Although the vertical correlation length is set much smaller than the horizontal one to match realistic field conditions, the ratio between both is chosen arbitrarily. Further model testing could determine how significant this choice is.

4 Summary & Conclusion

In this study, we developed a stochastic, numerical flow and transport model using MODFLOW 6 and FloPy (Python) to perform dipole tracer tests in a confined aquifer. First, a grid is constructed with refinements near the wells. The size of the grid scales with the distance between the injection and extraction wells. Next, random log-normal hydraulic conductivity field are generated in two configurations: stratified and fully heterogeneous. Ensembles were created for heterogeneous fields with different variances and correlation lengths and for stratified structures, varying in variance and the number of layers. For each ensemble a mean breakthrough curve is constructed at the extraction well from simulation results. We ran convergence tests on the ensembles, identifying the number of runs it takes to approach the ensemble mean. The model results are compared for identifying the impact of the stochastic parameters. Finally, the numerical model outcome was compared to the analytical solution for stratified aquifers presented by *Zech et al.* [2018]. The analytical solution and numerical solution do not match. One reason for this is that the analytical solution does not take dispersion into account. However, this alone is not enough to explain the discrepancy. All data and scrips are documented and published on GitHub.

Based on the results of the numerical model, we draw the following conclusions:

1. For a heterogeneous aquifer, both, the variance and the correlation length of the hydraulic conductivity impact the shape of the breakthrough curve at the extraction well.
   (a) An increase in variance leads to an earlier arrival of the tracer, due to the appearance of preferential flow. Transport happens predominantly through high conductivity lanes, bypassing low conductivity zones. However, the extremes in the low conductivity zones are also enhanced resulting in more tailing of the breakthrough curve.
   (b) A decrease in correlation length reduces the effects of an increase in variance. With shorter correlation lengths all flow paths are more likely to encounter both high and low conductivity zones. This means there is not a clear preferential flow path to bypass low conductivity zones, on average resulting in slower transport.
2. The BTC in fully heterogeneous media is different from the BTC observed in stratified or a homogeneous K-field. For a shorter correlation length (i.e. much smaller than the distance between the wells), the heterogeneous solution tends towards the solution for a homogeneous medium, while for a longer correlation length, the solution tends towards the solution for a stratified medium.

The numerical solution did not match the analytical solution from *Zech et al.* [2018]. This needs to be investigated further, as both solutions describe the same scenario (in case of a stratified subsurface).

Special thanks

I would like to thank Lukas van de Wiel for helping with getting the model to work on the Utrecht University cluster.

References

Bakker, M., et al., FloPy v3.6.0.dev0: U.s. geological survey software release, 2023.

Brindha, K., K. V. Neena Vaman, K. Srinivasan, M. Sathis Babu, and L. Elango, Identification of surface water-groundwater interaction by hydrogeochemical indicators and assessing its suitability for drinking and irrigational purposes in chennai, southern india, *Appl. Water Sci.*, *4*(2), 159–174, doi:https://doi.org/10.1007/s13201-013-0138-6, 2014.

Burchart-Korol, D., and P. Zawartka, Environmental life cycle assessment of septic tanks in urban wastewater system: A case study for poland, *Archives of Environmental Protection*, *45*(4), doi:10.24425/aep.2019.130243, 2019.

Gelhar, L., *Stochastic Subsurface Hydrology*, Prentice-Hall, 1993.

Gelhar, L. W., and L. S. Leonhart, Analysis of two-well tracer tests with a pulse input, *Rockwell International Corp*, 1982.

Grönwall, J., and K. Danert, Regarding groundwater and drinking water access through a human rights lens: Self-supply as a norm., *"MDPI"*, doi:https://doi.org/10.3390/w12020419, 2020.

Hagedorn, F., and M. Bundt, The age of preferential flow paths, *Geoderma*, *108*(1-2), 119–132, doi:https://doi.org/10.1016/S0016-7061(02)00129-5, 2002.

Hughes, J. D., and J. T. White, Use of general purpose graphics processing units with modflow, *Groundwater*, *51*(6), 833–846, doi:10.1111/gwat.12004, 2013.

Hussain, R., S. A. Khattak, L. Ali, S. Sattar, M. Zeb, and M. L. Hussain, Impacts of the linear flowing industrial wastewater on the groundwater quality and human health in swabi, pakistan, *Environ. Sci. Pollut. Res. Int.*, *28*(40), 56,741–56,757, doi:https://doi.org/10.1007/s11356-021-13842-5, 2021.

Kohlbecker, M. V., S. W. Wheatcraft, and M. M. Meerschaert, Heavy-tailed log hydraulic conductivity distributions imply heavy-tailed log velocity distributions, *Water resources research*, *42*(4), doi:https://doi.org/10.1029/2004WR003815, 2006.

Langevin, C., J. Hughes, E. Banta, A. Provost, R. Niswonger, and S. Panday, Modflow 6, the us geological survey modular hydrologic model, *US Geological Survey*, doi:https://doi.org/10.5066/F76Q1VQV, 2017.

Müller, S., and L. Schüler, Geostat-framework/gstools: Reverberating red (version v1. 1.0), doi:https://doi.org/10.5281/zenodo, 2019.

Muñoz-Carpena, R., C. M. Regalado, J. Álvarez-Benedi, and F. Bartoli, Field evaluation of the new philip-dunne permeameter for measuring saturated hydraulic conductivity, *Soil Science*, *167*(1), 9–24, 2002.

Nolan, B. T., and K. J. Hitt, Vulnerability of shallow groundwater and drinking-water wells to nitrate in the united states, *Environ. Sci. Technol.*, *40*(24), 7834–7840, doi:https://doi.org/10.1021/es060911u, 2006.

Shi, X., T. Lei, Y. Yan, and F. Zhang, Determination and impact factor analysis of hydrodynamic dispersion coefficient within a gravel layer using an electrolyte tracer method, *International Soil and Water Conservation Research*, *4*(2), 87–92, doi:https://doi.org/10.1016/j.iswcr.2016.05.001, 2016.

Van Genuchten, M. T., J. Šimunek, F. Leij, N. Toride, and M. Šejna, Stanmod: Model use, calibration, and validation, *Transactions of the ASABE*, *55*(4), 1355–1366, doi:10.13031/2013.42247, 2012.

Versteegh, J., and H. Dik, De staat van het drinkwater in nederland, *RIVM Rapport 2014-0137*, 2015.

Zech, A., C. D'Angelo, S. Attinger, and A. Fiori, Revisitation of the dipole tracer test for heterogeneous porous formations, *Adv. Water Resour.*, *115*, 198–206, doi:https://doi.org/10.1016/j.advwatres.2018.03.006, 2018.

```python
# -*- coding: utf-8 -*-
"""
Created on Thu Dec  7 10:00:25 2023

@author: Oscar Zeylmans
"""

#imports


# import os
import time
import flopy
import matplotlib.pyplot as plt
import numpy as np
import gstools as gs
from flopy.utils.gridgen import Gridgen
from shapely.geometry import Polygon


# Default settings. Do not touch!
DEF_settings = {

    # Grid settings:
    "delw"      : 1  , # distance between the wells in meter
    "nlay"      : 1  , # nr of layers
    "top"       : 0  , # position of top aquifer
    "Aqdepth"   : 8  , # Depth of the aquifer in m relative to top of the aquifer

    "ncol"      : 25 , # nr of columns and rows for coarse grid. Do not touch this!
    "h0"        : 0  , # initial hydraulic head in meter.

    # Subsurface properties
    "Kmean"     : np.log(1e-4) ,   # Mean value for hydraulic conductivity. hk = e^Kmean.
    "Kvar"      : 1  ,   # Variance in underlying normal distribution for hydraulic conductivity. ignored if 0
    "CorLen"    : 10 ,   # Corelation length in horizontal plain ($m$). Vertical correlation = 1/5 of horizontal
    "Layered"   : False, # Whether the subsurface is stratified (consists of homogeneous (random) layers)

    "porosity"  : 0.3,
    "sconc"     : 0.0,   # Background concentration
    "al"        : 0.5,   # Longitudinal dispersivity ($m$)
    "trpt"      : 1.0,   # Ratio of transverse to longitudinal dispersitivity


    # Well settings
    "qinjwell"  :  1e-4, # Discharge ($m3/s$)
    "qextwell"  : -1e-4,
    "Cwell"     : 1000,  # injection concentration in units/s

    # File location
    "sim_name"  : "DTTsim"  , # simulation name
    "ws"        : "./model" , # Folder to store all model files
    "mf6exe"    : "mf6"     , # Name of modflow 6 executable

    # Time settings
    "InjTime"       : 6,          # Duration of the injection ($hours$)
    "PostInjTime"   : 24 * 10,    # Measuring period after the injection has completed ($hours$)

    # Extra settings
    "length_units"  : "meters",
    "time_units"    : "seconds",

    "silent" : False,
    "seed"   : 20170519, # fixed seed for testing

    # Solver settings, dont touch unless you know what you're doing!
    "scheme" : "TVD", # other option: "UPSTREAM"
    "nouter" : 100,
    "ninner" : 300,
    "hclose" : 1e-6,
    "rclose" : 1e-6,
    "relax"  : 1.0,
    }


class DTTmodel(object):

    """
    A basic model of for a dipole tracer test in a heterogeneous medium.
    """
    def __init__(self, UseRandomSeed = True, flux_averaged = True, **settings):

        """

        """
        self.settings = DEF_settings.copy()
        self.settings.update(**settings)

        self.sim_name   = self.settings["sim_name"]
        self.ws         = self.settings["ws"]

        if UseRandomSeed:
            self.settings.update(seed = np.random.randint(1,2**31))

        self.delw = self.settings["delw"]
```

```
        self.ncol = self.settings["ncol"]
        self.nlay = self.settings["nlay"]
        self.botm  = np.linspace(self.settings["top"]-self.settings["Aqdepth"]/self.nlay,
                                 self.settings["top"]-self.settings["Aqdepth"], self.nlay)

        self.ath1 = self.settings["al"] * self.settings["trpt"]

        self.perlen  = [self.settings["InjTime"] * 3600, self.settings["PostInjTime"]*3600]
        self.nper    = len(self.perlen)
        self.nstp    = [self.settings["InjTime"], self.settings["PostInjTime"]]
        self.tsmult  = [1.0 for x in range(self.nper)]



        # Generate model grid

##########################################################################################################################################

        # The grid consist of a square of 25x25 coarse cells, each with sides 16 times the well distance.
        # The center cell of this coarse grid is refined into 64x64 cells with sides 0.25 times the well distance.
        # This refined grid is where likely all transport will take place.
        # A square of 2x2 times the well distance around the center is even further refined into cells of 0.0625 times the well
distance (1/16*delw).


        if not self.settings["silent"]:
            print("Building grid...")

        PolyTransport = 16*self.delw * np.asarray(   # The center cell of the coarse grid = 16x16 well distance
            [(int(self.ncol/2)   ,int(self.ncol/2)),
             (int(self.ncol/2)   ,int(self.ncol/2)+1),
             (int(self.ncol/2)+1 ,int(self.ncol/2)+1),
             (int(self.ncol/2)+1 ,int(self.ncol/2))])

        PolyFine = np.asarray(   # Refined grid of 2 by 2 well distances centered around the wells
            [(16*self.delw*self.ncol/2-self.delw, 16*self.delw*self.ncol/2-self.delw),
             (16*self.delw*self.ncol/2-self.delw, 16*self.delw*self.ncol/2+self.delw),
             (16*self.delw*self.ncol/2+self.delw, 16*self.delw*self.ncol/2+self.delw),
             (16*self.delw*self.ncol/2+self.delw, 16*self.delw*self.ncol/2-self.delw)])

        # Create a dummy model and regular grid to use as a base grid for gridgen
        name = "dummy"
        sim = flopy.mf6.MFSimulation(sim_name= name, sim_ws=self.ws, exe_name="mf6")
        self.gwf = flopy.mf6.ModflowGwf(sim, modelname=name)
        dis = flopy.mf6.ModflowGwfdis(self.gwf, nlay=self.nlay, nrow=self.ncol, ncol=self.ncol,
                                      delr=16*self.delw, delc=16*self.delw, top=self.settings["top"], botm=self.botm)

        # Create and build the gridgen model with a refined area in the middle
        PolyTrans = [Polygon(PolyTransport)]
        PolyWells  = [Polygon(PolyFine)]
        self.grid = Gridgen(self.gwf.modelgrid, model_ws=self.ws)
        self.grid.add_refinement_features(PolyTrans, "polygon", 6, range(self.nlay))
        self.grid.add_refinement_features(PolyWells, "polygon", 8, range(self.nlay))
        self.grid.build()

        gridprops_vg = self.grid.get_gridprops_vertexgrid()
        self.vgrid = flopy.discretization.VertexGrid(**gridprops_vg)

        # retrieve a dictionary of arguments to be passed
        # directly into the flopy disv constructor
        self.disv_gridprops = self.grid.get_gridprops_disv()
        self.disv_gridprops.keys()

        if not self.settings["silent"]:
            print("...Finished building grid")

        # Constant head boundary conditions

##########################################################################################################################################

        if not self.settings["silent"]:
            print("Setting boundaries...")

        self.chdspd = []
        for ly in range(self.nlay):
            for x in np.linspace(0.5*16*self.delw, (self.ncol-0.5)*16*self.delw, self.ncol, endpoint=True):
                for y in np.linspace(0.5*16*self.delw, (self.ncol-0.5)*16*self.delw, self.ncol, endpoint=True):
                    if x == 0.5*16*self.delw or x == (self.ncol-0.5)*16*self.delw or y == 0.5*16*self.delw or y ==
(self.ncol-0.5)*16*self.delw:
                        ra = self.grid.intersect([(x, y)], "point", 0)
                        ic = ra["nodenumber"][0]
                        if [(ly, ic), self.settings["h0"], 0.0] not in self.chdspd:
                            self.chdspd.append([(ly, ic), self.settings["h0"], 0.0])
        if not self.settings["silent"]:
            print("...Finished settings boundaries")


        # Hydraulic conductivity (K-field)

##########################################################################################################################################

        if not self.settings["silent"]:
            print("Setting up K-field...")

        nrCells = len(self.disv_gridprops["cell2d"])
```

```
        cellXY = self.grid.get_cellxy(nrCells)
        self.xPos = cellXY[:,0]
        self.yPos = cellXY[:,1]
        self.Kfield = np.ones((nrCells,self.nlay))
        self.Karray = np.ones((nrCells,self.nlay))
        botmExt = np.concatenate(([0],self.botm))

        if self.settings["Kvar"] == 0:
            self.hk = np.exp(self.settings["Kmean"]) * self.Karray

        elif self.settings["Layered"]:
            kval = np.random.lognormal(mean = self.settings["Kmean"],
                                       sigma = np.sqrt(self.settings["Kvar"]),
                                       size = self.nlay)
            self.hk = np.tile(kval, (nrCells,1))

        else:
            # print("Kvar =/= 0")
            for ly in range(self.nlay):
                depth = botmExt[ly]+botmExt[ly+1]
                z = depth*np.ones(nrCells)
                Field = gs.Gaussian(dim=3, var=self.settings["Kvar"],
len_scale=[self.settings["CorLen"],self.settings["CorLen"],self.settings["CorLen"]/5])
                srf = gs.SRF(Field, mean = self.settings["Kmean"], seed=self.settings["seed"])
                self.Kfield[:,ly] = srf.unstructured([self.xPos,self.yPos,z])
                self.Karray[:,ly] = srf.transform("lognormal")
            self.hk = self.Karray

        if not self.settings["silent"]:
            print("...Finished setting up K-field")


        # Pumping wells

#########################################################################################################################################################

        # Well Positions
        # Injecion well and extraction well are placed exactly 1 well distance apart in the middle of the center row.
        # The injection and pumping well actually consist out of 4 seperate wells to ensure the model is symmetric around the center
rows.
        if not self.settings["silent"]:
            print("Placing wells...")

        injwellXY = [ (self.ncol/2*16*self.delw - self.delw/2-self.delw/32              , self.ncol/2*16*self.delw),
                      (self.ncol/2*16*self.delw - self.delw/2-self.delw/16-self.delw/32 , self.ncol/2*16*self.delw),
                      (self.ncol/2*16*self.delw - self.delw/2-self.delw/32              , self.ncol/2*16*self.delw-self.delw/16),
                      (self.ncol/2*16*self.delw - self.delw/2-self.delw/16-self.delw/32 , self.ncol/2*16*self.delw-self.delw/16)]

        extwellXY = [ (self.ncol/2*16*self.delw + self.delw/2+self.delw/32              , self.ncol/2*16*self.delw),
                      (self.ncol/2*16*self.delw + self.delw/2+self.delw/16+self.delw/32 , self.ncol/2*16*self.delw),
                      (self.ncol/2*16*self.delw + self.delw/2+self.delw/32              , self.ncol/2*16*self.delw-self.delw/16),
                      (self.ncol/2*16*self.delw + self.delw/2+self.delw/16+self.delw/32 , self.ncol/2*16*self.delw-self.delw/16)]

        wellist_sp1 = []
        wellist_sp2 = []

        ki_ka = 1
        for ly in range(self.nlay):
            for xy in range(4):
                # if xy == 0:
                #     print(f"\n{injwellXY}")
                #     print(extwellXY)

                # Injection well settings
                raI = self.grid.intersect([injwellXY[xy]], "point", 0)
                icI = raI["nodenumber"][0]
                if flux_averaged:
                    ki_ka = self.hk[icI,ly] / np.mean(self.hk[icI,:])
                    print(ki_ka)

                wellist_sp1.append([(ly, icI) , self.settings["qinjwell"]/4/self.nlay*ki_ka , self.settings["Cwell"]])
                wellist_sp2.append([(ly, icI) , self.settings["qinjwell"]/4/self.nlay*ki_ka , 0])


                # # Extraction well settings
                raE = self.grid.intersect([extwellXY[xy]], "point", 0)
                icE = raE["nodenumber"][0]
                if flux_averaged:
                    ki_ka = self.hk[icE,ly] / np.mean(self.hk[icE,:])
                    print(f"{ki_ka}\n")
                wellist_sp1.append([(ly, icE) , self.settings["qextwell"]/4/self.nlay*ki_ka , 0])
                wellist_sp2.append([(ly, icE) , self.settings["qextwell"]/4/self.nlay*ki_ka , 0])

        self.spd_mf6 = {0:wellist_sp1, 1:wellist_sp2}

        if not self.settings["silent"]:
            print("...Finished placing wells")

    def writefiles(self):
        # MODFLOW 6
        gwfname = "gwf-" + self.sim_name

        self.sim = flopy.mf6.MFSimulation(
            sim_name=self.sim_name, sim_ws=self.ws, exe_name=self.settings["mf6exe"]
        )
```

```
        # Instantiating MODFLOW 6 time discretization
        tdis_rc = []
        for i in range(self.nper):
            tdis_rc.append((self.perlen[i], self.nstp[i], self.tsmult[i]))
        flopy.mf6.ModflowTdis(
            self.sim, nper=self.nper, perioddata=tdis_rc, time_units=self.settings["time_units"]
        )

        # Instantiating MODFLOW 6 groundwater flow model
        self.gwf = flopy.mf6.ModflowGwf(
            self.sim,
            modelname=gwfname,
            save_flows=True,
            model_nam_file=f"{gwfname}.nam",
        )

        # Instantiating MODFLOW 6 solver for flow model
        imsgwf = flopy.mf6.ModflowIms(
            self.sim,
            print_option="SUMMARY",
            outer_dvclose=self.settings["hclose"],
            outer_maximum=self.settings["nouter"],
            under_relaxation="NONE",
            inner_maximum=self.settings["ninner"],
            inner_dvclose=self.settings["hclose"],
            rcloserecord=self.settings["rclose"],
            linear_acceleration="BICGSTAB",
            scaling_method="NONE",
            reordering_method="NONE",
            relaxation_factor=self.settings["relax"],
            filename=f"{gwfname}.ims",
        )
        self.sim.register_ims_package(imsgwf, [self.gwf.name])

        # Instantiating MODFLOW 6 discretization package
        disv = flopy.mf6.ModflowGwfdisv(self.gwf,
                                        filename=f"{gwfname}.dis",
                                        **self.disv_gridprops)

        # Instantiating MODFLOW 6 node-property flow package
        flopy.mf6.ModflowGwfnpf(
            self.gwf,
            save_flows=False,
            xt3doptions=True,
            icelltype=0,
            k=self.hk,
            k33=self.hk,
            save_specific_discharge=True,
            filename=f"{gwfname}.npf",
        )

        # Instantiating MODFLOW 6 storage package (steady flow conditions, so no actual storage, using to print values in .lst file)
        flopy.mf6.ModflowGwfsto(self.gwf, ss=1.0e-05, filename=f"{gwfname}.sto")

        # Instantiating MODFLOW 6 initial conditions package for flow model
        flopy.mf6.ModflowGwfic(self.gwf, strt=self.settings["h0"], filename=f"{gwfname}.ic")

        # Instantiating MODFLOW 6 constant head package
        flopy.mf6.ModflowGwfchd(
            self.gwf,
            maxbound=len(self.chdspd),
            stress_period_data=self.chdspd,
            save_flows=False,
            auxiliary="CONCENTRATION",
            pname="CHD-1",
            filename=f"{gwfname}.chd",
        )

        # Instantiate the wel package
        flopy.mf6.ModflowGwfwel(
            self.gwf,
            print_input=True,
            print_flows=True,
            stress_period_data=self.spd_mf6,
            save_flows=True,
            auxiliary="CONCENTRATION",
            pname="WEL-1",
            filename=f"{gwfname}.wel",
        )

        # Instantiating MODFLOW 6 output control package for flow model
        flopy.mf6.ModflowGwfoc(
            self.gwf,
            head_filerecord=f"{gwfname}.hds",
            budget_filerecord=f"{gwfname}.bud",
            headprintrecord=[
                ("COLUMNS", 10, "WIDTH", 15, "DIGITS", 6, "GENERAL")
            ],
            saverecord=[("HEAD", "LAST"), ("BUDGET", "LAST")],
            printrecord=[("HEAD", "LAST"), ("BUDGET", "LAST")],
        )

        # Instantiating MODFLOW 6 groundwater transport package
        gwtname = "gwt_" + self.sim_name
```

```
        self.gwt = flopy.mf6.MFModel(
            self.sim,
            model_type="gwt6",
            modelname=gwtname,
            model_nam_file=f"{gwtname}.nam",
        )
        self.gwt.name_file.save_flows = True

        # create iterative model solution and register the gwt model with it
        imsgwt = flopy.mf6.ModflowIms(
            self.sim,
            print_option="SUMMARY",
            outer_dvclose=self.settings["hclose"],
            outer_maximum=self.settings["nouter"],
            under_relaxation="NONE",
            inner_maximum=self.settings["ninner"],
            inner_dvclose=self.settings["hclose"],
            rcloserecord=self.settings["rclose"],
            linear_acceleration="BICGSTAB",
            scaling_method="NONE",
            reordering_method="NONE",
            relaxation_factor=self.settings["relax"],
            filename=f"{gwtname}.ims",
        )
        self.sim.register_ims_package(imsgwt, [self.gwt.name])

        # Instantiating MODFLOW 6 transport discretization package
        flopy.mf6.ModflowGwtdisv(
            self.gwt,
            **self.disv_gridprops,
            filename=f"{gwtname}.dis",
        )

        # Instantiating MODFLOW 6 transport initial concentrations
        flopy.mf6.ModflowGwtic(self.gwt, strt=self.settings["sconc"], filename=f"{gwtname}.ic")

        # Instantiating MODFLOW 6 transport advection package
        flopy.mf6.ModflowGwtadv(self.gwt, scheme=self.settings["scheme"], filename=f"{gwtname}.adv")

        # Instantiating MODFLOW 6 transport dispersion package
        if self.settings["al"] != 0:
            flopy.mf6.ModflowGwtdsp(
                self.gwt,
                alh=self.settings["al"],
                ath1=self.ath1,
                filename=f"{gwtname}.dsp",
            )

        # Instantiating MODFLOW 6 transport mass storage package (formerly "reaction" package in MT3DMS)
        flopy.mf6.ModflowGwtmst(
            self.gwt,
            porosity=self.settings["porosity"],
            first_order_decay=False,
            decay=None,
            decay_sorbed=None,
            sorption=None,
            bulk_density=None,
            distcoef=None,
            filename=f"{gwtname}.mst",
        )

        # Instantiating MODFLOW 6 transport source-sink mixing package
        sourcerecarray = [("WEL-1", "AUX", "CONCENTRATION")]
        flopy.mf6.ModflowGwtssm(
            self.gwt, sources=sourcerecarray, filename=f"{gwtname}.ssm"
        )

        # Instantiating MODFLOW 6 transport output control package
        flopy.mf6.ModflowGwtoc(
            self.gwt,
            budget_filerecord=f"{gwtname}.cbc",
            concentration_filerecord=f"{gwtname}.ucn",
            concentrationprintrecord=[
                ("COLUMNS", 10, "WIDTH", 15, "DIGITS", 6, "GENERAL")
            ],
            saverecord=[("CONCENTRATION", "ALL"), ("BUDGET", "ALL")],
            printrecord=[("CONCENTRATION", "LAST"), ("BUDGET", "LAST")],
        )

        # Instantiating MODFLOW 6 flow-transport exchange mechanism
        flopy.mf6.ModflowGwfgwt(
            self.sim,
            exgtype="GWF6-GWT6",
            exgmnamea=gwfname,
            exgmnameb=gwtname,
            filename=f"{self.sim_name}.gwfgwt",
        )

        self.sim.write_simulation(silent=self.settings["silent"])



    def run_model(self):
        """
        Calls on modflow 6 executable and starts the run.
```

```
        """

        success, buff = self.sim.run_simulation(silent = self.settings["silent"])
        if not success:
            print(buff)


        self.headdata = self.gwf.output.head().get_data() # Retrieve hydraulic head data for last timestep
        self.conc_mf6 = self.sim.get_model(list(self.sim.model_names)[1]).output.concentration().get_alldata() # Retrieve
concentration data for all timesteps
        bud = self.gwf.output.budget() # Retrieve budget info (used for showing fluxes in graph)
        self.spdis = bud.get_data(text="DATA-SPDIS")[0]

        # Volume per cell over the whole column is determined by taking the cell area multiplied by the depth of the aquifer.
        # For every timestep the average concentration over the whole column is determined by taking the arithemic mean of the column.
        # Total mass per column is determined by multiplying the average concentration per column with the volume of the column,
corrected for porosity
        # Total mass in the system is summed. Total injected mass is known, thus extracted mass can be determined.
        CellVolume      = self.grid.get_area()[:np.shape(self.conc_mf6)[-1]]*self.settings["Aqdepth"] #m3 #volume per grid-column
        CellConcAv      = np.mean(np.squeeze(self.conc_mf6, axis = 2), axis = 1) # g/m3 # average concentration per grid-column
        CellMass        = CellConcAv * CellVolume * self.settings["porosity"]      # m3*g/m3=g # volume times concentration corrected
for porosity
        TotalMass       = np.sum(CellMass, axis = -1)

        TotalMassPump = np.zeros(sum(self.nstp))        # Creates numpy array for every timestep (per hour)
        TotalMassPump[:self.nstp[0]] = self.settings["Cwell"] * self.settings["qinjwell"] * 3600 # first 6 cells (injection period = 6
hours) are filled with injected mass
        TotalMassPump = np.cumsum(TotalMassPump) # Injected mass is summed, so TotalMassPump now contains the total amount of mass
injected
        self.BTC = TotalMassPump - TotalMass     # Total mass in the system per timestep is subtracted from the total mass injected.
        self.BTCnorm = self.BTC/TotalMassPump[-1] # total mass extracted as percentage of total mass injected


    def get_head(self):
        """
        Returns
        - headdata : hydraulic head distribution.

        """
        return(self.headdata)

    def get_conc(self):
        """
        Returns
        - conc_mf6 : concentration distribution per timestep.
        """

        return self.conc_mf6

    def get_flowbudget(self):
        """
        Returns
        - spdis    : groundwater flow budgets.

        """
        return self.spdis

    def get_BTC(self, norm = True):
        """
        Parameters
        ----------
        norm : Bool, optional
            If true returns normalised BTC (as fraction of total injected mass). The default is True.

        Returns
        -------
        ld numpy.array containing BTC data per timestep
        """
        if norm:
            return self.BTCnorm
        else:
            return self.BTC

    def plotgrid(self, dpi = 1000, lw = 0.01, savefig = False, figname = "Grid", **kwargs):
        """
        Plots the model grid.

        Parameters
        ----------
        dpi : int, optional
            Resolution of the generated plot. The default is 1000.
        lw : float, optional
            linewidth. The default is 0.01.
        **kwargs :
            ax, colors.  The remaining kwargs are passed into the
                the LineCollection constructor.)

        Returns
        -------
        None.

        """
        plt.figure(dpi = dpi)
        self.vgrid.plot(lw = lw, **kwargs)
        plt.title("Model Grid")
```

```
        if savefig:
            plt.savefig(f"{figname}.pdf")

    def plotKfield(self, layer = 0, logValue = True, dpi = 300, figsize = (6,6), levels = 50, savefig = False, figname = "Kfield",
**kwargs):
        """
        Plots the hydraulic conductivity distribution for a given layer.
        If the layer is homogeneous, no plot is made and the K-value is instead printed.

        Parameters
        ----------
        layer : int, optional
            The layer of which to plot the hydraulic conductivity. The default is 0.
        logValue : Bool, optional
            Whether to plot K or logK. The default is True.
        dpi : int, optional
            Resolution of the generated plot.. The default is 300.
        figsize : tuple, optional
            figure size. The default is (6,6).
        levels : int, optional
            Number of contours to divide K-values over. Higher values result in smoother plots. The default is 50.
        **kwargs :
            Remaining kwargs are passed on to matplotlib.pyplot.tricontourf().


        Returns
        -------
        Array with K-values for the given layer

        """
        plt.figure(dpi = dpi, figsize = figsize)

        if self.settings["Kvar"] != 0 and self.settings["Layered"] != True:
            if logValue:
                plt.tricontourf(self.xPos,self.yPos,self.Kfield[:,layer], levels = np.linspace(
                    np.min(self.Kfield[:,layer]),
                    np.max(self.Kfield[:,layer]),
                    levels),
                    **kwargs)
                plt.colorbar(format=lambda x, _: f"{x:,.2f}")
                plt.title("Hydraulic conductivity (log)")

            else:
                plt.tricontourf(self.xPos,self.yPos,self.hk[:,layer], levels = np.logspace(
                    self.settings["Kmean"] - 5*self.settings["Kvar"]**.5,
                    self.settings["Kmean"] + 5*self.settings["Kvar"]**.5,
                    levels), norm = "log",
                    **kwargs)
                plt.colorbar(format=lambda x, _: f"{x:,.2e}")
                plt.title("Hydraulic conductivity")
            plt.axis("equal")
            if savefig:
                plt.savefig(f"{figname}.pdf")

        elif self.settings["Layered"]:
            print(f"Layer {layer} is homogeneous with K-value = {self.hk[layer]}")

        else:
            print(f"Layer {layer} is homogeneous with K-value = {self.hk[layer]}")

        return self.Karray[:,layer]


    def plot_head(self, flow = True, grid = False, contours = False, levels = 10, layer = 0, linewidths = 1, colors =  "k",
                  figsize = (20,6), dpi = 300, fontsize_title_subplot = 12, fontsize_legend = 12, fontsize_labels = 12,
fontsize_ticks = 12, savefig = False, figname = "HeadDistr",
                  **kwargs):
        """

        Parameters
        ----------
        flow : Bool,
            If true will plot the flow around the wells as a vectorfield. The default is True.
        grid : Bool,
            If true will add gridlines to the plots. The default is False.
        contours : Bool,
            If true will add contourlines to the plot. The default is False.
        levels : int,
            The number of contourlines. Only does something if contours are enabled. The default is 10.
        layer : int,
            the layer of which to show the hydraulic head distribution. The default is 0.
        **kwargs :
            Remaining keyword arguments are passed to matplotlib.pyplot.pcolormesh

        Returns
        -------
        np-array containing the hydraulic head distribution for the given layer.

        """

        fig, axes = plt.subplots(1,3, figsize=figsize, constrained_layout=True, dpi = dpi)

        ax = axes[0]
        pmv = flopy.plot.PlotMapView(self.gwf, ax = ax)
        hd = pmv.plot_array(self.headdata[layer])
```

```
        if grid:
            pmv.plot_grid(colors="white", lw = 0.2)
        ax.set_title("Head distribution in the whole domain", fontsize = fontsize_title_subplot)
        ax.set_aspect("equal")
        ax.set_ylabel("y ($m$)", fontsize = fontsize_labels)
        ax.set_xlabel("x ($m$)", fontsize = fontsize_labels)
        ax.tick_params(axis = "both", labelsize = fontsize_ticks)

        ax = axes[1]
        pmv = flopy.plot.PlotMapView(self.gwf, ax = ax, extent = self.delw*np.asarray([190,210,190,210]))
        hd = pmv.plot_array(self.headdata[layer])
        if grid:
            pmv.plot_grid(colors="white", lw = 0.2)
        if contours:
            pmv.contour_array(self.headdata[layer], levels = np.linspace(np.min(self.headdata[layer]), np.max(self.headdata[layer]),
levels), **kwargs)
        ax.set_title("Head distribution in the refined grid", fontsize = fontsize_title_subplot)
        ax.set_ylabel("y ($m$)", fontsize = fontsize_labels)
        ax.set_xlabel("x ($m$)", fontsize = fontsize_labels)
        ax.set_aspect("equal")
        ax.tick_params(axis = "both", labelsize = fontsize_ticks)

        ax = axes[2]
        pmv = flopy.plot.PlotMapView(self.gwf, ax = ax, extent = self.delw*np.asarray([198.8,201.2,198.8,201.2]))
        hd = pmv.plot_array(self.headdata[layer])
        if grid:
            pmv.plot_grid(colors="white", lw = 0.2)
        if contours:
            pmv.contour_array(self.headdata[layer], levels = np.linspace(np.min(self.headdata[layer]), np.max(self.headdata[layer]),
levels), **kwargs)
        if flow:
            pmv.plot_vector(self.spdis["qx"], self.spdis["qy"], color="white")
        ax.set_title("Head distribution around the wells", fontsize = fontsize_title_subplot)
        ax.set_ylabel("y ($m$)", fontsize = fontsize_labels)
        ax.set_xlabel("x ($m$)", fontsize = fontsize_labels)
        ax.set_aspect("equal")
        ax.tick_params(axis = "both", labelsize = fontsize_ticks)
        plt.colorbar(hd, ax=axes[2])
        if savefig:
            plt.savefig(f"{figname}.pdf")

        return self.headdata[layer]

    def plot_conc(self):
        pass

    def plot_BTC(self, norm = True, dpi = 300, **kwargs):
        """
        Parameters
        ----------
        norm : Bool, optional
            Shows the BTC as fraction of the total injected mass. The default is True.
        **kwargs :
            Keyword arguments are passed on to matplotlib.pyplot.plot

        Returns
        -------
        None.

        """
        if norm:
            plt.figure(dpi = dpi)
            plt.plot(self.BTCnorm, **kwargs)
            plt.title("BTC at extraction well (normalised)")
            plt.ylabel("Mass ($-$)")
            plt.xlabel("Time ($hours$)")

        else:
            plt.figure(dpi = dpi)
            plt.plot(self.BTC, **kwargs)
            plt.title("BTC at extraction well")
            plt.ylabel("Mass ($g$)")
            plt.xlabel("Time ($hours$)")

    def BTC_toFile(self, filename = None, norm = True, **kwargs):
        """

        BTC data is saved as csv (ascii) file in the folder BTCoutput.
        If no such folder exists one is created in current workspace.


        Parameters
        ----------
        norm : Bool, optional
            If true writes normalised BTC data to file. The default is true.
        filename: str
        keyword arguments are passed to numpy.savetxt()
        Returns
        -------
        None.

        """
        if filename == None:
            filename = f"{self.sim_name}_BTC"
        if norm:
```

```
            np.savetxt(filename, self.BTCnorm, delimiter=",", **kwargs)
        else:
            np.savetxt(filename, self.BTC, delimiter=",", **kwargs)

if __name__ == "__main__":

    start_time = time.time()
    Q = 1e-4
    NewModel = DTTmodel(delw = 1, nlay = 1, Kvar = 0, PostInjTime = 24*1, Layered = False, InjTime = 6)

    # NewModel.plotgrid()
    # for i in range(5):
    #     NewModel.plotKfield(layer = i)

    NewModel.writefiles()

    start_run = time.time()

    NewModel.run_model()
    end_time = time.time()
    # kfield = NewModel.plotKfield(dpi = 500, figsize = (10,10))

    NewModel.plot_head(flow = False, contours=False, levels = 20, linewidths = 1,
                       colors =  "k", dpi = 500, grid = True, savefig = True)

    # NewModel.plot_BTC()

    # NewModel.BTC_toFile(norm = True, filename = "Homogeneous BTC",
    #                     header = ("Model name = "  + str(NewModel.sim_name) +
    #                               ", Welldist = "  + str(NewModel.delw) +
    #                               ", nlay = "      + str(NewModel.nlay) +
    #                               ", corlen = "    + str(NewModel.settings["CorLen"]) +
    #                               ", Kvar = "      + str(NewModel.settings["Kvar"]) +
    #                               ", Kmean = "     + str(NewModel.settings["Kmean"]) +
    #                               ", seed = "      + str(NewModel.settings["seed"])))

    # import Functions as fc
    # varlist = [0.01]#, 1, 2]#, 0.5, 1, 2, 4]
    # pdf1, cdf1, Tau = fc.TravelTime8(varlist, -30, 6, 1e4)
    # pdf2, cdf2, Tau = fc.TravelTime11(varlist, -30, 6, 1e4)
    # btc = NewModel.get_BTC()


    # n = NewModel.settings["porosity"] # Porosity used in the model
    # q = NewModel.settings["qinjwell"]/NewModel.settings["Aqdepth"]#/np.sqrt(2) # Discharge per unit depth in m2/s  (Q/L = m3/s/m)
    # a = NewModel.settings["delw"] # Distance between the wells (m)
    # TauToTime = np.pi * n * a**2 * Tau / q/ 3600 # Dimensionless Tau converted to real time in hours

    # plt.figure(dpi = 300)
    # for i in range(len(cdf1)):
    #     plt.plot(TauToTime[1:], cdf1[i], label = f"Analytical8_{varlist[i]}", linestyle = ":")
    #     plt.plot(TauToTime[1:], cdf2[i], label = f"Analytical11_{varlist[i]}", linestyle = ":")
    # plt.plot(btc, label = "Numerical")
    # plt.xlim(100, 2400)
    # # plt.ylim(0,.6)
    # plt.xscale("log")
    # plt.legend()
    # plt.show()


    # run_time = end_time - start_run
    # tot_run_time = end_time - start_time
    # print(round(run_time, 3), round(tot_run_time, 3))




    # bins = np.logspace(-5,1,100)
    # plt.figure(dpi = 300, figsize=(10,10))
    # hist = np.histogram(kfield, bins = bins, density=True)[0]
    # plt.plot(bins[1:], hist/100)
    # plt.xscale("log")
    # plt.title("Hydraulic conductivity distribution", fontsize = 18)
    # plt.xlabel("Hydraulic conductivity K $[m/s]$", fontsize = 16)
    # plt.ylabel("Probability", fontsize = 16)
    # plt.tick_params(axis = "both", labelsize = 16)
    # # plt.ylim(0,0.6)







    # nlay = 2
    # slurmID = 0

    # start_time = time.time()

    # Model = DTTmodel(delw = 10,
    #                  nlay = nlay,
    #                  layered = True,
    #                  PostInjTime = 24*10,
    #                  Kvar = 1,
    #                  sim_name = f"run_{slurmID}",
```

```
#                   ws = f"./model_{slurmID}")
# Model.writefiles()
# start_run_time = time.time()
# Model.run_model()

# end_time = time.time()

# modeltime = end_time - start_run_time
# totmodeltime = end_time - start_time

# Model.BTC_toFile(norm = True, filename = f"run_{slurmID}_lys{nlay}_oscar",
#                   header = ("Model name = "  + str(Model.sim_name) +
#                             ", Welldist = "  + str(Model.delw) +
#                             ", nlay = "      + str(Model.nlay) +
#                             ", Kvar = "      + str(Model.settings["Kvar"]) +
#                             ", Kmean = "     + str(Model.settings["Kmean"]) +
#                             ", modeltime = " + str(modeltime) +
#                             ", totaltime = " + str(totmodeltime)))


# for fileName in os.listdir(f"./model_{slurmID}"):
#     #Check file extension
#     if fileName.endswith('.cbc') or fileName.endswith('.ucn'):
#         # Remove File
#         os.remove(f"./model_{slurmID}/" + fileName)
```