Feflow to Modflow:

Groundwater flow and conservative contaminant transport in the Western Leibnitzer Field

Leander Wieser, BBSc.

Master‘s Thesis

for the Attainment of the Degree

Master of Science

at the University of Technology Graz

Supervisor: Univ.-Prof. Dr.rer.nat. Steffen Birk
(Professor of Hydrogeology, University of Graz)

and

Univ.-Doz. Dr. Johann Fank
Priv.-Doz. Dr. Hans Kupfersberger
(JR-AquaConSol)

August 2021

Abstract

Feflow and Modflow represent the most popular software packages for numerical groundwater modelling. Like Feflow, the most recent version of Modflow 6 enables contaminant transport simulation in addition to groundwater flow modeling. Apart from the methodological differences, Feflow offers some specific features that are not directly supported by Modflow. The variable leakage factors, which can be defined separately for effluent and influent conditions, and the interface manager, which enables a dynamic query of external model data, are to be mentioned in particular.

The literature shows that both modeling packages yield comparable results, when equivalent functions are used. However, it remains unclear to what extend also the direct translation of Feflow specific functions into the Modflow environment is possible and reasonable. To answer this question, the present work takes the attempt to directly transfer an existing Feflow model for groundwater flow and contaminant transport simulation to Modflow 6. Various methods are used to compensate the differences. Furthermore, an attempt is made to directly replicate the calculation methods of Feflow by executing Modflow in a loop, whereby the groundwater level can be queried per simulation step. The preliminary data processing, the setup of the Modflow model and the simulations execution is carried out with FloPy, a package in the Python environment. A calibration is performed with PEST and the Python package PyEMU.

The techniques used to compensate the groundwater level dependencies have yielded very similar results with respect to the flow model. The interaction with the rivers, and thus the leakage factors, have thereby shown to be a key point in the model translation. The temporal depth-variability of the recharge rate has a minor effect. The coarser discretization in Modflow leads to localized deviations, especially in marginal areas with high gradients. However, the emulation of the Feflow calculation method by looping Modflow has shown weaknesses and has not provided comparable results. For improved and more stable functionality, this method must be revised and optimized.

In the recently integrated groundwater transport model of Modflow 6, problems occurred with the input of coupled recharge rate and contaminant load. In combination with an error in the Modflow transport source code (input file size limitation), only limited results could be obtained.

This work reveals that the direct translation of Feflow specific functions into the Modflow environment is only partly possible. Especially for dynamic groundwater level queries, the currently available version of Modflow is less practicable. Therefore, the choice of the optimal software package depends on the research question as well as the characteristics of the aquifer.

Kurzfassung

Feflow und Modflow sind die meist genützten Softwarepakete für die numerische Grundwassermodellierung. Wie auch Feflow ermöglicht die neueste Version von Modflow 6 neben der Strömungsmodellierung auch die Simulation von Stofftransport. Abgesehen von den methodischen Unterschieden bietet Feflow einige spezifische Funktionen, die von Modflow nicht direkt unterstützt werden. Hier sind insbesondere die variablen Leakagefaktoren, die getrennt für influente und effluente Bedingungen definiert werden können und der Interface Manger, der eine dynamische Abfrage externer Modelldaten ermöglicht, zu nennen.

Bisherigen Erkentnissen zufolge führen beide Softwarepakete bei Verwendung gleichwertiger Funktionen zu ähnlichen Ergebnissen. Es bleibt jedoch unklar, inwieweit auch die direkte Übersetzung von Feflow-spezifischen Funktionen in die Modflow-Umgebung möglich und sinnvoll ist. Um diese Frage zu beantworten, wird in der vorliegenden Arbeit der Versuch unternommen, ein bestehendes Feflow-Modell zur Simulation von Grundwasserströmung und Schadstofftransport direkt auf Modflow 6 zu übertragen. Dabei werden verschiedene Methoden eingesetzt, um die Unterschiede zwischen den Programmen zu kompensieren. Außerdem wird versucht, die Berechnungsmethoden von Feflow direkt nachzubilden, indem Modflow in einer Schleife ausgeführt wird, wobei der Grundwasserstand pro Simulationsschritt abgefragt werden kann. Die vorausgehende Datenverarbeitung, der Aufbau des Modflow-Modells und die Ausführung der Simulationen erfolgt mit FloPy, einem Python-Paket. Eine Kalibrierung wird mit PEST und dem Python-Paket PyEMU durchgeführt.

Die angewandten Techniken zur Kompensation der Grundwasserstandsabhängigkeiten haben zu sehr ähnlichen Ergebnissen wie in jenem des Feflow-Strömungsmodell geführt. Die Interaktion mit den Flüssen, und damit die Leakagefaktoren, haben sich dabei als ein zentraler Punkt in der Modellübersetzung erwiesen. Die zeitliche Tiefenvariabilität der Neubildungsrate hat einen geringen Einfluss. Die gröbere Diskretisierung in Modflow führt zu lokalen Abweichungen, insbesondere in Randbereichen mit hohen Gradienten. Die Nachbildung der Feflow-Berechnungsmethode, bei der Modflow in einer Schleife ausgeführt wird, hat jedoch Schwächen gezeigt und keine vergleichbaren Ergebnisse geliefert. Für eine verbesserte und stabilere Funktionalität muss diese Methode überarbeitet und optimiert werden.

Im kürzlich integrierten Grundwassertransportmodell von Modflow 6 traten Probleme bei der Eingabe der gekoppelten Neubildungsrate und Schadstoffeingabe auf. In Kombination mit einem Fehler im Modflow-Transport-Quellcode (Größenbegrenzung der Eingabedatei) konnten nur limitierte Ergebnisse erzielt werden.

Diese Arbeit zeigt, dass die direkte Übertragung von Feflow auf Modflow nur teilweise praktikabel ist. Die Wahl des optimalen Softwarepakets hängt insbesondere von der Forschungsfrage als auch von den Eigenschaften des Untersuchungsgebiets bzw. des Aquifers ab.

Acknowledgements

I would like to take this opportunity to thank all those who have contributed to the creation of this work and supported me along the way.

First I would like to thank my supervisor at the University of Graz, Prof. Steffen Birk, for the expertise he provided, the support he gave me in terms of content and his scientific accuracy.

A big thanks goes to the JR-AquaConSol, and in particular Dr. Johann Fank and Dr. Hans Kupfersberger, for giving me the opportunity to write this thesis as part of the team and for supporting me with the infrastructure and know-how. Further I would like to thank my colleagues for their help, especially Janja and Johannes.

Also a thank you to the Modflow community of https://groups.google.com/g/modflow, where any questions are answered with a great helpfulness and a lot of professional knowledge.

Thanks also to you, Chri, for reviewing and for the mental support during the master thesis and the whole study.

None of this could have happened without my parents, who gave me the courage to open my mind and embrace the world. Thank you for your support!

Most of all, thank you, Lucia, for supporting and helping me in every way during this work.

Contents

List of Abbreviations

BC	Boundary Condition.
CHB	Constant Head Package (Modflow).
CNC	Constant Concentration Package (Modflow).
CVFD	control-volume Finite Difference.
DRN	Drainage Package (Modflow).
FD	Finite-Differences Method.
FE	Finite-Elements Method.
FF	Feflow.
GUI	Graphical User Interface.
GWF	Groundwater Flow Model.
GWL	Groundwater level.
GWT	Groundwater Transport Model.
IC	Initial Condition.
IMS	Iterative Model Solution Package (Modflow).
MF	Modflow.
MR	modelrun.
OBS	Observation.
Q50	50% quantile ≡ median.
RCH	Recharge Package (Modflow).
RIV	River Package (Modflow).
RMSE	Root Mean Square Error.
SHP	Shape File (.shp).
SP	Stress Period.
SSM	Source Sink Mixing Package (Modflow).
TDT	Time-Depth Table.

TS	Time Series.
USGS	United States Geological Survey.
WEL	Well Package (Modflow).

List of Figures

List of Tables

1. Introduction

1.1. Motivation

Numerical modeling of groundwater flow and contaminant transport is an important tool to answer questions concerning the dynamics, supply, quality or future availability of the resource water. Specifically, it can be used to assess groundwater levels or pollutant dispersal is space and time, which would otherwise be only possible through invasive interventions, if at all. A conceptual, systemic model serves as a basis, where characteristics of the aquifer, sources and sinks of water (or pollutants) are defined. The numerical modeling is based on physical and hydraulic principles and is performed over the spatially and temporally resolved model area.

There are different approaches for solving numerical models. The most common software packages are Feflow and Modflow. However, they differ in their solution method: Feflow is based on the finite-element method and therefore works node-centered, while Modflow (version 6) relies on control-volume finite differences and is therefore cell-centered.

Wang and Anderson [1982] have shown that from a purely mathematical point of view both methods lead to exactly the same results. The dispersion of a contaminant plume was simulated by Chapman et al. [2012] in an experimental laboratory setting and compared with the results of the numerical modeling programs Feflow v6.0 and Modflow 2005 in combination with MT3DMS. The models are constructed with the typical model grid in each case - triangular in the case of Feflow and a structured one in the case of Modflow. They show that the results of the flow and the transport simulation hardly differ and reproduce results of the experiment relatively accurately, whereby an adequate spatial and temporal discretization is required. A performance evaluation, based on a real aquifer, of the same software packages (Feflow and Modflow 2005 with MT3DMS), was carried out by Matiatos et al. [2019]. Almost identical results were obtained with a calibrated modelrun, however Feflow provided a slightly better match with observed values in respect of groundwater flow and transport simulation.

The papers mentioned have in common that they compare functions or packages, that are equally available in both software packages. An attempt that has not been carried out so far is a translation of Feflow specific functions into the Modflow environment. In particular, the interface manager, which enables an exchange with external models and thus a groundwater level-dependent definition of the recharge rate, as well as the direction-dependent leakage factor (inflow and outflow factor) must be mentioned.

The question that arises is to what extent the direct translation of the Feflow specific functions into a Modflow setting is possible and reasonable in terms of practicability. In this work, an existing Feflow model is rebuilt with the open-source code Modflow 6 [Hughes et al., 2017], developed and released by the United States Geological Survey (USGS). The commonalities and differences in both software packages are discussed, possibilities and limitations are investigated. The focus lies thus less on a conceptual model construction than on a direct transfer of existing parameters.

The existing Feflow model concerns the Western Leibnitzer Field and was generated for the SI-

MUR-AT project by JR-AquaConSol with a simulation period running from 1993 until 2018 [Mach et al., 2019]. The groundwater recharge and the Nitrate leaching in the unsaturated zone was calculated with STOTRASIM [Feichtinger, 1998].

1.2. General Approach

In principle, it is attempted to take over the parameters and transfer them directly from Feflow to the Modflow model without intervention. Three fundamental differences between the models and the underlying software make a direct transfer problematic and require a methodical approach:

- Discretization: vertex-based (Feflow) vs. structured square grid (current Modflow model[1]).
- Transfer-in and transfer-out rates in Feflow vs. one conductance in Modflow
- Interface for time-depth dependent recharge in Feflow

For these reasons, different approaches were chosen in order to reproduce the results from Feflow as accurately as possible. Table 1.1 lists a compilation of used methods and parameters.

Table 1.1.: Modelruns and parameters. GWL...groundwater level.

modelrun	**leakage**	**recharge**	**run mode**
MR 1	rivers as wells	mean GWL	classic
MR 2	manual selection	interpolated	classic
↳ MR 2a	calibration	interpolated	classic
MR 3	GWL query	GWL query	loop

1.3. Software and Graphical User Interface

Initially it was intended to use the commercial Graphical User Interface (GUI) Visual Modflow Flex from Waterloo Hydrogeologic. The direct parameter transfer in this work requires a high level of data processing. Visual Modflow Flex is strongly oriented towards conceptual modeling and not very suitable in handling and transposing large amounts of transient data and external time-series. Therefore, the author decided to switch to FloPy [Bakker et al., 2021], a Python package for creating, running, and post-processing Modflow-based models. The use of this package allows to benefit from the data processing possibilities of Python.

1.4. Project Area

The model area is located in southern Styria (Austria), approximately 30 km south of the city Graz. The river Mur divides the basin "Leibnitzer Feld" in a western and an eastern part; the western part with an area of around 40 km^2 is the subject of this thesis (Fig. 1.1). Along the river Mur, forming the eastern project edge, there are three barrages in the project area, partly with accompanying drainages. The western project edge is defined by the rivers Lassnitz and Sulm, with the Lassnitz flowing into the Sulm west of the town of Leibnitz.
The longitudinal extension stretches over about 13.5 km, from the southern foot of the Buchkogel

[1]Transport simulations were not supported for unstructured grids at the time of defining the object of study (Transport model was not included in Modflow 6 and MT3D does not support unstructured grids; see 2.3.2)

to the confluence of the Sulm and the Mur in the south; the maximum width of about 5.4 km is reached in the middle third. In general, the project area is relatively flat, with a slightly overall dipping to the south. The maximum natural elevation of the terrain surface is about 296 m in the north and 259 m in the south [Digitaler Atlas Steiermark, 2021]. Fank et al. [1993] and Mach et al. [2019] give a detailed description of land use, soil types, morphology and hydrology in the project area.

Geological Overview:
The geological conditions of the Leibnitz Field are characterized by accumulation of fluvioglacial and fluviatile sediments over an erosional relief created from predominantly Neogene deposits.

Along the western valley margin in the project area, Paleozoic basement is partly found, built up mainly of greenschists and phyllites.
Stratigraphically above are the Neogene deposits. These are mainly composed of blue-grey clay marls and sand layers and act as groundwater barrier for the shallow groundwater body. In the western marginal areas of the project area, siliciclastic sediments are also deposited here, indicating the shallow marine area of the Para-Tethys in the Neogene here. Furthermore, Leithakalke (Buchkogel) are found in the marginal areas [Fank et al., 1993].

The Quaternary cover can be differentiated into auzones and worm-age low terraces.
The auzones, situated along the Mur, show a structure of weakly silty, sandy gravels above the pre-Quaternary base. Partly this stratum is covered by a 1.5 to 3 m thick silt/clay layer [Fank et al., 1993].
The low terraces are divided into two sub-terraces and are mainly composed of low silty, sandy gravels with stones. The coarse-grained components are consistently well rounded and of crystalline and calcareous origin.

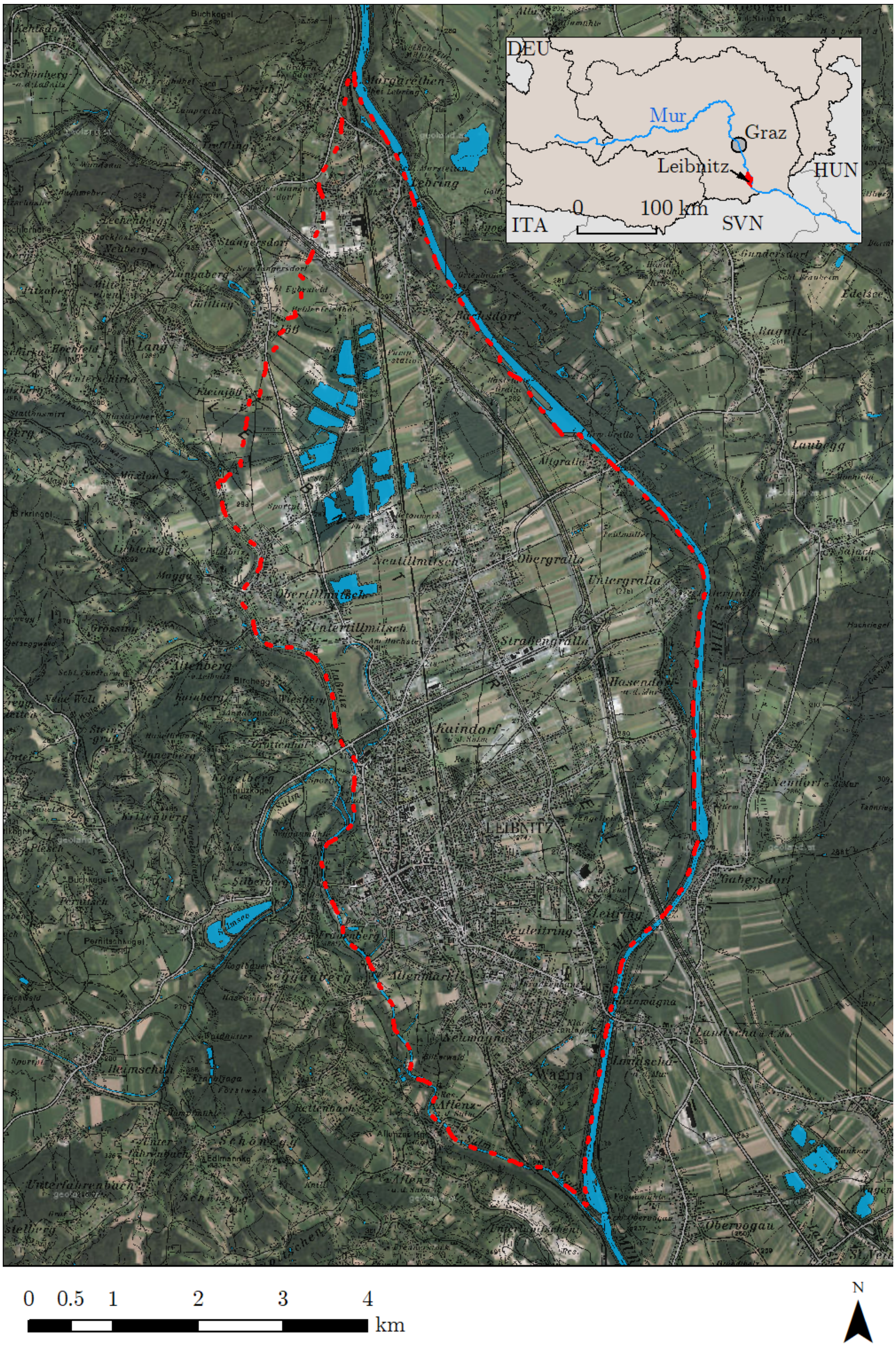


Figure 1.1.: Orthofoto with overlayed map of the model area: Western Leibnitzer Field (red). blue: lakes and rivers.

2. Fundamentals

2.1. Numerical Methods

2.1.1. Groundwater Flow

Compared to other methods (e.g. analytical models or analytical element models), numerical groundwater modeling offers advantages in spatial and temporal resolution or discretization and allows heterogeneous and anisotropic conditions to be represented. The solution is thereby not continuous, but calculated at discrete points in space and for specific units of time [Anderson et al., 2015]. In numerical groundwater modeling, three approaches in particular are used: finite-differences (FD), the finite-element (FE) and control volume finite-differences (CVFD) method. All methods are based on Darcy's law and the derived general governing differential equation for tree-dimensional transient groundwater flow in heterogeneous and anisotropic conditions ("groundwater equation"):

$$\frac{\delta}{\delta x}\left(K_{xx}\cdot\frac{\delta h}{\delta x}\right)+\frac{\delta}{\delta y}\left(K_{yy}\cdot\frac{\delta h}{\delta y}\right)+\frac{\delta}{\delta z}\left(K_{zz}\cdot\frac{\delta h}{\delta z}\right)+W=S_s\frac{\delta}{\delta t} \tag{2.1}$$

where:

K_{xx}, K_{yy}, K_{zz}	= hydraulic conductivities along x, y and z axis [LT^{-1}]
h	= piezometric head [L]
W	= volume flux per unit volume (source or sink of water) [T^{-1}]
S_s	= specific storage of the porous media [-]
t	= time [T]

The methods mentioned use different solution approaches to calculate the hydraulic head from the groundwater equation, or in the case of the CVFD method the groundwater equation is extended by the continuity equation [Anderson et al., 2015; Dehotin et al., 2011]. All approaches calculate in a direct or an iterative way - usually a combination of both is applied, or a specific solution can be selected by the user [Anderson et al., 2015].

Further, all methods use the same definitions of Boundary Condition (BC), which can be divided into the following three types: (1st) specified head (Dirichlet), (2nd) specified flow (Neumann) and (3rd) head-dependent (Robin[1]) BC. In general they represent locations in the model domain, where water flows into or out of the model due to external factors (sources and sinks).

FD is a straight forward method, that works cell-centered and hydraulic heads are calculated as an average (constant) value per cell in a structured, rectangular grid. Modflow up to version 5 is the most widely distributed and researched software package based on this method.

[1]Feflow uses the designation "Cauchy" for this type of BC. The USGS calls it "Robin"-BC. Both terms describe the same (third) type of a BC. Jazayeri and Werner [2019] point out the confusing nomenclature and refer to the consistent designation in mathematical literature, where the third BC is named after Victor Gustave Robin.

The CVFD method is an extension of FD due to the possibility of unstructured grids. Here, the intercell exchange is described with the help of conductance - a measure which includes the flow area of connected cell-surfaces in the hydraulic conductivity [Dehotin et al., 2011; Panday et al., 2013]. The model remains cell-centred and calculated values are constant per cell. With Modflow-USG (Unstructured Grid), the CVFD-code has gained popularity and has become a standard with the launch of Modflow version 6.

A FE model is constructed from nodes, which allows the model grid to be of flexible shapes (typically triangular). Hydraulic heads are calculated as continuous values along the nodes and (linearly) interpolated within the cells [Wang and Anderson, 1982]. Feflow is the common software tool used for FE.

In the final SI-MUR-AT report Mach et al., 2019 summarize the FE in general as follows: *"The finite element method represents a discrete way of describing groundwater flow that starts directly from the physical conditions of the groundwater flow. The finite element method makes it possible to determine the potential heights in the nodal points via the physical description in the interior and at the edges of the elements. In the discretized flow plane, the shape of the potential surface depends on the potential heights at the nodes of the mesh. It is assumed that the potential height within and on the edges of a triangular element changes linearly in each direction. This results in a linear system of equations.*
The fulfillment of the continuity condition depends on the degree of discretization. The smaller the triangular elements are, the better the exact course of the potential distribution can be approximated. A second dependence for the fulfillment of the continuity is given by the form of the triangle elements. An equilateral triangle represents the best shape of the elements."

2.1.2. Transport

The basic form of the solute transport equation [Konikow et al., 1996] is shown below:

$$\frac{\delta C}{\delta t} + \frac{V_i}{R_f}\frac{\delta C}{\delta x_i} - \frac{1}{\varepsilon R_f}\frac{\delta}{\delta x_i}\left(\varepsilon D_{ij} + \frac{\delta C}{\delta x_j}\right) - \frac{\sum[W(C' - C)]}{\varepsilon R_f} + \lambda C = 0 \tag{2.2}$$

where:

$C\,(C')$	= volumetric concentration (in source / sink) [M/L^3]
ε	= effective porosity [-]
V	= vector of interstitial fluid velocity components [LT^{-1}]
D	= second-rank tensor of dispersion coefficients [L^{2}T^{-1}]
W	= volumetric fluid sink (W<0) or fluid source (W>0) rate per unit volume of aquifer [T^{-1}]
λ	= dacay rate [T^{-1}]
t	= time [T]
R_f	= retardation coefficient [-]
x_i	= Cartesian coordinates [L]

Transport simulations are in any case coupled to the flow model - the solution approach works according to the same principle, but additional hydrodynamic processes are considered in the equation: advection, dispersion, diffusion, retardation and decay [Zheng and Wang, 1999].

Mach et al., 2019 describe FE transport modelling as follows: *"[...] Lagrangian and Eulerian methods are primarily used. In order to minimize the numerical error caused by the approximations in the solution of the transport equation, the Peclet and Courant criteria must be observed,* *which specify upper limits for the spatial and temporal discretization, i.e. the calculation set must not be too coarse and the time step must not be too long [Kinzelbach, 1987]."*

2.2. Feflow

Feflow (FF) is a software package for modeling fluid flow and transport of dissolved constituents and/or heat transport processes in porous or fractured media, in different dimensional representations and for variably saturated conditions [Diersch, 2014]. It is distributed by DHI (since 2007; `https://www.mikepoweredbydhi.com/products/feflow`), based on the finite element method and is usually operated via the integrated GUI.

Mach et al., 2019 summarize the features and advantages of Feflow in the final SI-MUR-AT report as follows: *"Feflow offers a variety of pre- and post-processing functionalities while being compatible with numerous data formats for importing and exporting data from and to external sources. Especially for the solution algorithms for the finite element method Feflow offers a high number of alternative choices, so that specific aspects of the problem can be addressed."*
Trefry and Chris, 2007 rate Feflow mainly positively in their test report, among other things because of the practical GUI functionality, the integrated PEST interface, the support and *"a full developer application programming interface that allows users to add custom code modules directly into the FEFLOW simulator."*. It is necessary to mention that Feflow v 5.3 was reviewed and the software was distributed by the german company WASY at that time.

2.3. Modflow

2.3.1. General

Modflow 6 (MF) is an object-oriented program and framework developed to provide a platform for supporting multiple local-scale groundwater models. It presently contains two types of hydrologic models, the Groundwater Flow (GWF) Model and the Groundwater Transport (GWT) Model. The GWF 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. Users can define the model grid using:

- a regular Modflow grid consisting of layers, rows, and columns,
- a layered grid defined by (x, y) vertex pairs, or
- a general unstructured grid based on concepts developed for Modflow-USG.

The source code is a free public domain software, written primarily in FORTRAN. Since the first developments in the 1980s, the USGS has released six core versions. The most current version, Modflow 6, was released in 2017.

Unlike Feflow, Modflow itself is a pure code for solving the equation systems - there is no direct or integrated GUI. However, there are several programmes that are based on the Modflow code and offer an integrated GUI. Among the most popular are Modelmuse (USGS), Visual Modflow Flex (Waterloo Hydrogeologic) and Processing Modflow (Wen-Hsing Chiang and Wolfgang Kinzelbach). In addition, the USGS has developed FloPy, a Python package for creating, running, and post-processing Modflow-based models.

2.3.2. Releases and Developement

In the course of this thesis (starting 01.07.2020), several updates of the current Modflow (MF) version 6 were released. The following list shows their history and the most important innovations for the present work (quoted from the corresponding release notes on `github.com/MODFLOW-USGS/modflow6/releases`):

- **MF 6 Version 6.1.1** *Released on 12.07.2020*
- **MF 6 Version 6.2.0** *Released on 22.10.2020*
  - A new Groundwater Transport (GWT) Model is introduced in this release as a way to simulate the fate and transport of a dissolved solute. Extensive testing of the GWT Model has been performed but changes to the code and input may be required in response to user needs and testing.
- **MF 6 Version 6.2.1** *Released on 18.02.2021*
  - The Source and Sink Mixing (SSM) Package for the Groundwater Transport Model was modified to include an alternative option for the concentration value assigned to sinks. A new AUXMIXED option was added to represent evaporation-like sinks where the solute or a portion of the solute may be left behind. The AUXMIXED option provides an alternative method for determining the groundwater sink concentration. If the cell concentration is larger than the user-specified sink concentration, then the concentration of the sink will be assigned as the specified concentration. Alternatively, if the specified concentration is larger than the cell concentration, then water will be withdrawn at the cell concentration. Thus, the AUXMIXED option is designed to work with the Evapotranspiration and Recharge packages where water may be withdrawn at a concentration that is less than the cell concentration.

2.3.3. Structure

Figure 2.1 shows the structure and components for a single Groundwater Flow Model (GWF) in Modflow. Accordingly, a model is initialised with FloPy. The simulation consists of a Timing Module, a Numerical Solution, and a GWF Model. Beneath the GWF Model are individual packages, which describe the hydrogeologic processes that are simulated. The only packages that do not fit into the hydrologic categories are the "Observation" and the "Output Control" Packages, which manage the printing and saving of GWF Model results to output files [Langevin et al., 2021].

With the release of MF 6.2.0 the possibility to initiate a further Groundwater Transport Model (GWT) within the same simulation and numerical solution instance was created. A so-called "Model Exchange object" couples two models, and is therefore specified at the simulation level [MODFLOW Development Team Revision 27bb36e1, 2020]. A transport model can also be run independently. To do this, all inter-cellular flows and budgets must be exported from the GWF model and transferred to the transport model.

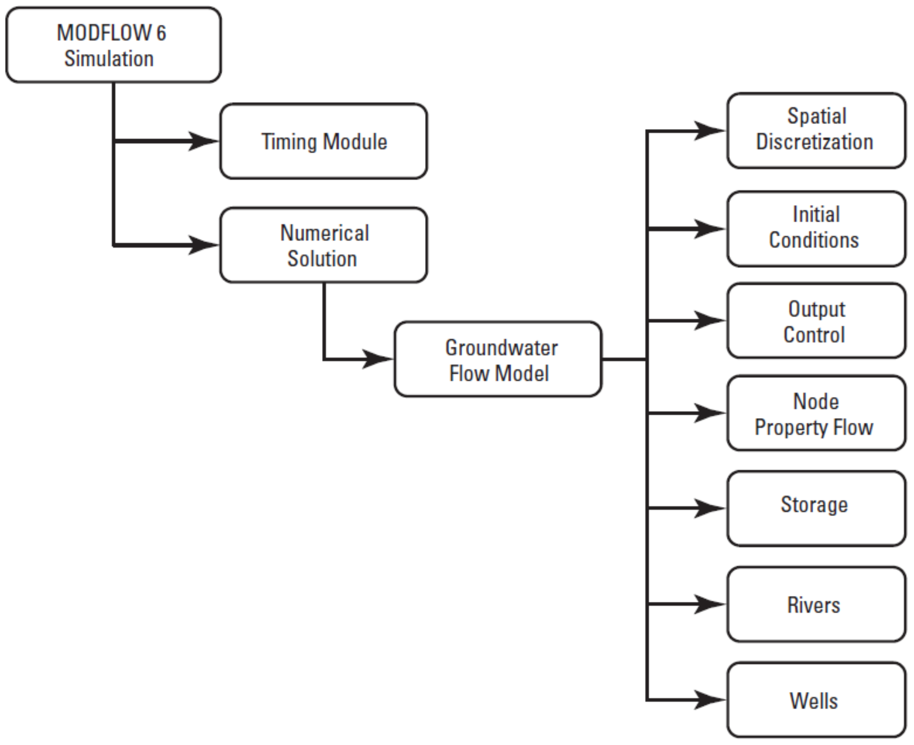


Figure 2.1.: Diagram showing the Modflow 6 components for a simulation. Langevin et al., 2021.

2.4. Main discrepancies

Wang and Anderson [1982] have demonstrated that FD and FE lead to the same results, provided that the cell sizes or node distances are "sufficiently" small, although there are different concepts behind it. They further have shown, that both methods yield the same set of algebraic equations through the solution process [Anderson et al., 2015].

Although both software packages use the same definitions of boundary conditions (BC), the input differs: Feflow basically follows the three base types of BC, whereby all are defined in a linear or areal context. In order to realize a point BC, i.e. wells, a fourth type is introduced [DHI-WASY GmbH, n.d.]. Table 2.1 lists the options in Feflow according BCs. FF allows further settings for each BC.

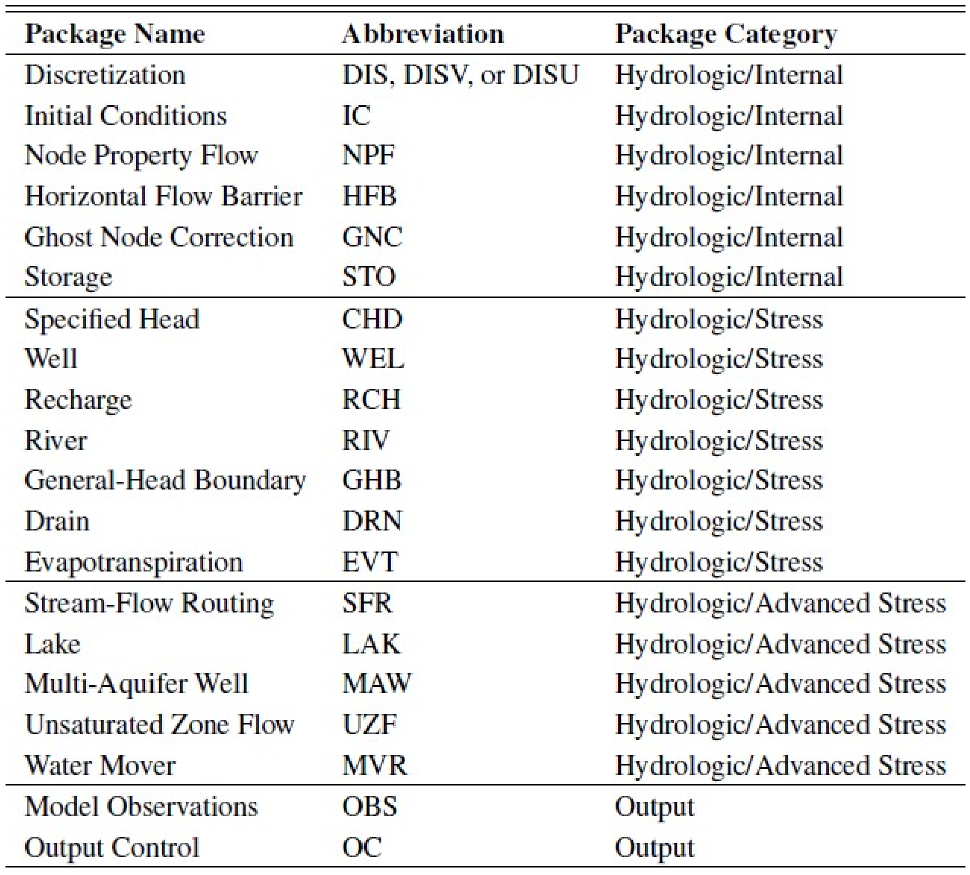
Modflow relies on packages that are based on the respective boundary conditions (Tab. 2.2). For example the river, drain, stream-flow routing and lake package work with a head-dependent flux of a third-type BC (Robin), but differ in functionality and input parameters. Modflow is stringent in setting up the packages - all parameters must be defined at all times. For a detailed describtion see Langevin et al. [2021].

The main discrepancies (especially concerning the current project) are listed in table 2.3. The typical grids of the finite-element and the finite differences method are shown in figure 2.2.

Table 2.1.: Boundary Conditions available in Feflow [DHI-WASY GmbH, n.d.].

BC	Short Description	Examples
Hydraulic-head BC	Fixed hydraulic head (1st kind/Dirichlet boundary condition).	Well-known groundwater level at boundary, Surface water body perfectly connected to the aquifer
Fluid-flux BC	Fixed flux (Darcy flux) across a model boundary (2nd kind/Neumann boundary condition).	Lateral inflow into the aquifer from a slope
Fluid-transfer BC	Fixed reference water level with additional transfer rate (3rd kind/Cauchy boundary condition)	River/lake with clogging layer, Partly clogged drain
Well BC	Fixed abstraction/infiltration at a single node or along a well screen.	Pumping/infiltration well

Table 2.2.: Modflow: List of packages available for use with the Groundwater Flow Model Langevin et al., 2021.



Package Name	Abbreviation	Package Category
Discretization	DIS, DISV, or DISU	Hydrologic/Internal
Initial Conditions	IC	Hydrologic/Internal
Node Property Flow	NPF	Hydrologic/Internal
Horizontal Flow Barrier	HFB	Hydrologic/Internal
Ghost Node Correction	GNC	Hydrologic/Internal
Storage	STO	Hydrologic/Internal
Specified Head	CHD	Hydrologic/Stress
Well	WEL	Hydrologic/Stress
Recharge	RCH	Hydrologic/Stress
River	RIV	Hydrologic/Stress
General-Head Boundary	GHB	Hydrologic/Stress
Drain	DRN	Hydrologic/Stress
Evapotranspiration	EVT	Hydrologic/Stress
Stream-Flow Routing	SFR	Hydrologic/Advanced Stress
Lake	LAK	Hydrologic/Advanced Stress
Multi-Aquifer Well	MAW	Hydrologic/Advanced Stress
Unsaturated Zone Flow	UZF	Hydrologic/Advanced Stress
Water Mover	MVR	Hydrologic/Advanced Stress
Model Observations	OBS	Output
Output Control	OC	Output

Table 2.3.: Main discrepancies: Feflow vs. Modflow 6

	Feflow	**Modflow 6**
general		
numerical solution:	finite elements (FE)	control-volume finite difference (CVFD)
leakage:	user input: transfer rate (T) $T = \frac{k_n}{d_n}$ > internal multiplication with peripheral cell edges > differentiation between inflow and outflow leakage	user input: conductance (C) $C = \frac{l_n \cdot w_n \cdot k_n}{d_n}$ > only one leakage factor available
concerning current project		
discretization:	unregular, triangular grid	structured square-grid
interface:	integrated interface for simultaneous groundwater level query $\rightarrow$ head-dependent recharge	-

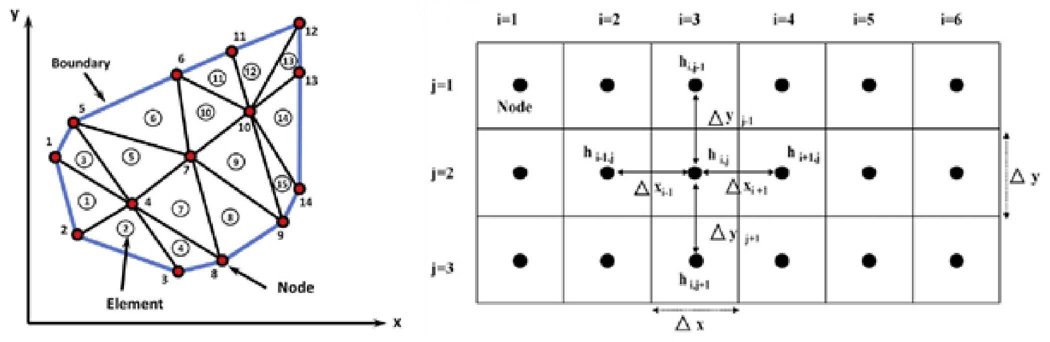


Figure 2.2.: Characteristic FE grid. Right: Currently applied finite difference (FD) grid. Anderson et al., 2015.

(k_n...hydraulic conductivity of riverbed ("clogged layer"), d_n...riverbed thickness, l_n/ w_n...length/width of the river as it crosses the cell n; all in cell n, or at node n, respectively)

2.4.1. Leakage[2]

In Modflow the user specifies the conductance [L^2T^{-1}]. In Feflow, the transfer rate (in and out - depending on the direction of the exchange flux) [T^{-1}] is user-defined, an internal multiplication with the relevant area results in the unit L^2T^{-1} [DHI-WASY GmbH, n.d.].

When creating the current Feflow model, it was assumed that leakage occurs via the side surfaces of the river cells (not via the bottom surface, therefore w=1). Figure 2.3 shows this schematically. According to Diersch [2014], Mach et al. [2019] and discussions with the model creators, the volumetric flow rate at river cell nodes is interpreted to function according to the following equations:

$$Q_n = \left(l_{n,1} LFout_{n,1} + l_{n,2} LFout_{n,2}\right) \cdot \left(h_n - RS_n\right) \cdot w \qquad if \qquad h_n > RS_n \tag{2.3}$$

[2]The term leakage factor is used equivalently to transfer rate or conductance per unit area.

$$Q_n = -\left(l_{n,1} LFin_{n,1} + l_{n,2} LFin_{n,2}\right) \cdot \left(RS_n - h_n\right) \cdot w \qquad if \qquad h_n < RS_n \tag{2.4}$$

where:

Q_n	= volumetric flow rate at cell n [L^3/T]
h_n	= piezometric head of cell n [L]
$LFin/out_n$	= leakage factor for influent/effluent conditions at cell [1/T]
RS_n	= River Stage in cell n [L]
$l_{n,x}$	= part of the cell edge [L]
w	= general width = 1 [L]

NOTE that unlike Modflow, in Feflow inflows are considered as negative, outflows as positive values

The user defines or calibrates the leakage factors (LF; [1/T]) per cell and for each direction. The basis for this is in particular riverbed permeability and thickness. Further river water levels per time step and the river bottom elevation as a static value are defined. In the current two dimensional model the simplified assumption is that the river bottom elevation equals the aquiclude elevation.

The fact that leakage factors are not only defined directly at the flow boundary condition (Fig. 2.3), but also at more distant cells results from a refinement (or cell size reduction) in the process of model development. These leakage definitions do not affect the model and remain relic.

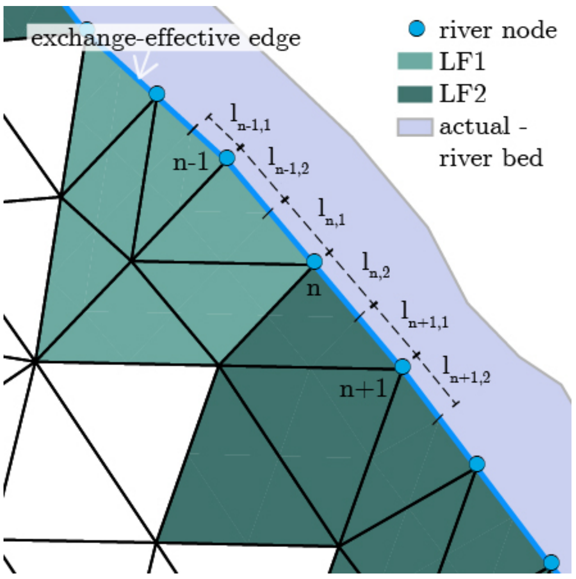


Figure 2.3.: Leakage factor in the current Feflow model.

2.4.2. Head-dependent Recharge

Feflow provides an interface where external scripts can be placed and executed at a defined work step. To enable a groundwater level-dependent recharge query, a script was written by JR-AquaConSol, which stops the simulation after each time step, retrieves the calculated groundwater level, and uses this value to determine the recharge rate for the following time step in the time-depth tables, calculated with STOTRASIM. An example of these tables is given in figure 3.9.

3. Data

3.1. Origin and Quantity

With the motivation to ensure a proper comparability, all input data for the Modflow model was exported as raw ASCII-files from the most current and calibrated Feflow model version, the relating GIS or STOTRASIM files of the SI-MUR-AT project. This procedure has the disadvantage that it deviates from a "classical" model generation: Data is partly already processed, interpolated and calibrated. Further, in retrospect it is no longer possible to determine the extent to which the data has been processed.

Table 3.1 lists the used data. All time-series (TS) are given in daily values and extend from 01.01.1993 to 31.12.2018, resulting in a total of 9495 time-values of each transient data point. For transport modeling the grid was partially refined to achieve the convergence criteria.

Feflow uses the so called "PowerID" to link a data-point in a shape file, that can be any type of boundary condition, to a time-series. This ID is used to reconstruct the connection.

Table 3.1.: Input data. Blue: transient data; orange: STOTRASIM time-depth tables (TDT); SHP: shape files with attributes.

	Type	Format	Quantity	
flow	model boundary	SHP	1	polygone
	hydraulic conductivity	SHP	30,911	points
	specific yield	SHP	30,911	points
	initial condition	SHP	15,947	points
	aquifer bottom elevation	SHP	30,911	points
	constant head boundary	SHP & TS	93	points
	wells	SHP & TS	11	points
	hydrotopes (recharge zones)	SHP & TS	2,433	polygones
	river / surface waters	SHP & TS	1,606	points
	river lines	SHP	3	lines
	transfer-in*	SHP	11,381	polygones
	transfer-out*	SHP	20,911	polygones
	observation measurements	SHP & TS	49	points
	STOTRASIM: recharge time-depth tables	TDT	278	TDT
transport	initial mass concentration	SHP	61,752	points
	mass transport porosity	SHP	121,540	points
	mass concentration BC	SHP & TS	4,425	points
	STOTRASIM: nitrogen time-depth tables	TDT	278	TDT

3.2. Feflow Model Grid and Aquifer Properties

The current Feflow model is constructed as a triangular grid, with irregular cell-sizes and in a single layer. The total cell number amounts 30,911 with an average size of 1,429 m^2 (Fig. 3.1). Areas around wells and along boundary conditions such as rivers have been refined - these areas are represented by small cell sizes. Furthermore, areas that tend to lead to numerical problems are refined in order to facilitate the convergence of the model run. Numerical problems or errors occur when the maximum error tolerance is not undercut within the allowed iteration steps, e.g. as a result of high gradients in the aquifer properties.

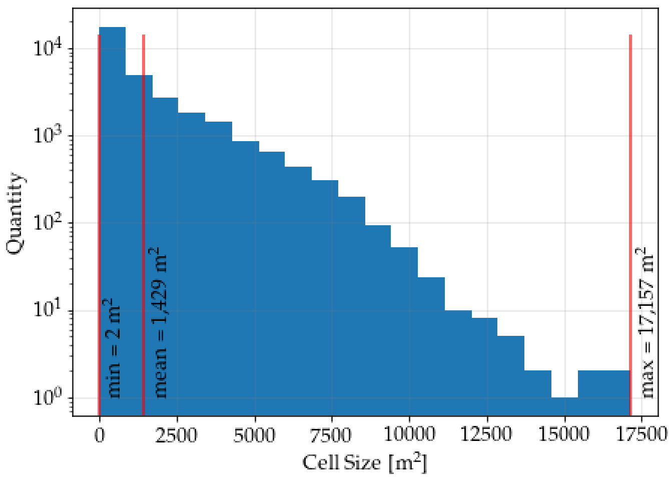


Figure 3.1.: Distribution of cell sizes (n = 30,911) in the Feflow model.

Figure 3.2 shows a section of the Feflow model-grid with local refinements. It can be considered to be representative for the entire model area. Aquifer properties (hydraulic conductivity, specific yield, aquifer bottom elevation, top elevation) are defined for each cell. The initial GWL (IC) and boundary conditions are assigned to nodes.

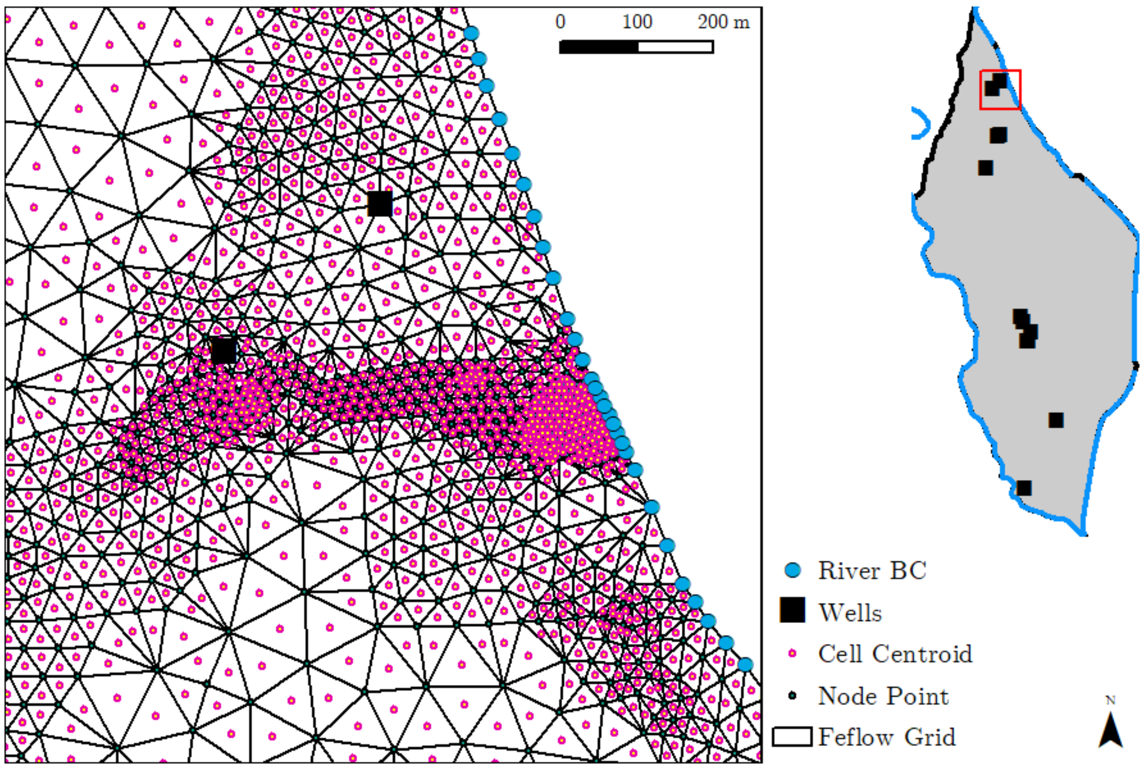


Figure 3.2.: 1 x 1 km section of the Feflow grid. Containing 1318 cells; 1266 cell centroids; 648 nodes; 29 River-BCs and two wells.

The definition of gravel dredgings ("lakes") was made by setting hydraulic conductivities to 1 m/s and specific yield to 1 [-]. Tributary waters or drainages ("surface waters"; located in the middle of the model area, and not along the borders) are defined with a fixed water level as a river boundary condition.

3.3. Boundary Conditions

3.3.1. Rivers, Wells and Constant Head Boundary

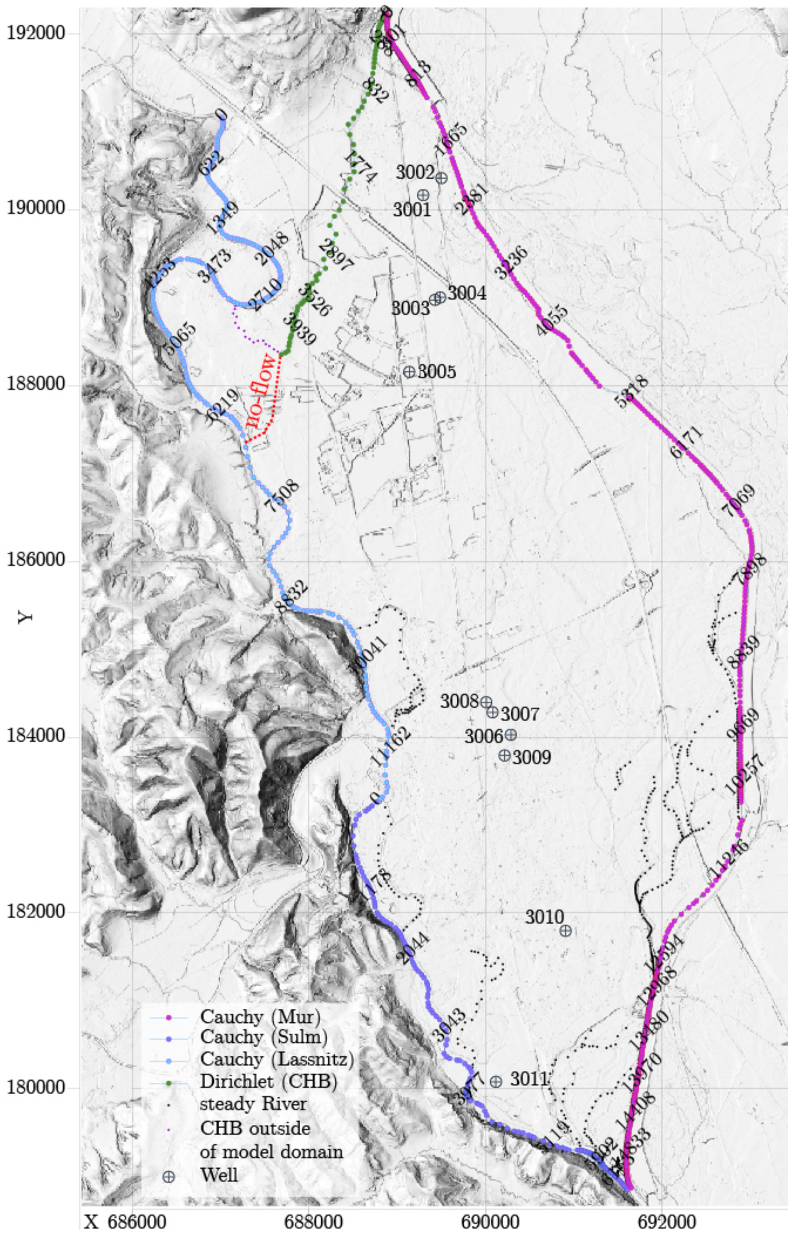


Figure 3.3.: Map of plotted TS-points. X and Y in meters. Inclined numbers describe the length of the river, starting in the north. The river Lassnitz starts outside of the model domain.

Figure 3.3 shows all data-points of boundary conditions at which time-series are deposited, with the exception of recharge. The model area is framed by the boundary conditions, whereby a gap of about 1.5 km in the north-western part remains undefined and is therefore treated as a no-flow boundary. Figures 3.4 - 3.6 depict the temporal and spatial variability of well pumping rates and rivers with leakage factors. The points are linked to the corresponding time-series with an integer ID, which is stored in the point attributes.

The pumping rates vary in the model area. The highest abstraction rates are recorded in the central area ("Kaindorfer Brunnenfeld"). Northern and southern wells show considerably lower abstractions, or in some cases are not even in operation for at least half of the observation period (Fig. 3.4).

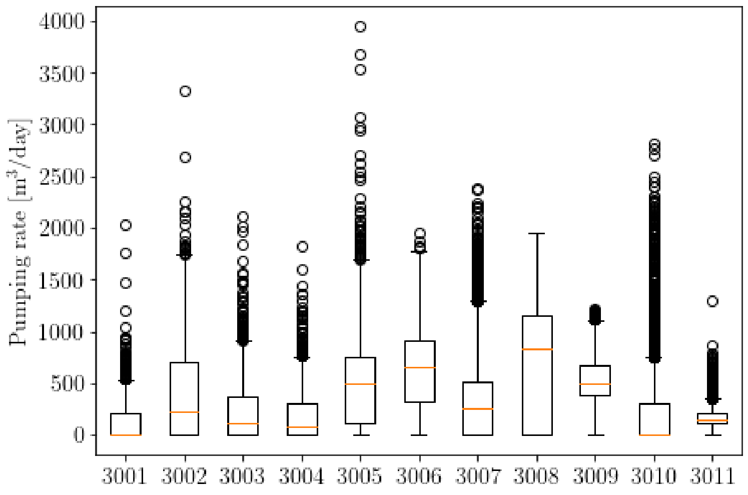


Figure 3.4.: Boxplot of well-pumping rates. Location of wells in fig. 3.3.

Along the Mur, the three distinctive barrages are visible (Fig. 3.5). The target level of the barrage can be read off approximately from the mean value before a drop. Only in the case of the second barrage is a free flow section (the bumps in this area mark the right-hand sided inflow of tributary waters), in all the others the root of the barrage extends over the entire flow section. A fourth barrage follows shortly outside the project area.

Within the project area (from about 6.5 km onwards), the Lassnitz has the steepest gradient of the rivers, with a constant range of extreme and average values. Along the sulm, this range decreases with the flow path (in the project area), especially the closer it gets to the confluence with the Mur. In the last three kilometers before the confluence, the riverbed increasingly flattens out, with the mean water level approaching the minimum more and more. This indicates that a relatively constant water level is maintained in this area due to the strong and presumably constant exfiltration from the project area. The maximum values are thereby reached by individual periodic high water levels.

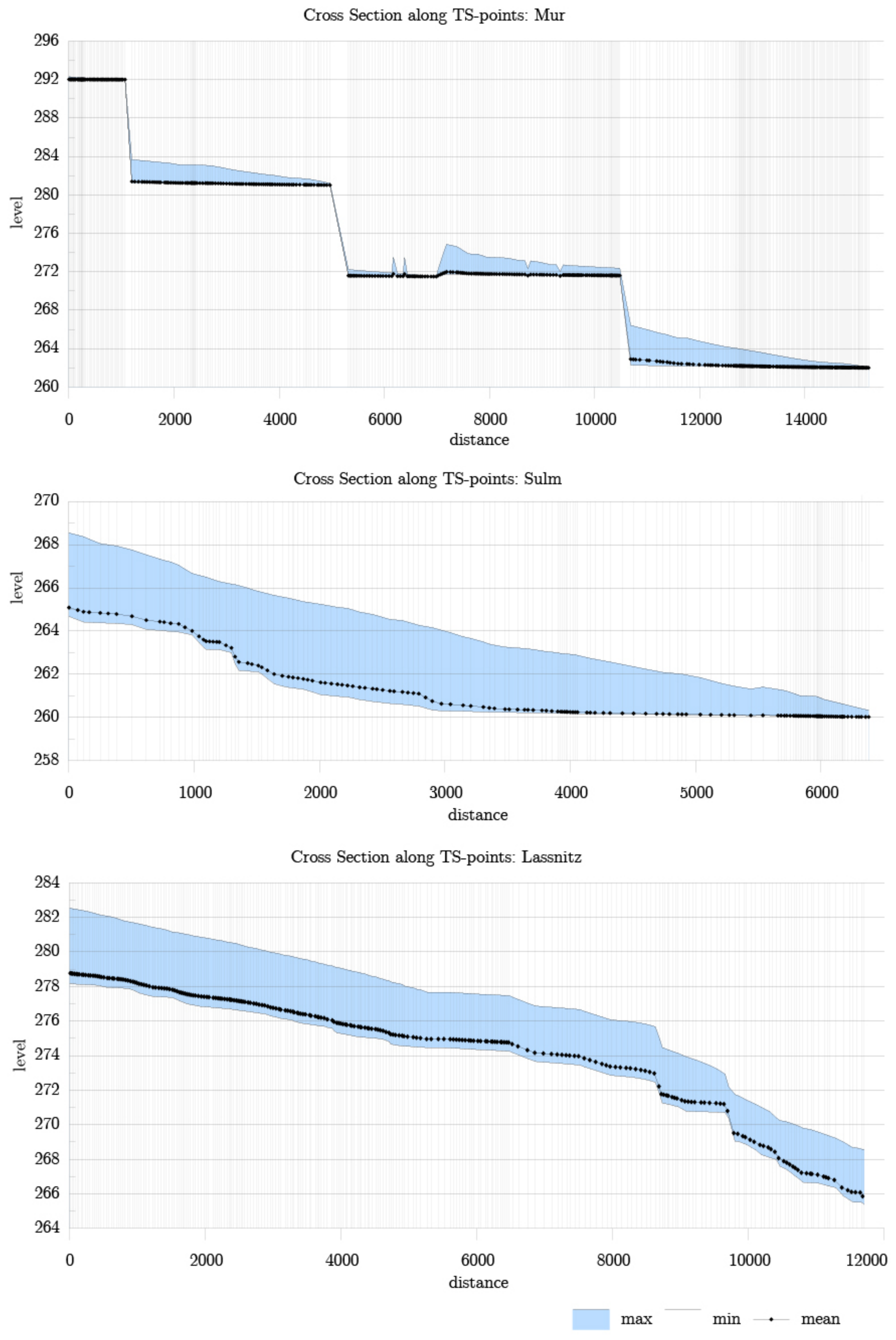


Figure 3.5.: Water levels along the rivers Mur, Sulm and Lassnitz. The jumps of the Mur indicate barrage stages. Location of TS-points in fig. 3.3.

"TRAF_OUT" and "TRAF_IN" represent leakage factors in the different directions of exchange fluxes in Feflow (Fig. 3.6, see 2.4.1). "Out" stands for effluent - from the aquifer into the river, "in" for influent - from the river into the aquifer. It turns out that the values in the same geographical regions differ by a factor of up to 10,000. Both values are calibration results.

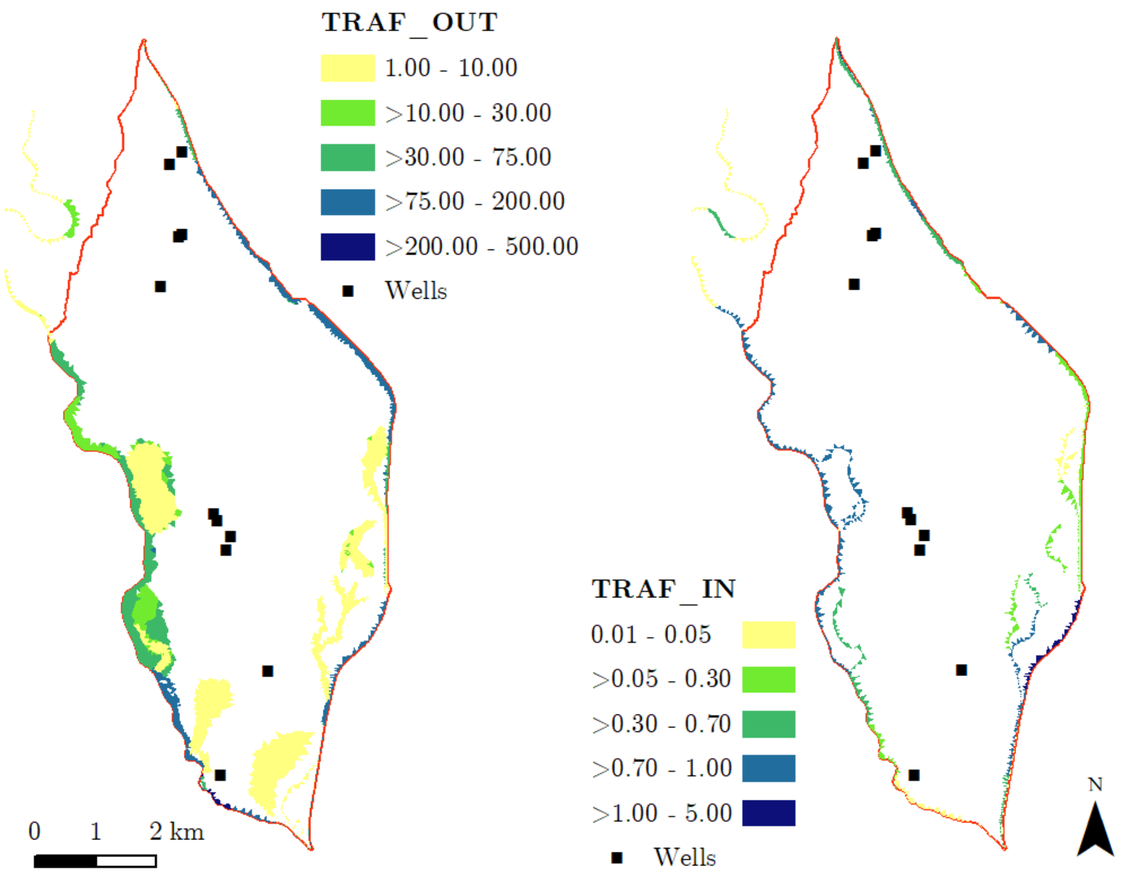


Figure 3.6.: Leakage factors

Transport model

In the transport model river and constant head boundaries are attributed a constant concentration in time and space with 2 mg/l nitrate in rivers and 20 mg/l at the constant head boundary (CHB).

A concentration time-series was assigned in the gravel dredgings (Fig. 3.7), since considerable degradation processes take place in their shallow water. This TS applies for every affected-cell or node, respectively.

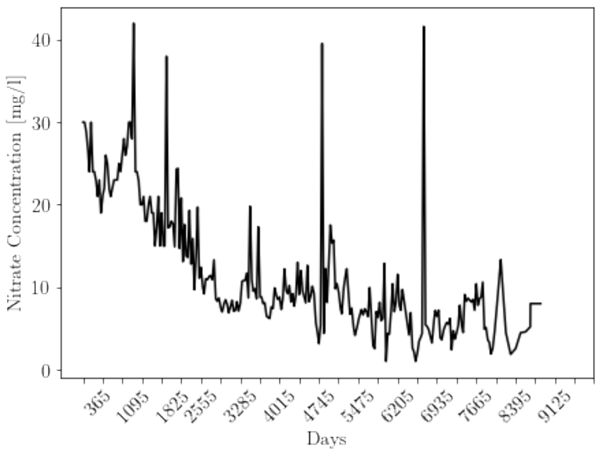


Figure 3.7.: Nitrate Concentration in the lakes over time.

3.3.2. Recharge and Nitrogen Sources

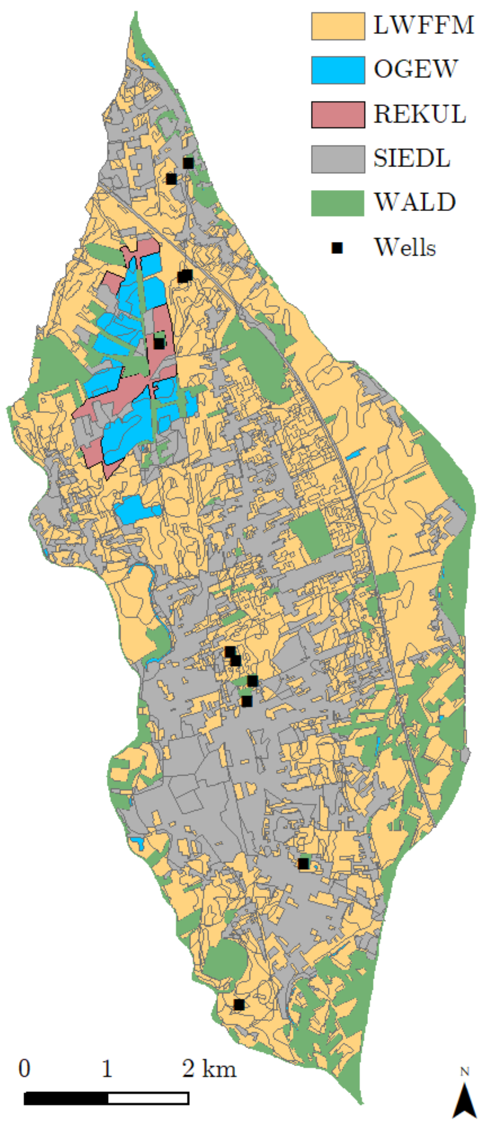


Figure 3.8.: Hydrotope distribution. LWFFM: agricultural areas with crop rotation; OGEW: surface waters; REKUL: restoration areas; SIEDL: urban areas; WALD: forests.

The calculation of groundwater recharge and nitrogen leaching was made externally with the model software packages SIMWASER and/or STOTRASIM [Feichtinger, 1998].
The calculation method is primarily selected according to the prevailing water dynamics, based on land use: agriculture, urban areas, forests, surface waters, and restorational areas. Land use is further blended with site, soil, and weather characteristics, resulting in homogeneous recharge areas and sources of nitrogene leaching called hydrotopes (Tab. 3.2). These models then describe water flows and nitrogen dynamics in the unsaturated zone (area between the ground surface and the water table) in a one-dimensional, vertical direction in a temporal and depth-dependent resolution. Background for the classification, resulting hydrotopes and basis of design are described in Mach et al. [2019].
Figure 3.8 depicts the distribution of land use in the model domain.

Nitrogen transport is entirely bound to water movement, taking into account convection and diffusion/dispersion processes. Representative for all soluble nitrogen compounds, only nitrate is dissolved in the soil water, but this completely and therefore all other nitrogen components (NH_4-N, N_{org}) must first be converted into nitrate in order to gain mobility in the soil. In return, however, any nitrogen demand (vegetation uptake, immobilization) is covered by NO_3-N [Mach et al., 2019].

Figure 3.9 shows an exemplary time-depth diagram of groundwater recharge in hydrotop 138 (agriculture), figure 3.10 the dynamics of nitrogen leaching in the same hydrotop.

Table 3.2.: Hydrotopes. *average annual recharge [mm]. Orange: time-depth dependent recharge.

land use	area [km^2]	recharge*	hydrotops
agriculture	23.32	337	1574
urban	12.54	550	485
forrest	6.16	229	282
surface water	1.45	186	88
restoration	0.65	543	4

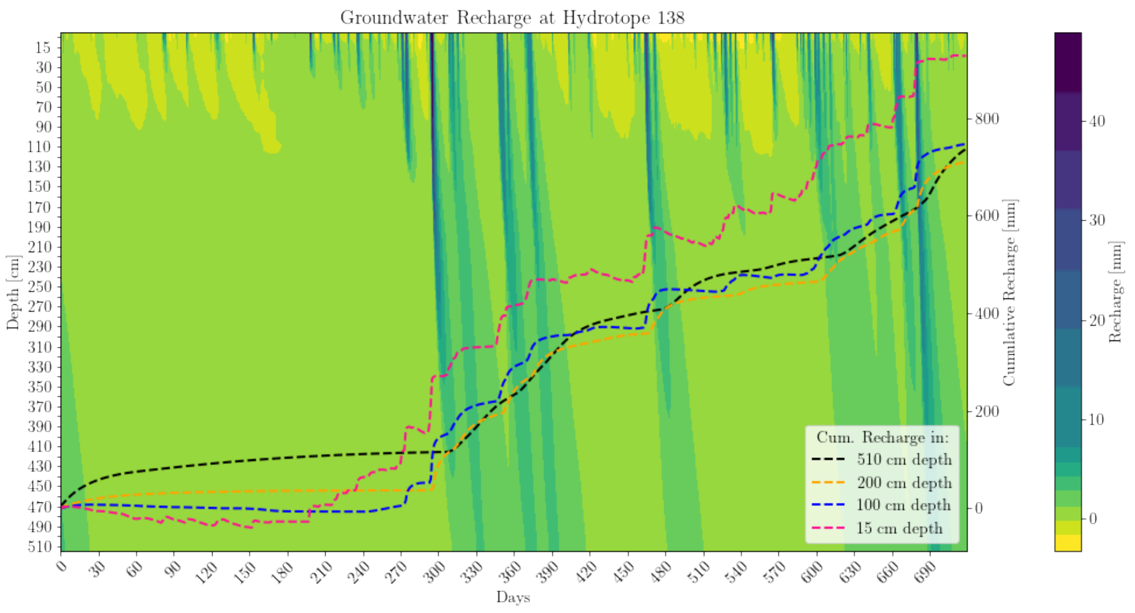


Figure 3.9.: Recharge in an agricultural hydrotope, considering crop rotation. Plotted for a period of 2 years.

In the upper soil layers (down to a depth of about 1 m), plant withdrawal and evaporation is shown by a negative recharge rates (Fig. 3.9). The spread of the wetting-front in the unsaturated zone is visible after precipitation (or watering) events - indicated by a high rate in the upper soil layers, which spreads downwards with a time lag and thus decreases in intensity overall. In order to make differences in the depth-levels easier to read, the cumulative sum is shown. The differences in the depth-dependent recharge rate decreases with increasing depth, especially below the root zone - from here the curve tends to be smoothed.

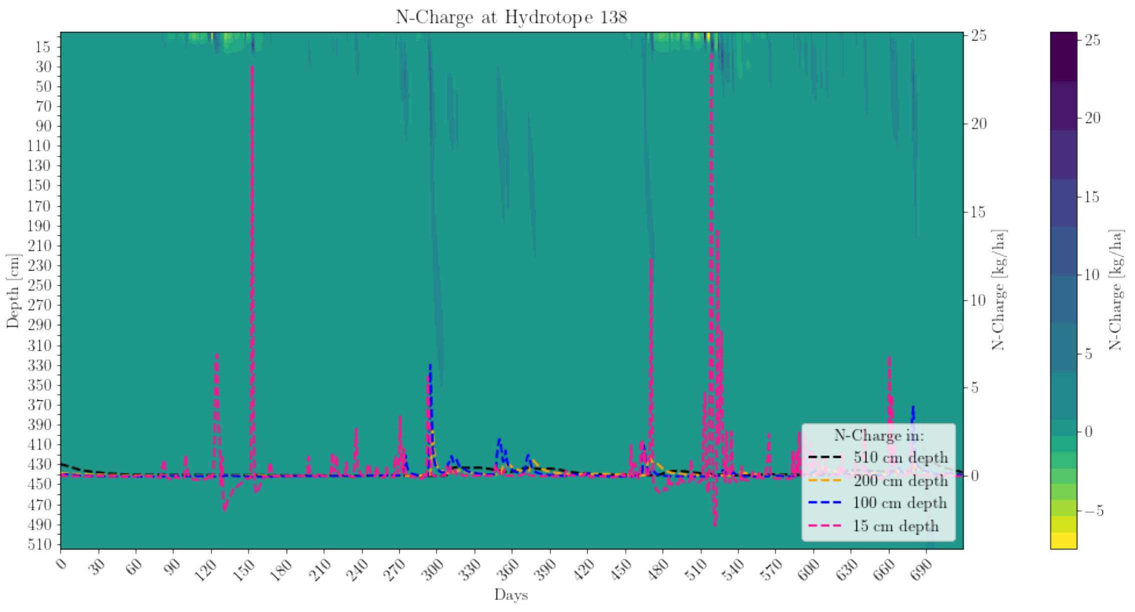


Figure 3.10.: Nitrogen charge in an agricultural hydrotop in kg per hectar, considering crop rotation. Plotted for a period of 2 years.

Negative values for nitrogen leaching are also found in the upper soil layers due to plant uptake, but these negative areas do not reach as deep as the negative recharge (Fig. 3.10). Possible fertilization events cannot be identified in the diagram, but a link to water dynamics is recognizable.

3.4. Feflow Results, Calibration Parameters and Reference Values

In the existing Feflow model, the hydraulic conductivity and the pore volume in particular were important calibration parameters. Furthermore, the river transfer rate was calibrated. The groundwater level measurements at the 48 selected observation wells, following the start of the model on 01.01.1993, were interpolated for the initial condition [Mach et al., 2019].

Figure 3.11 depicts the Feflow-calculated median groundwater level (Q50) in the project area for the time period between 01.01.1993 and 31.12.2018 and the 48 observation wells, that were used for model calibration.

The budget of the Feflow calculation (average daily rate of the first 3650 days) is given in table 3.3. The recharge rate was not recorded, since it was determined with the interface manager. The magnitude of the recharge (as sum of in- plus outflows) can be estimated approximately with the imbalance value. The high divergence between storage-in and -out is not traceable here. Mach et al. [2019] calculate an area-weighted average of 381 mm/a for groundwater recharge rates for the entire study area, which would result in an average recharge of about 41,750 m^3 per day for the entire area.

Table 3.3.: Feflow Budget. Average daily values [m^3/day].

SOURCE	**IN**	**OUT**
Storage	26,683.46	32,289.48
Dirichlet	14,343.41	819.84
Cauchy	16,256.36	57,031.10
Well	0.00	4,373.65
Imbalance		**-37,230.68**

Measured groundwater levels at the observation wells are present continuously at different time intervals, or only in individual periods. The stored Feflow values are available in daily or weekly intervals. Measured concentrations are only available selectively and only at individual observation wells. The existing transport model was not calibrated.

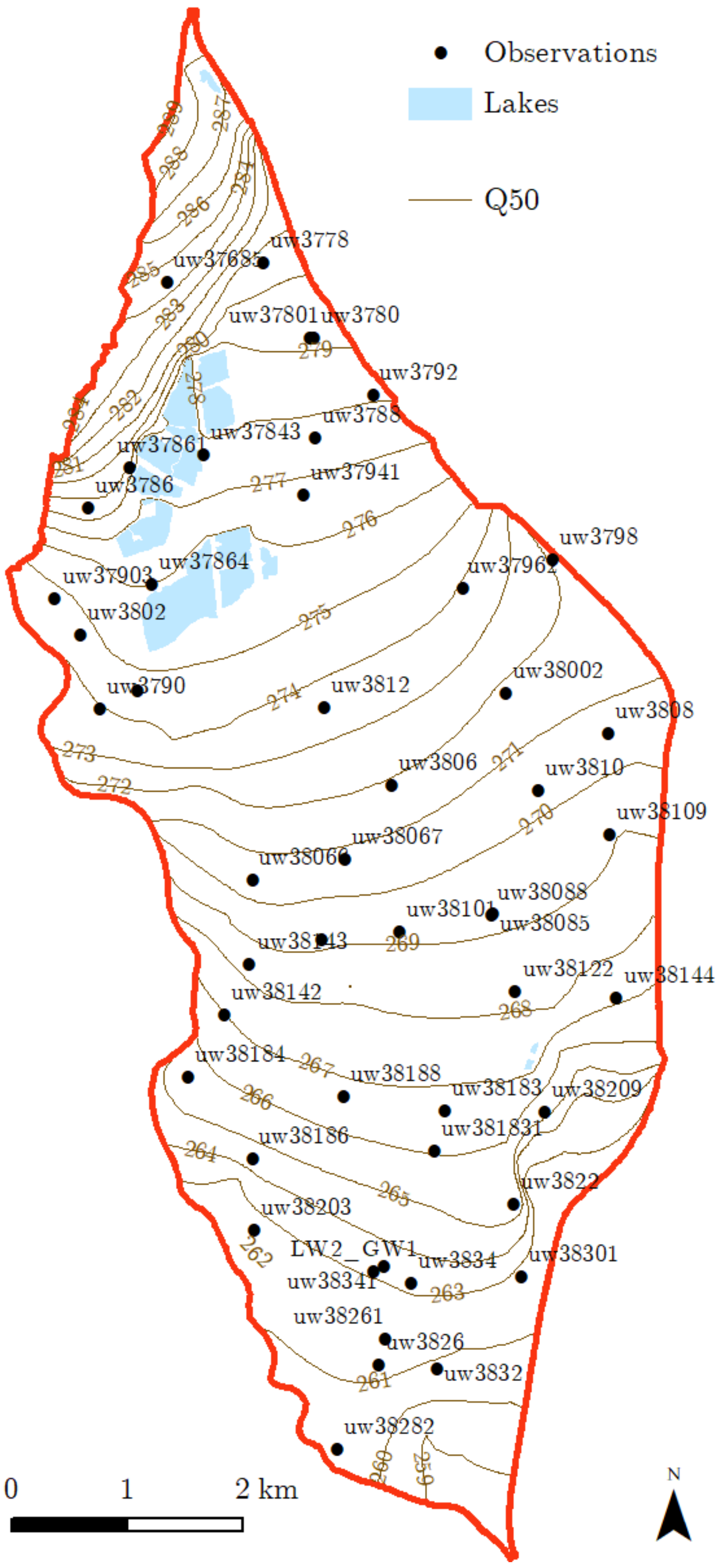


Figure 3.11.: Q50 water level position and observation wells

4. Methodology and Model Setup

4.1. General

As described in 2.4, parts of the Feflow environment (especially the leakage factors and groundwater recharge) cannot be directly transferred to Modflow. For this reason, different approaches were applied and compared in their results. All methods and model runs are identical in their spatial resolution and general in spatial discretisation, i.e. the definition of BC effected cells, or aquifer parameter distribution. Furthermore, the calculation and thus comparative period was limited uniformly to 3,650 days ("stress periods") - from 01.01.1993 to 30.12.2002.

At the time of defining the object of study (June 2020), internal transport modeling was not supported within Modflow (see 2.3). It was planned to use MT3D for this purpose - this transport module only supports structured model grids [Bedekar et al., 2016]. For this reason, the discretization was also performed with such a model grid. The current Modflow version supports transport simulations with vertex-based or unstructured grids. However, in a test not described in this work, it was shown that a changeover to a vertex-based grid could be implemented without major interventions in the Python scripts for a classical model setup. This was only done superficially and with a GWF simulation.

Units
All length units are defined in meter, time in days and concentrations in mg per liter ($\equiv$ g/m^3).

Time-series were formatted and summarized in a table according to the type of boundary condition and indexed with the ID as column name, and the timestamp as row name. All **time-depth tables** were formatted and saved for each individual hydrotop named with the ID value, with depth level in columns and timestamp in rows. Both was done preliminary with separate Python scripts.

4.1.1. Approaches

The current Modflow version 6.2.1 does not allow the groundwater level to be queried during the simulation[1] - the leakage direction and the depth-dependent recharge rate can therefore not be determined dynamically, i.e. according to a comparison of groundwater and river water level or a query in the time-depth tables. The following points give a brief description of applied solutions and a reference to the section in which they are described in detail. The results are presented and discussed in the equivalent section in chapter 5:

Leakage

- export the Feflow budget file and define the **river cells as wells** and thereby use the same explicite values as used by Feflow → Localization of boundary condition-effected cells
- **manual selection** of effluent and influent areas based on the Q50 water level (according to Fig. 3.11) → Localization of boundary condition-effected cells

[1]according to rumors this possibility will be integrated in a future version

- **calibration** of manual selected recharge factors → Calibration

Recharge

- a temporal and spatial **interpolation** of measured groundwater levels at observation wells to account for temporal variations → Temporal Discretication

Change the "run mode" - Loop Modflow

- execute Modflow for one timestep each, read in calculated data, redefine initial- and boundary conditions, execute Modflow again ... → Model Setup and Execution

4.2. Workflow

As the procedure of code-based model creation with the Python package FloPy had to be developed first, attention was paid to modularity, to make it easier to replace or edit parts: Similar steps are combined into single, executing Python files. The respective calculation results are stored in external files so that links between the executing files are prevented and the editing options are facilitated. Intermediate results can be reviewed or plotted. Figure 4.1 shows the schematic workflow with the three main parts (files): (1) Spatial Discretization, (2) Temporal Discretization and (3) Model Executive (RUNMF). These Files are listed in the appendix.

Global or basic properties such as cell size, number of time steps, model name and the order structure are defined in a separate file ("prop.py") to ensure, that the same parameters are used and not mixed up. To "clean up" the following files, packages and functions are also defined here.

4.2.1. Spatial Discretication (App.B.2)

The cell resolution was selected in order to ensure that no details are lost during automated processing, especially with regard to the boundary conditions (BC). Modflow does not allow multiple assignments of BCs of the same kind in the same cell (with the exception of wells, where several wells can be located in one cell) [Langevin et al., 2021]. Especially in the northern part of the project area, there are accompanying drainages (defined as river BC), which run relatively close to the river Mur (Fig. 4.4). In order to be able to represent these, a resolution of 40 x 40 m was chosen. This results in a total of 144 x 335 square-cells. Of these 48,240 cells 25,513 are located within the model boundary and thus active.

The core of the spatial discretisation is an intersection function in which the values of aquifer properties and BCs, which are mostly present as point shapes, are intersected with the cell grid. Further processing is particularly necessary in cases, when:

- multiple points (or features in general) intersects with an active cell or
- none intersects with an active cell (Modflow requires a consistent definition of aquifer properties in active cells)

Both situations lead to an error message from Modflow and to the termination of the simulation, but must be expected in any case. How this can be handled depends on the parameter type:

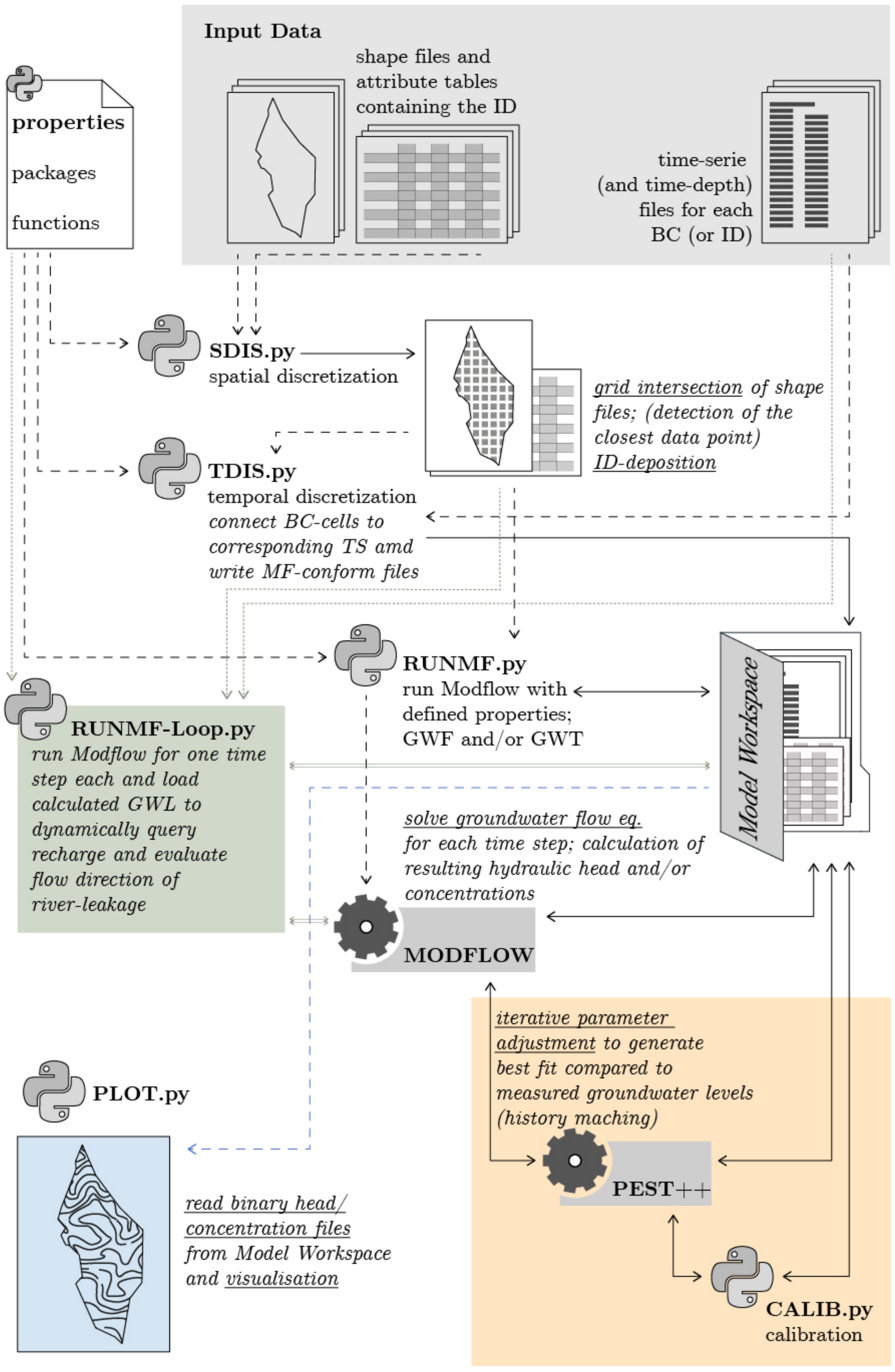

Input Data
shape files and attribute tables containing the ID
time-serie (and time-depth) files for each BC (or ID)
properties
packages
functions
SDIS.py
spatial discretization
grid intersection of shape files; (detection of the closest data point) ID-deposition
TDIS.py
temporal discretization
connect BC-cells to corresponding TS amd write MF-conform files
RUNMF.py
run Modflow with defined properties; GWF and/or GWT
Model Workspace
RUNMF-Loop.py
run Modflow for one time step each and load calculated GWL to dynamically query recharge and evaluate flow direction of river-leakage
solve groundwater flow eq.
for each time step; calculation of resulting hydraulic head and/or concentrations
MODFLOW
iterative parameter adjustment to generate best fit compared to measured groundwater levels (history maching)
PEST++
CALIB.py
calibration
PLOT.py
read binary head/ concentration files from Model Workspace and visualisation

Figure 4.1.: Schematic flowchart and file structure.

Aquifer properties and initial conditions

These parameters - **hydraulic conductivity, specific yield, effective transport-porosity, top and bottom elevation**, as well as the **initial conditions** - are constant over time, and characterized by a continuously changing float value, that allows interpolation. The basis for all these parameters are point shape files. These parameters are intersected with the grid. The average of multiple values in individual cells is calculated. For undefined cells, the nearest available data point is searched starting from its center and assigned. In the case of hydraulic conductivity, the unit must be converted from m/s to m/day - it is multiplied by the factor 86,400.

To prevent numerical issues or an unintentional change in the groundwater flow calculation method because of possible ponding, when initiating the model run, the aquifer top elevation is set to a constant, freely chosen level, above the highest elevation of the real ground level (at 1,000 m).

Interim Results

Model domain area: When creating the model gird, each cell that is intersected by model boundary is set active. It is irrelevant whether the whole cell is enclosed or only a part of it is sectioned. This results in an enlargement of the model domain area. The original Feflow model has an area of $4.00 \cdot 10^7$ m^2. The discretization results in 25,513 active cells of 40 x 40 m each and thus by 820,800 m^2 or 2 % more, where recharge can occur.

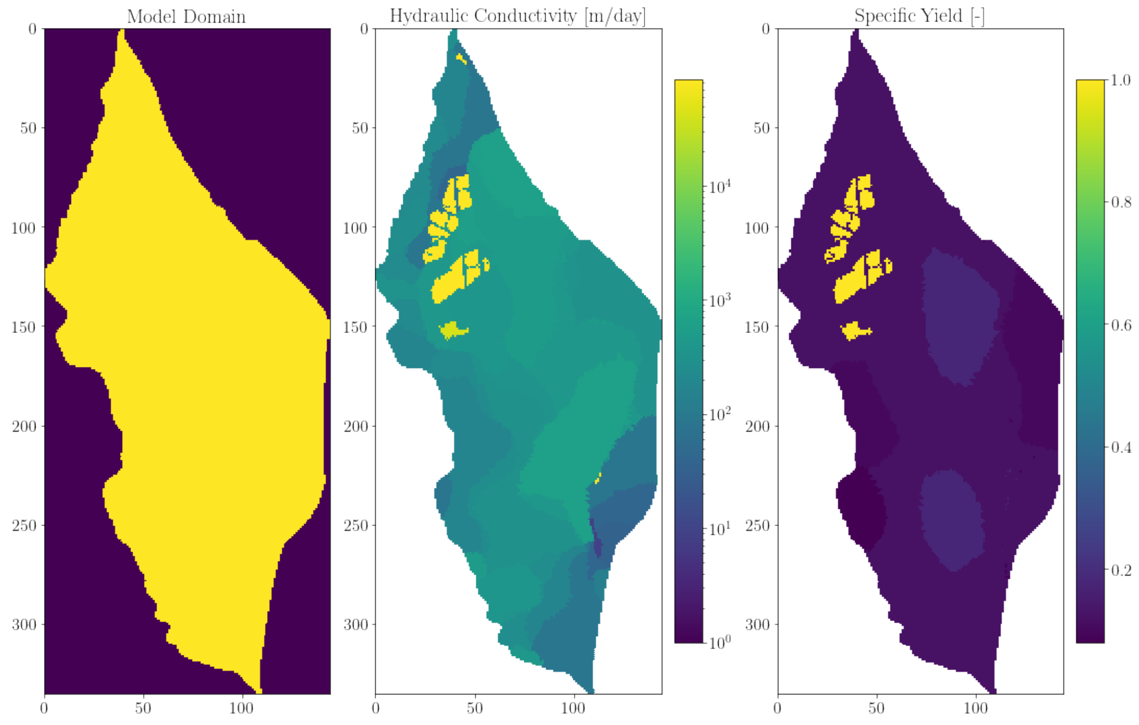


Figure 4.2.: Results of Spatial Discretization: Model area and aquifer parameter.

The transfer of **aquifer parameters and initial conditions** to the model grid is shown in figure 4.2 and 4.3. Thereby, 13,278 out of a total of 25,513 (52 %) cells can be defined by an average value (at least one data-point plots within a cell), in the case of bottom elevation, hydraulic conductivity and specific yield (cell properties in FF). At the remaining cells, the next data point must be searched for, whereby the maximum distance between data point and cell center is 103 m.

In case of the initial condition, where there are fewer data points (node property in FF), the proportion of cells where an average can be formed is lower (33 %), but the maximum distance to the next data point is not significantly higher (116 m).
Since parameters relating to the transport model are available in a much higher resolution, an average can be formed at 78 / 92 %, the maximum distance being 59 / 44 m, respectively (initial concentration / mass transport porosity).

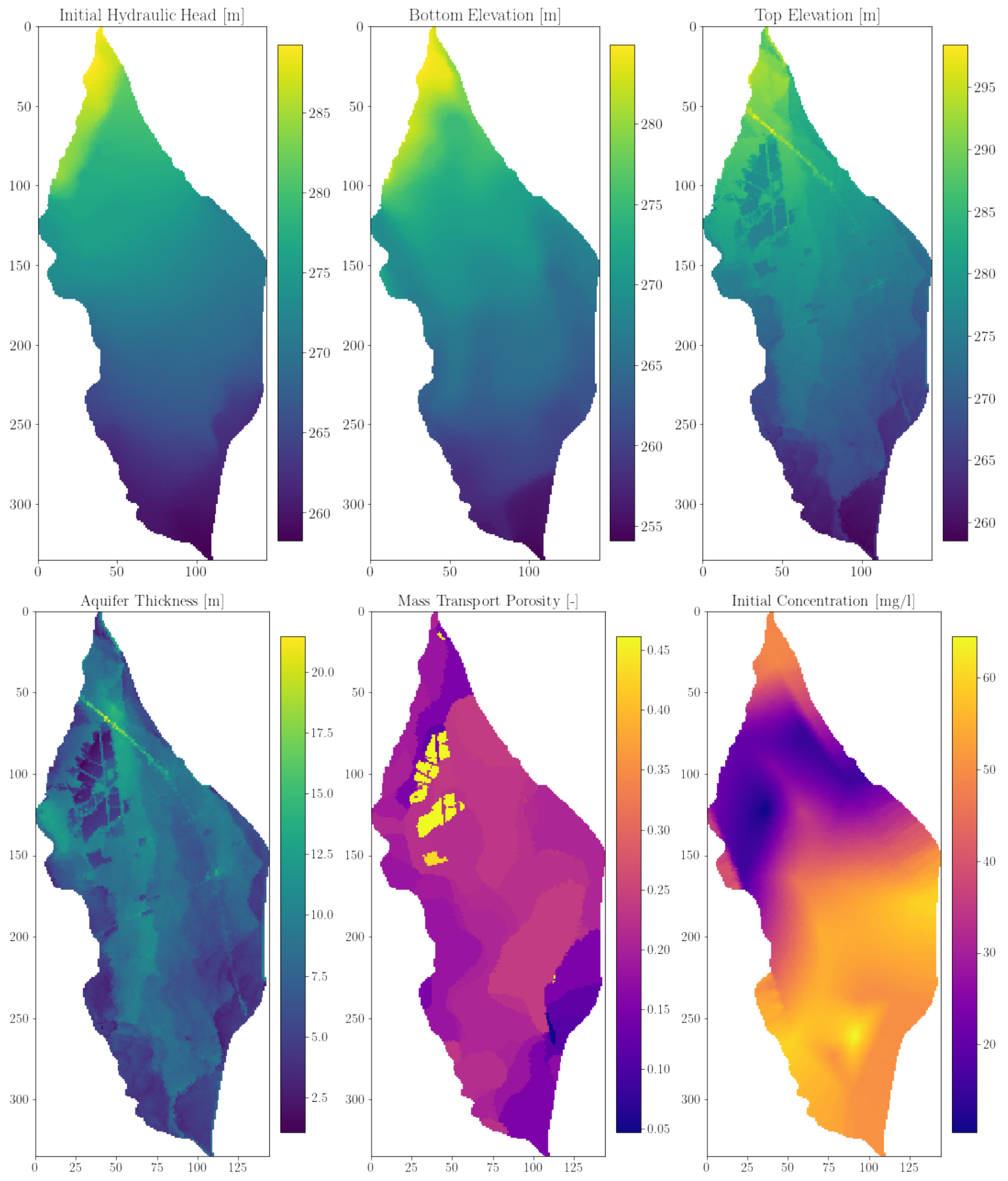

Initial Hydraulic Head [m]
Bottom Elevation [m]
Top Elevation [m]
Aquifer Thickness [m]
Mass Transport Porosity [-]
Initial Concentration [mg/l]

Figure 4.3.: Results of Spatial Discretization: Aquifer Parameter and Initial Conditions. Aquifer thickness results from a subtraction of top minus bottom elevation.

Apparently, all these discretization results correspond relatively exactly to the "original" data in Feflow (FF) (with the visual representation within the FF interface as a basis for comparison). The described parameters (with the exception of hydraulic conductivity and specific yields) show gradual values, without larger jumps. Further, if larger jumps would occur, a high data-point density (high cell resolution in FF) is present, leading to a more direct transmission. For these reasons, it is not assumed that important details are overlooked within the maximum 116 m spacing (approx. 3 cell lengths).

The hydraulic conductivity and specific yields are defined in a zoned manner. With the chosen discretization method it cannot be excluded that a gradient occurs at sharp boundaries (a "transition cell"). From the author's point of view, this fact must be accepted in the course of discretization. Visually, it can be observed that "small" zones with extreme values are apparently transferred to the model grid in a suitable extent (e.g. yellow spots in hydraulic conductivity in fig. 4.2).

Localization of boundary condition-effected cells

Table 4.1 describes the methodology for discretizing cells affected by boundary conditions according of their type and the parameters. The handling of river-BC cells requires several interventions and is explained in detail below.

For all types of BC, tables are created in which the affected cell indices, an ID to link the cell to the corresponding time-series, and further relevant parameters are stored. Table 4.2 shows an example for the river-BC table.

Rivers:
Data points of rivers are directly intersected with the grid. If more than one point fall into one cell, the first value is kept, duplicates are removed (random selection). Theses cells are the basis - other parameters are only added if they share the same cell indices. For the rivers, the node ID is also transmitted, because the Feflow cell budget is stored with this number - what is needed if the rivers are applied as wells.
The rivers leakage factors are available as polygon shape files. These are blended with the grid and a weighted average value is calculated for each cell, according to their intersecting area for each direction (leakage in and out).
The effective exchange edge of the river (compare 2.4.1) is further intersected with the grid and assigned to the corresponding cell indices, just like the polygons, that were created to classify inflow areas (the only shape file that was created manually in this thesis). This classification was done on the basis of the angle of incidence of groundwater level isolines to the river boundary condition in Fig. 3.11. The resulting table is shown in 4.2.

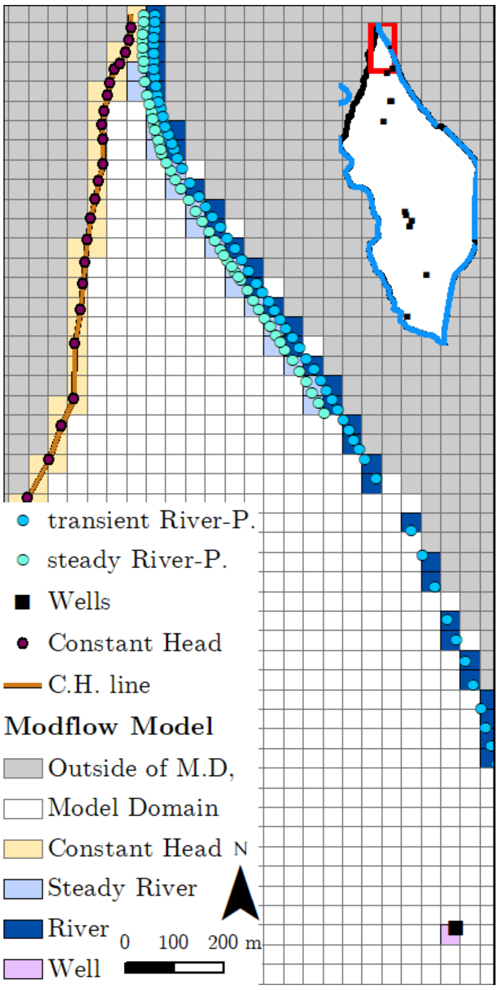


Figure 4.4.: Discretization in Modflow.

Approach: Rivers as wells

This approach is a simple method to get results without worrying about the transposition of the leakage factors. With the export of the budget per time step (each day) at each relevant node (with a river BC) in Feflow, an explicit value is available, which specifies the volumetric water flow at the respective node. This value corresponds to the same type of boundary condition as that of a well BC (well pumping rate). The +/- sign indicates whether water is going into or out of the cell (inflow / outflow).

The Modflow grid is then intersected with the river data points, duplicates in same cells are kept. In the next step (Temporal Discretization) a flow rate from the budget file is assigned to the cells according to their "Node ID" and saved as well. This value is positiv for influent and negative for effluent conditions.

Table 4.1.: Boundary Conditions: Descretization Methods. *stored in a table for each BC type.

	MF Requirements	**Available Data and Format**	**Method**	**Resulting Parameter***
WEL	cell indices; pumping rates;	point shape with ID of TS	direct intersection of point shape and ID assignment	cell indices; TS - ID;
CHB	cell indices; hydraulic heads;	point shape with ID of TS; line shape, connecting points;	intersection of line shape; assigning ID of nearest data point	cell indices; TS - ID;
RIV	cell indices; river stage; conductance; river bottom	point shape with ID of TS; line shape, connecting points; polygon shapes of leakage factors; polygons of manual selected leakage direction	intersection of point shape (randomly dropping duplicates), ID and Node ID assignment - identification of affected cells; intersection of all other shapes and merging on affected cell indices (using weighted average on leakage, according to their area per cell)	cell indices; TS - ID / stage; Node ID; type (steady?); length of line shape per cell; leakage_out; leakage_in; leakage direction;
RCH	cell indices; recharge;	polygon shape with hydrotope IDs	intersection of polygons: assignment of the hydrotop ID, that covers the largest area of the cell	cell indices; hydrotop ID;
CNC	cell indices; concentration;	point shape with ID	intersection of point shape (randomly dropping duplicates) and ID assignment	cell indices; TS - ID; type (steady?);

Table 4.2.: Section of the river-table. "cellids" describe row and coloum indices of the cell. If "steady" = 1, "F" defines a constant river stage; if "steady" = 0, F is used as an ID for the time-series.

	cellids	F	node	steady	lengths	lkg_in	lkg_out	riv_sec
19	(179, 133)	268.489	13147	1	40.016	0.05	6.55	
20	(180, 132)	268.471	13146	1	42.95	0.05	5	
21	(38, 57)	1218	13141	0	2.486	0.5	50	out
22	(90, 87)	10364	13139	0	40.384	0.699	120	in
23	(91, 88)	10363	13134	0	8.033	0.699	120	in

Interim Results (BCs)

Figure 4.5 depicts the location of BC-cells and gives an overview of the resulting leakage factors for a manual selection of the flow direction.

An independent evaluation of the discretization is difficult in this case, rather the effect must be assessed. However, it turns out that in the case of the river BC, isolated holes are produced, which does not make sense from a conceptual point of view (Fig. 4.4 and 4.5). Objective of this thesis is the comparison of results in numerical modeling - a priori these conceptual weaknesses are accepted. In the numerical calculation it is assumed that these holes are compensated by a larger length factor (compare 2.4.1). The transfer of BCs and their effects are presented and discussed in more detail together with Results.

4.2.2. Temporal Discretication (App. B.3)

For each BC, an iteration is performed over the calculation period (timestamp) and a second nested one over the affected cells (ID). The corresponding value is queried in the time-series with the timestamp and the ID. The resulting table is saved per time step in a format readable by Modflow.

Output: With the temporal discretization, one file is written out per boundary condition for each Stress Period (SP) (5 BCs · 3,650 SPs + nitrate leaching · 3,650 SPs = 21,900 files).

Rivers (manual leakage selection):
While iterating, an if condition is interposed: only if the flow direction in the cell ("riv_sec" is Tab. 4.2) is attributed with "in", the inflow leakage is multiplied by the intersection length - for all other values, the outflow leakage factor is multiplied. For outflow leakage, the intersection length is only multiplied if it is greater than the standard cell size (> 40), otherwise it is multiplied by the cell size.
The elevation of the aquiclude is transmitted to the river bottom. Table 4.2.2 shows an example of the output file for day 1,429 (same cells as listed in Tab. 4.2).

Recharge (GWL Interpolation) and Nitrate Concentration:
The fact that there is a relatively high density of observation wells is taken to advantage and a spatial interpolation is carried out at these support points for each time step. To obtain values in areas which cannot be calculated by interpolation (marginal areas - extrapolation would be necessary), the "nearest neighbor" value is used.
As the recharge rates are available in groundwater depth levels starting from ground surface, the calculated values are subtracted from the ground elevation level. All time-depth tables are read into the RAM to enable a faster query in these tables. For each cell, the depth level with the lowest height difference to the interpolated groundwater level is determined in the respective hydrotop.

Table 4.3.: wlf-wie.riv_stress_period_data_1429.txt. Columns: cell indices (first index describes the layer; Note that Python is zero-based, Modflow/Fortran starts counting with 1), river stage, conductance, river bottom and auxiliary variables.

```
...
1 180 134     268.4897355     262.1331180     264.6917103       1.00  0
1 181 133     268.4710066     214.7536166     264.6414261       1.00  0
1 39 58      281.3500000    2000.0000000     278.9455011       1.00  out
1 91 88      281.0832772      28.2688445     273.3883365       1.00  in
1 92 89      281.0774003       5.6234284     273.2612512       1.00  in
...
```

The determined value is converted to m^3 per m^2 and transferred to the cell. In the same step, the nitrogen value is queried in the nitrogen time-depth tables (which are structured in the same way as the recharge tables in regard of the depth levels).
Nitrogen, as a component of a nitrate molecule, is expressed in kg per hectare. To obtain the mass of nitrate, it is multiplied by the factor of 4.429 (molar mass of nitrate / molar mass of nitrogen). As Modflow only allows concentrations and not masses as input parameters for contaminant transport, the nitrate mass is divided by the recharge (both normalized to the same unit area).

Initially, the median groundwater level was used to interrogate the recharge value per stress-period in the time-depth tables. The use of an interpolated daily groundwater level represents an extension here. In general, this method has some weaknesses:

- only applicable if a high density of observations is available
- only applicable for history matching, not for a future forecast
- neglects aquifer properties; results in a more or less planar groundwater surface; Darcy's law is not included (compare difference maps; e.g. Fig. 5.5)
- the applied interpolation method only works linearly within the support points (observation wells) - marginal areas (where extrapolation would be needed) are assigned with the "nearest neighbor" value, which definitely results in little correct outcomes; BUT these areas are predominantly covered by hydrotopes of the "forest" land use type. In this land use type, no distinction is made between depth levels for the calculation of the recharge (one level).

4.2.3. Model Setup and Execution (App. B.4)

In Flopy, the model setup basically follows the structure described in section 2.3.3. With the initialization of the simulation instance, the workspace and folder structure is defined and the path to the executing Modflow file is provided. The amount and unit of time steps (called "stress periods") are defined in the timing module. In the following, convergence criteria for the numerical solution (IMS package- Iterative Model Solution) are defined. In all simulations, default values were used. At most, the parameter "complexity" was changed from "moderate" to "complex" if convergence could not be achieved. A model instance (both GWF and GWT) is created and registered together with the numerical solution for the simulation. The hydrogeological packages as well as the output control and observation packages are assigned to this model, whereby only basic properties are defined in the case of boundary conditions and paths to the previously generated files (in Temporal Discretication) are stored for each stress-period.

Flopy then generates files readable by Modflow, saving one file for each registered package and a parent "Name File" that contains the names of input and output files used in a model simulation and thus controls the parts of the model program that are active. With the execution of the program, the groundwater flow and/or transport is calculated and stored in a binary files per time step (depending on the selected properties in the output control package).

Loop Modflow

In this approach, the model is setup classically. However, all transient data (BC) are nested into a loop:

- setup simulation, GWF and/or GWT model and corresponding modules (Timing Module is set to one stress period)
- definition of steady parameters and initial condition in appropriate packages
- creation of a table to store values at observations for each iteration step as the MF-observation package does not work in the loop
- start **Loop:**
  - query the previously created stress period data for BCs that are not affected by GWL fluctuations (CHD, WEL and CNC); transfer to Modflow / Flopy so that it is assumed to be the first time step;
  - query of the initial condition for RIV affected cells, comparison with river stage and definition of leakage factor
  - query the initial condition for each RCH cell, access the TDT, definition of recharge and calculation of the related concentration input
  - write simulation files and run MF
  - remove all BC that are looped
  - load hydraulic head and/or concentrations from the binary files and redefine initial condition; further store the values at observations in the observation-table

This method is very time-consuming. The simulation of the 3.650 days takes about 42 hours[2].

4.2.4. Calibration

In the current Modflow model, the calibrated data basis can largely be taken directly from the Feflow model. One exception here is the leakage factor, and especially the resulting conductance (see 2.4.1).

For this purpose, the calibration program PEST (v 17.2) [Doherty, 2021] is used and operated via the Python framework pyEMU [White et al., 2016].
A direct interface to Modflow or FloPy is not integrated in pyEMU, and therefore manual intervention is necessary, for example to create the template files and to format the instruction files in the

[2]Given calculation times, always refer to a simulation done with a laptop with an Intel i7-8565U CPU with 1.80GHz (4 cores) and 16GB RAM.

required manner. A detailed documentation is given by Doherty [2021] and can be downloaded together with the executing files from `https://pesthomepage.org/`.

The exact selection of the calibration parameters (par) is based on model run 2 (see Modelrun 2) and the selected flow directions, chosen according the isohypses of the FF Q50 water-level, respectively:

par 1	inflow-, and
par 2	outflow areas of the rivers, following figure 3.6;
par 3	"problematic" area in southeastern model area - surface waters and drainages western of the river Mur (Mur itself is attributed with "inflow" in this area),
par 4	all other cells defined as river-BC in the central model area (other surface waters and drainages)

The calibration of the four parameters was carried out over seven iteration steps, with the simulation being carried out several times in each case. The calibration takes about 19 hours in total.

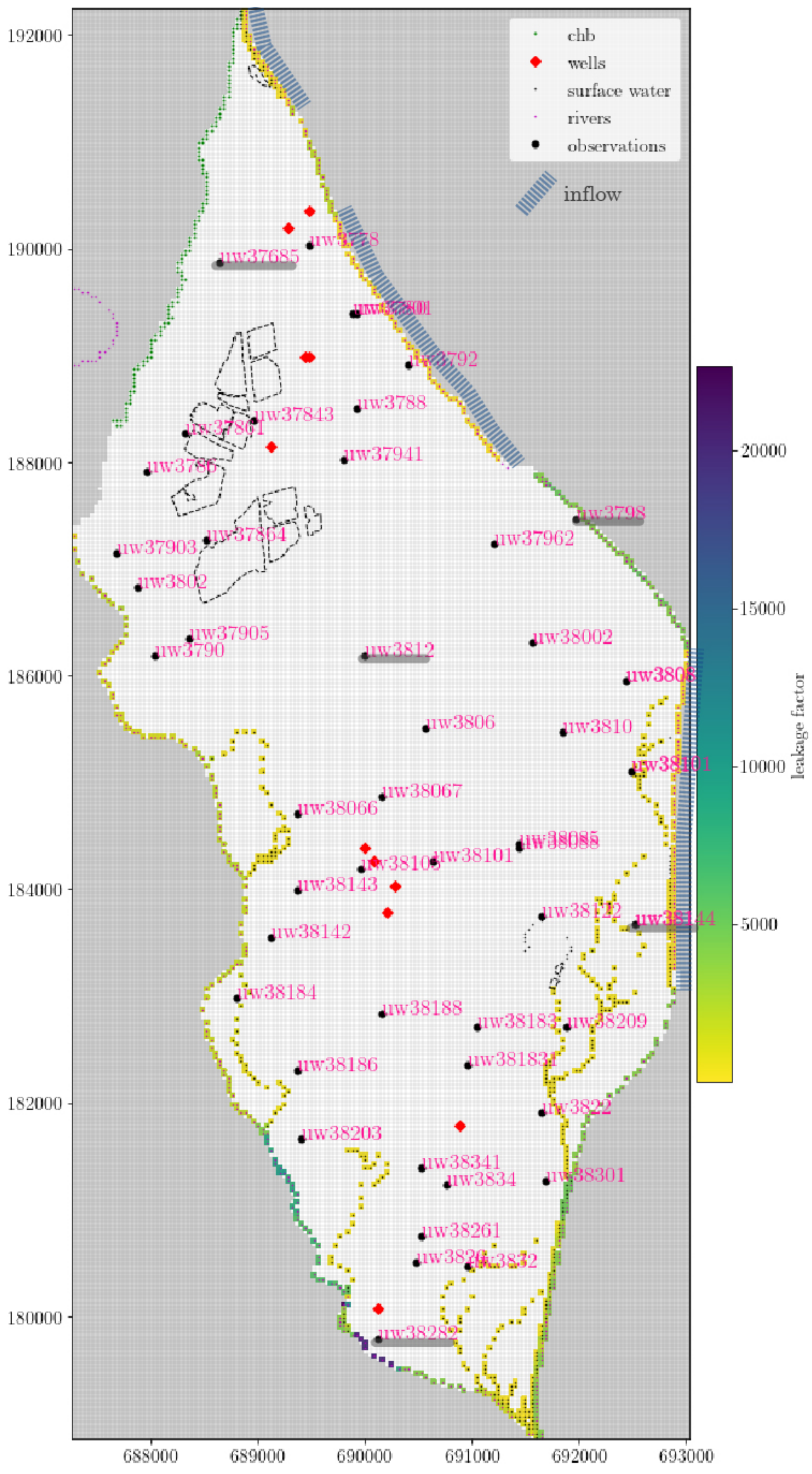


Figure 4.5.: Results of Spatial Discretization: Localization of BC-effected cells and leakage factors for manual selection of flow direction. Hydrographs of underlined observations are hereafter depicted in detail. Inflow areas are marked, other marginal-regions are attributed outflow-dominated.

5. Results

A compilation of the methods and modelrun (MR) that are described and discussed in this thesis is listed in the following table:

Table 5.1.: Modelruns and parameters. Page numbers refer to the respective method section.

modelrun	leakage	recharge	run mode
MR 1	rivers as wells (page 29)	mean GWL (page 30)	classic
MR 2	manual selection (page 28)	interpolated (page 30)	classic
↳ MR 2a	calibration (page 32)	interpolated (page 30)	classic
MR 3	GWL query (page 32)	GWL query (page 32)	loop

For the evaluation three types of groundwater levels are available. These are: (1) Current MF model run; (2) Calculated by Feflow; (3) measured at observation wells. Unless otherwise stated, comparative values always refer to the FF results. In order to compare the different model runs with each other, five hydrographs at the same locations are shown in detail. A difference map showing the difference between MF and FF calculated median (Q50) water level is presented for each model run. The median FF water level is calculated from the groundwater contour lines (Fig. 3.11) and is therefore only an approximate reference.
A compilation of the results at the observation wells is also presented in the form of scatter plots. Here, the average value of all time-steps per observation serves as the basis.
An overview plot of all observations is included in the appendix.

Further, the Root Mean Square Error (RMSE) - the square root of the average of squared differences between calculation and actual Feflow result - is used to verify the model results.

The location of observations that are shown in detail and selection criteria are described below and are plotted in figure 4.5:

UW37685	most northern OBS, ≈ 350 m (6 diagonal cells) distance to the CHB no influence by river-BC is expected here, but high influence of CHB
UW3798	located eastern, on the northern third, close (2 cells) to river Mur high response to river is expected; free-flow section of the Mur, high fluctuation expected
UW3812	centrally located OBS less direct river influence, perhaps a clearer signal of flow from northern part of aquifer;
UW38144	located eastern, on the southern third, between "surface waters" and river Mur high river (with inflow) and surface water (with outflow) influence;
UW38282	most southern OBS, close to river Sulm and Sulm-Mur confluence high groundwater outflow expected

Transport Simulation

In principle, the contaminant transport was simulated together with the GWF in each model run, or rather this was attempted. In advance, it can be stated that no GWT modelrun could be completed without errors - neither in a synchronous (coupled with GWF model) nor in a stand-alone simulation. The coupled simulation of GWF and GWT was always aborted after a maximum of 20 stress periods due to missing convergence. Even suppressing the convergence criterion did not yield any results.

A separate simulation terminates after (ALWAYS) 530 stress periods, with the explanation that the binary budget file is not readable for this input data. An example of this short-period results for modelrun 2 is given in the appendix (A.5).

Furthermore, the attempt to use MT3D for the transport simulation was unsuccessful. The MF 6 documentation describes that a coupling with MT3D is possible when using a structured model grid [Langevin et al., 2021]. However, it has been shown that in FloPy MF 6 models cannot be read in by MT3D. In the official channel of the USGS on GitHub, this problem has been listed as an open issue since January 2020 (Issue #775: `https://github.com/modflowpy/flopy/issues/775`).

5.1. Modelrun 1: Rivers as Wells, Mean GWL for Recharge

The scatter plot (Fig. 5.1) gives an overview of the results: While the calculated values of the higher altitude (northern) observations correspond quite exactly with the FF results, they deviate upwards below the middle altitude. The southernmost OBS has the highest deviation with an RMSE of 1.573 m (RMSE of the northernmost is 0.07 m). The overall RMSE amounts 0.764 m. This shows that there is too much water in the system and it is accumulating towards the south.

This high groundwater level in the south (partly significantly above ground level) is shown more clearly in the difference map (Fig. 5.2). The curvature of the isohypses corresponds relatively exactly to those of the Feflow calculation (Fig. 3.11) - only their values differ significantly. In addition, there are noticeable border areas further north which have markedly too high water levels (ΔQ50 > 1 m) values and in particular, the area along the CHB although there is no river in its vicinity.

At the selected observation wells (Fig. 5.3), it can be seen that northern one (UW37685) agrees very closely with the FF results over the entire time span of 3,650 days. UW3798 shows little similarity despite the spatial proximity to the Mur, which is defined by an identical volumetric flow. The dynamics of the FF calculation are roughly adopted in the current calculation, but with a higher amplitude. Towards the end of the first year, the calculated curve jumps up by about half a meter. This jump correlates roughly with a measured high and can also be observed to a lesser extent at the other observation wells. UW38144 behaves similarly asynchronously, with the divergence increasing even more with progressing simulation period. This observation well is somewhat further away from the Mur, but in the immediate vicinity of the surface waters. UW3812 and UW38282 are similar in their pattern. In the first years of calculation, a high degree of agreement with the FF results can be observed. At the southern located UW38282 a progressive divergence starts a little earlier (about 2.5 years) than at the central located UW3812 (about 4 years). Especially at the southern one, the divergence becomes more pronounced over time and amounts to more than 3m at the end of the calculation period (UW3812: 0.65m).

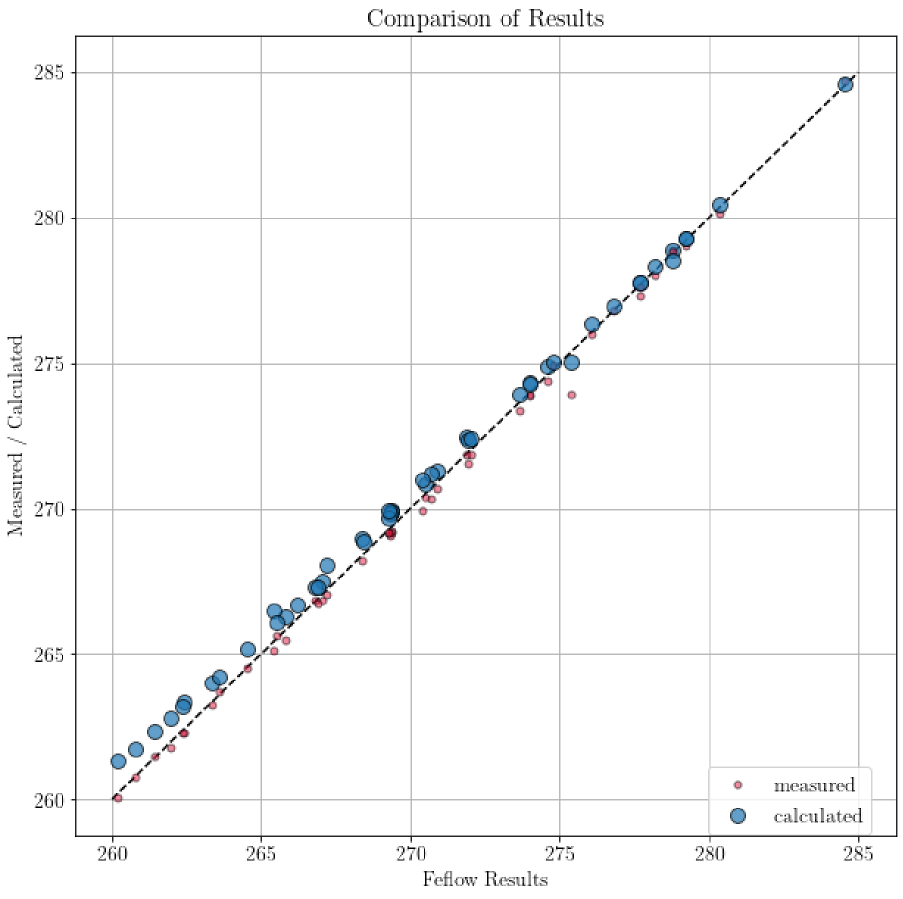


Figure 5.1.: Scatter plot of MR 1: Rivers as Wells.

The query in the budget file (Tab. 5.2) shows a distribution of water turnover of the BCs. The inflows into the system are strongly dominated by recharge with a share greater than 40%. The outflow occurs to 60% via the rivers. The difference of the totals shows an average of about 50 m^3 higher outflow than inflow (0.05%).

Table 5.2.: MR 1: Budget. Average m^3 / day. *sum of rivers and wells.

SOURCE	IN		OUT	
STO	25,452.51	26.94%	27,489.98	29.08%
WEL*	15,816.55	16.74%	61,679.21	65.26%
RCH	38,864.98	41.14%	2,666.19	2.82%
CHD	14,328.17	15.17%	2,679.68	2.84%
TOTAL	94,462.21		94,514.16	

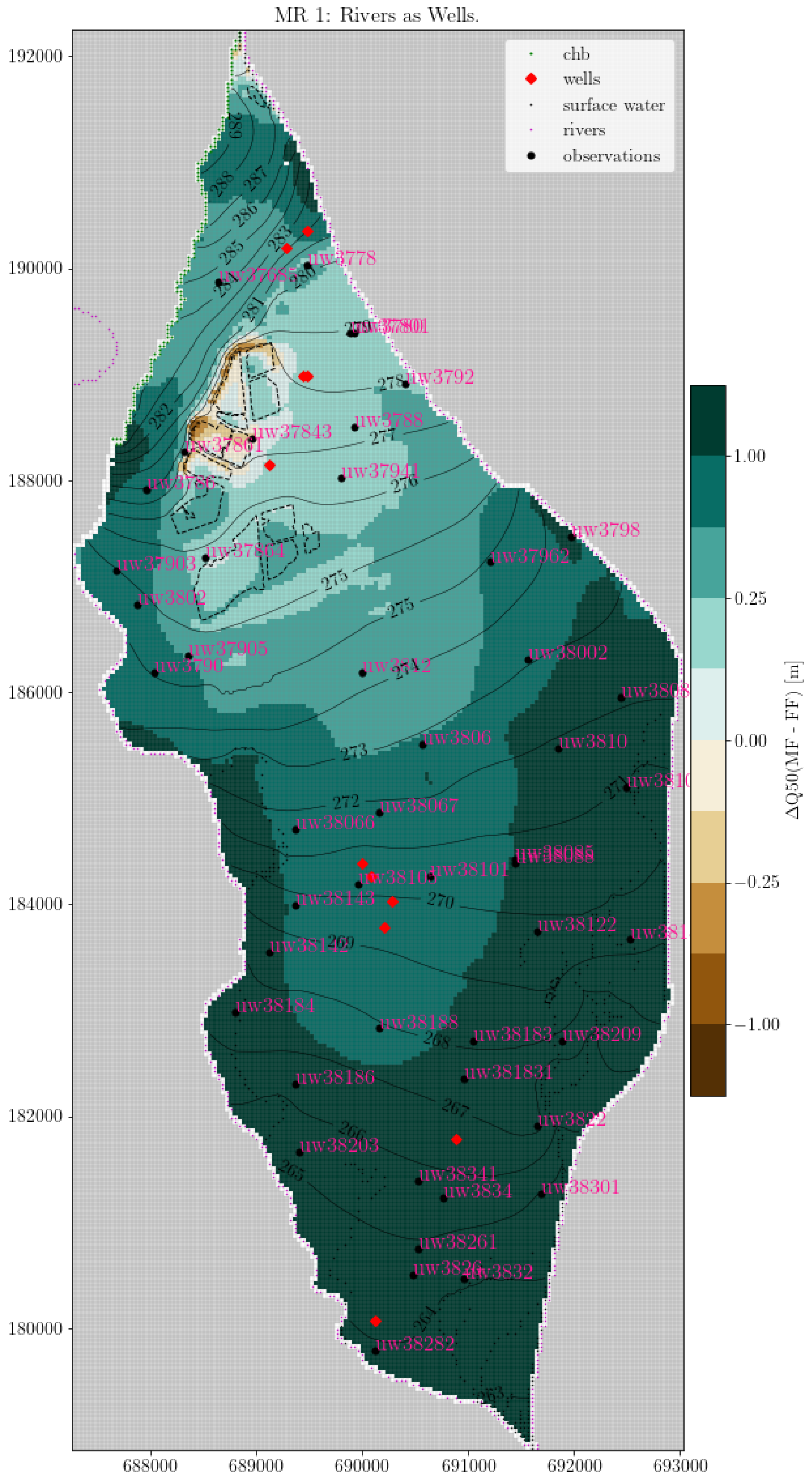


Figure 5.2.: MR 1: Rivers as Wells. Difference map (Q50).

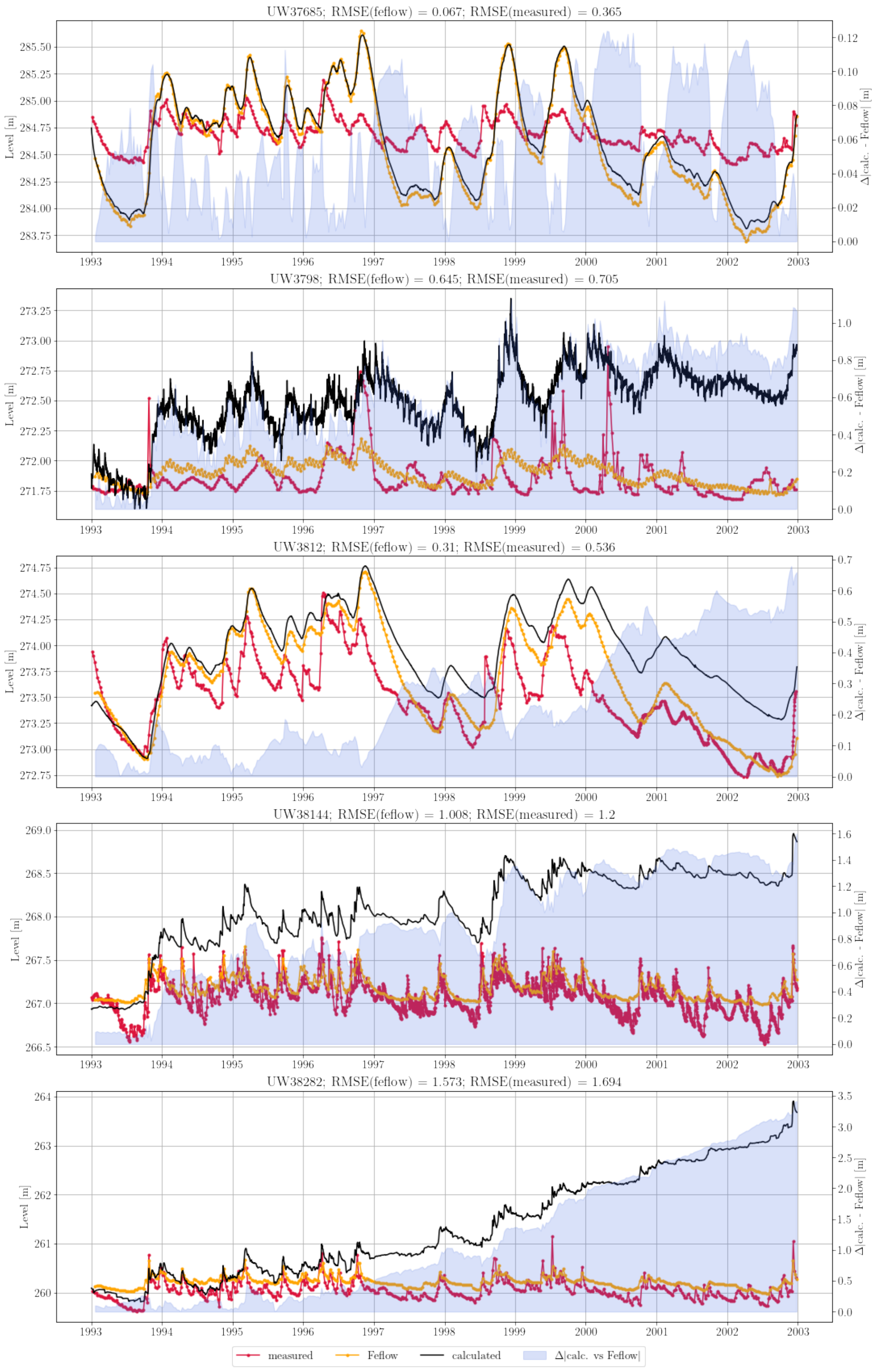


Figure 5.3.: MR 1: Rivers as Wells.

5.2. Modelrun 2: Selected flow direction at Rivers, Interpolated GWL for Recharge

The scatter-plot indicates a relatively high agreement to the FF results. In the central area a cluster of slightly increased results can be observed. The average RMSE of this simulation amounts 0.212 m and thus within the same result spectrum as the Feflow model (RMSE Feflow vs. measured = 0.273 m).

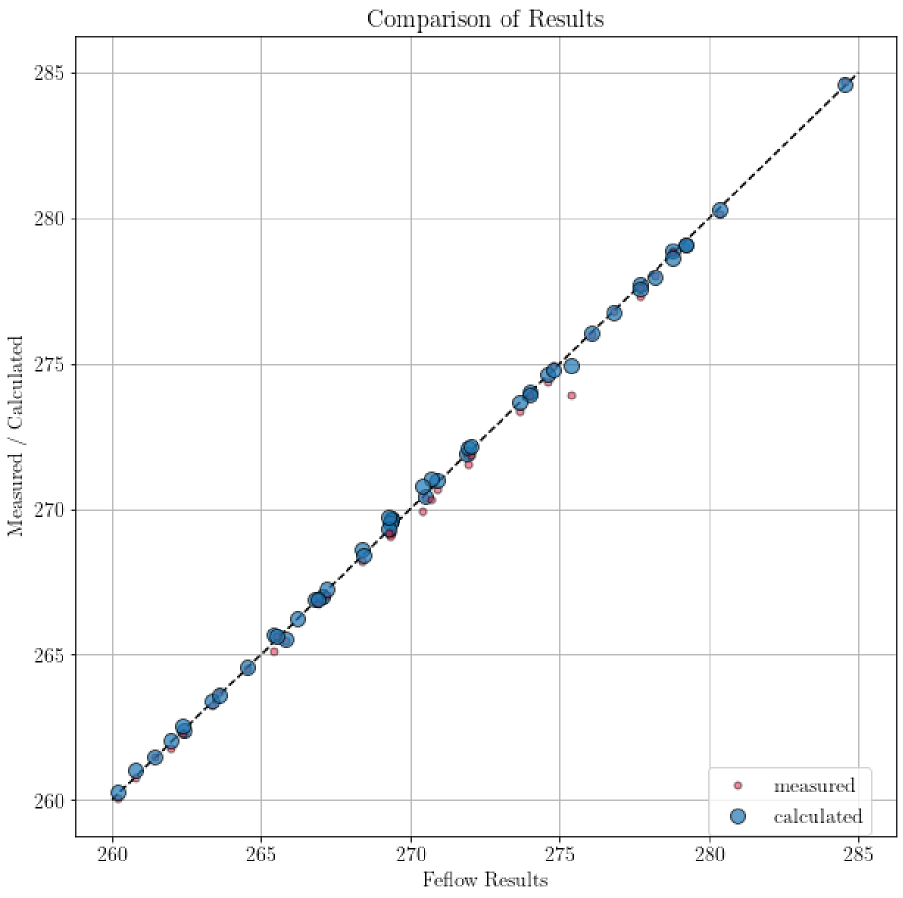


Figure 5.4.: MR 2: Manually selected flow direction. Interpolated GWL for Recharge. Scatter Plot.

The differences map (Fig. 5.5) confirms this: areas calculated significantly too high ($\Delta > 0.5$ m), which can be confirmed by observation wells, are located in the central eastern vicinity to the Mur, respectively to the surface waters. In addition, areas with a water surplus are found along the CHB and in the southern confluence of Mur and Sulm, compared to the FF result. In the upper and lower third, a water deficit is shown in the Q50 differences map in the central area. In particular, in the area of gravel dredgings. In the time-series this phenomenon is only conditionally confirmed (i.e. UW37861, A.2).

Selected observation wells (Fig. 5.6) show a high agreement, only the MF calculation of UW38144 exceeds that of FF by an average of about 0.5 m. The MF curve rises rapidly by about 0.5 meter at the beginning of the first calculation year. The dynamics correspond very closely to that of FF, acting almost as if the curve had been shifted upward by half a meter. In general, it is noticeable that high and longer lasting deviations (delta values) at the observation wells occur in particular when a low water level is calculated.

The budget file (Tab. 5.3) shows a massive increase in water fluxes compared to MR1 with the rivers assigned as wells. Both inflow and outflow are up about +40% (39,000 m^3 per average day). This increase is mainly attributable to the rivers (river inflow +45,000 m^3 and +42,000 m^3 outflow). The CHB budget decreases in inflow and increases ouflow. The water turnover of the recharge is almost exactly the same despite the different calculation method (mean vs. interpolated GWL).

Table 5.3.: MR 2: Budget. Average m^3 / day.

SOURCE	**IN**		**OUT**	
STO	21,226.29	15.90%	21,552.50	16.19%
WEL	0.00	0.00%	4,060.93	3.05%
RIV	61,067.31	45.73%	99,246.19	74.54%
RCH	38,886.84	29.12%	2,666.34	2.00%
CHD	12,345.98	9.25%	5,627.36	4.23%
TOTAL	133,526.41		133,153.33	

Improved results in this area are expected from a calibration with PEST.

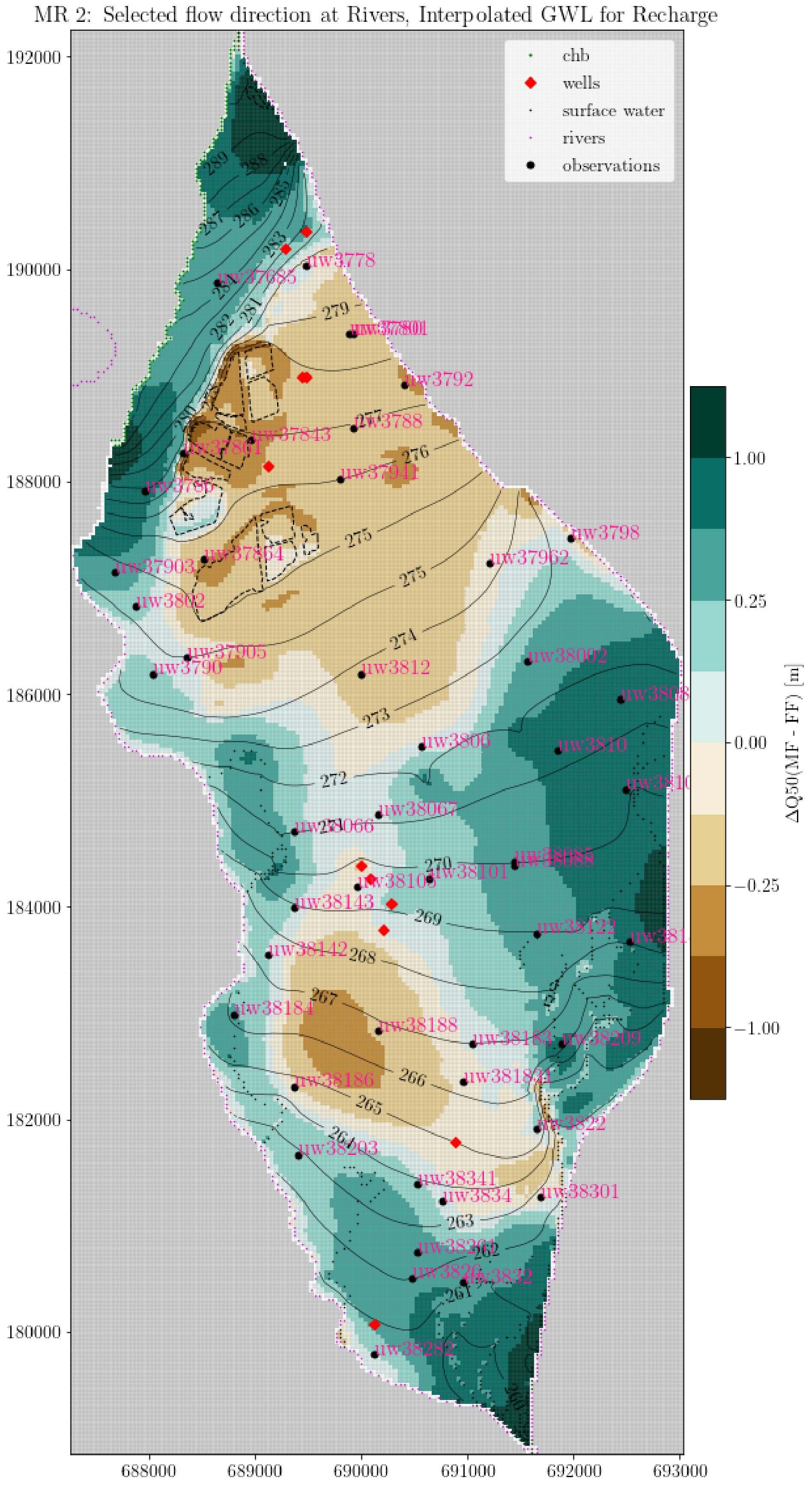


Figure 5.5.: MR 2: Difference map (Q50).

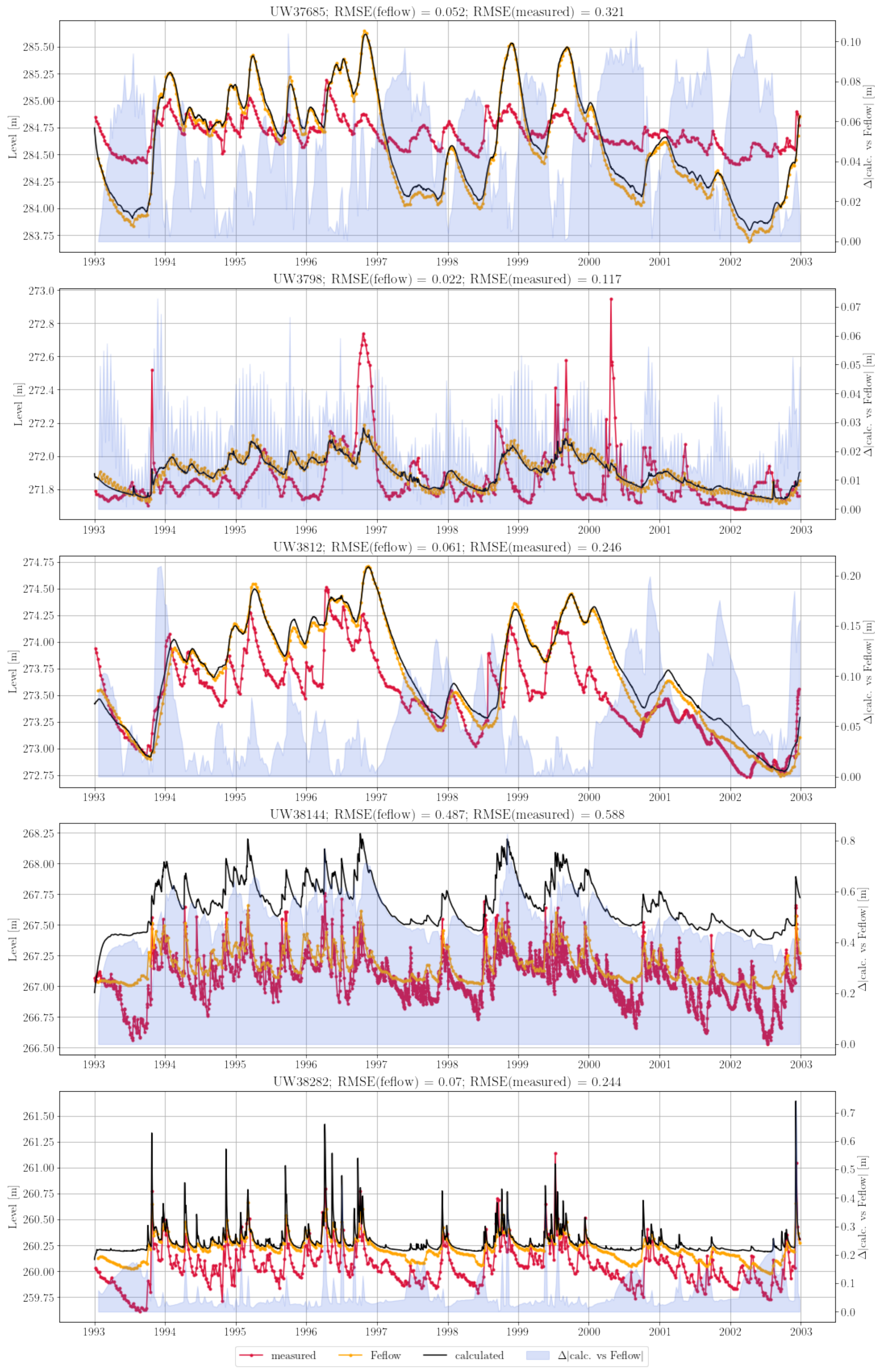


Figure 5.6.: MR 2: Manually selected flow direction. Interpolated GWL for Recharge.

5.3. Modelrun 2a: Calibration

The calibration lowers the overall RMSE to 0.136 m. The scatterplot (Fig. 5.7) shows that the results line up along the identity line (1:1 line) with a few minor outliners. Only four observations exceed an overall RMSE of 0.2 m. The calibrated factors are listed in table 5.4).

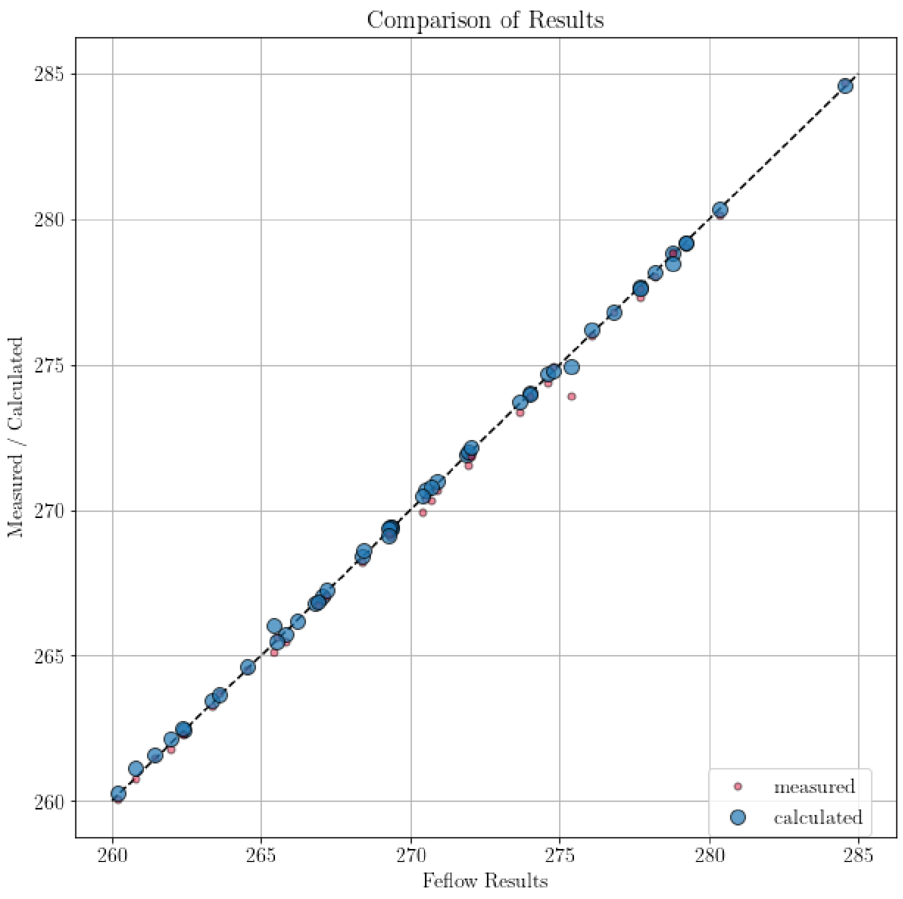


Figure 5.7.: MR 2: Calibrated Leakage factors. Scatter Plot.

Table 5.4.: Calibrated leakage factors.

parameter	area	factor	cells
par 1	inflow	1.433	199
par 2	outflow	1.678	431
par 3	SE-area	13.735	134
par 4	others	0.723	149

Reduced differentials show up in the difference map (Fig. 5.5), especially in the central and eastern model area. Along the CHB, the results apparently remain unchanged, as well as along Lassnitz and Sulm.
Positive differences, i.e. water surpluses, can be observed along the Mur, especially at the roots of the barrages. In addition, a local, higher overestimation can be found in the middle of the surface waters.

The indicated water deficit in the southwestern area (around UW38188) is not confirmed in a comparison with the time-series (A.3). The northernmost model tip is calculated with a lower median water level than that of the FF simulation.

With the exception of UW38144, the selected observation wells show the same pattern as in the uncalibrated model run (Fig. 5.9). The jump at the beginning of the simulation period can still be observed at this hydrograph, but only with 20 cm difference. Low water levels of the current simulation do not fall below the value of 267.2 and thus show the highest discrepancy to the FF calculation.

The budget has increased, compared to the uncalibrated model run (+16% IN, +16% OUT). Mainly due to the turnover in the rivers (+35% IN, +25% OUT). The outflow at the CHB has decreased by -54%.

Table 5.5.: MR 2a: Budget. Average m^3 / day.

SOURCE	**IN**		**OUT**	
STO	20,802.82	15.58%	21,130.22	15.87%
WEL	0.00	0.00%	4,061.70	3.05%
RIV	82,427.21	61.73%	123,969.06	93.10%
RCH	38,803.01	29.06%	2,651.90	1.99%
CHD	12,694.78	9.51%	2,587.89	1.94%
TOTAL	154,727.81		154,400.76	

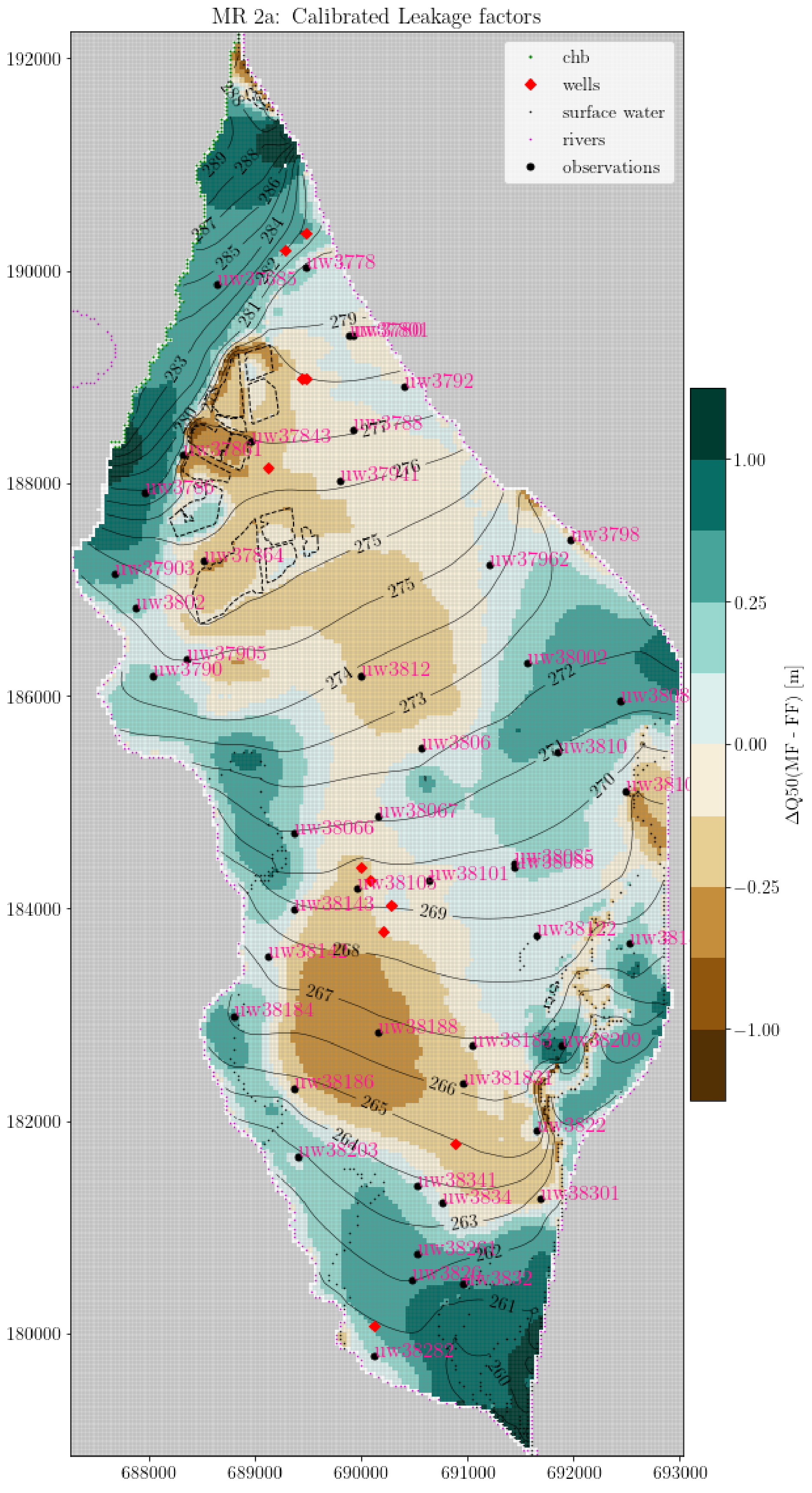


Figure 5.8.: MR 2a: Difference map (Q50).

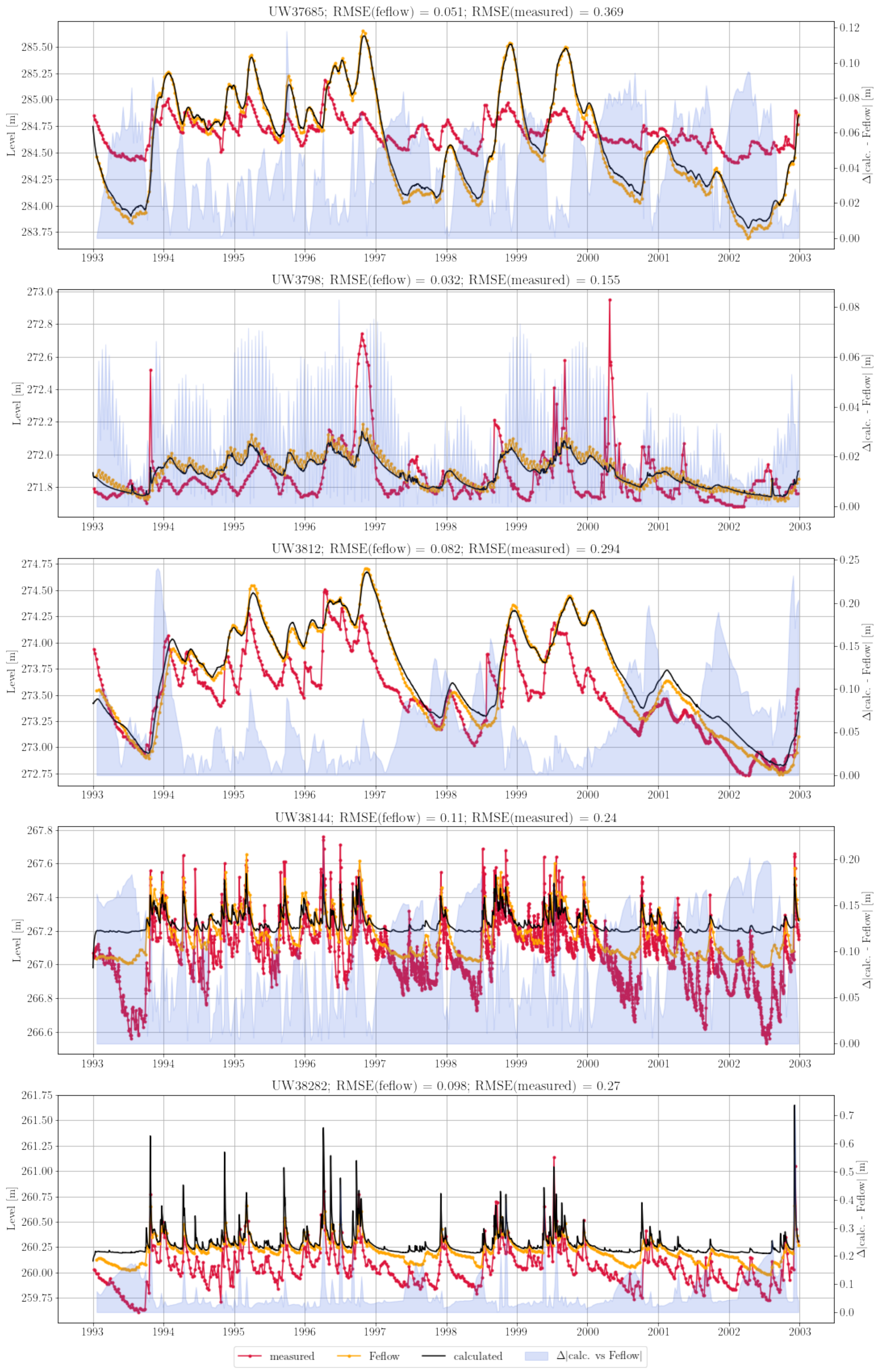


Figure 5.9.: MR 2a: Calibrated Leakage factors.

5.4. Modelrun 3: Loop Modflow

The loop variant results in very high deviations. The scatter plot indicates that, with the exception of a few observation wells located in the north, the calculated values are significantly too high (Fig. 5.10).

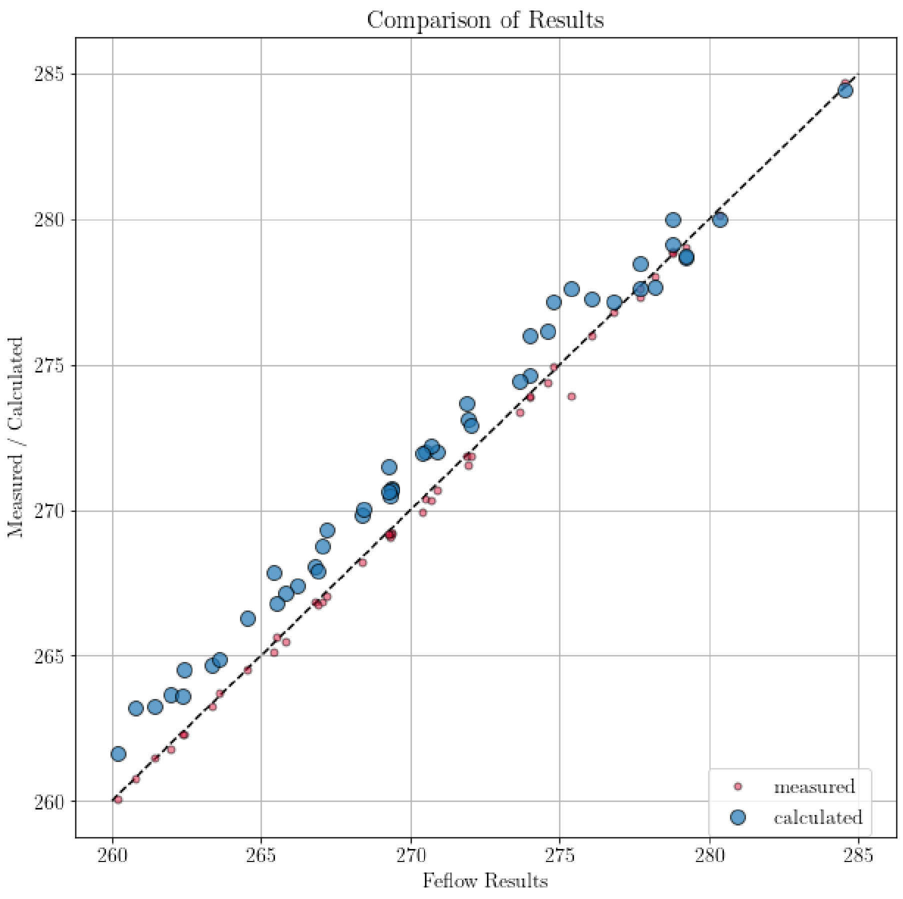


Figure 5.10.: MR 3: Loop Modflow. Scatter Plot.

The time-series show that the massive increase here occurs abruptly within the first time steps (Fig. 5.12 and A.4). Only the northernmost observations (among others UW37685) show an approximate correlation with the FF values. The hydrographs tend to be more jagged, those of previous modelruns appear smoothed in comparison.

The surplus of water is also shown in the difference map (Fig. 5.11). The curvature of the groundwater isohypses indicates effluent conditions along the entire marginal rivers. At the surface waters with fixed water levels the, in previous modelruns rather convex isohypses now appear more concave and indicate a further inflow of water (especially in the southeastern model area).

A map of the median of the first 10 days (not presented here) shows an almost identical picture than over the entire simulation period of 3,650 days, concerning both the isohypses and the differences.

The total budget is not available, since this output file was overwritten with every time step.

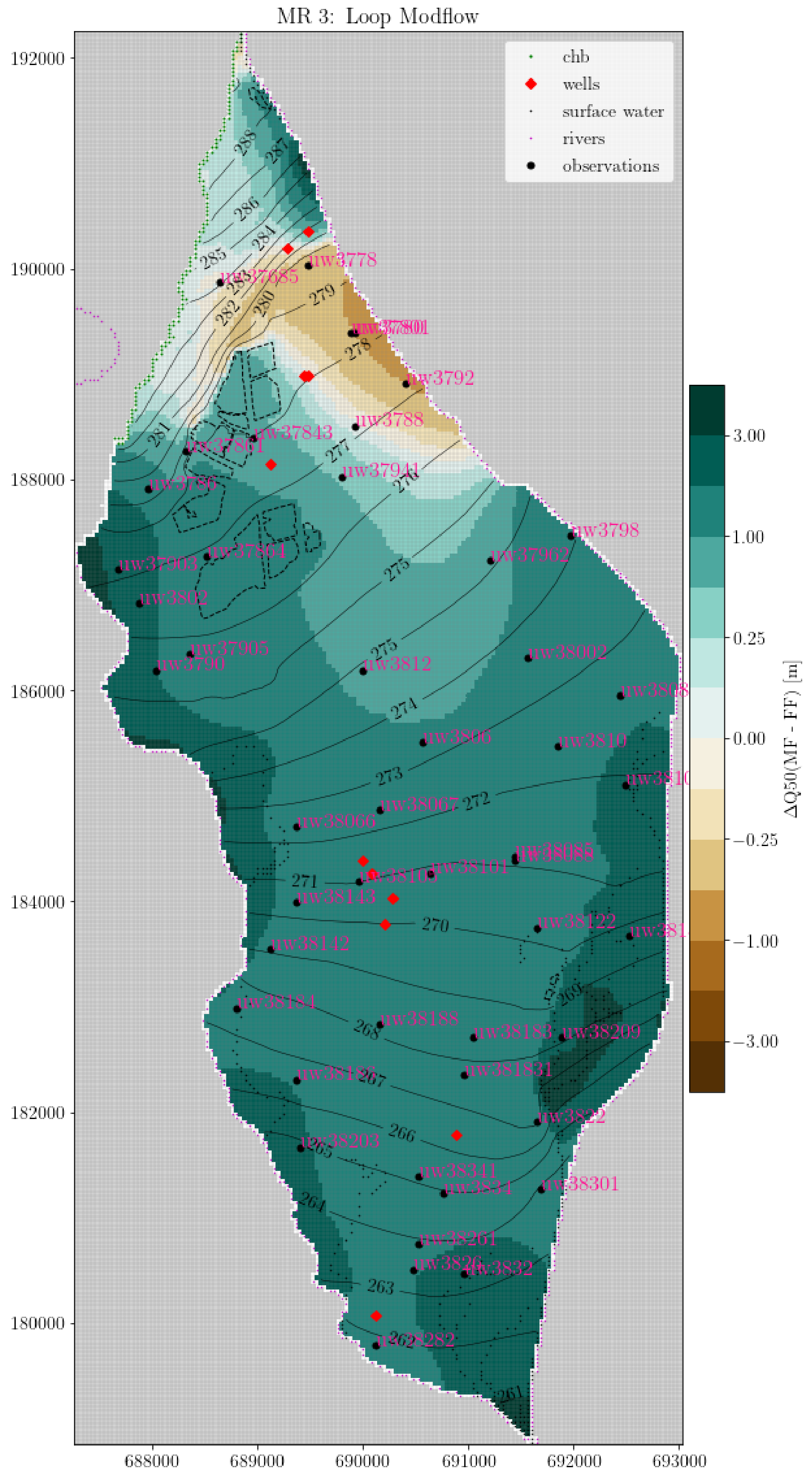


Figure 5.11.: MR 3: Difference map (Q50).

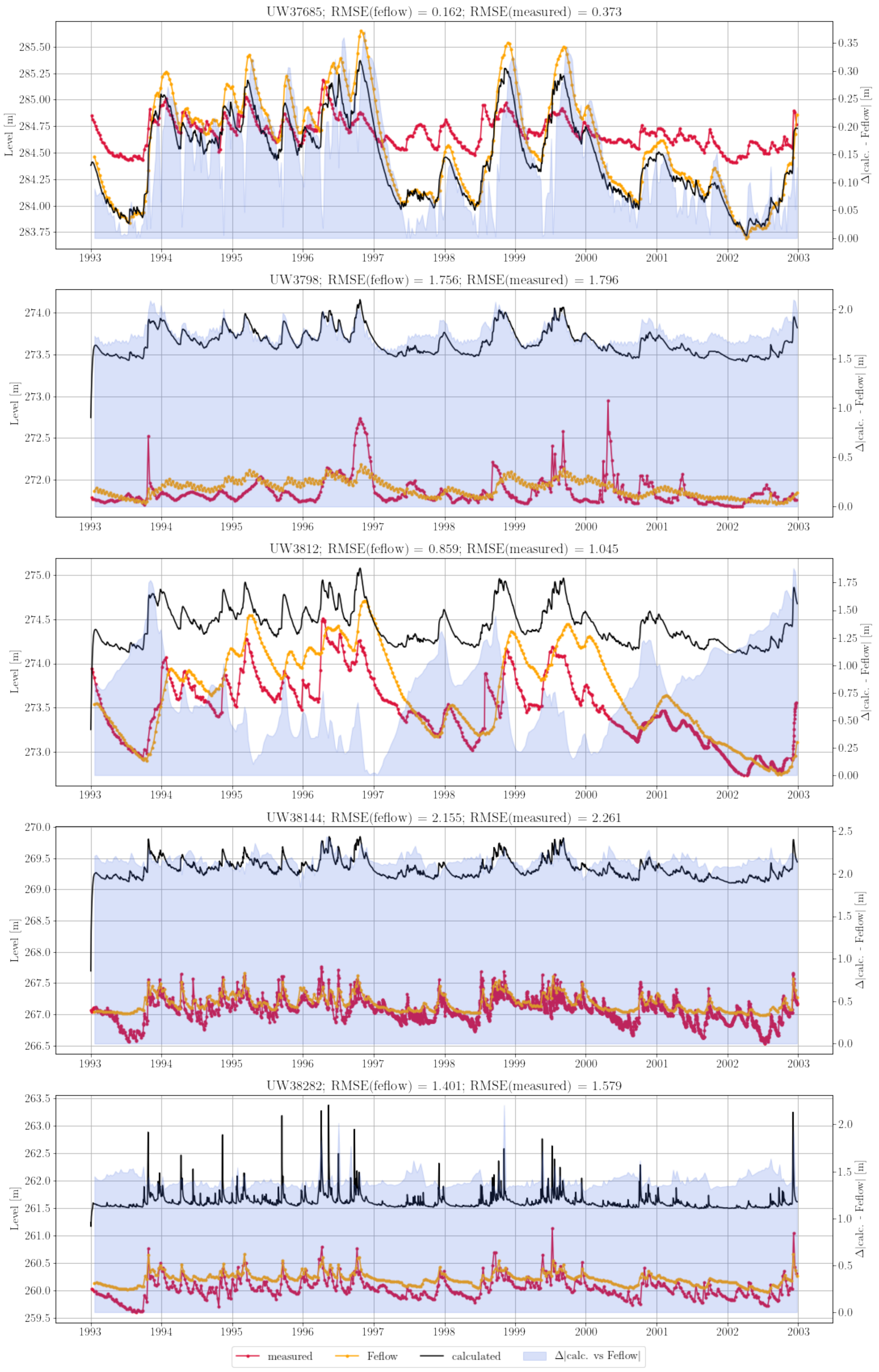


Figure 5.12.: MR 3: Loop Modflow.

6. Discussion

First, the results of the model runs are discussed separately. Following this, common properties or cross-model findings are summarized. Further, possible causes for the problems in the transport modeling are outlined software aspects are debated.

Modelrun 1

By substituting the explicit values of the well- for the river- BCs, one dependency is omitted. In the case of a head-dependent flux (with a river-BC), a systemic response can be expected (higher/lower water turnover at the river-BC cell because of higher/lower hydraulic head), which could conceal occurring discrepancies.

The results clearly show that there is no equilibrium between inflows and outflows. The budget file shows that this imbalance results in an increase in storage (OUT) and thus causes the increase of the groundwater level with time (average daily storage +2,036 m^3). The Feflow budget shows an even higher difference between storage in and out (average daily storage +5.606 m3). It remains unclear to what extent or whether the recharge rate, which can only be derived indirectly, influences the storage (see 3.4).

Only three sources can be responsible for the surplus of water:

- the CHB, because of discretization problems
- due to the recharge calculation by means of the used averaged groundwater level, or because a higher recharge occurs due to the larger area resulting from the discretization.
- there is in total already more water in the system at the beginning of the simulation (amplified by a larger model area): it shows that the initial groundwater level is on average 0.07 m higher per cell than the Q50 water level of the FF calculation. All in all, taking into account the specific yield, this results in a plus of 150,407 cubic meters. This plus of water could accumulate in the south, following the morphological gradient.

The comparison with the FF budget (see 3.3) shows that the inflow at the CHB is about the same in both models. However, the outflow value of the MF calculation is more than three times higher than that of the FF calculation. This increase is thought to result from a bypass in the northernmost model tip: water flowing into the aquifer from the Mur drains directly through the CHB. Therefore, in total (IN minus OUT), less water enters the system via the CHB, compared with the FF budget (-2,700 m^3).

The recharge rate is likely to be in the same order of magnitude as the FF calculation, but the data is not clear enough to make a reliable assessment (compare 3.4). It seems unlikely that the recharge rate is the only reason for the surplus of water.

Finally, it is acknowledged that the reason for the water surplus cannot be clearly traced back.

Modelrun 2 and 2a

Modelrun 2 represents an *as direct as possible* transfer of the model parameters from Feflow to Modflow in a classical approach and taking into account the differences and limitations. The results presented in the difference map of MR2 and the time-series are in line with the expectations for the applied methods and the discretization differences, respectively: overall a relatively high agreement can be observed, with outliners in problematic areas. With the calibration (MR 2a), the results improve accordingly. It is expected that an increase of the calibration parameters - e.g. finer zoning of the leakage factors - would lead to an even higher agreement with the FF results.

The significant increase in the water turnover of the rivers, which can be seen in the budget file, is particularly remarkable. The calibration mainly leads to an increase in the leakage factors and consequently to a further increase in the budget (the ratio of inflow to outflow increases only slightly). It cannot be clearly determined whether this surplus of water in the rivers budget is the result of fluctuations in the marginal areas (in the vicinity of the rivers, i.e. the Mur), flowing in and out again within short periods of time, or whether the aquifer is flowed through over a large area. The fact that only one leakage factor was assigned - in the majority the higher outflow factor - could speak in favor of the first theory. When river water levels fall, this more water flows back into the river.

Another cause in this context could be the non-continuous definition of the river-BC: In Feflow, the river-BC cells represent the outer model boundary. In the current Modflow model, however, some river-BC are surrounded by other cells which are not linked to a boundary condition (see Fig. 4.4). These cells could act as a buffer, thus increasing the water inflow or outflow, because the river-BC cells are fed by more neighboring cells.

A further possibility for the increased water exchange is the discretization. In FF, the marginal areas along the rivers are rather fine discretized. The resolution with 40 x 40 m cells in MF can partially result in harder jumps (upwards as well as downwards) in the river water stages and thus higher gradients. Consequently, the volumetric exchange rate increases, which in sum can potentially be compensated by positive and negative deviation.

A small portion of the Mur inflow drains directly through the CHB, whereby this proportion decreases due to the calibration.

The difference map indicates a surplus of water along the CHB and remains unchanged by the calibration. Possible reasons for this surplus are:

- a pure problem of presentation: in the 40 x 40 m cells, a (high) divergence results from the FF median, averaged over the entire cell area; amplified by the steep aquifer gradient
- the aquifer bottom has a steep gradient in this area: due to discretization, the CHB could be minimally displaced, what could lead to increased water inflow
- the calculation method of the recharge rate is relatively inaccurate in the marginal areas or the area relevant for the recharge is increased by the discretization

None of the nearby observation wells record a hydrograph that is calculated (significantly) too high, indicating a representation problem. The budget of the CHB shows a much lower turnover than in the FF model.

Modelrun 3

The erratic and strong deviation in the loop variant are surprising, because this method actually transfers all dependencies from the Feflow calculation to Modflow. However, with the implementation of groundwater level-dependent recharge and leakage factors, positive feedback effects may result: The recharge generally is highest at the surface of the terrain and decreases with depth. When the groundwater level rises, the recharge rate increases as well. Nevertheless, the sudden increase at the beginning of the simulation period cannot be solely due to an increased recharge rate.

The leakage factors for out- and inflow differ by up to 10,000. If, for example, an out- instead of an inflow factor were incorrectly assigned, up to 10,000 times more water could enter the system due to the linear relationship between leakage factor and resultant volume flow. In the first few time steps far too much water gets into the system, which also results in changed groundwater dynamics, i.e. flow directions. The system does not return to a balance after these first time steps.

It is assumed that areas in which rivers and surface waters or drainages (all defined as RIV) are located close together, are responsible for the strong and rapid groundwater level rise. For example, two directly adjacent river-BC cells in the southeast model area show very different water stages: the stage of the transient cell averages about 271.65 m (with very little fluctuation), while that with a fixed stage is 267.40 m. A complex dynamic of accumulation could develop here, which, however, cannot be traced exactly in retrospect. Such a situation has only been noticed very locally in the area between the Mur and the accompanying drainages. Presumably, in the case of surface waters, a strict division into areas with purely effluent and variable flow direction and a corresponding definition as a drainage (DRN Modflow package), which only allows water to flow out, would be more suitable in this modelrun.

The long calculation times (about 42 h) make troubleshooting difficult, especially in combination with the fact that the calculation results (especially budget files) are overwritten with each time step. It is rumored that a future Modflow version will allow a dynamic groundwater level interrogation, which would make this relatively complex and time consuming method obsolete. Therefore, it was decided that the time required to optimize this method was not commensurate with the results that might be expected.

Cross-model Findings

Modelrun 1 already shows that the water dynamics in the current Modflow model are not replicated in exactly the same way as in the Feflow model. This MR should in principle be regarded as a test run, as it uses results from the Feflow calculation, and it fulfills this purpose.

One main reason for this is certainly the different **discretization**: The coarser discretization leads to increased residuals in areas with a higher gradient. These areas are mainly located at the the model boundaries. In addition the larger model area in the Modflow model results in a higher total recharge. Further there already is more water in the system at the beginning of the simulation (IC is on average higher per cell than the Q50 water level of the FF calculation; see discussion of MR1). The non-continuous definition of the river-BC, as well as their spatial discretization (described in the discussion of MR 2) could also be problematic here. Although these are not large amounts when considered individually and over the entire model area, they can make an overall difference. Determining the significance of the problematic parameters or balancing them is even more difficult.

Outflow via rivers represents the dominant sink in the model domain, thus the adoption of **leakage factors** is a key point in model translation. A separate definition of inflow and outflow leakage factors has advantages. Especially at complex sites, for example in the area of barrages, variable water dynamics can be considered in this way. In particular, the different conditions can be directly addressed in the conceptual model. If only one leakage factor is available here, it must either be calibrated and determined for an average situation, or defined as a dependency on other conditions, e.g. the river water level, in a time-variable way. Nevertheless, modelrun 2 and its calibration (MR 2a) prove that approximately equal groundwater levels can be calculated with one available leakage factor. With a further, more fine-grained calibration, an even higher agreement would certainly be achieved.

Unfortunately, modelrun 3 does not provide satisfying results - a higher level of agreement was expected. The difficulties of this model run are thought to lie mainly in the river-BC definition (especially its leakage factors). Concerning this point, the modelrun highlights the conceptual weaknesses of the model translation. As a result, this modelrun unfortunately cannot be used for an evaluation of the method to obtain a GWL-dependent recharge rate.

The comparison of the modelruns 1 and 2 shows that **recharge** plays a central role as a source of water, but the temporal variability of depth-dependency has low sensitivity: there are no significant changes in the budget when the calculation method is changed from a mean- to the interpolated groundwater level.

It must also be mentioned here that the budget of the recharge in Feflow is unclear because this is calculated by the external interface manager and is therefore not included in the total budget (compare 3.4). Thus, an exact evaluation of this parameter is hardly possible.

Transport

The transport simulation has so far only provided inadequate results. Multicausal reasons are suspected behind the problems in the transport simulation: an input error in the Source Sink Mixing Package (Modflow) (SSM) package, what is responsible for the recharge coupled nitrate input and a bug in the Modflow 6.2.1 program.

The **SSM package** of the GWT model in Modflow 6 only allows concentrations as input values (MT3D or FF also support masses). To obtain a concentration, the leached nitrate mass is divided by the corresponding recharge value, both normalized to a uniform area. Very small recharge quantities lead to extremely high concentrations, which in rare cases would also become negative. Since this is neither desirable nor practically possible, it was prevented by forming the absolute value. This conversion offers a number of sources of error. Since no official documentation for the Modflow's GWT model was published yet, troubleshooting is difficult.

The official channel of the USGS on GitHub lists a **problem concerning the GWT model of Modflow 6.2.1** and large binary budget files. According to this issue, Modflow cannot read budget files larger than 2 GB and results in error messages (Issue #720: `https://github.com/MODFLOW-USGS/modflow6/issues/720`). The budget file of the current models has about 14 GB in size.

A switch to MT3D for the transport simulation was not possible due to a compatibility problem between Flopy, Modflow 6 and MT3D (see 5).

Software Aspects

Modflow itself and especially in combination with FloPy has proven to be a flexible and stable tool for groundwater flow modeling. The model construction works very fast and robust, as long as Python is mastered. For tricky questions, solutions can be found fairly quickly due to the large number of scientific packages and functions that are freely available for the Python environment. Numerous interfaces enable a wide variety of output and input formats. Nevertheless, a GUI would sometimes be practical, especially for complex models, e.g. to be able to examine spatial parameter distribution more quickly and easily.

In the course of the work, the impression has grown that the transport part of Modflow still requires some adjustments and, above all, a proper documentation in order to use it as a practical and reliable tool. In the context of FloPy, the question of applicability to everyday technical challenges arises. Individual parameter changes in the modelrun, for example, require intervention in several different files or sections, which means that error-proneness is high. With long calculation times, this can lead to annoying idle times. In sum, a long training period for the user is required. The approximately 22,000 files that are written per model run are not practical to handle and with a size of almost 30 GB per model run (if transport is to be calculated independently), the amount of data also causes problems.

7. Conclusion

The present work investigates to what extend the direct translation of the Feflow specific functions into a Modflow setting is possible and reasonable in terms of practicability. It can be concluded that some functions could directly be transferred to Modflow, however difficulties arose with the attempt to replicate some Feflow specific features.

The best results are obtained in the flow simulation, when the groundwater level-dependencies are not taken over directly, but are calculated by using alternative and comparatively simpler approaches. Especially the manual selection of the leakage factors, and calculation of the recharge rate using an interpolated or an averaged groundwater level have resulted in a high agreement with the Feflow calculation and are additionally improved by a calibration of the leakage factors.

The work has further shown that in the given hydrogeological setting, the quality of the results or the agreement with the Feflow model depends above all on the interaction with the rivers, and thus on the leakage factors. The rivers represent the dominant sink in the model domain. The applied methods for calculating the recharge rate (by interpolated and averaged groundwater level) yielded almost equal results and indicate a comparatively low sensitivity of this parameter, although it represents the prevailing source of water. It is concluded that the recharge rate has little variability in the range of groundwater level fluctuations. The coarser discretization leads to localized increased residuals in areas with higher gradients. High gradients occur mainly along the model margins (partly due to the river barrages), where additionally river- and constant head-BC are located and are partly characterized by low aquifer thicknesses. In total this leads to the fact that these areas are particularly susceptible to residuals, which however can be significantly improved and narrowed down by a calibration of the leakage factors. It is expected, that a refined discretization in this area would lead to further improvements. Considering the entire model domain in a large scale, the coarser discretization seems to have little effect on the calculated groundwater levels. The comparison of the water budget between Feflow and Modflow reveals differences, but cannot be assessed with certainty, because the budget of Feflow is not provided completely, since the recharge is calculated via the external interface manager and is therefore not included.

However, as already mentioned, not all features could be transferred directly to Modflow. Thus, the attempt to emulate the calculation methods of Feflow, by looping Modflow and query the groundwater level per time step, did not yield the expected (comparable) results. The evaluation of this method is complicated because of the interaction of several parameters, which influence each other. According to experts opinions, a future Modflow version might include the possibility of a dynamic groundwater level query, which would facilitate the transfer of the special Feflow features.

In order to evaluate the chosen methods in detail, a synthetic, more manageable and smaller model would be appropriate, since dependencies could be more easily switched on and off, and the respective effect could be better monitored. For a pure reproduction of the results it is assumed in retrospect that a more conceptual approach, based on the available data from Feflow, would lead to better results. I.e. to put the focus less on the uniformity of the model parameters than on the representation of the hydraulic situation and also to take advantage of the offer and properties of the different Modflow packages.

Since this work could not determine the exact causes of the problems in transport simulation, only insufficient results were obtained, which therefore cannot be evaluated. There is still no official documentation available for the transport simulation.

In summary, the present work has shown that the two programs with the differently available functions provide comparable results. Although the direct translation of some Feflow-specific features has its limitations. Especially for dynamic groundwater level queries, the currently available version of Modflow is less practicable. The choice of the optimal software package depends on the research question as well as the characteristics of the aquifer.

8. Bibliography

Anderson, Mary, William Woessner, and Randall Hunt (2015). *Applied Groundwater Modeling (Second Edition)*. Academic Press. DOI: https://doi.org/10.1016/B978-0-08-091638-5.00019-5.

Bakker, Mark, Vincent Langevin, J. D. Hughes, J. T. White, A. T. Leaf, S. R. Paulinski, J. D. Larsen, M. W. Toews, E. D. Morway, J. C. Bellino, J. J. Starn, and M. N. Fienen (Feb. 2021). “FloPy v3.3.4 — release candidate”. In: *U.S. Geological Survey Software Release*. DOI: http://dx.doi.org/10.5066/F7BK19FH.

Bedekar, V., E.D. Morway, C.D. Langevin, and M. Tonkin (2016). “MT3DMS updated with new and expanded transport capabilities for use with MODFLOW”. In: *U.S. Geological Survey Techniques and Methods 6 A53*. DOI: http://dx.doi.org/10.3133/tm6A53.

Chapman, Steven W., Beth L. Parker, Tom C. Sale, and Lee Ann Doner (2012). “Testing high resolution numerical models for analysis of contaminant storage and release from low permeability zones”. In: *Journal of Contaminant Hydrology* 136-137, pp. 106–116. ISSN: 0169-7722. DOI: https://doi.org/10.1016/j.jconhyd.2012.04.006.

Dehotin, Judicaël, R. Vázquez, Isabelle Braud, S. Debionne, and Patricia Viallet (Feb. 2011). “Modeling of Hydrological Processes Using Unstructured and Irregular Grids: 2D Groundwater Application”. In: *Journal of Hydrologic Engineering* 16, pp. 108–125. DOI: 10.1061/(ASCE)HE.1943-5584.0000296.

DHI-WASY GmbH (n.d.). *FEFLOW Online Help*. URL: www.feflow.info. accessed in May 2021.

Diersch, Hans-Joerg (2014). *FEFLOW - Finite Element Modeling of Flow, Mass and Heat Transport in Porous and Fractured Media*. Springer-Verlag Berlin Heidelberg. DOI: 10.1007/978-3-642-38739-5.

Digitaler Atlas Steiermark (2021). *A17 - Geoinformation*. URL: http://www.gis.steiermark.at. accessed at 15.04.2021.

Doherty, John E. (2021). “PEST: Model-Independent Parameter Estimation, User Manual”. In: *Watermark Numerical Computing*.

Fank, J., A. Jawecki, H. P. Nachtnebel, and Zojer H. (1993). “Hydrogeologie und Grundwassermodell des Leibnitzer Feldes”. In: *Berichte der wasserwirtschaftlichen Planung* 74/1.

Feichtinger, F. (1998). “STOTRASIM - Ein Modell zur Simulation der Stickstoffdynamik in der ungesättigten Zone eines Ackerstandortes”. In: *Schriftenreihe des Bundesamtes für Wasserwirtschaft* 7.

Hughes, J.D., C.D. Langevin, and E.R Banta (2017). “Documentation for the MODFLOW 6 framework”. In: *U.S. Geological Survey Techniques and Methods*. DOI: https://doi.org/10.3133/tm6A57.

Jazayeri, Amir and D. Werner Adrian (2019). “Boundary Condition Nomenclature Confusion in Groundwater Flow Modeling”. In: *Groundwater* 57-5.

Kinzelbach, Wolfgang (1987). *Numerische Methoden zur Modellierung des Transports von Schadstoffen im Grundwasser*. Habilitationsschrift Universität Stuttgatt1.

Konikow, L.F., D.J. Goode, and G.Z Hornberger (1996). “A three-dimensional method of characteristics solute-transport model (MOC3D)”. In: *U.S. Geological Survey Water-Resources Investigations Report 96–4267*.

Langevin, C.D., J.D. Hughes, E.R. Banta, A.M. Provost, R.G. Niswonger, and Sorab Panday (Feb. 2021). “MODFLOW 6 Modular Hydrologic Model version 6.2.1”. In: *U.S. Geological Survey Software Release*. DOI: https://doi.org/10.5066/F76Q1VQV.

Mach, J., G. Rock, G. Klammler, J. Draxler, H. Kupfersberger, and J. Fank (2019). *Gekoppelte Grundwasserströmungs- und Nitrattransportmodelle in den Grundwasserkörpern Leibnitzer Feld und Unteres Murtal*. JR-AquaConSol, p. 167.

Matiatos, Ioannis, Emmanouil A. Varouchakis, and Maria P. Papadopoulou (2019). “Performance Evaluation of Multiple Groundwater Flow and Nitrate Mass Transport Numerical Models”. In: *Environmental Modeling & Assessment* 24. ISSN: 1573-2967. DOI: 10.1007/s10666-019-9653-7.

MODFLOW Development Team Revision 27bb36e1 (2020). *MODFLOW 6 Documentation*. URL: https://modflow6.readthedocs.io/en/latest/mf6io.html.

Panday, Sorab, C.D. Langevin, R.G. Niswonger, M. Ibaraki, and J.D. Hughes (2013). “MODFLOW–USG version 1: An unstructured grid version of MODFLOW for simulating groundwater flow and tightly coupled processes using a control volume finite-difference formulation”. In: *U.S. Geological Survey Software Release*. DOI: https://pubs.usgs.gov/tm/06/a45.

Trefry, Mike G. and Muffels Chris (2007). *FEFLOW: A Finite-Element Ground Water Flow and Transport Modeling Tool*. National Ground Water Association. DOI: 10.1111/j.1745-6584.2007.00358.x.

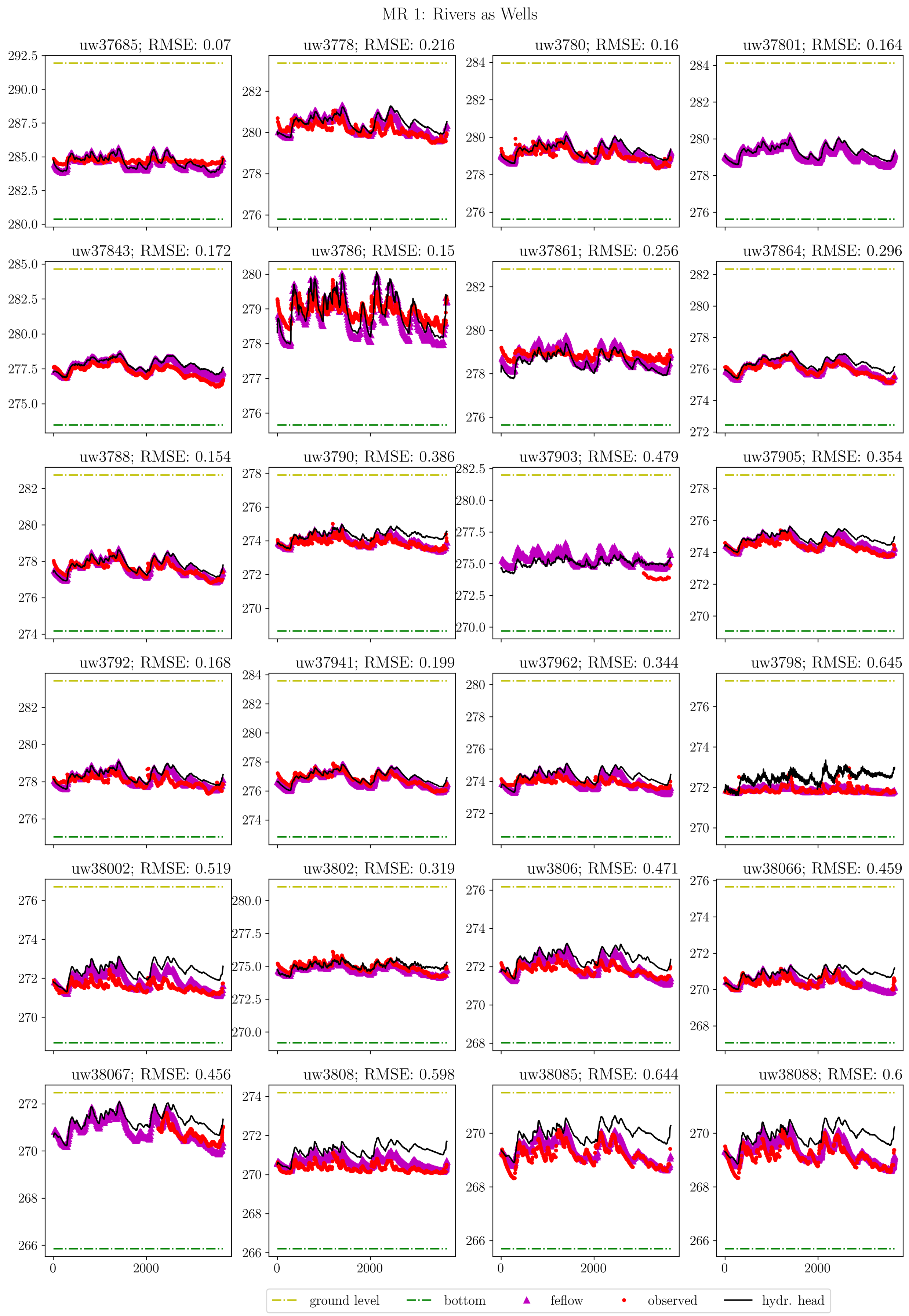
Wang, Herbert and Mary Anderson (1982). *Groundwater modelling with finite difference and finite element methods*. Elsevier Publishing.

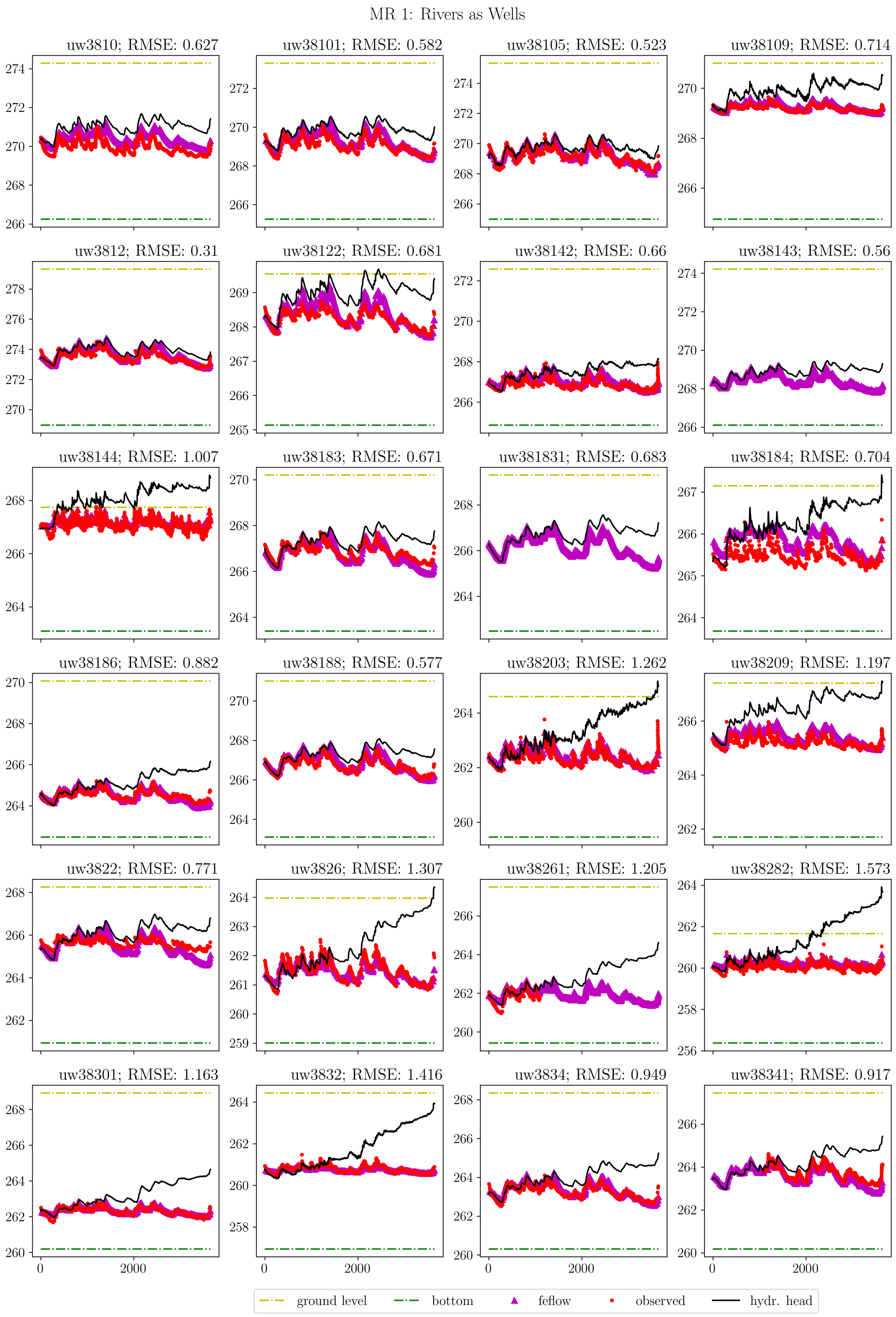
White, Jeremy T., Michael N. Fienen, and John E. Doherty (2016). “A python framework for environmental model uncertainty analysis”. In: *Environmental Modelling & Software* 85, pp. 217–228. ISSN: 1364-8152. DOI: https://doi.org/10.1016/j.envsoft.2016.08.017.

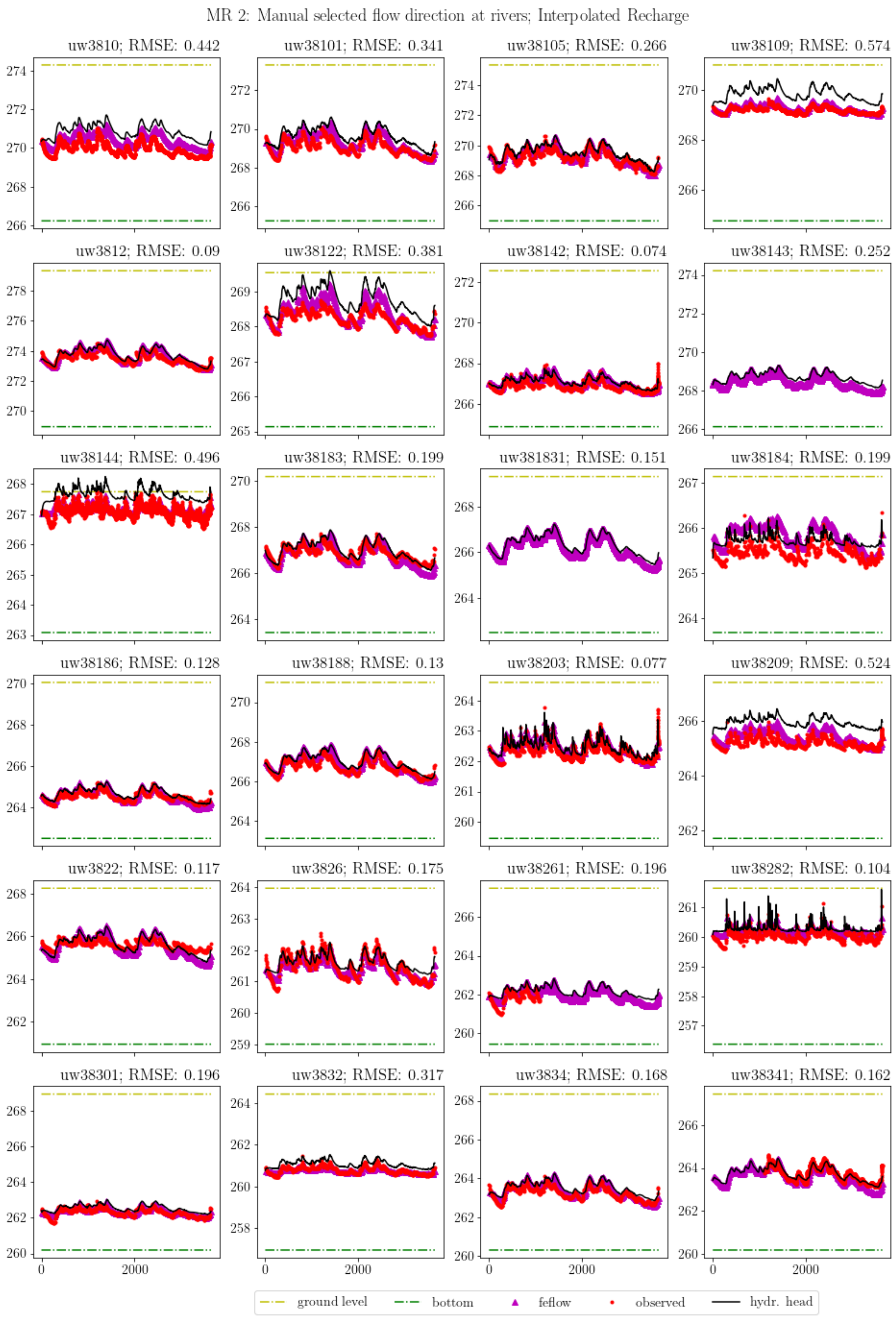
Zheng, C. and P.P. Wang (1999). “MT3DMS: A Modular Three-Dimensional Multispecies Transport Model for Simulation of Advection, Dispersion and Chemical Reactions of Contaminants in Groundwater Systems”. In: *US Army Corps of Engineers-Engineer Research and Development Center*.

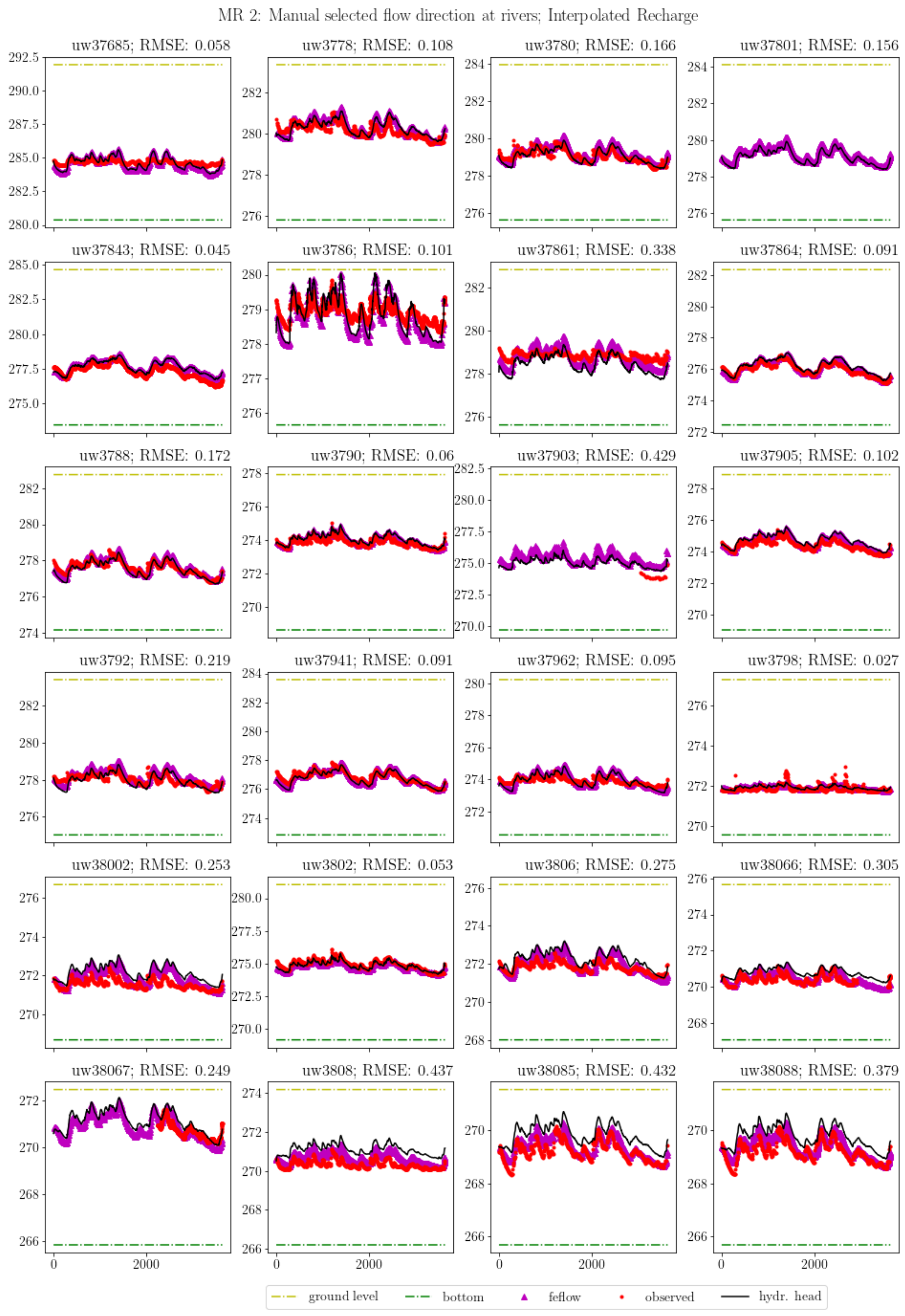
A. Appendix: Model Results

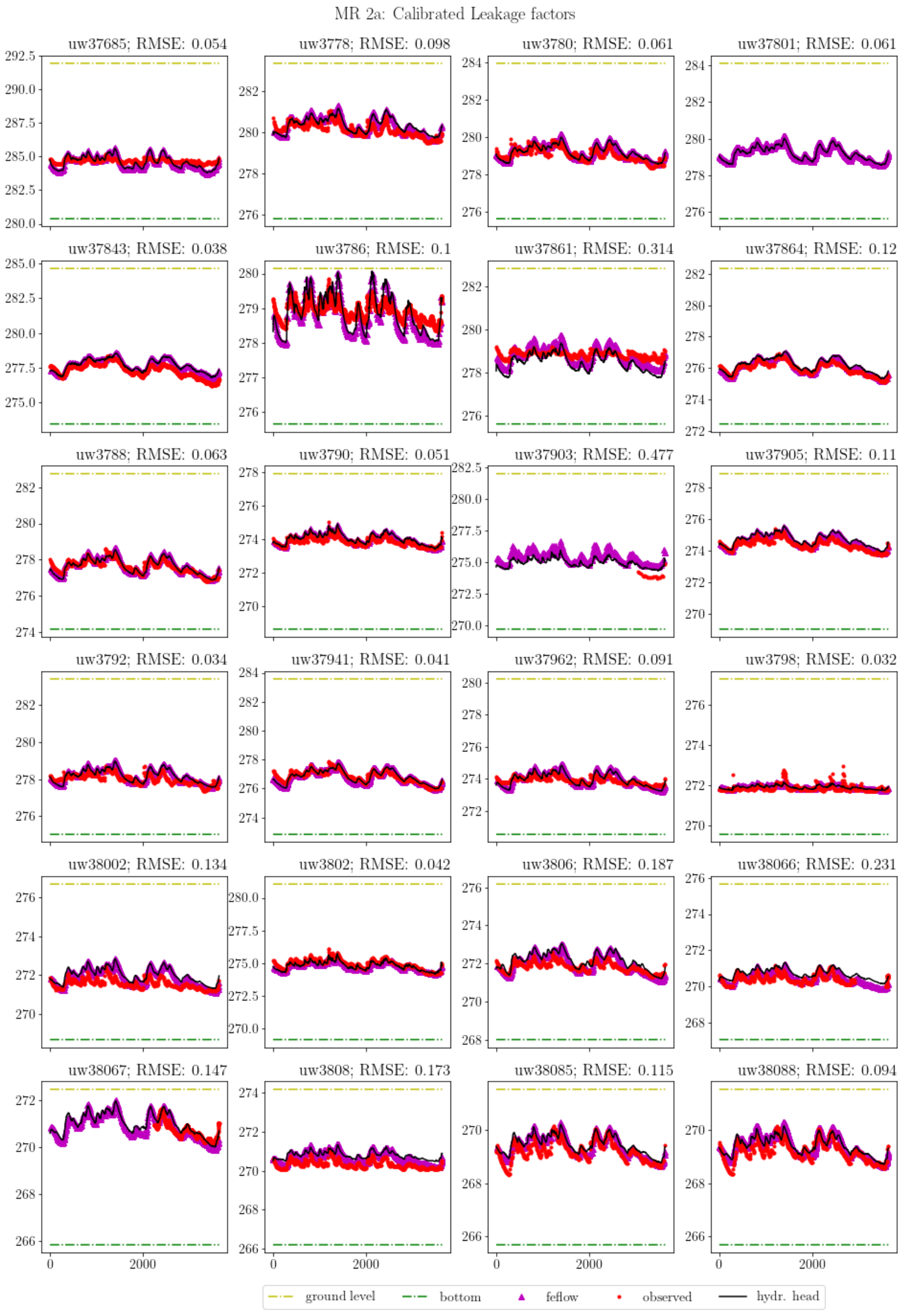
Contents

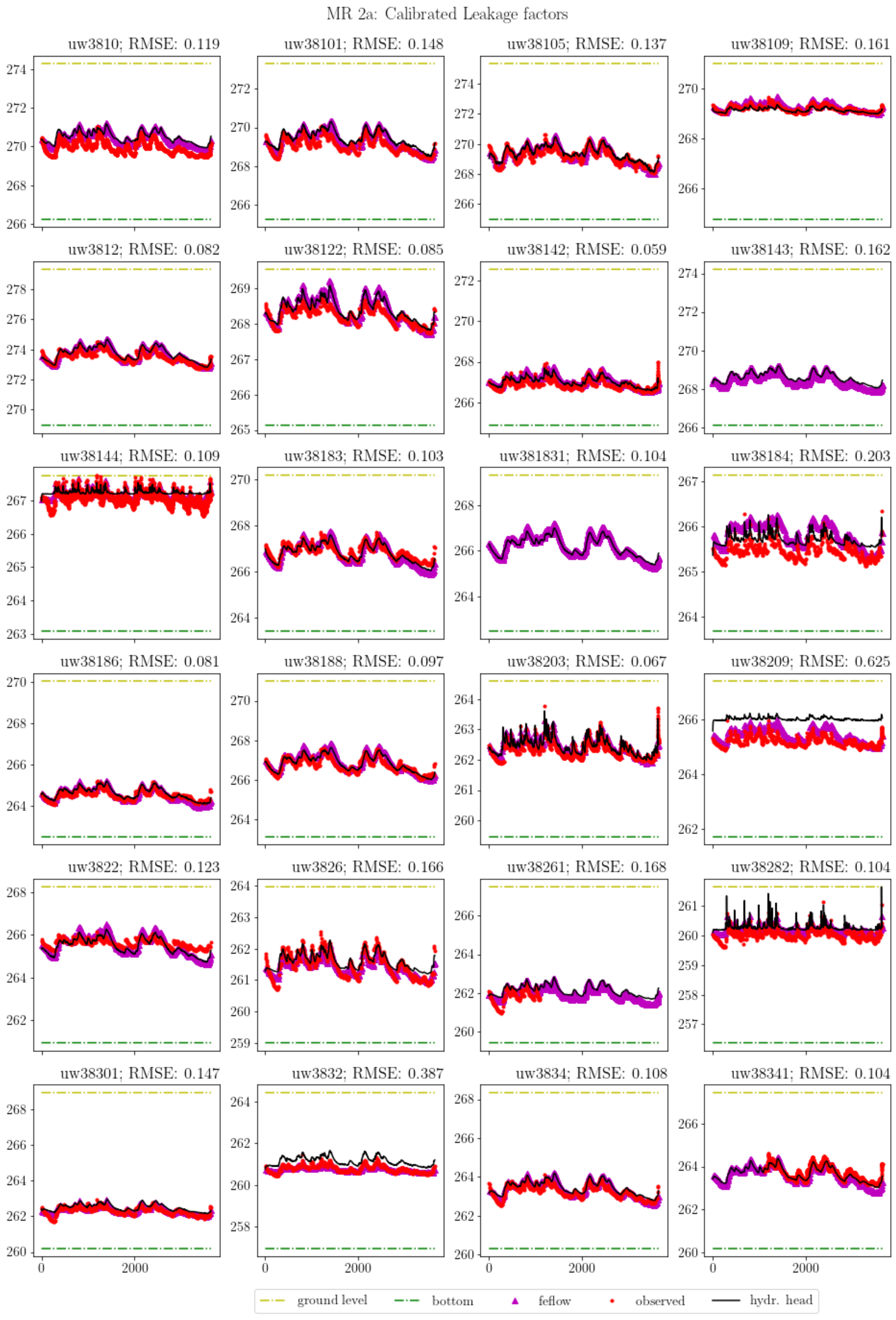

MR 1: Rivers as Wells
uw37685; RMSE: 0.07
uw3778; RMSE: 0.216
uw3780; RMSE: 0.16
uw37801; RMSE: 0.164
uw37843; RMSE: 0.172
uw3786; RMSE: 0.15
uw37861; RMSE: 0.256
uw37864; RMSE: 0.296
uw3788; RMSE: 0.154
uw3790; RMSE: 0.386
uw37903; RMSE: 0.479
uw37905; RMSE: 0.354
uw3792; RMSE: 0.168
uw37941; RMSE: 0.199
uw37962; RMSE: 0.344
uw3798; RMSE: 0.645
uw38002; RMSE: 0.519
uw3802; RMSE: 0.319
uw3806; RMSE: 0.471
uw38066; RMSE: 0.459
uw38067; RMSE: 0.456
uw3808; RMSE: 0.598
uw38085; RMSE: 0.644
uw38088; RMSE: 0.6
0
2000
ground level
bottom
feflow
observed
hydr. head

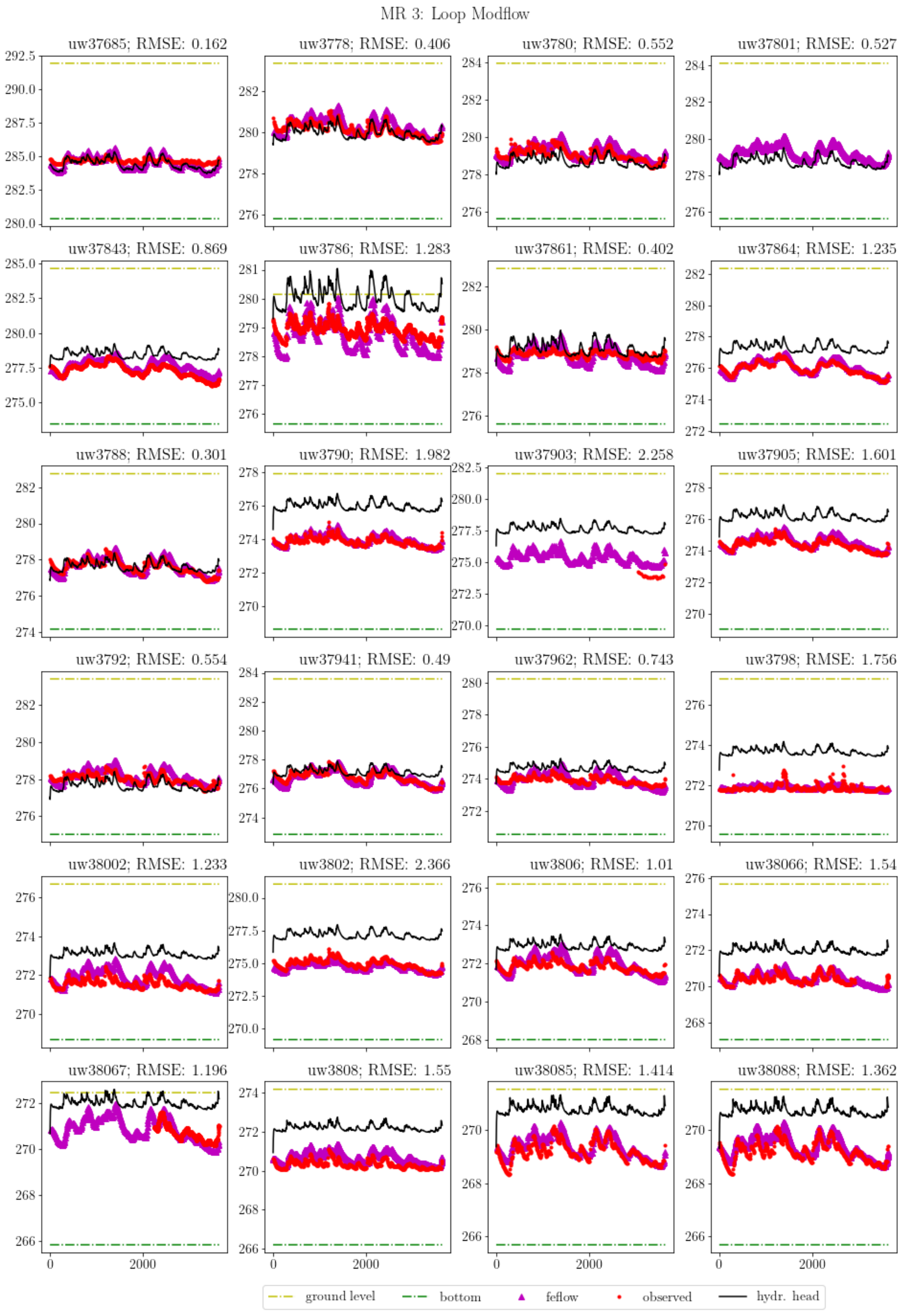

MR 1: Rivers as Wells
uw3810; RMSE: 0.627
uw38101; RMSE: 0.582
uw38105; RMSE: 0.523
uw38109; RMSE: 0.714
uw3812; RMSE: 0.31
uw38122; RMSE: 0.681
uw38142; RMSE: 0.66
uw38143; RMSE: 0.56
uw38144; RMSE: 1.007
uw38183; RMSE: 0.671
uw381831; RMSE: 0.683
uw38184; RMSE: 0.704
uw38186; RMSE: 0.882
uw38188; RMSE: 0.577
uw38203; RMSE: 1.262
uw38209; RMSE: 1.197
uw3822; RMSE: 0.771
uw3826; RMSE: 1.307
uw38261; RMSE: 1.205
uw38282; RMSE: 1.573
uw38301; RMSE: 1.163
uw3832; RMSE: 1.416
uw3834; RMSE: 0.949
uw38341; RMSE: 0.917
0
2000
ground level
bottom
feflow
observed
hydr. head

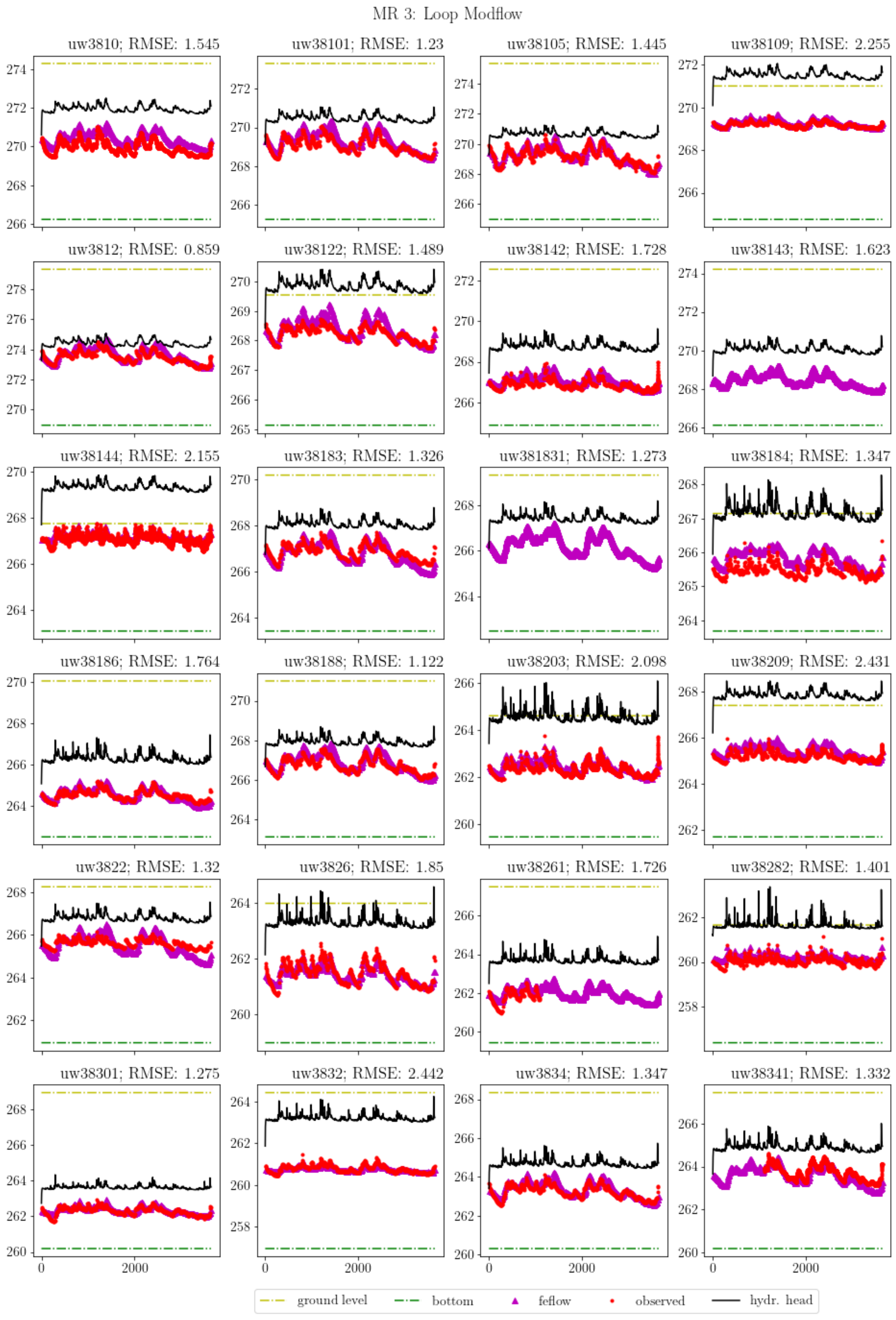

MR 2: Manual selected flow direction at rivers; Interpolated Recharge
uw3810; RMSE: 0.442
uw38101; RMSE: 0.341
uw38105; RMSE: 0.266
uw38109; RMSE: 0.574
uw3812; RMSE: 0.09
uw38122; RMSE: 0.381
uw38142; RMSE: 0.074
uw38143; RMSE: 0.252
uw38144; RMSE: 0.496
uw38183; RMSE: 0.199
uw381831; RMSE: 0.151
uw38184; RMSE: 0.199
uw38186; RMSE: 0.128
uw38188; RMSE: 0.13
uw38203; RMSE: 0.077
uw38209; RMSE: 0.524
uw3822; RMSE: 0.117
uw3826; RMSE: 0.175
uw38261; RMSE: 0.196
uw38282; RMSE: 0.104
uw38301; RMSE: 0.196
uw3832; RMSE: 0.317
uw3834; RMSE: 0.168
uw38341; RMSE: 0.162
0
2000
ground level
bottom
feflow
observed
hydr. head

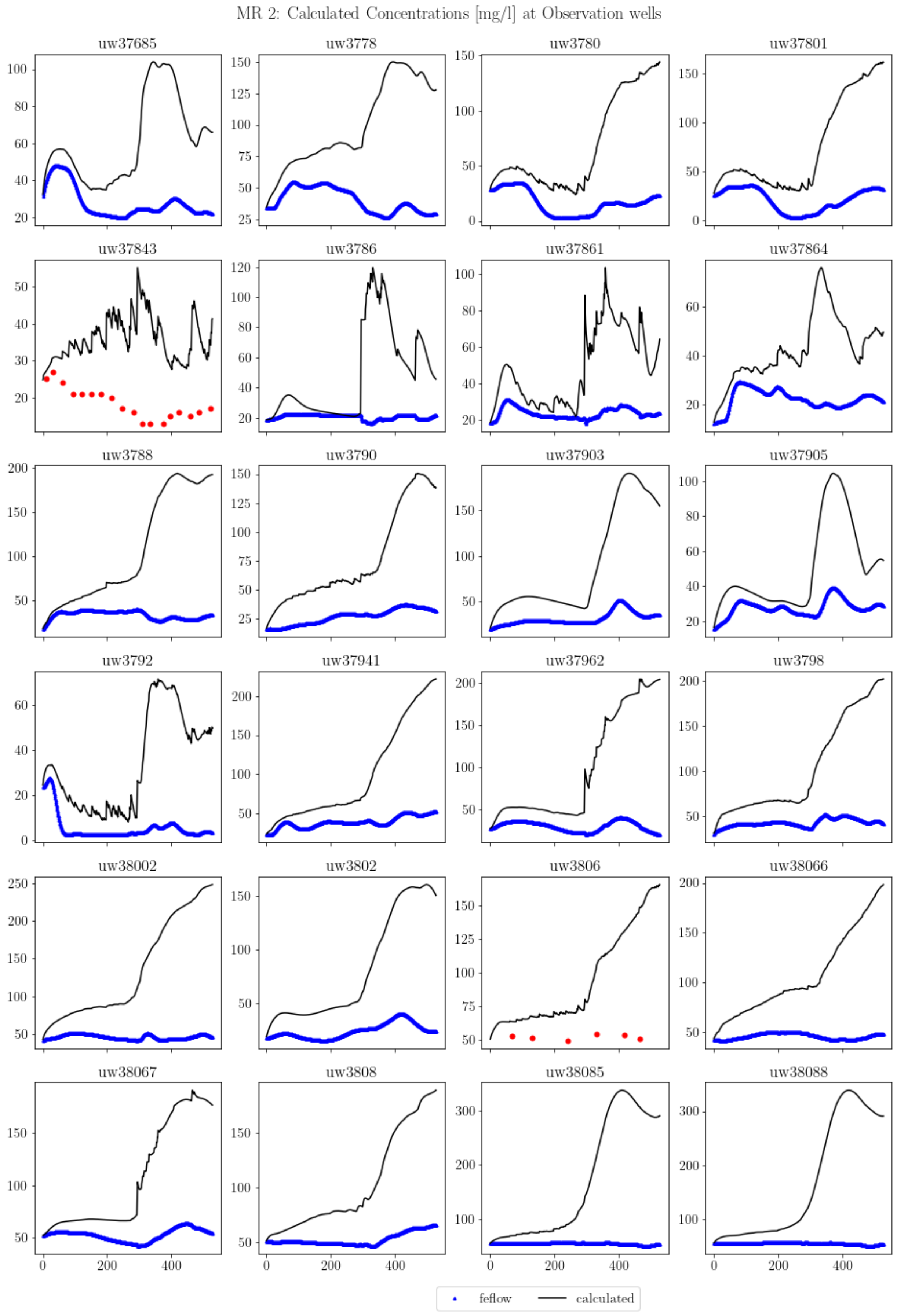

MR 2: Manual selected flow direction at rivers; Interpolated Recharge
uw37685; RMSE: 0.058
uw3778; RMSE: 0.108
uw3780; RMSE: 0.166
uw37801; RMSE: 0.156
uw37843; RMSE: 0.045
uw3786; RMSE: 0.101
uw37861; RMSE: 0.338
uw37864; RMSE: 0.091
uw3788; RMSE: 0.172
uw3790; RMSE: 0.06
uw37903; RMSE: 0.429
uw37905; RMSE: 0.102
uw3792; RMSE: 0.219
uw37941; RMSE: 0.091
uw37962; RMSE: 0.095
uw3798; RMSE: 0.027
uw38002; RMSE: 0.253
uw3802; RMSE: 0.053
uw3806; RMSE: 0.275
uw38066; RMSE: 0.305
uw38067; RMSE: 0.249
uw3808; RMSE: 0.437
uw38085; RMSE: 0.432
uw38088; RMSE: 0.379
ground level
bottom
feflow
observed
hydr. head


MR 2a: Calibrated Leakage factors
uw37685; RMSE: 0.054
uw3778; RMSE: 0.098
uw3780; RMSE: 0.061
uw37801; RMSE: 0.061
uw37843; RMSE: 0.038
uw3786; RMSE: 0.1
uw37861; RMSE: 0.314
uw37864; RMSE: 0.12
uw3788; RMSE: 0.063
uw3790; RMSE: 0.051
uw37903; RMSE: 0.477
uw37905; RMSE: 0.11
uw3792; RMSE: 0.034
uw37941; RMSE: 0.041
uw37962; RMSE: 0.091
uw3798; RMSE: 0.032
uw38002; RMSE: 0.134
uw3802; RMSE: 0.042
uw3806; RMSE: 0.187
uw38066; RMSE: 0.231
uw38067; RMSE: 0.147
uw3808; RMSE: 0.173
uw38085; RMSE: 0.115
uw38088; RMSE: 0.094
0
2000
ground level
bottom
feflow
observed
hydr. head

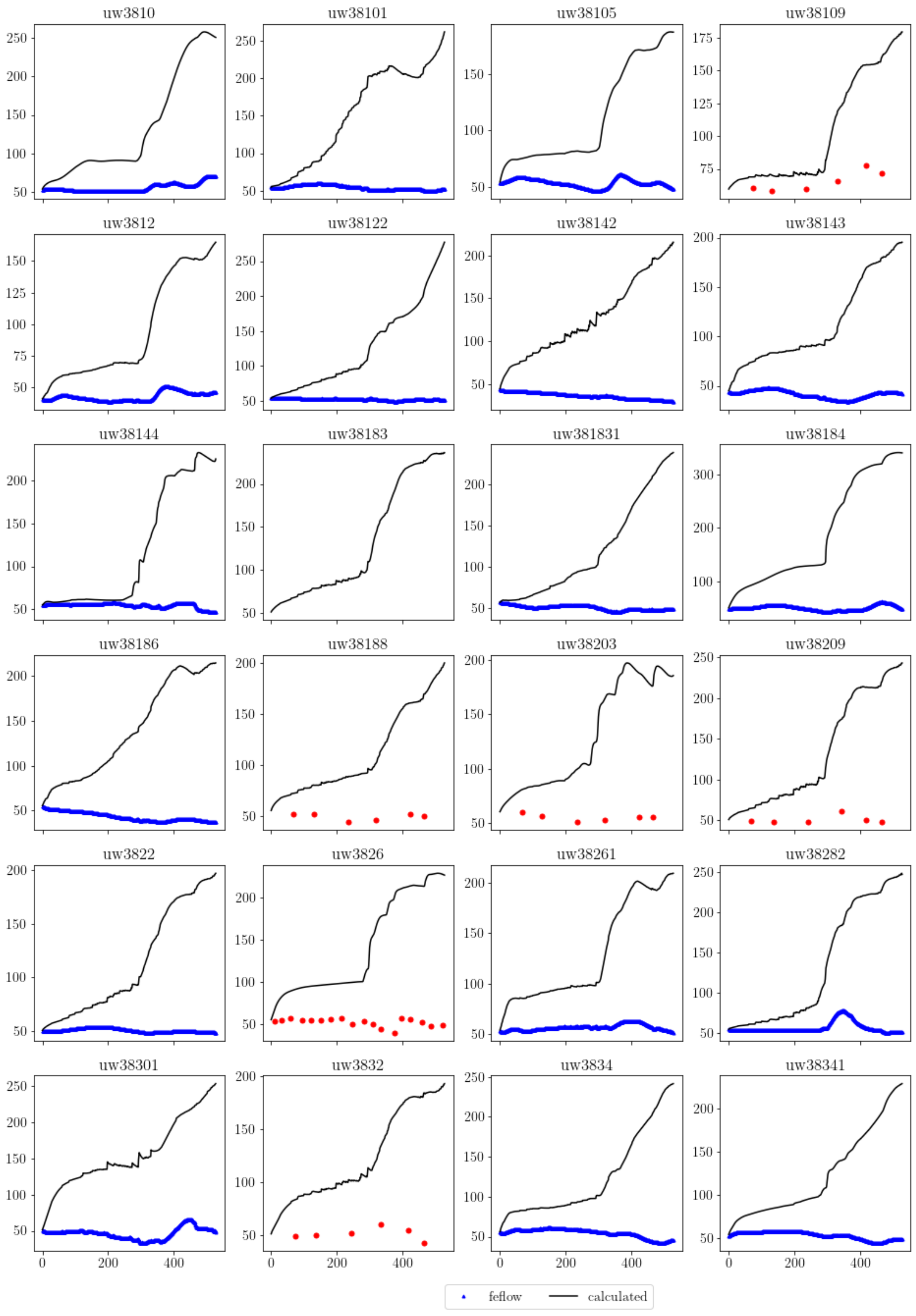

MR 2a: Calibrated Leakage factors
uw3810; RMSE: 0.119
uw38101; RMSE: 0.148
uw38105; RMSE: 0.137
uw38109; RMSE: 0.161
uw3812; RMSE: 0.082
uw38122; RMSE: 0.085
uw38142; RMSE: 0.059
uw38143; RMSE: 0.162
uw38144; RMSE: 0.109
uw38183; RMSE: 0.103
uw381831; RMSE: 0.104
uw38184; RMSE: 0.203
uw38186; RMSE: 0.081
uw38188; RMSE: 0.097
uw38203; RMSE: 0.067
uw38209; RMSE: 0.625
uw3822; RMSE: 0.123
uw3826; RMSE: 0.166
uw38261; RMSE: 0.168
uw38282; RMSE: 0.104
uw38301; RMSE: 0.147
uw3832; RMSE: 0.387
uw3834; RMSE: 0.108
uw38341; RMSE: 0.104
0
2000
ground level
bottom
feflow
observed
hydr. head


MR 3: Loop Modflow
uw37685; RMSE: 0.162
uw3778; RMSE: 0.406
uw3780; RMSE: 0.552
uw37801; RMSE: 0.527
uw37843; RMSE: 0.869
uw3786; RMSE: 1.283
uw37861; RMSE: 0.402
uw37864; RMSE: 1.235
uw3788; RMSE: 0.301
uw3790; RMSE: 1.982
uw37903; RMSE: 2.258
uw37905; RMSE: 1.601
uw3792; RMSE: 0.554
uw37941; RMSE: 0.49
uw37962; RMSE: 0.743
uw3798; RMSE: 1.756
uw38002; RMSE: 1.233
uw3802; RMSE: 2.366
uw3806; RMSE: 1.01
uw38066; RMSE: 1.54
uw38067; RMSE: 1.196
uw3808; RMSE: 1.55
uw38085; RMSE: 1.414
uw38088; RMSE: 1.362
0
2000
ground level
bottom
feflow
observed
hydr. head


MR 3: Loop Modflow
uw3810; RMSE: 1.545
uw38101; RMSE: 1.23
uw38105; RMSE: 1.445
uw38109; RMSE: 2.255
uw3812; RMSE: 0.859
uw38122; RMSE: 1.489
uw38142; RMSE: 1.728
uw38143; RMSE: 1.623
uw38144; RMSE: 2.155
uw38183; RMSE: 1.326
uw381831; RMSE: 1.273
uw38184; RMSE: 1.347
uw38186; RMSE: 1.764
uw38188; RMSE: 1.122
uw38203; RMSE: 2.098
uw38209; RMSE: 2.431
uw3822; RMSE: 1.32
uw3826; RMSE: 1.85
uw38261; RMSE: 1.726
uw38282; RMSE: 1.401
uw38301; RMSE: 1.275
uw3832; RMSE: 2.442
uw3834; RMSE: 1.347
uw38341; RMSE: 1.332
ground level
bottom
feflow
observed
hydr. head


MR 2: Calculated Concentrations [mg/l] at Observation wells
uw37685
uw3778
uw3780
uw37801
uw37843
uw3786
uw37861
uw37864
uw3788
uw3790
uw37903
uw37905
uw3792
uw37941
uw37962
uw3798
uw38002
uw3802
uw3806
uw38066
uw38067
uw3808
uw38085
uw38088
feflow
calculated

MR 2: Calculated Concentrations [mg/l] at Observation wells



B. Appendix: Python Files

The following files were exported directly from the Python environment of Jupyter Notebook. (expect properties file)

Contents

```python
############################################
### Properties, Packages and Functions ###
############################################

### Import packages ###
import os, sys, platform
import numpy as np
import datatable as dt
import matplotlib.pyplot as plt
import flopy
from flopy.utils import Raster
from shapely.geometry import Polygon, Point, LineString, MultiLineString,
MultiPoint, MultiPolygon
import shapefile as sf
import pandas as pd
import geopandas as gpd
from scipy.spatial import cKDTree
from shapely.strtree import STRtree
from scipy.interpolate import griddata
from collections import defaultdict
from descartes import PolygonPatch
import time


# Main Properties
gridsize = 40
stress_period_start = 0
stress_period_end = 3650

modelname = 'wlf-wie'
filenameJNB = '210321-SG'
raster_resample_quality = 'linear'

stress_periods = int(stress_period_end - stress_period_start)


### Workspace / Definition of Directories ###
model_ws = 'MF_Files_tr'
model_op = 'ModelOutput/'
vtk_path = 'VTK_Files/'

if not os.path.exists(vtk_path):
    os.makedirs(vtk_path)
if not os.path.exists(model_op):
    os.makedirs(model_op)
if not os.path.exists(model_ws):
    os.makedirs(model_ws)
plot_folder = os.path.join(filenameJNB + '_Plots/')
if not os.path.exists(plot_folder):
    os.makedirs(plot_folder)

triExeName = '../../EXE/triangle.exe'
mf6ExeName = '../../EXE/mf6.2.1/bin/mf6.exe'

shppath = '../../GIS_WIE/shp/'
shppathBC = '../../GIS_WIE/LFW_MasterArbeit/BC/'
shppathWIE = '../../GIS_WIE/shp_WIE'
storasimpath = '../../STORASIM/v4b DüMe erhöht DVx04/H2O'
transfolder = '../../GIS_WIE/LFW_MasterArbeit/TransportModel'

bcdata = '../BC'
obsdata = '../OBS'
rchdata = '../RCH'

outputfiles = os.path.join(model_ws, modelname)


### Functions ###

# find nearest data point
def ckdnearest(gdA, gdB):
    nA = np.array(list(gdA.centroid.apply(lambda x: (x.x, x.y))))
    nB = np.array(list(gdB.geometry.apply(lambda x: (x.x, x.y))))
    btree = cKDTree(nB)
    dist, idx = btree.query(nA, k=1)
    gdf = pd.concat(
        [gdA.reset_index(drop=True),
         gdB.iloc[idx].drop(columns="geometry").reset_index(drop=True),
         pd.Series(dist, name='dist')], axis=1)
    return gdf

# raster discretization
def raster_load_n_resample(fname, model):
    raster = Raster.load(os.path.join(shppathWIE, 'raster/', fname))
    raster = raster.resample_to_grid(model.xcellcenters,
                                     model.ycellcenters,
                                     band=raster.bands[0],
                                     method=raster_resample_quality)
    return raster


# combined intersection and nearest data point
def intersect_geometry(shps, p_shp, pathofshape, model):
    sgr = model.modelgrid
    ix = GridIntersect(sgr)

    ts = pd.DataFrame()
    intersection = pd.DataFrame()

    for i in shps:
        gdf = gpd.read_file(os.path.join(pathofshape, i), delimiter=",", header=0)
        for j in range(len(gdf.geometry)):
            if gdf.geom_type[0] == 'LineString':
                intersection = pd.concat([intersection, pd.DataFrame(ix.intersect_polyline(LineString(gdf.geometry[j])))],
                                         axis=0).reset_index(drop=True)
            if gdf.geom_type[0] == 'Point':
                intersection = pd.concat([intersection, pd.DataFrame(ix.intersect_point(Point(gdf.geometry[j])))],
                                           axis=0).reset_index(drop=True)
    intersection['centroid'] = [Point(sgr.xcellcenters[i], sgr.ycellcenters[i]) for i in intersection.cellids]
    intersection = ckdnearest(intersection, p_shp)

    return intersection


# root mean square error
def RMSE(measured, observed):
    return ((np.subtract(measured, observed)**2).mean())**(1/2)


# Super-class GridIntersect (by Davíd Brakenhoff); able to intersect structured and vertex based grid type
# (Is also available within the Flopy environement, but in the early stages of model creation (summer 2020), the built-in functiona did not fulfill expectations)
class GridIntersect:

    def __init__(self, mfgrid):

        self.mfgrid = mfgrid

        if mfgrid.grid_type == "structured":
            self.gridshapes = self._rect_grid_to_shape_list()
        elif mfgrid.grid_type == "vertex":
            self.gridshapes = self._vtx_grid_to_shape_list()

        self.strtree = STRtree(self.gridshapes)

    def _rect_grid_to_shape_list(self):
        shplist = []
        for i in range(self.mfgrid.nrow):
            for j in range(self.mfgrid.ncol):
                xy = self.mfgrid.get_cell_vertices(i, j)
                p = Polygon(xy)
                p.name = (i, j)
                shplist.append(p)
        return shplist

    def _vtx_grid_to_shape_list(self):
        shplist = []
        if isinstance(self.mfgrid._cell2d, np.recarray):
            for icell in self.mfgrid._cell2d.icell2d:
                points = []
                for iv in self.mfgrid._cell2d[["icvert_0", "icvert_1",
                "icvert_2"]][icell]:
                    points.append((self.mfgrid._vertices.xv[iv],
                                   self.mfgrid._vertices.yv[iv]))
                # close the polygon, if necessary
                if points[0] != points[-1]:
                    points.append(points[0])
                p = Polygon(points)
                p.name = icell
                shplist.append(p)
        elif isinstance(self.mfgrid._cell2d, list):
            for icell in range(len(self.mfgrid._cell2d)):
                points = []
                for iv in self.mfgrid._cell2d[icell][-3:]:
                    points.append((self.mfgrid._vertices[iv][1],
                                   self.mfgrid._vertices[iv][2]))
                # close the polygon, if necessary
                if points[0] != points[-1]:
                    points.append(points[0])
                p = Polygon(points)
                p.name = icell
                shplist.append(p)
        return shplist

    def _sort_strtree_result(self, shapelist):
        def sort_key(o):
            return o.name
        shapelist.sort(key=sort_key)
        return shapelist

    def intersect_point(self, shp, sort_by_cellid=True, return_all_ix=False):
        ixshapes = self.strtree.query(shp)
        if sort_by_cellid:
            ixshapes = self._sort_strtree_result(ixshapes)

        isectshp = []
        cellids = []
        vertices = []

        for i, r in enumerate(ixshapes):
            intersect = shp.intersection(r)
            if intersect.__geo_interface__["type"] == "Point" or
            intersect.__geo_interface__["type"] == "MultiPoint":
                # return all intersections (point can intersect with multiple cells)
                if return_all_ix:
                    pt = intersect.__geo_interface__["coordinates"]
                    isectshp.append(intersect)
                    vertices.append(pt)
                    cellids.append(r.name)
                else:
                    # return only point intersection with cell with lowest cellid
                    pt = intersect.__geo_interface__["coordinates"]
                    if pt in vertices:
                        continue
                    isectshp.append(intersect)
                    vertices.append(pt)
                    cellids.append(r.name)

        rec = np.recarray(len(isectshp), names=["cellids", "vertices", "intersects"],
                          formats=["O", "O", "O"])
        rec.intersects = isectshp
        rec.vertices = vertices
```

```python
        rec.cellids = cellids

        return rec

    def intersect_polygon(self, shp, sort_by_cellid=True, return_all_ix=False):
        ixshapes = self.strtree.query(shp)
        if sort_by_cellid:
            ixshapes = self._sort_strtree_result(ixshapes)

        isectshp = []
        cellids = []
        vertices = []
        areas = []

        for i, r in enumerate(ixshapes):
            intersect = shp.intersection(r)
            # return all intersections (also 0.0 area)
            if return_all_ix:
                if intersect.type == "GeometryCollection":
                    if len(intersect.__geo_interface__["geometries"]) == 0:  # no intersect
                        continue
                    for geom in intersect:
                        isectshp.append(geom)
                        if "Polygon" in geom.type:
                            areas.append(geom.area)
                        else:
                            areas.append(np.nan)
                        if "coordinates" in geom.__geo_interface__.keys():
                            vertices.append(geom.__geo_interface__["coordinates"])
                        else:
                            vertices.append(np.nan)
                        cellids.append(r.name)
                else:
                    isectshp.append(intersect)
                    if "Polygon" in intersect.type:
                        areas.append(intersect.area)
                    else:
                        areas.append(np.nan)
                    if "coordinates" in intersect.__geo_interface__.keys():
                        vertices.append(intersect.__geo_interface__["coordinates"])
                    else:
                        vertices.append(np.nan)
                    cellids.append(r.name)
            else:
                # return only intersections with area > 0.0
                if intersect.area > 0.0:
                    isectshp.append(intersect)
                    areas.append(intersect.area)
                    vertices.append(intersect.__geo_interface__["coordinates"])
                    cellids.append(r.name)

        rec = np.recarray(len(isectshp), names=["cellids", "vertices", "areas", "intersects"],
                          formats=["O", "O", "f8", "O"])
        rec.intersects = isectshp
        rec.vertices = vertices
        rec.areas = areas
        rec.cellids = cellids

        return rec

    def intersect_polyline(self, shp, sort_by_cellid=True, return_all_ix=False):
        result = self.strtree.query(shp)
        if sort_by_cellid:
            result = self._sort_strtree_result(result)

        isectshp = []
        cellids = []
        vertices = []
        lengths = []

        for i, r in enumerate(result):
            intersect = shp.intersection(r)
            # add all intersects (also if length == 0.0)
            # shapes can intersect with multiple cells (e.g. on boundary)
            if return_all_ix:
                if intersect.type == "GeometryCollection":
                    if len(intersect.__geo_interface__["geometries"]) == 0:  # no
                    intersect
                        continue
                    for geom in intersect:
                        # object is LineString or MultiLineString or Point
                        verts = geom.__geo_interface__["coordinates"]
                        vertices.append(verts)
                        if "LineString" in geom.type:
                            lengths.append(geom.length)
                        else:
                            lengths.append(np.nan)
                        isectshp.append(geom)
                else:  # object is LineString or MultiLineString or Point
                    verts = intersect.__geo_interface__["coordinates"]
                    vertices.append(verts)
                    if "LineString" in intersect.type:
                        lengths.append(intersect.length)
                    else:
                        lengths.append(np.nan)
                    isectshp.append(intersect)
                    cellids.append(r.name)
            else:
                # add only intersects with length > 0.0
                # linestring intersects with only one gridcell (with lowest cellid)
                if intersect.length > 0.0:
                    if intersect.type == "GeometryCollection":
                        for geom in intersect:
                            if "LineString" in geom.type:
                                verts = geom.__geo_interface__["coordinates"]
                                if verts in vertices:
                                    continue
                                isectshp.append(geom)
                                lengths.append(geom.length)
                                vertices.append(verts)
                                cellids.append(r.name)
                    else:  # result is MultiLineString or LineString
                        verts = intersect.__geo_interface__["coordinates"]
                        if verts in vertices:
                            continue
                        isectshp.append(intersect)
                        lengths.append(intersect.length)
                        vertices.append(verts)
                        cellids.append(r.name)

        rec = np.recarray(len(isectshp), names=["cellids", "vertices", "lengths",
        "intersects"],
                          formats=["O", "O", "f8", "O"])
        rec.intersects = isectshp
        rec.vertices = vertices
        rec.lengths = lengths
        rec.cellids = cellids

        return rec


    def plot_polygon(self, rec, ax=None, **kwargs):
        if ax is None:
            _, ax = plt.subplots()

        for i, ishp in enumerate(rec.intersects):
            ppi = PolygonPatch(ishp, facecolor="C{}".format(i%10), **kwargs)
            ax.add_patch(ppi)

        return ax

    def plot_polyline(self, rec, ax=None, **kwargs):
        if ax is None:
            _, ax = plt.subplots()

        for i, ishp in enumerate(rec.intersects):
            if ishp.type == "MultiLineString":
                for part in ishp:
                    ax.plot(part.xy[0], part.xy[1], ls="-", c="C{}".format(i%10),
                    **kwargs)
            else:
                ax.plot(ishp.xy[0], ishp.xy[1], ls="-", c="C{}".format(i%10),
                **kwargs)

        return ax

    def plot_point(self, rec, ax=None, **kwargs):
        if ax is None:
            _, ax = plt.subplots()

        x = [ip.x for ip in rec.intersects]
        y = [ip.y for ip in rec.intersects]

        ax.scatter(x, y, **kwargs)

        return ax

```

01_SDIS

May 25, 2021

1 Spactial Model Discretization

WLF Modflow Model

[1]:
```
from prop import *
%load_ext autotime
```

```
<IPython.core.display.HTML object>

time: 0 ns
```

1.1 Discretisation Setup

Temporary spatial model discretization to create the GridIntersection class. Definition of model area and boundary. DIS-model package remains unsaved at this stage - will be recreated in following files.

[2]:
```
shp_boundary = sf.Reader(os.path.join(shppath, 'LFW_active'))
boundary = np.array(shp_boundary.shapeRecords()[0].shape.points)

Lx = boundary.max(axis=0)[0]-boundary.min(axis=0)[0]
Ly = boundary.max(axis=0)[1]-boundary.min(axis=0)[1]
nlay = 1
delr, delc = gridsize, gridsize
nrow = int(round(Ly / delr + .49))
ncol = int(round(Lx / delc + .49))
top = 1000
xll=boundary.min(axis=0)[0]
yll=boundary.min(axis=0)[1]

temp_sim = flopy.mf6.MFSimulation(sim_name=modelname, version='mf6',␣
 ↪exe_name=mf6ExeName, sim_ws=model_ws)
temp_model = flopy.mf6.ModflowGwf(temp_sim, modelname=modelname,␣
 ↪model_nam_file='{}.nam'.format(modelname))
dis = flopy.mf6.ModflowGwfdis(temp_model, pname='dis', length_units='METERS',␣
 ↪nlay=nlay, nrow=nrow, ncol=ncol, delr=delr,
                              delc=delc, top=top, xorigin=xll, yorigin=yll,
                              filename='{}.dis'.format(modelname))

# Grid Intersection Class
```

1

```
sgr = temp_model.modelgrid
ix = GridIntersect(sgr)
```

```
time: 953 ms
```

1.2 Load and Discretize Aquifer Properties

```
[3]: ## idomain
act_cid = ix.intersect_polygon(Polygon(boundary))
idomain = flopy.mf6.ModflowGwfdis.idomain.empty(temp_model, layered=True,␣
 ↪default_value=-1)
for i in range(len(act_cid['cellids'])):
    idomain['data'][0][act_cid['cellids'][i][0], act_cid['cellids'][i][1]] = 1
np.savetxt(os.path.join(model_op, 'idomain.csv'), idomain['data'][0],␣
 ↪delimiter=',')
```

```
time: 21.4 s
```

```
[4]: act_cid = pd.DataFrame(act_cid)
act_cid['centroid'] = [Point(sgr.xcellcenters[i], sgr.ycellcenters[i]) for i in␣
 ↪act_cid.cellids]
act_cid.to_csv(os.path.join(model_op,'act_cid.csv'), index=True, header=True)
```

```
time: 6.02 s
```

```
[5]: botm_p = gpd.read_file(os.path.join(shppathBC, '../MaterialProperties/',␣
 ↪'BottomElevation_m_polygons.shp'))
botm_p.drop(['geometry'], axis=1)
botm_p = gpd.GeoDataFrame(botm_p, geometry=[Point(xy) for xy in zip(botm_p.
 ↪CENTER_X, botm_p.CENTER_Y)])

cond_p = gpd.read_file(os.path.join(shppathBC, '../MaterialProperties/',␣
 ↪'Conductivity_ms.shp'))
cond_p['COND'] = cond_p['COND'] * 86400

poro_p = gpd.read_file(os.path.join(shppathBC, '../MaterialProperties/',␣
 ↪'Porosity.shp'))

ic_p = gpd.read_file(os.path.join(shppathBC, '../InitialConditions/',␣
 ↪'HydraulicHead_m.shp'))

topelev_p = gpd.read_file(os.path.join(shppathBC, '../', 'topelev_points.shp'))


# Transport
imc_tr = gpd.read_file(os.path.join(shppathBC + '../TransportModel',␣
 ↪'MassConc_initial.shp'))
```

```
n_tr = gpd.read_file(os.path.join(shppathBC + '../TransportModel',␣
↪'Porosity-MassTransport.shp'))
```

```
time: 24.7 s
```

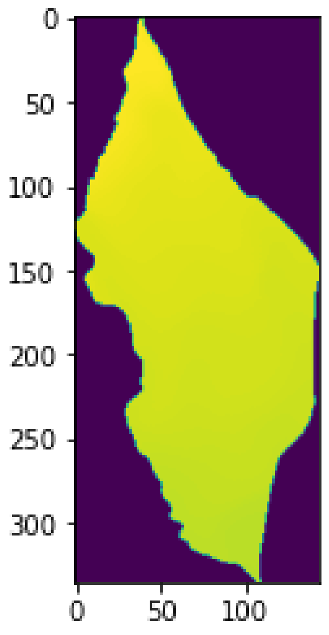
[6]:
```
def materialarray(inputgdf, savename, columname):
    idf = pd.DataFrame()
    for j in range(len(inputgdf)):
        df = pd.DataFrame(ix.intersect_point(inputgdf.loc[j, 'geometry']))
        df[columname] = inputgdf.loc[j, columname]
        idf = pd.concat([df, idf])
    idf = idf.groupby('cellids').agg(columname=(columname, 'mean')).
↪reset_index(drop=False)

    df = ckdnearest(act_cid, inputgdf)

    ea = np.zeros((nrow,ncol), dtype=float)

    counteridf = 0
    for i, cid in enumerate(act_cid.cellids):
        if cid in list(idf.cellids):
            counteridf += 1
            ea[cid[0], cid[1]] = float(idf[idf.cellids == cid].columname)
        else:
            ea[cid[0], cid[1]] = float(df.loc[i, columname])

    np.savetxt(os.path.join(model_op, savename), ea, delimiter=' ')
    plt.imshow(ea)
    print('cells by intersection: ', counteridf)
    print('max dist of nearest point m.: ', max(df.dist))
```

```
time: 0 ns
```

[7]:
```
materialarray(botm_p, 'botmelev.txt', 'BOTT')
```

```
cells by intersection:  13278
max dist of nearest point m.:  102.66667664725885
```

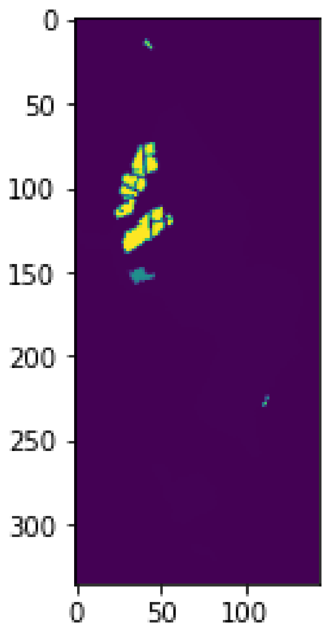
3



```
time: 2min 53s
```

[8]:
```
materialarray(cond_p, 'npf_hydcond.txt', 'COND')
```

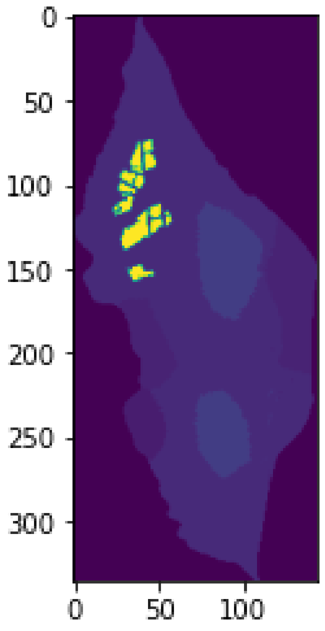
```
cells by intersection:  13278
max dist of nearest point m.:  102.66667664724011
```



```
time: 6min 4s
```

```
[9]: materialarray(poro_p, 'sy.txt', 'STOR')
```

```
cells by intersection:  13278
max dist of nearest point m.:  102.66667664724011
```

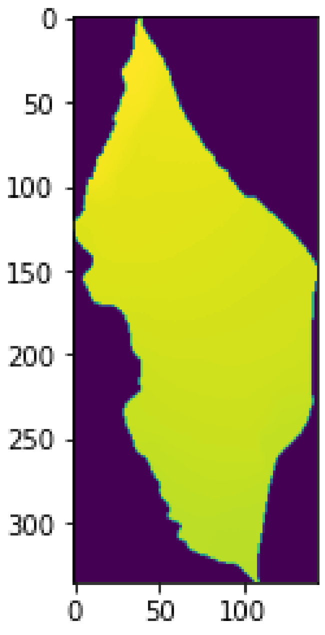


```
time: 5min 55s
```

```
[10]: materialarray(ic_p, 'strthead.txt', 'FINIT')
```

```
cells by intersection:  8427
max dist of nearest point m.:  116.73369350751005
```

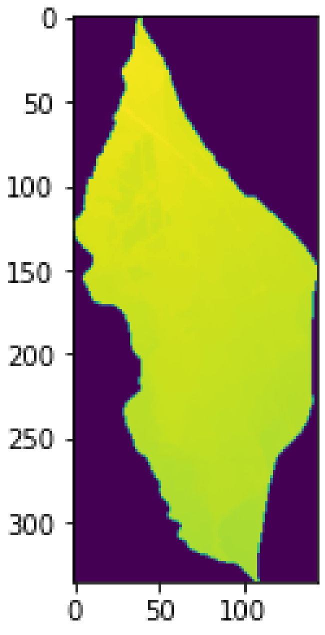
5



```
time: 2min 7s
```

[11]:
```
materialarray(topelev_p, 'topelev.txt', 'REF_E')
```

```
cells by intersection:  23374
max dist of nearest point m.:  44.194867289044964
```



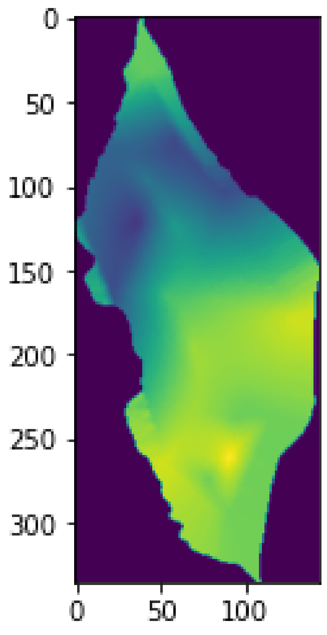
6

```
time: 19min 23s
```

1.2.1 Transport Params

```
[12]: materialarray(imc_tr, 'imc_tr.txt', 'MINIT')
```

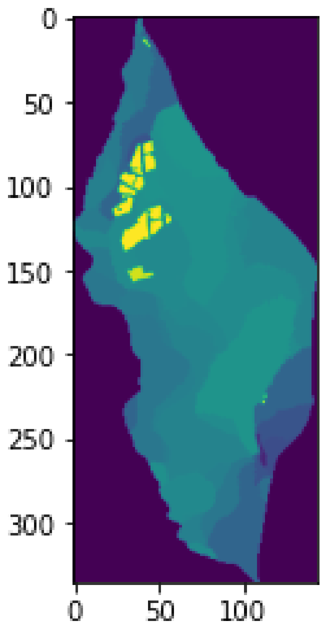
```
cells by intersection:  19801
max dist of nearest point m.:  59.34973145626214
```



```
time: 8min 4s
```

```
[13]: materialarray(n_tr, 'n_tr.txt', 'PORO')
```

```
cells by intersection:  23374
max dist of nearest point m.:  44.194867289044964
```



```
time: 19min 23s
```

1.3 Discretization of Observations

```
[14]: obs_geom = pd.read_csv(os.path.join(obsdata, 'obs_geom.csv'), delimiter=",",␣
       ↪header=0)
      obs_geom = gpd.GeoDataFrame(obs_geom['Name'], geometry=[Point(xy) for xy in␣
       ↪zip(obs_geom.X, obs_geom.Y)])

      obsX = pd.DataFrame()
      for i, geom in enumerate(obs_geom.geometry):
          obsX = pd.concat([obsX, pd.DataFrame(ix.intersect_point(geom))], axis=0).
       ↪reset_index(drop=True)
      obsX = pd.merge(obs_geom, obsX, how='left', left_index=True, right_index=True).
       ↪drop(columns=['vertices', 'intersects'])
      obsX.to_csv(os.path.join(model_op,'X_obs.csv'), index=True, header=True)
```

```
time: 250 ms
```

1.4 Find BC-effected cells

```
[15]: river_p = gpd.read_file(os.path.join('..\..
       ↪\GIS_WIE\Ganglinien_Q_Var24_Real_93_Kal77g', 'Couchy.shp'))

      rivX = pd.DataFrame()
      for i in range(len(river_p)):
```

8

```
    df = pd.DataFrame(ix.intersect_point(river_p.loc[i, 'geometry']))
    df['F'] = river_p.loc[i, 'F']
    df['steady'] = river_p.loc[i, 'STEADY']
    df['node'] = river_p.loc[i, 'NODE']
    rivX = pd.concat([df, rivX])
rivX = rivX.drop_duplicates(subset='cellids', keep='first').
 ↪reset_index(drop=True)
```

```
time: 10.8 s
```

[16]:
```
river_l = ['GWM_BC_Mur_ILine.lin.shp', 'GWM_BC_Sulm_ILine.lin.shp',␣
 ↪'GWM_BC_Lassnitz_ILine.lin.shp', 'Allg_Gewässer_Linien.shp']
rivX_l = pd.DataFrame()
for i in river_l:
    gdf = gpd.read_file(os.path.join(shppath, i))
    for j in range(len(gdf)):
        df = pd.DataFrame(ix.intersect_polyline(LineString(gdf.loc[j,␣
 ↪'geometry'])))
        rivX_l = pd.concat([df, rivX_l])
rivX_l = rivX_l.drop_duplicates(subset='cellids', keep='first').
 ↪reset_index(drop=True)
```

```
time: 32.6 s
```

[17]:
```
rivX = pd.merge(rivX, rivX_l, how='left', on='cellids')
rivX = rivX.loc[:, ['cellids', 'F', 'node', 'steady', 'lengths']]
```

```
time: 47 ms
```

[18]:
```
## CHB
p_shp = gpd.read_file(os.path.join('..\..\GIS_WIE\Ganglinien_Q_1st&3rdBC',␣
 ↪'Dirichle.shp'))
pathofshape, shp = shppathWIE, ['1stBC_ModellgrenzeNW.shp']
chb_intersect = intersect_geometry(shp, p_shp, pathofshape, temp_model)
#chb_intersect = chb_intersect.drop_duplicates(subset='cellids', keep='first').
 ↪reset_index(drop=True)
chb_intersect.to_csv(os.path.join(model_op,'X_chb.csv'), index=True,␣
 ↪header=True)
```

```
time: 2.56 s
```

[19]:
```
## wells
p_shp = gpd.read_file(os.path.join(shppathBC, '4thBC.shp'))
well_intersect = pd.DataFrame()
for i in range(len(p_shp)):
    df = pd.DataFrame(ix.intersect_point(p_shp.loc[i, 'geometry']))
    df['F'] = p_shp.loc[i, 'F']
    well_intersect = pd.concat([df, well_intersect])
```

```
well_intersect.to_csv(os.path.join(model_op,'X_wel.csv'), index=True,␣
↪header=True)
```

```
time: 110 ms
```

[20]:
```
riv_section = gpd.read_file(os.path.join(shppathWIE, 'river_groups.shp'),␣
↪delimiter=",", header=0)
```

```
time: 156 ms
```

[21]:
```
riv_sec = pd.DataFrame()
for i in range(len(riv_section)):
    df = pd.DataFrame(ix.intersect_polygon(riv_section.loc[i, 'geometry']))
    df['riv_sec'] = riv_section.loc[i, 'Name']
    riv_sec = pd.concat([df, riv_sec])

riv_sec = riv_sec.loc[:,['cellids', 'riv_sec']].
↪drop_duplicates(subset='cellids', keep='first').reset_index(drop=True)
```

```
time: 4.34 s
```

[22]:
```
leakage_out = gpd.read_file(os.path.join(shppathBC, 'FeFlow_Leakage\L_out.shp'))
leakage_out['leakage'] = leakage_out['TRAF_OUT']

leakage_in = gpd.read_file(os.path.join(shppathBC, 'FeFlow_Leakage\L_in.shp'))
leakage_in['leakage'] = leakage_in['TRAF_IN']

def leakX(leakage_file):
    df = pd.DataFrame()
    for i in range(len(leakage_file)):
        dfisec = pd.DataFrame(ix.intersect_polygon(leakage_file.loc[i,␣
↪'geometry']))
        dfisec['leakage'] = leakage_file.loc[i, 'leakage']
        df = pd.concat([df, dfisec]).reset_index(drop=True)

    wm = lambda x: np.average(x, weights=df.loc[x.index, "areas"])
    df = df.groupby('cellids').agg(leakage=('leakage', wm), areas=('areas',␣
↪sum)).reset_index(drop=False)
    return df
```

```
time: 2 s
```

[23]:
```
rivX = pd.merge(rivX, leakX(leakage_in).rename(columns={'leakage': 'lkg_in'}),␣
↪how='left', on='cellids')
rivX = pd.merge(rivX, leakX(leakage_out).rename(columns={'leakage':␣
↪'lkg_out'}), how='left', on='cellids')
rivX = pd.merge(rivX, riv_sec, how='left', on='cellids')
rivX.to_csv(os.path.join(model_op,'rivX.csv'), index=True, header=True)
```

```
time: 6min 46s
```

[24]:
```
hydrotopes = gpd.read_file(os.path.join(shppathWIE, 'hydrotope_clipped_sp.shp'))

rch_ids = pd.DataFrame()
for i in range(len(hydrotopes)):
    rchisec = pd.DataFrame(ix.intersect_polygon(hydrotopes.loc[i, 'geometry']))
    rchisec['id'] = hydrotopes.loc[i, 'JOBID_Var1']
    rch_ids = pd.concat([rch_ids, rchisec])

rch_ids = rch_ids.sort_values(by=['areas'], ascending=False, ignore_index=True)
rch_ids = rch_ids.drop_duplicates(subset='cellids', keep='first').
 ↪reset_index(drop=True)
rch_ids.to_csv(os.path.join(model_op,'X_rch.csv'), index=True, header=True)
```

```
time: 1min 44s
```

1.4.1 Transport

[25]:
```
cnbc_shp = gpd.read_file(os.path.join(transfolder, 'Mass-concentrationBC.shp'))
```

```
time: 359 ms
```

[26]:
```
cnbc_intersect = pd.DataFrame()
for i in range(len(cnbc_shp)):
    df = pd.DataFrame(ix.intersect_point(cnbc_shp.loc[i, 'geometry']))
    df['F'] = cnbc_shp.loc[i, 'F']
    df['STEADY'] = cnbc_shp.loc[i, 'STEADY']
    cnbc_intersect = pd.concat([df, cnbc_intersect])
```

```
time: 21.5 s
```

[27]:
```
cnbc_intersect = cnbc_intersect.groupby('cellids').agg(F=('F', 'mean'),␣
 ↪steady=('STEADY', sum)).reset_index(drop=False)
cnbc_intersect.to_csv(os.path.join(model_op,'cnbc_X.csv'), index=True,␣
 ↪header=True)
```

```
time: 47 ms
```

02_TDIS

June 8, 2021

1 Temporal Model Discretization

WLF Modflow Model

[1]:
```
from prop import *
%load_ext autotime

def mktpl(stringinput):
    return tuple(map(int, stringinput.split('(')[1].split(')')[0].split(', ')))
```

```
<IPython.core.display.HTML object>
```

```
time: 0 ns
```

Get number of rows and columns

[7]:
```
shp_boundary = sf.Reader(os.path.join(shppath, 'LFW_active'))
boundary = np.array(shp_boundary.shapeRecords()[0].shape.points)

Lx = boundary.max(axis=0)[0]-boundary.min(axis=0)[0]
Ly = boundary.max(axis=0)[1]-boundary.min(axis=0)[1]
nlay = 1
delr, delc = gridsize, gridsize
nrow = int(round(Ly / delr + .49))
ncol = int(round(Lx / delc + .49))
top = 1000
xll=boundary.min(axis=0)[0]
yll=boundary.min(axis=0)[1]

temp_sim = flopy.mf6.MFSimulation(sim_name=modelname, version='mf6',␣
 ↪exe_name=mf6ExeName, sim_ws=model_ws)
temp_model = flopy.mf6.ModflowGwf(temp_sim, modelname=modelname,␣
 ↪model_nam_file='{}.nam'.format(modelname))
dis = flopy.mf6.ModflowGwfdis(temp_model, pname='dis', length_units='METERS',␣
 ↪nlay=nlay, nrow=nrow, ncol=ncol, delr=delr,
                              delc=delc, top=top, xorigin=xll, yorigin=yll,
                              filename='{}.dis'.format(modelname))

# Grid Intersection Class
sgr = temp_model.modelgrid
```

1

```
ix = GridIntersect(sgr)
```

```
time: 797 ms
```

[8]:
```
idomain = np.asarray(dt.fread(os.path.join(model_op, 'idomain.csv')))

# botm elevation
botm = np.asarray(dt.fread(os.path.join(model_op, 'botmelev.txt')))
```

```
time: 63 ms
```

Load previously created dataframes. Interpolation of oberservation heads.

[9]:
```
obsX = pd.read_csv(os.path.join(model_op, 'X_obs.csv'), delimiter=",",␣
 ↪header=0, index_col=0)
topelev = pd.read_csv(os.path.join(model_op, 'topelev.txt'), delimiter=" ",␣
 ↪header=None, index_col=None)
```

```
time: 15 ms
```

[10]:
```
#budgIO_Fl = pd.read_csv(os.path.join(bcdata,'budgIO_Tr.csv'), delimiter=",",␣
 ↪header=0, index_col=0).drop(['date'], axis=1) #budget file for river-as-well

river_intersect = pd.read_csv(os.path.join(model_op, 'rivX.csv'),␣
 ↪delimiter=",", header=0, index_col=0).fillna(value = 0)
riv_ts = pd.read_csv(os.path.join(bcdata, 'riv_ts.csv'), delimiter=",",␣
 ↪header=0, index_col=0)
river_intersect['cellids'] = river_intersect['cellids'].apply(lambda x:␣
 ↪mktpl(x))
```

```
time: 1.22 s
```

[18]:
```
for ien, i in enumerate(range(stress_period_start, stress_period_end)):
    stress_period_data = []
    stress_period_data_dr = []

    for j , cid in enumerate(river_intersect.cellids):
        if not idomain[cid[0],cid[1]] == 1:
            continue

        if river_intersect.loc[j, 'steady'] == 0:
            stage = riv_ts.loc[i, str(river_intersect.loc[j, 'F']).split('.
 ↪')[0]]
        else:
            stage = float(river_intersect.loc[j, 'F'])

        botmr = botm[cid[0], cid[1]]
```

2

```
        if river_intersect.loc[j, 'riv_sec'] == 'in' and river_intersect.loc[j,␣
↪'steady'] == 0:
            lkg = river_intersect.loc[j, 'lkg_in'] * river_intersect.loc[j,␣
↪'lengths']
        elif river_intersect.loc[j, 'lengths'] < gridsize:
            lkg = river_intersect.loc[j, 'lkg_out'] * gridsize
        else:
            lkg = river_intersect.loc[j, 'lkg_out'] * river_intersect.loc[j,␣
↪'lengths']

        boundname = river_intersect.loc[j, 'riv_sec']

        if river_intersect.loc[j, 'steady'] == 0:
            stress_period_data.append([1, cid[0]+1, cid[1]+1,' ', stage, lkg,␣
↪botmr, 1.0, boundname])
        else:
            stress_period_data_dr.append([1, cid[0]+1, cid[1]+1,' ', stage,␣
↪lkg, 1.0, boundname])

    np.savetxt(os.path.join(model_ws, '_riv_stress_period_data_' + str(i) + '.
↪txt'), stress_period_data, delimiter=' ', fmt="%s")
    np.savetxt(os.path.join(model_ws, '_drn_stress_period_data_' + str(i) + '.
↪txt'), stress_period_data_dr, delimiter=' ', fmt="%s")
```

```
time: 8min 5s
```

[19]:
```
well_intersect = pd.read_csv(os.path.join(model_op, 'X_wel.csv'),␣
↪delimiter=",", header=0, index_col=0).reset_index(drop=True)
well_ts = pd.read_csv(os.path.join(bcdata, 'well_ts.csv'), delimiter=",",␣
↪header=0, index_col=0).interpolate(method='linear', axis=0,␣
↪limit_direction='both')

for ien, i in enumerate(range(stress_period_start, stress_period_end)):
    stress_period_data = []
    for j in range(len(well_intersect)):
        cx = int(well_intersect.iloc[j, 0].split(',')[0].split('(')[1])
        cy = int(well_intersect.iloc[j, 0].split(', ')[1].split(')')[0])
        f = str(well_intersect.loc[j, 'F']).split('.')[0]
        if idomain[cx,cy] == 1:
            stress_period_data.append([1, cx+1, cy+1, format(float(well_ts.
↪loc[i, f]*(-1)), ".10E")])

    np.savetxt(os.path.join(model_ws, '_wel_stress_period_data_' + str(i) + '.
↪txt'), stress_period_data, delimiter=' ', fmt="%s")
```

```
time: 16.7 s
```

[20]:
```
chb_intersect = pd.read_csv(os.path.join(model_op, 'X_chb.csv'), delimiter=",",␣
↪header=0, index_col=0).reset_index(drop=True)
chb_ts = pd.read_csv(os.path.join(bcdata, 'chb_ts1.csv'), delimiter=",",␣
↪header=0, index_col=0).interpolate(method='linear', axis=0,␣
↪limit_direction='both')

for ien, i in enumerate(range(stress_period_start, stress_period_end)):
    stress_period_data = []
    for j in range(len(chb_intersect)):
        cx = int(chb_intersect.iloc[j, 0].split(',')[0].split('(')[1])
        cy = int(chb_intersect.iloc[j, 0].split(', ')[1].split(')')[0])
        f = str(chb_intersect.loc[j, 'F']).split('.')[0]
        chd = float(chb_ts.loc[i, f])
        botmr = botm[cx, cy]
        if idomain[cx,cy] == 1 and chd > botmr:
            stress_period_data.append([1, cx+1, cy+1, format(chd, ".10E")])

    np.savetxt(os.path.join(model_ws, '_chd_stress_period_data_' + str(i) + '.
↪txt'), stress_period_data, delimiter=' ', fmt="%s")
```

```
time: 1min 38s
```

1.1 Recharge

Spatial interpolation of observation head levels and calculation of depth to watertable.

[21]:
```
rch_ids = pd.read_csv(os.path.join(model_op, 'X_rch.csv'), delimiter=",",␣
↪header=0, index_col=0)

obs_ts = pd.read_csv(os.path.join(obsdata, 'obs_ts.csv'), delimiter=",",␣
↪header=0, index_col=0)
obs_heads = obs_ts.interpolate(method='linear', axis=0, limit_direction='both').
↪drop(columns=['date'])
obs_heads.index.name = 'time'
```

```
time: 265 ms
```

Definition of recharge and groundwater folder to save data for each stress period.

[22]:
```
gwfolder = os.path.join('GW_level_cs{}'.format(gridsize))
if not os.path.exists(gwfolder):
    os.makedirs(gwfolder)
```

```
time: 0 ns
```

[28]:
```
x = np.linspace(0,ncol-1,ncol)
y =  np.linspace(0,nrow-1,nrow)
X, Y = np.meshgrid(x, y)
```

```
idomain = np.where(np.asarray(dt.fread(os.path.join(model_op, 'idomain.csv')))␣
 ↪== 1,
                    0, np.where(np.asarray(dt.fread(os.path.join(model_op,␣
 ↪'idomain.csv'))) == -1, 1, 0))

mean = 0
for j in range(stress_period_start, stress_period_end):
    pz, px, py = [], [], []
    for i, cid in enumerate(obsX.cellids):
        if np.isnan(obs_heads.iloc[j, i]):
            continue
        px.append(cid.split(', ')[1].split(')')[0])
        py.append(cid.split(',')[0].split('(')[1])
        pz.append(obs_heads.iloc[j, i])
    Ti = griddata((px, py), pz, (X, Y), method='linear')
    Ti2 = griddata((px, py), pz, (X, Y), method='nearest')
    Ti = np.where(np.isnan(Ti), Ti2, Ti)
    Ti = topelev - Ti
    Ti = np.ma.array(Ti, mask=idomain)
    mean = mean + Ti/stress_periods
    np.savetxt(os.path.join(gwfolder, str(j)+'.csv'), Ti.filled(np.nan),␣
 ↪delimiter=',')
np.savetxt(os.path.join(model_op, 'mean_gwl.csv'), mean.filled(np.nan),␣
 ↪delimiter=',')
```

```
time: 7min 42s
```

Load STOTRASIM files into RAM

[29]:
```
rch_files = [int(f.split('.')[0]) for f in os.listdir(rchdata) if f.endswith(".
 ↪csv")]

stotrasim_files = []
for file in rch_files:
    stotrasim_files.append([file, dt.fread(os.path.join(rchdata, str(file)+'.
 ↪csv'), header=True)])

stotrasim_dict = defaultdict(list)
for filename, file in stotrasim_files:
    stotrasim_dict[filename].append([file])

Nrchdata = '../RCH_N'
rch_Nfiles = [int(f.split('.')[0]) for f in os.listdir(Nrchdata) if f.
 ↪endswith(".csv")]

stotrasim_Nfiles = []
for file in rch_Nfiles:
```

5

```
    stotrasim_Nfiles.append([file, dt.fread(os.path.join(rchdata, str(file)+'.
↪csv'), header=True)])

stotrasim_Ndict = defaultdict(list)
for filename, file in stotrasim_Nfiles:
    stotrasim_Ndict[filename].append([file])
```

```
time: 29.3 s
```

Head-dependant recharge computation

```
[30]: for i in range(stress_period_start, stress_period_end):
    df = np.asarray([[0. for x in range(ncol)] for y in range(nrow)])
    dfN = np.asarray([[0. for x in range(ncol)] for y in range(nrow)])
    #gw_level = mean
    gw_level = dt.fread(os.path.join(gwfolder, str(i)+'.csv'), header=False)
    for j, pid in enumerate(rch_ids.id):
        cx = int(rch_ids.iloc[j, 0].split(',')[0].split('(')[1])
        cy = int(rch_ids.iloc[j, 0].split(', ')[1].split(')')[0])
        rchf = stotrasim_dict[pid][0][0]
        rchNf = stotrasim_Ndict[pid][0][0]

        gwnbp = np.asarray(rchf.names[1:]).astype(int)
        gwnb = gwnbp.flat[np.abs(gwnbp - gw_level[cx,cy]*100).argmin()]

        rch = float(rchf[i, str(gwnb)])/1000
        rchN = float(rchNf[i, str(gwnb)])*(62/14)
        df[cx, cy] = rch

        if rch == 0. or rchN == 0.:
            dfN[cx, cy] = 0
        else:
            dfN[cx, cy] = float((rchN / 10) / rch)

    np.savetxt(os.path.join(model_ws, modelname + '.rch_stress_period_data_' +␣
↪str(i) + '.txt'), df, delimiter='  ')
    np.savetxt(os.path.join(model_ws, modelname + '.rch_N_aux_' + str(i) + '.
↪txt'), dfN, delimiter='  ')
```

```
time: 4h 57s
```

1.2 Transport

```
[31]: lakes = pd.merge(pd.DataFrame(index=range(0,9500)),
                 pd.read_csv(os.path.join('../ZR/extra/N_See_Var2.pow'),␣
↪skiprows=3, skipfooter=2, sep='              ', header=None, engine='python'),
                  how='left', left_index=True, right_on=0).
↪reset_index(drop=True).interpolate(method='linear', axis=0,␣
↪limit_direction='both')
```

```
time: 156 ms
```

[32]:
```
cnbc_intersect = pd.read_csv(os.path.join(model_op, 'cnbc_X.csv'),␣
↪delimiter=",", header=0, index_col=0)
cnbc_intersect['cellids'] = cnbc_intersect['cellids'].apply(lambda x: mktpl(x))
```

```
time: 62 ms
```

[33]:
```
for ien, i in enumerate(range(stress_period_start, stress_period_end)):
    stress_period_data = []
    for j, cid in enumerate(cnbc_intersect.cellids):

        if not idomain[cid[0],cid[1]] == 1:
            continue
        if cnbc_intersect.loc[j, 'steady'] > 0:
            concentr = float(cnbc_intersect.loc[j, 'F'])
        else:
            concentr = lakes.iloc[i, 1]

        stress_period_data.append([1, int(cid[0]+1), int(cid[1]+1),␣
↪format(concentr, ".10E")])

    np.savetxt(os.path.join(model_ws, '_cnbc_stress_period_data_' + str(i) + '.
↪txt'), stress_period_data, delimiter=' ', fmt="%s")
```

```
time: 1min 12s
```

03_RUNMF-classic_

June 9, 2021

1 Setup GWF and GWT Model and run Modflow

WLF Modflow Model

```
[1]: from prop import *

     def dict_reorder(item):
         return {k: dict_reorder(v) if isinstance(v, dict) else v for k, v in␣
      ↪sorted(item.items())}

     def mktpl(stringinput):
         return tuple(map(int, stringinput.split('(')[1].split(')')[0].split(', ')))

     %load_ext autotime
```

```
<IPython.core.display.HTML object>

time: 0 ns
```

1.1 Definition of Simulation

```
[2]: sim = flopy.mf6.MFSimulation(sim_name=modelname, version='mf6',␣
      ↪exe_name=mf6ExeName,
                                  sim_ws=model_ws)

     tdis = flopy.mf6.ModflowTdis(sim, pname='tdis', time_units='DAYS',␣
      ↪nper=stress_periods,
                                  perioddata=([(1, 1, 1.0)]*stress_periods),
                                  start_date_time='01-01-1993')
```

```
time: 94 ms
```

```
[3]: gwf_model = flopy.mf6.ModflowGwf(sim, modelname=modelname, model_nam_file='{}.
      ↪nam'.format(modelname), save_flows=True)

     ims = flopy.mf6.ModflowIms(sim, print_option='SUMMARY',
                                complexity='COMPLEX',
                                outer_maximum=1000, under_relaxation='NONE',
                                inner_maximum=100,
                                rcloserecord=0.1, linear_acceleration='BICGSTAB',
```

1

```
                                         scaling_method='NONE', reordering_method='NONE',
                                         relaxation_factor=0.97)

sim.register_ims_package(ims, [gwf_model.name])
```

```
time: 0 ns
```

1.2 Definition of Internal Packages

[4]:
```
shp_boundary = sf.Reader(os.path.join(shppath, 'LFW_active'))
boundary = np.array(shp_boundary.shapeRecords()[0].shape.points)

Lx = boundary.max(axis=0)[0]-boundary.min(axis=0)[0]
Ly = boundary.max(axis=0)[1]-boundary.min(axis=0)[1]
nlay = 1
delr, delc = gridsize, gridsize
nrow = int(round(Ly / delr + .49))
ncol = int(round(Lx / delc + .49))
top = 1000
xll=boundary.min(axis=0)[0]
yll=boundary.min(axis=0)[1]

dis = flopy.mf6.ModflowGwfdis(gwf_model, pname='dis', length_units='METERS',␣
 ↪nlay=nlay, nrow=nrow, ncol=ncol, delr=delr,
                              delc=delc, top=top, xorigin=xll, yorigin=yll,
                              filename='{}.dis'.format(modelname))

ic = flopy.mf6.ModflowGwfic(gwf_model, pname='ic', filename='{}.ic'.
 ↪format(modelname))

npf = flopy.mf6.ModflowGwfnpf(gwf_model, pname='npf', icelltype=[1],
                              save_flows=True,
                              save_specific_discharge=True,
                              save_saturation=True,
                              filename='{}.npf'.format(modelname))

sto = flopy.mf6.ModflowGwfsto(gwf_model, ss=0.0001, storagecoefficient=False,␣
 ↪iconvert=[1], transient=True)

idomain = flopy.mf6.ModflowGwfdis.idomain.empty(gwf_model, layered=True,␣
 ↪default_value=-1)
idomain['data'][0] = np.asarray(dt.fread(os.path.join(model_op, 'idomain.csv')))
dis.idomain = idomain['data']

ich = flopy.mf6.ModflowGwfic.strt.empty(gwf_model, layered=True)
ich['data'] = np.asarray(dt.fread(os.path.join(model_op, 'strthead.txt')))
ic.strt = ich
```

```
k1 = flopy.mf6.ModflowGwfnpf.k.empty(gwf_model, layered=True)
k1['data'] = np.asarray(dt.fread(os.path.join(model_op, 'npf_hydcond.txt')))
npf.k = k1

sy = flopy.mf6.ModflowGwfsto.sy.empty(gwf_model, layered=True)
sy['data'] = np.asarray(dt.fread(os.path.join(model_op, 'sy.txt')))
sto.sy = sy

# botm elevation
botm = flopy.mf6.ModflowGwfdis.botm.empty(gwf_model, layered=True)
botm['data'] = np.asarray(dt.fread(os.path.join(model_op, 'botmelev.txt')))
dis.botm = botm

# top elevation
top = flopy.mf6.ModflowGwfdis.botm.empty(gwf_model, layered=True)
top['data'] = np.asarray(dt.fread(os.path.join(model_op, 'topelev.txt')))

oc = flopy.mf6.ModflowGwfoc(gwf_model,
                            budget_filerecord=['{}.bud'.format(modelname)],
                            head_filerecord=['{}.hds'.format(modelname)],
                            headprintrecord=[('COLUMNS', ncol, 'WIDTH', 15,␣
 ↪'DIGITS', 6, 'SCIENTIFIC')],
                            saverecord=[('HEAD', 'LAST'), ('BUDGET', 'LAST')])
```

```
time: 219 ms
```

1.3 Definition of Modflow Stress Packages

River Package

[5]:
```
riv_files = [f for f in os.listdir(model_ws) if "riv_stress_period_data_" in f␣
 ↪and f.endswith(".txt")]

riv_spd = {}
for i in riv_files:
    riv_spd[int(i.split('_')[-1].split('.')[0])] = {'filename': i, 'check':␣
 ↪False}

riv_spd = dict_reorder(riv_spd)

riv = flopy.mf6.ModflowGwfriv(gwf_model, stress_period_data=riv_spd,␣
 ↪auxiliary=['lkg_mult'], auxmultname='lkg_mult', boundnames=True)
```

```
time: 2min 38s
```

[6]:
```
drn_files = [f for f in os.listdir(model_ws) if "drn_stress_period_data_" in f␣
 ↪and f.endswith(".txt")]
```

```
drn_spd = {}
for i in riv_files:
    drn_spd[int(i.split('_')[-1].split('.')[0])] = {'filename': i, 'check':␣
↪False}

drn_spd = dict_reorder(drn_spd)

drn = flopy.mf6.ModflowGwfdrn(gwf_model, stress_period_data=drn_spd,␣
↪auxiliary=['lkg_mult'], auxmultname='lkg_mult', boundnames=True)
```

```
time: 2min 23s
```

Well Package

[7]:
```
wel_files = [f for f in os.listdir(model_ws) if "wel_stress_period_data_" in f␣
↪and f.endswith(".txt")]

wel_spd = {}
for i in wel_files:
    wel_spd[int(i.split('_')[-1].split('.')[0])] = {'filename': i, 'check':␣
↪False}

wel_spd = dict_reorder(wel_spd)

wel = flopy.mf6.ModflowGwfwel(gwf_model, print_input=True,␣
↪stress_period_data=wel_spd)
```

```
time: 1min 3s
```

Constant Head Boundary Package

[8]:
```
chb_files = [f for f in os.listdir(model_ws) if "chd_stress_period_data_" in f␣
↪and f.endswith(".txt")]

chb_spd = {}
for i in chb_files:
    chb_spd[int(i.split('_')[-1].split('.')[0])] = {'filename': i, 'check':␣
↪False}

chb_spd = dict_reorder(chb_spd)

chd = flopy.mf6.ModflowGwfchd(gwf_model, stress_period_data=chb_spd)
```

```
time: 1min 18s
```

Recharge Package

[9]:
```
rch_files = [f for f in os.listdir(model_ws) if "rch_stress_period_data_" in f␣
↪and f.endswith(".txt")]
```

```
rch_aux_files = [f for f in os.listdir(model_ws) if "rch_N_aux_" in f and f.
↪endswith(".txt")]
```

```
time: 93 ms
```

```
[10]: rchspd = {}
for i in rch_files:
    rchspd.update({int(i.split('_')[-1].split('.')[0]): os.path.join(i)})

rchspd = dict_reorder(rchspd)

rchaux = {}
for i in rch_aux_files:
    rchaux.update({int(i.split('_')[-1].split('.')[0]): os.path.join(i)})

rchaux = dict_reorder(rchaux)
```

```
time: 31 ms
```

```
[11]: rch = flopy.mf6.ModflowGwfrcha(gwf_model, pname='rch_1', readasarrays=True,␣
↪auxiliary=['rch_N_aux'], recharge=rchspd, aux=rchaux)
```

```
time: 1min 15s
```

Observation Package

```
[12]: obsX = pd.read_csv(os.path.join(model_op, 'X_obs.csv'), delimiter=",",␣
↪header=0, index_col=0)

obs_list = []
for i, cid in enumerate(obsX.cellids):
    cx = int(cid.split(',')[0].split('(')[1])
    cy = int(cid.split(', ')[1].split(')')[0])
    obs_list.extend([['calc_heads.csv', obsX.loc[i, 'Name'], 'head', (0, cx,␣
↪cy)]])

obs_dict = defaultdict(list)
for filename, obsname, obstype, cellid in obs_list:
    obs_dict[filename].append([obsname, obstype, cellid])

obsheads = flopy.mf6.modflow.mfutlobs.ModflowUtlobs(gwf_model, digits=10,␣
↪print_input=True, continuous=obs_dict, filename='wlf-wie.obs')
```

```
time: 15 ms
```

```
[13]: obs_feflow = pd.read_csv(os.path.join(obsdata, 'obs_feflow.csv'),␣
↪delimiter=",", header=0, index_col=0).drop(['date'], axis=1)
obs_feflow = obs_feflow.iloc[stress_period_start:stress_period_end,:].
↪reset_index(drop=True)
obs_feflow.index.name = 'time'
obs_feflow.to_csv(os.path.join(model_ws,'feflow_heads.csv'), index=True,␣
↪header=True)
```

```
time: 141 ms
```

1.4 Transport Model

[14]:
```
tmodelname = 'gwt_' + modelname
gwt_model = flopy.mf6.ModflowGwt(sim, modelname=tmodelname, model_nam_file='{}.
↪nam'.format(tmodelname), version='mf6', exe_name=mf6ExeName)

gwt_model.name_file.save_flows = True
```

```
time: 0 ns
```

[15]:
```
sim.register_ims_package(ims, [gwt_model.name])

gwtoc = flopy.mf6.ModflowGwtoc(gwt_model,
                            budget_filerecord="{}.cbc".format(tmodelname),
                            concentration_filerecord="{}.ucn".
↪format(tmodelname),
                            concentrationprintrecord=[("COLUMNS", 10, "WIDTH",␣
↪15, "DIGITS", 6, "GENERAL")],
                            saverecord = [('CONCENTRATION', 'ALL'), ('BUDGET',␣
↪'ALL')],
                            printrecord = [('CONCENTRATION', 'LAST'),␣
↪('BUDGET', 'LAST')])
```

```
time: 16 ms
```

[16]:
```
gwtdis = flopy.mf6.ModflowGwtdis(gwt_model,
            nlay=nlay,
            nrow=nrow,
            ncol=ncol,
            delr=delr,
            delc=delc,
            top=top,
            botm=botm,
            idomain=idomain['data'],
            filename="{}.dis".format(tmodelname),
        )
```

```
time: 16 ms
```

[17]:
```
#n_mt = flopy.mf6.ModflowGwtmst.porosity.empty(gwt_model)
n_mt = np.asarray(dt.fread(os.path.join(model_op, 'n_tr.txt')))
#n_mt['data'] = np.asarray(dt.fread(os.path.join(model_op, 'sy.txt')))
mst = flopy.mf6.ModflowGwtmst(gwt_model, porosity=n_mt, save_flows=True)
```

```
#strt_mt = flopy.mf6.ModflowGwtic.strt.empty(gwt_model, layered=True)
strt_mt = np.asarray(dt.fread(os.path.join(model_op, 'imc_tr.txt')))
icmt = flopy.mf6.ModflowGwtic(gwt_model, strt=strt_mt)
```

```
time: 125 ms
```

[18]:
```
adv = flopy.mf6.ModflowGwtadv(gwt_model, scheme='UPSTREAM')
dsp = flopy.mf6.ModflowGwtdsp(gwt_model, xt3d_off=False, alh=5., alv=5., ath1=.
 ↪5, atv=.5)
```

```
time: 0 ns
```

[19]:
```
sourcerecarray = [
    ("rch_1", "AUX", "rch_N_aux"),
]
ssm = flopy.mf6.ModflowGwtssm(gwt_model, sources=sourcerecarray)
```

```
time: 15 ms
```

[20]:
```
cnc_files = [f for f in os.listdir(model_ws) if "cnbc_stress_period_data_" in f␣
 ↪and f.endswith(".txt")]

cnc_spd = {}
for i in cnc_files:
    cnc_spd[int(i.split('_')[-1].split('.')[0])] = {'filename': i, 'check':␣
 ↪False}

cnc_spd = dict_reorder(cnc_spd)

cnc = flopy.mf6.ModflowGwtcnc(gwt_model, stress_period_data=cnc_spd)
```

```
time: 3min 49s
```

[21]:
```
gwfgwt = flopy.mf6.ModflowGwfgwt(sim, exgtype='GWF6-GWT6', exgmnamea=modelname,␣
 ↪exgmnameb=tmodelname, filename='{}.gwfgwt'.format(modelname)  )
```

```
time: 0 ns
```

1.5 Write Simulation Files

[14]:
```
sim.simulation_data.max_columns_of_data = ncol
#sim.set_all_data_external(True)
sim.write_simulation()
```

```
writing simulation…
  writing simulation name file…
  writing simulation tdis package…
  writing ims package ims_-1…
  writing model wlf-wie…
    writing model name file…
```

7

```
    writing package dis…
    writing package ic…
    writing package npf…
    writing package sto…
    writing package oc…
    writing package riv_0…
INFORMATION: maxbound in ('gwf6', 'riv', 'dimensions') changed to 478 based on
size of stress_period_data
    writing package drn_0…
INFORMATION: maxbound in ('gwf6', 'drn', 'dimensions') changed to 478 based on
size of stress_period_data
    writing package wel_0…
INFORMATION: maxbound in ('gwf6', 'wel', 'dimensions') changed to 11 based on
size of stress_period_data
    writing package chd_0…
INFORMATION: maxbound in ('gwf6', 'chd', 'dimensions') changed to 137 based on
size of stress_period_data
    writing package rch_1…
    writing package obs_0…
time: 6min 39s
```

1.6 Run Modflow

[15]:
```
success, buff = sim.run_simulation()
```

```
FloPy is using the following  executable to run the model:
../../EXE/mf6.2.1/bin/mf6.exe
                                   MODFLOW 6
                U.S. GEOLOGICAL SURVEY MODULAR HYDROLOGIC MODEL
                            VERSION 6.2.1 02/18/2021

   MODFLOW 6 compiled Feb 18 2021 08:24:05 with IFORT compiler (ver. 19.10.2)

This software has been approved for release by the U.S. Geological
Survey (USGS). Although the software has been subjected to rigorous
review, the USGS reserves the right to update the software as needed
pursuant to further analysis and review. No warranty, expressed or
implied, is made by the USGS or the U.S. Government as to the
functionality of the software and related material nor shall the
fact of release constitute any such warranty. Furthermore, the
software is released on condition that neither the USGS nor the U.S.
Government shall be held liable for any damages resulting from its
authorized or unauthorized use. Also refer to the USGS Water
Resources Software User Rights Notice for complete use, copyright,
and distribution information.


 Run start date and time (yyyy/mm/dd hh:mm:ss): 2021/06/09  8:47:12
```

```
Writing simulation list file: mfsim.lst
Using Simulation name file: mfsim.nam

   Solving:  Stress period:     1    Time step:     1
   Solving:  Stress period:     2    Time step:     1
   Solving:  Stress period:     3    Time step:     1
   Solving:  Stress period:     4    Time step:     1
   Solving:  Stress period:     5    Time step:     1
   Solving:  Stress period:     6    Time step:     1
   Solving:  Stress period:     7    Time step:     1
   Solving:  Stress period:     8    Time step:     1
   Solving:  Stress period:     9    Time step:     1
   Solving:  Stress period:    10    Time step:     1
   Solving:  Stress period:    11    Time step:     1
   Solving:  Stress period:    12    Time step:     1
   Solving:  Stress period:    13    Time step:     1
   Solving:  Stress period:    14    Time step:     1
   Solving:  Stress period:    15    Time step:     1
   Solving:  Stress period:    16    Time step:     1
   Solving:  Stress period:    17    Time step:     1
   Solving:  Stress period:    18    Time step:     1
   Solving:  Stress period:    19    Time step:     1
   Solving:  Stress period:    20    Time step:     1
   Solving:  Stress period:    21    Time step:     1
   Solving:  Stress period:    22    Time step:     1
   Solving:  Stress period:    23    Time step:     1
   Solving:  Stress period:    24    Time step:     1
   Solving:  Stress period:    25    Time step:     1
   Solving:  Stress period:    26    Time step:     1
   Solving:  Stress period:    27    Time step:     1
   Solving:  Stress period:    28    Time step:     1
   Solving:  Stress period:    29    Time step:     1
   Solving:  Stress period:    30    Time step:     1
   Solving:  Stress period:    31    Time step:     1
   Solving:  Stress period:    32    Time step:     1
   Solving:  Stress period:    33    Time step:     1
   Solving:  Stress period:    34    Time step:     1
   Solving:  Stress period:    35    Time step:     1
   Solving:  Stress period:    36    Time step:     1
   Solving:  Stress period:    37    Time step:     1
   Solving:  Stress period:    38    Time step:     1
   Solving:  Stress period:    39    Time step:     1
   Solving:  Stress period:    40    Time step:     1
   Solving:  Stress period:    41    Time step:     1
   Solving:  Stress period:    42    Time step:     1
   Solving:  Stress period:    43    Time step:     1
   Solving:  Stress period:    44    Time step:     1
```

```
    Solving:  Stress period:  3645    Time step:     1
    Solving:  Stress period:  3646    Time step:     1
    Solving:  Stress period:  3647    Time step:     1
    Solving:  Stress period:  3648    Time step:     1
    Solving:  Stress period:  3649    Time step:     1
    Solving:  Stress period:  3650    Time step:     1

 Run end date and time (yyyy/mm/dd hh:mm:ss): 2021/06/09  9:32:47
 Elapsed run time: 45 Minutes, 35.378 Seconds

 Normal termination of simulation.
time: 45min 20s
```

```
[ ]:
```

03_RUNMF-Loop

June 20, 2021

1 Setup Groundwaterflow Model and run Modflow

WLF Modflow Model

```
[1]: from prop import *

def mktpl(stringinput):
    return tuple(map(int, stringinput.split('(')[1].split(')')[0].split(', ')))

%load_ext autotime
```

```
<IPython.core.display.HTML object>

time: 0 ns
```

1.1 Definition of Simulation and Output-control

```
[2]: sim = flopy.mf6.MFSimulation(sim_name=modelname, version='mf6',␣
 ↪exe_name=mf6ExeName,
                             sim_ws=model_ws)

tdis = flopy.mf6.ModflowTdis(sim, pname='tdis', time_units='DAYS', nper=1,
                             perioddata=([(1, 1, 1.0)]*1))

gwf_model = flopy.mf6.ModflowGwf(sim, modelname=modelname, model_nam_file='{}.
 ↪nam'.format(modelname), save_flows=True)


ims = flopy.mf6.ModflowIms(sim, pname='ims', print_option='SUMMARY',
                           complexity='MODERATE',
                           outer_maximum=1000, under_relaxation='NONE',
                           inner_maximum=100,
                           rcloserecord=0.1, linear_acceleration='BICGSTAB',
                           scaling_method='NONE', reordering_method='NONE',
                           relaxation_factor=0.97)

sim.register_ims_package(ims, [gwf_model.name])
```

```
time: 141 ms
```

1.2 Definition of Internal Packages

[3]:
```
shp_boundary = sf.Reader(os.path.join(shppath, 'LFW_active'))
boundary = np.array(shp_boundary.shapeRecords()[0].shape.points)

Lx = boundary.max(axis=0)[0]-boundary.min(axis=0)[0]
Ly = boundary.max(axis=0)[1]-boundary.min(axis=0)[1]
nlay = 1
delr, delc = gridsize, gridsize
nrow = int(round(Ly / delr + .49))
ncol = int(round(Lx / delc + .49))
top = 1000
xll=boundary.min(axis=0)[0]
yll=boundary.min(axis=0)[1]

dis = flopy.mf6.ModflowGwfdis(gwf_model, pname='dis', length_units='METERS',␣
 ↪nlay=nlay, nrow=nrow, ncol=ncol, delr=delr,
                              delc=delc, top=top, xorigin=xll, yorigin=yll,
                              filename='{}.dis'.format(modelname))

ic = flopy.mf6.ModflowGwfic(gwf_model, pname='ic', filename='{}.ic'.
 ↪format(modelname))

npf = flopy.mf6.ModflowGwfnpf(gwf_model, pname='npf', icelltype=[0],
                              save_flows=True,
                              save_specific_discharge=True,
                              save_saturation=True,
                              filename='{}.npf'.format(modelname))

sto = flopy.mf6.ModflowGwfsto(gwf_model, ss=0.0001, storagecoefficient=False,␣
 ↪iconvert=[1], transient=True)

idomain = flopy.mf6.ModflowGwfdis.idomain.empty(gwf_model, layered=True,␣
 ↪default_value=-1)
idomain['data'][0] = np.asarray(dt.fread(os.path.join(model_op, 'idomain.csv')))
dis.idomain = idomain['data']

ich = flopy.mf6.ModflowGwfic.strt.empty(gwf_model, layered=True)
ich['data'] = np.asarray(dt.fread(os.path.join(model_op, 'strthead.txt')))
ic.strt = ich

k1 = flopy.mf6.ModflowGwfnpf.k.empty(gwf_model, layered=True)
k3 = flopy.mf6.ModflowGwfnpf.k33.empty(gwf_model, layered=True)
k1['data'] = np.asarray(dt.fread(os.path.join(model_op, 'npf_hydcond.txt')))
k3['data'] = k1['data'] * .1
npf.k = k1
npf.k33 = k3
```

```
sy = flopy.mf6.ModflowGwfsto.sy.empty(gwf_model, layered=True)
sy['data'] = np.asarray(dt.fread(os.path.join(model_op, 'sy.txt')))
sto.sy = sy

# botm elevation
botm = flopy.mf6.ModflowGwfdis.botm.empty(gwf_model, layered=True)
botm['data'] = np.asarray(dt.fread(os.path.join(model_op, 'botmelev.txt')))
dis.botm = botm

# top elevation
top = np.asarray(dt.fread(os.path.join(model_op, 'topelev.txt')))

oc = flopy.mf6.ModflowGwfoc(gwf_model,
                            budget_filerecord=['{}.cbb'.format(modelname)],
                            head_filerecord=['{}.hds'.format(modelname)],
                            headprintrecord=[('COLUMNS', ncol, 'WIDTH', 15,␣
 ↪'DIGITS', 6, 'GENERAL')],
                            saverecord=[('HEAD', 'ALL'), ('BUDGET', 'ALL')])
```

```
time: 329 ms
```

[4]:
```
#budgIO_Fl = pd.read_csv(os.path.join(bcdata,'budgIO_Tr.csv'), delimiter=",",␣
 ↪header=0, index_col=0).drop(['date'], axis=1)

river_intersect = pd.read_csv(os.path.join(model_op, 'rivX.csv'),␣
 ↪delimiter=",", header=0, index_col=0).fillna(value = 0)

riv_ts = pd.read_csv(os.path.join(bcdata, 'riv_ts.csv'), delimiter=",",␣
 ↪header=0, index_col=0)

obsX = pd.read_csv(os.path.join(model_op, 'X_obs.csv'), delimiter=",",␣
 ↪header=0, index_col=0)

river_intersect['cellids'] = river_intersect['cellids'].apply(lambda x:␣
 ↪mktpl(x))
```

```
time: 2.47 s
```

1.3 Definition of Modflow Stress Packages

[5]:
```
obsX = pd.read_csv(os.path.join(model_op, 'X_obs.csv'), delimiter=",",␣
 ↪header=0, index_col=0)
obs_calc = pd.DataFrame(columns=[obsX.Name], index=range(0,stress_period_end))
```

```
time: 31 ms
```

[6]:
```
rch_ids = pd.read_csv(os.path.join(model_op, 'X_rch.csv'), delimiter=",",␣
↪header=0, index_col=0)

rch_files = [int(f.split('.')[0]) for f in os.listdir(rchdata) if f.endswith(".
↪csv")]

stotrasim_files = []
for file in rch_files:
    stotrasim_files.append([file, dt.fread(os.path.join(rchdata, str(file)+'.
↪csv'), header=True)])

stotrasim_dict = defaultdict(list)
for filename, file in stotrasim_files:
    stotrasim_dict[filename].append([file])
```

```
time: 10.8 s
```

[7]:
```
for cycle in range(0, stress_period_end):

    stress_period_data = []
    for j , cid in enumerate(river_intersect.cellids):

        if not dis.idomain[0,cid[0],cid[1]] == 1:
            continue

        cellid = (0, cid[0], cid[1])
        if river_intersect.loc[j, 'steady'] == 0:
            stage = riv_ts.loc[cycle, str(river_intersect.loc[j, 'F']).split('.
↪')[0]]
        else:
            stage = float(river_intersect.loc[j, 'F'])

        botmr = dis.botm[0, cid[0], cid[1]] + 0.01

        if stage < botmr:
            stage = botmr

        if stage > ich['data'][cid[0], cid[1]]:
            lkg = river_intersect.loc[j, 'lkg_in'] * river_intersect.loc[j,␣
↪'lengths']

        elif river_intersect.loc[j, 'lengths'] < gridsize:
            lkg = river_intersect.loc[j, 'lkg_out'] * gridsize
        else:
            lkg = river_intersect.loc[j, 'lkg_out'] * river_intersect.loc[j,␣
↪'lengths']
```

```
        stress_period_data.append([cellid, stage, lkg, botmr])

    spd = {}
    spd[0] = stress_period_data

    riv = flopy.mf6.ModflowGwfriv(gwf_model, stress_period_data=spd)


    #### Well Package

    spd = {}
    fname = os.path.join(f"_wel_stress_period_data_{cycle}.txt")
    spd[0] = {'filename': fname, 'check': False}
    wel = flopy.mf6.ModflowGwfwel(gwf_model, print_input=True,␣
↪stress_period_data=spd)

    #### Constant Head Boundary Package

    spd = {}
    fname = os.path.join(f"_chd_stress_period_data_{cycle}.txt")
    spd[0] = {'filename': fname, 'check': False}
    chd = flopy.mf6.ModflowGwfchd(gwf_model, stress_period_data=spd)

    #### Recharge Package

    gwrch = (top - ich['data'])

    stress_period_data = []
    for j, rchid in enumerate(rch_ids.id):
        cx = int(rch_ids.loc[j,'cellids'].split(',')[0].split('(')[1])
        cy = int(rch_ids.loc[j,'cellids'].split(', ')[1].split(')')[0])
        if dis.idomain[0,cx,cy] == 1:
            rchf = stotrasim_dict[rchid][0][0]
            gwnbp = np.asarray(rchf.names[1:]).astype(int)
            gwnb = gwnbp.flat[np.abs(gwnbp - gwrch[cx, cy]*100).argmin()]
            stress_period_data.append([(0, cx, cy), float(rchf[cycle,␣
↪str(gwnb)])/(1000)])

    spd={}
    spd[0] = stress_period_data

    rch = flopy.mf6.ModflowGwfrch(gwf_model, stress_period_data=spd)

    ## Write Simulation Files

    sim.write_simulation()
```

5

```
    ## Run Modflow

    success, buff = sim.run_simulation()

    hdobj = flopy.utils.HeadFile((outputfiles + '.hds'), precision='double')
    head = hdobj.get_data()
    np.savetxt(os.path.join('head', 'calc_head_' + str(cycle) + '.txt'),␣
 ↪head[0], delimiter=' ', fmt="%s")


    for i, cid in enumerate(obsX.cellids):
        cx = int(cid.split(',')[0].split('(')[1])
        cy = int(cid.split(', ')[1].split(')')[0])
        ts = hdobj.get_ts((0, cx, cy))
        obs_calc.iloc[cycle, i] = ts[0][1]

    flopy.mf6.mfmodel.MFModel.remove_package(gwf_model, riv)
    flopy.mf6.mfmodel.MFModel.remove_package(gwf_model, chd)
    flopy.mf6.mfmodel.MFModel.remove_package(gwf_model, wel)
    flopy.mf6.mfmodel.MFModel.remove_package(gwf_model, rch)

    ich['data'] = head[-1]
    ic.strt = ich
    print(">>> cycle: ", cycle)

obs_calc.to_csv(os.path.join(model_ws,'calc_heads.csv'), index=True,␣
 ↪header=True)
```

```
writing simulation…
  writing simulation name file…
  writing simulation tdis package…
  writing ims package ims…
  writing model wlf-wie…
    writing model name file…
    writing package dis…
    writing package ic…
    writing package npf…
    writing package sto…
    writing package oc…
    writing package riv_0…
INFORMATION: maxbound in ('gwf6', 'riv', 'dimensions') changed to 933 based on
size of stress_period_data
    writing package wel_0…
INFORMATION: maxbound in ('gwf6', 'wel', 'dimensions') changed to 11 based on
size of stress_period_data
    writing package chd_0…
INFORMATION: maxbound in ('gwf6', 'chd', 'dimensions') changed to 137 based on
size of stress_period_data
```

```
    writing package riv_0...
INFORMATION: maxbound in ('gwf6', 'riv', 'dimensions') changed to 933 based on
size of stress_period_data
    writing package wel_0...
INFORMATION: maxbound in ('gwf6', 'wel', 'dimensions') changed to 11 based on
size of stress_period_data
    writing package chd_0...
INFORMATION: maxbound in ('gwf6', 'chd', 'dimensions') changed to 137 based on
size of stress_period_data
    writing package rch_0...
INFORMATION: maxbound in ('gwf6', 'rch', 'dimensions') changed to 25507 based on
size of stress_period_data
FloPy is using the following  executable to run the model:
../../EXE/mf6.2.1/bin/mf6.exe
                                   MODFLOW 6
                U.S. GEOLOGICAL SURVEY MODULAR HYDROLOGIC MODEL
                            VERSION 6.2.1 02/18/2021

   MODFLOW 6 compiled Feb 18 2021 08:24:05 with IFORT compiler (ver. 19.10.2)

This software has been approved for release by the U.S. Geological
Survey (USGS). Although the software has been subjected to rigorous
review, the USGS reserves the right to update the software as needed
pursuant to further analysis and review. No warranty, expressed or
implied, is made by the USGS or the U.S. Government as to the
functionality of the software and related material nor shall the
fact of release constitute any such warranty. Furthermore, the
software is released on condition that neither the USGS nor the U.S.
Government shall be held liable for any damages resulting from its
authorized or unauthorized use. Also refer to the USGS Water
Resources Software User Rights Notice for complete use, copyright,
and distribution information.


 Run start date and time (yyyy/mm/dd hh:mm:ss): 2021/06/20  7:13:01

 Writing simulation list file: mfsim.lst
 Using Simulation name file: mfsim.nam

    Solving:  Stress period:     1    Time step:     1

 Run end date and time (yyyy/mm/dd hh:mm:ss): 2021/06/20  7:13:03
 Elapsed run time:  1.596 Seconds

 Normal termination of simulation.
>>> cycle:  3649
time: 1d 18h 12min 58s
```

4340