# Entwicklung und Validierung eines modularen Simulationswerkzeuges für hydrogeologische und geothermische Fragestellungen mit einer interaktiven Schnittstelle zur direkten Implementierung dynamischer und rückkoppelnder Randbedingungen ModSimple

Phase 1

Abschlussbericht über ein Entwicklungsprojekt, gefördert unter dem Az: DBU-AZ 32961/01 von der Deutschen Bundesstiftung Umwelt

Autoren: Dr.-Ing. Mike Müller
Dr.-Ing. Falk Händel
Christian Engelmann, M.Sc.
Dipl.-Ing. Martin Binder

Wissenschaftlicher Beirat: Prof. Dr. Peter Huggenberger
Dr. Peter Börke
PD Dr. habil. Jannis Epting

September 2020

06/02

# Projektkennblatt
der
# Deutschen Bundesstiftung Umwelt

| Az | 32961/01 | Referat | 23 | Fördersumme | 122.953 € |
|---|---|---|---|---|---|

| | |
|---|---|
| **Antragstitel** | **Entwicklung und Validierung eines modularen Simulationswerkzeuges für hydrogeologische und geothermische Fragestellungen mit einer interaktiven Schnittstelle zur direkten Implementierung dynamischer und rückkoppelnder Randbedingungen {ModSimple}** |
| **Stichworte** | Simulation, Wasser/Gewässer |

| Laufzeit | Projektbeginn | Projektende | Projektphase(n) |
|---|---|---|---|
| **30 Monate** | **01.01.2018** | **30.06.2020** | **1** |

Zwischenberichte


| | | |
|---|---|---|
| **Bewilligungsempfänger** | hydrocomputing GmbH & Co. KG<br>Zur Schule 20<br>04158 Leipzig | Tel 0341 525 599 54<br>Fax 0341 520 4495<br>Projektleitung<br>Dr.-Ing. Mike Müller<br>Bearbeiter<br>Dr.-Ing. Mike Müller |
| **Kooperationspartner** | TU Dresden (Institut für Grundwasserwirtschaft)<br>Bergstraße 66<br>01062 Dresden | |


## *Zielsetzung und Anlass des Vorhabens*

Das Grundwasser dient weltweit als unersetzliche und schützenswerte Ressource für die Wasserversorgung sowie als Energieträger. Nicht nur bezüglich des Chemismus, sondern auch bezüglich des thermischen Zustandes werden zunehmend nachteilige Veränderungen beobachtet, insbesondere im urbanen Raum. Ein Grund mag in dem erheblichen Bedeutungszugewinn der Geothermie in den letzten Jahrzehnten liegen; ein entsprechendes nachhaltiges Management ist unverkennbar notwendig. Bei der Planung vieler kleiner geothermischer Anlagen sind jedoch teilweise sehr komplexe Untergrundverhältnisse, technische Infrastrukturen sowie die gegenseitige Beeinflussung der Anlagen zu beachten. Dazu sind numerische Modelle des Stoff- und Wärmetransportes als Bewertungswerkzeuge nötig. Die Beachtung der Wirkung von Einzelanlagen in einem numerischen Modell ist allerdings schwierig, da dies häufig Änderungen von oftmals proprietären Programm-Quell-Codes erfordern würde, um die spezifische Wirkung unterschiedlichster technischer Maßnahmen sinnvoll und zugleich auf großer Skala bewerten zu können.
Ziel dieses Vorhabens war die Schaffung von Werkzeugen, die diese Modellierung vereinfachen. Eine Open-Source-Lösung soll einen Modellierer in die Lage versetzen einfach eigene Randbedingungen zu entwickeln, die dynamisch zur Modelllaufzeit reagieren können, da sie in jedem Zeitschritt Zugriff auf alle nötigen Modellinformationen haben. Diese Randbedingungen können z.B. technische Anlagen wie Wärmegewinnungsbrunnen inklusive technischer Steuerregeln, vom Gesamtmodell nicht erfasste kleinräumliche Effekte oder Kopplungen an andere Modelle wie die ungesättigte Zone repräsentieren.

## *Darstellung der Arbeitsschritte und der angewandten Methoden*

Die Arbeiten erfolgten in drei Arbeitspaketen (AP). AP I umfasste die analytische Beschreibung von Wärme- und Stofftransport in Grundwasserleitern. Der Projektpartner TUD recherchierte zahlreiche analytische Lösungen aus diesem Bereich, beurteilte deren Verwendbarkeit und stellte eine umfangreiche Übersicht dazu zusammen. Der Projektpartner hydrocomputing setzte ausgewählte analytische Stoff- &

Wärmetransport-Lösungsleichungen programmtechnisch so um, dass sie in das in AP II entwickelte Programmsystem *ueflow* integrierbar sind.
In AP II entstand das neue Grundwasserströmungsmodell *ueflow*, das dynamische, durch den Nutzer spezifizierbare Randbedingungen ermöglicht. Die erste funktionsfähige Version basierte, wie ursprünglich geplant, auf FiPy. Da MODFLOW 6 mittlerweile wesentlich mehr für die Projektziele wichtige Merkmale bot, kam dieses anstatt FiPy als Basis für *ueflow* zum Einsatz. AP II umfasste außerdem eine Verifizierung von des von hydrocomputing programmierten *ueflow* durch den Projektpartner TUD.
Das Ziel von AP III war die Entwicklung eines Transportmodells auf Basis von *ueflow*. Wegen der Strategieänderung von FiPy zu MODFLOW 6 als Basis für *ueflow* (siehe AP II), wurde dieses AP nicht bearbeitet. Da MODFLOW 6 allerdings bereits ein Transportmodul beinhaltet, sind Teile von AP III indirekt umgesetzt.

## *Ergebnisse und Diskussion*

Im Projekt entstand die Python-basierte Software *ueflow* - mit dynamischen und Benutzer-implementierbaren Randbedingungen. Dabei kam der de-facto-Industriestandard MODFLOW 6 als numerische Grundlage zur Strömungssimulation zum Einsatz. Das in diesem Projekt entwickelte *ueflow*-Modul *pymf6* ermöglicht als Python-Erweiterung den Zugriff auf den Speichermanager von MODFLOW 6. Damit steht ein leistungsfähiges Werkzeug zur Verfügung, dass es dem Modellierer ermöglicht, direkt zur Laufzeit auf alle Modelldaten lesend und schreibend zuzugreifen. Dies stellt eine erhebliche Neuerung dar. Verifizierungen zeigten, dass *pymf6* richtig rechnet. Die Ergebnisse von Modellläufen mit *pymf6* mit dynamischer Anpassung von Randbedingungen durch den Benutzer zu Laufzeit stimmen vollständig mit den Ergebnissen äquivalenter Modellläufe mit MODFLOW 6 ohne Laufzeitbeeinflussung überein.
Ein weiteres Ergebnis des Projektes ist eine umfangreiche Übersicht von relevanten, analytischen Lösungen für den Wärmetransport. Es entstand ein System zur Umsetzung dieser Gleichungen in Python. Wichtige Gleichungen wurden mit diesem System in Python implementiert und sind damit in *ueflow* integrierbar.
Die Implementierung eines numerischen Wärmetransportmodells wurde wegen der Strategieänderung der Nutzung von MODFLOW 6 zu FiPy, nach Zustimmung der DBU, nicht direkt bearbeitet. Es ergibt sich aber ein indirekter Effekt, da MODFLOW 6 bereits Transportalgorithmen enthält, die sich via *pymf6* in *ueflow* nutzen lassen.
Eine derartige Strategieänderung mitten in einem Projekt ist mit dem Risiko behaftet, dass durch die Mehrfacharbeiten die Projektziele nicht erreicht werden können. Da das zugehörige AP III formal nicht bearbeitet wurde, scheint dies auch hier zuzutreffen. Die neue Lösung ist aber in anderer Hinsicht, wie der Kompatibilität zu bestehenden und weit verbreiteten Werkzeugen und dem Vorwissen der Anwender über MODFLOW 6 wesentlich besser als der ursprüngliche Ansatz mit FiPy. Damit überwiegen die Vorteile die Nachteile der Folgen der Strategieänderung deutlich.
Der Einsatz von *ueflow* eröffnet zahlreiche Umweltentlastungspotenziale. Mit der hiermit abgeschlossenen Phase 1 sind Anwendungen von *ueflow* für die Moorrenaturierung, optimierte Bewässerung in der Landwirtschaft oder einfachere Abbildung der vadosen Zone möglich. Nach Phase 2 werden mit *ueflow* bessere Planungen und Nutzung von insbesondere kleineren, urbanen Geothermieanlagen möglich sein.

## *Öffentlichkeitsarbeit und Präsentation*

Projektpartner präsentierten Zwischenergebnisse mit Vorträgen und Posterbeiträgen auf der Konferenz KGM-MODREG 2018, der Konferenz MODFLOW and More 2019, der GeoPython 2019 und EuroSciPy 2019. Dabei kam es zu sehr fruchtbaren Gesprächen zu inhaltlichen und programmtechnischen Aspekten. So kam es auf der MODFLOW and More zu einem direkten Austausch mit den Kernentwicklern von MODFLOW 6. Dieser Gedankenaustausch war die Ursache für die oben erwähnte Strategieänderung und Nutzung von MODFLOW 6 als Grundlage für das numerische Modell.

## *Fazit*

Das Projekt konnte erfolgreich abgeschlossen werden. Mit *ueflow* ist ein Werkzeug entstanden, dass einer wesentlich flexiblere Grundwasserströmungsmodellierung ermöglicht. Die Umsetzung der dynamischen Randbedingungen in *ueflow* funktioniert zuverlässig. Die recherchierten und programmtechnisch umgesetzten analytischen Lösungen bieten gute Ansatzpunkte für Modellierer unterschiedliche dynamische Randbedingungen selbst zu gestalten.
Damit existiert eine solide Grundlage für die Erweiterung von *ueflow* auf Transportprozesse und die Anwendung auf einen Standort. Diese Arbeiten sollen in Phase 2 dieses Projektes erfolgen.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Verzeichnis von Begriffen, Abkürzungen und Definitionen

## Begriffe

**MODFLOW 6**
: weit verbreite Software für die Grundwassermodellierung, vom USGS entwickelt und kostenfrei vertrieben

## Abkürzungen

**MF6**
: MODFLOW 6

**pymf6**
: Python-Interface zu MODFLOW 6, in diesem Projekt entwickelt

**RB**
: Randbedingung

**ueflow**
: **U**ser **E**xtensible **Flow** Model

# 1. Zusammenfassung

Das Projekt *ModSimple* entwickelt benutzerfreundliche Software-Werkzeuge zur Abbildung von Grundwasserströmung sowie Stoff- und Wärmetransport. Eine wichtige Anwendung dieser Werkzeuge ist die effiziente Bewertung von geothermischen Maßnahmen in urbanen Räumen.

Im Rahmen dieses Projektes wurden:

- umfangreiche Recherchen zu analytischen Wärmetransportgleichungen und geothermischen Randbedingungen durchgeführt,
- davon ausgewählte Gleichungen mit einem Python-basierten System softwaretechnisch umgesetzt,
- *pymf6* entwickelt, das *MODFLOW6* als Python-Modul verfügbar macht und die Beeinflussung einer Vielzahl von *MODFLOW6*-Modell-Variablen zur Laufzeit mit Python-Programmen erlaubt und
- *ueflow* (user-extensible flow model) für die Grundwasserströmungsmodellierung auf der Basis von *pymf6* entwickelt, das die Integration der programmierten analytischen Gleichungen ermöglicht.

*ueflow* (user-extensible flow model) ist in der Lage, Strömungsprozesse im Grundwasser prozessbasiert und recheneffizient abzubilden. Über die interaktive Schnittstelle *pymf6* ist es dem Anwender möglich, spezielle, für das Ressourcenmanagement notwendige Randbedingungen sowie Quell- und Senkenterme, in *MODFLOW6*, den de-facto-Industriestandard der numerischen Grundwassersimulation, einzubinden.

Im Rahmen der erfolgreichen 1. Projektphase wurden mehrere wichtige Grundlagen für die künftige Weiterentwicklung gelegt. In der 2. Projektphase soll die Software *ueflow* auf Stoff- und Wärmetransportprozesse durch die Einbindung der *MODFLOW6*-Transportkomponenten erweitert werden. Weiterhin soll *ueflow* für ein Standortmodell eines urbanen Raumes zum Einsatz kommen. *pymf6* und *ueflow* sind als Open-Source-Software konzipiert. Sie werden Anwendern nach der zweiten Projektphase kostenfrei zur Verfügung stehen.

Dieses Projekt hat die Deutsche Bundesstiftung Umwelt unter dem Aktenzeichens 32961/01 gefördert. Die durchführenden Partner waren die hydrocomputing GmbH & Co KG, Leipzig (Bewilligungsempfänger) und die TU Dresden, Institut für Grundwasserwirtschaft (Kooperationspartner).

# 2. Einleitung

Urbane Räume sind hinsichtlich ihres Stoff- und Wärmeausbreitungverhaltens im Untergrund als sehr komplex einzuschätzen. Betrachtet man die Hydrologie und Hydrogeologie solcher Ballungsräume auf holistischer Skala, so können diese in die folgenden Teilkompartimente unterteilt werden: die atmosphärische Grenzschicht, die Erdoberfläche (inklusive des Vegetationsbestandes, der Oberflächengewässer und anthropogener Bebauung), die variabel wassergesättigte Bodenzone (auch als vadose Zone oder Aerationszone bekannt) sowie die Grundwasserzone. Zwischen diesen Kompartimenten existieren zahlreiche, bidirektionale Wechselwirkungsprozesse und damit Austauschvorgänge hinsichtlich Wärme- und Stoffausbreitung (siehe Abbildung 2.1).

Die atmosphärische Grenzschicht leitet als Transitzone die Wettereinflüsse der Atmosphäre (wie Niederschlag und Temperatur) zur Erdoberfläche weiter. In gegensätzlicher Richtung findet u.a. Verdunstung und Staubexposition statt. Die Erdoberfläche selbst zeichnet sich in urbanen Räumen durch eine Vielzahl an Nutzungsarten aus. Dicht besiedelte Bereiche (Gebäude, Industrie- und Gewerbeanlagen und andere versiegelte Flächen) wechseln sich hierbei mit locker bebauten, sowie teilweise auch naturbelassenen Bereichen ab. Dies führt zu einer erheblichen räumlichen Variabilität nicht nur des Grades der Versiegelung und somit der Kapazität zur Infiltration auftreffenden Niederschlags, sondern auch der Intensität anderweitiger Einträge (z.B. Wärmeeinträge). Letztlich in die Bodenzone gelangende Wassermengen (sowie die darin transportierten Stoffe) unterliegen während der Versickerung zahlreichen Umsatzprozessen. Der Versickerungsprozess erfolgt vorrangig vertikal; Partikel und gelöste Stoffe werden somit in Richtung des Grundwasserspiegels transportiert, wobei je nach Situation auch ein Rückhalt und/oder Abbau in den Bodenschichten stattfinden kann. Natürliche wie auch anthropogene Temperatursignale (z.B. beheizte Keller) pausen sich ebenso in vertikaler Richtung durch; mit zunehmender Bodentiefe werden diese infolge der erheblichen Wärmespeicherfähigkeiten des hydrogeologischen Untergrundes abgeschwächt. Künstliche Systeme wie Netzwerke der öffentlichen und privaten Kanalisation sowie Trinkwasserverteilernetze stellen je nach Zustand erhebliche Quellen und Senken für sowohl Wasser- als auch Stoffdargebot dar. Gefahren für das Schutzgut Grundwasser ergeben sich hierbei z.B. durch den Eintrag von hoch-persistenten organischen Schadstoffen.

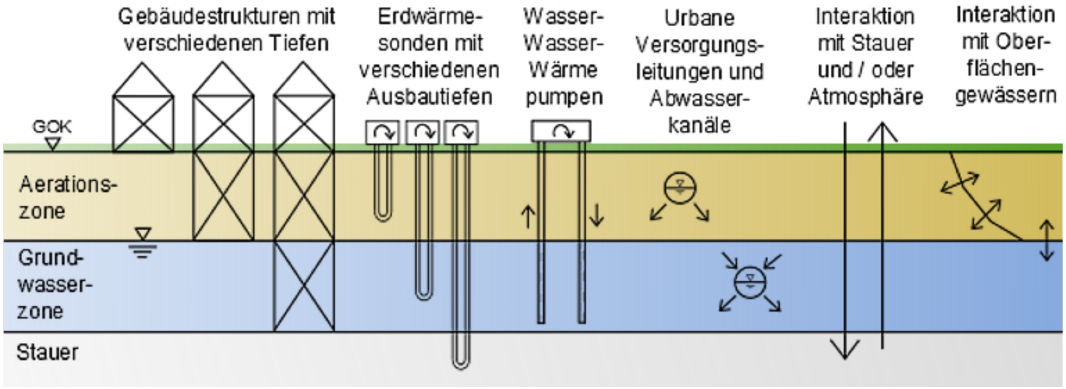



Abbildung 2.1.: Geothermische Randbedingungen als Beispiel für die vielfältigen Austauschprozesse in urbanen Räumen (Köhler et al., 2015)

In der Grundwasserzone findet die maßgebliche, vorrangig horizontale Fließbewegung des unterirdischen Wassers statt. In Abhängigkeit vom hydraulischen Gradienten (Energiegefälle zwischen zwei Punkten) sowie der hydrogeologischen Stratigrafie (insbesondere die räumliche Verteilung der geohydraulischen Leitfähigkeit) werden gelöste Stoffe advektiv-dispersiv mit der Strömung transportiert, an der Gesteinsmatrix adsorbiert sowie teilweise chemisch-biologisch abgebaut oder anderweitig umgewandelt. Analog zum Transport von Stoffen findet der Transport von Wärme statt, welcher zusätzlich von den Wärmeleitungseigenschaften des Gesteins abhängt. In Ergänzung zum natürlichen, saisonal oft fluktuierenden Eingangssignal (seitens der vadosen Zone sowie des Stauers [geothermischer Gradient] führt der Betrieb von Erdwärmesonden (EWS) oder gar ganzen EWS-Feldern zur Ausbildung erheblicher Temperaturanomalien im Grundwasser.

Aus einer möglichen, der Wassergüte abträglichen, Beeinflussung des Schutzgutes Grundwasssers, insbesondere (aber nicht nur) hervorgerufen durch die anthropogenen Komponenten, kann eine Vielzahl an Nutzungskonflikten hervorgehen (z.B. zwischen geothermischen Nutzern, aber auch gegenüber konventionellen Nutzern von Grundwasser als Ressource). Ein nachhaltiges, aber dynamisches Management dieser Ressource wird dringendst benötigt, nicht nur bezüglich des thermischen Haushalts, sondern auch für weitere qualitative und quantitative Parameter.

Dies wiederum setzt das Vorhandensein adäquater Bewertungswerkzeuge voraus. Auf regionaler Skala werden die Wasserströmung sowie der Transport von Stoffen und Wärme in der Grundwasserzone i.d.R. mittels numerischer Modelle simuliert. Dies kann als quasi-3D-Simulation erfol-

gen (2D-System mit einem einzelnen Berechnungslayer einer definierten Mächtigkeit) sowie in Voll-3D; letztere Variante berücksichtigt somit auch vertikale Prozesse. In Abhängigkeit von der Größe des Grundwassersystems können Simulationszeiten von numerischen Voll-3D-Modellen jedoch sehr groß und damit unpraktikabel werden. Die prozessgetreue, detaillierte Abbildung von z.B. Erdwärmesondenfeldern oder undichten Abwasserkanalnetzen erfordert jedoch die Berücksichtigung der Vertikalen. Dieser Komplexitätsanspruch widerspricht den Anforderungen einer nutzerfreundlichen Anwendbarkeit von Modellen auf regionaler urbaner Skala (d.h. akzeptable Rechenzeiten); und erfordert vereinfachte Zwischenlösungen zur Abbildung u.a. der vertikalen Prozesse. Nicht nur Modellrechenzeit-Limitierungen, sondern auch unzureichende Kenntnisse über vorliegende Systemeigenschaften, welche der Parametrisierung von komplexen Randbedingungen dient, stützen daher die Verwendung derartiger, vereinfachter Randbedingungen.

In diesem Kontext war und ist es nun die Aufgabe des Projektes *ModSimple*, ein geeignetes, modular erweiterungsfähiges Softwaresystem zur Modellierung der Grundwasserströmung sowie des Stoff- und Wärmetransports zu entwickeln und, parallel dazu, zu validieren. Diese Modellumgebung soll es in effizienter und anwenderfreundlicher Weise ermöglichen, anthropogene wie auch natürliche Einflüsse auf die Umweltressource Grundwasser unter unmittelbarer Verwendung derartiger Approximationslösungen zu modellieren. Eine der Innovationen von *ModSimple* stellt an dieser Stelle die vorgesehende enge, aber zugleich dynamische Kopplung zwischen einem frei verfügbaren, numerischen Strömungs-, Stoff- und Wärmetransportmodell-Framework (MODFLOW (USGS, 2020c), MT3D (Bedekar, Morway, Langevin & Tonkin, 2016)) und, umgesetzt in der Skriptsprache Python, ausgewählter (semi-)analytischen Randbedingungen des Stoff- und Wärmetransports dar.

Ziele sind hierbei die Erreichung eines praktikablen Kompromisses zwischen möglichst exakter Prozessnachbildung und Recheneffizienz. Weiterhin soll eine möglichst starke Flexibilität bezüglich der gewählten Komplexität der Abbildung von Prozessen (z.B. Transport von Nitrat, gelösten Schadstoffen und Wärme) erhalten werden; bei zugleich hoher Anwenderfreundlichkeit (grafische Benutzeroberfläche, keine fortgeschrittenen Programmierkenntnisse erforderlich). Realisiert wird dies durch die Schaffung einer interaktiven Schnittstelle, welche es dem Nutzer ermöglichen soll, komplexe Randbedingungen und Prozesse (siehe oben) auch auf einer regionalen Betrachtungsskala zu integrieren. Bei bestehenden, teilweise auch sehr kos-

tenintensiven Softwarelösungen, ist dies oftmals nicht möglich oder mit fortgeschrittenen Anforderungen an Programmierkenntnisse verbunden. Die Software soll zudem unter einer Open-Source-Lizenz stehen, um deren Verbreitung und Nutzerkreis zu erhöhen.

Das Projekt *ModSimple* untergliedert sich hierbei in zwei Bearbeitungsphasen. Phase 1 wurde im Rahmen des bisherigen Bewilligungszeitraumes bearbeitet (dieser Bericht). Diese umfasste die Entwicklung und Implementierung eines Python-Packages (*ueflow*) zur Simulation der Grundwasserströmung. Zusätzlich wurden Benchmarks zur Verifizierung von *ueflow* sowie (semi-)analytische/empirische Randbedingungen des Stoff- und Wärmetransports recherchiert und zum Teil implementiert. In Phase 2 (siehe Folgeantrag) soll die Modelliersoftware *ueflow* anhand eines realen Untersuchungsstandortes im urbanen Raum (z.B. Basel Stadt, Kooperation mit Universität Basel, zeitgleich Teil des wissenschaftlichen Beirates) verifiziert werden. Unter Einbindung lokaler Entscheidungsträger sollen vor Ort Untersuchungen zur Umweltbelastung durch Stoff- und Wärmedargebot vorgenommen werden. Durch gezielte Anwendung auf real gemessene Daten sollen die Nutzbarkeit und mögliche Vorteile von *ueflow* gegenüber existierenden Modellansätzen evaluiert werden. Zusätzlich soll eine umfangreiche Dokumentation den Transfer in die Praxis, d.h. zur Anwendung durch Endnutzer, erleichtern.

Im nachfolgenden Kapitel 3 werden der hierfür notwendige Lösungsansatz von *ModSimple* im Detail beschrieben und der Projektablauf gemäß der ursprünglichen Planung kurz dargestellt. Dem folgt eine auf die jeweiligen Arbeitspakete (AP) bezogene Darstellung der wichtigsten Arbeiten und Ergebnisse innerhalb des Projektes.

- Abschnitte 3.2 und 3.3 - AP I : Recherche und programmtechnische Implementierung von Randbedingungen und Quell-/Senkentermen
- Abschnitte 3.4 und 3.5 - AP II : Programmierung und Validierung der Komponente 'Simulation der Grundwasserströmung'
- Abschnitte 3.6 und 3.7 - AP III : Programmierung und Validierung der Komponente 'Simulation des Stoff- und Wärmetransports'

Aus entwicklungsstrategischen Gründen lag der Fokus der Bearbeitung auf den APs I und II.

# 3. Hauptteil

## 3.1. Lösungsansatz von ModSimple

### 3.1.1. Überblick der Arbeiten und Features

Die zur Berechnung der Strömungs- und Transportprozesse benötigten Steuerungsalgorithmen wurden innerhalb des Projekts *ModSimple* recheneffizient programmiert. Das übergeordnete Programmsystem hat den Namen *ueflow* erhalten - dies steht für *user-extensible flow model*. Der modulare Aufbau und die implementierte Schnittstelle zu (semi-)analytischen/empirischen Randbedingungen ermöglichen es der Software *ueflow* (siehe Abbildung 3.8 auf Seite 31), diese auf eine Vielzahl weiterer, umweltrelevanter Fragestellungen anzuwenden (z.B. diffuser Eintrag von Stoffen in Grundwasserleiter). Für den Stoff- und Wärmetransport im Grundwasser relevante Randbedingungen wurden hierfür in der Literatur recherchiert. Im Ergebnis der Recherche wurde eine Vielzahl an (semi-)analytischen und empirischen Lösungsgleichungen bezüglich ihrer Eignung bewertet; ausgewählte Gleichungen wurden für eine Implementierung über die Schnittstelle vorbereitet.

Die folgenden Haupt-Features wurden für *ueflow* definiert (u.a. auch nachzulesen im Posterbeitrag *KGM-MODREG 2018*, siehe Abbildung A.1; siehe auch Vortrag auf der *GeoPython 2019*, siehe Abbildung A.2):

**Open Source**
: Die Software *ueflow* soll unter Open-Source-Lizenz der Allgemeinheit quelloffen zur Verfügung stehen. Dies erhöht die Sichtbarkeit und Anwendung von *ueflow*. Darüber hinaus fördert es die Erweiterbarkeit auf weitere Fragestellungen.

**Dokumentation**
: Zusätzlich zum frei verfügbaren Quellcode soll eine umfassende Dokumentation dem Nutzerkreis zugänglich gemacht werden. Diese umfasst u.a. ein Handbuch und Online-Tutorials.

**Programmiersprache**
: Python soll als Programmiersprache verwendet werden. Neben allgemeinen Aspekten wie klarer Syntax, nützlichen Datentypen, einfacher Modularisierung, guter Fehlerverwaltung sowie Plattformunabhängigkeit ermöglicht Python es, externen Code (z.B. C, C++, Fortran) auf

einfache Art und Weise einzubinden. Letztere Funktionalität ist für die Definition von Schnittstellen relevant.

**Effiziente Datenhaltung**
Die Speicherung von Bewegungs- und Ausgabedaten soll möglichst speicher- und recheneffizient erfolgen. Eine Möglichkeit bietet sich hier mit dem HDF5-Datenbankformat.

**Paralleler Gleichungslöser**
Bei Bedarf soll ein parallelisierter Gleichungslöser zur numerischen Lösung von Differentialgleichungen für Grundwasserströmung sowie Stoff- und Wärmetransport genutzt werden. Die für Python frei verfügbare *FiPy*-Bibliothek stellt hier eine vielversprechende Lösung dar.

**Benchmarking**
Die Funktionalität von *ueflow* soll mittels ausgewählter, automatisierter Benchmarks gegen existierende Industriestandard-Software (z.B. MODFLOW, MT3D) gewährleistet sein. Die Erfüllung dieses Features soll unabhängig sein vom verwendeten Gleichungslöser.

**Schnittstelle zur Kopplung von Randbedingungen**
Eine in Python implementierte Schnittstelle soll es dem Nutzer ermöglichen, (semi-)analytische/empirische Randbedingungen des Stoff- und Wärmetransports zu definieren und mit dem numerischen Stoff- und Wärmetransportmodell zu koppeln.

**Erweiterbarkeit**
Die Software *ueflow* soll durch ihren modularen Aufbau jederzeit um weitere Funktionalitäten (z.B. Strömung und Transport in der ungesättigten Bodenzone) erweiterbar sein.

### 3.1.2. Projektablauf

Der übergeordnete Projektablaufplan von *ModSimple* ist in Abbildung 3.1 dargestellt.

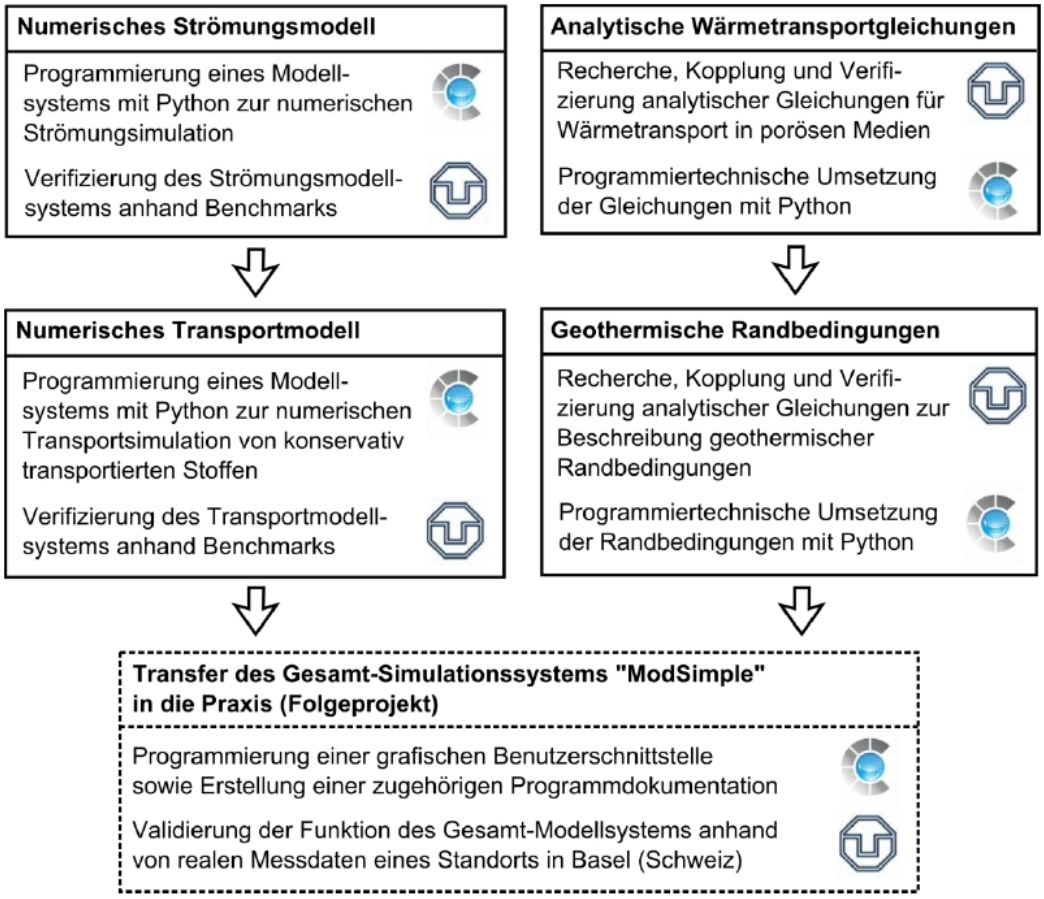



Abbildung 3.1.: Ablaufschema des Projekts ModSimple gemäß Antrag der ersten Phase

## 3.2. AP I/1 - Analytische Beschreibung von Wärme- und Stofftransport in Grundwasserleitern - Erfassung und grundlegende Beurteilung von Berechnungsalgorithmen und Randbedingungen

### 3.2.1. Recherche von (semi-)analytischen/empirischen Randbedingungen für den Stoff- und Wärmetransport

Das AP I/1 beinhaltete eine umfangreiche Recherche von (semi-)analytischen Ansätzen zur Abbildung von Stoff- und Wärmetransportrandbedingungen. Seit der Mitte des 20. Jahrhunderts wurden eine Vielzahl von Studien veröffentlicht, welche sich mit der Beschreibung des Stoff- und Wärmetransports beschäftigten. Randbedingungen (RB) eines Modells spiegeln die hydrologischen Bedingungen innerhalb eines klar begrenzten Gebietes wieder. RB können z.B. den Zufluss und den Abstrom aus dem Modellgebiet beschreiben (hierbei handelt sich um eine Randbedingung der 2. Art). Alternativ können diese RB auch räumlich und/oder zeitlich bekannte Systemvariablen (z.B. Wasserstände an Oberflächengewässern, Temperaturmessungen) festhalten (Randbedingung der 1. Art). Desweiteren existieren Randbedingungen der 3. Art, welche eine Kombination der beiden zuvor genannten RB darstellen (Beispiel: Wasserströmung durch die Kolmationsschicht eines Fließgewässers).

Die Spannbreite von RB reicht hierbei von verhältnismäßig einfachen Ansätzen (z.B. eindimensionale Abbildung der Bodenzone und damit indirekte Kopplung zwischen Grundwasserleiter und atmosphärischen Wärmeeintrag), über RB mittleren Komplexitätsgrades (z.B. zweidimensionale thermische Anbindung von Oberflächengewässern), bis hin zu komplexeren RB-Typen (z.B. Erdwärmesonden, welche darüber hinaus durch Clustering als Erdwärmesondenfelder abgebildet werden können). Unterscheiden lassen sich diese Ansätze in äußere Randbedingungen (z.B. Atmosphäre, Grundwasserstauer, Oberflächenwasser-Grundwasser-Interaktion) sowie interne Quellen / Senken. Geothermische Nutzungen (z.B. Erdwärmesonden, Wasser-Wasser-Wärmepumpen) und urbane Strukturen (Gebäudekomplexe in der ungesättigten und gesättigten Zone, Tiefgaragen, Tunnel) gehören z.B. zur letztgenannten Gruppe. Innerhalb des Projektes ModSimple werden Quell- und Senkenterme (wie z.B. Wasserentnahmen an Brunnen) mathematisch als Randbedingungen betrachtet.

Die Komplexität der verfügbaren Ansätze reicht dabei von empirischen

(Blackblox / Greybox, d.h. Parameter nicht direkt messbar), über semi-analytische (d.h. Parameter teilweise messbar) bis hin zu analytischen (d.h. alle Parameter messbar) Ansätzen. Die Dimensionalität wird meist auf ein- oder zweidimensionaler Ebene betrachtet. Einige der Ansätze sind zeitlich stationär, und andere wiederum instationär. Die Ansätze stellen entweder Punktquellen (direkt) oder flächenhafte (diffus) Quellen und Senken dar.

Während der Recherche wurde eine Vielzahl an Literaturquellen als relevant identifiziert und klassifiziert. Eine entsprechende Datenbank liefert eine Übersicht zu existierenden Ansätzen zur vereinfachten Abbildung oben genannter Randbedingungen. Wichtige Eigenschaften dieser Ansätze wie beispielsweise Dimensionalität, Stationarität, betrachtete Teilprozesse, Eingangsparameter und Ausgabewerte wurden systematisch kategorisiert. Um einige ausgewählte Ansätze mit dem numerischen Stoff- und Wärmetransportmodell zu koppeln, wurden diese Ansätze in separate Python-Skripte implementiert und mittels Parametersensitivitätsanalyse auf Plausibilität geprüft. Tabelle B.1 (im Anhang auf Seite 64) liefert eine Übersicht über die recherchierten Referenzen und darin beschriebene Ansätze.

### 3.2.2. Beschreibung ausgewählter Gleichungen inklusive möglicher Anwendungsszenarien

Ausgewählte Lösungsgleichungen wurden in Python-Code überführt, um diese im späteren Projektverlauf (Phase 2) in den Rechenkern einzubinden. Vier ausgewählte Ansätze seien im Folgenden beschrieben; die erste Gleichung dient als Beispiel für eine Variante der Stofftransportsimulation, drei weitere als Beispiele für Wärmetransportsimulationen. Als Beispiel für eine Python-Implementierung der Ogata-Banks-Gleichung sei auf Abschnitt 3.3 verwiesen, drei weitere Implementierung-Codes (zugehörig zu den Wärmetransport-Gleichungen) befinden sich im Anhang C. Auf eine Detail-Darstellung der Python-Codes der anderen Gleichungen im Fließtext wird zwecks Lesbarkeit hier verzichtet.

#### Konservativer 1-D-Stofftransport: Ogata-Banks-Gleichung

Die Ogata-Banks-Lösungsgleichung (Ogata & Banks, 1961) ermöglicht die Abbildung des 1-D-Transportes eines Stoffes im porösen, homogenen Medium, welcher kontinuierlich an einem bekannten Ort im Zustrom eingeben

wird. Diese 1D-Lösungsgleichung berücksichtigt Advektion, hydrodynamische Dispersion sowie Sorption (linear). Die Dispersion wird über die Gauss'sche Fehlerfunktion *erfc* abgebildet.

$$c(x,t) = \frac{C_0}{2} \times erfc\left(\frac{x - v_a/R \times t}{2 \times \sqrt{\alpha_L \times v_a/R \times t}}\right) + \frac{C_0}{2} \times e^{\frac{x}{\alpha_L}} \times erfc\left(\frac{x + v_a/R \times t}{2 \times \sqrt{\alpha_L \times v_a/R \times t}}\right) \quad (3.1)$$

$c(x,t)$ ist die Konzentration für eine definierte Kombination aus Prognosezeitpunkt $t$ und Prognoseort $x$. $v_a$ ist die Abstandsgeschwindigkeit, welche die mittlere Geschwindigkeit eines Wassermoleküls aufgrund von Advektion beschreibt. $R$ ist der Retardationsfaktor, mit welchem lineare Adsorptionsisothermen eingebunden werden können (ein Wert von 1 für $R$ entspricht konservativem Transport). $\alpha_L$ schließlich entspricht der longitudinalen Dispersion in Fließrichtung.

Anwendungsszenarien:

- Abbildung eines künstlichen Tracerversuches mit kontinuierlicher Eingabe in einem homogenen Strömungsfeld
- Abbildung des Stoffzustromes orthogonal zu einem Modellrand
- Abbildung des Wärmezustromes orthogonal zu einem Modellrand bei vorwiegend konvektiven Bedingungen

### Stationärer 1-D-Wärmetransport über Konduktion und Konvektion um eine vertikale Linienquelle

Diese Gleichung, entnommen aus Hähnlein, Molina-Giraldo, Blum, Bayer und Grathwohl, 2010, ermöglicht - in weitestgehender Analogie zur Ogata-Banks-Gleichung - die Abbildung des 1-D-Transportes von Wärme im porösen, homogenen Medium. Als Quelle fungiert eine vertikale Linie. Berücksichtigt werden hierbei Konvektion, thermische Dispersion sowie Konduktion, wobei die zugehörigen Parameter als temperaturunabhängig und homogen verteilt angesehen werden. Als Ergebnis wird der stationäre Fall im direkten Abstrom ausgegeben, d.h. die Simulationszeit wird als "unendlich lang" betrachtet.

$$\Delta T(x) = \frac{F_L e^{\frac{v_\theta x}{2D_{th}}} K_0 \left(\frac{v_\theta x}{2D_{th}}\right)}{2 \times D_{th} \times c_w \times n \times \rho_w \pi} \tag{3.2}$$

$\Delta T(x)$ ist die Temperaturänderung an einem definierten Punkt im Abstand $x$ zur vertikalen Linienquelle bzw. Liniensenke. $F_L$ ist die Entzugsleistung pro Längeneinheit (in der Vertikalen); hierbei ist eine Angabe der Gesamt-Entzugsleistung $E$ und der Gesamtlänge $L_{th}$ möglich mit $F_L = E/L_{th}$. $n$ stellt die Gesamtporosität dar. $D_{th}$ ist die thermische Dispersion, und ergibt sich aus einer rein konvektiv-dispersiven Komponente und einer konduktiven Komponente (Wärmeleitung). $c_V$ ist die volumetrische Wärmekapazität des Wassers; diese kann auch als Produkt "Dichte des Wassers $\rho_w$" mal "spezifische Wärmekapazität des Wassers $c_w$" angegeben werden. $K_0$ schließlich ist die modifizierte Bessel-Funktion der nullten Ordnung.

Anwendungsszenarien:

- Abbildung des Wärmezustromes orthogonal zu einem Modellrand bei konduktiven oder auch konvektiven Bedingungen
- Abbildung des Wärmetransportes im Abstrom einer senkrechten, geothermischen Anlage unter Annahme einer kontinuierlichen, gleichmäßgien Nutzung

### Stationärer 2-D-Wärmetransport über Konduktion und Konvektion um eine vertikale Linienquelle

Diese Gleichung, ebenfalls entnommen aus Hähnlein et al., 2010, erweitert die zuvor beschriebene 1-D-Lösungsgleichung auf zwei Dimensionen entlang und orthogonal zur Strömungsrichtung. Alle weiteren Annahmen sind analog zur 1-D-Gleichung.

$$\Delta T(x,y) = \frac{F_L e^{\frac{c_v v_f x}{2C_{tx}}} K_0 \left(0.5 c_v v_f \sqrt{\frac{C_{tx} y^2 + C_{ty} x^2}{C_{tx}^2 C_{ty}}}\right)}{2\pi\sqrt{C_{tx} C_{ty}}} \tag{3.3}$$

$\Delta T(x,y)$ ist die Temperaturänderung an einem definierten Punkt im Abstand $x$ (in Abstromrichtung) und Abstand $y$ (senkrecht zur Abstromrichtung) zur vertikalen Linienquelle bzw. Liniensenke. $v_f$ ist die Darcy-Geschwindigkeit bzw. Filtergeschwindigkeit und ist verbunden mit der

Abstandsgeschwindigkeit über $v_f = v_a \times n_e$, mit $n_e$ als durchströmungswirksame bzw. mobile Porosität. $C_{tx}$ und $C_{tx}$ entsprechen den effektiven thermischen Dispersionskoeffizienten in x- und in y-Richtung. Zu deren Berechnung ist die Angabe der kombinierten Wärmeleitfähigkeit aus Gesteinsmatrix und Wasser notwendig; bezüglich der Parametrisierung sei auf entsprechende Literaturwerte verwiesen. Des weiteren gelten die bisherigen Symboldefinitionen.

Anwendungsszenarien: siehe 1-D-Gleichung

### Instationärer radialer Wärmetransport über Konduktion um eine vertikale Linienquelle

Im Unterschied zu den vorigen Wärmetransportgleichungen ermöglicht die nachfolgend beschriebene Gleichung (Hecht-Méndez, Molina-Giraldo, Blum & Bayer, 2010) die instationäre, d.h. zeitabhängige Berechnung der Entwicklung der Wärmeverteilung um eine vertikale Linienquelle. Die Ausgabe erfolgt unter Annahme eines radialen Koordinatensystems. Eine interne Überführung in ein kartesisches Koordinatensystem ist hierbei notwendig, um eine Nutzung als Randbedingung/Quelle/Senke zu realisieren.

$$\Delta T(r,t) = -\frac{F_L \,\mathrm{Ei}\left(-\frac{c_v r^2}{4\lambda_m t}\right)}{4\lambda_m \pi} \tag{3.4}$$

$r$ entspricht dem Radius, d.h. dem Abstand von der vertikalen Linienquelle/-senke; $\lambda_m$ ist die effektive Wärmeleitfähigkeit der Matrix. $Ei$ schließlich stellt die sogenannte Integralexponentialfunktion dar. Die weitere Symbolik entspricht den zuvor beschriebenen Definitionen.

Anwendungsszenarien:

- Abbildung der Wärmeentwicklung um eine Erdwärmesonde außerhalb des eigentlichen durchströmten Grundwasserleiters (z.B. innerhalb eines Geringleiters)
- Abbildung der Wärmeentwicklung um vertikale, geothermische Anlage im Allgemeinen unter Annahme weitestgehend konduktiver Bedingungen
- explizite Abbildung der zeitabhängigen Nutzungscharakteristik einer Erdwärmesonde (ggfs. über Superposition mehrerer Gleichungsinstanzen

## 3.3. AP I/2 - Analytische Beschreibung von Wärme- und Stofftransport in Grundwasserleitern - Programmierung der analytischen Lösungen

### 3.3.1. Grundsätzliche Strategie der Implementierung

Auf Grundlage der Recherche der (semi-)analytischen/empirischen Gleichungen zur Definition von Randbedingungen des Stoff- und Wärmetransports wurden ausgewählte Ansätze in Python implementiert. Ziel war es, eine direkte Kopplung zwischen dem numerischen Stoff- und Wärmetransportmodell (siehe Abschnitt 3.6) sowie der jeweiligen Randbedingung möglichst recheneffizient zu realisieren. Mittels Kenntnis über den Modellstatus zu jedem Zeitschritt über die *pymf6*-Schnittstelle (siehe Abschnitt 3.4) können die Randbedingungen, die auf den analytischen Ansätzen beruhen, dynamisch an das numerische Modell in *ueflow* gekoppelt werden.

### 3.3.2. Beispiel-Implementierungen

Stellvertretend wurden einige der recherchierten Ansätze (siehe Abschnitt 3.2) ausgewählt und jeweils in Python-Code überführt. Dabei kam SymPy (Meurer et al., 2017) zum Einsatz. SymPy ist eine Python-Bibliothek für symbolische Mathematik vergleichbar mit anderen Computer-Algebra-Systemen (CAS). Damit lassen sich die ausgewählten analytischen Gleichung mathematisch behandeln und gleichzeitig mit Python ausführen. Vereinfachungen und Umwandlungen von Gleichungen in symbolischer Form gehört genauso zum Umfang von SymPy wie die Darstellung der Gleichung in mathematischer Notation oder die Umsetzung in effektiven numerischen Code auf Basis von NumPy (van der Walt, Colbert & Varoquaux, 2011).

Die Ausgabe kann als Wert pro Rechenschritt sowie als Gesamtausgabe für alle Rechenschritte erfolgen. Je nach Gleichung können dies Zeitschritte und/oder Ortsschritte sein. Die Eingangsparameter sind Instanzen einer Klasse *Parameter* mit Eigenschaften wie Name, Wert, Einheit, Langname (d.h. ausführlicher Name) und Beschreibung, wobei nur Name und Wert obligatorisch sind. Diese Parameter haben in der von Grundwassermodellierern, die mit Python arbeiten, häufig genutzten Arbeitsumgebung Jupyter Notebook eine erweiterte Darstellung. Damit lassen sie sich komfortabel untersuchen.

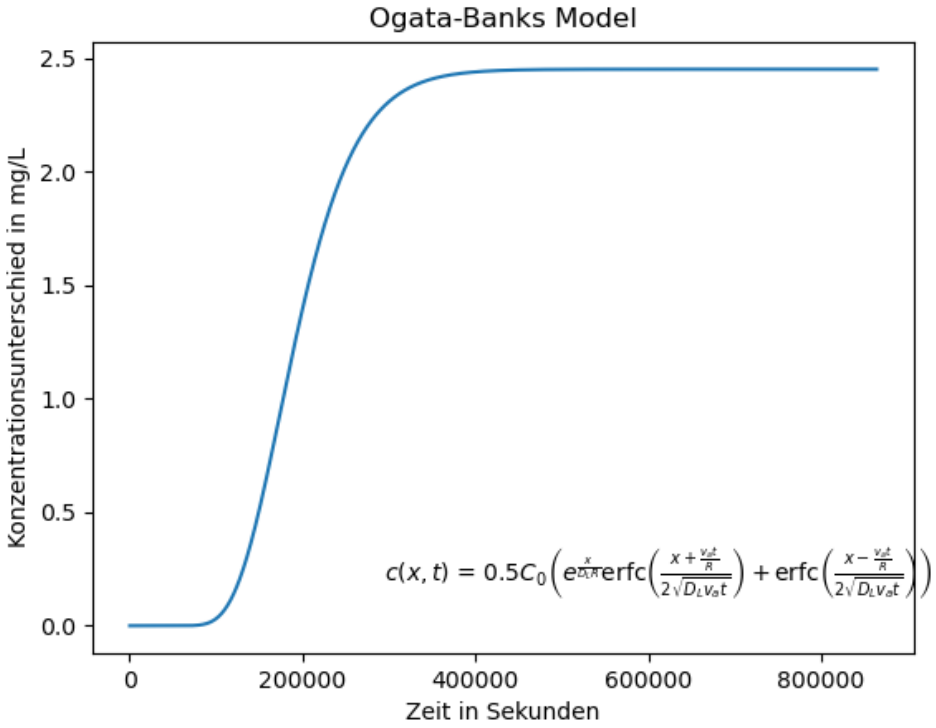



Abbildung 3.2.: Umgesetzte analytische Lösung für das Ogata-Banks-Modell

Im Folgenden sei die Umsetzung der Ogata-Banks-Lösungsgleichung im Detail beschrieben. Weitere Implementierungen mit analoger Grundstruktur können dem Anhang C entnommen werden.

### Umsetzung der Lösungsgleichung als Python-Code

Die Abbildung 3.2 zeigt das Ergebnis des im Folgenden dargestellten Beispiels. Die umgesetzte Gleichung ist ebenfalls im Bild dargestellt. Diese Gleichung ist auch eigenständig abrufbar.

Die Klasse *OgataBanks* mit ihrer Funktion *make_equation* setzt die eigentliche Ogata-Banks-Gleichung als Objekt in Python-Code um. Dabei kommen die Symbole $t$ und $x$ vor. Diese werden später bei der Berechnung durch konkrete Werte für einen Berechnungspunkt in Raum und Zeit ersetzt.

Die Parameter sind als *self.params* definiert. Um die Gleichung zu kürzen, wurde der Kurzname *p* als Alias vergeben, so dass alle Parameter als *p.<parameter_name>* verfügbar sind:

```
class OgataBanks(Model):
    """Ogata-Banks model.
    """

    def make_equation(self):
        """Create the equation.
        """
        p = self.params
        x = self.symbols['x']
        t = self.symbols['t']
        exp = sympy.exp
        erfc = sympy.erfc
        sqrt = sympy.sqrt
        self.equation = (
            0.5 * p.C_0 * (
                erfc((x - (p.v_a / p.R) * t) /
                     (2 * sqrt((p.v_a * p.D_L) * t))) +
            (exp((p.v_a / p.R * x) / (p.v_a * p.D_L)) *
            erfc((x + (p.v_a / p.R) * t) /
                 (2 * sqrt((p.v_a * p.D_L) * t)))
            )))

        return self.equation
```

## Erzeugen einer Modell-Instanz

Die Funktion *make_model* definiert die dafür notwendigen Eingabeparameter als Liste von Parametern. Mit dieser Liste erzeugt diese Funktion eine Instanz der oben gezeigten Klasse *OgataBanks*:

```
def make_model():
    """Create the model.
    """
    param_list = [
    Parameter(
        'C_0',
        value=2.45,
        unit='mg/L',
        long_name='input concentration',
        description='uniformely distributed starting
                                        concentration '),
```

```
    Parameter(
        'R',
        value=1,
        long_name='retardation factor'),
    Parameter(
        'v_a',
        value=0.5 / 86400,
        unit='m/s',
        long_name='pore velocity',
        description=''),
    Parameter(
        'D_L',
        value=0.05,
        unit='m',
        long_name='longitudinal dispersivity',
        description='dispersivity in main flow direction')
    ]

    return OgataBanks(
        param_list,
        model_name='Ogata-Banks Model',
        left_side=r'$c(x,t)$')
```

## Test-Aufruf

Nachfolgende Funktion *test* zeigt einen Aufruf der Funktion *make_model* mit vorher erzeugten NumPy-Arrays als Eingabedaten. Das Ergebnis des Aufrufs der Methode *plot_1d* ist die Darstellung in Abbildung 3.2.

```
def test():
    """A little test.
    """

    t_values = np.linspace(start=864, stop=864000, num=1000)
    x_values = np.array([1.15] * t_values.size)
    ogata_banks = make_model()
    res = ogata_banks.calc_array({'x': x_values, 't': t_values}
                                 )
    ogata_banks.plot_1d(
        x_values=t_values,
        y_values=res,
        xlabel='Zeit in Sekunden',
        ylabel='Konzentrationsunterschied in mg/L',
        file_name='ogata-banks.png'
        )
```

## Vergleich mit einer numerischen Lösung

Die Abbildung 3.3 zeigt eine Durchbruchskurve für die simulierte Konzentration basierend auf dem folgenden Parameterset: $x = 1,15m$, $t = 864s...864000s$ (1000 Schritte), $v_a = 0,5m/d$, $R = 1,0$, $\alpha_L = 0,05m$, $c_0 = 2,45mg/L$. Dies entspricht den Angaben im zuvor gezeigten Python-Code-Ausschnitt. Im Vergleich dazu ist eine numerische Vergleichsrechnung, realisiert über *MT3D* dargestellt. Mit beiden Methoden entsteht das gleiche Ergebnis.

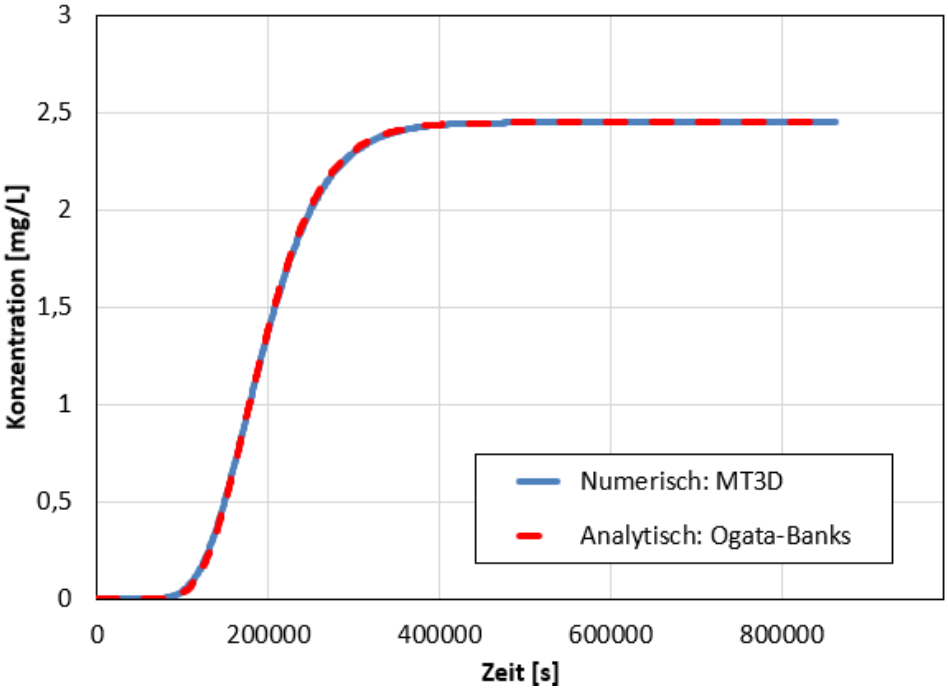



Abbildung 3.3.: Vergleich zwischen analytischer Lösung nach Ogata-Banks und numerischer Abbildung in einem quasi-1-D-Modellsystem mit *MODFLOW* und *MT3D*

## Einbau in *ueflow*

Der Aufruf der Methode *calc_array* erlaubt die Übergabe von Einzelwerten oder gesamten Arrays von Werten. Damit ergeben sich verschiedene Möglichkeiten analytische Ergebnisse in *ueflow* und über *pymf6* dynamisch an die numerische Berechnung als Randbedingungen weiterzugeben. Wenn

sich die Bedingungen über die Zeit ändern, kann für jeden Zeitschritt eine Neuberechnung einer analytische Lösung erfolgen. Wenn sich die Bedingungen nicht ändern, können mehrere Werte einmal berechnet und Schritt für Schritt genutzt werden. Das kann zu einer schnellere Laufzeit führen. Die Kombination beider Methoden, also die Berechnung von Teilbereichen und die Neuberechnung von weiteren Teilbereichen nach Änderung von Bedingungen, ist natürlich möglich.

## Interaktives Arbeiten mit der Lösungsgleichung im Jupyter Notebook

Bei der Umsetzung von neuen analytischen Lösungen in Python oder der Untersuchung, ob sich bestehende Lösungen für das aktuelle Problem eignen, kann die interaktive Arbeit in Jupyter-Notebooks sehr nützlich sein.

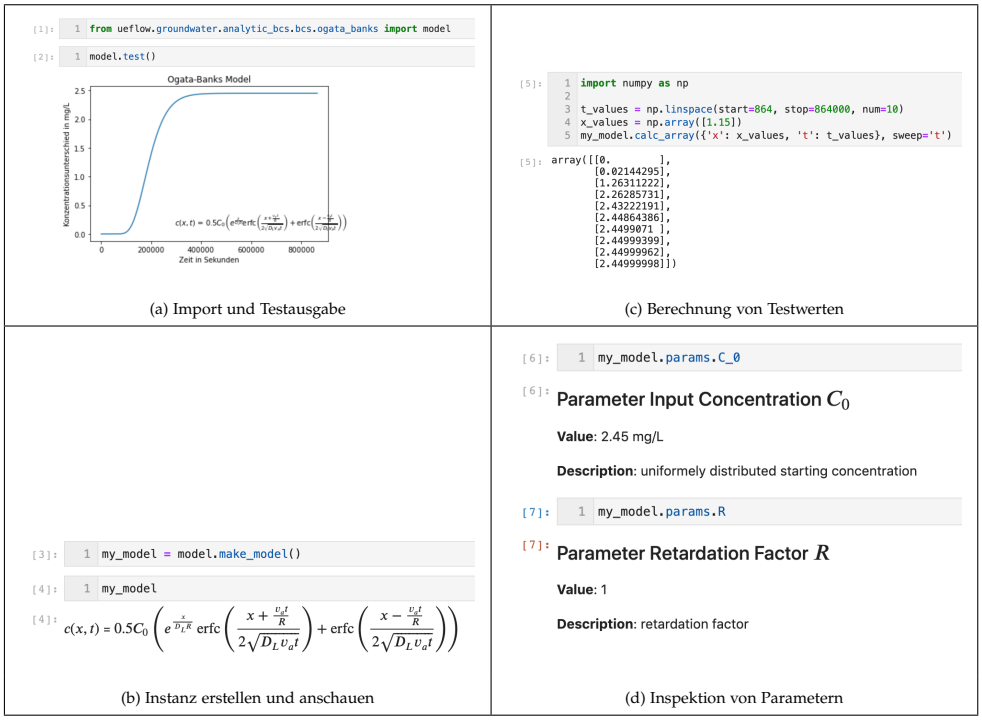



Abbildung 3.4.: Interaktive Arbeit mit einer analytische Lösung am Beispiel des Ogata-Banks-Modell

Abbildung 3.4 zeigt einen Ausschnitt eines typischen Arbeitsablaufes in einem Jupyter Notebook. Weitere Beispiele befinden sich im Anhang C.

Abbildung 3.4(a) zeigt wie nach einem Import des Modells die Funktion *test* verfügbar ist. Deren Ausführung zeigt den Plot interaktiv. Das Notebook bietet auch einen interaktiven Bildmodus, der die Vergrößerung von Plot-Ausschnitten (Zoom) erlaubt.

Die Schritte lassen sich aber einzeln ausführen. Abbildung 3.4(b) zeigt, wie sich eine Instanz einer Modell-Klasse bei Aufruf zurückmeldet: sie stellt sich unmittelbar über Ihre mathematische Gleichung dar. Damit lässt sich gut überprüfen, ob die Gleichung richtig programmiert ist, da sich eine mathematische Gleichung meist besser lesen lässt als die äquivalente Umsetzung in Quelltext.

Um zu überprüfen, ob die Berechnungsergebnisse richtig sind, eignet sich die Methode *calc_array*. Abbildung 3.4(c) zeigt einen Aufruf dieser Methode. Durch das Zuweisen des Ergebnisse an einen Variablennamen lassen sich Berechnungsergebnisse programmatisch weiter untersuchen.

Die Parameter haben wie das Modell eine erweitere Repräsentation und stellen sich im Notebook als formatierter Text dar. Abbildung 3.4(d) zeigt dies für zwei Parameter. Dabei sind auch Formelzeichen mit mathematischer Notation möglich. Die Anzeige der Einheit kann ebenfalls helfen, das Verständnis zur verbessern und mögliche Fehler zu vermeiden.

## 3.4. AP II/1 - Entwicklung der Grundwasserströmungssoftware - Programmierung des Strömungssimulationsalgorithmus mit Python

### 3.4.1. Entwicklung des Strömungsmodells unter Nutzung von FiPy

Die Python-Bibliothek FiPy (Guyer, Wheeler & Warren, 2009) bietet die Möglichkeit, partielle Differentialgleichungen (PDGL) mit einer der mathematischen Beschreibung sehr nahen Syntax direkt in Python auszudrücken. Damit lassen sich numerische Lösungen für Probleme wie Grundwasserströmung und -transport schnell auf einem hohen Abstraktionsniveau umsetzen. FiPy nutzt sehr performante C++-Algorithmen im Hinterund. Der Nutzer programmiert aber in Python und hat mit C++ keinen direkten Kontakt. Python ist, nach einer kurzen Einarbeitungszeit, in aller Regel bedeutend leichter zu benutzen als C++. Python-Programme sind regelmäßig kürzer, besser lesbar und exponieren weniger interne Details, die zwar für schnelle Programme wichtig sein können, aber für den Anwendungsprogrammierer

nicht zur inhaltlichen Problemlösung beitragen. Als numerische Methode nutzt FiPy finite Volumen. Diese numerische Methode ist lokal massenbilanztreu und eignet sich deshalb sowohl für die Strömungs- als auch die Transport-Modellierung. Die Wärmetransportmodellierung entspricht mathematisch der Transport-Modellierung und ist deshalb auch mit FiPy abbildbar.

Die Darcy-Gleichung:

$$\mathbf{q} = -\mathbf{K}\nabla h \tag{3.5}$$

mit:

**q** Vektor der Darcy-Geschwindigkeit

**K** Tensor der hydraulischen Konduktivität

$\nabla h$ hydraulischer Gradient

führt, kombiniert mit der Kontinuitätsgleichung, zur Strömungsgleichung; hier in vereinfachter Form ohne Flüsse über die Modellgrenzen dargestellt:

$$SS\frac{\partial h}{\partial t} = \frac{\partial}{\partial x}(K_{xx}\frac{\partial h}{\partial x}) + \frac{\partial}{\partial y}(K_{yy}\frac{\partial h}{\partial y}) + \frac{\partial}{\partial z}(K_{zz}\frac{\partial h}{\partial z}) \tag{3.6}$$

Dies lässt sich mit FiPy direkt in Python-Code umsetzen:

```
def make_eq(self):
    """Create the FiPy equations.
    """
    self.gw_flow_eq = (
        fipy.TransientTerm(self.sp_store, var=self.head)
        ==
        fipy.DiffusionTerm(
            self.T.harmonicFaceValue, var=self.head))
```

Komplexere Formulierungen für ungespannte Verhältnisse und Strömungen über den Modellrand sind möglich. Das Code-Beispiel ist ein Ausschnitt aus einer Python-Klasse. Damit integriert sich die Lösung der PDGL nahtlos in die objekt-orientierte Programmierung. Ein Nutzer mit grundlegenden

Kenntnissen in den beiden Gebieten (1) PDGL für die Grundwasserströmung und (2) Python-Programmierung kann mit geringem Lernaufwand verstehen, welche Aufgabe der Code löst.

In diesem Projekt kam FiPy als Werkzeug zum Einsatz, um die Grundwasser-Strömungs-Modellierung numerisch umzusetzen. Die entwickelte *FiPy*-basierte Implementierung der gesättigten Strömungsimulation in porösen Medien für gespannte und ungespannte Verhältnisse erlaubt den direkten Eingriff des Modellierers auf die Rechnung zu Modelllaufzeit. Damit kann der Programmierer Randbedingungen dynamisch gestalten. So können sich Werte für Randbedingungen abhängig von anderen Werten im Modell ändern. Ein typisches Beispiel ist ein Brunnen, dessen Förderrate sich in Abhängigkeit von Wasserständen an bestimmen Stellen im Modell in bestimmten Grenzen ändert.

Die FiPy-basierte Implementierung der Grundwasserströmung erbrachte Ergebnisse, die nur geringfügig, bedingt durch die unterschiedlichen numeriche Ansätze, von den von MODFLOW 2005 abweichen. Verschiedene Testmodelle mit gleichen Diskretisierungen, gleichen hydrogeologischen Parametern und gleichen Randbedingungen in MODFLOW und FiPy erbrachten berechnete Wasserstände die im direkten Brunnenbereich maximal 4 cm voneinander abwichen. Die Abweichungen in anderen Bereichen war wesentlich geringer im Bereich von 1 bis 2 mm. Damit kann die FiPy-Implementierung grundlegende MODFLOW-basierte Grundwassermodelle reproduzieren. Abbildung 3.5 zeigt einen Vergleich der Ergebnisse einer Rechnung mit MF-2005 und FiPy.

### Vor- und Nachteile der FiPy-basierten Grundwassermodellierung

Die FiPy-Implementierung bietet diese Vorteile:

- vollständige Kontrolle über die Abbildung der partiellen Differentialgleichung
- damit Abbildung gesättiger oder ungesättiger Strömung möglich
- reine Python-Umsetzung (C++-Erweiterung durch FiPy-Team)
- einfacher Zugriff auf alle Modellzustände zu jedem Simulationszeitschritt

Dem stehen ein paar Nachteile gegenüber:

- eigene Umsetzung aller Randbedingungen für Grundwassermodelle

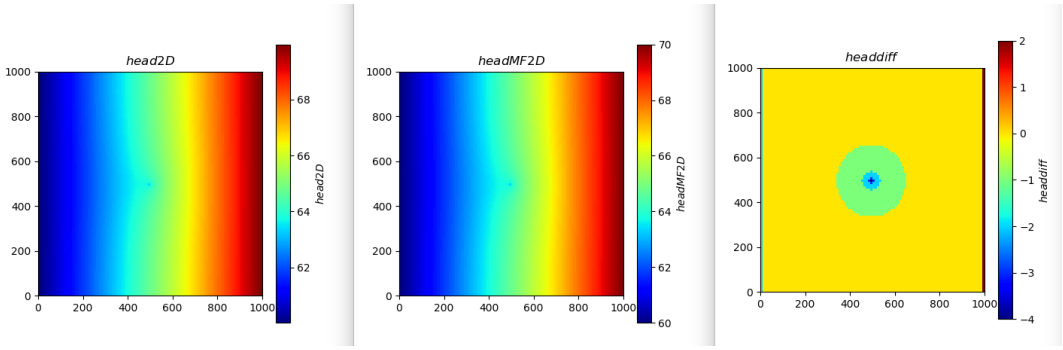



Abbildung 3.5.: Beispielhafte Verwendung von FiPy für ein vereinfachtes Szenario der numerischen Simulation von Grundwasserströmung für 2D Strömung mit zentralem Entnahmebrunnen (links: FiPy-Rechnung (mNN), Mitte: MODFLOW-Rechnung (mNN), Rechts: Abweichungen in cm, großflächig ca. 1 bis 2 mm, maximal 4 cm direkt am Brunnen bedingt durch Diskretisierung),

- eigene Umsetzung von anderen Berechnungsalgorithmen wie Bilanzierung oder Verhalten von atypischen Modellzellen bei Entwässerung einer Modellschicht
- keine externen Werkzeuge für die Erzeugung von Eingabedaten
- keine spezielle, externen Werkzeuge für die Auswertung von Ergebnissen

### 3.4.2. Strategieänderung zur Nutzung von MODFLOW 6

MODFLOW (USGS, 2020c) ist seit langer Zeit Stand der Technik. Die wissenschaftliche Community and viele Ingenieure weltweit nutzen MODFLOW für verschiedene Anwendungsfälle. Seit etwa Mitte 2018 ist die neueste Version von MODFLOW (MODFLOW 6, kurz MF6) (USGS, 2020b) offiziell verfügbar. Dieser Status bedeutet, dass die Software die hohen Qualitätsstandards des U.S. Geological Survey (USGS) an Funktionalität und ausführlicher Dokumentation erfüllt. Im Gegensatz zur offiziellen Vorgänger-Version MF-2005, welche nicht mehr weiterentwickelt und technisch unterstützt wird, wird MF6 stetig um neue Funktionalitäten ergänzt. MF6 beruht auf der finiten Volumen-Methode und erlaubt neben in MF-2005 genutzten strukturierten Modell-Grids auch unstrukturierte Grids in verschiedene Ausprägungen. Weiterhin erhält MF6 schrittweise neue Funktionalitäten, u.a. extrahiert aus anderen MODFLOW-Versionen mit anderen Gleichungslösern oder anderen interessanten Merkmalen. Es ist ist abzusehen, das MF6

in den nächsten Jahren die bisher leistungsfähigste MODFLOW-Version werden wird.

Ein wichtiger Aspekt ist die freie Verfügbarkeit der Quelltexte von MF6 für alle Anwendungsgebiete. Die Quelltexte sind als Public Domain sowohl für kommerzielle als auch für jede andere Nutzung ohne jegliche Einschränkung einsetzbar. Diese Quelltexte sind, wie bei vielen anderen Open-Source-Projekten, auf GitHub verfügbar (USGS, 2020a). Diese Plattform nutzt das moderne, dezentralisierte Versionsverwaltungssystem Git und bietet zahlreiche weitere Möglichkeiten der quelloffenen, verteilten, effizienten Entwicklung von Software. Dies vereinfacht auch den Prozess eigene Änderungsvorschläge in den Quelltext zu bringen. Mit einem sogenannten "Pull Request" (PR) kann auch ein Außenstehender, hier also ein GitHub-Nutzer ohne Zugriffsrechte auf das USGS-Repository, Quelltext-Änderungen einreichen. Nach einen Review-Prozess durch die Entwickler des USGS werden die Änderungen im PR entweder Bestandteil von MF6 oder es gibt Rückmeldungen wie die Änderung angepasst werden sollte, damit eine Übernahme möglich wird.

Bei der Präsentation des Zwischenstands von *ModSimple* auf der Konferenz *Modflow and More 2019* ergab sich ein intensiver Austausch mit den primären Entwicklern von MF6. Der Projektleiter Herr Dr. Müller führte intensive Gespräche mit Herrn Dr. Langevin vom USGS, dem Hauptentwickler von MF6. In diesen Gesprächen stelle sich heraus, dass die aktuelle Version von MF6 mit dem sogenannten *memory manager* zu jedem Zeitschritt vollen lesenden und schreibenden Zugriff auf alle Modelldaten ermöglicht. Eine vergleichbare Funktionalität bestand in früheren MF-Versionen nicht und war zum Zeitpunkt des Projektstarts nicht absehbar. Damit eröffnete sich ein Weg, die umfangreichen Möglichkeiten von MF6 mit vertretbarem Aufwand über eine Python-Schnittstelle verfügbar zu machen.

Das Projektteam und Mitglieder des wissenschaftlichen Beirates haben dieses Thema während eines Projekttreffens am 26. Juni 2019 intensiv beraten und beschlossen, die Strategie zu ändern. Statt FiPy sollte MF6 im *ueflow*-Modul groundwater für die Strömungssimulation zum Einsatz kommen. Da diese Strategieänderung zu mehr Aufwand führen würde, beschlossen die Teilnehmer der Beratung einen Antrag auf kostenneutrale Verlängerung zu stellen. Die Projektleitung hat diesen Antrag mit einer ausführlichen Begründung der Strategieänderung am 20. Juli 2029 an die DBU versandt. Die Bestätigung der Verlängerung und der Strategieänderung durch die DBU erfolgte am 8. August 2019.

Als Teil der Strategieänderung entstanden in der Projekt-Phase 1 neue Aufgaben. Deshalb sind zwei Arbeitspakete der Phase 1 in die Phase 2 übergegangen. MF6 unterstützt mittlerweile auch die Transportmodellierung. Damit kann die Bearbeitung der Arbeitspakete zum Thema Transport in Phase 2 schneller als ursprünglich geplant erfolgen. Der Gesamtumfang beider Projekt-Phasen ändert sich daher nicht.

### 3.4.3. Entwicklung des Strömungsmodells unter Nutzung von MODFLOW 6

MODFLOW 6 (MF6) ist in Fortran programmiert. MF6 enthält ein Modul namens *memory manager*, mit welchem (nahezu) alle Modellwerte zur Laufzeit zur Verfügung stehen. Die Variablen sind über Namen ansprechbar. Je nachdem, welche Randbedingungen in einem konkreten MF6-Lauf aktiv sind, so ändert sich die Liste der verfügbaren Namen für den Zugriff via *memory manager*.

In diesem Projekt ist ein Python-Programm *pymf6* entstanden, das den Fortran-Quelltext von MF6 als eine sogenannten Python-Erweiterung zur Verfügung stellt. Der englische Begriffe für diese Technik ist Wrapping, da Python den Fortran-Code "einwickelt". In *pymf6* ist der Fortran-Quelltext von MF6 in kompilierter Form enthalten. Der Nutzer arbeitet aber nur mit dem wesentlich leichter zu handhabenden Python-Modul. Über den so von Python aus zugänglichen *memory manager* kann ein Python-Programm zu jedem Modellzeitschritt alle MF6-Modellwerte auslesen und modifizieren. Damit ergibt sich volle Laufzeitkontrolle über MF6-Berechnungen.

Das Python-Programm *ueflow* nutzt *pymf6*, um die dynamische Rückkopplung von Randbedingungen effizient und benutzerfreundlich abzubilden. Dabei bietet *pymf6* den vollen Zugriff auf alle Details von MF6. Es ist deshalb nur für Programmierer mit entsprechender Erfahrung geeignet. Im Gegensatz dazu bietet *ueflow* Zugang mit einem höheren Abstraktionsniveau und ist deshalb auch für Grundwassermodellierer mit grundlegenden Python-Kenntnissen sinnvoll nutzbar.

#### Programmtechnische Umsetzung

Die nötigen Änderungen am MF6-Fortran-Quelltext für die Erstellung der Python-Erweiterung waren vom Umfang gering und technisch einfach. Die

technischen Herausforderungen liegen im Python-Wrapping-Code, der so gestaltet sein muss, dass Fortran- und Python-Code defacto gleichzeitig zur Laufzeit aktiv sind. Es handelte sich um programmtechnische Änderungen. Es gibt keine inhaltlichen Änderungen der Funktionalität im Fortran-Code. MF6 kann mit der gleichen Quelltext-Basis sowohl weiterhin als eigenständiges Programm, als auch als Python-Erweiterung laufen. Bedingt durch die Offenheit der Quelltexte war es möglich, die nötigen Änderungen in neue Versionen von MF6 direkt einzuarbeiten. Die meisten manuellen Schritte um den MF6-Quelltext für die Erweiterung vorzubereiten, lassen sich automatisieren. Damit ergibt sich keine unmittelbare Abhängigkeit von *ueflow* von eine künftigen Umgestaltung der MF6-Quelltexte durch den USGS.

Die programmtechnische Herausforderung besteht darin, den Fortran-Code so wenig wie möglich zu ändern, und trotzdem einen interaktiven Datenaustausch zu jedem Simulationszeitschritt zwischen Fortran und Python zu erreichen. Die Ursache ist der monolithische Aufbau des Fortran-Codes. Die gesamte Funktionalität im MF6-Fortran-Code lag in einem Hauptprogramm vor, das Unterprogramme aufruft. Diese Unterprogramme in Fortran heißen *subroutine*. Python kann mit dem Werkzeug *f2py* eine solche Subroutine als Python-Funktion zur Verfügung stellen. Nach dem in diesem Projekt durchgeführten Umbau des MF6-Fortran-Code liegt das Hauptprogramm in einer Subroutine vor. Diese lässt sich nun in Python einbauen. Das Hauptproblem liegt aber nun darin, dass nach dem Aufruf dieser Subroutine aus Python MF6 ohne Stopp bis zum Ende durchläuft. Ziel ist aber ein Datenaustausch zu jedem Simulationszeitschritt. Deshalb kam der Callback-Mechanismus zum Einsatz. Abbildung 3.6 zeigt das Prinzip.

Die Fortran-Subroutine *mf6_sub* akzeptiert das Argument *cback*. Dabei ist *cback* als externe Funktion mit *external cback* definiert. Diese Subroutine ruft nun *cback* vor Beginn eines neuen Simulationszeitschritts auf. Die Subroutine *mf6_sub* steht als Python-Erweiterung zur Verfügung. Python ruft also die Fortran-Subroutine auf. Dabei ist *cback* aber eine Python-Funktion, die nun zu jedem Simulationszeitschritt aktiv wird. Damit dies funktioniert, muss *mf6_sub* in einem Thread laufen, da Python sonst bis zum Ende der Abarbeitung von *mf6_sub* warten würde. Der Datenaustausch zwischen Fortran und Python erfolgt über ein Fortran *module*. Alle über den *memory manager* verfügbaren Variablen können nach Aufforderung ihre Werte dort ablegen und neue Werte von dort übernehmen. Python kann ebenfalls dort Daten abholen und ablegen.

Diese komplexen Vorgänge sind in *pymf6* verborgen. Der Nutzer muss diese

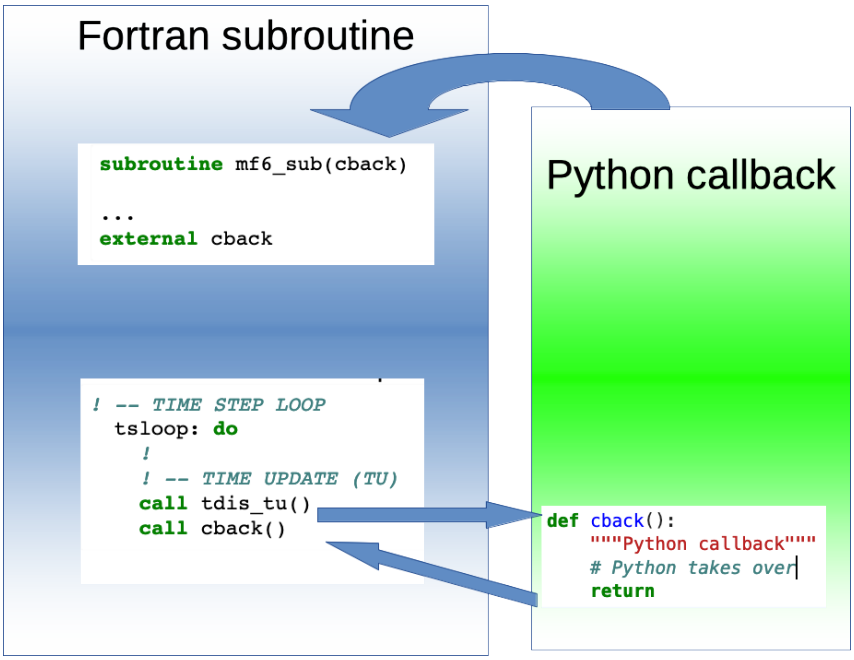



Abbildung 3.6.: Callback-Mechanismus für den interaktiven Datenaustausch zwischen Fortran und Python

Details nicht kennen. Wichtig ist die Funktion *cback*. In Python gibt es das Konzept eines aufrufbaren Objektes, das sich so wie eine Funktion verhält aber nicht zwingend eine Funktion sein muss. Eine Instanz einer Python-Klasse, die eine Methode `__call__` enthält ist ein solches rufbares Objekt. Es bietet die Möglichkeit komfortabel Zustandsinformationen zwischen Funktionsaufrufen zu speichern. Dies ist hier nötig, um in der Funktion Informationen über vorhergehende Zeitschritte zu haben. Folgendes Listing zeigt wie eine solche als Funktion fungierende Klasse aussehen kann:

```
class MyFunc(Func):
    """Class whose instances act like a function,
     i.e. are callables
    """

    def __init__(self):
        super().__init__()
        self.model = self.simulation.models[0]
        self.sim = self.simulation.solution_groups[0]
        self.chd_changed = False
```

In der Methode `__init__` sind alle Schritte zusammengefasst, die zu Beginn

des Modelllauf erfolgen sollen. Über die Speicherung in Instanz-Variablen `self.sim` oder `self.chd_changed` sind diese Informationen später verfügbar.

```
    def __call__(self):
        """
        Change 'CHD' for stress period 2.
        """
        super().__call__()
        # pylint: disable-msg=no-member
        if (not self.chd_changed and
            self.simulation.TDIS.KPER.value == 2):
            self.chd_changed = True
            self.model.CHD_0.BOUND[0][0][::2] = 2
            self.model.CHD_0.BOUND[0][0][1::2] = 5
```

Die Methode `__call__` ruft Fortran über den Callback-Mechanismus zu jedem Simulationszeitschritt auf. In diesem Beispiel setzt Python in der dritten Stressperiode, also wenn *KPER* gleich 2 ist (Python beginnt bei *0* mit dem Zählen) die Werte für die Randbedingungen 1. Art *CHD* am linken und rechten Modellrand auf 2 bzw. 5 m. Die Namen wie *KPER* und *CHD* kommen direkt aus MF6 und sind in der MF6-Programm-Dokumentation ausführlich beschrieben.

Neben der Steuerung von MF6 über das Schreiben einer Callback-Funktion gibt es die Möglichkeit *‚pymf6* interaktiv zu nutzen. Abbildung 3.7 zeigt eine typische Sitzung, in der der Nutzer interaktiv in einem Jupyter-Notebook arbeiten und die MF6-Variablen direkt inspizieren und deren Werte ändern kann.

Dieser Ansatz erlaubt es auch immer einen Simulationszeitschritt ablaufen zu lassen und dann wieder auf MF6-Variablen zuzugreifen. Diese Methode ist sehr gut geeignet, um sich mit den für den aktuellen Modelllauf zur Verfügung stehenden Variablen vertraut zu machen. Ein Nutzer kann damit komfortabel heraus finden welche Variablen dann in der Callback-Funktion zum Einsatz kommen sollen.

### Integration MF6 via *pymf6* in *ueflow*

Abbildung 3.8 zeigt den Aufbau von *ueflow*. Die Grundstruktur orientiert sich am typischen Ablauf einer Modellierung mit Datenaufbereitung *pre*, Rechnung *run* und Ergebnisauswertung *post*.

```
[2]: from pymf6.threaded import MF6
```

make an instance of this class:

```
[3]: mf6 = MF6()
```

## Meta Data

This instance offers meta data such as the names of the models:

```
[4]: mf6.simulation.model_names
[4]: ['GWF_1']
```

or the time unit:

```
[5]: mf6.simulation.time_unit
[5]: 'DAYS'
```

as well as the associated time multiplier to convert to seconds:

```
[6]: mf6.simulation.time_multiplier
[6]: 86400
```

## Temporal Discretization (TDIS)

Data about stress periods and time steps are also available. Get the number of stress periods:

```
[7]: mf6.simulation.TDIS.NPER
[7]:
```

### Variable NPER

| | |
|---|---|
| value: | 4 |
| location: | TDIS |

or the total simulation time (in days in this case):

```
[8]: mf6.simulation.TDIS.TOTALSIMTIME
[8]:
```

### Variable TOTALSIMTIME

| | |
|---|---|
| value: | 31.0 |
| location: | TDIS |

```
[9]: mf6.simulation.TDIS.var_names
[9]: ['NPER',
      'ITMUNI',
      'KPER',
      'KSTP',
      'READNEWDATA',
      'ENDOFPERIOD',
      'ENDOFSIMULATION',
      'DELT',
      'PERTIM',
      'TOTIM',
      'TOTIMC',
      'DELTSAV',
      'TOTIMSAV',
```

Abbildung 3.7.: Interaktive Nutzung von *pymf6* via Jupyter

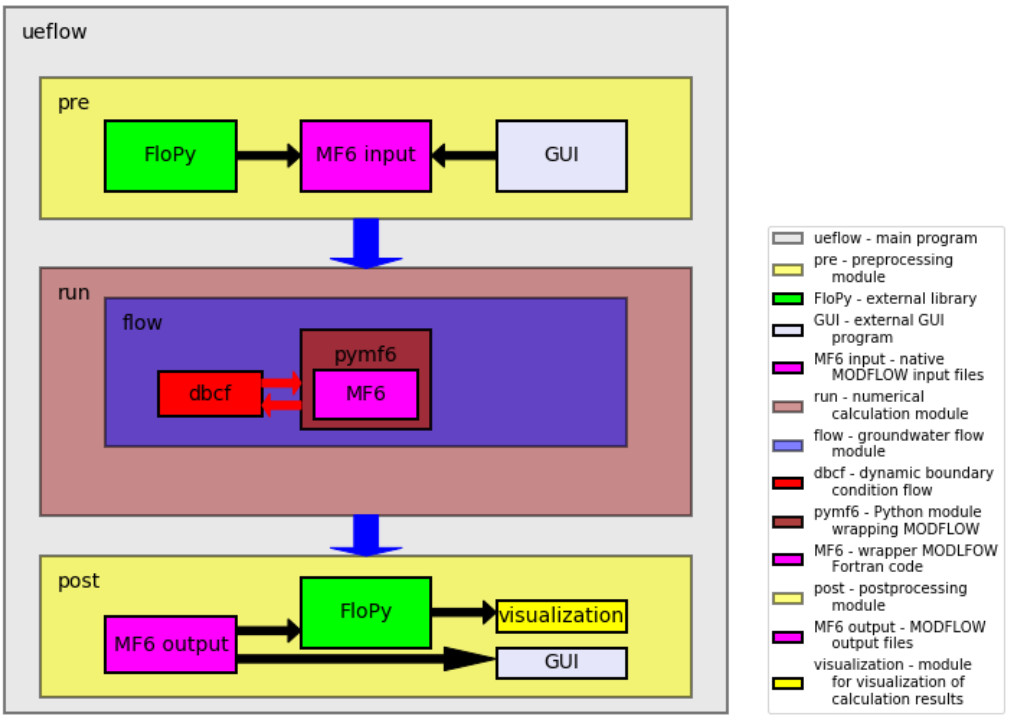



Abbildung 3.8.: Aufbau von ueflow

Für die Datenaufbereitung und die Ergebnisauswertung kommen existierende Werkzeuge wie die Python-Bibliothek FloPy (Bakker et al., 2016) und das grafische ModelMuse (Winston, 2020) zum Einsatz. Diese Werzeuge lassen sich leicht an die Bedürfnisse von *ueflow* anpassen. Insbesondere FloPy bietet einen hohen Grad an Automatisierung und damit Wiederverendbarkeit.

Im Haupteil von *ueflow* steht das Modul für die Rechnung *run*. Den Fortran-Kern *MF6* macht die Python-Erweiterung *pymf6* verfügbar. Das Modul *dbcf* enthält den vom Modellnutzer geschriebenen Code mit der Callback-Funktion. Das Modul *flow* setzt die eigentliche Grundwasserströmungsmodellierung um. In Projekt-Phase 2 wird parallel zu *flow* das Modul *transport* hinzukommen.

### Vor- und Nachteile der MF6-basierten Grundwassermodellierung

Durch die weite Verbreitung von MF6 gibt es viele Werkzeuge aller Art, die sofort auch für die Arbeit mit *ueflow* nutzbar sind. Konkret bietet die MF6-Implementierung diese Vorteile:

- alle MF6 Möglichkeiten für die Modellierung verfügbar
- externe Werkzeuge für die Datenaufbereitung verfügbar
- externe Werkzeuge für die Ergebnisauswertung verfügbar
- neue MF6-Merkmale schnell verfügbar

Anderseits hat die MF6-Implementierung auch Nachteile:

- kein direkter Einfluss auf die Erstellung und Lösung der partiellen Differentialgleichungen
- Erstellung einer neuen Python-Erweiterung für jede neue MF6-Version

Diese Nachteile sind jedoch nicht erheblich. Die Implementierung der partiellen Differentialgleichungen in MF6 ist für dieses Projekt ausreichend. Das Erstellung einer neuen Python-Erweiterung lässt sich weitestgehend automatisieren.

### Weiternutzung der FiPy-basierten Programmteile

Die zu Beginn des Projektes entwickelten FiPy-basierten Programmteile lassen sich für andere Zwecke weiter nutzen. Zum Beispiel können sie Endbenutzer-definierte Randbedingungen und Prozesse wie ungesättigte

Strömung in der Bodenzone oder Oberflächenwasserströmungen umsetzen.

Andere, bereits entwickelte *ueflow*-Komponenten lassen sich mit dem neuen Modul verbinden.

## 3.5. AP II/2 - Entwicklung der Grundwasserströmungssoftware - Verifizierung des Modellalgorithmus mit Szenarien aus Literatur

Aufgrund der Strategieänderung (siehe Kapitel II.1) wurde MODFLOW in der Version 6 (MF6) mit dem über Python ansteuerbaren *memory manager* verwendet. Ein Benchmarking der MF6-Software selbst ist aufgrund der intensiven Testung von MF6 durch die Entwickler nicht erforderlich; jedoch erfordert der Speicher-Zugriff des *pymf6*-Moduls auf den *memory manager* eine erneute Testung der grundlegenden Funktionalität. Dies hatte den Zweck, die Wahrscheinlichkeit unbeabsichtigter Eingriffe in den reinen Simulationsablauf (d.h. die eigentliche numerische Lösung der Gleichungssysteme) zu minimieren.

Zur besseren Vergleichbarkeit wurden die Benchmarks in drei Hauptkategorien mit jeweils unterschiedlicher Komplexität des Modells untergliedert:

- A: unbeeinflusster Grundwasserleiter
- B: Zwei-Brunnen-Grundwasserströmungsmodell
- C: Kanalnetzleckage (vereinfachte Darstellung in Form einer Linienquelle)

Modelltyp A stellt einen Grundwasserleiter ohne anthropogene Beeinflussung dar. Der Aquifer ist in zwei gespannte Teilgrundwasserleiter untergliedert. Einflussfaktoren sind lediglich ein hydraulischer Gradient von West nach Ost (definiert durch Randbedingungen 1. Art) sowie flächenhaft homogene Grundwasserneubildung. Über drei Zeitperioden (1. stationär, 2. und 3. instationär) werden Grundwasserstände simuliert. Dieses Beispiel dient als Basisszenario für alle weiteren Modelltypen.

Modelltyp B umfasst Modelltyp A und besitzt zudem zwei Brunnen. Der westliche Brunnen dient als Entnahmebrunnen im unteren Grundwasserleiter, wohingegen der östliche Brunnen als Injektionsbrunnen im unteren Grundwasserleiter dient. Dieses Beispiel dient als typischer Anwendungsfall für eine ökonomische Pumpensteuerung.

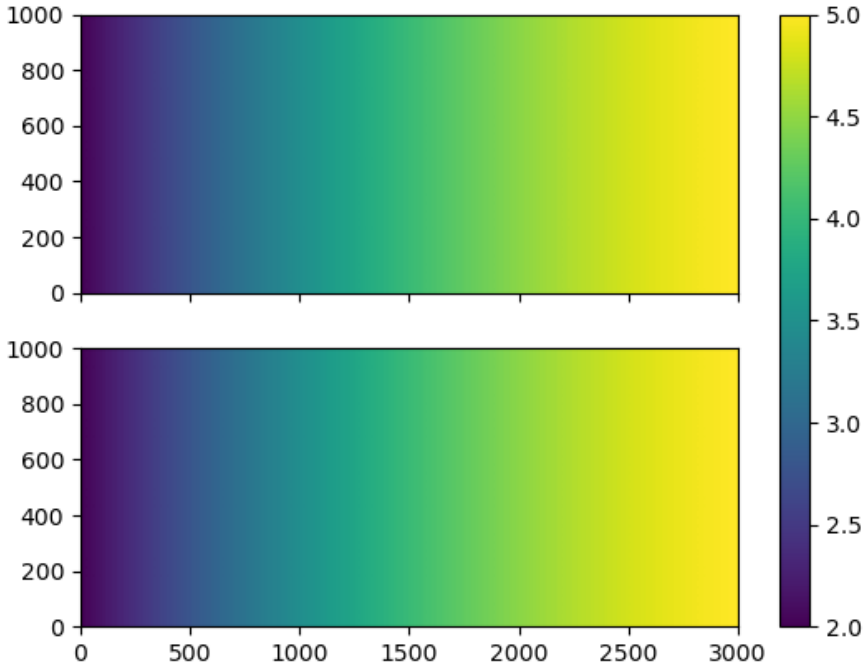



Abbildung 3.9.: Szenario A - Aufbau mit Endwasserstand in [m] in beiden Schichten

Modelltyp C umfasst Modelltyp A und besitzt zusätzlich einen Einzelstrang eines öffentlichen Kanalisationsnetzes. Dieses Beispiel dient als typischer Anwendungsfall für urban beeinflusste Grundwassersysteme. Unter anderem können hiermit hydraulische Einflüsse (z.B. künstliche Grundwasseranreicherung durch Kanalnetzleckagen) auf den Grundwasserleiter untersucht werden.

Um eine Basis für die Benchmark-Szenarien von *ueflow* zu schaffen, wurde jeder der oben genannten Modelltypen zunächst in der grafischen Nutzeroberfläche ModelMuse (Winston, 2020) erstellt und mögliche Parameterspannweiten erörtert. Die Spannweiten sollten dabei in einem möglichst realistischen Rahmen für die zu untersuchende lokale urbane Skala liegen. Mögliche bis dahin existierende Restriktionen von MF6 (z.B. keine ungespannten Layer) wurden berücksichtigt. Die räumliche Diskretisierung wurde so gewählt, dass angesichts einer Vielzahl von Realisationen ein Kompromiss aus Rechengenauigkeit und Rechenzeit erzielt wurde.

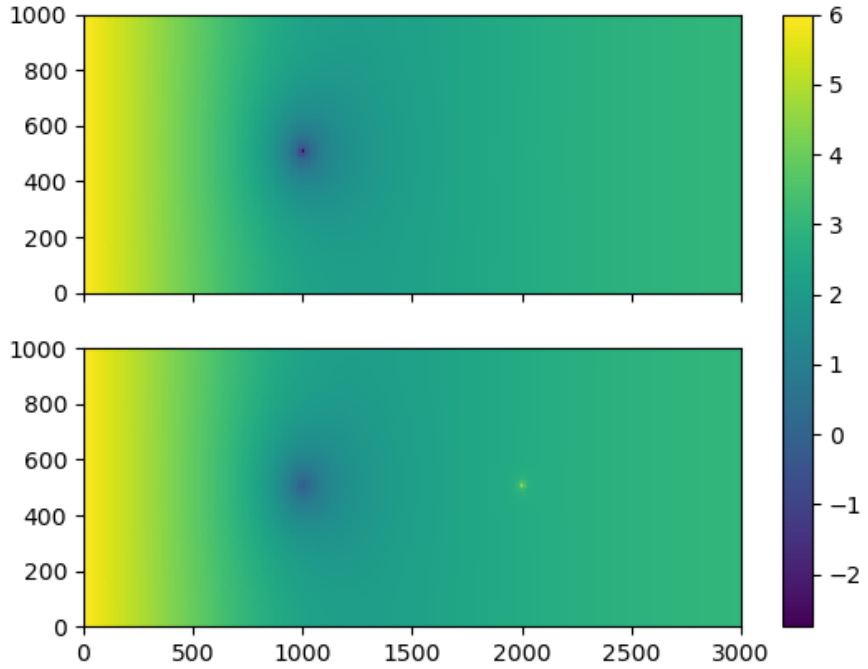



Abbildung 3.10.: Szenario B - Aufbau mit Endwasserstand in [m] in beiden Schichten

Nachdem sinnvolle Parameterspannweiten gefunden wurden (siehe Tabelle B.2 im Anhang auf Seite 66), wurde eine Unterteilung der jeweiligen Parameterspannweiten in eine Vielzahl an Parametervariationen vorgenommen. Eine Realisierung war demnach durch einen Vorwärtslauf des MF6- und *ueflow*-Modells unter Variation desselben Einzelparameters definiert.

Hierbei wurden vorerst jeweils Modelle mit gleichen Eingabedaten für die *MF6*-Modelle und *pymf6*-Modellen erstellt. Bei den *pymf6*-Modellen wurden dann die Randbedingungswerte in den Eingabedateien absichtlich geändert. Ziel war es, diese Werte während der Laufzeit mit Hilfe der dynamischen *pymf6*-Mechanismen zu ändern, um die gleichen Ergebnisse wie mit den *MF6*-Modell zu berechnen.

Es wurden dann Läufe:

- I: ohne dynamische Rückkopplung, d.h. absichtlich keine Funktionalität in *pymf6* genutzt und

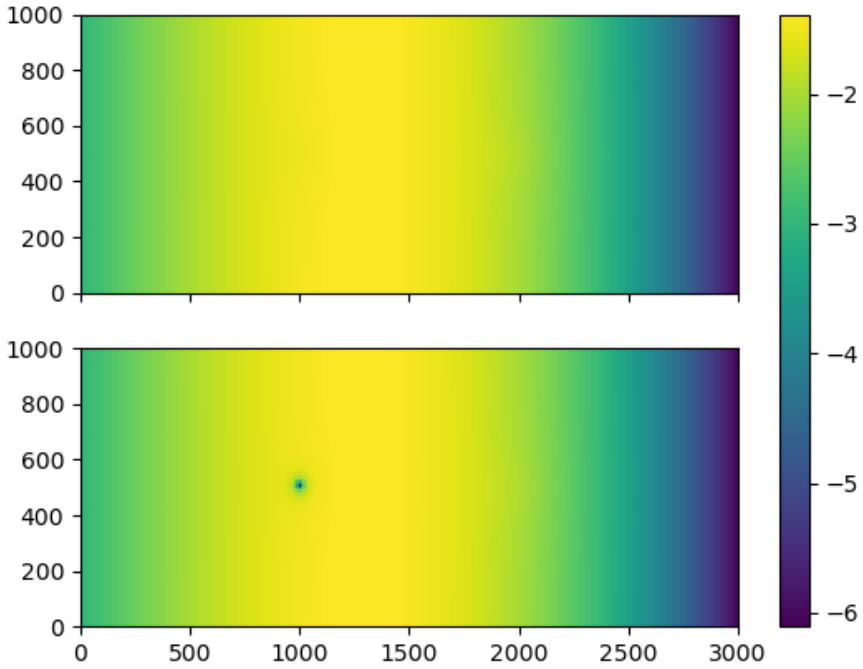



Abbildung 3.11.: Szenario C - Aufbau mit Endwasserstand in [m] in beiden Schichten

- II: Modelle mit dynamischer Kopplung per *pymf6*, die die vorgegebenen Randbedingungswerte in *MF6* reproduzieren sollen

durchgeführt.

Für Typ I ergaben sich folgerichtig großer Unterschiede in den Ergebnissen, da die Randbedingungen zwischen den verglichenen Modellen anders sind. Dieses Ergebnis zeigt, dass ohne die Nutzung der dynamischen Mechanismen von *pymf6* de-facto zwei unterschiedliche Modelle zum Einsatz kommen.

Bei Typ II nutzt die gleichen Eingabe-Dateien wie Typ1. Im Unterschied zu Typ 1, greift *pymf6* zur Laufzeit ein und ändert die Randbedingungswerte dynamisch. Damit müsste für den *pymf6*-Lauf das gleiche Ergebnis wie für den *MF6*-Lauf entstehen. Trotz unterschiedlicher Datei-Eingabe-Daten entsteht das gleiche Ergebnis, da *pymf6* die Modell-Werte dynamisch entsprechend ändert, also letztlich korrigiert.

Entsprechend der Parameter-Spannweiten wurden jeweils zahlreiche Realisationen gezielt und automatisiert durchlaufen, und ausgewertet. Für jedes Modellsetup wurden die piezometrischen Höhen jeder Gitterzelle für den letzten Zeitschritt der letzten Zeitperioden ausgelesen und miteinander verglichen. Zusätzlich wurden die kumulativen Volumenbilanzen (gesamt, Einzelkomponenten von Randbedingungen und Speichern) beider Modellsetups miteinander verglichen.

Dazu wurde der root-mean-squared error (RMSE) berechnet:

$$RMSE = \sqrt{\frac{1}{n}\Sigma_{i=1}^{n}\left(d_i - f_i\right)^2}$$

berechnet.

Hierbei entspricht $n$ der Stichprobenzahl (= Anzahl der Gitterzellen). $d_i$ minus $f_i$ entspricht dem Residuum zischen den simulierten Wasserständen in *ueflow* und *MF6*. Bei den Volumenbilanzen entspricht $n$ der Anzahl der verfügbaren Werte, die je nach Art des Speichers oder der Randbedingung sehr unterschiedlich sein kann.

Um eine andere Wichtung der Abweichungen zu berücksichtigen, wurde zudem der mean-squared error (MSE) verwendet:

$$MSE = \frac{1}{n}\Sigma_{i=1}^{n}\left(d_i - f_i\right)^2$$

### Szenarienvergleich

Für verschiedenste Parameter-Kombinationen ergab sich dieses kumulative Ergebnis für Szenario A:

```
A
{'MSE': {'max': 0.0, 'mean': 0.0, 'min': 0.0, 'n': 210, 'std': 0.0},
 'RMSE': {'max': 0.0, 'mean': 0.0, 'min': 0.0, 'n': 210, 'std': 0.0}}

n - Stichprobengröße = Anzahl der Modellläufe * verglichener Werte
    (Piezometerhöhe und kumulativen Volumenbilanzen)
```

für Szenario B:

```
B
{'MSE': {'max': 0.0, 'mean': 0.0, 'min': 0.0, 'n': 24, 'std': 0.0},
 'RMSE': {'max': 0.0, 'mean': 0.0, 'min': 0.0, 'n': 24, 'std': 0.0}}

n - Stichprobengröße = Anzahl der Modellläufe * verglichener Werte
    (Piezometerhöhe und kumulativen Volumenbilanzen)
```

für Szenario C:

```
C
{'MSE': {'max': 0.0, 'mean': 0.0, 'min': 0.0, 'n': 27, 'std': 0.0},
 'RMSE': {'max': 0.0, 'mean': 0.0, 'min': 0.0, 'n': 27, 'std': 0.0}}

n - Stichprobengröße = Anzahl der Modellläufe * verglichener Werte
    (Piezometerhöhe und kumulativen Volumenbilanzen)
```

Es gibt also keinerlei Abweichungen zwischen den *pymf6*- und *MF6*-Läufen. Dies ist zu erwarten, wenn die gleichen Eingabedaten mit dem selben numerischen Algorithmus verarbeiten werden. Der numerische Algorithmus ist per Definition gleich. Die gleichen Eingabedaten werden aber von *pymf6* erst zur Laufzeit generiert. Wichtig ist, dass in *pymf6* die Werte auf die gleiche Anzahl an Nachkommastellen gerundet werden, welche die Eingabe-Dateien zur Verfügung stellen. Dies ist insbesondere von Bedeutung, wenn Einheiten umzuwandeln sind und damit viele Nachkommastellen entstehen können (z.B. infolge Division). Das Schreiben dieser Werte mit FloPy in MF6-Eingabe-Dateien ist auf weniger Nachkommastellen beschränkt, als das in *pymf6* standardmäßig verwendete Zahlenformat *float* abbilden kann. Ohne Runden kann es durchaus zu geringfügigen numerischen Abweichungen kommen. Diese sind aber bedingt durch minimale Unterschiede in den genutzten Eingabe-Werten. Die Rechnungen zeigen, dass *pymf6* funktioniert und bei gezielter Nutzung keine Abweichungen in den Modellierungsergebnissen erzeugt.

## 3.6. AP III/1 - Entwicklung der Transportsoftware mit Schnittstelle zu analytische Lösungen und Randbedingungen - Programmierung des Transportsimulationsalgorithmus mit Python

Aufgrund der Strategieänderung (siehe Kapitel 3.4) während des Projektverlaufes wurden die ueflow-Module für Stofftransport (solutes) und Wärmetransport (heat) nicht im Projektzeitraum der Phase 1 entwickelt. Diese Teilaufgaben werden im Folgeprojekt (*ModSimple - Phase2*) angegangen. Eine Python-basierte Kopplung mit dem Transportmodul von MF6 wird ebenfalls im Folgeprojekt in Analogie zur Kopplung des Strömungsmodells (*pymf6*) realisiert werden.

## 3.7. AP III/2 - Entwicklung der Transportsoftware mit Schnittstelle zu analytische Lösungen und Randbedingungen - Verifizierung des Modellalgorithmus mit Szenarien aus der Literatur

Aufgrund der Strategieänderung (siehe Kapitel 3.4 und 3.6) während des Projektverlaufes wurden die ueflow-Module für Stofftransport (solutes) und Wärmetransport (heat) nicht im Projektzeitraum der Phase 1 entwickelt; damit verbunden konnte auch die Verifizierung bisher nicht erfolgen. Diese Teilaufgaben werden im Folgeprojekt (*ModSimple 2*) angegangen.

## 3.8. Umweltentlastungspotenziale

### 3.8.1. Bisheriger Stand von Wissenschaft und Technik (Praxis)

#### Grundaufgabe "Simulation der Grundwasserströmung“

Zur Simulation der Wasser-gesättigten Grundwasserströmung sowie der Wasser-teilgesättigten Strömung in der vadosen Zone stehen eine Reihe von Simulationswerkzeugen zur Verfügung. In Ingenieur- und Beratungsunternehmen mit allgemeiner Ausrichtung sind verschiedene Werkzeuge im Einsatz. Für die gesättigte Strömung sind vor allem verschiedene Versionen der MODFLOW-Reihe (USGS, 2020c) sehr verbreitet. Deshalb wird das vom USGS kostenfrei und als Open-Source-Software zur Verfügung gestellte MODFLOW oft als de-facto Industriestandard bezeichnet. Für

die praktische Nutzung kommen normalerweise grafische Benutzeroberflächen zum Einsatz. Neben kommerziell vertriebenen Oberflächen wie Visual MODFLOW (Waterloo Hydrogeologic, 2020) , GMS (Aquaveo, 2020), Groundwater Vistas (Rumbaugh & Rumbaugh, 2020) und PMWIN (Chiang & Kinzelbach, 2003) gibt es das ebenfalls vom USGS kostenfrei angebotene ModelMuse (Winston, 2020). Für die Transport-Modellierung haben sich MODFLOW-Erweiterungsmodule wie MT3D (Bedekar et al., 2016) und PHT3D (Prommer & Post, 2010) etabliert. Weitere genutzte Werkzeuge sind die Software FEFLOW (Diersch, 2014) zur Abbildung der gesättigten und ungesättigten Strömung, die Software HYDRUS (Šimůnek, Šejna und van Genuchten, 2018; Šimůnek, van Genuchten und Šejna, 2016) für die ungesättigte Strömung, PCGEOFIM (Sames & Blankenburg, 2019) für den Einsatz im Braunkohletage und OpenGeoSys (Kolditz et al., 2012) mit Fokus auf Grundlagenforschung.

Diese Simulationswerkzeuge ermöglichen die Abdeckung einer Vielzahl von Aufgaben unter Verwendung vorgefertigter RB-Module. Dazu gehört die Abbildung spezieller RB und Quell-/Senkenterme. So erlaubt beispielsweise MODFLOW mit dem Modul "RIV" die Abbildung von Fließgewässern inklusive der Kolmation, also der teilweisen Abdichtung der Flusssohle.

Für viele Problemstellungen reichen diese Möglichkeiten der Abbildung von Prozessen im Modell nicht aus. Dies ist insbesondere der Fall, wenn Rückkopplungen eine Rolle spielen. Dabei müssen sich Werte für Randbedingungen in Abhängigkeit von im Modell berechneten Werten wie Grundwasserständen oder Volumenströme ändern. Oft ist der Modell-Nutzer gezwungen, ein Modell mehrfach laufen zu lassen. Nach jedem Lauf kann er dann Modellergebnisse nutzen, um Werte für Randbedingungen für den nächsten Lauf anzupassen. Dieser iterative Methode ist allerdings sehr arbeits- und rechenzeitaufwendig. Aus diesem Grund eignet sich dieser Ansatz nur für einfache Anwendungen mit relativ wenigen Iterationen.

Bisher war es nur möglich, mit dem Interface-Manager der kommerziellen, Closed-Source-Software FEFLOW Benutzer-spezifische Algorithmen einzubinden. Damit kann der Nutzer FEFLOW in Grenzen erweitern. Allerdings verursacht die Anschaffung von FEFLOW signifikante Lizenzkosten. Weiterhin erfordert die Nutzung des Interface-Managers vertiefte Programmierkenntnisse, da mit der Programmiersprache C DLL-Bibliotheken zu erstellen sind. Dies setzt eine entsprechende Compiler-Installation und -Nutzung voraus. Damit verkompliziert sich dessen Einsatz und macht es für bestimmte Projekte de-facto unzugänglich.

### Spezialaufgabe "Planung von geothermischen Anlagen"

Bisher ist es seitens Planern von geothermischen Anlagen übliche Praxis, Tabellenwerte oder vereinfachte mathematische Ansätze zur Auslegung der Technik wie Erdwäremesonden (EWS) und zur Abschätzung der Betriebssicherheit bei vorliegenden hydrogeologischen Bedingungen zu verwenden. Derartige Ansätze stellen zumeist analytische, semi-analytische oder empirische Modelle dar. Sie können kleinere Geothermieanlagen wie beispielsweise einzelne EWS oder kleine Sondenfelder mit symmetrischen Aufbau abbilden. Beispiele für Software-Werkzeuge für diese Aufgaben sind Earth Energy Designer (EED, (Hellström & Sanner, 2000)), TRNSYS (Klein et al., 2007) und Geo-Hand-Light (Koenigsdorff & Veser, 2008).

Aus dem Widerspruch zwischen den vorgenannten Ansätzen und der Komplexität des Untergrundes ergeben sich unter anderem die folgenden Problematiken, insbesondere bei Zunahme der Anlagenkomplexität:

- Einfache Abschätzformeln zur Planung von geothermischen Anlagen, die z.B. EWS mit fester Entfernung bis zur Grundstückgrenze abbilden, erlauben nur EWS-Felder mit einfacher Geometrie und können keine anderen hydraulische und/oder geothermische Nutzung desselben hydrogeologischen Untergrundes berücksichtigen. Eine modellbasierte numerische Beschreibung der Interaktion von mehreren geothermischen Anlagen im Gesamtsystem erscheint daher notwendig. Bei Anlagen mit Heiz-/Kühlleistungen über 30 KW ist eine exakte, i.d.R. numerische Abbildung vorgeschrieben.
- Der zeitlich-räumlich veränderliche Einfluss von Änderungen des gesamtheitlichen Systems ist nicht in analytischen Ansätzen abbildbar. Deshalb können sie z.B. saisonale Effekte mit Änderung des Wirkbereich der EWS nicht berücksichtigen. Unter dem zunehmenden Einfluss künftiger Nutzungsänderungen wie fortschreitende Urbanisierung und Landnutzungsänderungen sowie Klimawandeleffekte auf regionaler Skala sind jedoch genauere Prognosen notwendig, um die Planung und den Betrieb von geothermischen Anlagen effizient und nachhaltig zu gestalten.

Darüber hinaus ergeben sich durch Vorgaben seitens des Gesetzgebers und verschiedener technischer Normen diverse Restriktionen für den Betrieb von geothermischen Anlagen und der Eingabe von Wärme in den Untergrund im Allgemeinen. Wichtige Normen sind u.a. die VDI 4640 (z.B. Blatt 2:

Thermische Nutzung des Untergrunds - Erdgekoppelte Wärmepumpenanlagen) in Deutschland und die SIA 384/6 (Erdwärmesonden) in der Schweiz. Beispielhaft seien drei typische Anforderungen genannt, die eingehalten werden müssen:

- Eine Einleitung von Wasser mit einer Temperatur von mehr als 20 °C darf nicht dauerhaft erfolgen. Nur kurzfristig und in Ausnahmefall darf die Einleitwassertemperatur bis zu 25 °C betragen.
- Die gegenseitige Beeinflussung mehrerer Erdwärmesonden sollte möglichst gering sein und im Bereich von 1 K liegen. Zur Vermeidung der gegenseitigen Beeinflussung von geothermischen Anlagen gilt meist ein horizontaler Mindestabstand von 10 m.
- Bei falscher EWS-Dimensionierung bezüglich der Entnahmeleistung kann es zu lokalen Gefrierschäden durch zu starke Abkühlung des Wärmeträgerfluides kommen. Dies kann unter Umständen zur Instabilität des Untergrundes beitragen. Laut der schweizerischen Norm darf hier eine maximale Änderung der Untergrundtemperatur um 3 bis 4 K nicht überschritten werden.

Derlei technische Restriktionen können vorgenannte vereinfachte Ansätzen nur für einfache Geometrien in engen Grenzen berücksichtigten. Eine Abbildung von komplexen geothermischen Anlagen in numerischen Modellen ist zwar möglich, aber mit teils hohem Aufwand hinsichtlich der Parameterisierung und Rechenzeiten verbunden.

Aus diesem Grund bietet sich der neuartige Ansatz von ModSimple an, analytische Ansätze zur lokalen Abbildung von Randbedingungen und Quell-/Senkentermen mit numerischen Modellen dynamisch zu koppeln.

### 3.8.2. Beispielszenarien Phase 1

Nach ModSimple "Phase 1" ermöglicht *ueflow* bereits jetzt, Modellierungsaufgaben besser zu lösen und damit die Umwelt indirekt zu entlasten. Das Potential für weitere Verbesserungen hinsichtlich Funktionalität schätzen wir als groß ein. Dazu gehören die Erweiterung von *ueflow*-Modul *transport* zur numerischen Simulation von Stoff-/Wärmetransport und die damit verbundene dynamische Kopplung von Transport-Randbedingungen und -Quell-/Senkentermen mittels *pymf6*). Dies ermöglicht die prozessgenaue Abbildung von realistischen Anwendungsfällen von Geothermie in komplexen Untergrundverhältnissen. Wichtig dabei ist die Berücksichtigung

der beschriebenen technischen Restriktionen. Auch die Wirtschaftlichkeit hinsichtlich Recheneffizienz und damit besserer Ressourcengebrauch, Endnutzerfreundlichkeit und die Möglichkeit Open-Source-Software stetig weiterzuentwickeln spielen ein wichtige Rolle. Einige ausgewählte, konkrete Umsetzungen sind in den nächsten Abschnitten dargestellt.

### Methodische Verbesserungen

Das in Phase 1 entstandene Simulationssoftware *ueflow* bietet einige methodische Verbesserungen bei der Strömungsmodellierung. Diese Verbesserungen erlauben eine bessere Abbildung der Wirklichkeit mit Modellen. Dadurch ergeben sich Umweltentlastungspotentiale.

- **Automatisierte Arbeitsabläufe** Bei vielen Aufgaben muss der Anwender ein Modell mehrfach laufen lassen, die Modellergebnisse auswerten und darauf basierend die Eingabe entsprechend anpassen. Mit *ueflow* kann der Modellierer viele dieser Arbeitsabläufe ganz oder teilweise automatisieren. Die dynamischen Randbedingungen in *ueflow* können oft die gleichen Effekte in einem Modelllauf erreichen, die der Modellierer vorher durch manuelle Anpassung von Eingabewerten erzeugt hat. Damit kann der Arbeitsablauf wesentlich schneller werden. Als Folge wird es ermöglicht, effizient verschiedene Varianten zu untersuchen, um eine Lösung zu finden, welche die Umwelt höchstmöglich entlastet. Desweiteren wird eine Wiederverwendung von automatisierten Arbeitsabläufen für ähnliche Problemstellungen (jedoch mitunter andere Projekte) möglich.
- **Feinere zeitliche Auflösung der Interaktionen** Eine manuelle, schrittweise Anpassung von Randbedingungen und Quell-/Senkentermen führt meist zu eher großen Iterations-Intervallen zwischen den Anpassungen, um den Arbeitsaufwand in vertretbaren Grenzen zu halten. Mit dem dynamischen *ueflow*-Ansatz kann die zeitliche Auflösung der Steuerung der Randbedingungen beliebig fein sein, ohne dabei den Aufwand für den Modellierer zu erhöhen. Dies kann zu besseren Lösungen mit geringerer Umweltbelastung führen.
- **Inkorporation analytischer Lösungen** (Semi-)Analytische und empirische Lösungen lassen sich mit *ueflow* einfacher in das numerische Modell einbauen. Damit wird es möglich, komplexe numerische Abbildungen von Prozessen durch rechentechnisch wesentlich schnellere

analytische Lösungen zu ersetzen, die für den Modellzweck ausreichend genaue Ergebnisse liefern. Dies kann, je nach Fall, zu um Größenordnungen reduzierten Rechenzeiten führen. Wenn Rechenzeiten damit z.B. von Tagen (oder auch Wochen) auf Stunden sinken, werden manche Modelle erst praktikabel nutzbar. Derartige Anwendungsfälle können Untersuchungen auf regionaler Skala sein (z.B. ganze Stadtteile). Die exaktere Definition von Modellgrenzen durch Nutzung größerskaliger Modelle kann wiederum zur verbesserten Abbildungen der tatsächlichen Verhältnisse im Modell führen, und damit für die Umwelt verträglichere Lösungen zu identifizieren.

- **Partielle Verringerung der Dimensionalität** Die Abbildung von Strömungsprozessen in wassergesättigten porösen Medien erfolgt oft dreidimensional. Wenn für Teile des Modells andere Einflussverhältnisse wie z.B. Strömung in teilgesättigten Böden eine Rolle spielen, müsste das Modell diese auch dreidimensional abbilden. Das kann zu äußerst langen, nicht praktikablen Rechenzeiten führen. Mit *ueflow* ist es möglich einen Modellteilbereich mit einem zweidimensionalen oder mehreren eindimensionalen Modell(en) abzubilden. Für viele Anwendungsfälle ist die Genauigkeit für die Aufgabenstellung durchaus ausreichend. Im Ergebnis lassen sich modellbasierte, umweltrelevante Aussagen treffen, die sonst nicht oder nur mit wesentlich höherem Aufwand bezüglich Rechenperformance und Parametrisierung möglich wären.

### Anwendungsszenario: Vadose-Zone im Grundwassermodell

Für bestimmte hydrogeologische Fragestellungen ist auch die Beachtung der teilgesättigten Strömung in der vadosen Zone wichtig. Da die gesättigte Strömung im Grundwasser ein vereinfacht zu berechnender Grenzfall der teilgesättigten Strömung ist, könnte ein dreidimensionales Modell für die teilgesättigte Strömung beide Prozesse abbilden. Dieser Ansatz benötigt aber für reale Modellgrößen auf regionaler Skala exorbitante Rechenzeiten. Außerdem sind die numerischen Algorithmen recht anfällig für nicht-konvergierende Modellläufe. Nicht zuletzt sind die Unsicherheiten bezüglich der Modellparametrisierung aufgrund der Nichtlinearität der teilgesättigten Strömung ungleich größer als für Wasserströmung im vollgesättigten Bereich. Daher werden derartige Modelle auf großen Skalen nur recht selten angewandt und benötigen eine sehr gute Ausgangsbasis hinsichtlich Messdaten wie Eigenschaften des porösen Mediums und räumlich

verteilte Messungen der Saugspannung. Durch die Nutzung vereinfachter (semi-)analytischer/empirischer Ansätze lassen sich die Vorgänge z.B. für eindimensionale Prozesse in der vadosen Zone für viele Aufgaben unter gängigen Annahmen (z.B. bevorzugt vertikale Fließbewegung des Wassers) mit ausreichender Genauigkeit beschreiben. Durch Laufzeitkopplung dieser Lösungen an ein 3-D-Grundwassermodell über *ueflow* kann ein Modell sowohl die gesättigte Strömung direkt als auch die Wirkung der ungesättigten Strömung indirekt abbilden. Durch die Kombination der Methoden "(semi-)analytische/empirische Lösungen" und "partielle Verringerung der Dimensionalität" lassen sich somit "hybride" Modelle erstellen, deren Rechenzeit in praktikablen Grenzen liegt und welche trotzdem aussagekräftige Ergebnisse liefern. Da die Parameter für die Charakterisierung der vadosen Zonen häufig mit großen Ungenauigkeiten behaftet sind, sind derartige Modell-Vereinfachungen oft ohne praktische Auswirkungen auf die Modellergebnisse.

### Anwendungsszenario: Moor-Renaturierung

Moore sind wichtige Naturelemente, die neben anderen Ökosystemleistungen große Mengen Kohlenstoff binden und für einen lokalen Wasserrückhalt sorgen. In der Vergangenheit sind viele Moore ganz oder teilweise trocken gelegt worden. Bei der Renaturierung von Mooren spielt die Regulierung flurnaher Grundwasserstände oft eine entscheidende Rolle. Ein Grundwassermodell kann mit Randbedingungen wie Fluss, Drainage oder Brunnen technische Maßnahmen auf den Grundwasserstand im Moor abbilden. Dabei sind meist viele Modellelemente nötig, die nach bestimmten Regeln zusammenspielen. Ein Modellierer kann diese Elemente manuell steuern, indem die Ergebnisse eines Modelllaufes zur Korrektur von Werten von Randbedingungen sowie Quell-/Senkentermen dienen. Die mehrfache Wiederholen dieses Prozesses kann schrittweise zu einer Steuerung der flurnahen Grundwasserstände durch technische Maßnahmen führen. Dieser iterative Prozess ist allerdings äußerst arbeits- und zeitintensiv. Dies führt meist zu groben Steuerungsschritten und vereinfachter Abbildung technischer Maßnahmen.

Mit *ueflow* kann der Modellierer beliebig viele Randbedingungen und Quell-/Senkenterme, die technische Maßnahme repräsentieren, programmatisch verbinden. So kann ein Entwässerungsbrunnen bei Erreichen eines bestimmten Grundwasserstandes an einem oder mehreren definierten Punkten seine

Pumprate ändern oder ganz außer Betrieb gehen. Drainagen an verschiedenen Stellen im Modell können zur gleichen Zeit andere Ablaufwasserstände erhalten, die ihrerseits wieder von vom Modell errechneten Grundwasserständen zu bestimmten Zeitpunkten abhängen. Als Ergebnis kann eine detaillierte Steuerstrategie entstehen, die eine schnellstmögliche Renaturierung eines Moores mit den verfügbaren Mitteln ermöglicht.

### Anwendungsszenario: Klimaanpassungsmaßnahmen in der Landwirtschaft

Landwirtschaftlich genutzte Flächen stellen weltweit einen großen Anteil der durch den Menschen genutzten Landoberfläche dar. Im Zuge des globalen Bevölkerungszuwachses ergibt sich eine steigende Nachfrage nach Nahrungsmitteln zur Grundversorgung. Weiterhin kommt es im Zuge des anthropogen verstärkten Klimawandels in zunehmenden Maße zu Änderungen des Wasserdargebotes, insbesondere zu sinkenden Grundwasserständen. Da die Versorgung der Nutzpflanzen mittlerweile stark vom Grundwasserdargebot abhängt, sind weiterführende Anpassungsstrategien notwendig.

Die Abbildung der komplexen Verhältnisse und die Untersuchung der möglichen Änderungen aufgrund von Bewirtschaftungswechseln im ländlichen Raum erfordert adequate Planungswerkzeuge. Einfache Randbedingungen wie Brunnen muss der Modellierer manuell anpassen, um die gegenseitige Abhängigkeit von technischen Anlagen zu simulieren und die Bewirtschaftung zu optimieren. Ein Beispiel dafür ist die Wasserentnahme mit mehreren Pumpbrunnen und die Verteilung des Wassers auf Bewässerungsanlagen mit räumlich unterschiedlicher Bewässerungsrate, um eine vegetations- und bodenabhängige Bewässerungscharakteristik zu berücksichtigen. Über *ueflow* kann der Endnutzer beliebig viele hydraulische Randbedingungen und Quell-/Senkenterme in komplexer Zusammenschaltung automatisch koppeln, um derartige Bewässerungssysteme nachzubilden; und somit als kontinuierliches Planungswerkzeug für dessen Nutzungsoptimierung zu verwenden. Die Bewirtschaftung von landwirtschaftlichen Flächen wird somit optimiert; Ertragssteigerung und/oder Kosteneinsparung können die Folge sein. Als Folge lassen sich oft Auswirkungen auf die Umwelt durch Schonung des Grundwasserspeichern durch gezielte Entnahme/Bewässerung verringern. Die Planung von neuen Flächen kann unter Berücksichtigung von regionalen Klimaänderungsprognosen bei Simulation der

gegenseitigen Abhängigkeit von mehreren Bewirtschaftern mit berücksichtigt werden.

### 3.8.3. Beispielszenarien Phase 2

In ModSimple "Phase 2"' entsteht ein Werkzeug, welches die Planung von Geothermieanlagen in urbanen Räumen wesentlich effizienter macht und für eine nachhaltigere Nutzung des Untergrundes für geothermische Zwecke unter Berücksichtigung von verschiedenen Szenarien benachbarter Nutzer sorgen kann. Beispiele für erhöhte Wirtschaftlichkeit und Nachhaltigkeit sind unter anderem:

- Geothermische Anlagen können bei besserer Planung/Prognose mit gleichbleibender elektrischer Leistungsaufnahme eine potentiell größere Wärmemenge extrahieren. Dadurch kann das System effizienter arbeiten.
- Die Berücksichtigung von maximalen Wärme-Entnahmemengen im Verbund mehrerer EWS erlaubt eine bessere Planung. Dies kann eine geringere Belastung des Grundwassers oder alternativ eine bessere Auslastung einzelner EWS bedeuten.

Einige weitere Aspekte sind in den folgenden Abschnitten beschrieben.

#### Rückkopplung innerhalb von Anlagen

Generell können dynamische Rückkopplungen zwischen verschiedenen Gitterzellen des numerischen Modells, zwischen Gitterzellen und Randbedingungen bzw. Quell-/Senkentermen - "äußere Kopplung" - sowie innerhalb von Randbedingungen bzw. Quell-/Senkentermen - "innere Kopplung" - definiert und implementiert werden. Diesen Funktionsumfang wird *ueflow* mit Abschluss von ModSimple, Phase 2 haben. Als Anwendungsbeispiele für äußere und innere Kopplungen ergeben sich unter anderem die folgenden zwei Fälle:

- Innere Kopplung: Durch die dynamische Rückkopplung können u.a. Wasser-Wasser-Wärmepumpen effizient abgebildet werden, wobei die Entnahmemenge von der Temperaturdifferenz zwischen Entnahme und Eingabe abhängig ist.

- Äußere Kopplung: Verschiedene Elemente von Systemen zur künstlichen Grundwasseranreichung zur Stärkung des lokalen/regionalen Wasserhaushalts (Managed Aquifer Recharge - MAR) können durch dynamische Kopplung in ihrer Interaktion abgebildet werden. Beispielsweise kann nicht benötigtes Bewässerungswasser an einem Ort mit Wasserdefizit gezielt und automatisch in den Untergrund eingebracht werden. Ähnliche Interaktionen können auch auf Wärmeaustrag/-eintrag übertragen werden.

### Bessere Planung von Geothermieanlagen

Wie in Abschnitt 3.8 einleitend erwähnt, bestehen bereits mehrere analytische Tools zur Planung von Geothermieanlagen (z.B. EED); deren Nutzung auf lokaler Skala ist weithin anerkannt. Als Spezialfall der analytisch-numerischen Kopplung ist es (langfristig) denkbar, die dort zugrundeliegenden Algorithmen unmittelbar als Randbedingung bzw. als Quell-/Senkenterm in *ueflow* zu implementieren, um somit einen nahtlosen Übergang zwischen der lokalen Planung und dem regionalen Management des thermischen Grundwasserhaushaltes zu realisieren. Dies wiederum kann, im Zusammenspiel mit den bereits erwähnten Szenarien (siehe oben), zu einer verbesserten Planung und somit auch zu effizienteren Anlagen sowie verringertem Einfluss von Geothermieanlagen auf die Umwelt führen.

### Vereinfachte Abbildung der vertikalen Wärmeaustauschprozesse auf regionaler Skala

In Händel et al. (2013, Dr.-Ing. F. Händel war Teilprojektpartner der Phase 1 und ist Mitglied des wissenschaftlichen Beirates der Phase 2) wurde die Methodik der Kopplung (semi-)analytischer/empirischer und numerischer Modelle zur vereinfachten Berechnung des vertikalen Wärmetransports bereits erfolgreich eingesetzt und validiert. Dies erfolgte über den Interface-Manager des kommerziellen Programms *FEFLOW* (Diersch, 2014). Hintergrund war die Simulation von Grundwassertemperaturen und des Wärmehaushalts eines flachen Grundwasserleiters im ländlichen Raum Süd-Österreichs. Hierbei war es Teil der angewandten Forschungsaufgabe, sowohl den Einfluss der Atmosphäre als der stauenden Schichten unterhalb

des Grundwasserleiters zu berücksichtigen. Vergleichende, vollständig dreidimensionale Berechnungen des Wärmetransports waren mit sehr langen Rechenzeiten verbunden.

Der dort entwickelte 'vertikale' 1-D-Wärme-Austausch-Algorithmus kann auch in *ueflow* implementiert werden, sobald die Kopplung an das aktiv in Entwicklung befindliche Transportmoduls für MODFLOW6 in Phase 2 umgesetzt ist. Das Know-How zur technischen Umsetzung steht dem Projekt unmittelbar zur Verfügung. Die Implementierung wird es ermöglichen, derartige Probleme der oberflächennahen Geothermie mittels einer Open-Source-Software auch auf Regionalskala umzusetzen. Hier ist es denkbar, dass ein solches Feature zur langfristigen Optimierung von Grundwassertemperatur-Modellen in vielen Städten zum Einsatz kommen kann; insbesondere da es auf dem Industriestandard MODFLOW basiert, welches in zahlreichen Planungsbüros zur modelltechnischen Standardausrüstung zählt. Ein konkreter Anwendungsfall für die Stadt Basel (Schweiz) ist für Phase 2 geplant.

### 3.8.4. Werkzeug für andere Forschungsprojekte

Da *ueflow* als Open-Source-Ware verfügbar ist, können es andere Forschungsprojekte nutzen. Es gab dazu bereits Vorgespräche mit dem Projekt AZ 33923/01 - 33/2. Dieses Projekt untersucht Grundwasserökosysteme unter thermischem Stress. Dabei kommen vor allem statistische Modelle zur Korrelation von gemessenen Grundwassertemperaturen und Grundwasserökologie zum Einsatz. Die im o.g. Projekt abgeleiteten statistischen Modelle lassen sich durchaus in *ueflow*, analog zu analytischen Lösungen, einbauen. Damit ließe sich die Strömungs- und Transportmodellierung um ökologische Fragestellungen im Grundwasser erweitern. Die Bearbeiter des Projektes sind sehr daran interessiert, *ueflow* zu einem späteren Zeitpunkt einzusetzen.

Die Nutzung von *ueflow* in anderen Forschungsprojekten ist sehr erwünscht. Damit eröffnen sich weitere Umweltentlastungspotenziale, indem andere Forscher neue Lösungsansätze zur Lösung von Umweltproblemen entwickeln und dabei *ueflow* einsetzen.

# 4. Fazit

## 4.1. Gesamteinschätzung

Die erste Phase des Projektes *ModSimple* wurde erfolgreich abgeschlossen. Es sind für die Praxis relevante Werkzeuge für die effiziente Abbildung der Grundwasserströmung einstanden, die es einem fortgeschrittenem Modellierer ermöglichen, beliebig in den Ablauf einer Simulation einzugreifen. Bedingt durch die dynamische Entwicklung von MODFLOW 6 gab es in der Projektlaufzeit eine Änderung des Ansatzes. Statt FiPy kam MODFLOW 6 für die Abbildung der numerischen Grundwassermodellierung zum Einsatz. Der dadurch entstandene zusätzliche Arbeitsaufwand hat sich gelohnt, da mit *pymf6* ein leistungsfähiges System zur Steuerung von MODLFOW 6 zur Laufzeit entstanden ist.

Es wurden somit zahlreiche Grundlagen für die vorgesehene 2. Projektphase gelegt:

- hohe Flexibilität und Prozessgenauigkeit des *ueflow*-Moduls als Ganzes, insbesondere die *pymf6*-Schnittstelle
- Schaffung eines in Python umgesetzten Strömungssimulators im *base*-Modul (realisiert über Gleichungslöser *FiPy*)
- umfangreiche Datenbank (ca. 30 Gleichungen) für Randbedingungen und Quell-/Senkenterme

## 4.2. Arbeitspakete und Meilensteinerreichung

### 4.2.1. AP I - Analytische Wärmetransportgleichungen und Geothermische Randbedingungen

Aus der umfangreichen Recherche sind zahlreiche Gleichungen hervorgegangen, die sich für die vereinfachte, aber für viele Fälle adäquate Abbildung von Prozessen und Randbedingungen eignen. Die Gleichungen wurden in Python programmiert. Damit stehen diese den Nutzern komfortable für den Einbau in *ueflow* zur Verfügung. Das Projekt hat die Ziele dieses Arbeitspaketes voll erreicht; der zugehörige Meilenstein “Erfassung und Beurteilung der Berechnungsalgorithmen abgeschlossen” wurde entsprechend erreicht.

### 4.2.2. AP II - Numerisches Strömungsmodell

Das numerische Strömungsmodell ist einsatzfähig. Alles Tests haben eine vollständige Übereinstimmung der in *ueflow* integrierten Bibliothek *pymf6* mit MODFLOW 6 ergeben. Der ursprünglich auf FiPy basierende Ansatz hat Teilergebnisse geliefert, die sich, zusätzlich zu analytischen Ansätzen, für die Umsetzung von dynamische Randbedingungen, nutzen lassen. Der zugehörige Meilenstein "Programmierung des Grundwasserströmungsalgorithmus abgeschlossen" wurde in abgewandelter Form erreicht.

### 4.2.3. AP III - Numerisches Transportmodell

Es gibt keine direkten Ergebnisse für dieses Arbeitspaket, da durch die Umstellung auf MODFLOW 6 andere Arbeiten Vorrang hatten. Das Konzept von *pymf6* lässt sich auch auf die Transportkomponente von MODFLOW 6 übertragen, sodass im Folgeprojekt eine schnellere Umsetzung erfolgen kann. Der zugehörige Meilenstein 'Programmierung des Transportalgorithmus inkl. Schnittstelle für analytische Algorithmen abgeschlossen' wurde entsprechend nicht erreicht.

## 4.3. Ausblick

Die in diesem Projekt entstandenen Werkzeuge haben sich als effektiv erwiesen. Im Folgeprojekt sollen diese Werkzeug mit der Transportkomponente komplettiert und auf einen Feldstandort angewendet werden. Die Grundlagen für die erfolgreiche Umsetzung des Folgeprojektes sind gelegt. Insbesondere die in diesem Projekt erarbeiten Lösungsansätze lassen sich gut auf den Transport übertragen. Die bisher sehr guten numerischen und analytischen Ergebnisse der entwickelten Modelle bieten gute Voraussetzungen für eine Anwendung an einem Standort.

# Literatur


Aquaveo. (2020). Groundwater Modeling System Introduction. Zugriff unter https://www.aquaveo.com/software/gms-groundwater-modeling-system-introduction

Bakker, M., Post, V., Langevin, C. D., Hughes, J. D., W., J. T., S., … Fienen, M. N. (2016). Scripting MODFLOW model development using Python and FloPy. *Groundwater*. doi:doi:10.1111/gwat.12413

Bedekar, V., Morway, E. D., Langevin, C. D. & Tonkin, M. J. (2016). MT3D-USGS version 1: A U.S. Geological Survey release of MT3DMS updated with new and expanded transport capabilities for use with MODFLOW. *Techniques and Methods*, (6-A53). doi:10.3133/tm6A53

Chiang, W. & Kinzelbach, W. (2003). *3D-Groundwater Modeling with PMWIN*. doi:10.1007/978-3-662-05549-6

Diersch, H.-J. G. (2014). *FEFLOW: Finite element modeling of flow, mass and heat transport in porous and fractured media*. Berlin: Springer.

Guyer, J. E., Wheeler, D. & Warren, J. A. (2009). FiPy: Partial Differential Equations with Python. *Computing in Science & Engineering*, *11*(3), 6–15. doi:10.1109/MCSE.2009.52

Hähnlein, S., Molina-Giraldo, N., Blum, P., Bayer, P. & Grathwohl, P. (2010). Ausbreitung von Kältefahnen im Grundwasser bei Erdwärmesonden. *Grundwasser*, *15*(2), 123–133. doi:10.1007/s00767-009-0125-x

Hecht-Méndez, J., Molina-Giraldo, N., Blum, P. & Bayer, P. (2010). Evaluating MT3DMS for Heat Transport Simulation of Closed Geothermal Systems. *Groundwater*, *48*(5), 741–756. doi:10.1111/j.1745-6584.2010.00678.x

Hellström, G. & Sanner, B. (2000). Earth Energy Designer: EED. Lund, Sweden: Lund University.

Klein, S., Beckman, W., Duffie, J., Duffie, N., Mitchell, J., Braun, J., … Arias, D. (2007). TRNSYS: a TRaNsient SYstem Simulation program: Vol. 1 - Getting Started. Madison, WI 53706 – U.S.A.: Solar Energy Laboratory, University of Wisconsin-Madison.

Koenigsdorff, R. & Veser, S. (2008). GEO-HAND light: Computerprogramm zur Berechnung des Auslegung von Erdwärmesonden für Heiz- und Kühlzwecke. Germany: Hochschule Biberbach, Institute of Building & Energy Systems.

Köhler, M., Händel, F., Epting, J., Binder, M., Mueller, M. H., Huggenberger, P. & Liedl, R. (2015). Numerical Evaluation and Optimization of Depth-oriented Temperature Measurement for the Investigation of Thermal

Influences on Groundwater Resources. *Energy Procedia*, *76*, 371–380. European Geosciences Union General Assembly 2015 - Division Energy, Resources and Environment, EGU 2015. doi:https://doi.org/10.1016/j.egypro.2015.07.844

Kolditz, O., Bauer, S., Bilke, L., Böttcher, N., Delfs, J. O., Fischer, T., … Zehner, B. (2012). OpenGeoSys: an open-source initiative for numerical simulation of thermo-hydro-mechanical/chemical (THM/C) processes in porous media. *Environmental Earth Sciences*, *67*(2), 589–599. doi:10.1007/s12665-012-1546-x

Meurer, A., Smith, C. P., Paprocki, M., Čertík, O., Kirpichev, S. B., Rocklin, M., … Scopatz, A. (2017). SymPy: symbolic computing in Python. *PeerJ Computer Science*, *3*, e103. doi:10.7717/peerj-cs.103

Ogata, A. & Banks, R. B. (1961). A solution of the differential equation of longitudinal dispersion in porous media. doi:10.3133/pp411A

Prommer, H. & Post, V. (2010). PHT3D: A Reactive Multicomponent Transport Model for Saturated Porous Media. Australia. Zugriff unter http://www.pht3d.org

Rumbaugh, J. O. & Rumbaugh, D. B. (2020). Groundwater Vistas. Environmental Simulations, Inc.: Environmental Simulations, Inc.

Sames, D. & Blankenburg, R. (2019). Anwenderdokumentation. Version 2016, 29.11.2019. Zugriff unter https://ibgw-leipzig.de/index.php/hauptpcgeofim/dokumentation

Šimůnek, J., Šejna, M. & van Genuchten, M. T. (2018). New features of version 3 of the HYDRUS (2D/3D) computer software package. *Journal of Hydrology and Hydromechanics*, *66*(2), 133–142. doi:10.1515/johh-2017-0050

Šimůnek, J., van Genuchten, M. T. & Šejna, M. (2016). Recent Developments and Applications of the HYDRUS Computer Software Packages. *Vadose Zone Journal*, *15*(7), vzj2016.04.0033. doi:10.2136/vzj2016.04.0033

USGS. (2020a). MODFLOW 6 Repository. GitHub. Zugriff unter https://github.com/MODFLOW-USGS/modflow6

USGS. (2020b). MODFLOW 6: USGS Modular Hydrologic Model. USGS. Zugriff unter https://www.usgs.gov/software/modflow-6-usgs-modular-hydrologic-model

USGS. (2020c). MODFLOW and Related Programs. USGS. Zugriff unter https://www.usgs.gov/mission-areas/water-resources/science/modflow-and-related-programs?qt-science_center_objects=0#qt-science_center_objects

van der Walt, S., Colbert, C. & Varoquaux, G. (2011). The NumPy Array: A Structure for Efficient Numerical Computation. *Comput. Sci. Eng.*

Waterloo Hydrogeologic. (2020). Visual MODFLOW Flex | Waterloo Hydrogeologic. Zugriff unter https://www.waterloohydrogeologic.com/visual-modflow-flex/

Winston, R. (2020). *ModelMuse version 4.3: U.S. Geological Survey Software Release*. U.S. Geological Survey Software. doi:doi:10.5066/P9XMX92F

# Anhang

# Anhang A.

# Abbildungen

TECHNISCHE UNIVERSITÄT DRESDEN

# Entwicklung eines Simulationswerkzeuges für hydrogeologische und geothermische Fragestellungen unter Nutzung dynamischer, gekoppelter Randbedingungen – Konzeptionelle Aspekte und erste Ergebnisse

**Engelmann, Christian[1] (Email: Christian.Engelmann@tu-dresden.de, Tel.: +49 351 463-40552); Müller, Mike[2] (info@hydrocomputing.com); Händel, Falk[1]; Binder, Martin[1]; Epting, Jannis[3]; Huggenberger, Peter[3]; Börke, Peter[4]**

1 Institut für Grundwasserwirtschaft, Fakultät Umweltwissenschaften, Technische Universität Dresden, Bergstraße 66, 01069 Dresden
2 hydrocomputing GmbH & Co. KG Leipzig, Zur Schule 20, 04158 Leipzig
3 Angewandte und Umweltgeologie, Departement Umweltwissenschaften, Universität Basel, Bernoullistraße 32, 4056 Basel, Schweiz
4 Abteilung Wasser, Boden, Wertstoffe, Sächsisches Landesamt für Umwelt, Landwirtschaft und Geologie, Zur Wetterwarte 11, 01109 Dresden

## HINTERGRUND

Komplexität von üblichen geothermischen Fragestellungen und Prozessen des Wärmetransports im Untergrund auf regionaler Skala sehr hoch, daher numerische 3-D-Modellierung unter Abbildung aller Prozesse des Wärmetransports oft nicht durchführbar, u.a. bedingt durch:

- erhebliche Anforderungen an die Rechentechnik
- hohe Anzahl an unsicheren Modellparametern
- hoher Grad an hydro(geo)logischen Heterogenitäten
- Vielzahl von sich gegenseitig beeinflussende Wärmerandbedingungen

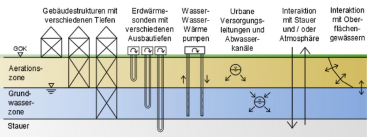



**Abb. 1** Beispiele für geothermische Interaktionen mit oberflächennahen Grundwasserleitern (verändert nach [1]).

**Ziele:**

- praktikabler Kompromiss zwischen möglichst exakter Prozessnachbildung und Recheneffizienz
- starke Flexibilität bei der gewählten Komplexität der Abbildung von Prozessen (z.B. Nitrat, gelöste Schadstoffe, Wärme)
- hohe Anwenderfreundlichkeit (grafische Benutzeroberfläche, keine fortgeschrittenen Programmierkenntnisse erforderlich)

## METHODIK

**ueflow** user-extensible flow model

- Software zur freien Verwendung und Erweiterung (*open source*)
- umfassende Dokumentation inkl. Handbuch und Online-Tutorials
- Python [2] als Programmiersprache
- speichersparende Datenhaltung im Datenbankformat HDF5 [3]
- Finite-Volumen-Gleichungslöser mit Parallelisierung (FiPy-Bibliothek [4])
- Benchmark des numerischen Strömungsmodells gegen MODFLOW [5]
- Benchmark des numerischen Transportmodells gegen MT3D [6]
- Schnittstelle zur dynamischen Kopplung mit (1-D, 2-D, 3-D) analytischen und einfachen numerischen Lösungen durch den Nutzer
- Erweiterung um zusätzliche Komponenten (siehe Abb. 2) durch modularen Aufbau möglich

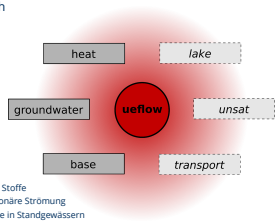



**Abb. 2** Modularer Aufbau von *ueflow*. Zusätzliche Komponenten (gestrichelt) können zu einem späteren Zeitpunkt ergänzt werden.

*unsat* ... wasserungesättigte Strömung
*base* ... generelle Programmfunktionen
*transport* ... (reaktiver) Transport gelöster Stoffe
*groundwater* ... wassergesättigte (in-)stationäre Strömung
*lake* ... Strömungs- und Transportprozesse in Standgewässern
*heat* ... konvektiv-diffusiver Wärmetransport inkl. Wärmeleitung

## AKTUELLER STAND

- Recherche von analytischen/einfachen numerischen Lösungen zur Approximation des Wärmetransports
- prinzipiell lauffähige Version des numerischen Strömungsmodells
- Recherche und Automatisierung von Benchmarks unter Nutzung der FloPy-Bibliothek [7]

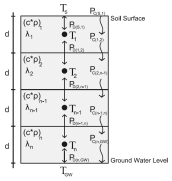



**Abb. 3** Beispiel für vereinfachte Wärmetransportrandbedingung, hier: Kopplung der Oberfläche mit dem Grundwasser über die ungesättigte Zone [8].

$T$ ... Temperatur
$P_c$ ... Wärmekonvektion
$P_d$ ... Wärmeleitung
$c\,p$ ... Wärmekapazität des Bodens
$\lambda$ ... Wärmeleitfähigkeit des Bodens

## AUSBLICK

- Finalisierung des numerischen Strömungsmodells (ca. Anfang 2019)
- Finalisierung des numerischen Transportmodells (ca. Mitte 2019)
- Kopplung von analytischen/einfachen numerischen Lösungen an numerische Modelle (ca. Ende 2019)
- grafische Benutzeroberfläche und Validierung an realen Teststandort (Folgeprojekt)

**Referenzen**: [1] Köhler, M., Händel, F., Epting, J., Binder, M., Mueller, M.H., Huggenberger, P., Liedl, R. (2015), Numerical Evaluation and Optimization of Depth-oriented Temperature Measurement for the Investigation of Thermal Influences on Groundwater Resources, Energy Procedia 76, 371-380, https://doi.org/10.1016/j.egypro.2015.07.844; [2] PSF (2018), Python Software Foundation, Python Language Reference, Version 3.6, *http://www.python.org*; [3] Hierarchical Data Format, Version 5, National Center for Supercomputing Applications, *www.hdfgroup.org*; [4] Guyer, J. E., Wheeler, D., Warren, J. A. (2009), FiPy: Partial Differential Equations with Python, Computing in Science & Engineering 11(3): 6-15 (2009), doi:10.1109/MCSE.2009.52, *http://www.ctcms.nist.gov/fipy*; [5] Harbaugh, A.W. (2005), MODFLOW-2005 - The U.S. geological Survey Modular Groundwater Model - the Ground-Water Flow Process, U.S. Geological Survey Techniques and Methods 6-A16, U.S. Geological Survey, Reston/Virginia; [6] Zheng, C. et al. (1990), MT3D, A modular three-dimensional transport model for simulation of advection, dispersion and chemical reactions of contaminants in groundwater systems, Rockville, Maryland; [7] Bakker, M., Post, V., Langevin, C. D., Hughes, J. D., White, J. T., Starn, J. J. and Fienen, M. N. (2016), Scripting MODFLOW Model Development Using Python and FloPy, Groundwater 54: 733–739, doi:10.1111/gwat.12413; [8] Händel, F., Liedl, R., Fank, J., Rock, G. (2013), Regional modeling of geothermal energy systems in shallow aquifers: the Leibnitzer Feld case study (Austria), Environmental Earth Sciences 70(8): 3433-3446

Mitglied im Netzwerk von: DRESDEN concept
gefördert durch: DBU Deutsche Bundesstiftung Umwelt
in Kooperation mit: LANDESAMT FÜR UMWELT, LANDWIRTSCHAFT UND GEOLOGIE; Freistaat SACHSEN; Universität Basel

Abbildung A.1.: Posterbeitrag, veröffentlicht auf der Konferenz *KGM-MODREG 2018* in Seggau, Österreich.

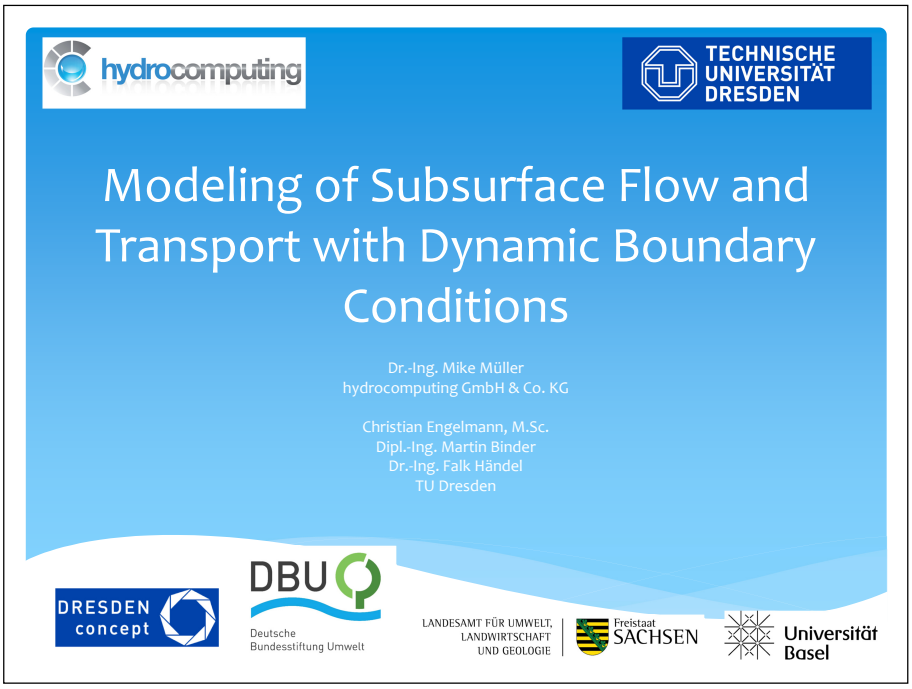



Abbildung A.2.: Vortragsbeitrag, veröffentlicht auf der Konferenz *GeoPython 2019* in Basel, Schweiz.

# Anhang B.

# Tabellen

Tabelle B.1.: Übersicht über ausgewählte (semi-)analytische Ansätze zur Abbildung von Stoff- und Wärmetransportrandbedingungen (nächste Seite)

| Reference | Surface | Dimension 1D | 2D | 3D | Boundary Condition Type | Steady (Y/N) | Isotropic (Y/N) | Homogeneous (Y/N) | Processes | Input variables Time dependent | Space dependent | Time and space dependent | Constants | Aquifer heat capacity | Rock heat capacity | Aquifer heat conductivity | Rock heat conductivity | Layer thickness / Hydraulic diameter (Karst) | Darcy velocity / hydraulic flow rate / flow velocity (Karst) | Hydraulic conductivity | Porosity | Energy abstraction capacity / flux | Dispersivity | Diffusivity | Borehole depth | Convection heat transfer coefficient | Conductive heat transfer coefficient | Water and conduit temperature | Kronecker delta function | Initial temperature | Water saturation | Pipe radius / geometry | Contact surface area | Mean and amplitude of soil surface temperature | Temperature difference between construction and unsaturated zone+ | Output |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Händel et al. (2013)** | Soil surface | | x | | 1st | N | N | Y | Heat conduction, convection, storage | Percolation water flux, Time step length | | Temperature of previous layer and time step | Heat transport aquifer properties | Y | N | Y | N | Y | N | N | N | N | N | N | N | N | N | N | N | N | N | N | N | N | N | Temperature difference |
| **Molina-Giraldo et al. (2011)** | Geothermal probe | | x | | 3rd | Y, N | Y, N | Y | Conduction, convection, storage | | | | Thermal and hydraulic transport properties | Y | N | Y | N | N | Y | N | N | N | N | N | N | N | N | N | N | N | N | N | N | N | N | Temperature difference |
| **Gringarten and Sauty (1975)** | Geothermal probe | x | x | | 3rd | Y | N | Y | (Conduction), convection, storage | | | | Thermal and hydraulic transport parameters | Y | Y | Y | Y | N | Y | N | Y | N | N | N | N | N | N | N | N | Y | N | N | N | N | N | Temperature difference |
| **Pannike et al. (2006)** | Geothermal probe | | x | | 2nd | | N | Y | Convection, conduction | | | | All parameters | | | | | | | | | | | | | | | | | | | | | | | Dimension of thermal plumes |
| **Hähnlein et al. (2010)** | Geothermal probe | | x | x | 2nd | Y | N | Y | Convection, conduction, storage | | | | Thermal and hydraulic transport parameters | Y | N | Y | N | N | Y | N | Y | Y | Y | N | N | N | N | N | N | Y | N | N | N | N | N | Temperature change |
| **Banks et al. (2009)** | Geothermal probe | x | | | 2nd | Y | Y | Y | (Convection), conduction, storage | | Temperature | | Thermal and hydraulic transport parameters | | | | | | | | | | | | | | | | | | | | | | | | time of thermal feedback (temperature and solute concentration change) |
| **Diao et al. (2004)** | Geothermal probe | | x | | 3rd | Y, N | Y | Y | Convection, conduction, storage | | (Radial) coordinate, polar angle | | Heat capacity, thermal diffusivity, advection velocity, heat flux | Y | N | Y | N | N | Y, N | Y, N | N | Y | N | Y | N | N | N | N | N | N | N | N | N | N | N | N | Temperature rise |
| **Shook (2001)** | Geothermal probe | x | | | 2nd | N | N | N | Convection, conduction, storage | Temperature, flow rate, Tracer Concentration | | | Thermal transport parameters | | | | | | | | | | | | | | | | | | | | | | | | Predicted temperature based on tracer slug test |
| **Hecht-Méndez et al. (2010)** | Geothermal probe | x | | | 3rd | | Y | Y | 1. Convection 2. Conduction | | | | | | | | | | | | | | | | | | | | | | | | | | | | Temperature difference |
| **Luckner et al. (1987)** | Geothermal probe | x | | | 1st | Y | Y | Y | Convection, conduction | | | | Thermal and hydraulic transport parameters | Y | N | Y | N | Y | Y | N | N | N | Y | N | N | N | N | N | N | N | N | N | N | N | N | Temperature |
| **Diao et al. (2004)** | Fields of geothermal probes | | x | | 2nd | N | Y | Y | Conduction, Diffusion | | Depth, vertical distance | | Thermal transport parameters | N | N | Y | N | N | N | N | N | Y | N | Y | Y | N | N | N | N | Y | N | N | N | N | N | Temperature of borehole wall |
| **Javed (2012)** | Fields of geothermal probes | | x | | 2nd | Y | Y | Y | Diffusion, Conduction | | Distance | | Thermal transport parameters | Y | N | Y | N | N | N | N | N | Y | N | Y | Y | N | N | N | N | N | N | N | N | N | N | Mean temperature for N boreholes |
| **Luhmann et al. (2015)** | Karst | | x | | 1st, 3rd | Y | Y | Y | 1. Advection 2. Conduction | | Temperature | | Thermal transport parameters | Y, N | Y, N | Y, N | Y, N | Y, N | Y, N | N | N | N | N | Y, N | N | Y, N | N | Y, N | N | N | N | N | N | N | N | N | Rock and water temperature change in time |
| **Anderson (2005)** | Surface water | | | x | 1st | N | Y | Y | Advection, diffusion, dispersion | | | | Thermal transport parameters | Y | Y | N | Y | N | Y | N | N | N | N | N | N | N | N | N | N | Y | N | N | N | N | N | Temperature change |
| **Goto et al. (2005)** | Surface water | x | | | 1st | N | Y | Y | Diffusion, 1. Conduction 2. Convection | Amplitude and phase of boundary temperature variation | | | Thermal and hydraulic transport parameters | Y | N | Y | N | N | Y | N | N | N | N | Y | N | N | N | N | N | N | N | N | N | N | N | Subsurface Temperature |
| **Constantz (2010)** | Surface water | x | | x | | N | Y, N | Y | Convection, dispersion | | Temperature | | Thermal and hydraulic transport parameters | Y | N | Y | N | N | Y,N | N | Y, N | Y, N | Y, N | N | N | N | N | N | Y,N | N | N | N | N | N | N | Temperature change |
| **Rau et al. (2010)** | Surface water | x | | | 1st | Y, N | Y | Y | 1. Dispersive 2. Conductive | Sinusoidal temperature fluctuations | | | Thermal and hydraulic transport parameters | Y | N | Y | N | N | Y | N | N | N | Y | Y, N | N | N | N | N | N | Y | N | N | N | N | N | Sediment temperature response |

Tabelle B.2.: Übersicht über die Modelltypen A, B und C - große Version. Dargestellt sind Parameterwerte für die Basismodelle sowie untersuchte Parameterspannweiten (nächste Seite).

| scenario | parameter class | parameter | unit | default value | lower range | upper range | comments |
|---|---|---|---|---|---|---|---|
| A | model domain | number of layers | - | 2 | 2 | 2 | |
| | | domain length in x-direction | m | 3000 | 100 | 5000 | |
| | | domain length in y-direction | m | -1000 | -100 | -5000 | |
| | | model top | m | 0 | -20 | 20 | |
| | | upper aquifer bottom | m | -10 | -30 | 10 | |
| | | lower aquifer bottom | m | -20 | -40 | 0 | |
| | | | | | | | |
| | spatial discretization | mesh size width | m | 10 | 0.1 | 50 | |
| | | mesh size height | m | 10 | 0.1 | 50 | |
| | | | | | | | |
| | temporal discretization | length time period 1 | s | 1 | 1 | 86400 | |
| | | length time period 2 | d | 30 | 10 | 50 | to be converted to s |
| | | length time period 3 | d | 180 | 90 | 270 | to be converted to s |
| | | time step size period 1 | s | 1 | 1 | 86400 | |
| | | time step size period 2 | d | 1 | 0.1 | 10 | |
| | | time step size period 3 | d | 1 | 0.1 | 10 | |
| | | | | | | | |
| | initial conditions | initial hydraulic head (homogeneous) | m | 0 | -50 | 50 | |
| | | | | | | | |
| | hydraulic parameters layer 1 (confined) | horizontal hydraulic conductivity Kx | m/s | 1.00E-03 | 1.00E-04 | 1.00E-02 | |
| | | horizontal hydraulic conductivity Ky | m/s | 1.00E-03 | 1.00E-04 | 1.00E-02 | |
| | | vertical hydraulic conductivity Kz | m/s | 1.00E-04 | 1.00E-05 | 1.00E-03 | |
| | | specific storage | - | 1.00E-05 | 1.00E-06 | 1.00E-04 | |
| | | specific yield | - | 0.2 | 0.1 | 0.3 | |
| | | | | | | | |
| | hydraulic parameters layer 2 (confined) | horizontal hydraulic conductivity Kx | m/s | 1.00E-05 | 1.00E-06 | 1.00E-04 | |
| | | horizontal hydraulic conductivity Ky | m/s | 1.00E-05 | 1.00E-06 | 1.00E-04 | |
| | | vertical hydraulic conductivity Kz | m/s | 1.00E-06 | 1.00E-07 | 1.00E-05 | |
| | | specific storage | - | 1.00E-04 | 1.00E-05 | 1.00E-03 | |
| | | specific yield | - | 0.3 | 0.2 | 0.4 | |
| | | | | | | | |
| | solver configuration | complexity | - | moderate | | | |
| | | Outer Hclose (nonlinear) | m | 1.00E-04 | | | |
| | | 1 | - | 200 | | | |
| | | Inner maximum iterations (linear) | - | 200 | | | |
| | | Inner Hclose | m | 1.00E-05 | | | |
| | | Inner Rclose | m | 0.01 | | | |
| | | | | | | | |
| | fixed-head boundary conditions | western hydraulic head | m | 6 | 3 | 9 | for all stress periods, both layers |
| | | eastern hydraulic head | m | 3 | 0 | 6 | for all stress peripds, both layers |
| | | | | | | | |
| | areal recharge | areal recharge | m/s | 4.76E-09 | 4.76E-10 | 9.52E-09 | 150 mm annual recharge in urban areas, homogeneously distributed, for all stress periods |
| | | | | | | | |
| B | abstraction well | x coordinate | m | 1000 | 800 | 1200 | |
| | | y coordinate | m | -500 | -300 | -500 | |
| | | abstraction rate | $m^3/s$ | -0.001157407 | -1.15741E-05 | -0.115740741 | 100 $m^3/d$ to be converted to $m^3/s$ |
| | | | | | | | |
| | injection well | x coordinate | m | 2000 | | | |
| | | y coordinate | m | -500 | -300 | -500 | |
| | | injection rate | $m^3/s$ | -0.001157407 | 1.15741E-05 | 0.115740741 | 100 $m^3/d$ to be converted to $m^3/s$ |
| | | | | | | | |
| C | abstraction well | x coordinate | m | 1000 | 800 | 1200 | |
| | | y coordinate | m | -500 | -300 | -500 | |
| | | abstraction rate | $m^3/s$ | -0.001157407 | -1.15741E-05 | -0.115740741 | |
| | | | | | | | |
| | single municipal sewer pipe | western starting x coordinate | m | 1900 | 1800 | 2000 | |
| | | eastern ending x coordinate | m | 2100 | 2000 | 2200 | |
| | | y coordinate | m | -500 | -500 | -500 | |
| | | outflow rate | $m^3/s$ | 0.000115741 | 1.15741E-05 | 0.001157407 | 10 $m^3/d$ to be converted to $m^3/s$ |

*all remaining model parameters are default values pre-defined by MF6*

# Anhang C.

# Weitere Beispiel-Codes zur Implementierung der analytischen Gleichungen

## Stationärer 1-D-Wärmetransport über Konduktion und Konvektion im direkten Abstrom einer Linienquelle

Umsetzung der eigentlichen Lösungsgleichung als Python-Code

```
class Stationary1D(Model):
    """Stationary 1D model.
    """

    def make_equation(self):
        """Create the equation.
        """


        def K_0(z):
            """Helper function for the Bessel function 0-th
                                                order.
            """
            return sympy.functions.special.bessel.besselk(0, z)

        p = self.params
        x = self.symbols['x']
        exp = sympy.exp
        pi = sympy.pi
        self.equation = (
            (p.F_L / (2 * pi * p.n * p.rho_w * p.c_w * p.D_th))
                                                    *
            exp((p.v_a * x) / (2 * (p.D_th))) *
            K_0((p.v_a * x) / (2 * (p.D_th)))
            )
        return self.equation
```

Aufruf der Modell-Instanzen

```
def make_model():
    """Create the model.
    """
    # pylint: disable=invalid-name, non-ascii-name
    param_list = []
```

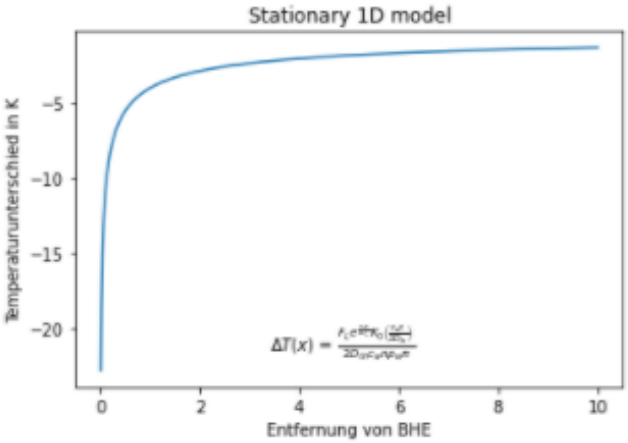
```
v_a = Parameter(
    'v_a',
    value=0.5 / 86400,
    unit='m/s',
    long_name='pore velocity',
    description='pore velocity')
param_list.append(v_a)

n = Parameter(
    'n',
    value=0.4,
    long_name='total porosity')
param_list.append(n)

D = Parameter(
    'D',
    value=0.1,
    unit='m',
    long_name='dispersivity',
    description='longitudinal dispersivity for heat
                                        transport')
param_list.append(D)

ratio = Parameter(
    'ratio',
    value=0.1,
    long_name='dispersivity ratio',
    description='ratio between longitudinal and transversal
                                         dispersivity')
param_list.append(ratio)

rho_w = Parameter(
    'rho_w',
    value=1000,
    unit='kg/m3',
    long_name='density',
    description='water density 1000 kg per m^3')
param_list.append(rho_w)

c_w = Parameter(
    'c_w',
    value=4186,
    unit='J/Kg/K',
    long_name='specific heat capacity',
    description='water specific heat capacity')
param_list.append(c_w)

T = Parameter(
```

```
        'T',
        value=285.15,
        unit='K',
        long_name='temperature',
        description=(
            'background temperature value in K '
            'e.g., 285.15 K is 10 degree centigrade'))
    param_list.append(T)

    L = Parameter(
        'L',
        value=1,
        unit='m',
        long_name='length BHE',
        description='length of borehole heat exchanger')
    param_list.append(L)

    E = Parameter(
        'E',
        value=-50,
        unit='m',
        long_name='energy extraction',
        description='absolute energy extraction of BHE')
    param_list.append(E)

    C_h = Parameter(
        'C_h',
        value=1.115,
        unit='',
        long_name='heat conductivity ',
        description='effective thermal conductivity')
    param_list.append(C_h)

    # special

    F_L = Parameter(
        'F_L',
        value=E.value / L.value,
        unit='',
        long_name='energy extraction PL',
        description='energy extraction PL')
    param_list.append(F_L)

    c_v = Parameter(
        'c_v',
        value=rho_w.value * c_w.value,
        unit='J/K/m^3',
        long_name='vol heat conductivity',
```

```
        description='volumetric heat capacity of water')
    param_list.append(c_v)

    D_th = Parameter(
        'D_th',
        value=(
            (C_h.value / (c_v.value * n.value)) +
            D.value * ratio.value * v_a.value),
        unit='J/K/m^3',
        long_name='thermal dispersion coefficient')
    param_list.append(D_th)

    return Stationary1D(
        param_list,
        model_name='Stationary 1D model',
        left_side=r'$\Delta T(x)$')
```

[1]:
```
from ueflow.groundwater.analytic_bcs.bcs.stationary_1d import model
```

[2]:
```
model.test()
```




[3]:
```
my_model = model.make_model()
```

[4]:
```
my_model
```

[4]:
$$\Delta T(x) = \frac{F_L e^{\frac{v_a x}{2D_{th}}} K_0\left(\frac{v_a x}{2D_{th}}\right)}{2D_{th} c_w n \rho_w \pi}$$

[5]:
```
my_model.params.v_a
```

[5]:
## Parameter Pore Velocity $v_a$

**Value**: 6e-06 m/s

**Description**: pore velocity

[6]:
```
my_model.params.ρ_w
```

[6]:
## Parameter Density $\rho_w$

**Value**: 1000 kg/m3

**Description**: water density 1000 kg per m³

[7]:
```
import numpy as np

x_values = np.linspace(start=0.01, stop=10, num=10)
my_model.calc_array({'x': x_values})
```

[7]:
```
array([-22.8062872 ,  -3.79189825,  -2.71981794,  -2.23191153,
       -1.93786667,  -1.73598952,  -1.58639837,  -1.46982705,
       -1.37567804,  -1.29757631])
```



Abbildung C.1.: Testsession im Jupyter Notebook - Stationärer 1-D-Wärmetransport

## Stationärer 2-D-Wärmetransport über Konduktion und Konvektion um eine vertikale Linienquelle

### Umsetzung der eigentlichen Lösungsgleichung als Python-Code

```
class Stationary2D(Model):
    """Stationary 1D model.
    """

    def make_equation(self):
        """Create the equation.
        """
        #  pylint: disable=invalid-name, non-ascii-name, no-
                                            member

        def K_0(z):
            """Helper function for the Bessel function 0-th
                                                order.
            """
            return sympy.functions.special.bessel.besselk(0, z)

        p = self.params
        x = self.symbols['x']
        y = self.symbols['y']
        sqrt = sympy.sqrt
        exp = sympy.exp
        pi = sympy.pi

        self.equation = (
            p.F_L / (2 * pi * sqrt(p.C_t_x * p.C_t_y)) *
            exp(p.c_v * p.v_f * x / (2 * p.C_t_x)) *
            K_0(p.c_v * p.v_f * 0.5 * sqrt(
                ((p.C_t_y * x**2) + (p.C_t_x * y**2)) /
                (p.C_t_y * p.C_t_x**2))
                )
            )
        return self.equation
```

### Aufruf der Modell-Instanzen

```
def make_model():
    """Create the model.
    """
    # pylint: disable=invalid-name, non-ascii-name
    param_list = []
    v_f = Parameter(
        'v_f',
```

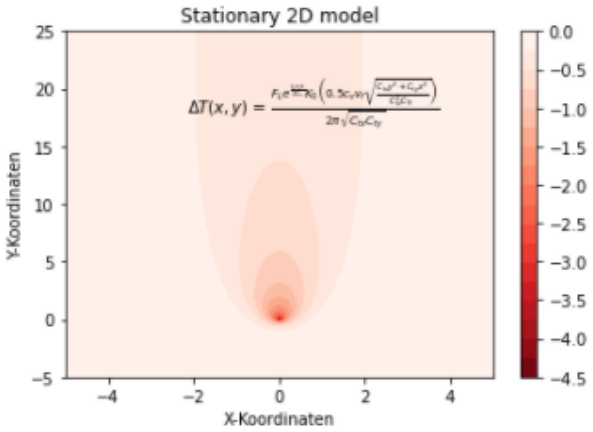
```
        value=0.5 / 86400,
        unit='m/s',
        long_name='Darcy velocity',
        description='Darcy velocity')
    param_list.append(v_f)

    D = Parameter(
        'D',
        value=0.5,
        unit='m',
        long_name='dispersivity',
        description='longitudinal dispersivity for heat
                                        transport')
    param_list.append(D)

    ratio = Parameter(
        'ratio',
        value=0.1,
        long_name='dispersivity ratio',
        description='ratio between longitudinal and transversal
                                         dispersivity')
    param_list.append(ratio)

    rho_w = Parameter(
        'rho_w',
        value=1000,
        unit='kg/m3',
        long_name='density',
        description='water density 1000 kg per m^3s')
    param_list.append(rho_w)

    c_w = Parameter(
        'c_w',
        value=4186,
        unit='J/Kg/K',
        long_name='specific heat capacity',
        description='water specific heat capacity')
    param_list.append(c_w)

    T = Parameter(
        'T',
        value=285.15,
        unit='K',
        long_name='temperature',
        description=(
            'background temperature value in K '
            'e.g., 285.15 K is 10 degree centigrade'))
    param_list.append(T)
```

```
L = Parameter(
    'L',
    value=1,
    unit='m',
    long_name='length BHE',
    description='length of borehole heat exchanger')
param_list.append(L)

E = Parameter(
    'E',
    value=-50,
    unit='m',
    long_name='energy extraction',
    description='absolute energy extraction of BHE')
param_list.append(E)

C_t = Parameter(
    'C_t',
    value=1.115,
    unit='',
    long_name='bulk thermal conductivity ',
    description='heat conductivity of matrix')
param_list.append(C_t)

# special

F_L = Parameter(
    'F_L',
    value=E.value / L.value,
    unit='',
    long_name='energy extraction PL',
    description='energy extraction PL')
param_list.append(F_L)

c_v = Parameter(
    'c_v',
    value=rho_w.value * c_w.value,
    unit='J/K/m^3s',
    long_name='vol heat conductivity',
    description='volumetric heat capacity of water')
param_list.append(c_v)

disp_x = D.value * rho_w.value * c_w.value * v_f.value

C_t_x = Parameter(
    'C_t_x',
    value=C_t.value + disp_x,
```

```
        unit='',
        long_name='effective thermal dispersion coefficient x')
    param_list.append(C_t_x)

    C_t_y = Parameter(
        'C_t_y',
        value=C_t.value + ratio.value * disp_x,
        unit='',
        long_name='effective thermal dispersion coefficient y')
    param_list.append(C_t_y)

    return Stationary2D(
        param_list,
        model_name='Stationary 2D model',
        left_side=r'$\Delta T(x, y)$')
```

[1]:
```
from ueflow.groundwater.analytic_bcs.bcs.stationary_2d import model
```

[2]:
```
model.test()
```




[3]:
```
my_model = model.make_model()
```

[4]:
```
my_model
```

[4]:
$$\Delta T(x,y) = \frac{F_L e^{\frac{c_v v_f x}{2C_{tx}}} K_0\left(0.5 c_v v_f \sqrt{\frac{C_{tx} y^2 + C_{ty} x^2}{C_{tx}^2 C_{ty}}}\right)}{2\pi\sqrt{C_{tx} C_{ty}}}$$

[5]:
```
my_model.params.C_t_x
```

[5]:
**Parameter Effective Thermal Dispersion Coefficient X $C_{tx}$**

**Value**: 13.227269

**Description**: effective thermal dispersion coefficient x

[6]:
```
my_model.equation
```

[6]:
$$\frac{F_L e^{\frac{c_v v_f x}{2C_{tx}}} K_0\left(0.5 c_v v_f \sqrt{\frac{C_{tx} y^2 + C_{ty} x^2}{C_{tx}^2 C_{ty}}}\right)}{2\pi\sqrt{C_{tx} C_{ty}}}$$

Abbildung C.2.: Testsession im Jupyter Notebook - Stationärer 2-D-Wärmetransport

[7]:
```
import numpy as np

x_values = np.linspace(start=-5, stop=+25., num=10)
y_values = np.linspace(start=-5., stop=+5., num=5)

my_model.calc_array({'x': x_values, 'y': y_values}, sweep='x')
```

[7]:
```
array([[-3.83448062e-08, -5.47745370e-06, -8.64542849e-05,
        -5.47745370e-06, -3.83448062e-08],
       [-1.89870604e-06, -5.55552389e-04, -6.44492844e-02,
        -5.55552389e-04, -1.89870604e-06],
       [-4.01860836e-05, -1.17582576e-02, -1.36406810e+00,
        -1.17582576e-02, -4.01860836e-05],
       [-3.63546865e-04, -5.19317037e-02, -8.19672527e-01,
        -5.19317037e-02, -3.63546865e-04],
       [-1.65100784e-03, -1.00563254e-01, -6.40915343e-01,
        -1.00563254e-01, -1.65100784e-03],
       [-4.62505768e-03, -1.38128711e-01, -5.43968675e-01,
        -1.38128711e-01, -4.62505768e-03],
       [-9.43666474e-03, -1.63135028e-01, -4.80891863e-01,
        -1.63135028e-01, -9.43666474e-03],
       [-1.56732490e-02, -1.78903929e-01, -4.35660039e-01,
        -1.78903929e-01, -1.56732490e-02],
       [-2.27249254e-02, -1.88501945e-01, -4.01184962e-01,
        -1.88501945e-01, -2.27249254e-02],
       [-3.00482304e-02, -1.94054025e-01, -3.73782968e-01,
        -1.94054025e-01, -3.00482304e-02]])
```

[8]:
```
my_model.calc_one({'x': -5, 'y': -5})
```

[8]: $-3.83448062499796 \cdot 10^{-8}$

Abbildung C.3.: Testsession im Jupyter Notebook - Stationärer 2-D-Wärmetransport (Fortsetzung)

## Instationärer radialer Wärmetransport über Konduktion um eine vertikale Linienquelle

Umsetzung der eigentlichen Lösungsgleichung als Python-Code

```
class RadialHeat(Model):
    """Stationary 1D model.
    """

    def make_equation(self):
        """Create the equation.
        """

        #  pylint: disable=invalid-name, non-ascii-name, no-
                                        member
```

```
        pi = sympy.pi
        p = self.params
        r = self.symbols['r']
        t = self.symbols['t']
        Ei = sympy.Ei
        self.equation = (
            -(p.F_L / (4 * pi * p.lambda_m )) *
            Ei(-(r**2 / (4 * (p.lambda_m / p.c_v) * t)))
            )
        return self.equation
```

## Aufruf der Modell-Instanzen

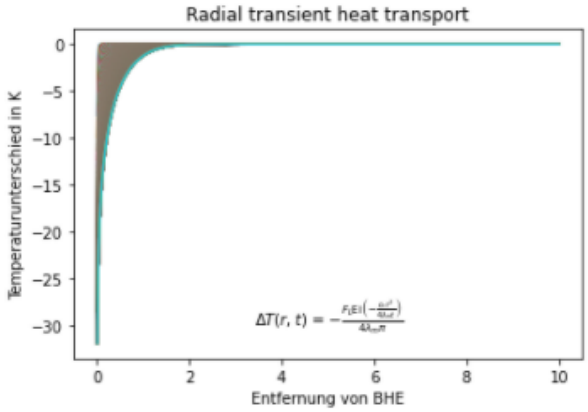
```
def make_model():
    """Create the model.
    """
    # pylint: disable=invalid-name, non-ascii-name
    param_list = []

    n = Parameter(
        'n',
        value=0.25,
        long_name='total porosity')
    param_list.append(n)

    rho_w = Parameter(
        'rho_w',
        value=1000,
        unit='kg/m^3',
        long_name='density',
        description='water density 1000 kg per m^3')
    param_list.append(rho_w)

    c_w = Parameter(
        'c_w',
        value=4186,
        unit='J/Kg/K',
        long_name='specific heat capacity',
        description='water specific heat capacity')
    param_list.append(c_w)

    rho_s = Parameter(
        'rho_s',
        value=2650,
        unit='kg/m^3',
        long_name='solid density'
        )
    param_list.append(rho_s)
```

```
c_s = Parameter(
    'c_s',
    value=880,
    unit='J/Kg/K',
    long_name='solid specific heat capacity',
    description='solid specific heat capacity')
param_list.append(c_s)

L = Parameter(
    'L',
    value=1,
    unit='m',
    long_name='length BHE',
    description='length of borehole heat exchanger')
param_list.append(L)

E = Parameter(
    'E',
    value=-50,
    unit='m',
    long_name='energy extraction',
    description='absolute energy extraction of BHE')
param_list.append(E)

lambda_m = Parameter(
    'lambda_m',
    value=1.115,
    unit='',
    long_name='heat conductivity ',
    description='effective thermal conductivity')
param_list.append(lambda_m)

# special

F_L = Parameter(
    'F_L',
    value=E.value / L.value,
    unit='',
    long_name='energy extraction PL',
    description='energy extraction PL')
param_list.append(F_L)

c_v = Parameter(
    'c_v',
    value=(
        n.value * rho_w.value * c_w.value +
        (1 -n.value) * rho_s.value * c_s.value
```

```
            ),
        unit='J/K/m^3',
        long_name='vol heat conductivity',
        description='volumetric heat capacity of water')
    param_list.append(c_v)


    return RadialHeat(
        param_list,
        model_name='Radial transient heat transport',
        left_side=r'$\Delta T(r, t)$')
```

[1]:
```
from ueflow.groundwater.analytic_bcs.bcs.transient_heat_radial import model
```

[2]:
```
model.test()
```




[3]:
```
my_model = model.make_model()
```

[4]:
```
my_model
```

[4]:
$$\Delta T(r,t) = -\frac{F_L \,\mathrm{Ei}\left(-\frac{c_v r^2}{4\lambda_m t}\right)}{4\lambda_m \pi}$$

[5]:
```
my_model.params.F_L
```

[5]:
**Parameter Energy Extraction PI $F_L$**

**Value**: -50.0

**Description**: energy extraction PL

[6]:
```
my_model.equation
```

[6]:
$$-\frac{F_L \,\mathrm{Ei}\left(-\frac{c_v r^2}{4\lambda_m t}\right)}{4\lambda_m \pi}$$

[7]:
```
my_model.calc_one({'r': 0.01, 't': 864})
```

[7]:
−7.55656066108061

[10]:
```
my_model.calc_array({'r': [0.01, 0.1, 1], 't': [864, 8640]}, sweep='t')
```

[10]:
```
array([[-7.55656066e+000, -3.09399657e-004, -4.26453256e-318],
       [-1.55449103e+001, -1.27128941e+000, -1.51309567e-033]])
```

Abbildung C.4.: Testsession im Jupyter Notebook - Instationärer radialer Wärmetransport

# Anhang D.

# Veröffentlichungen

## D.1. Posterbeitrag auf der Konferenz KGM-MODREG

| | |
|---|---|
| *Ort:* | Seggau, Steiermark, Österreich |
| *Datum:* | 20. November 2018 |
| *Autoren:* | Christian Engelmann; Mike Müller; Falk Händel; Martin Binder; Jannis Epting; Peter Huggenberger; Peter Börke |

### D.1.1. Titel

Entwicklung eines Simulationswerkzeuges für hydrogeologische und geothermische Fragestellungen unter Nutzung dynamischer, gekoppelter Randbedingungen - Konzeptionelle Aspekte und erste Ergebnisse

### D.1.2. Abstract

Grundwasser dient weltweit als Ressource für Trink-, Bewässerungs- und Prozesswasser sowie als Energieträger für geothermische Anlagen. Insbesondere in urbanen Räumen wird seit einiger Zeit eine deutliche thermische Beeinflussung dieser Ressource beobachtet. Neben negativen umweltrelevanten Konsequenzen führt dies unter Umständen zu Nutzungskonflikten zwischen konventionellen und geothermischen Nutzern sowie mehreren geothermischen Nutzern untereinander (z. B. mehrere Erdwärmesonden), was eine nachhaltige energiewirtschaftliche Nutzung von oberflächennaher Geothermie verhindert. Insbesondere auf regionaler Skala sind deshalb fundierte Kenntnisse über hydraulische und thermische Prozesse in flachen Grundwasserkörpern notwendig. Spezielle Fragestellungen umfassen hier, unter anderem, den wechselseitigen Wärmeaustausch zwischen Atmosphäre in Abhängigkeit von Landnutzung, Oberflächengewässern, urbanen Strukturen (z. B. Keller, Kanalisation) sowie geothermischen Anlagen (z. B. Erdwärmesonden) mit der Bodenzone bzw. den sich darunter anschließenden Grundwasserstockwerken. Nur eine dem Systemverständnis angepasste Berücksichtigung der genannten Prozesse wird einen wirtschaftlichen und

gleichzeitig möglichst umweltschonenden Betrieb von geothermischen Anlagen, auch nachhaltig, ermöglichen.

Grundvoraussetzung für ein solches nachhaltiges Management, welches neben thermischen Aspekten auch weitere qualitative sowie quantitative Parameter involviert, sollten daher geeignete Bewertungswerkzeuge darstellen. Die von derartigen Modellen abzubildenden Prozesse, speziell Randbedingungen des Wärmetransports, sind teilweise relativ komplex und benötigen ein hohes Maß an Kenntnissen bezüglich Prozessparametern. Hier stellen jedoch vereinfachte analytische Ansätze oftmals eine sinnvolle Alternative dar, da diese unter gewissen Systemzuständen Randbedingungen mit ähnlich hoher Genauigkeit, jedoch weitaus geringerem Komplexitätsgrad sowie Rechenaufwand abbilden können. Die Implementierung solcher Randbedingungen ist für viele existierende Softwarepakete oftmals nicht möglich. Falls doch, so sind dazu oft tiefgreifende Kenntnisse über den zumeist komplexen Aufbau sowie der jeweiligen Programmiersprache des jeweiligen Softwarepakets Voraussetzung. Derartige Erweiterungen sind im Consultingbereich meist nicht wirtschaftlich.

Im Rahmen des Forschungsprojektes „Entwicklung und Validierung eines modularen Simulationswerkzeuges für hydrogeologische und geothermische Fragestellungen mit einer interaktiven Schnittstelle zur direkten Implementierung dynamischer und rückkoppelnder Randbedingungen ModSimple“ wird derzeit eine modulare Open-Source-Modellumgebung entwickelt. Diese ermöglicht es auf effiziente und anwenderfreundliche Weise, anthropogene Einflüsse aus der geothermischen Nutzung auf die Umweltressource Grundwasser zu modellieren und zu prognostizieren. Die Entwicklung umfasst hierbei, neben der üblichen numerischen Abbildung von Strömungs- und Transportprozessen in gesättigten porösen Medien, die flexible Anbindung von Schnittstellen zur Kopplung des numerischen Modells mit analytischen Randbedingungen des Wärmetransports. Damit wird es für den Endanwender mit grundlegenden Programmierkenntnissen in der Sprache Python (PSF 2018) möglich sein, zusätzliche, für den einzelnen Fall relevante Prozesse zu implementieren. Die nötigen Fähigkeiten lassen sich typischer Weise in einem Zeitraum erlernen, die dem für die Einarbeitung in eine komplexe Modellierungssoftware entsprechen. Die Nutzung von Python ermöglicht es, Modelleingangsdatensätze flexibel zu modifizieren und Modellergebnisdaten effizient zu analysieren. Damit ergeben sich für viele Modellierungsaufgaben teilweise deutliche Effizienzgewinne gegenüber der Nutzung von traditionellen Nutzeroberflächen von Simulationsmodellen.

Grundlage für die sich in Entwicklung befindliche Modellumgebung ist *FiPy* (Guyer et al. 2009). Dieser auf der Finite-Volumen-Methode basierende Löser für partielle Differentialgleichungen bietet die Möglichkeit neue Strömungs- und Transportalgorithmen auf hohen Abstraktionsniveau zu definieren und numerisch zu lösen. *FiPy* ist in der Programmiersprache Python implementiert und nutzt für die numerische Lösung mehrere bestehende Gleichungslöser. Diese Löser selbst sind in Sprachen wie C, C++ und FORTRAN implementiert. Sie sind hoch-effizient und bieten unter anderem auch parallelisierte Algorithmen. Ein GPU-basierter Löser ist in Entwicklung. In der ersten Phase geht es darum die Funktionalität bereits existierender Simulationsprogramme umzusetzen. Als Orientierung dienen numerische Simulationsmodelle zur prozessbasierten Abbildung von gesättigter Strömung (MODFLOW-2005) sowie Stoff- und Wärmetransport (MT3DMS) unter Einbindung der Stofftransportprozesse Advektion, Dispersion, Diffusion und Sorption sowie der Wärmetransportprozesse Konvektion, Konduktion und Wärmespeicherung.

Die Umsetzung erfolgt mit der Programmiersprache Python. Die flexible Zuschaltung von Modulen erlaubt es die Funktionalität schrittweise zu erweitern. Ein wichtiges Modul ist die bidirektionale Kopplung von analytischen Randbedingungen des Wärmetransports (z. B. die eindimensionale Abbildung von Konvektion und Konduktion in mehreren Bodenschichten; Händel et al. 2013). Bisherige Arbeiten umfassen unter anderem die Entwicklung des Strömungsmodells. Eine Vielzahl an betrachteten Benchmarks zeigt eine im Vergleich zu existierenden Softwarepaketen vergleichbar hohe Recheneffizienz sowie Abbildungsgenauigkeit hydraulischer Prozesse. Darüber hinaus befindet sich derzeit eine Datenbank für analytische Wärmetransportrandbedingungen mit unterschiedlichen Komplexitätsgraden im Aufbau.

### D.1.3. Referenzen

Guyer, J. E., Wheeler, D., Warren, J. A. (2009). FiPy: Partial Differential Equations with Python. Computing in Science & Engineering 11(3) pp. 6—15 (2009), doi:10.1109/MCSE.2009.52, http://www.ctcms.nist.gov/fipy

Händel, F., Liedl, R., Fank, J., Rock, G. (2013). Regional modeling of geothermal energy systems in shallow aquifers: the Leibnitzer Feld case study (Austria). Environmental Earth Sciences 70(8): 3433-3446.

PSF (2018) Python Software Foundation. Python Language Reference, Version 3.6. Verfügbar unter http://www.python.org

## D.2. Posterbeitrag auf der Konferenz MODFLOW and More

| | |
|---|---|
| *Ort:* | Golden, Colorado, USA |
| *Datum:* | 4. Juni 2019 |
| *Autoren:* | Mike Müller; Christian Engelmann; Martin Binder; Falk Händel |

### D.2.1. Titel

Developing a MODFLOW-Compatible Groundwater Model with Dynamic Boundary Conditions

### D.2.2. Abstract

Setting up realistic boundary conditions is a major part of groundwater modeling. Typically, boundary conditions are specified for the whole model run. Sometimes boundary condition values depend on internal model conditions. For example, the flow rate of an infiltration well that re-infiltrates water is equal to the pumping rate of the extraction well. Such dynamic boundary conditions can be useful for groundwater treatment or geothermal applications.

MODFLOW is a well-established model that has proven useful over the last decades. The typical method to represent dynamic boundary conditions involves multiple model runs where subsequent runs use the results of previous runs as input.

The newly developed model system, *ueflow*, allows the user to implement dynamic boundary conditions by writing a Python-based plugin. This gives the user a lot of flexibility. For example, the previously described re-infiltration problem can be further modified by implementing other features such as energy costs for pumping, capacities of water treatment facilities, maintenance schedules for pumps based on pumping regimes, or other technical constrains.

This poster gives a short overview of *ueflow* that is based on the finite volume model framework *FiPy* (Guyer et al. 2009). *FiPy* is implemented in Python and offers multiple, high-performance solvers as well as several tools for generating grids and other input data. *ueflow* strives to be fully MODFLOW-compatible. Existing MODFLOW input data files can be read and seamlessly transformed into the *ueflow* input data format. First comparisons of simulation results generated by *ueflow* and MODLFOW runs show a good agreement.

### D.2.3. Referenzen

Guyer, J. E., Wheeler, D., Warren, J. A. (2009). FiPy: Partial Differential Equations with Python. Computing in Science & Engineering 11(3) pp. 6—15 (2009), doi:10.1109/MCSE.2009.52, http://www.ctcms.nist.gov/fipy

## D.3. Vortrag auf der GeoPython 2019

| | |
|---|---|
| *Ort:* | Basel, Schweiz |
| *Datum:* | 25. Juni 2019 |
| *Autoren:* | Mike Müller; Christian Engelmann; Martin Binder; Falk Händel |

### D.3.1. Titel

Modeling of Subsurface Flow and Transport with Dynamic Boundary Conditions

### D.3.2. Abstract

A newly developed model makes groundwater modeling customizable by applying dynamic boundary conditions. The user of the model can implement the behavior of such boundary conditions by writing Python plugins.

Boundary conditions are essential for groundwater models. Typically, there are three major types of boundary conditions (1) specified head (Dirichlet), (2) specified flow (Neumann), and (3) the combination of both former ones

into a boundary condition of flow with resistance (Cauchy). The user can specify values for these boundary conditions such as a well at a certain location with a given pumping rate for a specified duration. For some special applications, however, the specified values may further depend on internal model conditions. For example, the flow rate of an infiltration well that re-infiltrates water is equal to the pumping rate of the extraction well. This can be useful for geothermal applications within groundwater bodies. The newly developed model, *ueflow*, allows the user to implement such a scheme by writing a plugin. In addition to just using the pumping rate as infiltration rate, the user can incorporate other constrains such as energy costs for pumping, capacities of water treatment facilities, maintenance schedules for pumps based on pumping regimes, or other technical constrains.

The talk gives a short overview of *ueflow* that is based on the finite volume model framework *FiPy* (Guyer et al. 2009). *FiPy* is implemented in Python and offers multiple, high-performance solvers as well as several tools for generating grids and other input data. The next steps in developing *ueflow* involve adding a graphical user interface based on Python GIS libraries. GeoPython is a good place to connect to scientists having similar tasks, leading to a fruitful exchange and collaboration for this open-source project.

### D.3.3. Referenzen

Guyer, J. E., Wheeler, D., Warren, J. A. (2009). FiPy: Partial Differential Equations with Python. Computing in Science & Engineering 11(3) pp. 6—15 (2009), doi:10.1109/MCSE.2009.52, http://www.ctcms.nist.gov/fipy

## D.4. Posterbeitrag auf der EuroSciPy 2019

| | |
|---|---|
| *Ort:* | Bilbao, Spanien |
| *Datum:* | 4. September 2019 |
| *Autoren:* | Mike Müller; Christian Engelmann; Martin Binder; Falk Händel |

### D.4.1. Titel

From Modeler To Programmer

## D.4.2. Abstract

Boundary conditions are essential for groundwater models. The user can specify values for these boundary conditions such as a well at a certain location with a given pumping rate for a specified duration. For some special applications, however, the specified values may further depend on internal model conditions. For example, the flow rate of an infiltration well that re-infiltrates water is equal to the pumping rate of the extraction well. This can be useful for geothermal applications within groundwater bodies. The newly developed model, *ueflow*, allows the user to implement such a scheme by writing a plugin. In addition to just using the pumping rate as infiltration rate, the user can incorporate other constrains such as energy costs for pumping, capacities of water treatment facilities, maintenance schedules for pumps based on pumping regimes, or other technical constrains.

The poster gives a short overview of *ueflow* that is based on the finite volume model framework *FiPy* (Guyer et al. 2009). *FiPy* is implemented in Python and offers multiple, high-performance solvers as well as several tools for generating grids and other input data.

## D.4.3. Referenzen

Guyer, J. E., Wheeler, D., Warren, J. A. (2009). FiPy: Partial Differential Equations with Python. Computing in Science & Engineering 11(3) pp. 6—15 (2009), doi:10.1109/MCSE.2009.52, http://www.ctcms.nist.gov/fipy