Computers and Geosciences 109 (2017) 106–123

Contents lists available at ScienceDirect

Computers and Geosciences

journal homepage: www.elsevier.com/locate/cageo

Research paper

# pyGIMLi: An open-source library for modelling and inversion in geophysics

Carsten Rücker [a,*], Thomas Günther [b], Florian M. Wagner [c]

[a] Berlin University of Technology, Department of Applied Geophysics, Berlin, Germany
[b] Leibniz Institute for Applied Geophysics, Hannover, Germany
[c] University of Bonn, Steinmann Institute, Department of Geophysics, Bonn, Germany

A B S T R A C T

Many tasks in applied geosciences cannot be solved by single measurements, but require the integration of geophysical, geotechnical and hydrological methods. Numerical simulation techniques are essential both for planning and interpretation, as well as for the process understanding of modern geophysical methods. These trends encourage open, simple, and modern software architectures aiming at a uniform interface for interdisciplinary and flexible modelling and inversion approaches. We present pyGIMLi (Python Library for Inversion and Modelling in Geophysics), an open-source framework that provides tools for modelling and inversion of various geophysical but also hydrological methods. The modelling component supplies discretization management and the numerical basis for finite-element and finite-volume solvers in 1D, 2D and 3D on arbitrarily structured meshes. The generalized inversion framework solves the minimization problem with a Gauss-Newton algorithm for any physical forward operator and provides opportunities for uncertainty and resolution analyses. More general requirements, such as flexible regularization strategies, time-lapse processing and different sorts of coupling individual methods are provided independently of the actual methods used. The usage of pyGIMLi is first demonstrated by solving the steady-state heat equation, followed by a demonstration of more complex capabilities for the combination of different geophysical data sets. A fully coupled hydrogeophysical inversion of electrical resistivity tomography (ERT) data of a simulated tracer experiment is presented that allows to directly reconstruct the underlying hydraulic conductivity distribution of the aquifer. Another example demonstrates the improvement of jointly inverting ERT and ultrasonic data with respect to saturation by a new approach that incorporates petrophysical relations in the inversion. Potential applications of the presented framework are manifold and include time-lapse, constrained, joint, and coupled inversions of various geophysical and hydrological data sets.

## 1. Introduction

In modern geophysical applications, it is often desired to maximize information on the subsurface by a combination of different measurement methods. When dynamic changes are monitored, an additional link to corresponding process models (e.g., hydrogeological or geomechanical models) can lead to an improved process understanding and offers opportunities to estimate multi-physical parameters of the subsurface. Joint and process-based inversions are therefore thriving research topics in the emerging field of hydrogeophysics (e.g., Binley et al., 2015; Linde and Doetsch, 2016).

However, such endeavors are associated with considerable technical challenges. The required coupling of different numerical models represents a potential impediment for many practitioners and students. Even technically versatile users commonly end up building individually tailored solutions by linking various existing (and potentially commercial) software packages through scripts, which hinders the reproducibility of scientific findings (Peng, 2011). This motivates and supports the need for open, simple, and modern software architectures for the main numerical tasks in geophysics.

Uieda et al. (2013) present a software for geophysical data analysis with a focus on gravity and magnetic methods. Forward modelling routines are available to simulate gravitational and magnetic fields on various 2D and 3D meshes including tesseroids (spherical prisms). The inversion package enables non-linear parameter estimation with Levenberg-Marquardt, steepest-decent, as well as Gauss-Newton approaches. Hansen et al. (2013) provide a general inversion software for geophysical problems. Linear inverse Gaussian problems are solved using a least-squares solver, whereas general non-linear (i.e. non-Gaussian) inverse problems are solved with an extended Metropolis algorithm. While the software provides flexible inversion approaches for geophysical problems, its forward modelling functionality focuses on linear and non-linear travel time computations. Schaa et al. (2016) present a finite-element library for the solution of linear and non-linear, coupled, and time-dependent partial differential equations. The authors demonstrate the modelling capabilities based on 3D electrical resistivity and 2D magnetotelluric simulations. Cockett et al. (2015) present a software for simulation and gradient based parameter estimation in geophysics. Their approach is based on finite volume discretizations on structured and semi-structured meshes and includes convex optimization algorithms.

* Corresponding author.
*E-mail address:* carsten.ruecker@mailbox.org (C. Rücker).

http://dx.doi.org/10.1016/j.cageo.2017.07.011
Received 12 January 2017; Received in revised form 11 July 2017; Accepted 31 July 2017
Available online 8 August 2017

The authors state that joint and integrated inversions are generally possible by means of multiple misfit functions, physics, and regularization functionals.

The above-mentioned software packages are open-source, well-documented, and aim at extensibility and reproducibility of geophysical simulations. With a comparable motivation in mind, we present pyGIMLi (Geophysical Inversion and Modelling Library in Python), a versatile and computationally efficient framework for modelling and inversion in geophysical applications. In contrast to existing approaches, pyGIMLi explicitly targets the modern aspirations in geophysics including constrained, joint, and process-based inversions together with the required forward modelling necessities.

pyGIMLi has been in active development since 2009 and offers modular functionality accessible from different levels of abstraction aiming at satisfying the diverse needs in research and education. The Python programming language was chosen as the basis for pyGIMLi for its free, flexible, and cross-platform-compatible nature, which therefore makes it widely used in the (geo)scientific community (e.g., Guyer et al., 2009; Logg and Wells, 2010; Pérez et al., 2011; Wellmann et al., 2012; Uieda et al., 2013; Cockett et al., 2015; Weigand and Kemna, 2016; Hector and Hinderer, 2016; Schaa et al., 2016). One distinct advantage is that it can be easily extended by compiled modules from C or Fortran for example, allowing users to extend legacy code or outsource time-consuming parts in computationally efficient extensions. We make use of this flexibility and have implemented all runtime sensitive parts in a C++ core library. Complete Python bindings to this core library are complemented by functionality written in pure Python, thus offering both efficiency and flexibility for the rapid development of robust modelling and inversion applications. The modelling component offers mesh management as well as finite-element and finite-volume solvers in 1D, 2D and 3D. The inversion component is based on a deterministic Gauss-Newton algorithm and works with any physical forward operator provided. Several post-processing routines are provided to visualize results in 2D using Matplotlib (Hunter, 2007) and in 3D using the software ParaView (Ayachit, 2015) or Mayavi (Ramachandran and Varoquaux, 2011).

After an introduction of the software architecture including detailed descriptions of the different abstraction levels, the generalized inversion framework is presented. This is followed by a demonstration on how to perform basic modelling. We then emphasize the main purpose of pyGIMLi, i.e. readily integrating interdisciplinary data sets, by two applications: i) a fully coupled hydrogeophysical inversion that directly inverts for the hydraulic conductivity distribution of the aquifer and ii) a new petrophysical joint inversion based on electrical resistivity and travel time tomography to directly estimate water saturation.

## 2. Software design

The main tasks to solve with pyGIMLi are modelling and inversion. The modelling component is realized with a finite element/volume toolbox with all common cell shapes for linear and quadratic base functions in 1D, 2D, and 3D domains. The default inversion is implemented with a generalized Gauss-Newton approach with flexible regularization.

In addition to the core library, the extension part of pyGIMLi also consists of third-party dependencies that provide advanced functionality. We apply mesh generators like Triangle (Shewchuk, 1996), Tetgen (Si, 2015) and Gmsh (Geuzaine and Remacle, 2009), as well as high-level numerical solvers like SuiteSparse (Davis, 2006).

Fig. 1 shows the basic architecture of pyGIMLi with different abstraction levels written in Python. Based on the extension part, there are Python modules to exploit the easy-to-use scripting ability and to provide classes that are easier to maintain and to test. The equation level provides a general interface for the solution of common partial differential equations (PDE). The modelling level provides functionality of forward operators that aim for supporting specific geophysical methods.

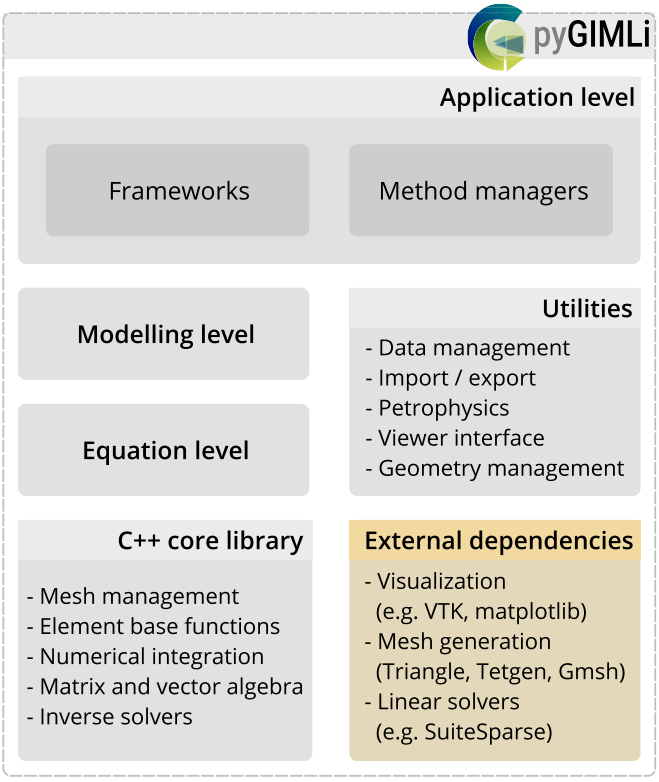



**Fig. 1.** Software architecture of pyGIMLi illustrating its components and different abstraction levels. The Python library has been built on top of a C++ core and several external dependencies (yellow box). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

In the application level, we define general frameworks to solve basic and advanced inversion tasks like time-lapse or joint inversion. The method managers combine all levels to solve the complete task for geophysical problems with specific data. All these abstraction levels communicate through unified interfaces and can thus be combined independently from the underlying geophysical method.

### 2.1. C++ core library

Python is a powerful scripting language for rapid development progress, but lacks runtime performance for pure Python code. It supports multiple paradigms that allow functional and object oriented programming and can be easily extended by pre-compiled high performance C++ extensions. We developed a C++ core library with a strong object oriented design for all runtime sensitive needs and provide full access to all parts of this core library by automatically generated Python bindings.

One main benefit of using Python is abstract prototyping, e.g., we implemented the main inverse solver in C++ with the use of an abstract base matrix and can use it with any custom advanced matrix directly from Python. This results in flexibility of the used matrix types with minimal coding effort and minimal runtime impairments. In addition to dense and different types of sparse matrices, there are specialized matrices like row or column-scaled matrices or Kronecker matrices. Furthermore, we implemented a block matrix containing references to a number of matrices of arbitrary type. It can be used for efficient constraint matrices or joint Jacobian matrices without performance loss.

For the high flexibility of regularization, we implemented a so called “region concept”, where one can define parts of the inversion domain and treat, configure, or couple these regions individually. An overview of the

background and different possibilities of incorporating information is given by Rücker (2011). Coscia et al. (2011) present an ERT inversion example, where different subsurface parts (geological layers, boreholes) have been decoupled and treated by different regularization operators.

The main performance drawback of the Python interpreted language is repeating code through nested loops due to the dynamic type conversion of the Python interpreter, which needs considerable time between two script instructions. A numerical package applying advanced geometry and mesh features spends significant runtime in mesh management, numerical integration, interpolation, or common tasks like nearest-neighbor search. Therefore, we implemented mesh management and related functionality as part of the C++ core library.

The majority of numerical approaches to the solution of partial differential equations (PDE) are based on a spatial discretization, called mesh or grid. A mesh combines $N$ nodes, $C$ cells, and $B$ boundaries and represents the modelling domain $\Omega = \cup_{i=1}^{C} \mathscr{C}_i$ and its outer boundary $\Gamma = \partial\Omega = \cup_{j=1}^{B} \mathscr{B}_j$. A mesh can be imported from external mesh generators or generated by provided utility functions.

We consider a grid to be a structured form of a mesh (e.g., a regular discretization into quadrangles or hexahedrons) and treat them equally. Nodes $\mathscr{N} = \{\mathscr{N}_i(\mathbf{r})\}$ with $i = 1 \ldots N$ represent $N$ discrete position vectors $\mathbf{r} \in \mathbb{R}^1$, $\mathbb{R}^2$, or $\mathbb{R}^3$. A cell $\mathscr{C}$ is a collection of nodes that spans the subdomain $\Omega_{\mathscr{C}} = \cup \mathscr{N}_j$ in the same dimensional space of the nodes. As cell shapes we implemented simplexes such as triangles, quadrangles in $\mathbb{R}^2$ and tetrahedrons, hexahedrons, or prisms in $\mathbb{R}^3$. Different cell types can be combined in one mesh, to combine structured discretization in the region of interest with progressively unstructured coarsening towards the boundaries for example. A boundary $\mathscr{B}$ is a collection of nodes spanning a subdomain $\Gamma_{\mathscr{C}} = \partial\Omega_{\mathscr{C}} = \cup \mathscr{N}_j$ representing the outer boundary of a cell. The boundary shapes arise as a result of the associated cell shape and are edges in $\mathbb{R}^2$ and triangles or quadrangles in $\mathbb{R}^3$. For all shapes we provide numerical integration rules for several base functions to assemble the numerical prerequisites for finite element and finite volume analyses.

### 2.2. Equation level

The equation level provides an interface to solve common PDEs on a given mesh, which comprises all geometric specifications, e.g., topography or known subsurface structures. Currently the equation level provides entry points for solving the following two main PDE types, which cover a wide range of methods in applied geophysics from potential fields to wave propagation.

The finite element method (FEM) with linear or quadratic basis functions solves

$$\begin{aligned} \frac{\partial u}{\partial t} &= \nabla\cdot(a\nabla u) + bu + f(\mathbf{r},t) \\ u(\mathbf{r},t) &= u_B \quad \mathbf{r} \in \Gamma_{\text{Dirichlet}} \\ \frac{\partial u(\mathbf{r},t)}{\partial \mathbf{n}} &= u_{\partial\text{B}} \quad \mathbf{r} \in \Gamma_{\text{Neumann}} \\ u(\mathbf{r},t=0) &= u_0 \quad \text{with} \quad \mathbf{r} \in \Omega \end{aligned} \tag{1}$$

for $u = u(\mathbf{r},t)$ with $\mathbf{r}$ be the node positions by calling:

```
u = pygimli.solver.solveFiniteElement(mesh, a, b, f, uB, duB, t, u0)
```

The finite volume method (FVM) with linear basis functions solves

$$\begin{aligned} \frac{\partial u}{\partial t} + \mathbf{v}\cdot\nabla u &= \nabla\cdot(a\nabla u) + bu + f(\mathbf{r},t) \\ u(\mathbf{r},t) &= u_B \quad \mathbf{r} \in \Gamma_{\text{Dirichlet}} \\ \frac{\partial u(\mathbf{r},t)}{\partial \mathbf{n}} &= u_{\partial\text{B}} \quad \mathbf{r} \in \Gamma_{\text{Neumann}} \\ u(\mathbf{r},t=0) &= u_0 \quad \text{with} \quad \mathbf{r} \in \Omega \end{aligned} \tag{2}$$

for $u = u(\mathbf{r},t)$ with $\mathbf{r}$ at the cell centers by calling:

```
u = pygimli.solver.solveFiniteVolume(mesh, a, b, v, f, uB, duB, t, u0)
```

The solution space $u = u(\mathbf{r},t)$ can be $\in \mathbb{R}|\mathbb{C}$ for $\mathbf{r} \in \mathbb{R}^1, \mathbb{R}^2, \mathbb{R}^3$. The coefficients $a, b, c, v, u_{\text{B}}, u_{\partial\text{B}}$ might be zero, scalars, arrays or `functions(N|C|B, t)` that are evaluated at runtime, e.g., they might be the solution of another solver run.

Note, that both solver functions are designed to give the user an easy as possible access to the modelling capabilities of the core library. However, most common elliptical, parabolic, and advection-type problems can be solved with these two functions, hyperbolic or curl-operator based systems can only be approached by nesting two parabolic equations. The accuracy of parabolic FVM is limited due to the linear base functions. Fortunately, highly specialized state-of-the art Python libraries exist (Guyer et al., 2009; Logg and Wells, 2010), which can be easily integrated if higher accuracy is needed.

### 2.3. Modelling level

The physics level represents a collection of classes to solve a forward or simulation task for a specific geophysical discipline by utilizing the equation level or applying suitable calculations. A forward operator (FOP) $\mathscr{F}(\mathbf{m}(\mathbf{r},t))$ maps a discrete parameter distribution $\mathbf{m} = \{m_j\}$ with $j = 1 \ldots M$ model parameters to the data vector $\mathbf{d} = \{d_i\}$ with $i = 1 \ldots N$ data. Each FOP is inherited from the base class `ModellingBase` implemented in the C++ core library and is accessed by the generalized interface:

```
class ForwardOperator(ModellingBase):

    ...

    def response(self, model):
        """Perform forward modelling d=f(m)."""

        ...

        return data

    def createJacobian(model):
        """Create Jacobian matrix for a given model."""

        ...

        # Code to fill the predefined Matrix


# typical uses are:

fop = ForwardOperator(...)

data = fop.response(model)

fop.createJacobian(model)

J = fop.jacobian()
```

The method `response` needs to be filled with the appropriate numerical calculation. Note that the spatial or temporal discretization required in inversion is usually different to the numerical needs for accurate forward modelling. Therefore, pyGIMLi provides inter- and extrapolation tools to map parameters from one mesh to another or assigning them to specific regions.

The method `createJacobian` is the interface for calculating the entries of the Jacobian matrix:

$$\mathbf{J}(\mathbf{m}) = \frac{\partial \mathscr{F}(\mathbf{m})}{\partial \mathbf{m}} = \{J_{i,j}\} \quad \text{with} \quad J_{i,j} = \frac{\partial d_i}{\partial m_j} \tag{3}$$

that might be needed for an inversion, but as it depends on the modelling problem, its part of the forward operator. It is beneficial to implement

**Table 1**
Currently implemented forward modelling algorithms in pyGIMLi.

| Method | Implementation | Dimension | Key references |
|---|---|---|---|
| Vertical Electrical soundings | Filter coefficients | 1D | Günther and Müller-Petke (2012) |
| Electrical Resistivity Tomography | FE | 2D/3D | Rücker et al. (2006); Günther et al. (2006) |
| (Spectral) Induced Polarization | FE (complex-valued) | 2D/3D | Günther et al. (2016) |
| Frequency-domain EM (FDEM) | Hankel transformation | 1D | Günther (2013) |
| Time-domain EM | FDEM | 1D | Costabel et al. (2017) |
| Magnetic resonance | FDEM | 1D/2D | Günther and Müller-Petke (2012); Dlugosch et al. (2014) |
| Magnetotellurics | Wait algorithm | 1D | |
| Seismic traveltime/refraction | shortest path or fast marching method | 2D | Heincke et al. (2010); Ronczka et al. (2017) |
| Gravimetry and magnetics | line integrals | 2D/3D | |
| Flow and transport | FE/FV | 2D/3D | subsection 3.2 |
| Streaming potential | FE | 2D/3D | |

`createJacobian` with a method-specific Jacobian generation approach. However, if this function is not implemented, the base class `ModellingBase` provides a parallelized default mechanism to fill the entries for **J** using a finite-difference (brute-force) approach, i.e., repeated forward calculations with perturbed model parameters.

The implemented forward operators are summarized in Table 1:

### 2.4. Application level

The highest abstraction layer is the application level, and therefore it represents the first entrance point for scientific development and end-user interaction. It contains method-independent frameworks and method-specific managers with embedded apps.

#### 2.4.1. Method managers and applications

The method managers represent the state of application that provides a full set of actions to proceed all tasks for a single discipline in applied geophysics and are considered entry points for end user interaction. These classes use a preconfigured instance of the generalized inversion framework and apply an appropriate forward operator with unified interfaces so that the methods `simulate` and `invert` are automatically available and controlled by keyword lists. Additionally, each method should provide the method-specific functions `loadData`, `showData` and `showResults`.

A typical use of a method manager to visualize and invert data for the example of ERT is:

```
ert = ERTManager()
ert.loadData('data.dat')
ert.showData()
ert.invert(**suitableOptions)
ert.showResults()
```

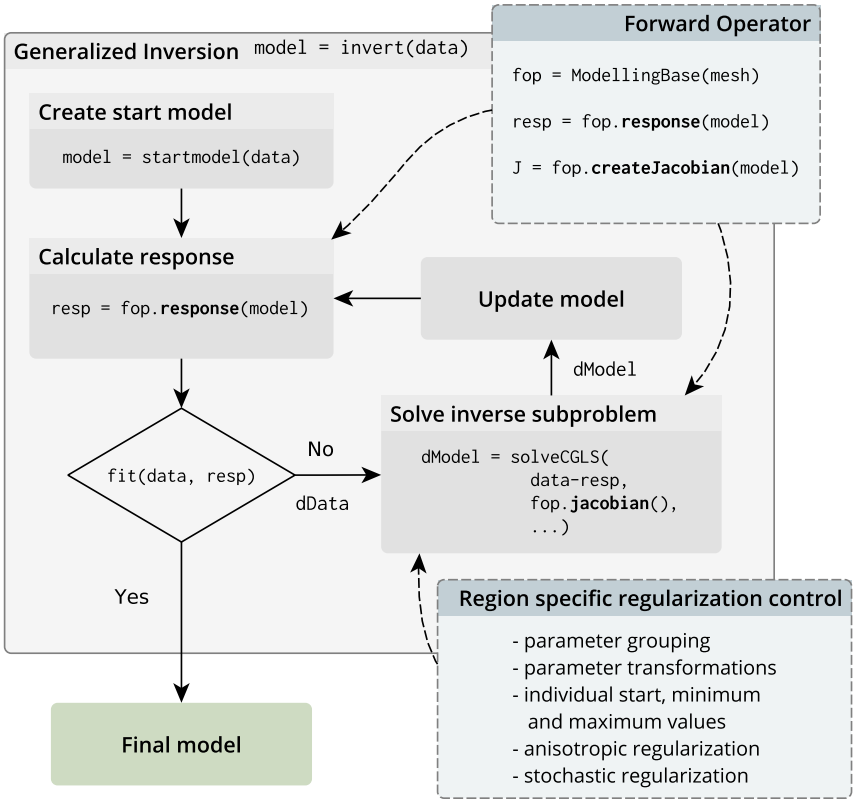



**Fig. 2.** Generalized inversion scheme. Already implemented (Table 1) or custom forward operators can be used that provide the problem specific response function and its Jacobian. Various strategies are available to regularize the inverse problem.

There are managers for practically all methods (Table 1) and combinations of them. Some are already including frameworks like LCI-type inversion (e.g., Costabel et al., 2016) or block-joint inversion (e.g., Günther and Müller-Petke, 2012). Moreover, there are managers involving simpler forward algorithms like fitting of complex conductivity/permittivity spectra (Loewer et al., 2016; Hupfer et al., 2016).

### 2.4.2. Inversion frameworks

Inversion frameworks are generalized, abstract approaches to solve a specific inversion problem without specifying the appropriate geophysical methods. This can be a specific regularization strategy, an alternative formulation of the inverse problem or algorithms of routine inversion. It is initialized by specific forward operators or managers that provide them. An example of how such an inversion framework is constructed for a petrophysical joint inversion is given in Appendix C.

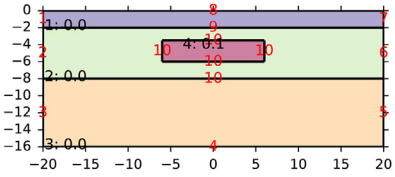
*2.4.2.1. Gauss-Newton inversion.* The default inversion framework is based on the generalized Gauss-Newton method and is compatible with any given forward operator and thus applicable to various physical problems. We state the inversion problem as minimization of an objective function consisting of data misfit and model constraints:

$$||\mathbf{W}_\mathrm{d}(\mathcal{F}(\mathbf{m}) - \mathbf{d})||_2^2 + \lambda||\mathbf{W}_\mathrm{m}(\mathbf{m} - \mathbf{m}_0)||_2^2 \rightarrow \min \quad (4)$$

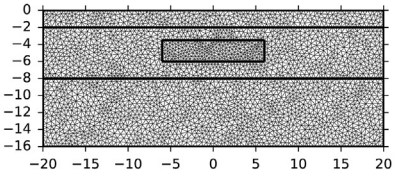
Note that we do not include inequality constraints in the minimization but use transformations to restrict parameters to reasonable ranges (e.g.,

```python
import pygimli as pg
import pygimli.meshtools as mt

# Create geometry definition for the modelling domain
world = mt.createWorld(start=[-20, 0], end=[20, -16],
                       layers=[-2, -8], worldMarker=False)
# Create a heterogeneous block
block = mt.createRectangle(start=[-6, -3.5], end=[6, -6],
                           marker=4, boundaryMarker=10)
# Merge geometrical entities
geom = mt.mergePLC([world, block])
pg.show(geom, boundaryMarker=True, savefig='geometry.pdf')
```




```python
# Create a mesh from the geometry definition
mesh = mt.createMesh(geom, quality=33, area=0.2)
pg.show(mesh, savefig='mesh.pdf')
```



```python
# Solve for T: ∇ · (a∇T) = 0 with T(bottom) = 1, T(top) = 0
T = pg.solver.solveFiniteElements(mesh, a=[[1, 1.0], [2, 2.0],
                                           [3, 3.0], [4, 0.1]],
                                  uB=[[8, 1.0], [4, 0.0]])
ax, _ = pg.show(mesh, data=T,
                label='Temperature $T$', cmap="hot_r")
pg.show(geom, ax=ax, fillRegion=True, savefig='T_field.pdf')
```

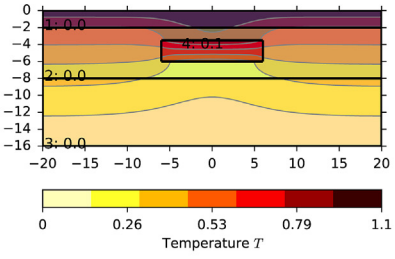



**Fig. 3.** Commented pyGIMLi code to simulate the steady heat equation.

Kim and Kim, 2011). $\mathbf{W}_d$ is the data weighting matrix containing the inverse data errors, $\mathbf{W}_m$ is the model constraint matrix (e.g., a first-order roughness operator), and $\mathbf{m}_0$ is a reference model. The dimensionless factor $\lambda$ scales the influence of the regularization term. There is a wide range of different regularization methods (different kinds of smoothness and damping, mixed operators, anisotropic smoothing) The existing ones can be used flexibly to constrain different model parameters or subsurface parts (regions), but also be extended by own functions. The application of the Gauss-Newton scheme on minimizing (4) yields the model update $\Delta\mathbf{m}^k$ in the $k^{\text{th}}$ iteration (Park and Van, 1991):

$$\begin{aligned}\left(\mathbf{J}^T\mathbf{W}_d^T\mathbf{W}_d\mathbf{J}+\lambda\mathbf{W}_m^T\mathbf{W}_m\right)\Delta\mathbf{m}^k &= \mathbf{J}^T\mathbf{W}_d^T\mathbf{W}_d\left(\Delta\mathbf{d}^k\right)\\ &\quad -\lambda\mathbf{W}_m^T\mathbf{W}_m\left(\mathbf{m}^k-\mathbf{m}^0\right)\\ \text{with}\quad \Delta\mathbf{d}^k=\mathbf{d}-\mathscr{F}\left(\mathbf{m}^k\right)\quad &\text{and}\quad \Delta\mathbf{m}^k=\mathbf{m}^k-\mathbf{m}^{k-1}\end{aligned} \tag{5}$$

which is solved using a conjugate-gradient least-squares solver (Günther et al., 2006). The inversion process including the region-specific regularization is sketched in Fig. 2.

All matrices of the inversion formulation can be directly accessed from Python and thereby offer opportunities for uncertainty and resolution analysis as well as experimental design (e.g., Wagner et al., 2015). Beyond different inversion approaches there are so-called frameworks for typical inversion (mostly regularization) tasks. Examples that are already implemented in pyGIMLi are for instance:

**Marquardt scheme** inversion of few independent parameters, e.g., fitting of spectra (Loewer et al., 2016)

**Soil-physical model reduction** incorporating soil-physical functions (Igel et al., 2016; Costabel and Günther, 2014)

**Classical joint inversion** of two data sets for the same parameter like DC and EM (Günther, 2013)

**Block joint inversion** of several 1D data using common layers, e.g., MRS+VES (Günther and Müller-Petke, 2012)

**Sequential (constrained) inversion** successive independent inversion of data sets, e.g., classic time-lapse inversion (e.g., Hübner et al., 2015)

**Simultaneous constrained inversion** of data sets of data neighbored in space (LCI, e.g., Costabel et al., 2016), time (full time-lapse) or frequency (Günther and Martin, 2016)

**Structurally coupled cooperative inversion** of disparate data based on structural similarity (e.g., Ronczka et al., 2017)

**Structure-based inversion** using layered 2D models (Attwa et al., 2014)

## 3. Exemplary applications

To demonstrate the usage of pyGIMLi, several examples are given along with the code. For the sake of brevity, all examples use minimalist 2D geometries, but are directly transferable to 3D and more complex geometries, if a corresponding mesh is provided.

### 3.1. Simulating heat transfer based on a simple geometry

A simplistic modelling example shows the basic steps for solving a steady-state heat equation on the equation level. Fig. 3 shows the complete Python source code and the resulting images for model creation and finite element calculation. In the preamble (Fig. 3, lines 1,2) the necessary pyGIMLi namespace and the pyGIMLi mesh generation package are imported and abbreviated with the alias names `pg` and `mt`, respectively.

We assume three layers with a block inside the second layer. pyGIMLi provides basic geometry building utilities accessible through the package alias `mt`. The commands `mt.createWorld` and `mt.createBlock` create the desired geometric entities that are combined by the command `mt.mergePLC` (Fig. 3, line 12). The resulting geometry definition, so called piecewise linear complex (PLC), contains nodes, boundary elements, and region descriptions to represent the entire model geometry. Fig. 3 line: 13 creates an image of the model geometry using the `pg.show` command, which is the most straightforward way to view meshes, geometries, and data.

As we need a mesh, we forward the given PLC to the external mesh generator called Triangle (Shewchuk, 1996). The `pg.show` command is used again to view the resulting mesh (Fig. 3 line: 15).

In the next step, we use `solveFiniteElement` from the equation level directly with the generated mesh to perform the FEM calculation. The arguments configure the requested PDE and control the underlying material parameters and boundary conditions. Most arguments are treated in a flexible manner, e.g., in this case `a` is a map that translates the four markers of the four geometry regions into a distribution of the thermal diffusivity $a$ to create a layered background of $a = [1, 2, 3]$ m$^2$/s with the heterogeneous block (region 4) to be $a = 0.1$ m$^2$/s. The boundary conditions are controlled with the `uB` argument, e.g., as of Dirichlet type with a fixed temperature $T = 1$ K for the boundary with marker 4 (bottom) and a fixed temperature $T = 0$ K at the boundary with marker 3 (surface). The other boundaries obtain natural Neumann (no-flow) boundary conditions by default.

The array for the resulting temperature distribution `T` can then be viewed by the calling `pg.show` again. The `pg.show` command can be

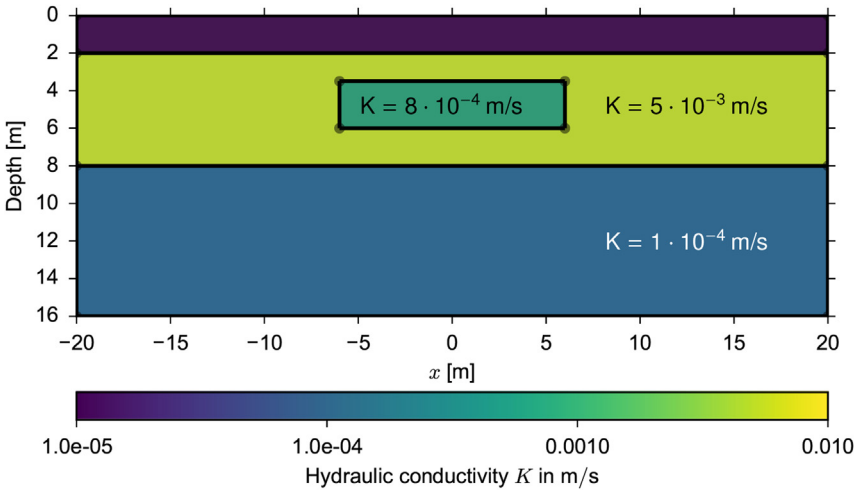



**Fig. 4.** Hydraulic conductivity distribution for the fully coupled joint inversion test case.

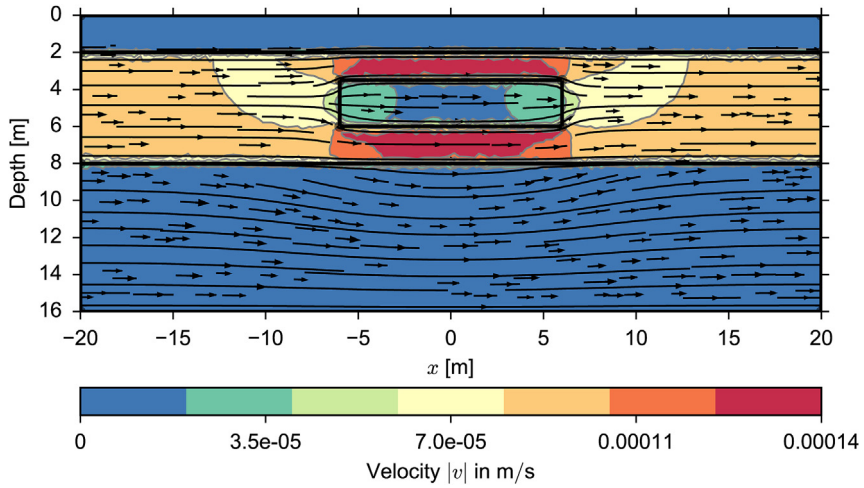



**Fig. 5.** Hydraulic velocity field for the given hydraulic conductivity as a result of a horizontal potential gradient.

also used to combine several plotting approaches to achieve the final result image of the given geometry with the estimated temperature distribution.

### 3.2. Fully coupled hydrogeophysical inversion

Monitoring of hydraulic processes with geophysical methods has become very popular. The classic way is to solve the geophysical problem, followed by later analysis to infer hydraulic properties. In recent years, a small number of approaches were presented that estimate hydraulic conductivity directly from geophysical observations (e.g., Pollock and Cirpka, 2010; Mboh et al., 2012; Camporese et al., 2015; Wagner, 2016). The small number of available approaches, however, mostly relies on customized and often proprietary software prohibiting reuse and advancement by other researchers. In the following, we show that coupled problems can be readily solved using pyGIMLi in a consistent and fully reproducible manner by a concatenation of hydraulic and geophysical simulations linked through petrophysical transformations.

We apply the geometry of the example from Sec. 3.1 and map hydraulic conductivities to the different regions of the model as shown in Fig. 4. The first layer of the model is considered a non-conducting topsoil followed by a conductive aquifer and a less conductive basement. The heterogeneous block inside the aquifer is assumed to represent a low conductive anomaly.

The fluid flow velocity in a porous medium of slow non-viscous and non-frictional hydraulic movement are governed by the Darcy equation (Whitaker, 1986):

$$K^{-1}\mathbf{v} + \nabla p = 0 \tag{6}$$

$$\nabla\cdot\mathbf{v} = 0 \tag{7}$$

leading to

$$\nabla\cdot(K\nabla p) = 0 \quad \text{on} \quad \Omega \tag{8}$$

with the hydraulic conductivity tensor $K$.

We can solve equation (8) on the equation level for a given hydraulic potential $p = p_0 = 0.75$ m on the left and $p = 0$ on the right boundary of the modelling domain, equaling a hydraulic gradient of 1.75%. The sought hydraulic velocity distribution can then be calculated as the gradient field of $\mathbf{v} = -\nabla p$. Extending the example with the following code fragment:

```
# Map regions to hydraulic conductivity in m/s
kMap = [[1, 1e-8], [2, 5e-3], [3, 1e-4], [4, 8e-4]]
# Map conductivity value per region to each given mesh cell
K = pg.solver.parseMapToCellArray(kMap, mesh)
# Dirichlet conditions for hydraulic potential
pBound = [[[1, 2, 3], 0.75], [[5, 6, 7], 0.0]]
# Solve for hydraulic potential
p = pg.solver.solveFiniteElements(mesh, a=K, uB=pBound)
# Solve velocity as gradient of hydraulic potential
vel = -pg.solver.grad(mesh, p) * np.asarray([K, K, K]).T
```

leads to the velocity distribution displayed in Fig. 5.

The flow direction is from left to right and exhibits an increased velocity in the aquifer due to its larger hydraulic conductivity. The anomaly in the aquifer considerably interferes the velocity field and causes the field to circumvent this rectangular body.

In the next step we use this velocity field to simulate the dynamic movement of a particle (e.g., salt) concentration $c(\mathbf{r}, t)$ in the aquifer by using the advection-diffusion equation (e.g., Bechtold et al., 2012)

$$\frac{\partial c}{\partial t} = \nabla\cdot(D\nabla c) - \mathbf{v}\cdot\nabla c + S \tag{9}$$

as a result of a source $S$. The molecular diffusion coefficient $D$ (in water $\approx 1\cdot10^{-9}$ m$^2$/s) is negligible. However, in porous media a diffusion-like spreading characteristics called dispersion takes place that is governed by the same term. We choose a common velocity-depending dispersion coefficient $D = \alpha|\mathbf{v}|$ with a dispersivity $\alpha = 1\cdot10^{-2}$ m (Bechtold et al., 2012). The particles are injected for six days at the position $\mathbf{r}_s(x = -19.1, y = -5.0)$ in the aquifer with an amount of $S = 0.001$ g/ls. The whole simulation time is 12 days in summary and we used 1600 time steps to fulfill the Courant-Friedrichs-Lewy condition (Courant et al., 1967) ensuring the particle movement does not exceed cell dimensions within one time step.

Solving equation (9) on the equation level with the finite volume solver results in a particle concentration $c(\mathbf{r}, t)$ (in g/l) for each cell center and time step. The following extension to the minimalist script:

```
# Fill injection source vector for a fixed injection position
sourceCell = mesh.findCell([-19.1, -4.6])
S[sourceCell.id()] = 1.0/sourceCell.size() #g/(l s)
# Choose 800 time steps for 6 days in seconds
t = pg.utils.grange(0, 6 * 24 * 3600, n=800)
# Create dispersitivity, depending on the absolute velocity
dispersion = pg.abs(vel) * 1e-2
# Solve for injection time, but we need velocities on cell nodes
vel = mt.cellDataToNodeData(mesh, vel)
c1 = pg.solver.solveFiniteVolume(mesh, a=dispersion, f=S, vel=vel,
                                 times=t, uB=[1, 0],
                                 scheme='PS', verbose=0)
# Solve without injection starting with last result
c2 = pg.solver.solveFiniteVolume(mesh, a=dispersion, f=0, vel=vel,
                                 u0=c1[-1], times=t, uB=[1, 0],
                                 scheme='PS', verbose=0)
# Stack results together
c = np.vstack((c1, c2))
```

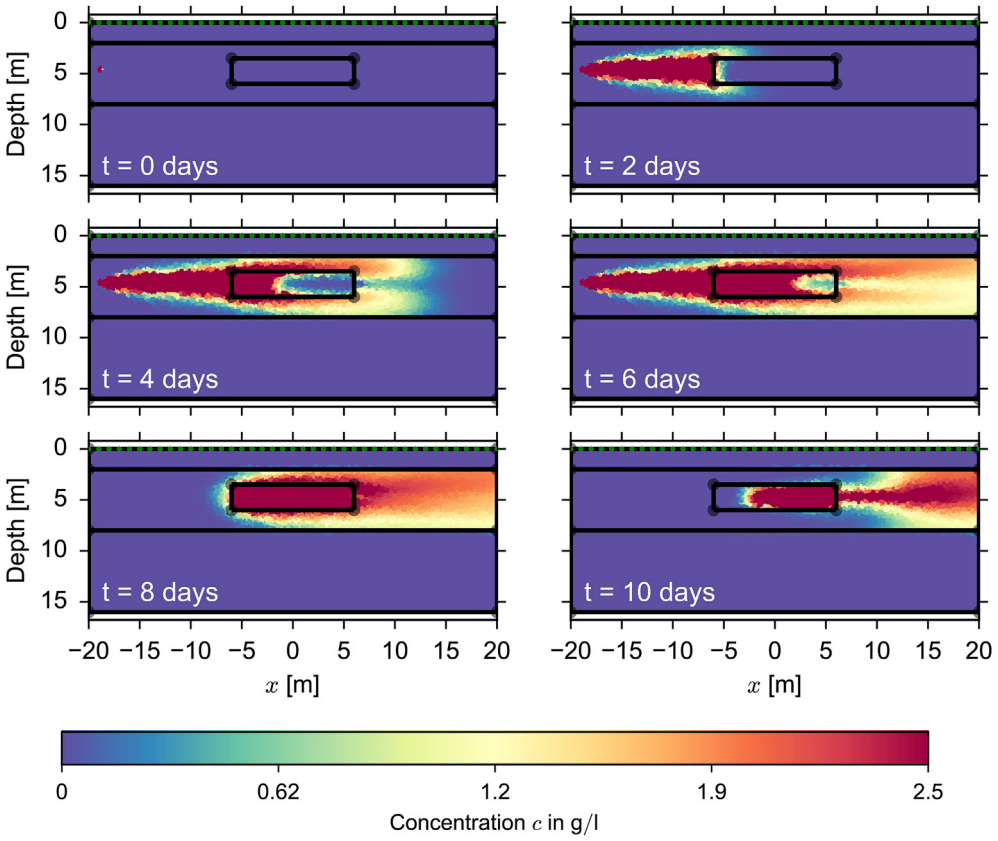



**Fig. 6.** Simulated salt concentration in the aquifer for selected time steps. Green dots mark the electrode positions. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

leads to the resulting temporal and spatial distribution of the concentration $c$ and is shown in Fig. 6 after 0, 2, 4, 6, 8, and 10 days.

We clearly see the particles spreading from the injection point and moving within the aquifer. The low conductive heterogeneous block disturbs the flow field and forces a large portion of the salt tracer to circumvent the obstacle. After 6 days the injection stops and the particle are washed out by the groundwater flow. The particles penetrate the heterogeneous block more slowly and remain longer due to the lower flow velocity.

If we assume a dominance of electrolytic conduction and associate the concentration with salt content, the tracer experiment can be well monitored by geoelectric measurements (Nguyen et al., 2009; Doetsch et al., 2012; Wagner et al., 2013). The fluid resistivity $\rho_f$ is obtained by a linear transformation:

$$\rho_f = \frac{1}{0.1 \cdot c + \sigma_0} \tag{10}$$

with a conductivity of groundwater $\sigma_0 = 0.1$ mS/cm and a conversion factor after Sulzbacher et al. (2012) based on in-situ data.

The Archie equation relates the bulk electrical resistivity of a medium $\rho$ to the fluid resistivity $\rho_f$ and saturation $S$ depending on porosity $\phi$ (Archie, 1942):

$$\rho = a\, \rho_f\, \phi^{-m} S^{-n} \tag{11}$$

The tortuosity factor $a$ and the saturation exponent $n$ are commonly assumed to be $a = 1$ and $n = 2$, respectively. The cementation exponent $m$ is assumed to equal 2 (standard for sandstones). We assume no fluid saturation in the top layer and full saturation in the aquifer with a porosity of $\phi = 0.3$.

To simulate synthetic data, i.e., apparent resistivities, we apply a dipole-dipole array with 41 equally spaced electrodes. Note that, compared to the Darcy simulation, ERT modelling requires specific boundary conditions and mesh refinement (Rücker et al., 2006). Therefore, we create a suitable ERT forward mesh and interpolate the bulk resistivity values. From the 1600 advection time frames, we select 10 to simulate the ERT data by applying the resistivity values and the measuring scheme to the ERT Manager and call the `simulate` interface. The minimalist example code can be extended by the following code snippet:

```
# Create survey measurement scheme
ertScheme = ert.createERTData(pg.utils.grange(-20, 20, dx=1.0),
                              schemeName='dd')
# Create suitable mesh for ert forward calculation
meshERT = mt.createParaMesh(ertScheme, quality=33,
                            paraMaxCellSize=0.2,
                            boundaryMaxCellSize=50)
# Select 10 time frame to simulate ERT data
timesERT = pg.IndexArray(np.floor(np.linspace(0, len(c)-1, 10)))
# Create conductivity of fluid for salt concentration c
sigmaFluid = c[timesERT] * 0.1 + 0.01
# Calculate bulk resistivity based on Archie's Law
# interpolate them to the ert mesh
resBulk = petro.resistivityArchie(rFluid=1. / sigmaFluid,
                                  porosity=0.3, m=1.3, mesh=mesh,
                                  meshI=meshERT, fill=1)
for i, rBulkI in enumerate(resBulk):
    resis[i] = 1. / ((1./rBulkI) + 1./rhoaBackground)
# Initialize ERT method manager
ERT = ERTManager(verbose=False)
# Run  simulation for  the apparent resistivities
rhoa = ERT.simulate(meshERT, resis, ertScheme, returnArray=True)
```

to create ERT dataset of apparent resistivities, which are shown in Fig. 7. The main characteristics of the tracer movement are clearly reflected by the measurement.

There are several approaches for inversion of time-lapse ERT data, like sequential inversion (e.g., Coscia et al., 2011), difference inversion (e.g., Bechtold et al., 2012), or 4-D inversion (Karaoulis et al., 2013). These approaches are available through pyGIMLi frameworks and result in a spatio-temporal resistivity distribution. Depending on temporal and spatial resolution, it might be possible to find out a single hydraulic conductivity value by analyzing breakthrough curves or similar techniques. However, in this case we go one step further and combine all simulation steps together into a sequentially coupled forward operator

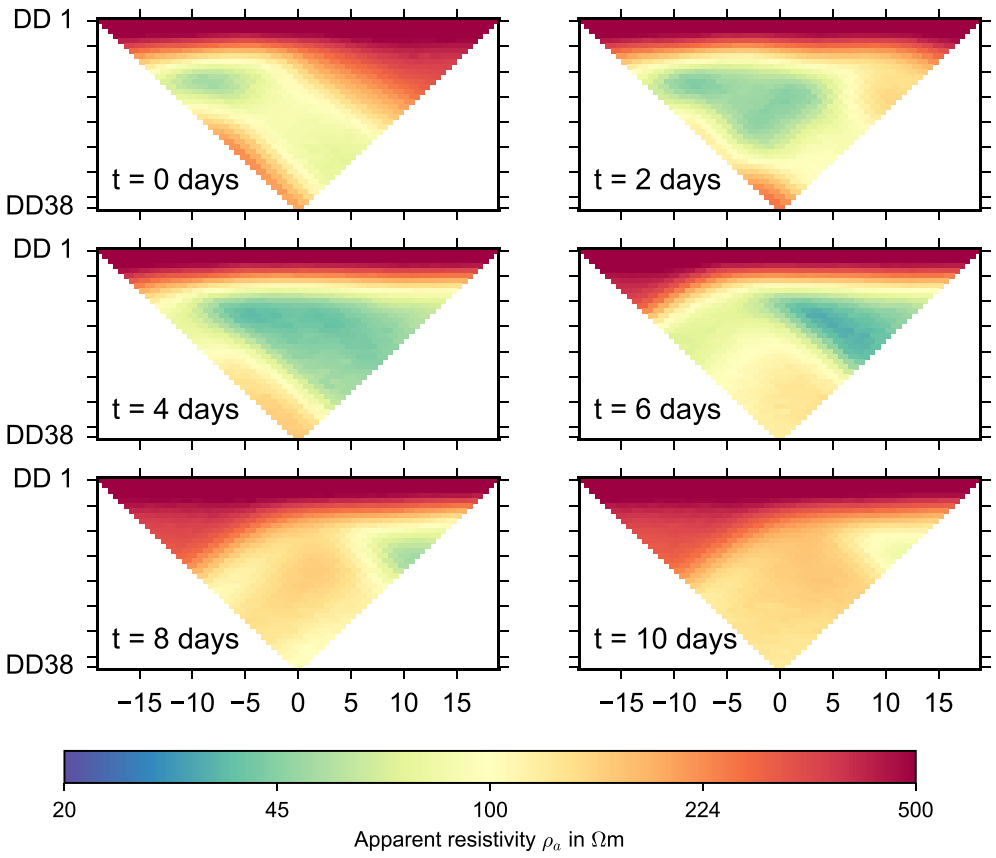



**Fig. 7.** Simulated dipole-dipole data of six consecutive measurements to monitor the migration of a saline tracer illustrated in Fig. 6.

that maps a distribution of hydraulic conductivities (Fig. 4) into a set of apparent resistivities (Fig. 7) for each timestep. As a result, we are able to invert for the hydraulic conductivity directly (Mboh et al., 2012).

The forward operator provides the necessary interface and can be used by the basic inversion framework to perform a fully coupled hydrogeophysical inversion from measured apparent resistivities to an image of hydraulic conductivity within the aquifer. As an efficient and optimized exact generation of the Jacobian matrix is not easily possible for an arbitrarily coupled system, a brute-force Jacobian calculation is performed by solving the forward problem for each unknown model cell. As this task can be easily parallelized, it is automatically done if multiple processors are detected. Furthermore, we limit the number of unknown parameters regarding the possible resolution of the inverse problem and to reduce the numerical effort. We apply the pyGIMLi region concept to introduce three regions, of which two are treated as fixed regions with known hydraulic conductivities on the top and at the bottom of the domain. The assumed aquifer is subdivided into 256 rectangular parameter regions and further into a fine triangular forward mesh above to ensure numerical accuracy (Fig. 8). The inversion starts with a homogeneous model and converged fitting the simulated ERT data within the assumed data error. The hydraulic conductivity distribution illustrated by Fig. 8 resembles the main characteristics of the synthetic model both structurally and concerning the mean values.

Note that the example serves as a proof of concept and is intended to

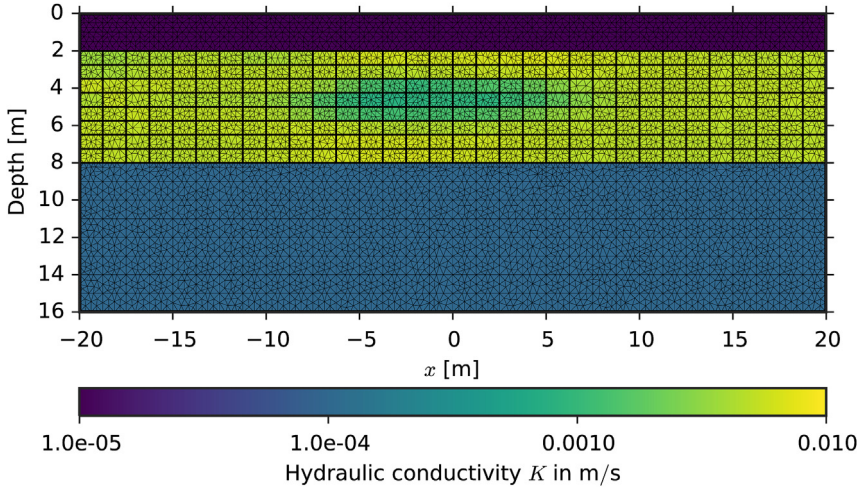



**Fig. 8.** Parameterization of forward (triangles) and inversion (rectangles) with inversion result of the fully coupled hydrogeophysical inversion.

demonstrate that complex workflows can be easily realized using the flexible modelling and inversion framework presented. Yet it should be noted that geophysics-based reconstruction of hydraulic conductivity is considerably challenged if the scenario becomes more realistic (e.g., 3D, unsaturated flow, temperature effects).

### 3.3. Petrophysical joint inversion

Joint inversion of different geophysical techniques helps to improve both resolution and interpretability of the resulting images. Different data sets can be directly coupled, if there is a link to an underlying target parameter. This is also possible if the relations are not known beforehand (Heincke et al., 2017). The following example assumes the relations as known and demonstrates the framework concept for a petrophysically coupled joint inversion of two geophysical methods in a few simple steps. We use the existing method managers for ERT (`pg.physics.ERTManager`) and travel-time tomography (TT) (`pg.physics.Refraction`), which both provide ready-to-use simulation and inversion capabilities.

Initially, we apply both method managers to create synthetic data sets. Fig. 9 shows the used mesh (`mMesh`) for a laboratory-scale circular model (e.g., a sandstone column) with constant porosity $\phi = 0.3$ and heterogeneous water saturation $S$, which is investigated using ERT and ultrasonic tomography.

To create geophysical parameter distributions, we apply common empirical petrophysical models, e.g., Archie's equation (eq. (11)) that provides the resistivity $\rho$ as a function of saturation $S$. Sonic velocity $v$ (or its reverse, the slowness $s$) as a function of porosity $\phi$ and saturation $S$ is given by the time-average equation (Wyllie et al., 1956):

$$s = \frac{1}{v} = (1-\phi)\frac{1}{v_m} + \phi S \frac{1}{v_w} + \phi(1-S)\frac{1}{v_a} \tag{12}$$

We assume a matrix velocity $v_m = 4000$ m/s, a water velocity $v_w = 1484$ m/s and an air velocity $v_a = 343$ m/s.

pyGIMLi contains a collection of mechanisms for forward- and inverse value mapping provided by `Trans` classes. They can be used for data and model scaling in the inversion process, e.g., by using logarithmic barriers (Kim and Kim, 2011) to restrict parameters between given ranges. Similarly, petrophysical relations can be easily defined and also nested with other transforms as range constraints. Assume the intrinsic parameter $p$ depends on the model parameter $m$. A transformation class needs to provide a forward and a backward transformation as well as the derivative of the transform, which is determined by a Newton algorithm if not specified.

| | | | |
|---|---|---|---|
| `Trans.fwd(S)` | $p(m)$ | $\rho = a\,\rho_f\,\phi^{-m}S^{-n}$ | (13) |
| `Trans.inv(rho)` | $m(p)$ | $S = \left(\frac{a\rho_f}{\rho\phi^m}\right)^{1/n}$ | (14) |
| `Trans.deriv(rho)` | $\frac{\partial p}{\partial m}$ | $\frac{\partial \rho}{\partial S} = -\frac{n\,a\,\rho_f}{\phi^m S^{n+1}}$ | (15) |

We apply these classes and create transformations for the mentioned petrophysical relations under the names `ArchieTrans` for ERT and `WyllieTrans` for TT, respectively. The resistivity `res` and velocity `vel` are generated for a given saturation by running the simple code:

```
ertT = ArchieTrans(rFluid=20, phi=0.3)
res = ertT.fwd(saturation)
ttT = WyllieTrans(vm=4000, phi=0.3)
vel = 1.0/ttT.fwd(saturation)
```

Fig. 10 shows the images for the resulting mapped $\rho$ (`res`) and $v$ (`vel`) values. Note that although we show the commonly used velocity, the slowness is the intrinsic linear parameter used inside.

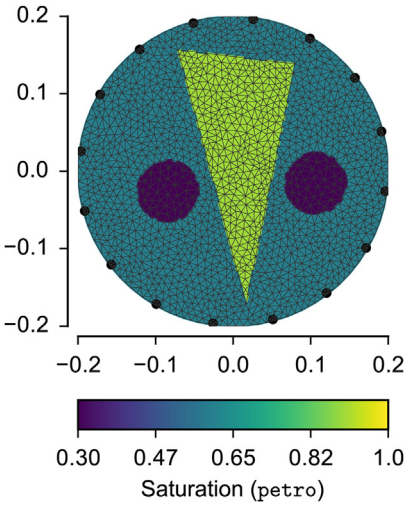



**Fig. 9.** Finite element mesh and assumed water saturation used for petrophysical inversion.

To create synthetic data sets, we assume 16 equally-spaced sensors on the circumferential boundary of the mesh. For the ERT modelling we build a complete dipole-dipole array. For the ultrasonic tomography we simulate the travel time for every possible sensor pair. The modelling itself is performed by calling the `simulate` interface command of the individual method managers, which create the data sets `ertData` for the resistivity and `ttData` for the velocity parameter distribution. Additionally, we add Gaussian noise with a standard deviation of 1% to the synthetic data sets.

To avoid an inverse crime, where identical meshes are used for both forward modelling and inversion (e.g. Colton and Kress, 1992), we create a new mesh for inversion, which holds no prior information on the anomalies to be reconstructed. Therefore, we create a second mesh `pMesh` representing the circular model domain that is coarser compared to the simulation mesh and contains no prior information on the anomalies. The simulated data are inverted on the new parameterization using the default inversion framework accessed through the interface command `invert` provided by both method managers and can be read as:

```
ertS = createData(sensors, schemeName='dd')
ERT = pg.physics.ERTManager()
ertData = ERT.simulate(mMesh, res, ertS, noisify=1)
resInv = ERT.invert(ertData, mesh=pMesh)

ttS = createRAData(sensors)
TT = pg.physics.Refraction()
ttData = TT.simulate(mMesh, vel, ttS, noisify=1)
velInv = TT.invert(ttData, mesh=pMesh)
```

Fig. 11 shows the resulting resistivity and velocity models. Generally, they recover the main structures of the synthetic model. The image of the resistivity distribution is more blurry but shows both the low and high resistive parts. In contrary, the velocity image shows sharper contrasts but lacks the low-velocity regions.

To reconstruct saturation values from the resulting images one can easily map the results back using the two transformation classes

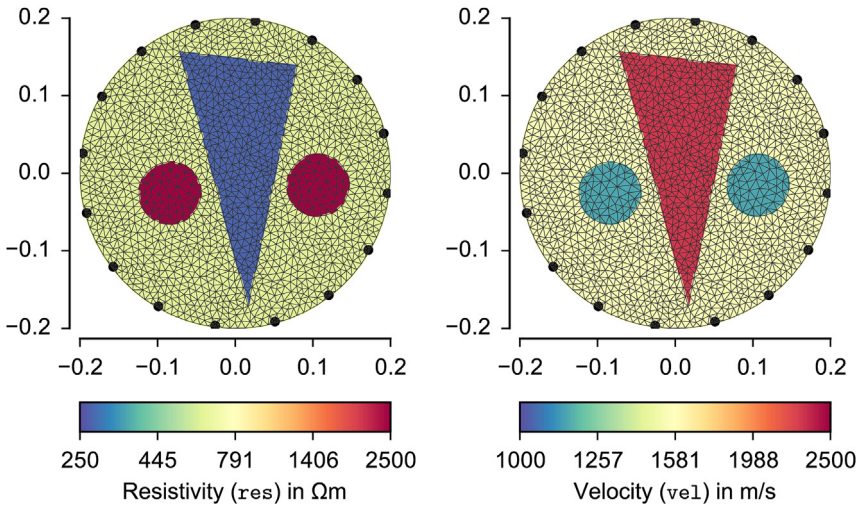



**Fig. 10.** Discretization of synthetic resistivity and velocity distribution.

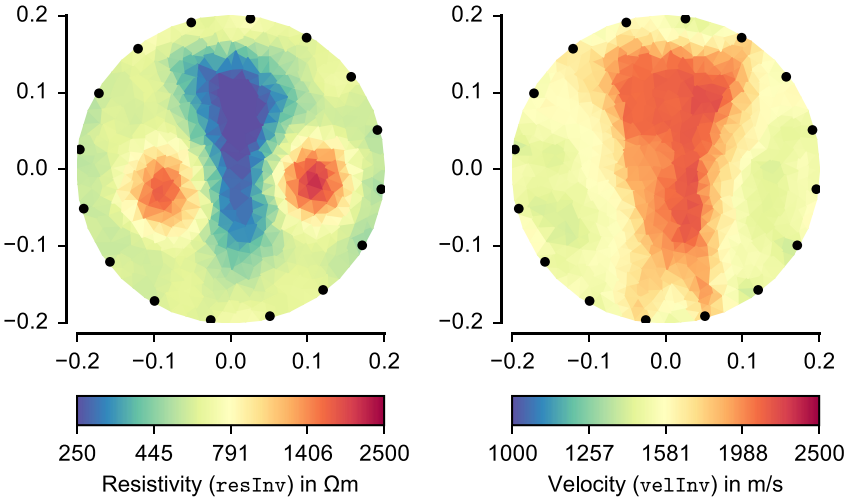



**Fig. 11.** Resulting models of simple inversion for single geophysical parameters.

discussed above. However, we rather want a petrophysically constrained inversion as a more sophisticated way to directly recover the saturation $\widehat{\mathbf{m}} = \{S\}$.

To calculate the forward response $\mathscr{F}(\mathbf{m})$ and the Jacobian $\mathbf{J}$ for a given saturation, we create a new forward operator called `PetroModelling`, which is method independent but initialized with preconfigured instances of a forward operator and a transformation class. It handles the necessary transformation steps and delegates the `response` and `createJacobian` commands appropriately. The forward response for the transformed problem is written as:

$$\widehat{\mathscr{F}}(\mathbf{m}) = \mathscr{F}(\mathbf{p}(\mathbf{m})) \quad \text{with} \quad \text{trans} : S \mapsto \rho \ (\text{ERT}) \quad \text{and} \quad \text{trans} : S \mapsto v \ (\text{TT}) \tag{16}$$

The Jacobian matrix for the transformed problem is obtained by multiplication with the inner derivative of the transformation:

$$\mathbf{J} = \frac{\partial \mathscr{F}(\mathbf{m})}{\partial \mathbf{m}} = \frac{\partial \mathscr{F}(\mathbf{p})}{\partial \mathbf{p}} \frac{\partial \mathbf{p}}{\partial \mathbf{m}} \tag{17}$$

For this we use a special matrix `MultRightMatrix` that holds an inner matrix (for which the Jacobian generation already exists) and a vector to be multiplied with from the right. The complete and concise implementation of the `PetroModelling` class is given in Appendix A and can be directly used with the generalized inversion framework.

As this is often used, an inversion framework `PetroInversion` for this type of petrophysical inversion is provided. It can be directly initialized by a method-specific manager and a transformation class. All relevant information is requested by the method managers provided and allows to construct a dedicated petrophysical forward operator with a corresponding inversion interface command (`invert`) and an additional argument `limits` for the expected model value ranges (log-transformation). The complete petrophysical inversion can then be written in two lines of code:

```
ertPet = PetroInversion(ERT, ertTrans)
satERT = ertPet.invert(ertData, mesh=pMesh, limits=[0, 1])

ttPet = PetroInversion(TT, ttTrans)
satTT = ttPet.invert(ttData, mesh=pMesh, limits=[0, 1])
```

Fig. 12 shows the resulting models for the petrophysically constrained inversion for saturation of both synthetic data sets.

The characteristics from both results is similar to the conventional

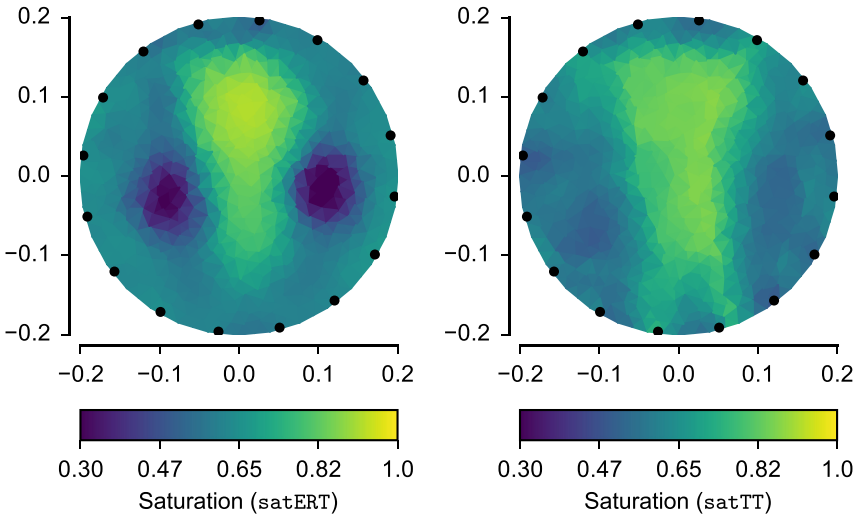



**Fig. 12.** Resulting models of the hydraulic saturation for single petrophysical inversion of ERT and travel-time data.

geophysical inversions, but the saturation models are automatically constrained to meaningful values between of 0 and 1.

To combine the inherent information of both data sets in the estimation of the same parameter, a general framework for joint inversions is provided in pyGIMLi. The core for this framework is a forward operator called `JointModelling`, which allows a simple stacking of $F$ different forward operators to create the appropriate model response and entries for the Jacobian matrix.

The model response is calculated by simply concatenating the responses of the individual operators. The Jacobian matrix for this type of concatenated problem can be efficiently formulated using a a special kind of matrix that we call `BlockMatrix`. This class allows arbitrary combinations of different matrix types that are directly used in the inverse solver. In this case (different data and different models) it has the shape of a block-Jacobi matrix with the individual Jacobian matrices on the diagonal. Block matrices is extremely helpful for a number of problems where independent models are simultaneously inverted, e.g., for LCI/SCI, time-lapse or spectral inversion (Günther and Martin, 2016). A minimal but complete implementation for the `JointModelling` class if given in Appendix B.

To keep the usage simple, we created a petrophysical joint inversion framework `JointPetroInversion` (see Appendix C), which combines the two forward operators `PetroModelling` and `JointModelling`. It is initialized by a set of method managers and corresponding petrophysical transforms. The framework manages the creation and delegation of the `JointModelling` forward operator, synchronizes data and error handling and provides the usual interface commands to perform the calculation via:

```
jointPet = JointPetroInversion([ERT, TT],
                               [ertTrans, ttTrans])
satJoint = jointPet.invert([ertData, ttData],
                           mesh=pMesh, limits=[0, 1])
```

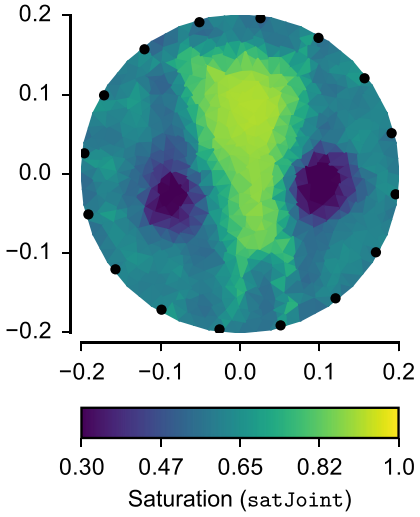



**Fig. 13.** Resulting model of the hydraulic saturation for the petrophysical joint inversion of simulated ERT and TT data.

Fig. 13 shows the resulting saturation image for the petrophysical joint inversion. The result of the joint inversion provides an improved image compared to both single inversion results as combines the benefits of both methods: It shows sharper contrasts than pure ERT and the regions of low and high saturation are more clearly visible compared to single TT.

In real life, the parameters of the petrophysical relations might not be known beforehand. However, it is straightforward to include a petrophysical parameter or even a distribution of it into the inversion process along with appropriate range constraints.

## 4. Conclusions

We have presented pyGIMLi, a versatile open-source framework for modelling and inversion in geophysics, which, due to its generalized and object-oriented design, is particularly useful to couple different measurement methods in joint or coupled inversions. This was demonstrated by a fully coupled inversion of time-lapse ERT data that allows to directly estimate the hydraulic conductivity distribution of the aquifer. The multiphysical forward operator provides the hydraulic and geoelectrical response of the tracer migration. It would thus be straightforward to also include hydrological data in the inversion to better constrain hydraulic aquifer properties.

A new petrophysical joint inversion approach was introduced and applied to ultrasonic and ERT combines the benefits of the two methods for a more accurate quantification of water saturation. In case of non-exactly known petrophysical parameters (e.g., porosity) this scheme could be easily extended, even without fixed relations. The availability of different forward operators and a generalized inversion framework

provides unprecedented means of mutual model constraints. For example, one could easily combine the joint inversion approach with the hydrogeophysical inversion so that a joint hydrogeophysical inversion for ERT data with hydrogeological (e.g. salinity or hydraulic head) data is achieved. All examples were generated with pyGIMLi version 1.0 and are fully reproducible using the scripts provided at http://cg17.pygimli.org.

Although the examples are based on simple 2D geometries for the sake of demonstration, we point out that the dimension of the problem is solely governed by the mesh provided, meaning that all code examples are directly transferable to more complex 3D geometries. The modular functionality is accessible from different abstraction levels and thereby offers entry points for a wide range of users with different degrees of programming experience:

**Application level** Data owners can easily analyze, invert and visualize their data using predefined method managers.

**Modelling level** Users with custom forward operators can readily setup a corresponding inversion workflow with flexible discretization and regularization controls.

**Equation level** Technically versatile practitioners can directly access the finite element and finite volume solvers to approach various physical problems.

There is a lot of potential for improving the package on all levels, starting from other PDE or element types over the integration of new methods to the extension of the inversion and regularization approaches. Particularly, we see benefits of combination with other open Python/C++ packages focusing on different methods.

To facilitate adoption and contribution of the geoscientific community, we have made the software package and all examples shown in this paper freely available under the permissive Apache 2.0 license. Corresponding downloads and a comprehensive documentation can be found on the project website http://www.pygimli.org. The source code is hosted on GitHub (https://github.com/gimli-org/gimli). By following modern software development principles such as unit testing and continuous integration, robustness and validity of existing and new functionality is ensured. It is anticipated that the presented software and ensuing developments will contribute to meet the modern data integration needs of researchers, practitioners, and students, particularly in, but not limited to, the hydrogeophysical community.

## Acknowledgments

The constructive reviews of Rowan Cockett and two anonymous reviewers have considerably improved the manuscript. We thank Jennifer Geacone-Cruz, Nico Skibbe, and Thomas Heinze for careful reading and valuable hints. We would also like to express our appreciation for the growing number of pyGIMLi users and their important feedback enabling a continuous improvement of the software.

## Appendices.

The following appendices contain the codes to carry out a petrophysical joint inversion as shown in 3.3. The two modelling operators for petrophysical modelling and joint modelling are used in the framework to carry out the inversion. Note that the implementations are independent on the actual methods and relations to be used.

### *Appendix A. Forward code for petrophysical inversion*

The following class is a simple forward operator, which yields the forward response of a target parameter (e.g., saturation) that is connected to an intrinsic parameter (e.g., resistivity) by a petrophysical relation (e.g., Archies law).

```
class PetroModelling(Modelling):
    """Combine relation m(p) with modelling f(p)."""
    def __init__(self, fop, petro):
        """Initialize with instance of forward operator
        and Trans., create a MultRightMatrix Jacobian."""
        Modelling.__init__(self)
        self.fop = fop
        self.petro = petro
        self.jac = pg.MultRightMatrix(self.fop.jacobian())
        self.setJacobian(self.jac)
    def response(self, model):
        """Transform and compute response f(p(m))."""
        tModel = self.petro.fwd(model)
        return self.fop.response(tModel)
    def createJacobian(self, model):
        """Jacobian with inner derivative of the trans."""
        self.fop.createJacobian(self.petro.fwd(model))
        self.jac.r = self.petro.deriv(model)
```

*Appendix B. Forward code for joint inversion*

This modelling class can be used to combine several forward operators that use the same model parameter (e.g., DC and EM using resistivity). The responses are just concatenated and the Jacobian is a block matrix consisting of the individual ones.

```
class JointModelling(Modelling):
    """Cumulative (joint) forward operator."""
    def __init__(self, fopList):
        """Initialize with lists of forward operators.
        Create a BlockMatrix Jacobian."""
        Modelling.__init__(self)
        self.fops = fopList
        self.jac = pg.BlockMatrix()
    def response(self, model):
        """Concatenate responses for all fops."""
        resp = []
        for f in self.fops:
            resp.extend(f.response(model))
        return resp
    def createJacobian(self, model):
        """Fill the individual Jacobian matrices."""
        for f in self.fops:
            f.createJacobian(model)
    def setData(self, data):
        nData = 0 # start on top
        for i, fi in enumerate(self.fops):
            fi.setData(data[i])
            self.jac.addMatrix(fi.jacobian(), nData, 0)
            nData += data[i].size()
        self.setJacobian(self.jac)
```

*Appendix C. Inversion framework for petrophysical joint inversion*

The inversion framework combines the two modelling classes and carries out a petrophysical joint inversion.

```
class JointPetroInversion(MeshInversion):
    """Framework combining joint and petro inversion."""
    def __init__(self, mgrs, petros):
        """Initialize with lists of Managers and Trans."""
        MeshInversion.__init__(self)
        self.mgrs = mgrs

        self.fops = [PetroModelling(m.createFOP(), p) \
                     for m, p in zip(mgrs, petros)]

        self.tM = mgrs[0].tM
        self.tD = pg.TransCumulative()
        self.fop = JointModelling(self.fops)
        self.setForwardOperator(self.fop)

    def setData(self, data):
        """Set all data and let the Framework manage data
        transformation and error handling"""
        self.fop.setData(data)
        self.dataVals = pg.Vector(0)
        self.dataErrs = pg.Vector(0)

        for i, mgr in enumerate(self.mgrs):
            self.tD.add(mgr.tD, data[i].size())
            self.dataVals = pg.cat(self.dataVals,
                                   mgr.dataVals(data[i]))
            self.dataErrs = pg.cat(self.dataErrs,
                                   mgr.relErrorVals(data[i]))
        self.inv.setTransData(self.tD)

    def invert(self, data, **kwargs):
        limits = kwargs.pop('limits', [0., 1.])
        self.tM.setLowerBound(limits[0])
        self.tM.setUpperBound(limits[1])
        self.inv.setTransModel(self.tM)
        kwargs['startModel'] = (limits[1]-limits[0])/2.
        return MeshInversion.invert(self, data, **kwargs)
```

## References


Archie, G., 1942. The electrical resistivity log as an aid in determining some reservoir characteristics. Trans. AIME 146, 54–62. http://dx.doi.org/10.2118/942054-g.

Attwa, M., Akca, I., Basokur, A.T., Günther, T., 2014. Structure-based geoelectrical models derived from genetic algorithms: a case study for hydrogeological investigations along Elbe River coastal area, Germany. J. Appl. Geophys. 103, 57–70. http://dx.doi.org/10.1016/j.jappgeo.2014.01.006.

Ayachit, U., 2015. The ParaView Guide: A Parallel Visualization Application. Kitware, Inc., USA.

Bechtold, M., Vanderborght, J., Weihermüller, L., Herbst, M., Günther, T., Ippisch, O., Kasteel, R., Vereecken, H., 2012. Upward transport in a three-dimensional

heterogeneous laboratory soil under evaporation conditions. Vadose Zone J. 11 http://dx.doi.org/10.2136/vzj2011.0066.
Binley, A., Hubbard, S.S., Huisman, J.A., Revil, A., Robinson, D.A., Singha, K., Slater, L.D., 2015. The emergence of hydrogeophysics for improved understanding of subsurface processes over multiple scales. Water Resour. Res. 51, 3837–3866. http://dx.doi.org/10.1002/2015WR017016.
Camporese, M., Cassiani, G., Deiana, R., Salandin, P., Binley, A., 2015. Coupled and uncoupled hydrogeophysical inversions using ensemble Kalman filter assimilation of ERT-monitored tracer test data. Water Resour. Res. 51, 3277–3291. http://dx.doi.org/10.1002/2014wr016017.
Cockett, R., Kang, S., Heagy, L.J., Pidlisecky, A., Oldenburg, D.W., 2015. SIMPEG: an open source framework for simulation and gradient based parameter estimation in geophysical applications. Comput. Geosci. 85, 142–154. http://dx.doi.org/10.1016/j.cageo.2015.09.015.
Coscia, I., Greenhalgh, S., Linde, N., Doetsch, J., Marescot, L., Günther, T., Green, A., 2011. 3D crosshole apparent resistivity static inversion and monitoring of a coupled river-aquifer system. Geophysics 76, G49–G59. http://dx.doi.org/10.1190/1.3553003.
Costabel, S., Günther, T., 2014. Noninvasive estimation of water retention parameters by observing the capillary fringe with magnetic resonance sounding. Vadose Zone J. 13 http://dx.doi.org/10.2136/vzj2013.09.0163.
Costabel, S., Günther, T., Dlugosch, R., Müller-Petke, M., 2016. Torus-nuclear magnetic resonance: quasicontinuous airborne magnetic resonance profiling by using a helium-filled balloon. Geophysics 81, WB119–WB129. http://dx.doi.org/10.1190/geo2015-0467.1.
Costabel, S., Siemon, B., Houben, G., Günther, T., 2017. Geophysical investigation of a freshwater lens on the island of Langeoog, Germany – insights from combined HEM, TEM and MRS data. J. Appl. Geophys. 136, 231–245. http://dx.doi.org/10.1016/j.jappgeo.2016.11.007.
Colton, D.L., Kress, R., 1992. Integral Equation Methods in Scattering Theory. Springer. http://dx.doi.org/10.1007/978-3-662-02835-3.
Courant, R., Friedrichs, K., Lewy, H., 1967. On the partial difference equations of mathematical physics. IBM J. Res. Dev. 11, 215–234. http://dx.doi.org/10.1147/rd.112.0215.
Davis, T.A., 2006. Direct Methods for Sparse Linear Systems. SIAM.
Dlugosch, R., Günther, T., Müller-Petke, M., Yaramanci, U., 2014. Two-dimensional distribution of relaxation time and water content from surface nuclear magnetic resonance. Near Surf. Geophys. 12, 231–241. http://dx.doi.org/10.3997/1873-0604.2013062.
Doetsch, J., Linde, N., Vogt, T., Binley, A., Green, A.G., 2012. Imaging and quantifying salt-tracer transport in a riparian groundwater system by means of 3D ERT monitoring. Geophysics 77, B207–B218. http://dx.doi.org/10.1190/geo2012-0046.1.
Geuzaine, C., Remacle, J.F., 2009. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. Int. J. Numer. Methods Eng. 79, 1309–1331. http://dx.doi.org/10.1002/nme.2579.
Guyer, J.E., Wheeler, D., Warren, J.A., 2009. FiPy: partial differential equations with Python. Comput. Sci. Eng. 11, 6–15. http://dx.doi.org/10.1109/MCSE.2009.52.
Günther, T., 2013. On inversion of frequency domain electromagnetic data in salt water problems - sensitivity and resolution. In: Ext. Abstr., 19th European Meeting of Environmental and Engineering Geophysics, Bochum, Germany. http://dx.doi.org/10.3997/2214-4609.20131387.
Günther, T., Martin, T., 2016. Spectral two-dimensional inversion of frequency-domain induced polarisation data from a mining slag heap. J. Appl. Geophys. 135, 436–448. http://dx.doi.org/10.1016/j.jappgeo.2016.01.008.
Günther, T., Müller-Petke, M., 2012. Hydraulic properties at the North Sea island of Borkum derived from joint inversion of magnetic resonance and electrical resistivity soundings. Hydrol. Earth Syst. Sci. 16, 3279–3291. http://dx.doi.org/10.5194/hess-16-3279-2012.
Günther, T., Rücker, C., Spitzer, K., 2006. Three-dimensional modelling and inversion of dc resistivity data incorporating topography - Part II: Inversion. Geophys. J. Int. 166, 506–517. http://dx.doi.org/10.1111/j.1365-246X.2006.03011.x.
Günther, T., Martin, T., Rücker, C., 2016. Spectral Inversion of SIP field data using pyGIMLi/BERT. In: Ext. Abstract, 4th International Workshop on Induced Polarization, Aarhus, Denmark.
Hansen, T.M., Cordua, K.S., Looms, M.C., Mosegaard, K., 2013. SIPPI: a Matlab toolbox for sampling the solution to inverse problems with complex prior information: Part 1 – Methodology. Comput. Geosci. 52, 470–480. http://dx.doi.org/10.1016/j.cageo.2012.09.004.
Hector, B., Hinderer, J., 2016. pygrav, a python-based program for handling and processing relative gravity data. Comput. Geosci. 91, 90–97. http://dx.doi.org/10.1016/j.cageo.2016.03.010.
Heincke, B., Günther, T., Dahlsegg, E., Rønning, J.S., Ganerød, G., Elvebakk, H., 2010. Combined three-dimensional electric and seismic tomography study on the Åknes rockslide in western Norway. J. Appl. Geophys. 70, 292–306. http://dx.doi.org/10.1016/j.jappgeo.2009.12.004.
Heincke, B., Jegen, M., Moorkamp, M., Hobbs, R.W., Chen, J., 2017. An adaptive coupling strategy for joint inversions that use petrophysical information as constraints. J. Appl. Geophys. 136, 279–297. http://dx.doi.org/10.1016/j.jappgeo.2016.10.028.
Hunter, J., 2007. Matplotlib: a 2D graphics environment. Comput. Sci. Eng. 9, 90–95. http://dx.doi.org/10.1109/MCSE.2007.55.
Hupfer, S., Martin, T., Weller, A., Günther, T., Kuhn, K., Ngninjio, V.D.N., Noell, U., 2016. Polarization effects of unconsolidated sulphide-sand-mixtures. J. Appl. Geophys. 135, 456–465. http://dx.doi.org/10.1016/j.jappgeo.2015.12.003.
Hübner, R., Heller, K., Günther, T., Kleber, A., 2015. Monitoring hillslope moisture dynamics with surface ert for enhancing spatial significance of hydrometric point measurements. Hydrol. Earth Syst. Sci. 19, 225–240. http://dx.doi.org/10.5194/hess-19-225-2015.
Igel, J., Stadler, S., Günther, T., 2016. High-resolution investigation of the capillary transition zone and its influence on GPR signatures. In: Ext. Abstr., SAGEEP, Denver, USA. http://dx.doi.org/10.4133/SAGEEP.29-046.
Karaoulis, M., Revil, A., Tsourlos, P., Werkema, D.D., Minsley, B.J., 2013. IP4DI: a software for time-lapse 2D/3D DC-resistivity and induced polarization tomography. Near Surf. Geophys. 54, 164–170. http://dx.doi.org/10.3997/1873-0604.2013004.
Kim, H.J., Kim, Y.H., 2011. A unified transformation function for lower and upper bounding constraints on model parameters in electrical and electromagnetic inversion. J. Geophys. Eng. 8, 21–26. http://dx.doi.org/10.1088/1742-2132/8/1/004.
Linde, N., Doetsch, J., 2016. Joint inversion in hydrogeophysics and near-surface geophysics. In: Moorkamp, M., Leliévre, P.G., Linde, N., Khan, A. (Eds.), Integrated Imaging of the Earth: Theory and Applications. John Wiley & Sons, Inc, pp. 117–135. http://dx.doi.org/10.1002/9781118929063.ch7.
Loewer, M., Igel, J., Wagner, N., 2016. Spectral decomposition of soil electrical and dielectric losses and prediction of in situ GPR performance. IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens. 9, 212–230. http://dx.doi.org/10.1109/jstars.2015.2424152.
Logg, A., Wells, N., 2010. DOLFIN. ACM Trans. Math. Softw. 37, 1–28. http://dx.doi.org/10.1145/1731022.1731030.
Mboh, C., Huisman, J., Gaelen, N.V., Rings, J., Vereecken, H., 2012. Coupled hydrogeophysical inversion of electrical resistances and inflow measurements for topsoil hydraulic properties under constant head infiltration. Near. Surf. Geophys. 10, 413–426. http://dx.doi.org/10.3997/1873-0604.2012009.
Nguyen, F., Kemna, A., Antonsson, A., Engesgaard, P., Kuras, O., Ogilvy, R., Gisbert, J., Jorreto, S., Puildo-Bosch, A., 2009. Characterization of seawater intrusion using 2D electrical imaging. Near Surf. Geophys. 7, 377–390. http://dx.doi.org/10.3997/1873-0604.2009025.
Park, S.K., Van, G.P., 1991. Inversion of pole-pole data for 3-d resistivity structure beneath arrays of electrodes. Geophysics 56, 951–960. http://dx.doi.org/10.1190/1.1443128.
Peng, R.D., 2011. Reproducible research in computational science. Science 334, 1226–1227. http://dx.doi.org/10.1126/science.1213847.
Pérez, F., Granger, B.E., Hunter, J.D., 2011. Python: an ecosystem for scientific computing. Comput. Sci. Eng. 13, 13–21. http://dx.doi.org/10.1109/MCSE.2010.119.
Pollock, D., Cirpka, O.A., 2010. Fully coupled hydrogeophysical inversion of synthetic salt tracer experiments. Water Resour. Res. 46 http://dx.doi.org/10.1029/2009wr008575.
Ramachandran, P., Varoquaux, G., 2011. Mayavi: 3D visualization of scientific data. Comput. Sci. Eng. 13, 40–51. http://dx.doi.org/10.1109/MCSE.2011.35.
Ronczka, M., Hellman, K., Günther, T., Wisen, R., Dahlin, T., 2017. Electric resistivity and seismic refraction tomography: a challenging joint underwater survey at Äspö Hard Rock Laboratory. Solid Earth 13, 671–682. http://dx.doi.org/10.5194/se-8-671-2017.
Rücker, C., 2011. Advanced Electrical Resistivity Modelling and Inversion Using Unstructured Discretization. Ph.D. thesis. University of Leipzig. http://nbn-resolving.de/urn:nbn:de:bsz:15-qucosa-69066.
Rücker, C., Günther, T., Spitzer, K., 2006. Three-dimensional modelling and inversion of dc resistivity data incorporating topography – Part I: Modelling. Geophys. J. Int. 166, 495–505. http://dx.doi.org/10.1111/j.1365-246X.2006.03010.x.
Schaa, R., Gross, L., Du Plessis, J., 2016. PDE-based geophysical modelling using finite elements: examples from 3D resistivity and 2D magnetotellurics. J. Geophys. Eng. 13, S59–S73. http://dx.doi.org/10.1088/1742-2132/13/2/S59.
Shewchuk, J.R., 1996. Triangle: engineering a 2D quality mesh generator and delaunay triangulator. In: Lin, M.C., Manocha, D. (Eds.), Applied Computational Geometry: towards Geometric Engineering, Volume 1148 of Lecture Notes in Computer Science. Springer-Verlag, pp. 203–222. http://dx.doi.org/10.1016/S0925-7721(01)00047-5. From the First ACM Workshop on Applied Computational Geometry.
Si, H., 2015. TetGen, a delaunay-based quality tetrahedral mesh generator. ACM Trans. Math. Softw. 41, 1–36. http://dx.doi.org/10.1145/2629697.
Sulzbacher, H., Wiederhold, H., Siemon, B., Grinat, M., Igel, J., Burschil, T., Günther, T., Hinsby, K., 2012. Numerical modelling of climate change impacts on freshwater lenses on the North Sea Island of Borkum using hydrological and geophysical methods. Hydrol. Earth Syst. Sci. 16, 3621–3643. http://dx.doi.org/10.5194/hess-16-3621-2012.
Uieda, L., Oliveira Jr., V.C.O., Barbosa, V.C.F., 2013. Modeling the earth with fatiando a terra. In: van der Walt, S., Millman, J., Huff, K. (Eds.), Proceedings of the 12th Python in Science Conference, pp. 96–103.
Wagner, F.M., 2016. New Developments in Electrical Resistivity Imaging with Applications to Geological CO2 Storage. Ph.D. thesis. ETH Zürich. http://dx.doi.org/10.3929/ethz-a-010636965.
Wagner, F.M., Günther, T., Schmidt-Hattenberger, C., Maurer, H., 2015. Constructive optimization of electrode locations for target-focused resistivity monitoring. Geophysics 80, E29–E40. http://dx.doi.org/10.1190/geo2014-0214.1.
Wagner, F.M., Möller, M., Schmidt-Hattenberger, C., Kempka, T., Maurer, H., 2013. Monitoring freshwater salinization in analog transport models by time-lapse electrical resistivity tomography. J. Appl. Geophys. 89, 84–95. http://dx.doi.org/10.1016/j.jappgeo.2012.11.013.

Weigand, M., Kemna, A., 2016. Debye decomposition of time-lapse spectral induced polarisation data. Comput. Geosci. 86, 34–45. http://dx.doi.org/10.1016/j.cageo.2015.09.021.

Wellmann, J.F., Croucher, A., Regenauer-Lieb, K., 2012. Python scripting libraries for subsurface fluid and heat flow simulations with TOUGH2 and SHEMAT. Comput. Geosci. 43, 197–206. http://dx.doi.org/10.1016/j.cageo.2011.10.011.

Whitaker, S., 1986. Flow in porous media I: a theoretical derivation of Darcy's law. Transp. Porous Media 1, 3–25. http://dx.doi.org/10.1007/BF01036523.

Wyllie, M.R.J., Gregory, A.R., Gardner, L.W., 1956. Elastic wave velocities in heterogeneous and porous media. Geophysics 21, 40–70. http://dx.doi.org/10.1190/1.1438217.