Reproducible hydrogeophysical inversions through the open-source library pyGIMLi

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

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

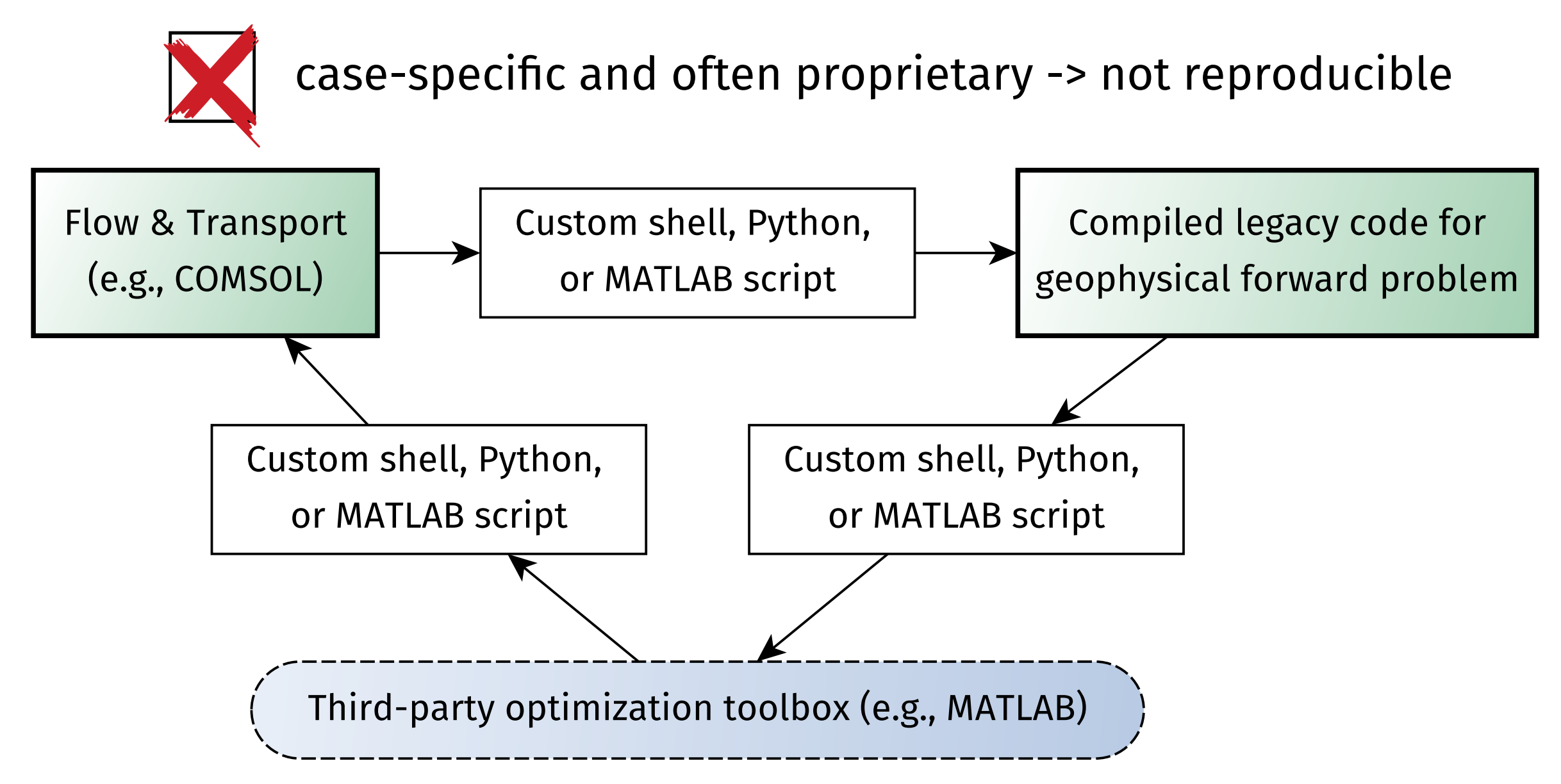
Leibniz Institute for Applied Geophysics

UNIVERSITÄT BONN

NS41B-0016

Computational hydrogeophysics is currently not reproducible

- In hydrogeophysics, researchers gain quantitative information on the subsurface by studying the dynamic process of interest together with its geophysical response.
- This requires coupling of different numerical models → obstacle for many practitioners and students.
- Even technically versatile users tend to build individually tailored solutions by coupling different (and often proprietary) forward simulators.
- The lack of reproducibility represents an impediment for the advancement of hydrogeophysics.


case-specific and often proprietary -> not reproducible
Flow & Transport (e.g., COMSOL)
Custom shell, Python, or MATLAB script
Compiled legacy code for geophysical forward problem
Custom shell, Python, or MATLAB script
Custom shell, Python, or MATLAB script
Third-party optimization toolbox (e.g., MATLAB)

Versatile open-source software opens up new possibilities

We argue that the reproducibility of studies in computational hydrogeophysics, and therefore the advancement of the field itself, requires versatile open-source software.

✓ free, open-source, platform compatible -> reproducible

jupyter python powered simpeg

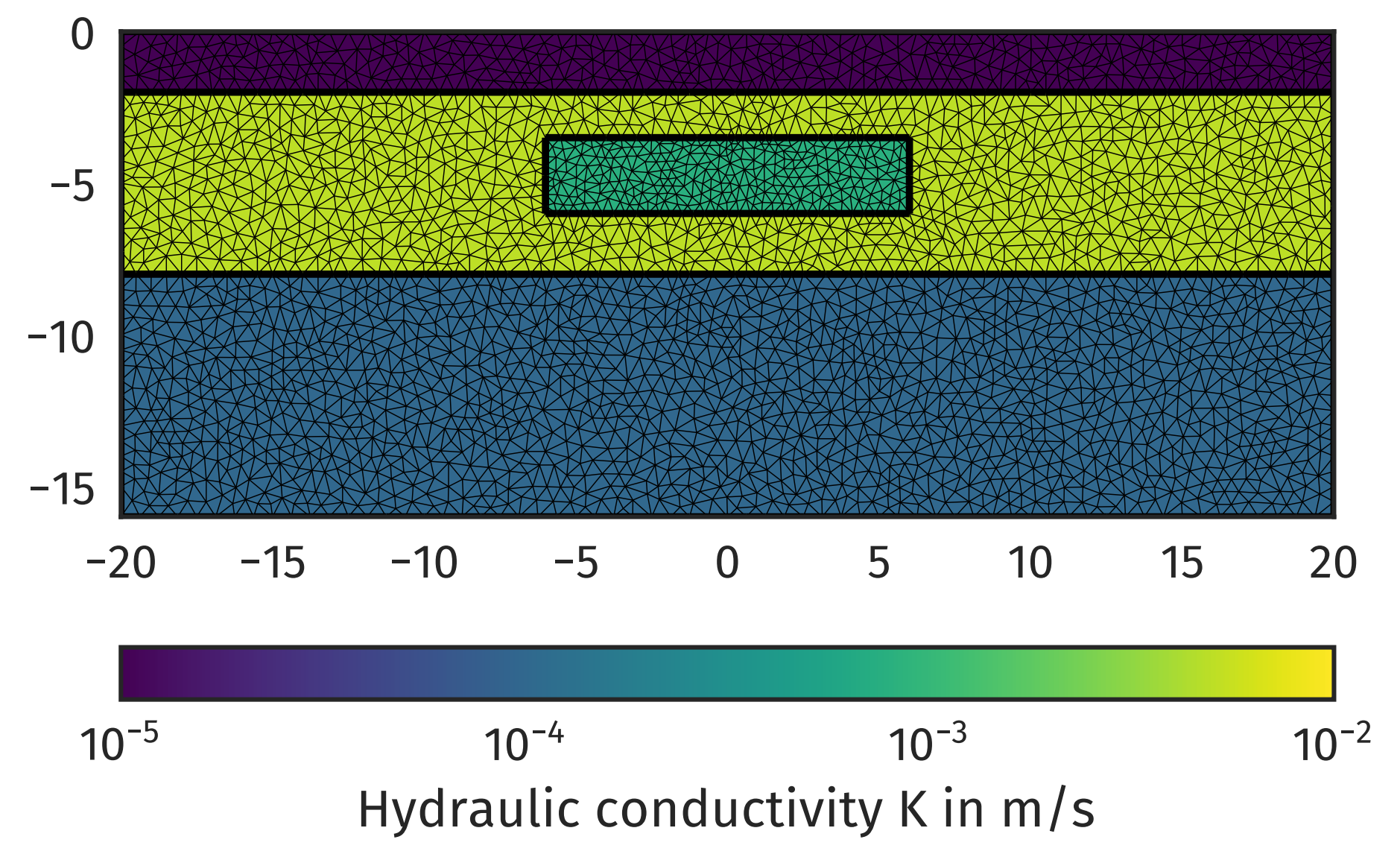
pyGIMLi is an object-oriented library for modeling and inversion in geophysics and offers:

- management of structured and unstructured meshes in 2D & 3D
- computationally efficient finite-element and finite-volume solvers
- various geophysical forward operators
- Gauss-Newton based frameworks for constrained, joint and fully-coupled inversions
- region-specific regularization control
- open-source, platform compatible, documented & tested code

① Model creation

```python
import pygimli as pg
import pygimli.meshtools as mt
# Create geometry definition of domain
world = mt.createWorld(
    start=[-20, 0], end=[20, -16],
    layers=[-2, -8])
# Create a heterogeneous block
block = mt.createRectangle(
    start=[-6, -3.5], end=[6, -6.0],
    marker=4, area=0.1)
# Merge geometrical entities
geom = mt.mergePLC([world, block])
# Create a mesh from the geometry definition
mesh = mt.createMesh(geom, quality=32)
# Map regions to hydraulic conductivity (m/s)
kMap = [[1, 1e-8], [2, 5e-3],
        [3, 1e-4], [4, 8e-4]]
```

```python
# Map conductivity values to each cell
K = pg.solver.parseMapToCellArray(kMap, mesh)
pg.show(mesh, K, savefig="mesh_with_K.pdf")
```



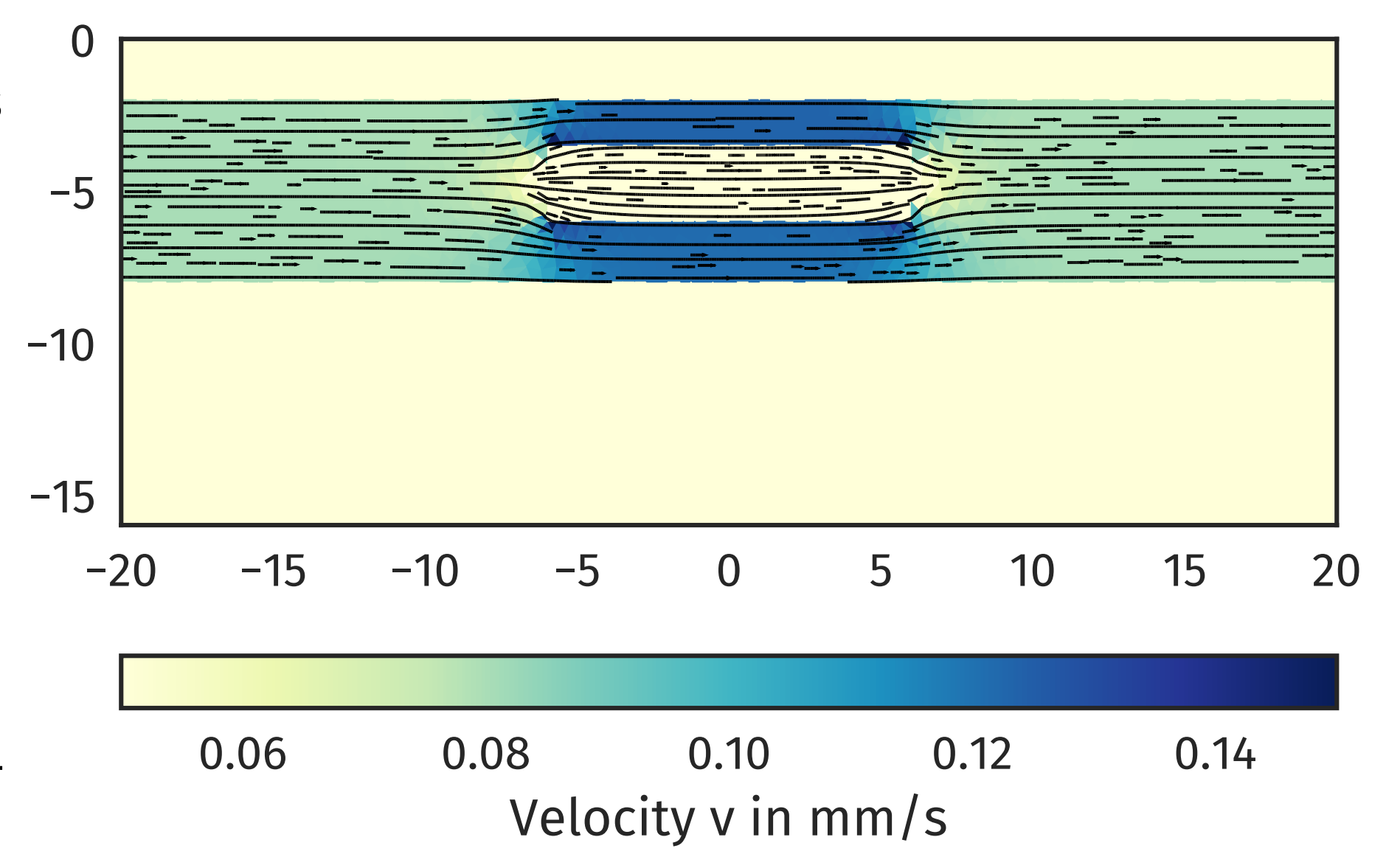
② Modeling groundwater flow

```python
from pygimli.solver import grad
from pygimli.solver import solveFiniteElements
# Dirichlet conditions for hydraulic potential
pBound = [[[1, 2, 3], 0.75], [[5, 6, 7], 0.0]]
# Solve for hydraulic potential
p = solveFiniteElements(mesh, a=K, uB=pBound)
```

Solve Darcy's Law:

$$\nabla \cdot (K \nabla p) = 0$$

```python
# Calculate velocity as gradient of p
vel = -grad(mesh, p) * np.asarray([K, K, K]).T
```



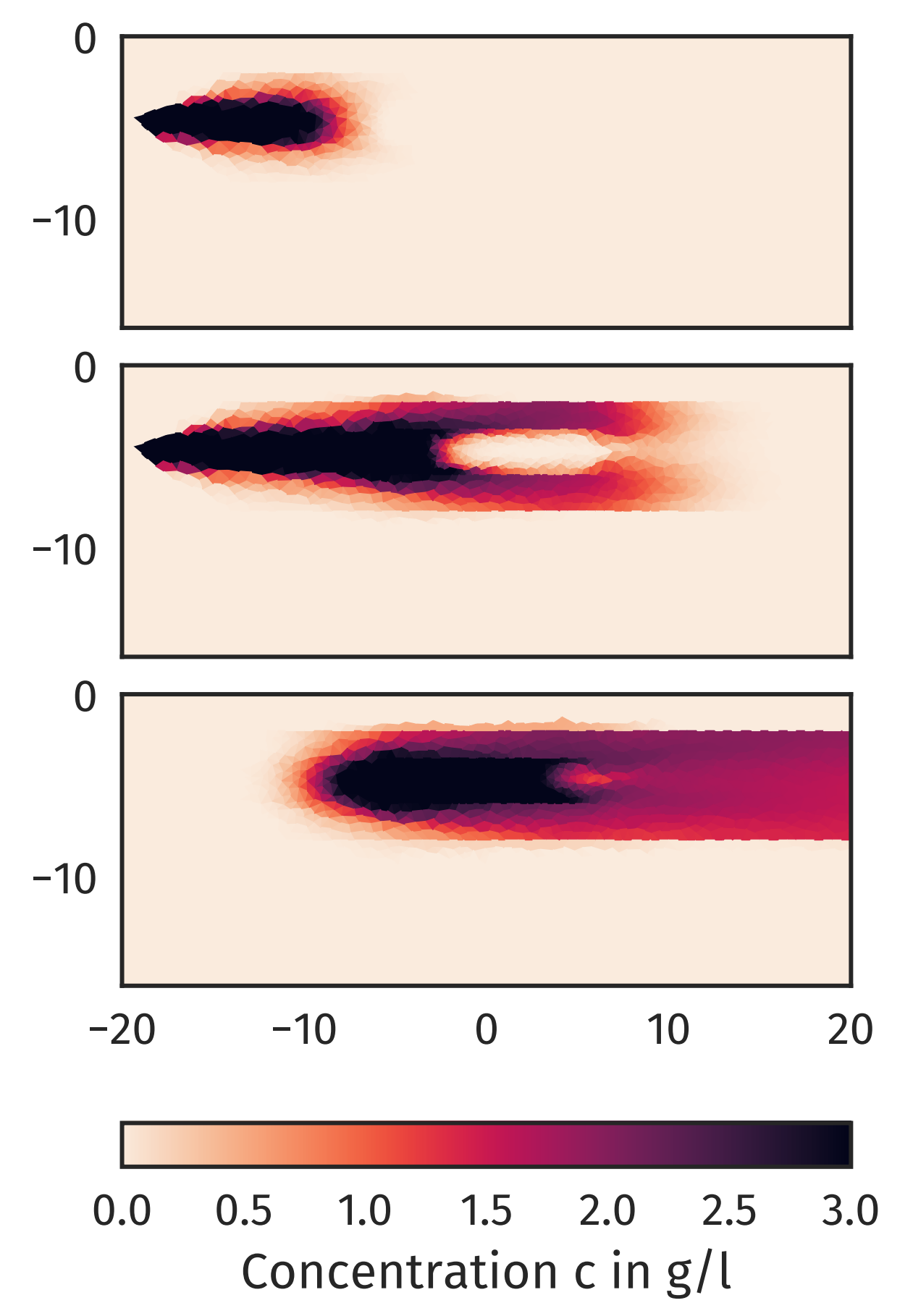
③ Modeling tracer transport

```python
# Create source vector
S = pg.RVector(mesh.cellCount(), 0.0)
# Fill 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 velocity
dispersion = pg.abs(vel) * 1e-2
# We need velocities on cell nodes
veln = mt.cellDataToNodeData(mesh, vel)
```

Solve advection-diffusion equation:

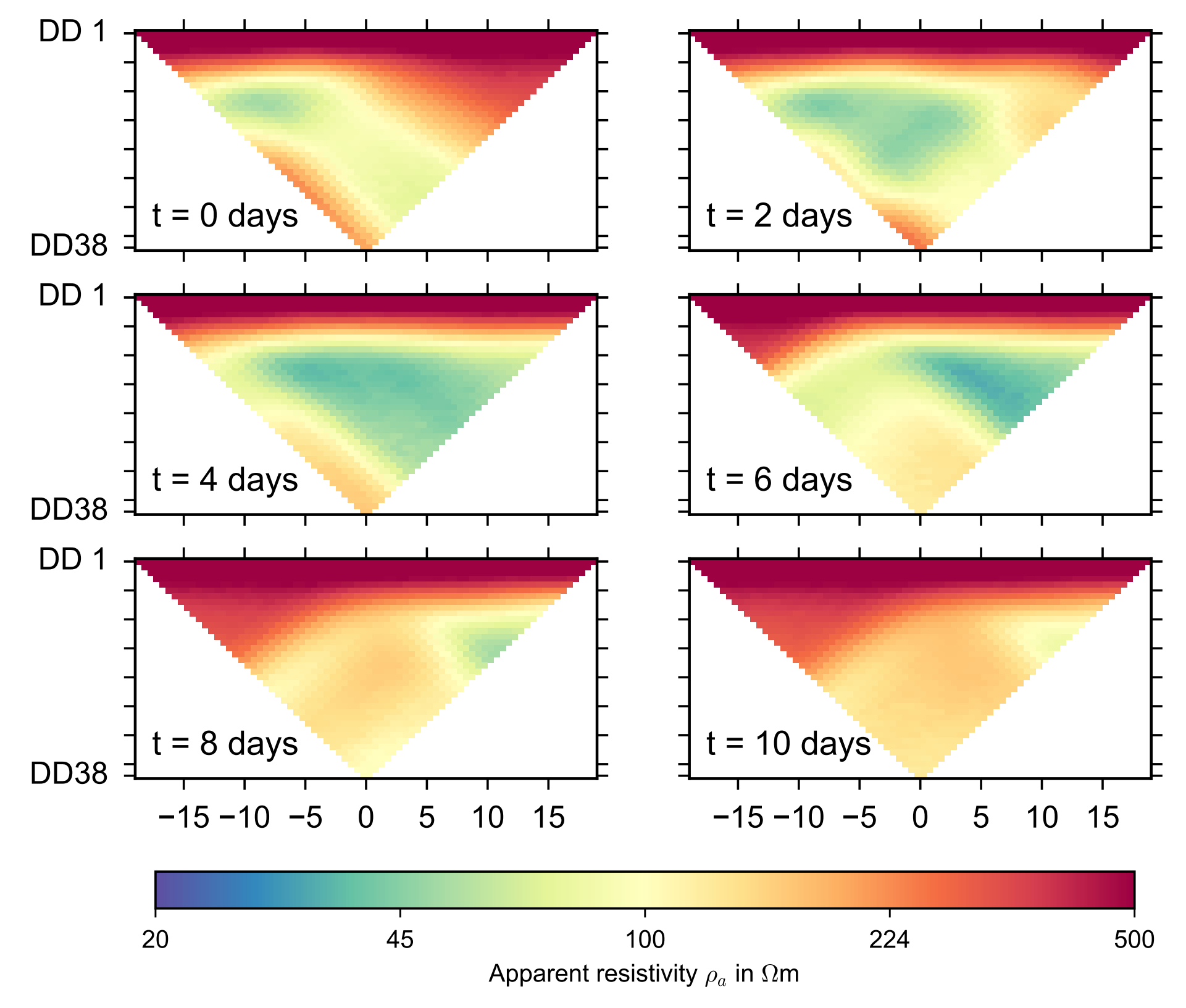
$$\frac{\partial c}{\partial t} = \nabla \cdot (D \nabla c) - \mathbf{v} \cdot \nabla (c) + S$$

```python
conc = pg.solver.solveFiniteVolume(
    mesh, a=dispersion, f=S, vel=veln,
    times=t, uB=[1, 0],
    scheme="PS", verbose=0)
```

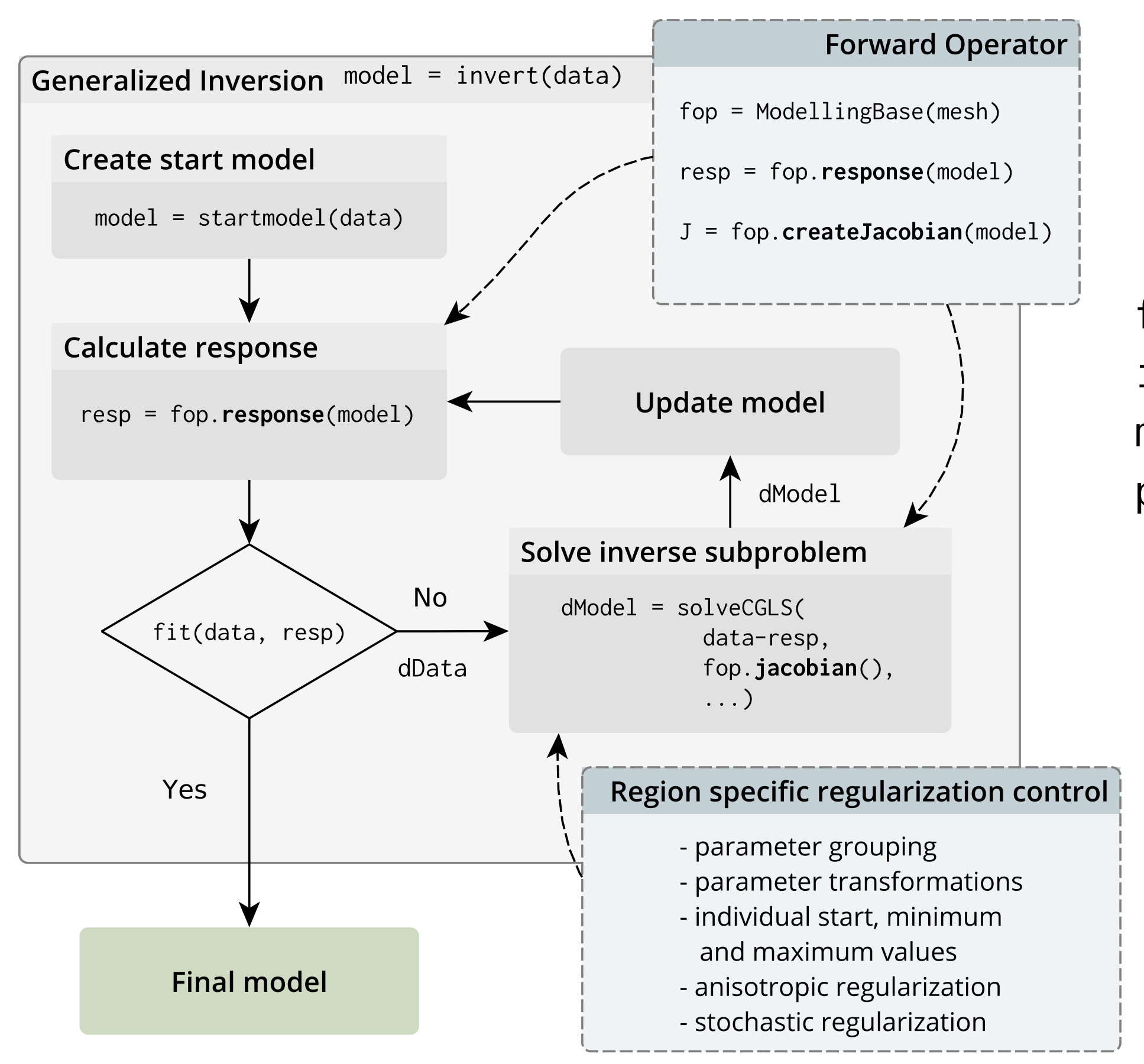


④ Simulating time-lapse resistivity measurements

```python
# Create survey measurement scheme
scheme = ert.createERTData(
    pg.utils.grange(-20, 20, dx=1.0),
    schemeName="dd")
# Create suitable mesh for ERT simulation
meshERT = mt.createParaMesh(
    scheme, quality=33, paraMaxCellSize=0.2,
    boundaryMaxCellSize=50, smooth=[1, 2])
# Select 10 time frame to simulate ERT data
timesERT = pg.IndexArray(
    np.floor(np.linspace(0, len(c)-1, 10)))
# Create fluid conductivity for concentration
sigmaFluid = c[timesERT] * 0.1 + 0.01
# Calculate bulk resistivity based on Archie
res = petro.resistivityArchie(
    rFluid=1. / sigmaFluid,
    porosity=0.3, m=1.3,
    mesh=mesh, meshI=meshERT, fill=1)
rhoa = ERT.simulate(meshERT, res, scheme, returnArray=True)
```

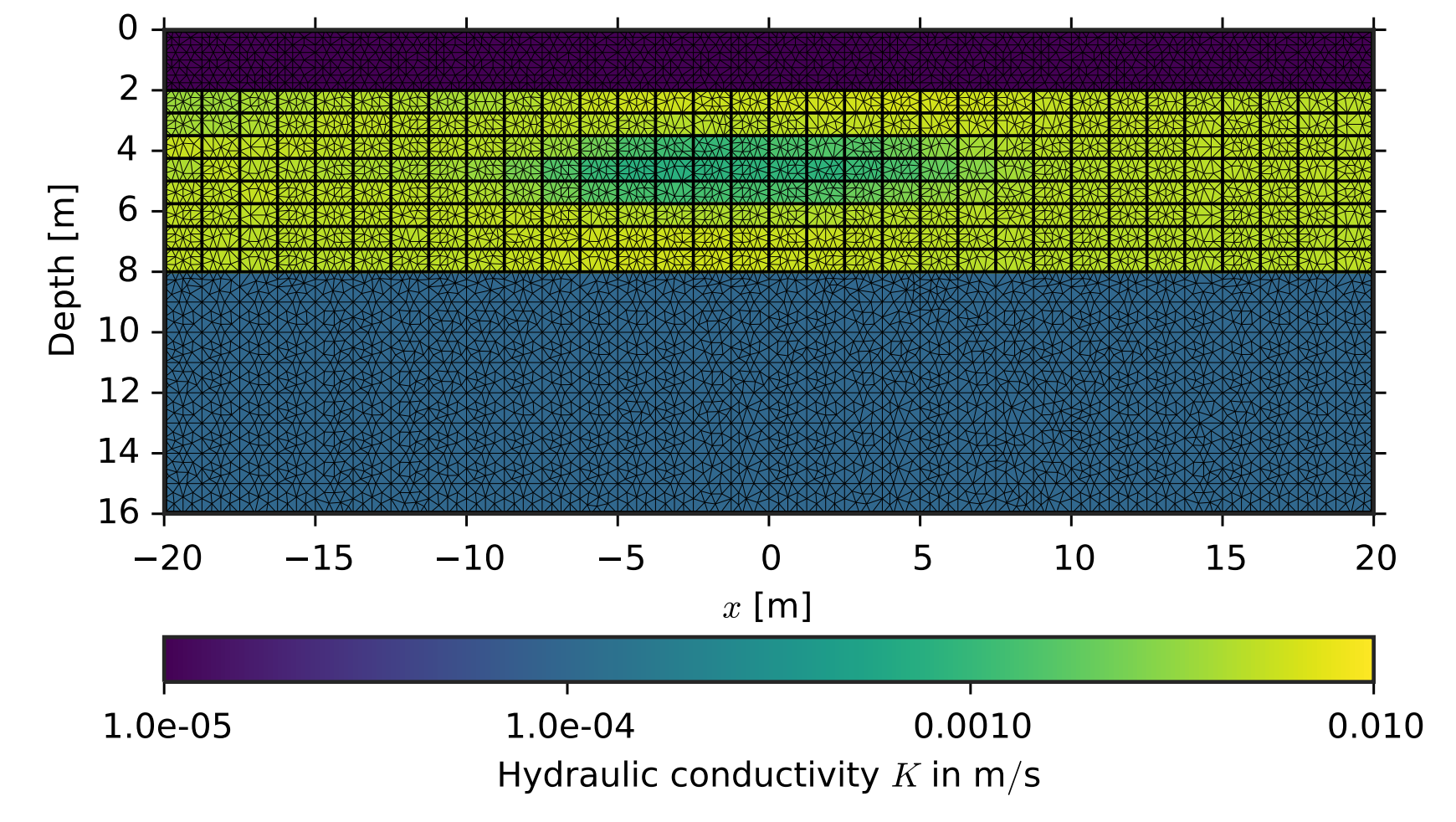


⑤ Fully-coupled inversion


Generalized Inversion
model = invert(data)
Create start model
model = startmodel(data)
Calculate response
resp = fop.response(model)
Update model
dModel
fit(data, resp)
No
dData
Yes
Solve inverse subproblem
dModel = solveCGLS(data-resp, fop.jacobian(), ...)
Final model
Forward Operator
fop = ModellingBase(mesh)
resp = fop.response(model)
J = fop.createJacobian(model)
Region specific regularization control
- parameter grouping
- parameter transformations
- individual start, minimum and maximum values
- anisotropic regularization
- stochastic regularization

```python
class MyForwardOperator(pg.ModellingBase):
    def response(self, model):
        """Perform forward modeling d=f(m)."""
        # call steps 2-4
        return data

fop = MyForwardOperator()
inv = pg.RInversion(fop, mesh)
model = inv.run(data, **inversion_settings)
pg.show(mesh, model)
```



Additional resources

Scan the QR code or go to agu17.pygimli.org for paper, poster and a live demo of the example presented.

Reference

Rücker, C., Günther, T., Wagner, F.M., 2017. pyGIMLi: An open-source library for modelling and inversion in geophysics, *Computers and Geosciences*, 109, 106-123, doi:10.1016/j.cageo.2017.07.011.

Acknowledgments

The first author gratefully acknowledges travel funding received by the Dr. Erich Ritter foundation in cooperation with the Water Science Alliance.