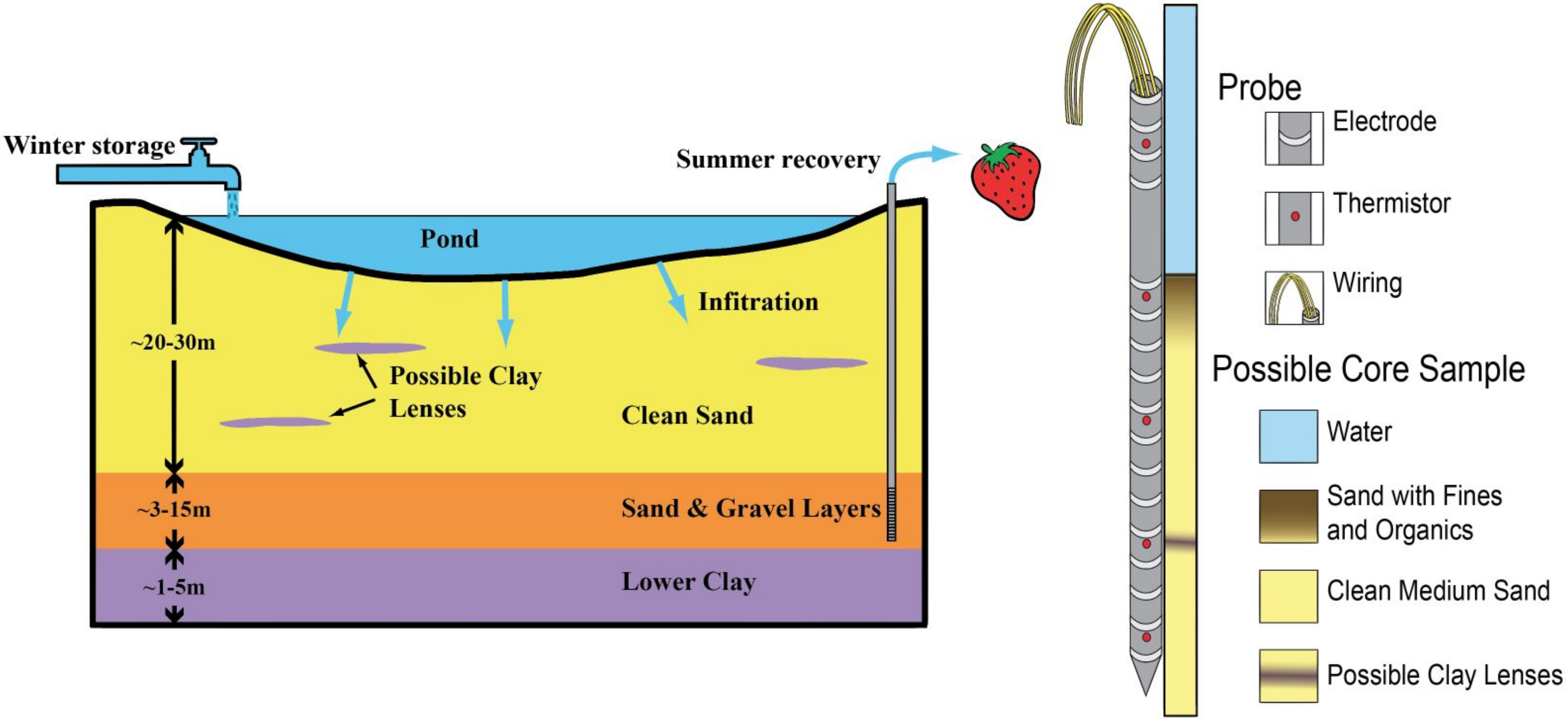
Richards equation

Forward modeling and inversions

Rowan Cockett
Nov 22, 2016

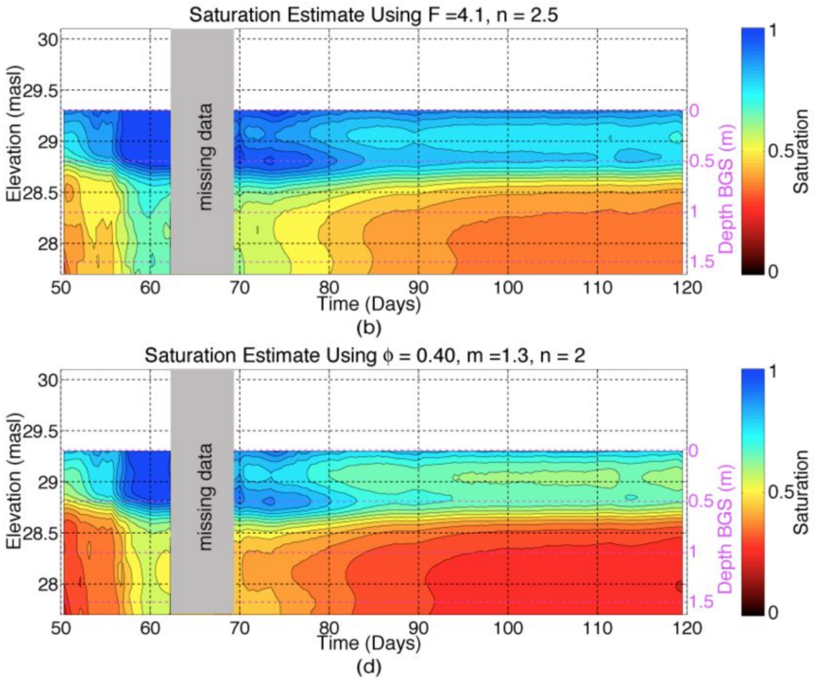
simpeg

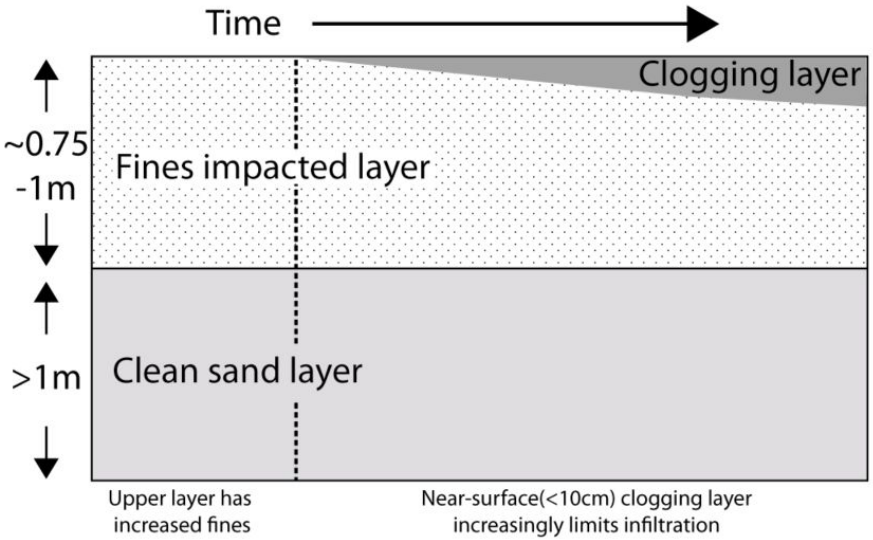


Pidlisecky, **Cockett** & Knight, 2013

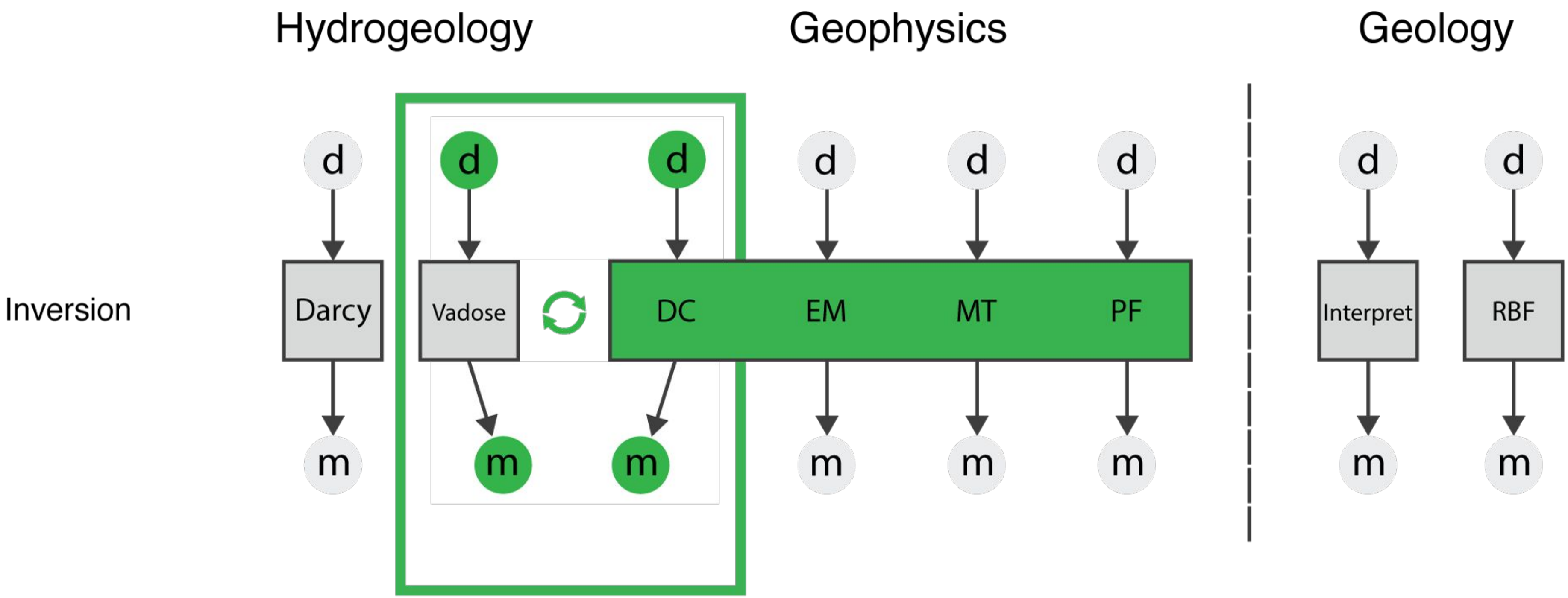
Predict & Decide

Data → Physics → Interpret → Decide



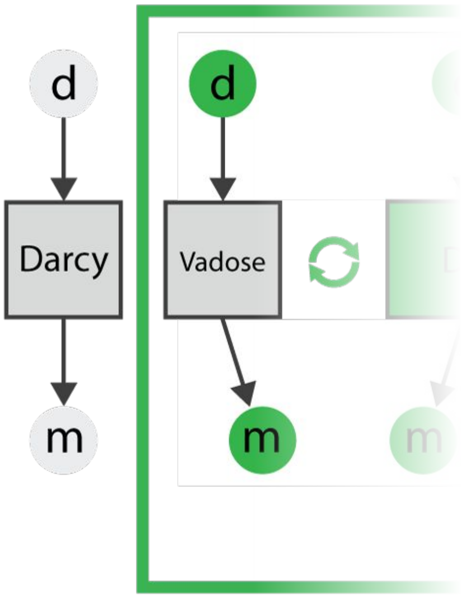


Pidlisecky, **Cockett** & Knight, 2013


Hydrogeology
Geophysics
Geology
Inversion
d
Darcy
m
d
Vadose
m
d
DC
m
d
EM
m
d
MT
m
d
PF
m
d
Interpret
m
d
RBF
m

Inversion

Hydrogeology



Richards equation:

$$\frac{\partial \theta(\psi)}{\partial t} - \underbrace{\nabla \cdot k(\psi) \nabla \psi}_{\text{Diffusion}} - \underbrace{\frac{\partial k(\psi)}{\partial z}}_{\text{Advection}} = 0 \quad \psi \in \Omega$$

Empirical equation (e.g. van Genuchten-Mualem):

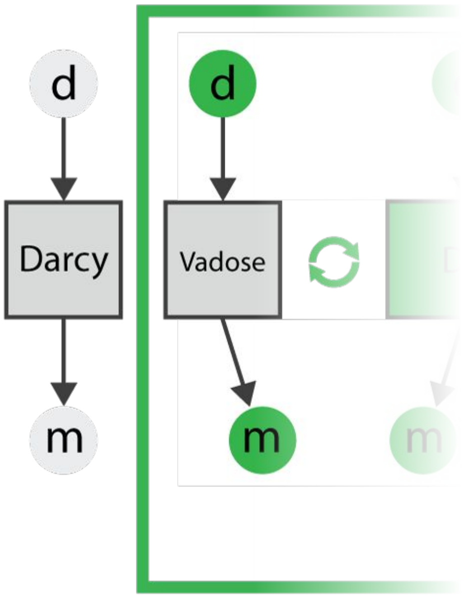
$$\theta(\psi) = \begin{cases} \theta_r + \dfrac{\theta_s - \theta_r}{(1 + |\alpha\psi|^n)^m} & \psi < 0 \\ \theta_s & \psi \geq 0 \end{cases}$$

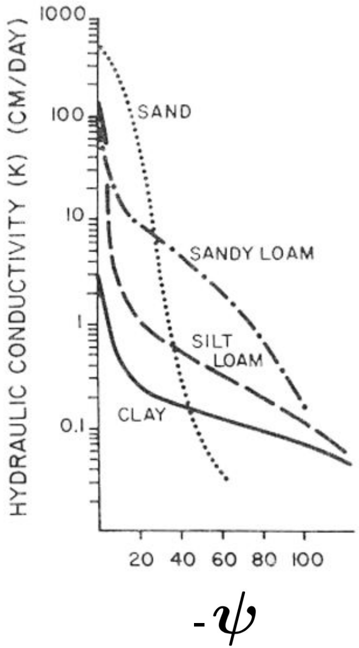
$$k(\psi) = \begin{cases} K_s \theta_e(\psi)^l (1 - (1 - \theta_e(\psi)^{-m})^m)^2 & \psi < 0 \\ K_s & \psi \geq 0 \end{cases}$$

$$\theta_e(\psi) = \frac{\theta(\psi) - \theta_r}{\theta_s - \theta_r}, \qquad m = 1 - \frac{1}{n}, \qquad n > 1$$

Hydrogeology

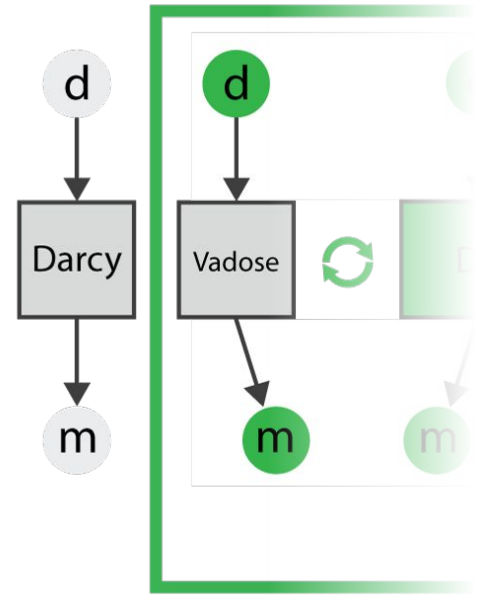
Inversion





Hydrogeology

Inversion



$$\frac{\partial \theta(\psi)}{\partial t} - \nabla \cdot k(\psi)\nabla\psi - \frac{\partial k(\psi)}{\partial z} = 0 \quad \psi \in \Omega$$

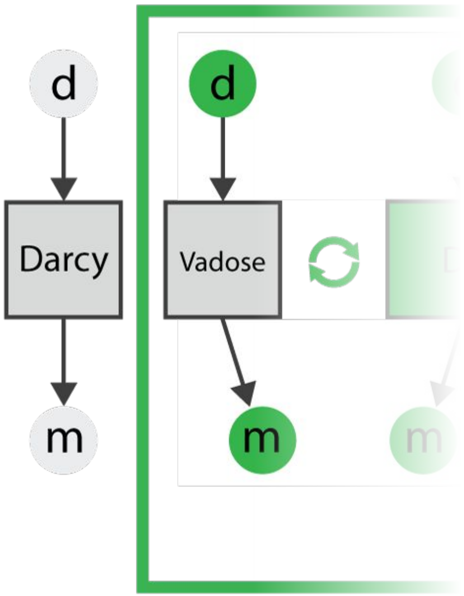
$$\theta(\psi) = \begin{cases} \theta_r + \dfrac{\theta_s - \theta_r}{(1 + |\alpha\psi|^n)^m} & \psi < 0 \\ \theta_s & \psi \geq 0 \end{cases}$$

$$k(\psi) = \begin{cases} K_s \theta_e(\psi)^l (1 - (1 - \theta_e(\psi)^{-m})^m)^2 & \psi < 0 \\ K_s & \psi \geq 0 \end{cases}$$

$$\theta_e(\psi) = \frac{\theta(\psi) - \theta_r}{\theta_s - \theta_r}, \qquad m = 1 - \frac{1}{n}, \qquad n > 1$$

Hydrogeology

Inversion



$$\frac{\partial \theta(\psi)}{\partial t} - \nabla \cdot k(\psi)\nabla\psi - \frac{\partial k(\psi)}{\partial z} = 0 \quad \psi \in \Omega$$

Heterogeneous?

$$\theta(\psi) = \begin{cases} \theta_r + \dfrac{\theta_s - \theta_r}{(1 + |\alpha\psi|^n)^m} & \psi < 0 \\ \theta_s & \psi \geq 0 \end{cases}$$

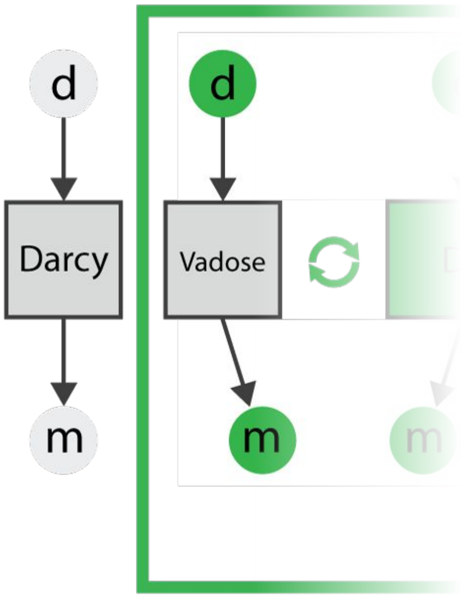
$$k(\psi) = \begin{cases} K_s \theta_e(\psi)^l (1 - (1 - \theta_e(\psi)^{-m})^m)^2 & \psi < 0 \\ K_s & \psi \geq 0 \end{cases}$$

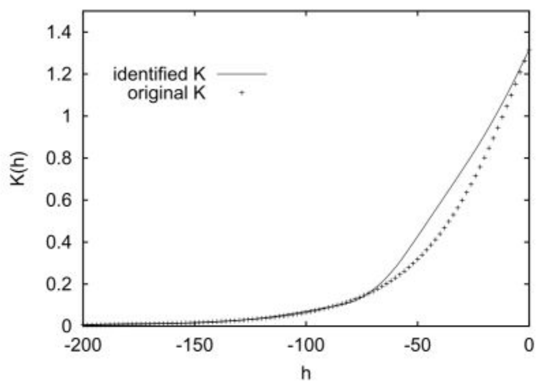
$$\theta_e(\psi) = \frac{\theta(\psi) - \theta_r}{\theta_s - \theta_r}, \qquad m = 1 - \frac{1}{n}, \qquad n > 1$$

Anisotropic?

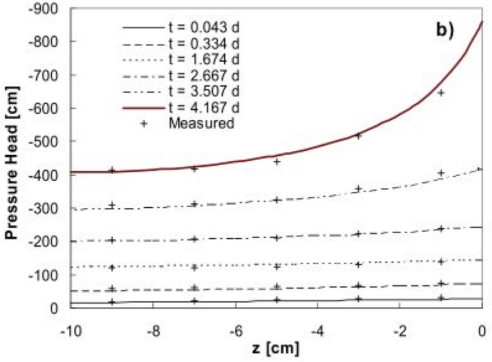
Hydrogeology

Inversion



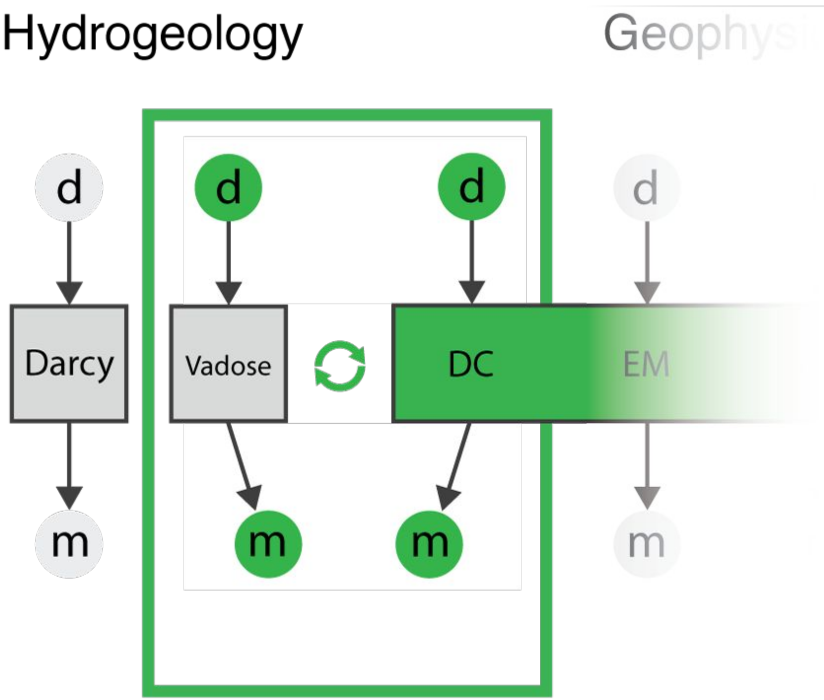


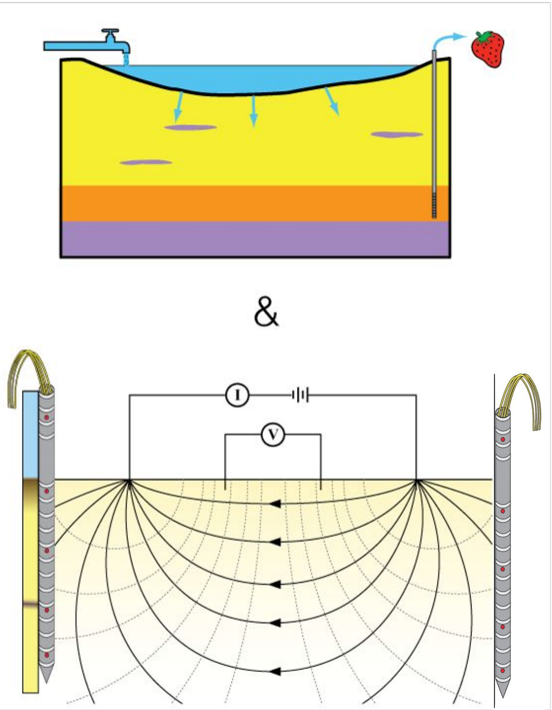
Bitterlich and Knabner, 2002



Šimůnek, van Genuchten, and Senja, 2012

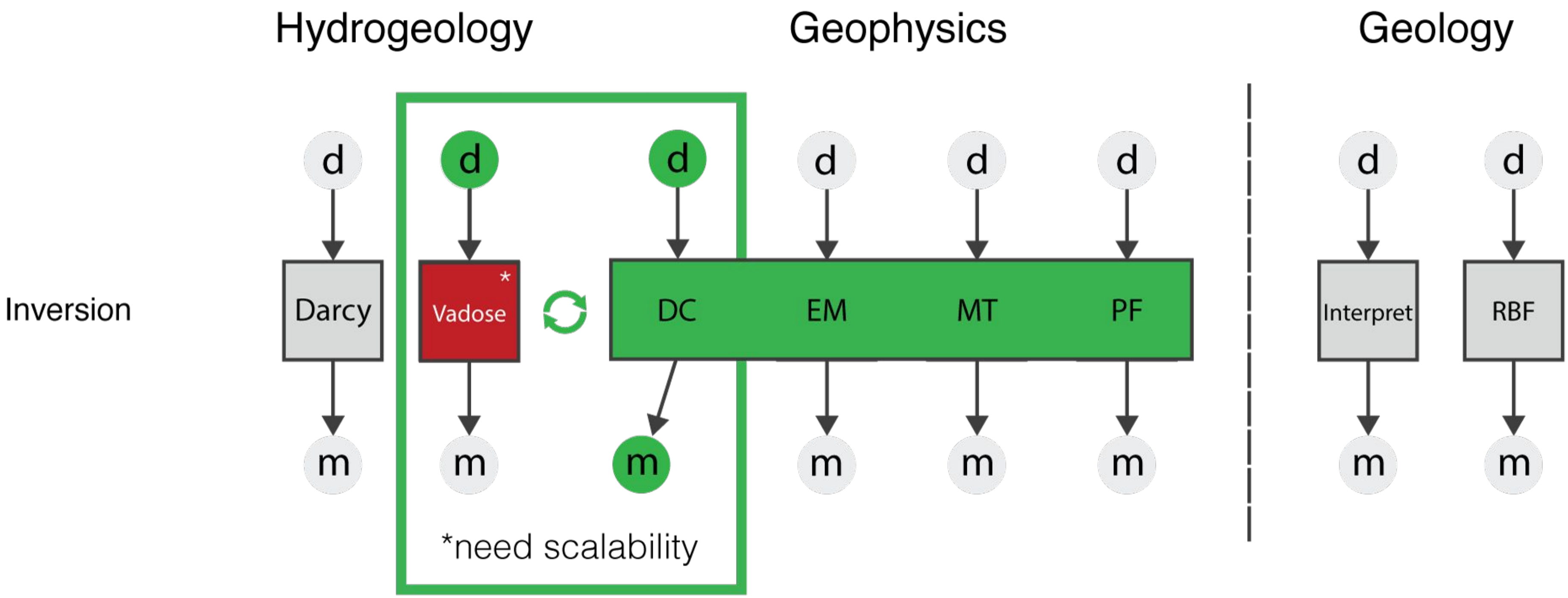
Inversion





Number of parameters: PEST 100s → 1000s

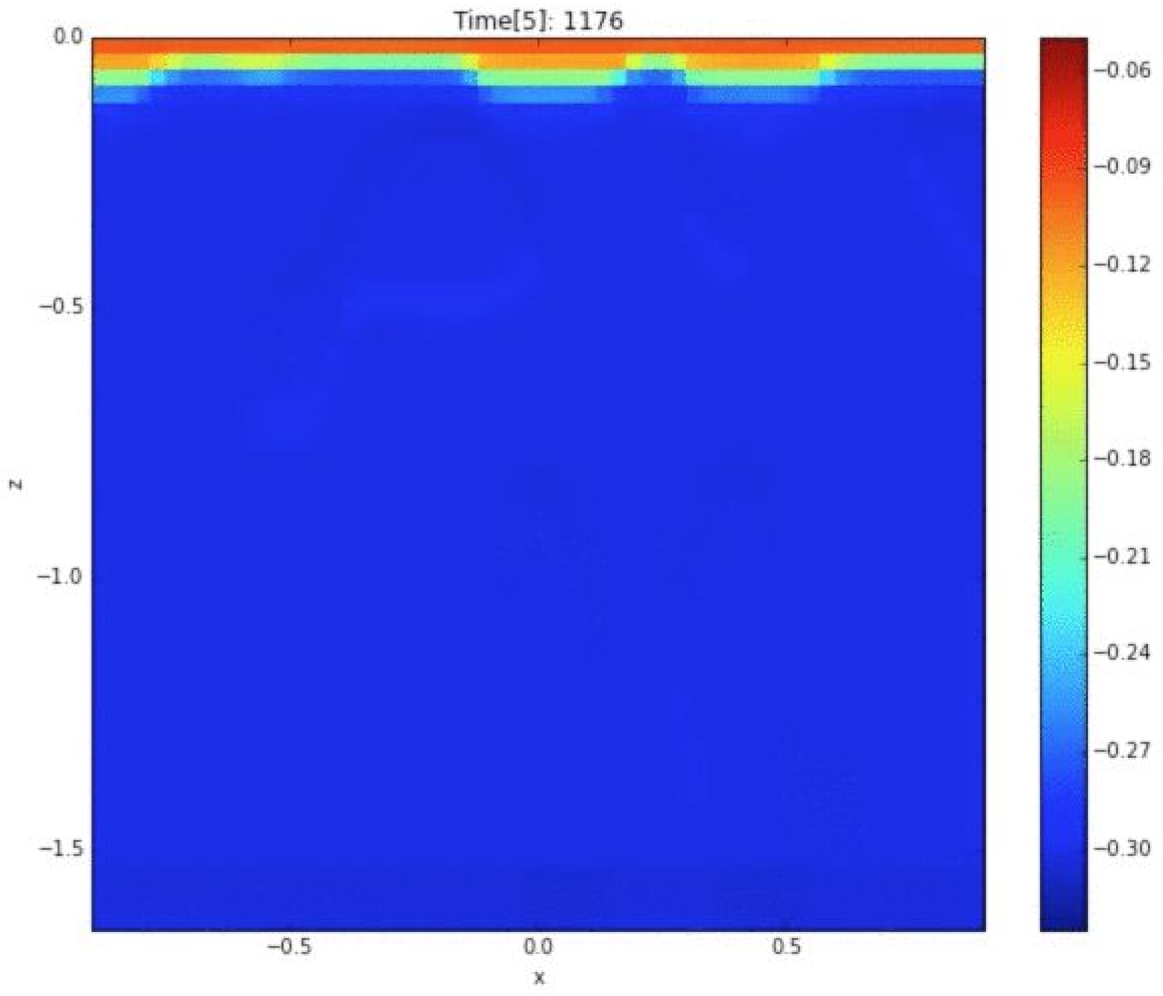
100,000s → 10,000,000s


Hydrogeology
Geophysics
Geology
Inversion
d
d
d
d
d
d
d
d
Darcy
Vadose
*
DC
EM
MT
PF
Interpret
RBF
m
m
m
m
m
m
m
m
*need scalability

Scalable Sensitivity?

1) Investigate forward
2) Implicit sensitivity
3) For example

(Cockett, Heagy & Haber, in prep)


Time[5]: 1176
0.0
−0.5
−1.0
−1.5
z
−0.5
0.0
0.5
x
−0.06
−0.09
−0.12
−0.15
−0.18
−0.21
−0.24
−0.27
−0.30

$$\frac{\partial \theta(\psi)}{\partial t} - \underbrace{\nabla \cdot K(\psi) \nabla \psi}_{\text{Diffusion}} - \underbrace{\frac{\partial K(\psi)}{\partial z}}_{\text{Advection}} = 0$$

$$\theta(\psi) = \begin{cases} \theta_r + \dfrac{\theta_s - \theta_r}{(1 + |\alpha\psi|^n)^m} & \psi < 0 \\ \theta_s & \psi \geq 0 \end{cases}$$

$$k(\psi) = \begin{cases} K_s \theta_e(\psi)^l (1 - (1 - \theta_e(\psi)^{-m})^m)^2 & \psi < 0 \\ K_s & \psi \geq 0 \end{cases}$$

$$\theta_e(\psi) = \frac{\theta(\psi) - \theta_r}{\theta_s - \theta_r}, \qquad m = 1 - \frac{1}{n}, \qquad n > 1$$

class SimPEG.FLOW.Richards.RichardsProblem.Richards

Bases: SimPEG.Problem.BaseTimeProblem

docstring for RichardsProblem

Required

Parameters:

- **boundaryConditions** (Array) – boundary co
  ‘float’>, <type ‘int’> with shape (*)
- **Solver** (Property) – Numerical Solver, Defau
- **mapping** (Property) – the mapping
- **tolRootFinder** (Float) – Maximum iteratio
  Default: 0.0001
- **maxIterRootFinder** (Integer) – Maximum i
  integer, Default: 30
- **initialConditions** (Array) – boundary condi
  ‘float’>, <type ‘int’> with shape (*)
- **debug** (Bool) – Show all messages, a boolea
- **model** (Model) – Inversion model., a numpy
  shape (*)
- **doNewton** (Bool) – Do a Newton iteration
  Default: False
- **method** (StringChoice) – Formulation used,
  mixed or head

```python
class RichardsProblem(Problem.BaseTimeProblem):
    """RichardsProblem"""

    mapping = properties.Property("the mapping")

    boundaryConditions = properties.Array("boundary conditions.")
    initialConditions = properties.Array("boundary conditions.")

    surveyPair = RichardsSurvey
    mapPair = RichardsMap

    debug = properties.Bool("Show all messages")

    Solver = properties.Property("Numerical Solver", default=lambda: Solver)
    solverOpts = {}

    method = properties.StringChoice(
        "Formulation used, See notes in Celia et al., 1990.",
        choices=['mixed', 'head']
    )

    doNewton = properties.Bool(
        "Do a Newton iteration vs. a Picard iteration ",
        default=False
    )

    maxIterRootFinder = properties.Integer(
        "Maximum iterations for rootFinder iteration.",
        default=30
    )

    tolRootFinder = properties.Float(
        "Maximum iterations for rootFinder iteration.",
        default=1e-4
    )
```

Getting rid of mapping

```python
prob = Richards.RichardsProblem(
    mesh,
    mapping=E,
    timeSteps=[(40, 3), (60, 3)],
    Solver=Solver,
    boundaryConditions=bc,
    initialConditions=h,
    doNewton=False,
    method='mixed',
    tolRootFinder=1e-6,
    debug=False
)
```

```python
class RichardsMap(object):
    """docstring for RichardsMap"""

    mesh = None  #: SimPEG mesh

    @property
    def thetaModel(self):
        """Model for moisture content"""
        return self._thetaModel

    @property
    def kModel(self):
        """Model for hydraulic conductivity"""
        return self._kModel

    def __init__(self, mesh, thetaModel, kModel):
        self.mesh = mesh
        assert isinstance(thetaModel, NonLinearMap)
        assert isinstance(kModel, NonLinearMap)

        self._thetaModel = thetaModel
        self._kModel = kModel

    def theta(self, u, m):
        return self.thetaModel(u, m)

    def thetaDerivM(self, u, m):
        return self.thetaModel.derivM(u, m)

    def thetaDerivU(self, u, m):
        return self.thetaModel.derivU(u, m)
```

Getting rid of mapping

```python
prob = Richards.RichardsProblem(
    mesh,
    mapping=E,
    timeSteps=[(40, 3), (60, 3)],
    Solver=Solver,
    boundaryConditions=bc,
    initialConditions=h,
    doNewton=False,
    method='mixed',
    tolRootFinder=1e-6,
    debug=False
)
```

```python
kModel = Richards.Empirical.Haverkamp_k(mesh)
thetaModel = Richards.Empirical.Haverkamp_theta(mesh)
kModel.KsMap = Maps.ExpMap(nP=mesh.nC)

prob = Richards.RichardsProblem(
    mesh,
    kModel=kModel,
    thetaModel=thetaModel,
    timeSteps=[(40, 3), (60, 3)],
    Solver=Solver,
    boundaryConditions=bc,
    initialConditions=h,
    doNewton=False,
    method='mixed',
    tolRootFinder=1e-6,
    debug=False
)
```

$$\frac{\partial \theta(\psi)}{\partial t} - \underbrace{\nabla \cdot K(\psi)\nabla\psi}_{\text{Diffusion}} - \underbrace{\frac{\partial K(\psi)}{\partial z}}_{\text{Advection}} = 0$$

```python
class NonLinearModel(Props.BaseSimPEG):

    model = Props.Model("model")

    def __init__(self, mesh, **kwargs):
        self.mesh = mesh
        super(NonLinearModel, self).__init__(**kwargs)

    @property
    def nP(self):
        """Number of parameters in the model."""
        return self.mesh.nC

    def derivU(self, u, model):
        self.model = model
        return df_du

    def derivM(self, u, model):
        self.model = model
        return df_dm
```

$$\frac{\partial \theta(\psi)}{\partial t} - \underbrace{\nabla \cdot K(\psi)\nabla\psi}_{\text{Diffusion}} - \underbrace{\frac{\partial K(\psi)}{\partial z}}_{\text{Advection}} = 0$$

```python
class BaseThetaModel(NonLinearModel):

    def __call__(self, u, model):
        self.model = model
        f = theta(u)
        return f

class BaseKModel(NonLinearModel):

    def __call__(self, u, model):
        self.model = model
        f = k(u)
        return f
```

$$\theta(\psi) = \begin{cases} \theta_r + \dfrac{\theta_s - \theta_r}{(1 + |\alpha\psi|^n)^m} & \psi < 0 \\ \theta_s & \psi \geq 0 \end{cases}$$

$$k(\psi) = \begin{cases} K_s \theta_e(\psi)^l (1 - (1 - \theta_e(\psi)^{-m})^m)^2 & \psi < 0 \\ K_s & \psi \geq 0 \end{cases}$$

$$\theta_e(\psi) = \frac{\theta(\psi) - \theta_r}{\theta_s - \theta_r}, \qquad m = 1 - \frac{1}{n}, \qquad n > 1$$

$$\frac{\partial \theta(\psi)}{\partial t} - \underbrace{\nabla \cdot K(\psi) \nabla \psi}_{\text{Diffusion}} - \underbrace{\frac{\partial K(\psi)}{\partial z}}_{\text{Advection}} = 0$$

```python
class BaseThetaModel(NonLinearModel):

    def __call__(self, u, model):
        self.model = model
        f = theta(u)
        return f

class BaseKModel(NonLinearModel):

    def __call__(self, u, model):
        self.model = model
        f = k(u)
        return f
```

$$\theta(\psi) = \begin{cases} \theta_r + \dfrac{\theta_s - \theta_r}{(1 + |\alpha\psi|^n)^r} \\ \theta_s \end{cases}$$

$$k(\psi) = \begin{cases} K_s \theta_e(\psi)^l (1 - (1 - \theta_e(\psi) \\ K_s \end{cases}$$

$$\theta_e(\psi) = \frac{\theta(\psi) - \theta_r}{\theta_s - \theta_r}, \qquad m = 1 -$$

```python
class Haverkamp_k(BaseKModel):

    Ks, KsMap, KsDeriv = Props.Invertible(
        "Saturated hydraulic conductivity",
        default=24.96
    )

    A, AMap, ADeriv = Props.Invertible(
        "fitting parameter",
        default=1.175e+06
    )

    gamma, gammaMap, gammaDeriv = Props.Invertible(
        "fitting parameter",
        default=4.74
    )

    def derivM(self, u, model):
        self.model = model
        return self._derivKs(u) + self._derivA(u) + self._derivGamma(u)
```

Inverting for multiple parameters

```
opts = [
    ('Ks',
        dict(KsMap=expmap), 1),
    ('A',
        dict(AMap=expmap), 1),
    ('gamma',
        dict(gammaMap=expmap), 1),
    ('Ks-A',
        dict(KsMap=expmap*wires2.one, AMap=expmap*wires2.two), 2),
    ('Ks-gamma',
        dict(KsMap=expmap*wires2.one, gammaMap=expmap*wires2.two), 2),
    ('A-gamma',
        dict(AMap=expmap*wires2.one, gammaMap=expmap*wires2.two), 2),
    ('Ks-A-gamma', dict(
        KsMap=expmap*wires3.one,
        AMap=expmap*wires3.one,
        gammaMap=expmap*wires3.two), 3),
]
```

$\mathbf{J}$ is large and requires dense linear algebra

$$\mathbf{J} = \mathbf{P}\frac{\partial \Psi(\mathbf{m})}{\partial \mathbf{m}}$$

To build $\mathbf{J}$, look at function over every time-step:

$$F(\psi^n, \psi^{n+1}, m) = \frac{\theta^{n+1}(\psi^{n+1}) - \theta^n(\psi^n)}{\Delta t} - \nabla \cdot K(\psi^{n+1}, m)\nabla\psi^{n+1} - \frac{\partial}{\partial z}K(\psi^{n+1}, m) = 0$$

Take the devivative, $\frac{\partial F(\psi^n, \psi^{n+1}, m)}{\partial m}$

$$\frac{1}{\Delta t}\left(\frac{\partial \theta^{n+1}}{\partial \psi^{n+1}}\frac{\partial \psi^{n+1}}{\partial m} - \frac{\partial \theta^n}{\partial \psi^n}\frac{\partial \psi^n}{\partial m}\right) - \nabla \cdot \ \mathrm{diag}\left(\nabla \psi^{n+1}\right)\left(\frac{\partial K}{\partial m} + \frac{\partial K}{\partial \psi^{n+1}}\frac{\partial \psi^{n+1}}{\partial m}\right)$$
$$- \nabla \cdot K(\psi^{n+1})\nabla\frac{\partial \psi^{n+1}}{\partial m} - \frac{\partial}{\partial z}\left(\frac{\partial K}{\partial m} + \frac{\partial K}{\partial \psi^{n+1}}\frac{\partial \psi^{n+1}}{\partial m}\right) = 0$$

$$\overbrace{\left[-\frac{1}{\Delta t}\frac{\partial \theta^n}{\partial \psi^n}\right]}^{\mathbf{A}_{-1}(\psi^n)} \frac{\partial \psi^n}{\partial m} + \overbrace{\left[\frac{1}{\Delta t}\frac{\partial\theta^n}{\partial\psi^{n+1}} - \nabla\cdot\ \mathrm{diag}\left(\nabla\psi^{n+1}\right)\frac{\partial K}{\partial \psi^{n+1}} - \nabla\cdot K(\psi^{n+1})\nabla - \frac{\partial}{\partial z}\frac{\partial K}{\partial\psi^{n+1}}\right]}^{\mathbf{A}_0(\psi^{n+1})}\frac{\partial\psi^{n+1}}{\partial m}$$

$$= \underbrace{\nabla\cdot\ \mathrm{diag}\left(\nabla\psi^{n+1}\right)\frac{\partial K}{\partial m} + \frac{\partial}{\partial z}\frac{\partial K}{\partial m}}_{\mathbf{b}(\psi^{n+1},m)}$$

$$\overbrace{\begin{bmatrix} \mathbf{A}_0(\psi_1) & & & & \\ \mathbf{A}_{-1}(\psi_1) & \mathbf{A}_0(\psi_2) & & & \\ & \mathbf{A}_{-1}(\psi_2) & \mathbf{A}_0(\psi_3) & & \\ & & \ddots & \ddots & \\ & & & \mathbf{A}_{-1}(\psi_{n-1}) & \mathbf{A}_0(\psi_n) \end{bmatrix}}^{\mathbf{A}(\Psi,m)} \overbrace{\begin{bmatrix} \frac{\partial\psi_1}{\partial m} \\ \frac{\partial\psi_2}{\partial m} \\ \vdots \\ \frac{\partial\psi_{n-1}}{\partial m} \\ \frac{\partial\psi_n}{\partial m} \end{bmatrix}}^{\frac{\partial\Psi(m)}{\partial m}} = \overbrace{\begin{bmatrix} \mathbf{b}(\psi_1, m) \\ \mathbf{b}(\psi_2, m) \\ \vdots \\ \mathbf{b}(\psi_{n-1}, m) \\ \mathbf{b}(\psi_n, m) \end{bmatrix}}^{\mathbf{B}(\Psi,m)}$$

$$\mathbf{J} = \mathbf{P}\frac{\partial\Psi(\mathbf{m})}{\partial\mathbf{m}}$$

$$\mathbf{J} = \mathbf{P}\mathbf{A}(\Psi, m)^{-1}\mathbf{B}(\Psi, m)$$

$\mathbf{A}_{-1}(\psi$

$\left[-\frac{1}{\Delta t}\frac{\partial}{\partial}\right.$

$\frac{\partial \psi^{n+1}}{\partial m}$

```
# Compute part of the derivative of:
#
#       DIV*diag(GRAD*hn1+BC*bc)*(AV*(1.0/K))^-1

DdiagGh1 = DIV*Utils.sdiag(GRAD*hn1+BC*bc)
diagAVk2_AVdiagK2 = (
    Utils.sdiag((AV*(1./K1))**(-2)) *
    AV*Utils.sdiag(K1**(-2))
)

# The matrix that we are computing has the form:
#
#   -                                      -   -  -     -  -
#  |  Adiag                                 | | h1 |   | b1 |
#  |   Asub    Adiag                        | | h2 |   | b2 |
#  |            Asub    Adiag               | | h3 | = | b3 |
#  |                 ...     ...            | | .. |   | .. |
#  |                         Asub    Adiag  | | hn |   | bn |
#   -                                      -   -  -     -  -

Asub = (-1.0/dt)*dT

Adiag = (
    (1.0/dt)*dT1 -
    DdiagGh1*diagAVk2_AVdiagK2*dK1 -
    DIV*Utils.sdiag(1./(AV*(1./K1)))*GRAD -
    Dz*diagAVk2_AVdiagK2*dK1
)

B = DdiagGh1*diagAVk2_AVdiagK2*dKm1 + Dz*diagAVk2_AVdiagK2*dKm1
```

$\mathbf{A}_0($ $m)$

$\mathbf{A}_{-1}$ $m)$

$, m)$

$m)$

Literature scraping of hydrogeologic params

```
In [1]: from SimPEG import FLOW

In [2]: VG = FLOW.Richards.Empirical.VanGenuchtenParams()

In [7]: print VG.sand
        print VG.loamySand

{'Ks': 5.83333333333333e-05, 'alpha': 13.8, 'theta_r': 0.02, 'theta_s': 0.417, 'n': 1.592}
{'Ks': 1.69675925925925 9e-05, 'alpha': 11.5, 'theta_r': 0.035, 'theta_s': 0.401, 'n': 1.474}
```

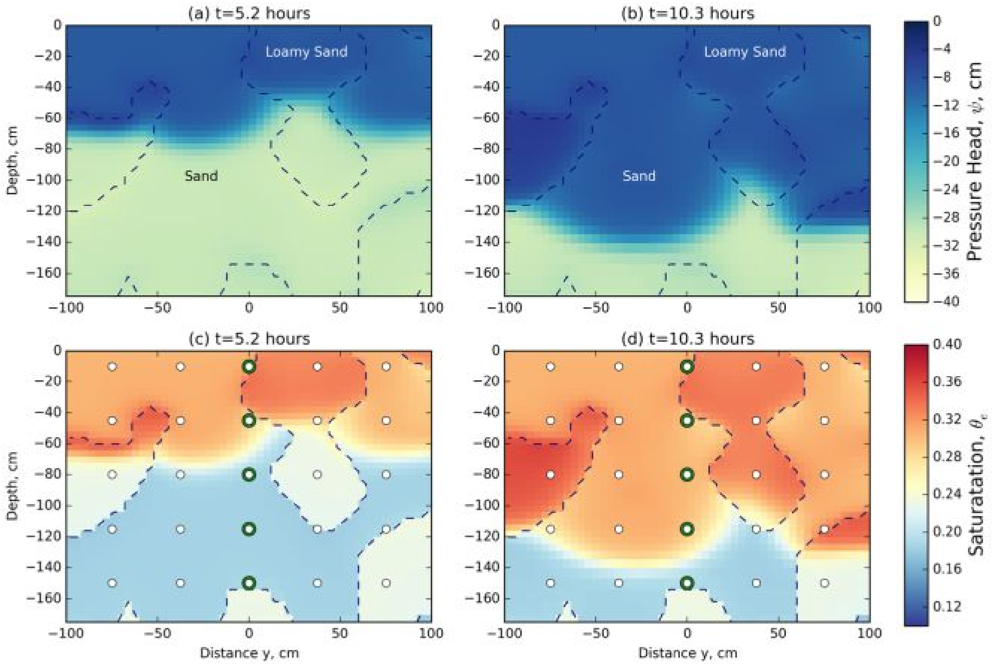


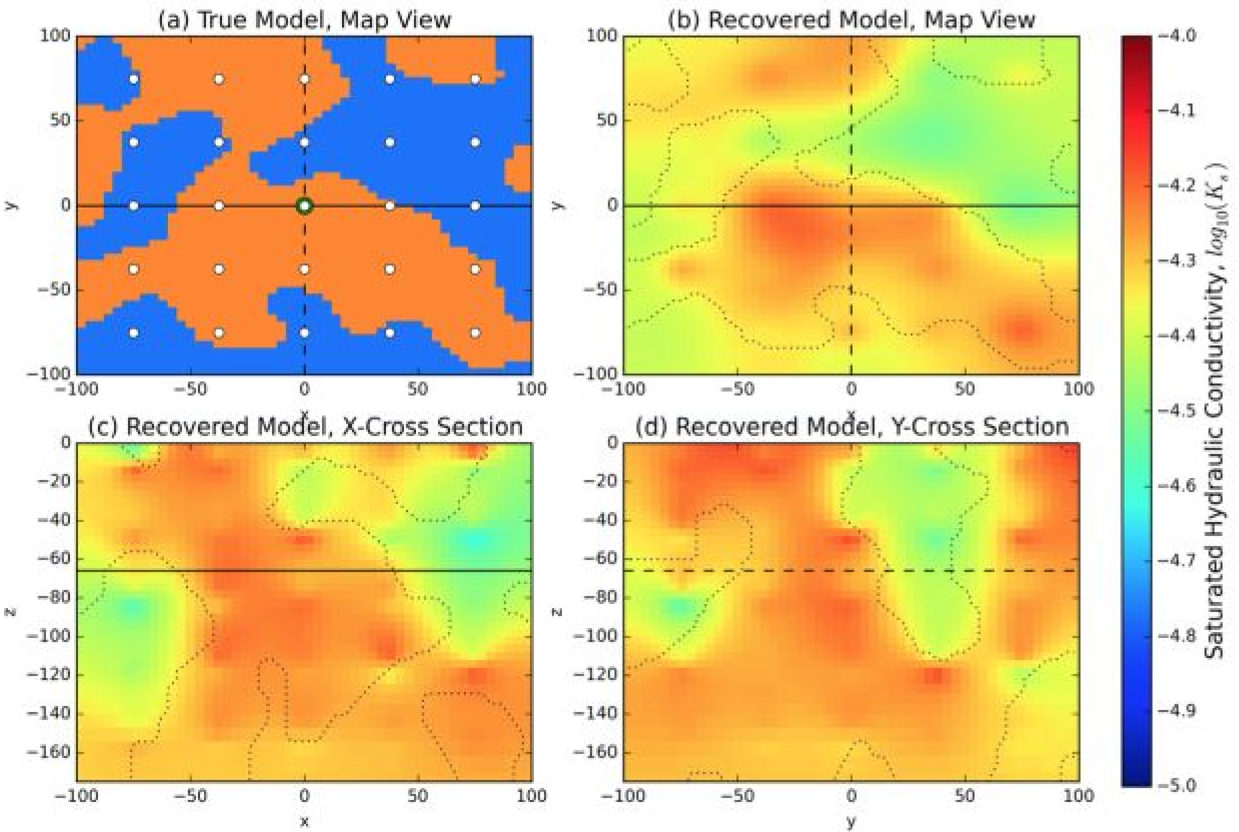
Figure 3: Vertical cross sections through the pressure head and saturation fields from the numerical simulation at two times: (a) pressure head field at $t = 5.2$ hours and (b) $t = 10.3$ hours; and (c) saturation field at $t = 5.2$ hours and (d) $t = 10.3$ hours. The saturation field plots also show measurement locations and green highlighted regions that are shown in Figure 4. The true location of the two soils used are shown with a dashed outline.

4cm discretization
50*50*45 cells in space (1.125E5)
12.3 hours
53 times, (2 - 15 min)
Fields 5.9 million elements

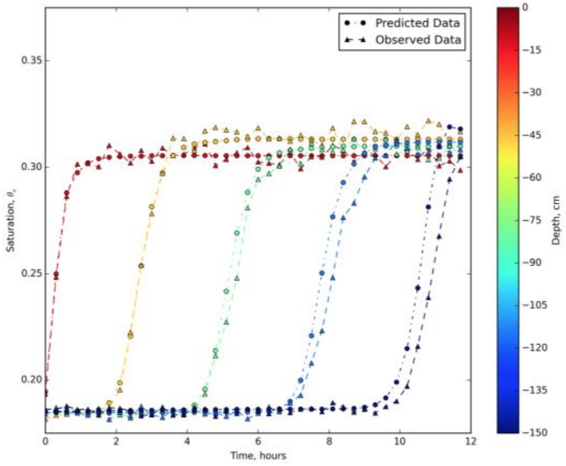
125 saturation estimates (35 cm apart)
40 times (18 min apart)
5000 data points

What can we get out?!

```
In [1]: from SimPEG import FLOW

In [2]: VG = FLOW.Richards.Empirical.VanGenuchtenParams()

In [7]: print VG.sand
        print VG.loamySand

{'Ks': 5.833333333333333e-05, 'alpha': 13.8, 'theta_r': 0.02, 'theta_s': 0.417, 'n': 1.592}
{'Ks': 1.696759259259259e-05, 'alpha': 11.5, 'theta_r': 0.035, 'theta_s': 0.401, 'n': 1.474}
```



Assume everything is sand, and invert for K_s



5000 saturation estimates

112 500 parameters

The dense Jacobian matrix would have 562 million elements