SINTEF

ADJOINT-BASED SENSITIVITIES IN MRST

WITH APPLICATION TO MULTI- SEGMENT WELLS AND CO2 INJECTION

Stein Krogstad, SINTEF Digital, Mathematics and Cybernetics
and coworkers at the Computational Geosciences Group

Workshop III: Data Assimilation, Uncertainty Reduction, and Optimization for Subsurface Flow, May 22-26, 2017

Outline

1) Short background in MRST
2) Obtaining gradients and sensitivities with adjoints using automatic differentiation
3) Example: valve model scaling sensitivities for optimizing valve distributions
4) Example: matching/analysing model parameters for the Sleipner CO2 injection case
5) Concluding remarks

MRST – Matlab Reservoir Simulation Toolbox

Originally:

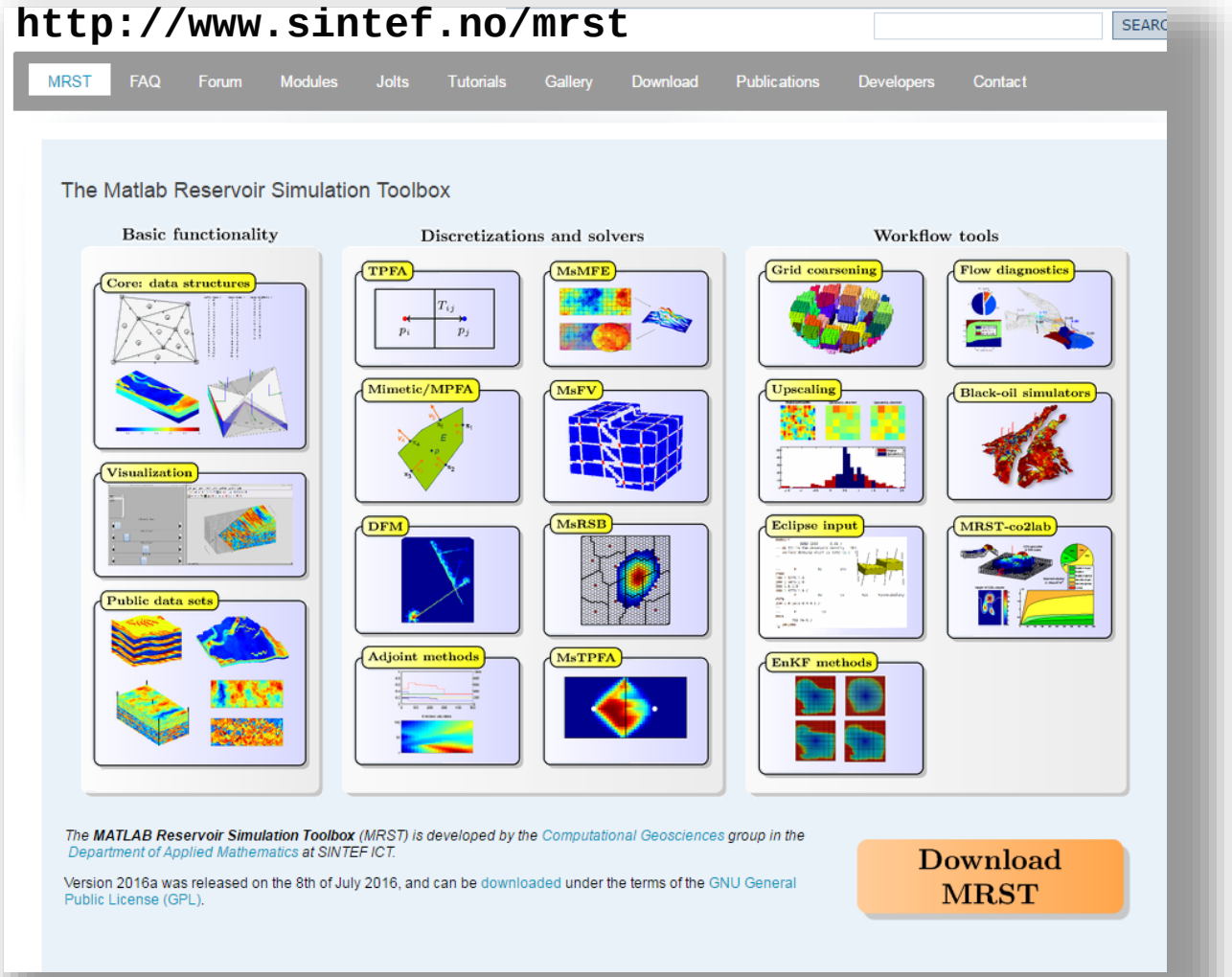
- developed to support research on *multiscale methods* and *discretization*
- first public release as open source, April 2009

Today:

- general toolbox for rapid prototyping and verification of new computational methods
- wide range of applications
- two releases per year each release has from 400 (R2012b) to 2100 (R2015b) unique downloads

Users:

- academic institutions, oil and service companies
- large user base in USA, Norway, China, Brazil, UK, Iran, Germany, Netherlands, France, Canada, …



SINTEF

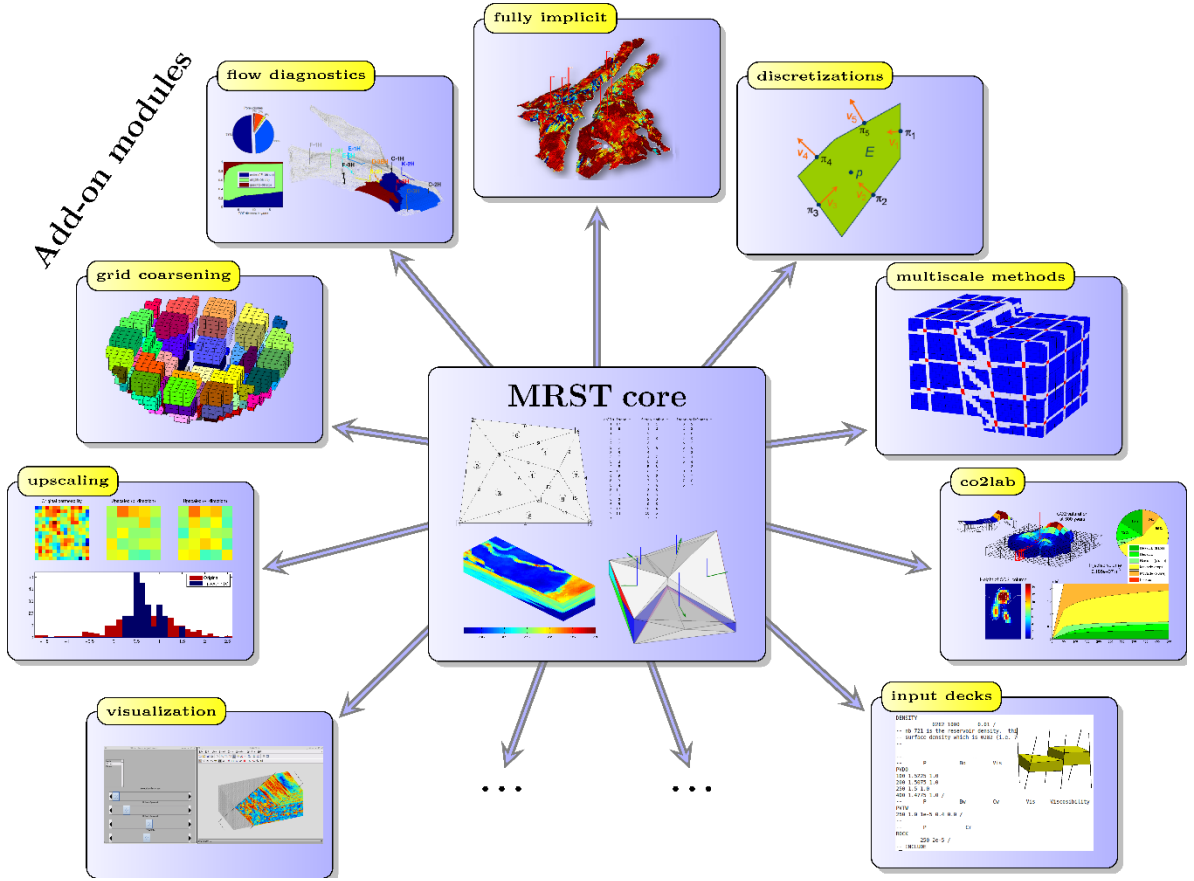
MRST – Matlab Reservoir Simulation Toolbox

Includes (release):

- fully implicit black-oil simulators based on AD with adjoint capabilities
- upscaling, grid coarsening and multiscale methods
- CO2-lab: modelling of CO2 storage (VE-models, optimization, visualization)
- EOR, geomechanics
- flow diagnostics
- input/output
- ++++

To appear:

- Compositional simulator
- Multisegment wells modelling



SINTEF

MRST – Accelerating the development cycle

Rapid prototyping:

- Focus on clean and simple implementation *close to the mathematics*

Key ingredients:

- ***hide specific details*** of grid and discretization
- ***vectorization and discrete operator representations,***
  - 1-to-1 between continuous and discrete
- ***unstructured grid format***
  - grid- independent implementation
- ***automatic differentiation (AD)***
  - no need to derive Jacobians by hand
  - maintainable *adjoint* code for computing gradients/sensitivities for optimization and analysis

equations

$$\vec{u} = -\mathbf{K}\nabla p$$
$$\nabla \cdot \vec{u} - q = 0$$

code

```
u = -T.*grad(p);
eq = div(u)-q;
```

SINTEF

Automatic differentiation in MRST

Need to work on *sub*-Jacobians rather than on full Jacobian

- An autodiff object contains a value (vector) and a ***list*** of derivatives (Jacobians).

Example: For primary variables given by vectors **x** and **y,** we have

```
      x = (x,{I,O})
      y = (y,{O,I})
      f = (f,{Fx,Fy})
   f.*g = (f.*g,{diag(g)Fx + diag(f)Gx,…
                 diag(g)Fy + diag(f)Gy })
```

All low-level autodiff class functions have double class function counterparts

High-level functions work for both *autodiffs* and *doubles,* e.g.,

```
equation(x,y) = residual
equation(x,y) = (residual, jacobian)
```

```
>> [x,y] = initVariablesADI([1 2]', [3 4]');
>> z = (x + 1).*y.^2

z =

  ADI with properties:

    val: [2×1 double]
    jac: {[2×2 double]  [2×2 double]}

>> z.val

ans =

    18
    48

>> full(z.jac{1})

ans =

     9     0
     0    16

>> full(z.jac{2})

ans =

    12     0
     0    24
```


SINTEF

Automatic differentiation in MRST

Forward model:

previous state
__unknown__ current state
current controls
model parameters

$$F_n(\mathbf{x}^{n-1}, \mathbf{x}^n, \mathbf{u}^n, \mathbf{m}) = 0, \quad n = 1, \ldots, N$$

Unknown state found with Newton:

$$\mathbf{x}^{n,\nu+1} \leftarrow \mathbf{x}^{n,\nu} + \delta\mathbf{x}^{n,\nu}$$

$$\frac{\partial F_n(\mathbf{x}^{n-1}, \mathbf{x}^{n,\nu}, \mathbf{u}^n, \mathbf{m})}{\partial \mathbf{x}^{n,\nu}} \delta\mathbf{x}^{n,\nu} = -F_n(\mathbf{x}^{n-1}, \mathbf{x}^{n,\nu}, \mathbf{u}^n, \mathbf{m})$$

Slight abuse of notation

$$\frac{\partial F_n(\mathbf{x}^{n-1}, \mathbf{x}^{n,\nu}, \ldots)}{\partial \mathbf{x}^{n,\nu}} := \left.\frac{\partial F_n(\mathbf{x}^{n-1}, \mathbf{x}, \ldots)}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}^{n,\nu}}$$

Approximate state initialized as AD:

$$F_n(\mathbf{x}^{n-1}, \mathbf{x}^{n,\nu}, \mathbf{u}^n, \mathbf{m}) \Rightarrow \left(F_n(\mathbf{x}^{n-1}, \mathbf{x}^{n,\nu}, \mathbf{u}^n, \mathbf{m}), \frac{\partial F_n}{\partial \mathbf{x}^{n,\nu}} \right)$$

residual

Jacobian

SINTEF

Adjoint equations for finding gradients wrt controls

Forward equations for n = 1, …,N:

$$F_n(\mathbf{x}^{n-1},\mathbf{x}^n,\mathbf{u}^n,\mathbf{m})=0, \quad n=1,\ldots,N$$

Objective:

$$J=\sum J_n(\mathbf{x}^n,\mathbf{u}^n)$$

Adjoint equations for n = N,…,1:

$$\left(\boxed{\frac{\partial F_n}{\partial \mathbf{x}^n}}\right)^T \boldsymbol{\lambda}_n = -\left(\frac{\partial J_n}{\partial \mathbf{x}^n}\right)^T - \left(\boxed{\frac{\partial F_{n+1}}{\partial \mathbf{x}^n}}\right)^T \boldsymbol{\lambda}^{n+1}$$

Gradient:

$$\nabla_{\mathbf{u}^n} J^n = \left(\frac{\partial J^n}{\partial \mathbf{u}^n}\right)^T + \left(\frac{\partial F_n}{\partial \mathbf{u}^n}\right)^T \boldsymbol{\lambda}^n$$

Main implementation challenge:

- Bug-free and maintainable code for computing partial derivatives.

p: previous
c: current

```
function eq = F(xp, xc, uc, m)
if forward
      xc  = initAD(xc)
elseif reverse
      xp = initAD(xp)
end
…
```


SINTEF

Implementation details:

Choice of primary variables and formulation of equations are such that:

- Controls appear explicitly and only in control equations.

$$\nabla_{\mathbf{u}^n} J^n = \left(\frac{\partial J^n}{\partial \mathbf{u}^n}\right)^T + \left(\boxed{\frac{\partial F_n}{\partial \mathbf{u}^n}}\right)^T \boldsymbol{\lambda}^n$$

$$\begin{pmatrix} \mathrm{O} \\ \mathrm{O} \\ \vdots \\ \mathrm{O} \\ \mathrm{I} \end{pmatrix}$$

Ex: bottom-hole pressure control

```
contrEq = u(i)-bhp(i)
```

Ex: liquid rate control

```
contrEq = u(i)-qws(i)-qos(i)
```


SINTEF

Adjoint equations for finding gradients wrt parameters

Forward equations for n = 1, …,N:

$$F_n(\mathbf{x}^{n-1},\mathbf{x}^n,\mathbf{u}^n,\mathbf{m})=0$$

Objective:

$$J=\sum J_n(\mathbf{x}^n,\mathbf{m})$$

Adjoint equations for n = N,…,1:

$$\left(\frac{\partial F_n}{\partial \mathbf{x}^n}\right)^T \boldsymbol{\lambda}_n = -\left(\frac{\partial J_n}{\partial \mathbf{x}^n}\right)^T - \left(\frac{\partial F_{n+1}}{\partial \mathbf{x}^n}\right)^T \boldsymbol{\lambda}^{n+1}$$

Gradient/sensitivities:

$$\nabla_{\mathbf{m}} J = \left(\frac{\partial J}{\partial \mathbf{m}}\right)^T + \sum_{n=1}^{N}\left(\boxed{\frac{\partial F_n}{\partial \mathbf{m}}}\right)^T \boldsymbol{\lambda}^n$$

Have considered two equivalent implementations:

- Add parameters as *primary variables* in equations
- Keep equations unchanged and compute $\frac{\partial F_n}{\partial \mathbf{m}}$ directly by initializing **m** to AD and calling *F:*

$$F_n(\mathbf{x}^{n-1},\mathbf{x}^n,\mathbf{u}^n,\mathbf{m}) \Rightarrow \left(F_n(\mathbf{x}^{n-1},\mathbf{x}^n,\mathbf{u}^n,\mathbf{m}), \frac{\partial F_n}{\partial \mathbf{m}}\right)$$

Adjoint equations for finding gradients wrt parameters

Add parameters to equations:

Example: obtaining sensitivities of a vector of transmissibility multipliers **m**:

```
function eqs = getEquations(...)
   [p,sw,...,mvar] = initAD(p,sw,...,m);
   trans = mvar.*trans;
   eqs{1} = ...
       ⋮
   eqs{n} = ...
   eqs{n+1} = m - mvar;
end
```

$$\nabla_{\mathbf{m}} J = \left(\frac{\partial J}{\partial \mathbf{m}}\right)^T + \sum_{n=1}^{N} \left(\boxed{\frac{\partial F_n}{\partial \mathbf{m}}}\right)^T \boldsymbol{\lambda}^n \qquad \begin{pmatrix} \mathrm{O} \\ \mathrm{O} \\ \vdots \\ \mathrm{O} \\ \mathrm{I} \end{pmatrix}$$

Keeping equations unchanged:

- Straightforward for parameters appearing *explicitly* in equations, e.g., transmissibility, well connection factors, pore volumes, …
- Slightly more work required for other parameters, e.g., permeability, fluid parameters, …
- Note that each adjoint simulation step requires three function evaluation calls:

$$F_n(\mathbf{x}^{n-1}, \mathbf{x}^n, \mathbf{u}^n, \mathbf{m}) \Rightarrow \left(F_n(\mathbf{x}^{n-1}, \mathbf{x}^n, \mathbf{u}^n, \mathbf{m}), \frac{\partial F_n}{\partial \mathbf{x}^n} \right)$$

$$F_{n+1}(\mathbf{x}^n, \mathbf{x}^{n+1}, \mathbf{u}^{n+1}, \mathbf{m}) \Rightarrow \left(F_{n+1}(\mathbf{x}^n, \mathbf{x}^{n+1}, \mathbf{u}^{n+1}, \mathbf{m}), \frac{\partial F_{n+1}}{\partial \mathbf{x}^n} \right)$$

$$F_n(\mathbf{x}^{n-1}, \mathbf{x}^n, \mathbf{u}^n, \mathbf{m}) \Rightarrow \left(F_n(\mathbf{x}^{n-1}, \mathbf{x}^n, \mathbf{u}^n, \mathbf{m}), \frac{\partial F_n}{\partial \mathbf{m}} \right)$$

SINTEF

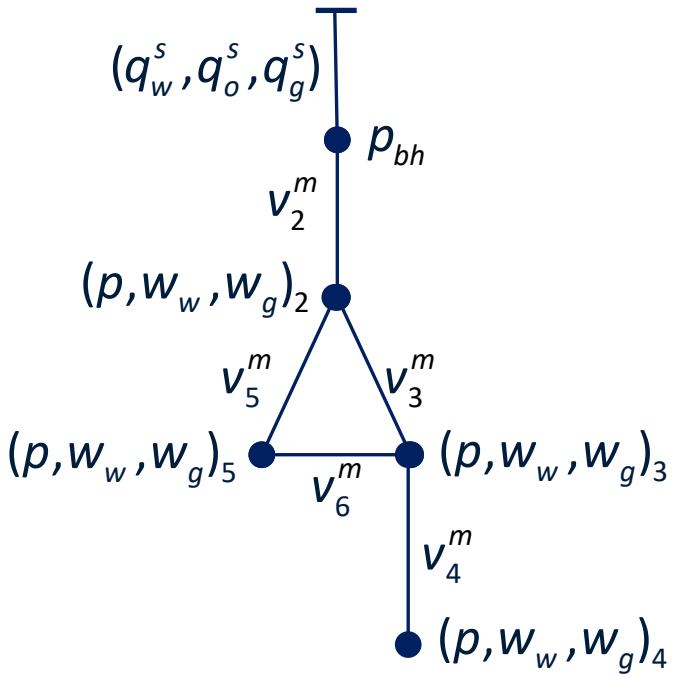
Example: valve scaling for multi-segment wells

Standard wells

- Primary variables: bottom-hole-pressure, component rates
- Instantaneous flow along wellbore
- Explicit treatment of pressure along wellbore

Multi-segment wells (in MRST)

- General topology network (graph)
- Primary variables: pressure and mass fractions at nodes, total mass rates at edges
- Component mass balance at nodes
- General pressure drop relation along edges (h)



$$\text{topo} = \begin{pmatrix} 0 & 1 \\ 1 & 2 \\ 2 & 3 \\ 3 & 4 \\ 2 & 5 \\ 5 & 3 \end{pmatrix}$$

Discrete equations:

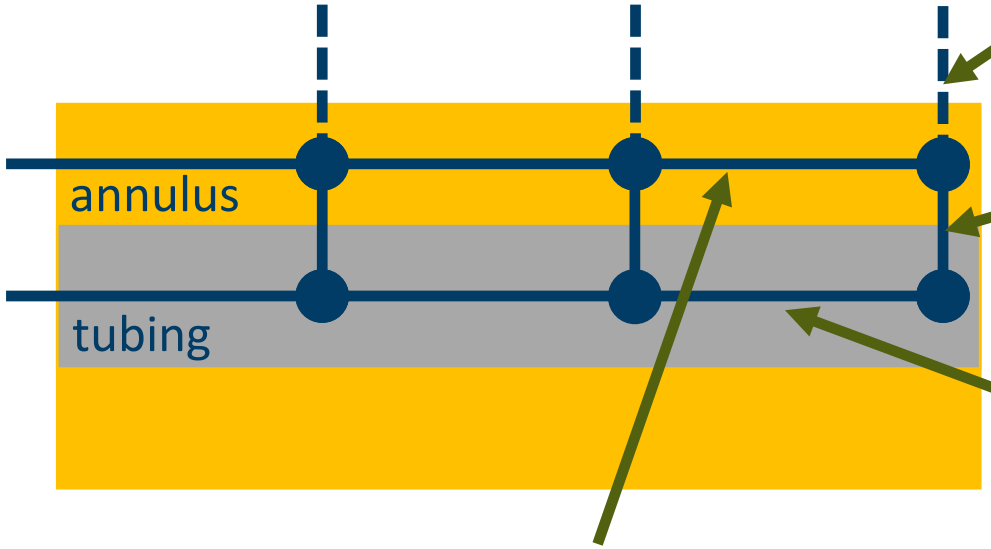
$$\frac{\mathbf{V}}{\Delta t}\left(\mathbf{w}_c\boldsymbol{\rho} - \mathbf{w}_c^0\boldsymbol{\rho}^0\right) + \operatorname{div}(\mathbf{v}_c^m) - \mathbf{q}_c^m = \mathbf{0}$$

$$\operatorname{grad}(\mathbf{p}) - g\ \operatorname{avg}(\boldsymbol{\rho})\operatorname{grad}(\mathbf{z}) - h(\mathbf{v}^m, \mathbf{p}, \mathbf{w}) = \mathbf{0}.$$

SINTEF

Example: valve scaling for multi-segment wells

Model with annular flow and valves between annulus and tubing



Connection:

$$q_p = \lambda_p T(\Delta p + g\rho\Delta z)$$

Valve model (next slide)

Frictional pressure drop along tubing:

$$\Delta p = 2f\frac{L}{D}\rho u^2$$

$$\sqrt{\frac{1}{f}} = -3.6\log_{10}\left(\frac{6.9}{\mathrm{Re}} + \left(\frac{e}{3.7D}\right)^{10/9}\right)$$

Annular flow:

- Frictional pressure drop if open
- Darcy flow if gravel
- No flow if packed

SINTEF

Example: valve scaling for multi-segment wells

Here we consider two valve models:

1. Nozzle type valve:

$$\Delta p = \frac{\rho_{mix} v_c^2}{2C_v}$$

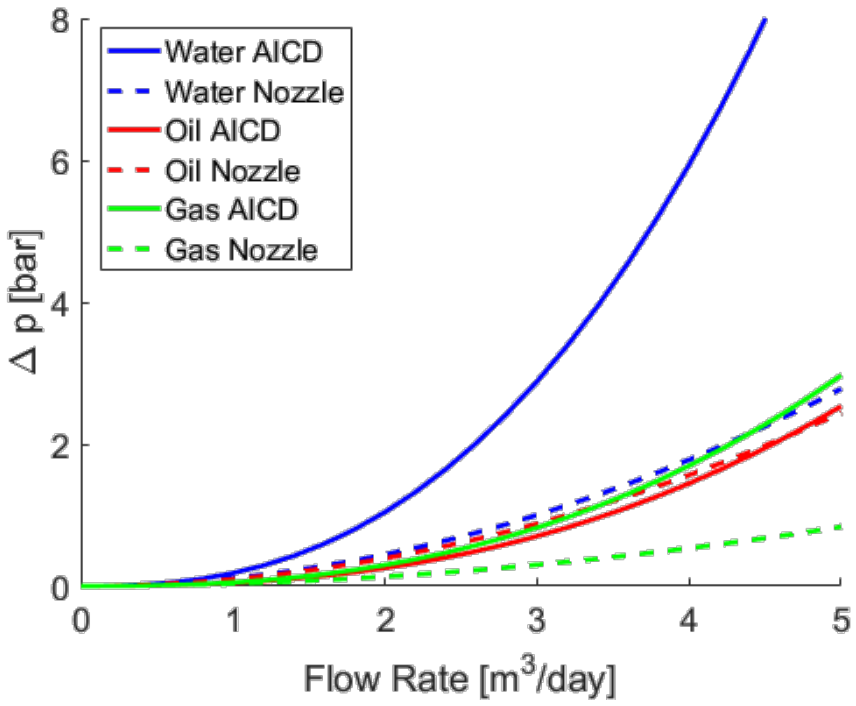
$$\rho_{mix} = \alpha_o \rho_o + \alpha_w \rho_w + \alpha_g \rho_g$$

2. AICD (autonomous inflow control device)

$$\Delta p = \left(\frac{\rho_{mix}^2}{\rho_{cal}}\right)\left(\frac{\mu_{cal}}{\mu_{mix}}\right)^y a_{AICD} q^x$$

$$\rho_{mix} = \alpha_o^a \rho_o + \alpha_w^b \rho_w + \alpha_g^c \rho_g$$

$$\mu_{mix} = \alpha_o^d \mu_o + \alpha_w^e \mu_w + \alpha_g^f \mu_g$$



- Water: 1000 kg/m^3, 0.3cp
- Oil: 879 kg/m^3, 3cp
- Gas: 300 kg/m^3, 0.03cp

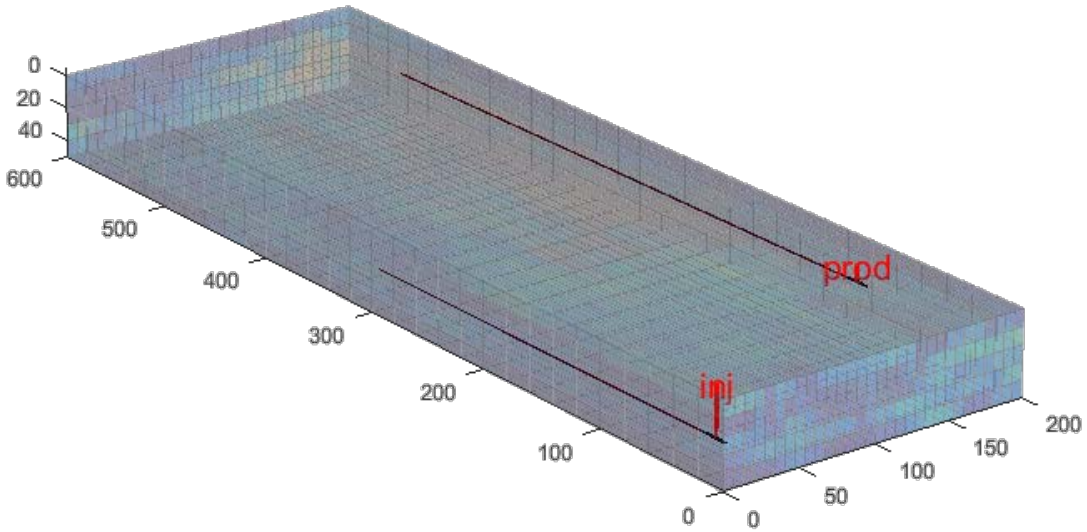
AICD Model parameters from J. Videla, Statoil

SINTEF

Example: valve scaling for multi-segment wells

Model setup:

- Producer set at constant oil rate 200 m^3/day
- Inject gas at 250 bar
- Run for 1000 days
- No annular flow
- Distribute valves evenly 0.1 per meter tubing length (2 valves per connecting grid cell).



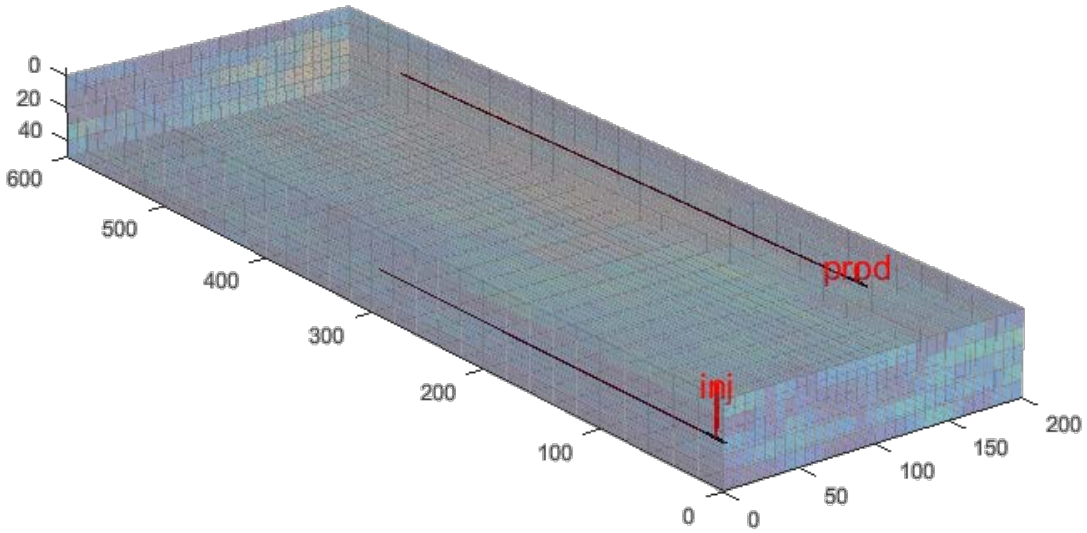
Task: find optimal distribution of valves to minimize gas production.

SINTEF

Example: valve scaling for multi-segment wells

Consider a valve model $\Delta p = h(v, \ldots)$

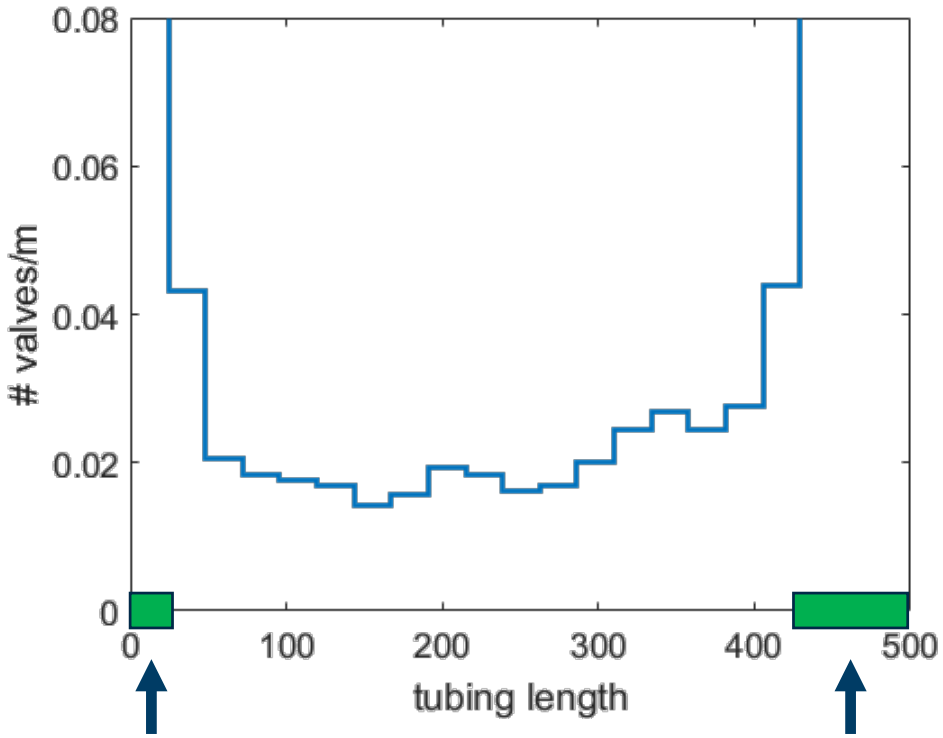
- Each discrete valve connection may represent more than one valve, hence we introduce a scaling parameter c, such that $\Delta p = h(cv, \ldots)$
  - a connection representing two valves results in c=0.5
- By adjoints we can compute the sensitivities/gradients of total gas production wrt all scaling parameters.
- Use sensitivities to find *optimal* parameters for both the nozzle type valves and the AICDs.



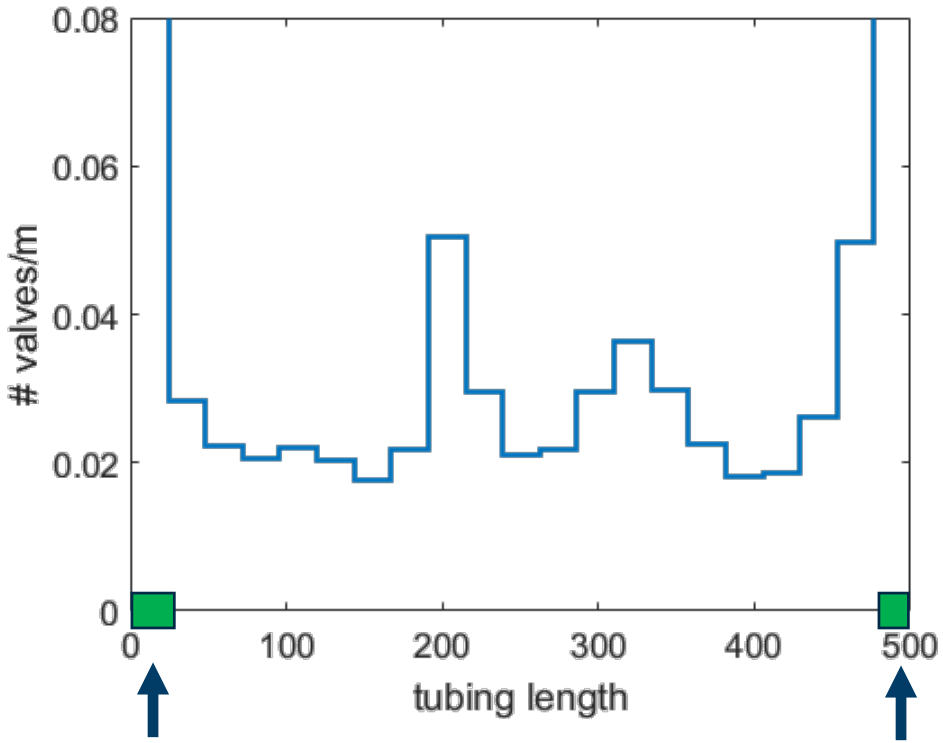
SINTEF

Example: valve scaling for multi-segment wells

Optimal distribution: nozzle valves



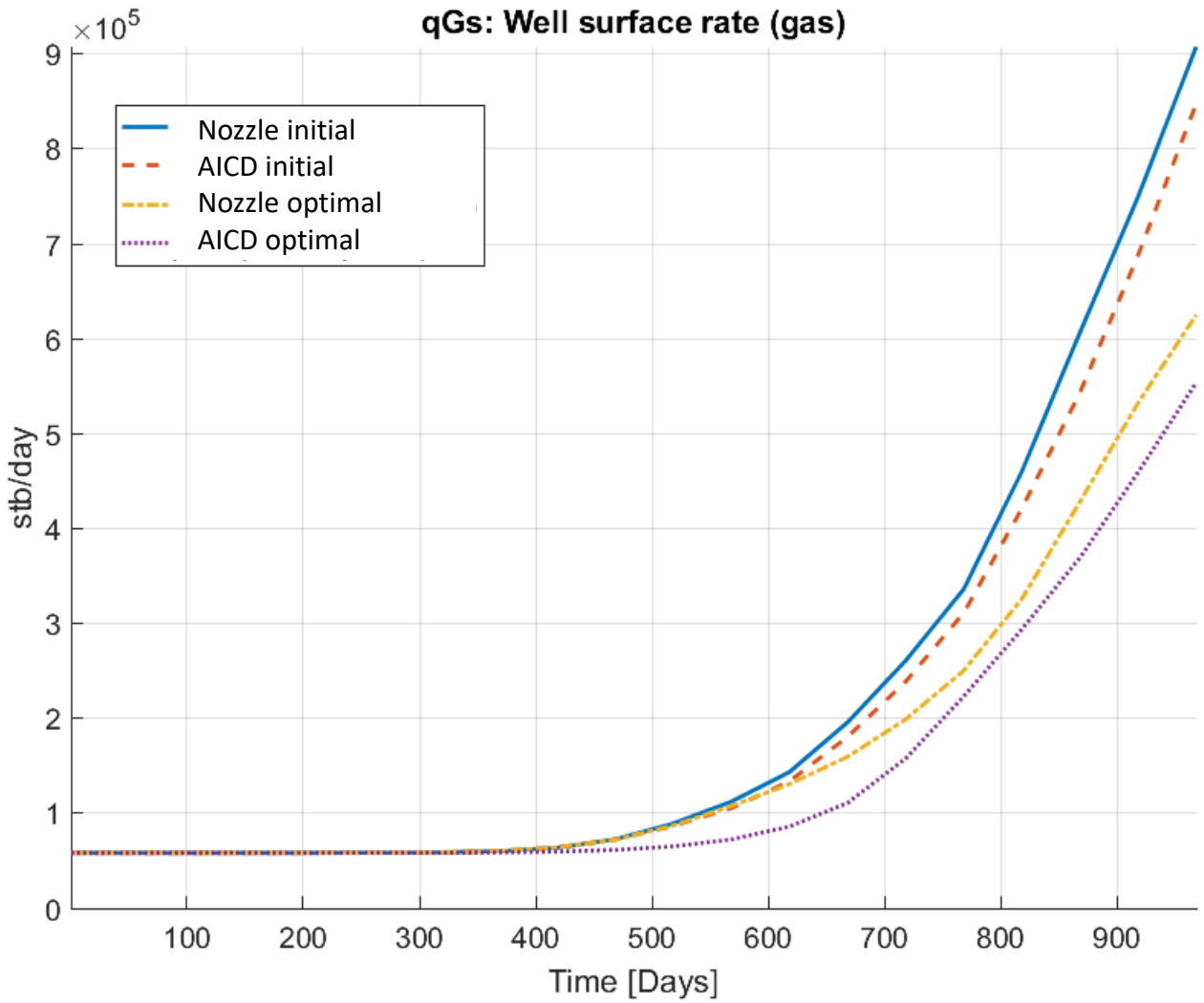
Optimal distribution: AICDs



At heel and toe, flow should not be restricted by valves

SINTEF

Example: valve scaling for multi-segment wells



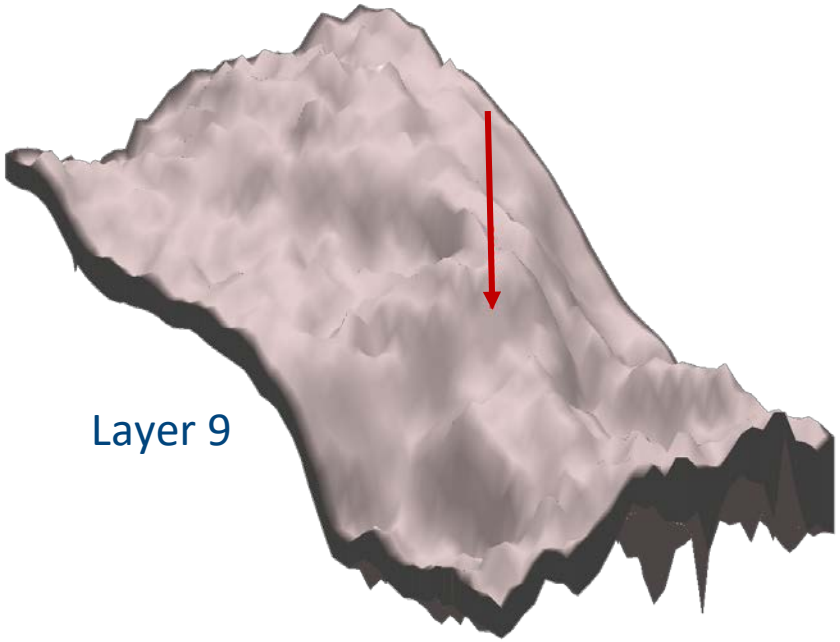
SINTEF

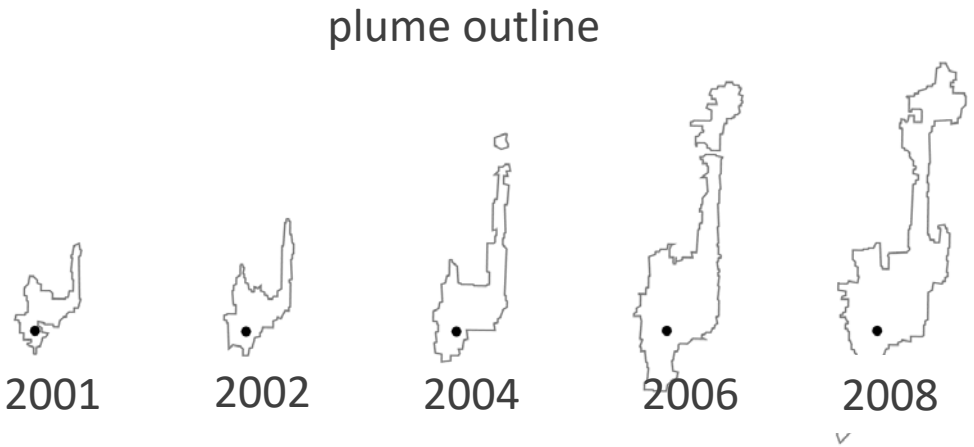
Matching model parameters for the Sleipner CO2 injection case

- Ongoing carbon capture & storage project
- Model made available by IEAGHG as the Sleipner benchmark.

Here

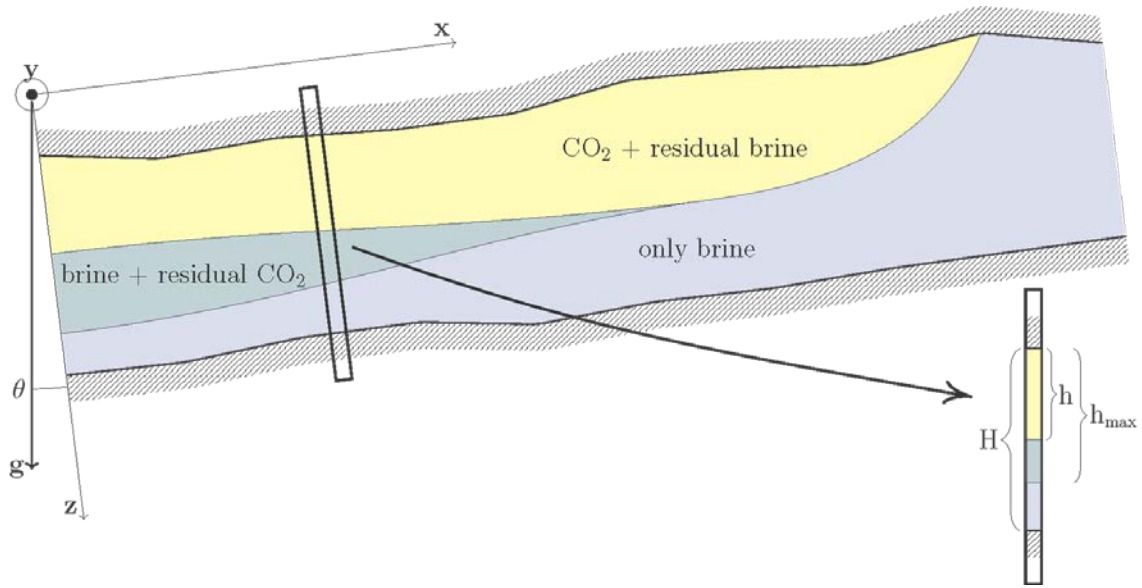
- Use vertical equilibrium (VE) for fast simulation of plume heights
- Use adjoint-based sensitivities to adjust combination of geometry and physical parameters to obtain a better match between simulated and *observed* plume heights.





SINTEF

Matching model parameters for the Sleipner CO2 injection case



$$\frac{\partial(\phi\rho_\alpha s_\alpha)}{\partial t} + \nabla\cdot\rho_\alpha\vec{u}_\alpha = \rho_\alpha q_\alpha,$$

$$\vec{u}_\alpha = -k\lambda_\alpha(\nabla p_\alpha - \rho_\alpha\vec{g}),$$

$$s_w + s_g = 1,$$

$$\lambda_\alpha = \lambda_\alpha(s_w),$$

$$p_g = p_w - P_c(s_w)$$

Vertical equilibrium flow equations:

$$\vec{u}_t = -\lambda_t(h)k\Big(\nabla p_i - \big[\rho_g f_g(h) + \rho_w f_w(H-h)\big]g\nabla(z_t + h)\Big)$$

$$= -\lambda_t(h)k\Big(\nabla p_i - \big[\rho_w - \Delta\rho f_g(h)\big]g\nabla(z_t + h)\Big),$$

$$\nabla\vec{u}_t = q_t,$$

$$\phi\frac{\partial h}{\partial t} + \nabla f(h)\Big(\vec{u}_t - k\Delta\rho\lambda_g(h)\lambda_w(H-h)g\nabla(z_t + h)\Big) = q_g.$$

SINTEF

Matching model parameters for the Sleipner CO2 injection case

Consider the following set of parameters

1. Top surface height adjustments for each grid cell
2. Scalar multipliers for rate, CO2 density, permeability and porosity

$$\mathbf{m} = \{\mathbf{dz}, m_q, m_\rho, m_k, m_\phi\}$$

Aim: match simulated plume heights to observed heights at times m = 1,2,…

$$J = \sum J_m$$

$$J = \sum_{i=1}^{N} V_i (h_i^m - h_{i,obs}^m)^2$$

Published seismic interpretations:
Singh2010, Chadwick2010, Furre2014

Explore objective invariant subspaces by equation manipulation **or** finding the null-space of the Hessian:

$$H(\mathbf{m}) = d^2 J / d\mathbf{m}^2$$

Exact invariant subspace:

$$dm_1 = \text{span}([\mathbf{0}^T, 1, 0, 1, 1]^T)$$

Invariant subspace in the *incompressible limit*

$$dm_2 = \text{span}([\mathbf{0}^T, 0, 1, \frac{\rho_w}{\rho_g} - 1, 1]^T)$$

Hence, any perturbation in the direction of dm_1 and dm_2 has no and *little* effect, respectively.

SINTEF

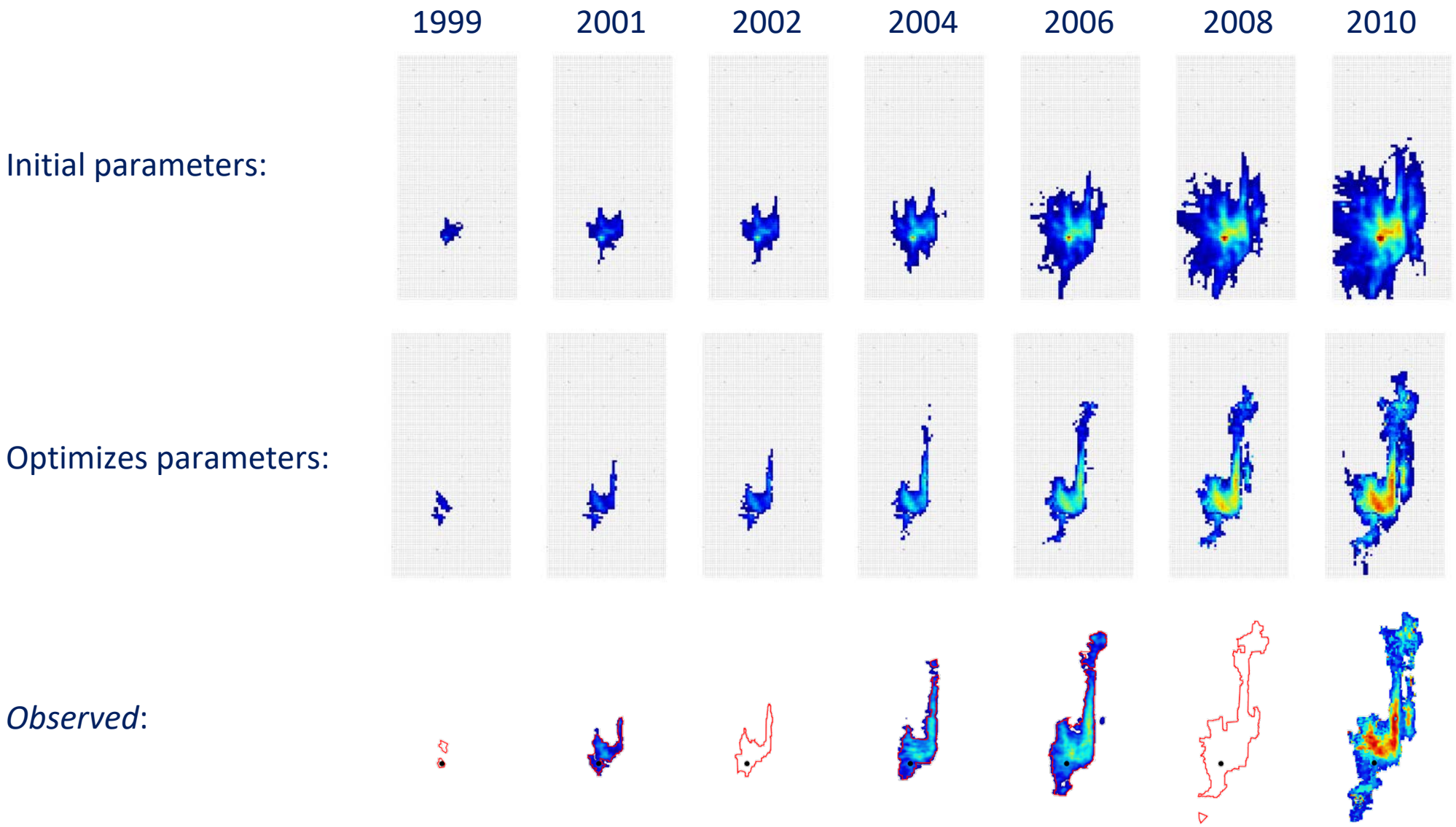
Matching model parameters for the Sleipner CO2 injection case

Some parameter combinations giving equally good matches:

	q_3			d_3		d_4	
s	-0.5	0	0.5	-0.5	0.5	-0.5	0.5
rate multiplier, m_p	0.59	0.92	1.25	0.92	0.92	1.19	0.65
density (kg/m^3)	565	478	391	657	298	478	478
permeability (darcy)	10.2	12.7	15.1	16.8	8.5	16.3	9.0
porosity	0.24	0.37	0.51	0.37	0.37	0.48	0.27
thermal gradient $(°C/km)$	34.9	35.4	35.8	32.8	38.5	35.4	35.4
$k\Delta\rho$ (kg/m)	4.58×10^{-9}	6.79×10^{-9}	9.37×10^{-9}	6.02×10^{-9}	6.04×10^{-9}	8.71×10^{-9}	4.83×10^{-9}

SINTEF

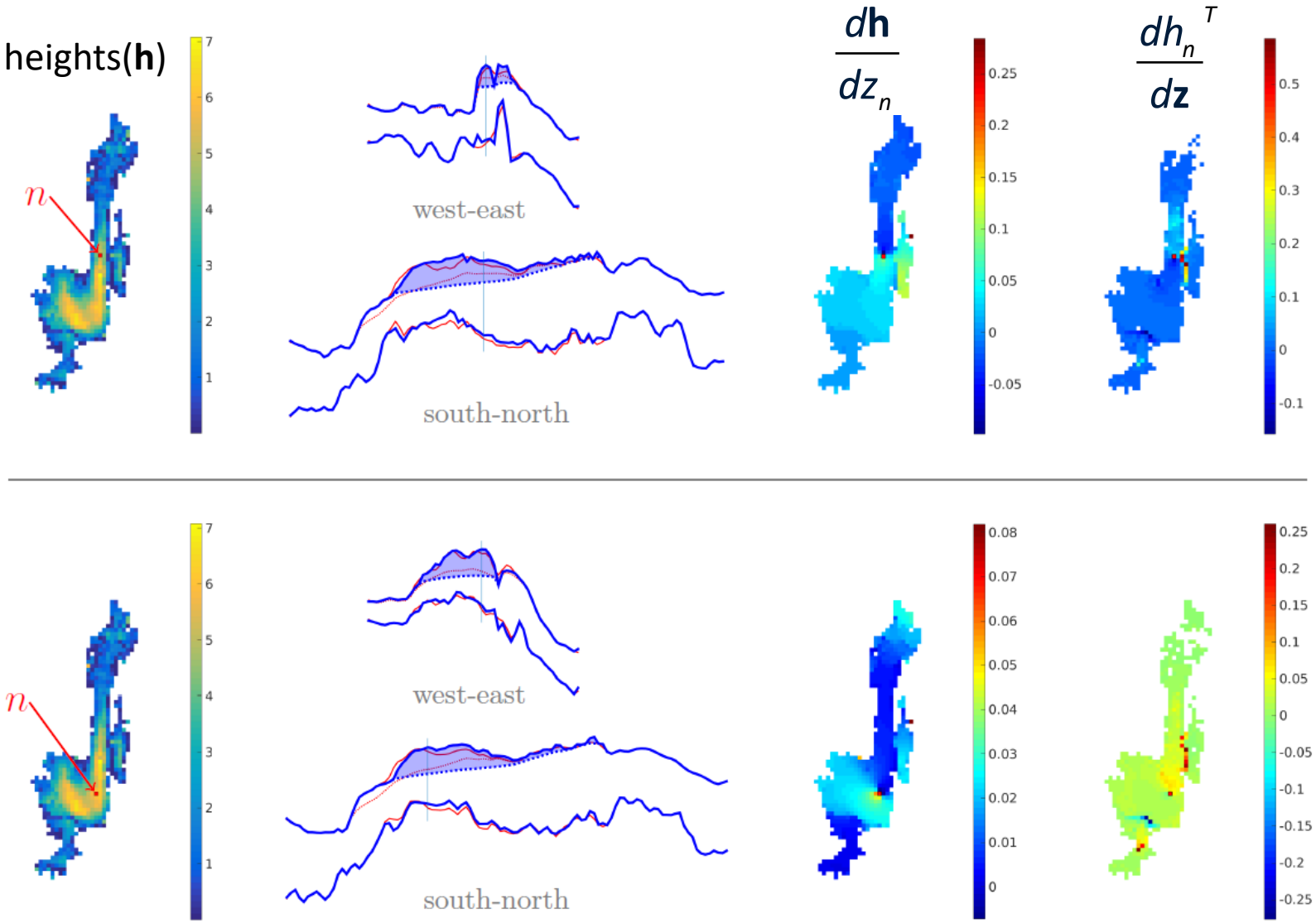
Matching model parameters for the Sleipner CO2 injection case



SINTEF

Matching model parameters for the Sleipner CO2 injection case

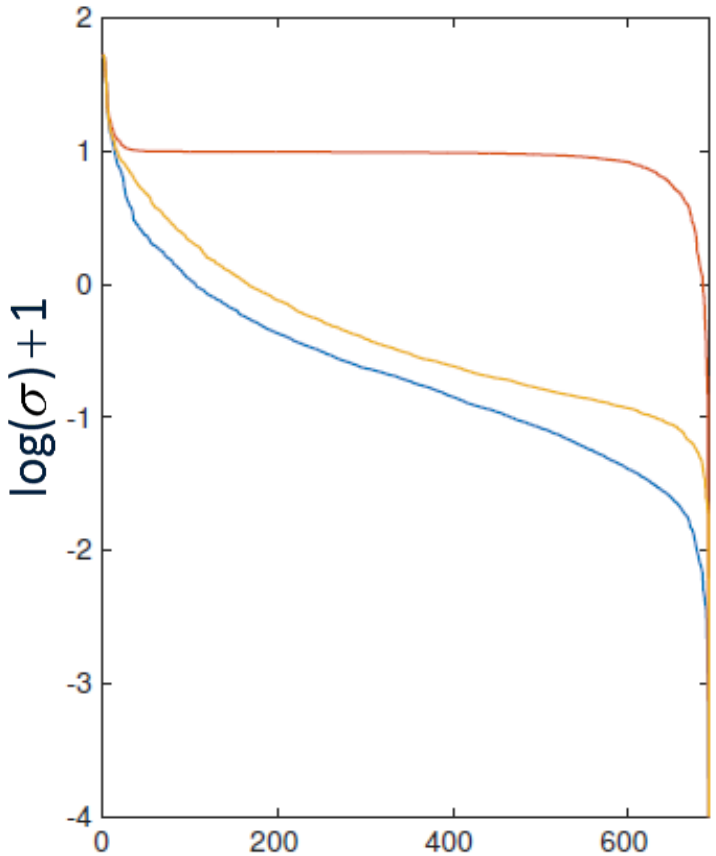
Can also compute top-surface depth to plume height sensitivities at given locations



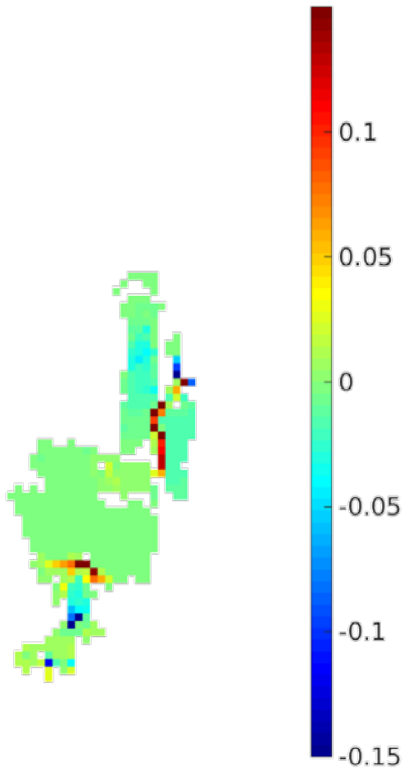
SINTEF

Matching model parameters for the Sleipner CO2 injection case

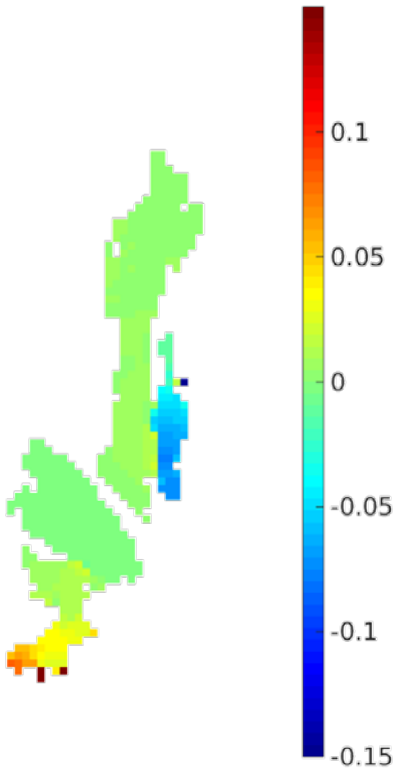
Analysis of the full sensitivity matrix $A = d\mathbf{h}/d\mathbf{z}$.



singular values for the matrices *A*, *A-I* and *A*-diag(*A*)



Right singular vector for largest singular value for *A-I* (*most influential dz*)



Left singular vector for largest singular value for *A-I* (corresponding response)

Concluding remarks

- Demonstrated adjoint capabilities for computing parameter sensitivities in MRST
  - By automatic differentiation and minor code organization choices, the adjoint code comes out *almost* for free.
- Illustrated using two examples:
  - Distribution of wells along horizonalt well
  - Parameter estimation for the Sleipner benckmark

SINTEF