Nonlinear solvers in OPM Flow

Atgeirr Flø Rasmussen

SINTEF Digital, Mathematics and Cybernetics

SIAM GS 2017
12th September 2017

Overview of talk

Introduction

Mathematical formulation

Solving the nonlinear equations

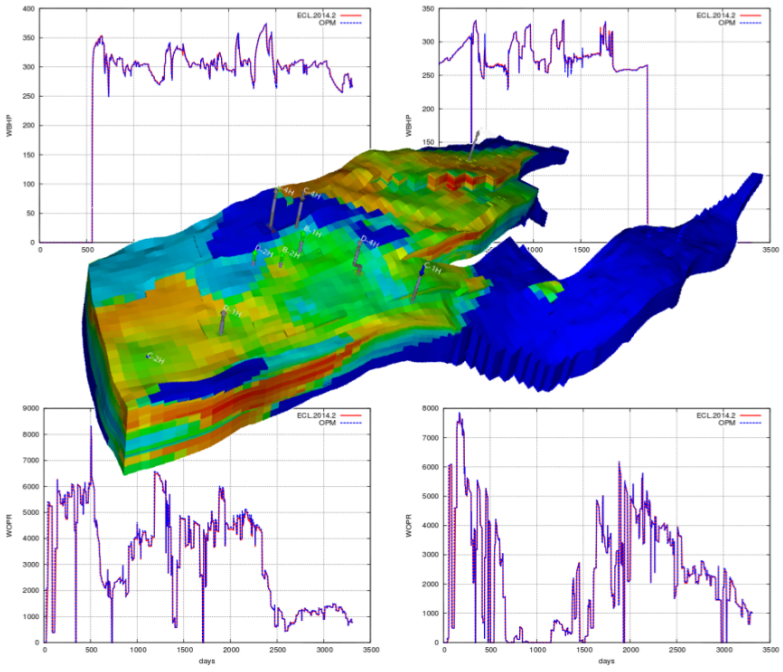
The reordering approach

Numerical example

Conclusion

OPM Flow at a glance

- Open source
- Competitive performance
- Full industrial complexity
  - Well controls
  - EOR: CO2, polymer
  - CO2 sequestration
- Automatic differentiation (AD)



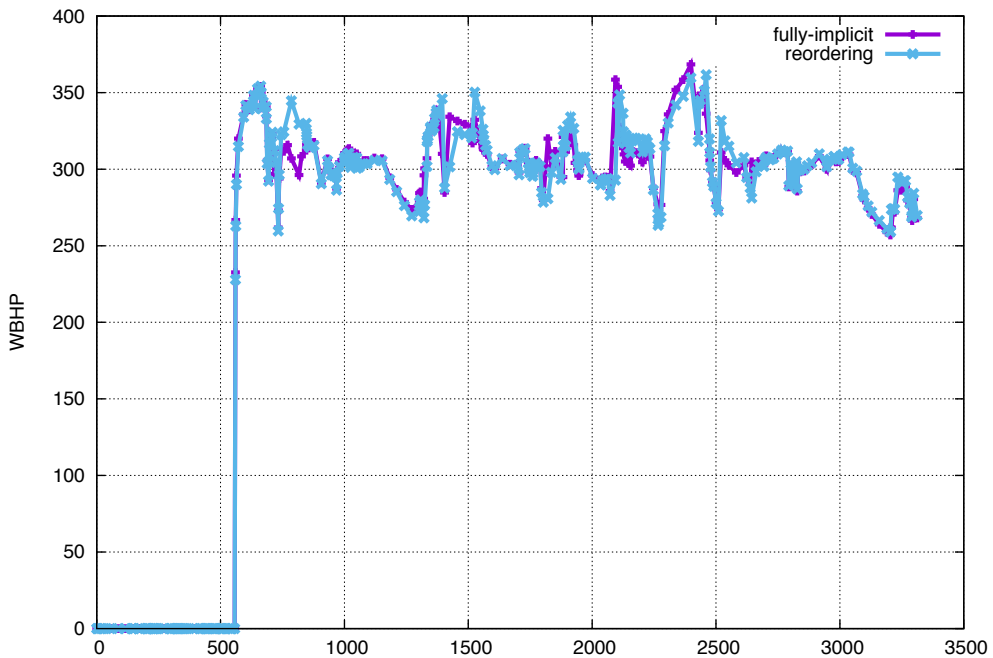
Ambition: to be a strong base for both industrial development and academic research

Main idea of this talk

We can gain performance by

- sequential splitting,
- reordering solvers,
- (nonlinear preconditioners)

... without losing the ability to run *industrial full field models*.



Case: Norne
Well: C-3H
Plot: BHP

The black-oil model, physical laws

"Black-oil" model assumptions

Lump hydrocarbon species into two pseudo-components (oil, gas)

Fluid Phase	Pseudo-component Water	Oil	Gas
Aqueous	x		
Oleic		x	x
Gaseous		x	x

(Subset of more general *compositional* model)

Conservation of mass (per component α)

$$\frac{\partial}{\partial t}\left(\phi A_\alpha\right) + \nabla \cdot \mathbf{u}_\alpha = Q_\alpha$$

Darcy's law (per phase β)

$$\mathbf{v}_\beta = -(k_{r,\beta}/\mu_\beta)\mathbf{K}(\nabla p_\beta - \rho_\beta \mathbf{g})$$

System of equations (fully implicit)

System of PDEs, one for each pseudo-component α:

$$\frac{\partial}{\partial t}\left(\phi A_\alpha\right) + \nabla \cdot \mathbf{u}_\alpha = Q_\alpha$$

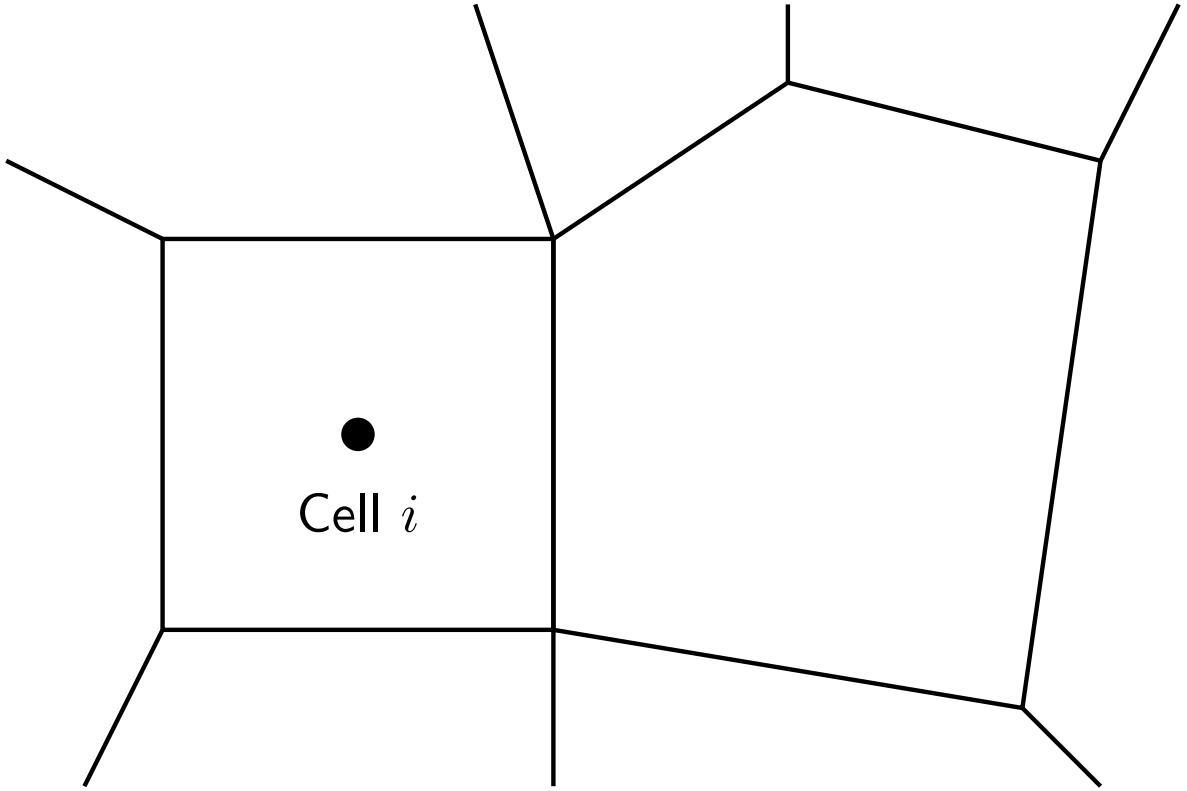
where

$$\begin{aligned} A_w &= b_w s_w, & \mathbf{u}_w &= b_w \mathbf{v}_w, \\ A_o &= b_o s_o + r_V b_g s_g, & \mathbf{u}_o &= b_o \mathbf{v}_o + r_V b_g \mathbf{v}_g, \\ A_g &= b_g s_g + r_S b_o s_o, & \mathbf{u}_g &= b_g \mathbf{v}_g + r_S b_o \mathbf{v}_o, \end{aligned}$$

and also:

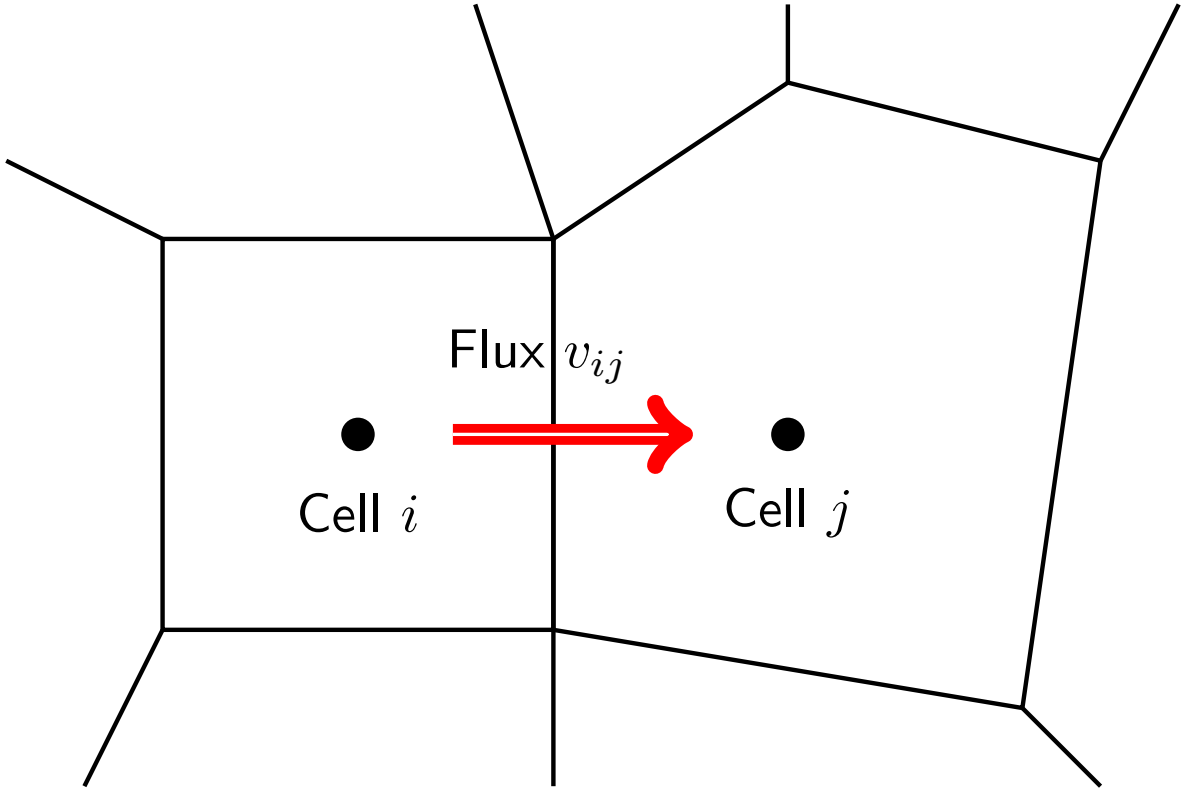
$$\begin{aligned} s_w + s_o + s_g &= 1 \\ p_o - p_w &= p_{cow} \\ p_o - p_g &= p_{cog}. \end{aligned}$$

Discretization (fully implicit)



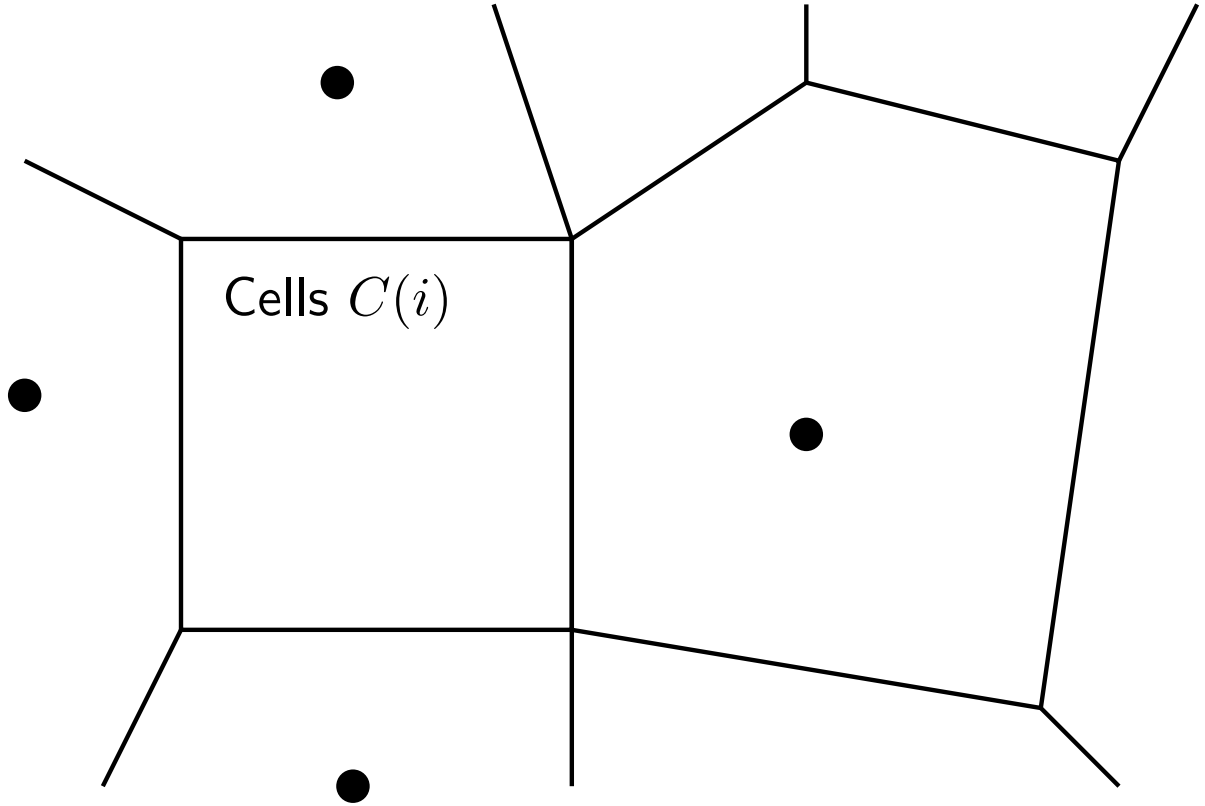
Notation:
Cell value: x_i or x_j
Connection value: x_{ij}

Discretization (fully implicit)



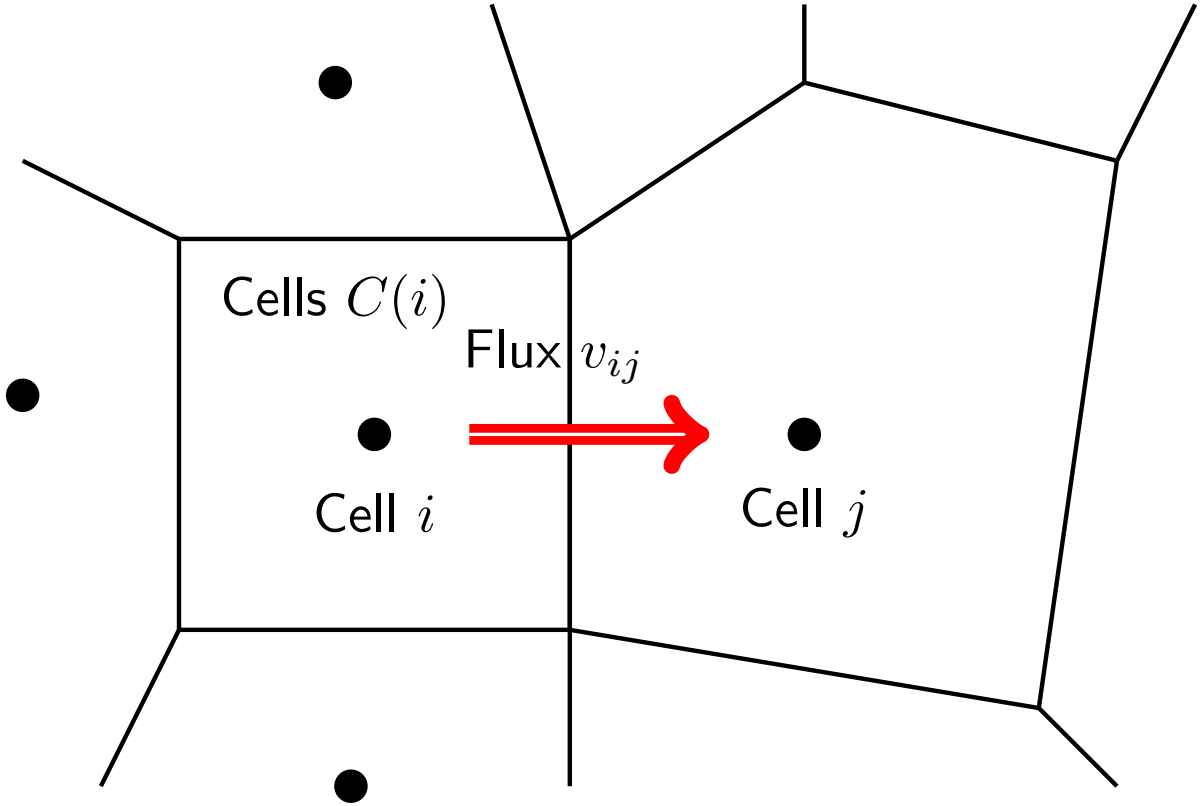
Notation:
Cell value: x_i or x_j
Connection value: x_{ij}

Discretization (fully implicit)



Notation:
Cell value: x_i or x_j
Connection value: x_{ij}

Discretization (fully implicit)



Notation:
Cell value: x_i or x_j
Connection value: x_{ij}

Discrete material balance

$$R_{\alpha,i} = \frac{\phi_i V_i}{\Delta t}\left(A_{\alpha,i} - A^0_{\alpha,i}\right) + \sum_{j \in C(i)} u_{\alpha,ij} - Q_{\alpha,i} = 0$$

Sequential implicit equations

Pressure equation: linear combination to eliminate saturation dep.

$$R_p = \sum_\alpha \sigma_\alpha R_\alpha = 0.$$

Store $v_{T,ij} = \sum_\alpha v_{\alpha,ij}$ for transport solver.

Transport equations

$$R_o = 0, \qquad R_g = 0,$$

with fluxes derived from v_T:

$$(b_\alpha v_\alpha)_{ij} = b_{\alpha,ij} \frac{\lambda_{\alpha,ij}}{\sum_\beta \lambda_{\beta,ij}} (v_T + T_{ij} G_{ij}).$$

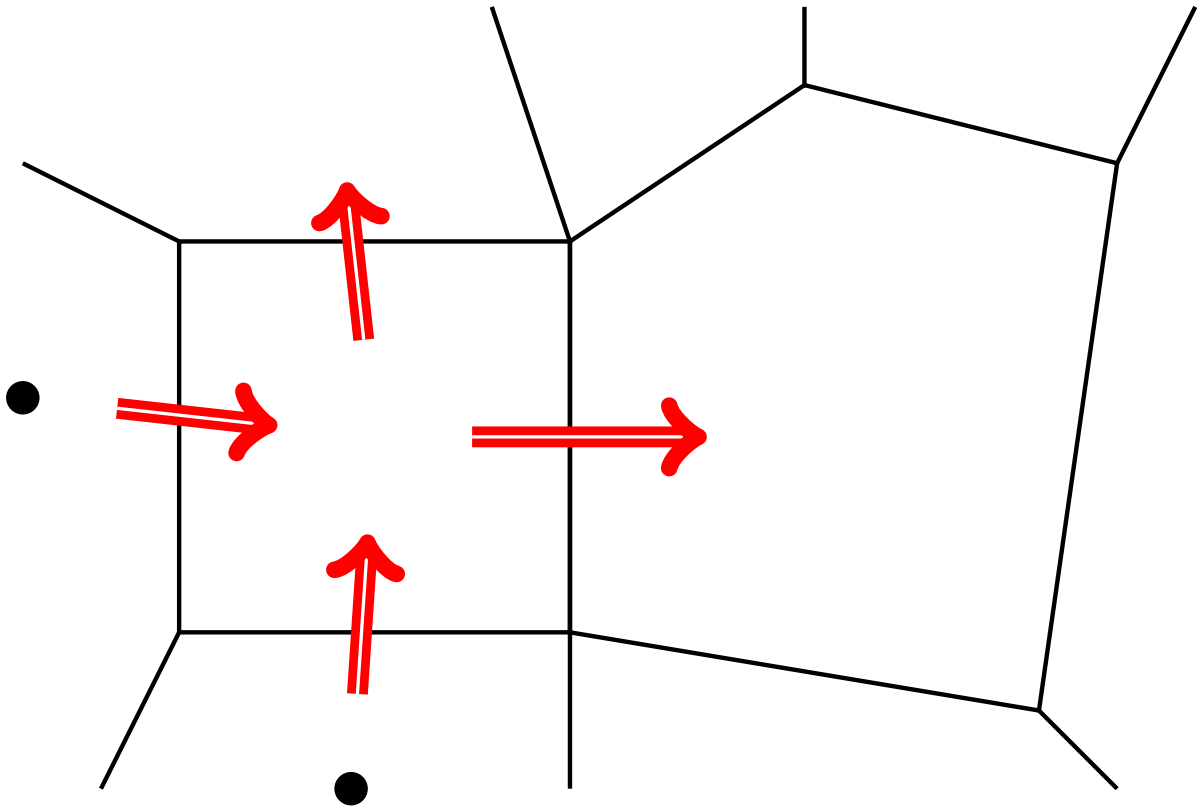
Upwind b_{ij}, λ_{ij} and gravity term G_{ij} following Brenier and Jaffré "Upstream Differencing for Multiphase Flow in Reservoir Simulation"

The reordering idea

Advection-dominated transport problems:

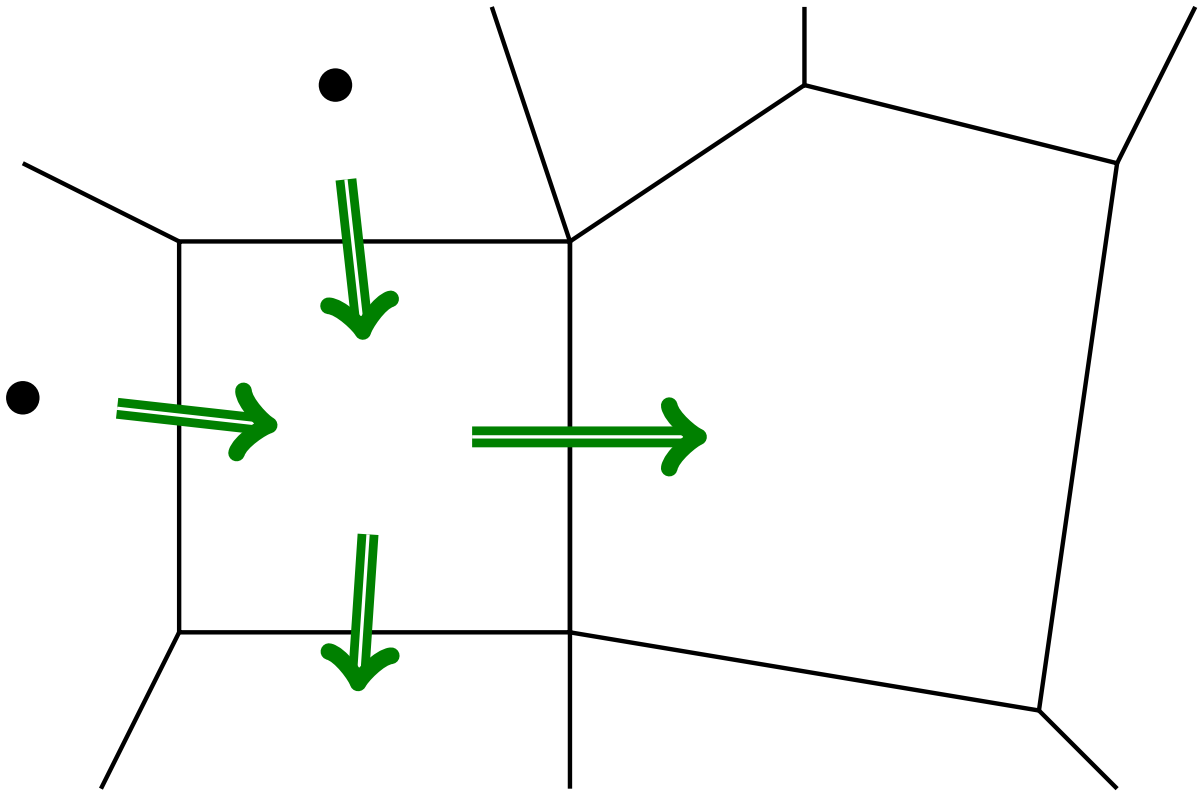
- Upwind discretization
- Information flows in direction of fluid flow
- Solution in upstream cells does not depend on solution in downstream cells
  (... unless we have loops or countercurrent flow)

Discretization setting (transport)



Fluxes $v_{g,ij}$, $j \in C(i)$.
Cells $U(i, g)$.

Discretization setting (transport)



Fluxes $v_{o,ij}$, $j \in C(i)$.
Cells $U(i, o)$.

Reordering the transport equations

Rewrite transport equation (for cell i, phase α):

$$F_i(x_i) + \sum_{j \in C(i)} G_i(x_i, x_j) v_{ij}(x_i, x_j, v_{T,ij}) = 0$$

v_{ij}: signed flux from cell i to cell j
x_i: unknowns in cell i

With upstream weighting we can write:

$$F_i(x_i) + \sum_{j \in U(i)} G_i^U(x_j) v_{ij}(x_j, v_{T,ij}) + \sum_{j \in D(i)} G_i^D(x_i) v_{ij}(x_i, v_{T,ij}) = 0$$

Given $x_j, j \in U(i)$, we can solve for x_i separately!

Countercurrent flow $\implies$ can only do this per phase
(but we will relax this later)

For 1D case: can solve sequentially from injector to producer!

Newton-Raphson vs. Nonlinear Gauss-Seidel

Newton-Raphson

$r = F(x)$
while $||r|| > tolerance$ **do**
    Compute Jacobian matrix $J = dF/dx$
    Solve $Je = r$
    Update $x = x - e$
    $r = F(x)$

Gauss-Seidel

$r = F(x)$ **while** *any* $||r_i|| > tolerance$ **do**
    for *all cells* i **do**
        Solve single-cell problem i
        Update x_i
    $r = F(x)$

Perfect ordering: can drop outer loop!

General case: computing an ordering

What quantity to use?

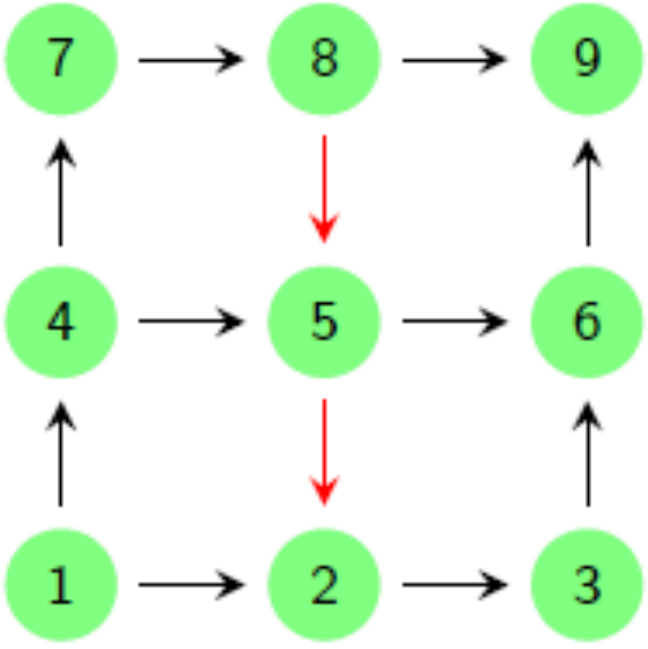
Single-phase flow with no gravity: sort according to pressure.

General case:

- Phase pressure? Which phase?
- Phase fluxes? Which phase?
- Total flux? What about countercurrent flow?

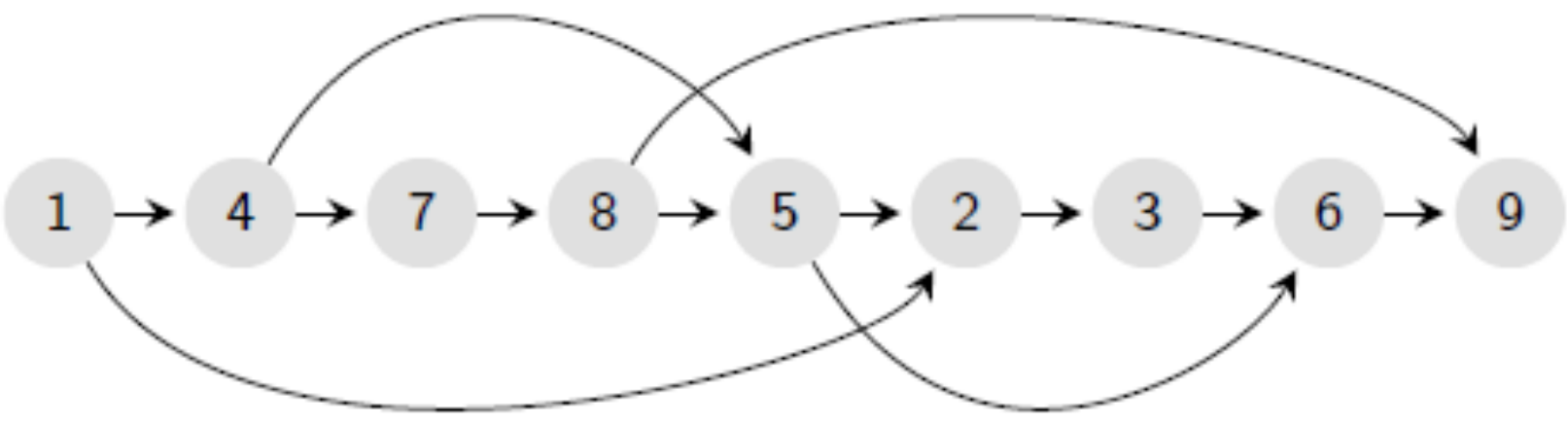
Our answer is: use total flux

Small example



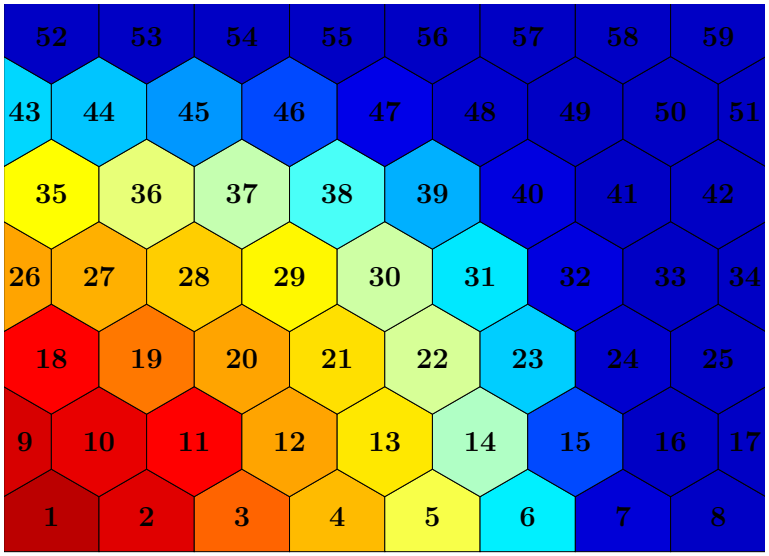
$$V = \begin{bmatrix} * & & & & & & & & \\ * & * & & & * & & & & \\ & * & * & & & & & & \\ * & & & * & & & & & \\ & & & * & * & & & * & \\ & & * & & * & * & & & \\ & & & * & & & * & & \\ & & & & & & * & * & \\ & & & & & * & & * & * \end{bmatrix}$$

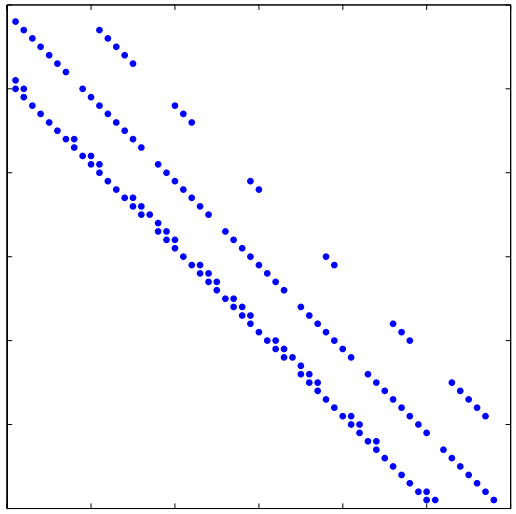
After topological sorting (Tarjan's algorithm): unidirectional graph



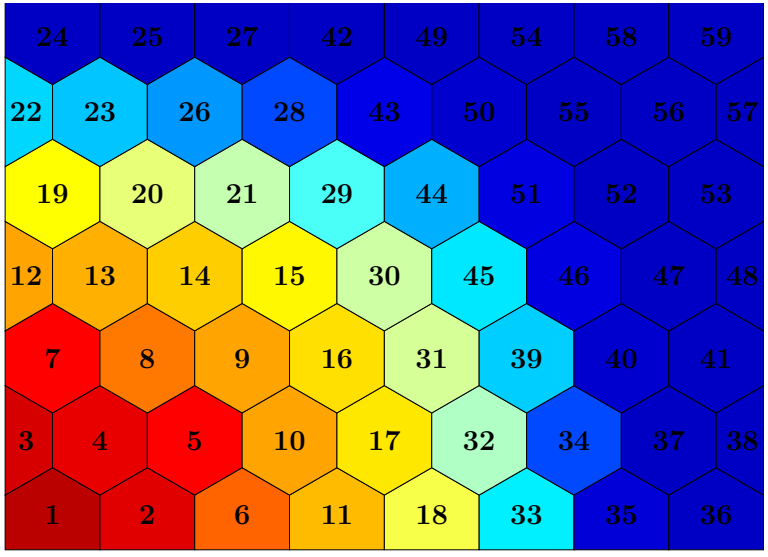
Slightly bigger example

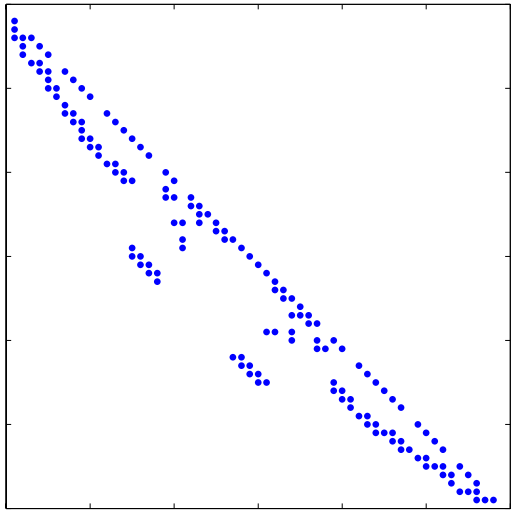
Natural ordering:





After reordering (Tarjan):

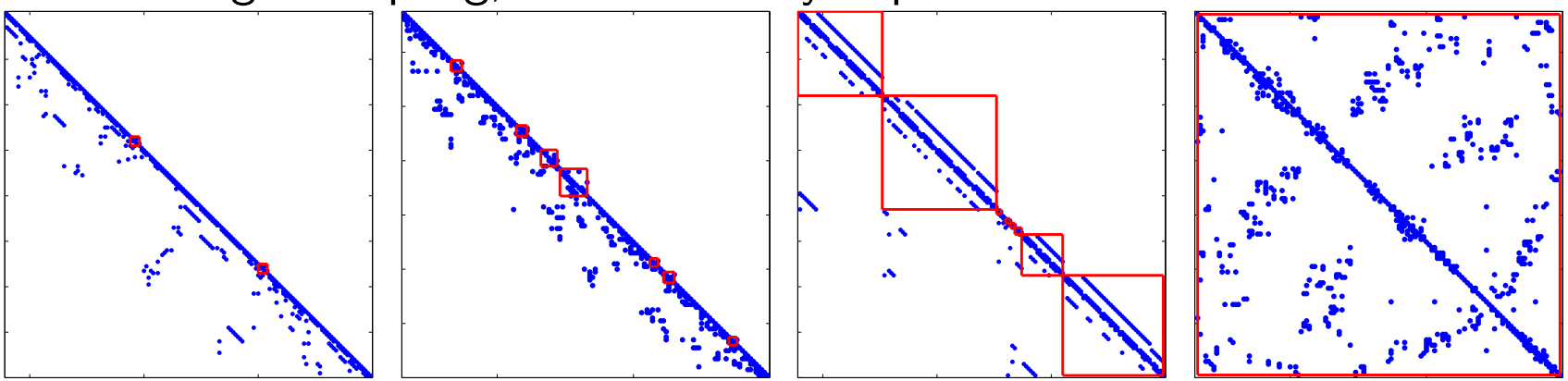




Ordering challenges

- Circular flow (gravity)
- Countercurrent flow (gravity, capillary pressure)

With stronger coupling, more mutually dependent cells:



How to handle such sets (strongly connected components)?

Solution: Gauss-Seidel iterations

Gauss-Seidel

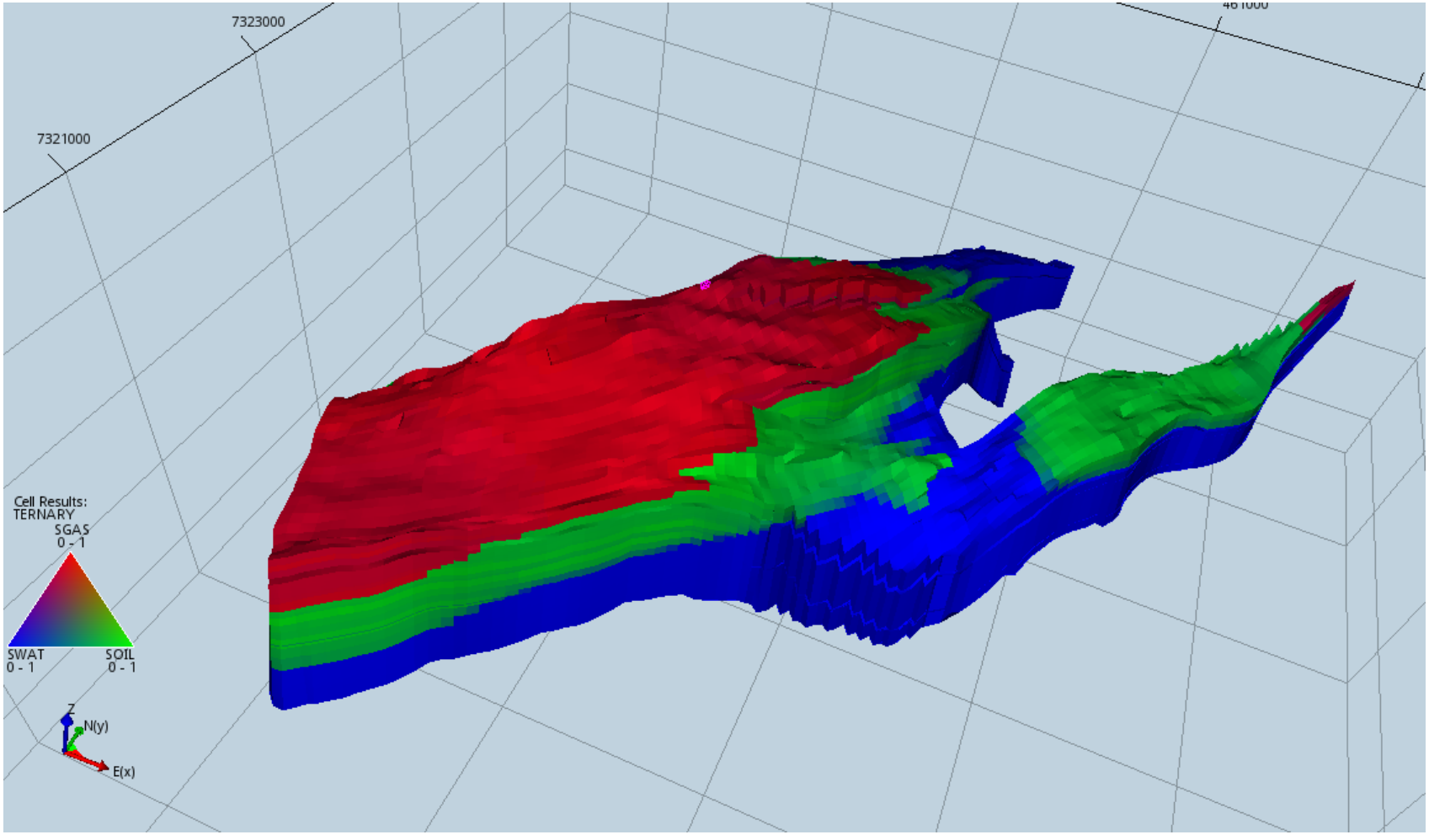
$r = F(x)$ **while** *any* $||r_i|| > tolerance$ **do**
- **for** *all cells* i **do**
  - Solve single-cell problem i
  - Update x_i
- $r = F(x)$

Apply outer loop, but *only for strongly connected cells*.

Convergence proofs:

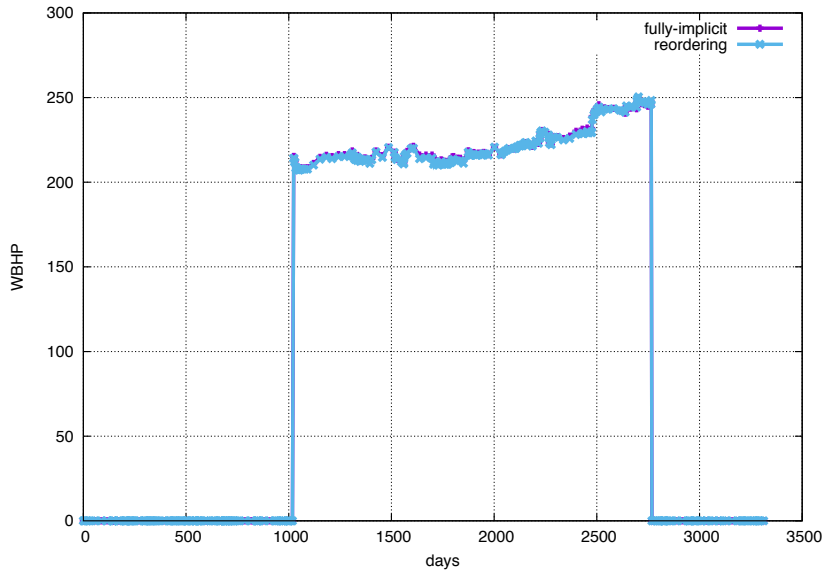
2-phase + polymer	yes
3-phase black-oil	no (but promising numerical results)

Norne field case

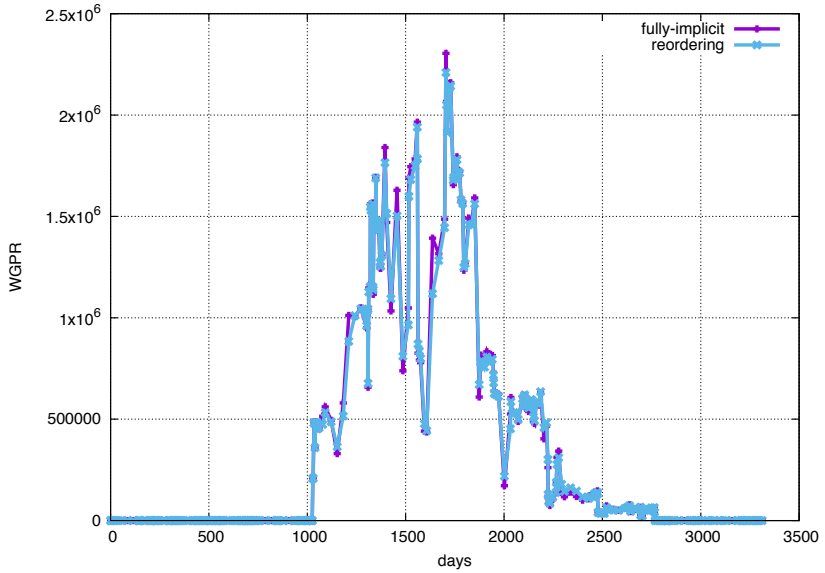


Norne real field case, initial saturation values

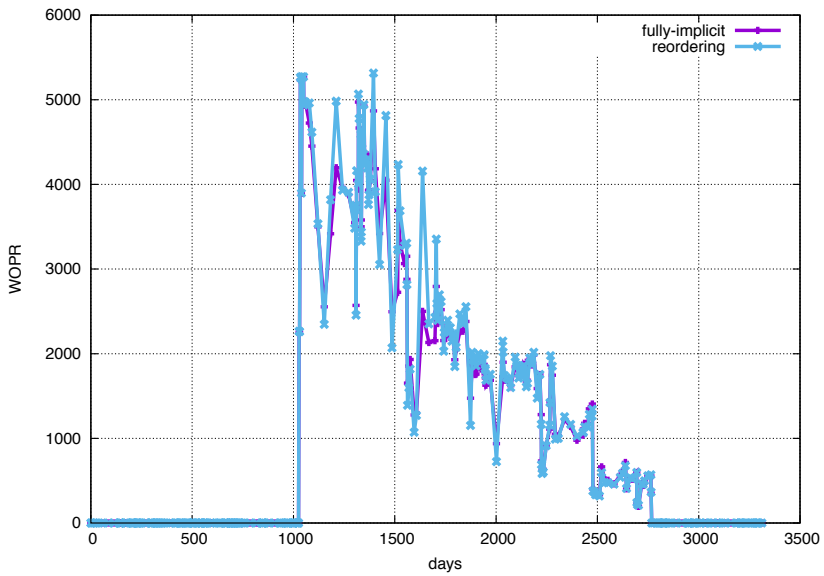
Norne field case: well D-3AH



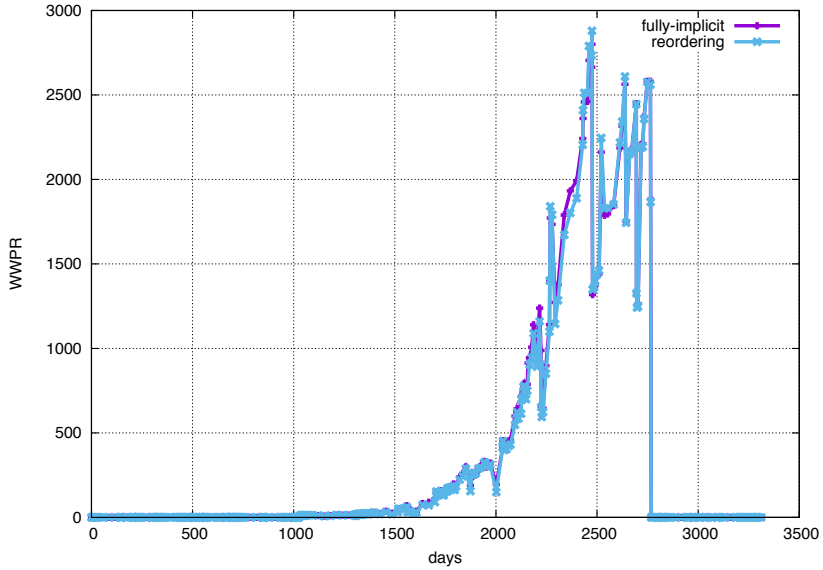
Bottom-hole pressure



Gas production rate

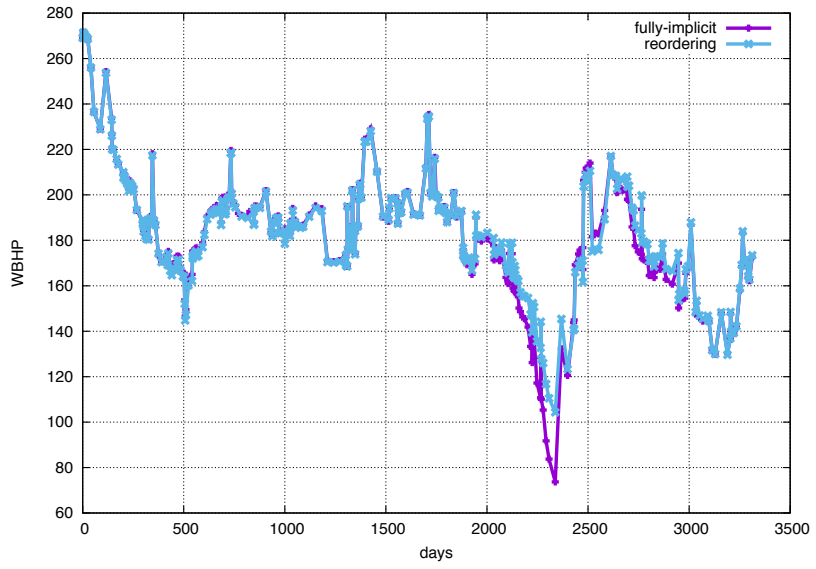


Oil production rate

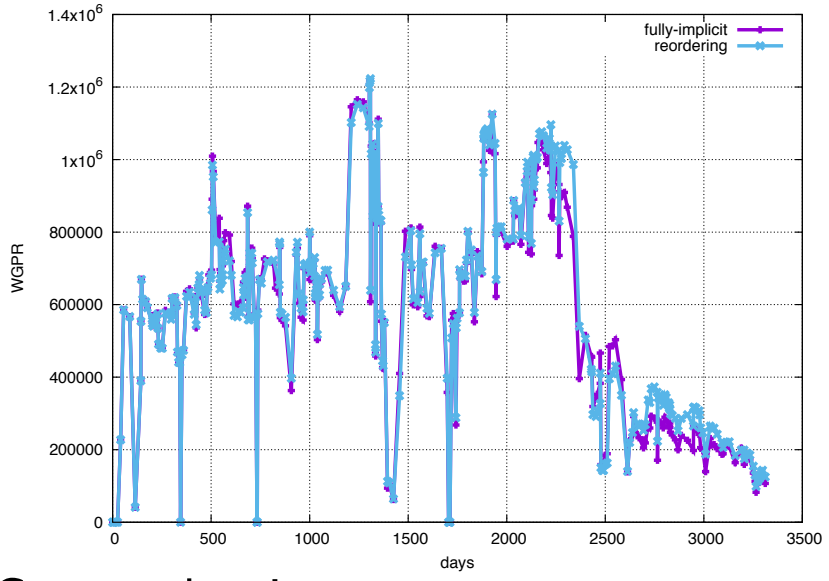


Water production rate

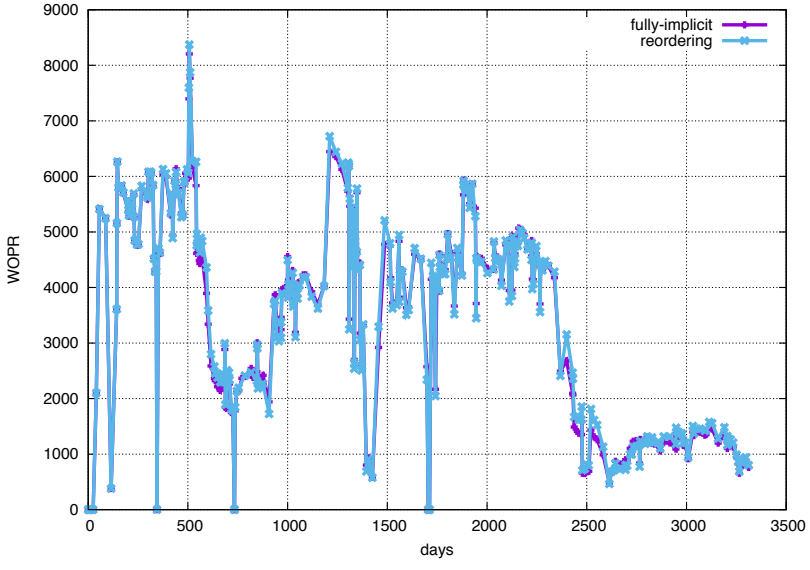
Norne field case: well B-2H



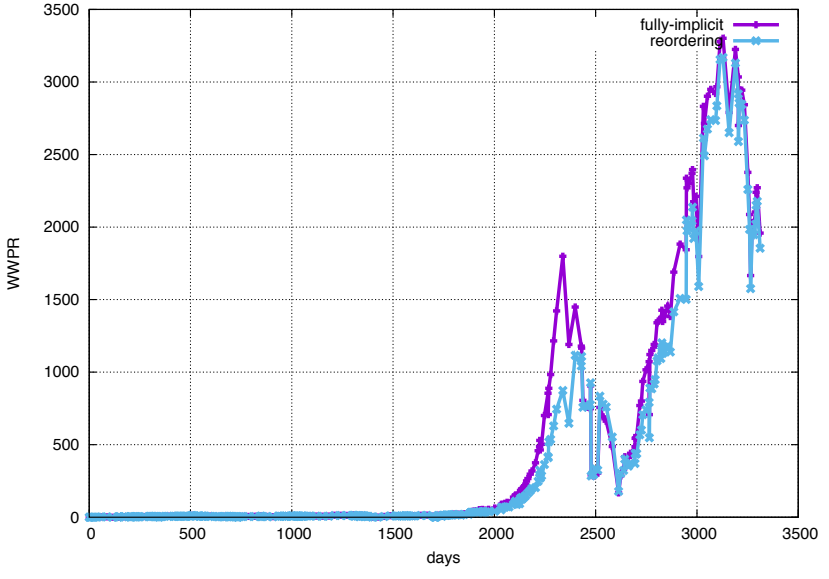
Bottom-hole pressure



Gas production rate



Oil production rate



Water production rate

Conclusion

- We can use sequential implicit methods on real-world field cases
  - ... at least for history-matching runs
- We can use reordering methods to solve the transport problem

Ongoing and future work

- Performance
  - Hope to make sequential implicit method roughly twice as fast as fully implicit.
  - Use transport solver as nonlinear preconditioner to improve performance of fully implicit simulation, possibly combined with CPR.
- Parallelization
  - Investigate possible approaches (multigrid-like, domain decomposition, etc.)
  - (Not in the near term)
- Robustness
  - Ensure successful runs for all available testcases (you can help!)
  - Investigate alternative solvers for single-cell problem (nested bracketed solver rather than Newton)

Availability

All simulators used are free and open-source.

- OPM website: opm-project.org
- OPM software sources: github.com/OPM

To run Norne as shown in this talk:

- fully implicit:
  `flow NORNE_ATW2013.DATA output_dir=fully-implicit`
  (also see Norne tutorial on opm website)
- reordering:
  `flow_reorder NORNE_ATW2013.DATA ds_max=0.1`
  `output_dir=reorder`

`flow_reorder` is available as source in current master on GitHub, also as binary starting in upcoming release (2017.10)

Acknowledgements

Colleagues at SINTEF (ideas and discussions)

The OPM community (code and feedback)

Statoil (funding and data)

The CLIMIT program (funding)

The MSO4SC project (funding)

Thank you for listening!

Extra slides

Performance outlook

Fully implicit solver	
Stage	Time (s)
Assembly	321
Linear solver	299
Update	13

Reordering sequential solver	
Stage	Time (s)
Pressure solver	1380
Transport solver	345

Hoped-for potential	
Stage	Time (s)
Pressure solver	260
Transport solver	70

Rationale behind hoped-for potential:

- New pressure solver, does about half the assembly work (value and 1 derivative, rather than value and 3 derivatives) and linear solver deals with 1×1 rather than 3×3 blocks.
- Current transport solver does no convergence checking, brute-forces 5 global Gauss-Seidel iterations. Norne experiment leads us to expect average of close to 1 Gauss-Seidel iteration per cell.
- Current transport solver writes excessive log output, taking significant time.

Discrete equations (I)

The discretized equations and residuals are, for each pseudo-component α and cell i:

$$R_{\alpha,i} = \frac{\phi_i V_i}{\Delta t}\left(A_{\alpha,i} - A^0_{\alpha,i}\right) + \sum_{j \in C(i)} u_{\alpha,ij} - Q_{\alpha,i} = 0 \tag{1}$$

where

$$A_w = b_w s_w, \qquad u_w = b_w v_w, \tag{2}$$
$$A_o = b_o s_o + r_{og} b_g s_g, \qquad u_o = b_o v_o + r_{og} b_g v_g, \tag{3}$$
$$A_g = b_g s_g + r_{go} b_o s_o, \qquad u_g = b_g v_g + r_{go} b_o v_o. \tag{4}$$

Also:

$$s_w + s_o + s_g = 1 \qquad p_{cow} = p_o - p_w \qquad p_{cog} = p_o - p_g. \tag{5}$$

Discrete equations (II)

The fluxes are given for each connection ij by:

$$(b_\alpha v_\alpha)_{ij} = (b_\alpha \lambda_\alpha)_{UP(\alpha,ij)} T_{ij} \Delta H_{\alpha,ij} \tag{6}$$

$$(r_{\beta\alpha} b_\alpha v_\alpha)_{ij} = (r_{\beta\alpha} b_\alpha \lambda_\alpha)_{UP(\alpha,ij)} T_{ij} \Delta H_{\alpha,ij} \tag{7}$$

$$\Delta H_{\alpha,ij} = p_{\alpha,i} - p_{\alpha,j} - g\rho_{\alpha,ij}(z_i - z_j) \tag{8}$$

$$\rho_{\alpha,ij} = (\rho_{\alpha,i} + \rho_{\alpha,j})/2 \tag{9}$$

$$UP(\alpha, ij) = \begin{cases} i & \Delta H_{\alpha,ij} \geq 0 \\ j & \Delta H_{\alpha,ij} < 0 \end{cases} \tag{10}$$

Pressure equation

Mass balance residuals (again)

$$R_{\alpha,i} = \frac{\phi_i V_i}{\Delta t}\left(A_{\alpha,i} - A^0_{\alpha,i}\right) + \sum_{j \in C(i)} u_{\alpha,ij} - Q_{\alpha,i} = 0$$

$$\begin{aligned} A_w &= b_w s_w, & u_w &= b_w v_w, \\ A_o &= b_o s_o + r_V b_g s_g, & u_o &= b_o v_o + r_V b_g v_g, \\ A_g &= b_g s_g + r_S b_o s_o, & u_g &= b_g v_g + r_S b_o v_o. \end{aligned}$$

Pressure equation: linear combination to eliminate saturations

$$R_p = \sum_\alpha \sigma_\alpha R_\alpha = 0$$

$$\sigma_w = 1/b_w \qquad \sigma_o = \frac{1/b_o - r_S/b_g}{1 - r_S r_V} \qquad \sigma_g = \frac{1/b_g - r_V/b_o}{1 - r_S r_V}$$

Store $v_{T,ij} = \sum_\alpha v_{\alpha,ij}$ for transport solver.

Transport equations

Mass balance residuals (once more...)

$$R_{\alpha,i} = \frac{\phi_i V_i}{\Delta t}\left(A_{\alpha,i} - A^0_{\alpha,i}\right) + \sum_{j \in C(i)} u_{\alpha,ij} - Q_{\alpha,i} = 0$$

$$\begin{aligned} A_w &= b_w s_w, & u_w &= b_w v_w, \\ A_o &= b_o s_o + r_V b_g s_g, & u_o &= b_o v_o + r_V b_g v_g, \\ A_g &= b_g s_g + r_S b_o s_o, & u_g &= b_g v_g + r_S b_o v_o. \end{aligned}$$

Transport equations

$$R_o = 0 \qquad R_g = 0$$

Transport equation fluxes: fractional flow formulation

Establish upwind directions for each phase (following Brenier and Jaffré):

$$UP(\alpha, ij) \in i, j$$

(Function of p_β, ρ_β, λ_β for all β at both cells i and j; T_{ij} and v_T).

Phase fluxes are then given by:

$$(b_\alpha v_\alpha)_{ij} = b_{\alpha,ij} \frac{\lambda_{\alpha,ij}}{\sum_\beta \lambda_{\beta,ij}} (v_T + T_{ij} G_{ij})$$

where

$$\begin{aligned}
b_{\alpha,ij} &= b_{\alpha,UP(\alpha,ij)} \\
\lambda_{\alpha,ij} &= \lambda_{\alpha,UP(\alpha,ij)} \\
G_{\alpha,ij} &= \sum_{\beta \neq \alpha} \lambda_{\beta,ij} (D_{\alpha,ij} - D_{\beta,ij}) \\
D_{\alpha,ij} &= p_{o,j} - p_{o,i} - (p_{\alpha,j} - p_{\alpha,i} - g\rho_{\alpha,ij}(z_i - z_j))
\end{aligned}$$

What does AD provide

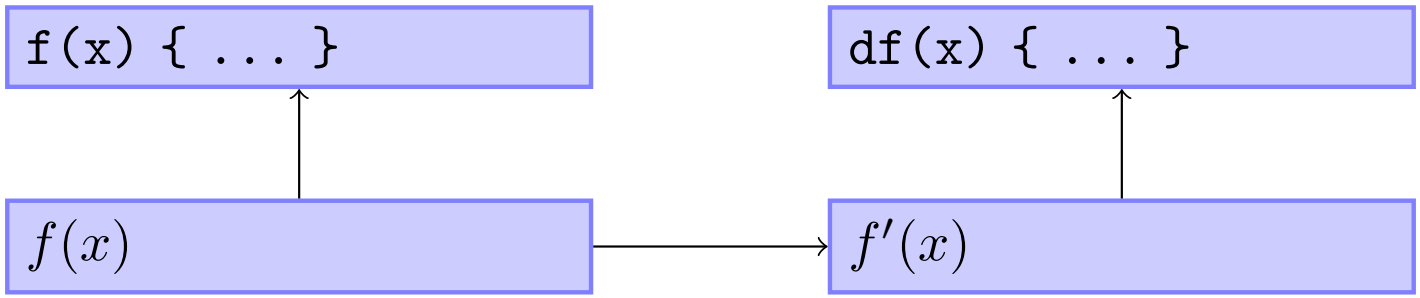
```
f(x) { ... }
```

```
df(x) { ... }
```

$f(x)$

$f'(x)$

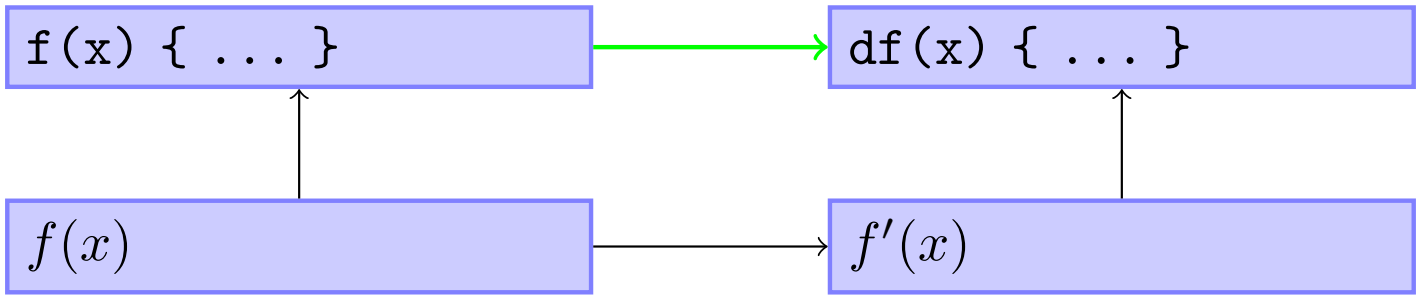
What does AD provide



Traditional Process

- Human implements code to evaluate $f(x)$
- Manual or symbolic calculation to derive $f'(x)$
- Human implements code to evaluate $f'(x)$

What does AD provide



Traditional Process

- Human implements code to evaluate $f(x)$
- Manual or symbolic calculation to derive $f'(x)$
- Human implements code to evaluate $f'(x)$

Automatic Differentiation (AD)

- Human implements code to evaluate $f(x)$
- Computer code to evaluate $f'(x)$ is automatically generated

Benefits of using AD

AD makes it easier to create simulators:

- only specify nonlinear residual equation
- automatically evaluates Jacobian
- sparsity structure of Jacobian automatically generated

Note that AD is *not* the same as finite differencing!

- no need to define a 'small' epsilon
- as precise as hand-made Jacobian
- ... but much less work!

Performance (of equation assembly) will usually be somewhat slower than a *good* hand-made Jacobian implementation.

Basic idea

A numeric computation $y = f(x)$ can be written (D = derivative)

$$\begin{aligned}
y_1 &= f_1(x) & \frac{dy_1}{dx}(x) &= Df_1(x) \\
y_2 &= f_2(y_1) & \frac{dy_2}{dx}(x) &= Df_2(y_1) \cdot Df_1(x) \\
&\vdots \\
y &= f_n(y_{n-1}) & \frac{dy}{dx}(x) &= Df_n(y_{n-1}) \cdot Df_{n-1}(y_{n-2}) \cdots Df_1(x)
\end{aligned}$$

Automatic Differentiation:

- make each line an elementary operation
- compute right derivative values as we go using chain rule

Implementation approaches

Two main methods:

Operator overloading

- requires operator overloading in programming language
- syntax (more or less) like before (non-AD)
- efficiency can vary a lot, depends on usage scenario
- easy to implement and experiment with
- Examples: OPM, Sacado (Trilinos), ADOL-C

Source transformation with AD tool

- can be implemented for almost any language
- may restrict language syntax or features used
- efficiency can be high (depends on AD tool)
- Examples: TAPENADE, OpenAD

Types of AD

Two different approaches.
(We compute $f(x)$, u is some intermediate variable.)

Forward Mode

Carry derivatives with respect to independent variables:

$$(u, \frac{du}{dx})$$

Reverse Mode

Carry derivatives with respect to dependent variables (adjoints):

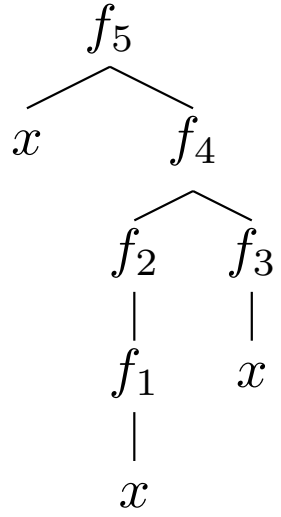
$$(u, \frac{df}{du})$$

Forward AD example (1)

Example function: $f(x) = x(sin(x^2) + 3x)$.
Sequence of elementary functions:

$$
\begin{aligned}
f_1(u) &= u^2 & f_1'(u) &= 2uu' \\
f_2(u) &= \sin(u) & f_2'(u) &= \cos(u)u' \\
f_3(u) &= 3u & f_3'(u) &= 3u' \\
f_4(u,v) &= u + v & f_4'(u,v) &= u' + v' \\
f_5(u,v) &= u \cdot v & f_5'(u,v) &= u'v + uv'
\end{aligned}
$$

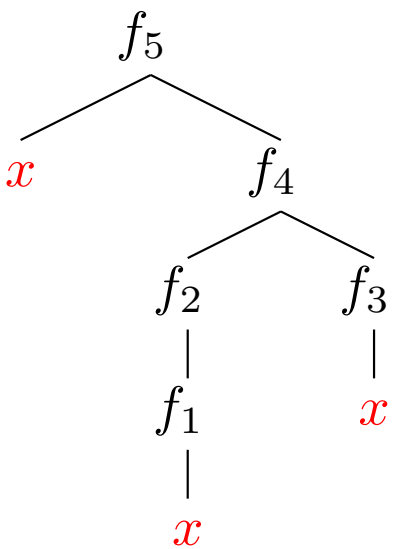
Rewritten:
$f(x) = f_5(x, f_4(f_2(f_1(x)), f_3(x)))$

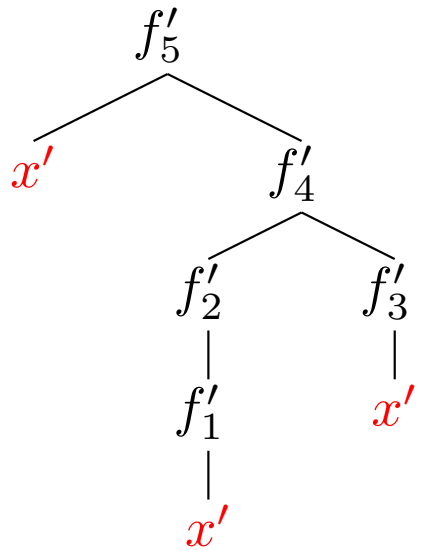


Forward AD example (2)

Example function: $f(x) = x(sin(x^2) + 3x)$. Computing $f(3), f'(3)$.
Sequence of elementary functions:

$$
\begin{aligned}
f_1(u) &= u^2 & f_1'(u) &= 2uu' \\
f_2(u) &= \sin(u) & f_2'(u) &= \cos(u)u' \\
f_3(u) &= 3u & f_3'(u) &= 3u' \\
f_4(u,v) &= u + v & f_4'(u,v) &= u' + v' \\
f_5(u,v) &= u \cdot v & f_5'(u,v) &= u'v + uv'
\end{aligned}
$$

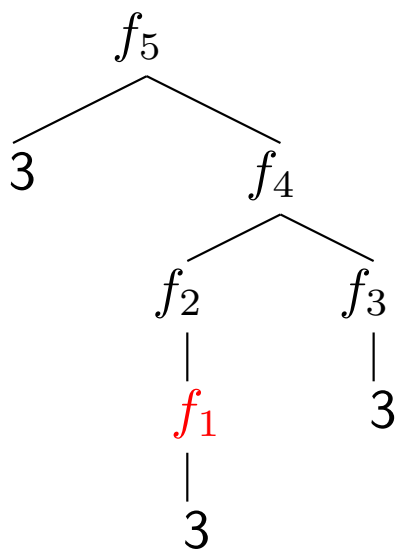


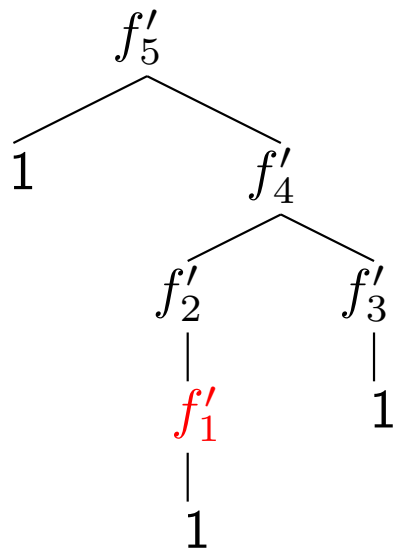


Forward AD example (2)

Example function: $f(x) = x(sin(x^2) + 3x)$. Computing $f(3), f'(3)$.
Sequence of elementary functions:

$$
\begin{aligned}
f_1(u) &= u^2 & f_1'(u) &= 2uu' \\
f_2(u) &= \sin(u) & f_2'(u) &= \cos(u)u' \\
f_3(u) &= 3u & f_3'(u) &= 3u' \\
f_4(u,v) &= u + v & f_4'(u,v) &= u' + v' \\
f_5(u,v) &= u \cdot v & f_5'(u,v) &= u'v + uv'
\end{aligned}
$$

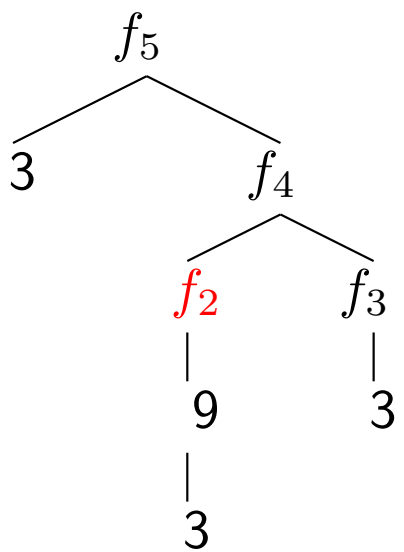


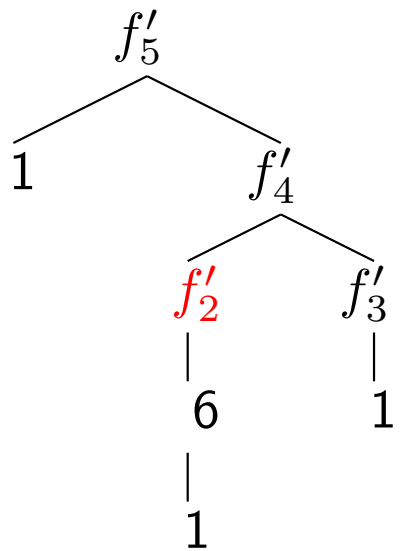


Forward AD example (2)

Example function: $f(x) = x(sin(x^2) + 3x)$. Computing $f(3), f'(3)$.
Sequence of elementary functions:

$$
\begin{aligned}
f_1(u) &= u^2 & f_1'(u) &= 2uu' \\
f_2(u) &= \sin(u) & f_2'(u) &= \cos(u)u' \\
f_3(u) &= 3u & f_3'(u) &= 3u' \\
f_4(u,v) &= u + v & f_4'(u,v) &= u' + v' \\
f_5(u,v) &= u \cdot v & f_5'(u,v) &= u'v + uv'
\end{aligned}
$$

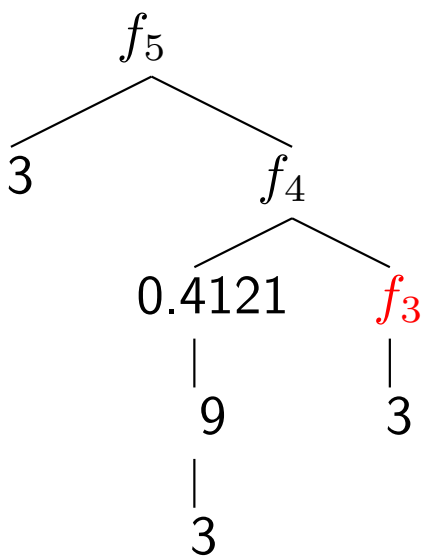


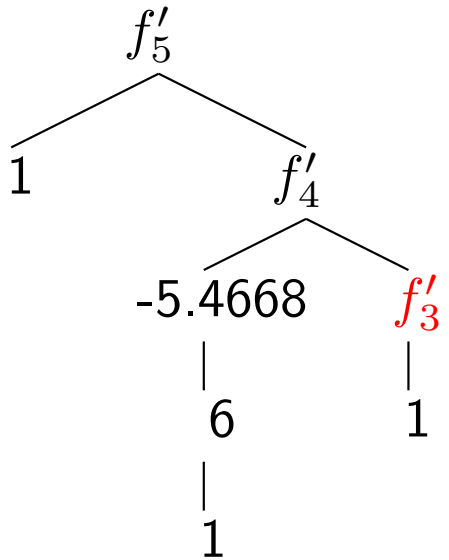


Forward AD example (2)

Example function: $f(x) = x(sin(x^2) + 3x)$. Computing $f(3), f'(3)$.
Sequence of elementary functions:

$$
\begin{aligned}
f_1(u) &= u^2 & f_1'(u) &= 2uu' \\
f_2(u) &= \sin(u) & f_2'(u) &= \cos(u)u' \\
f_3(u) &= 3u & f_3'(u) &= 3u' \\
f_4(u,v) &= u + v & f_4'(u,v) &= u' + v' \\
f_5(u,v) &= u \cdot v & f_5'(u,v) &= u'v + uv'
\end{aligned}
$$

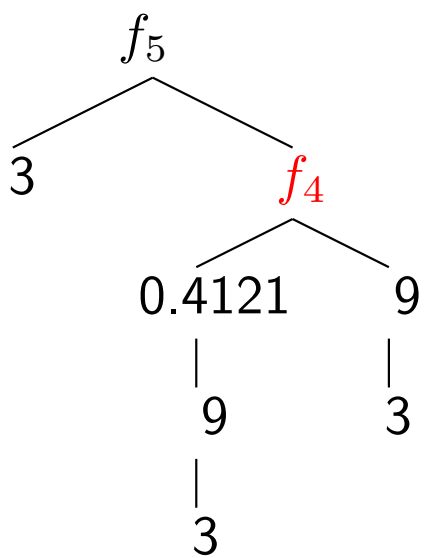


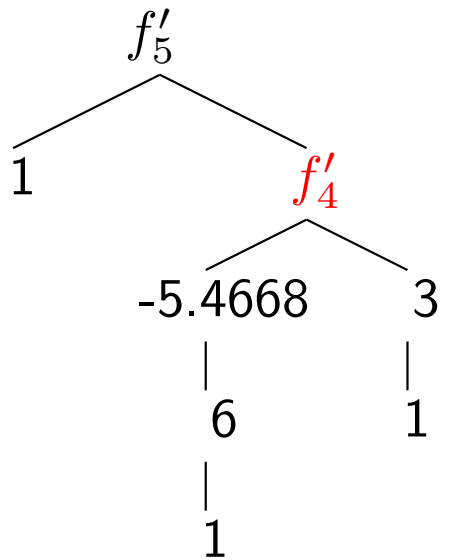


Forward AD example (2)

Example function: $f(x) = x(sin(x^2) + 3x)$. Computing $f(3), f'(3)$.
Sequence of elementary functions:

$$
\begin{aligned}
f_1(u) &= u^2 & f_1'(u) &= 2uu' \\
f_2(u) &= \sin(u) & f_2'(u) &= \cos(u)u' \\
f_3(u) &= 3u & f_3'(u) &= 3u' \\
f_4(u,v) &= u + v & f_4'(u,v) &= u' + v' \\
f_5(u,v) &= u \cdot v & f_5'(u,v) &= u'v + uv'
\end{aligned}
$$

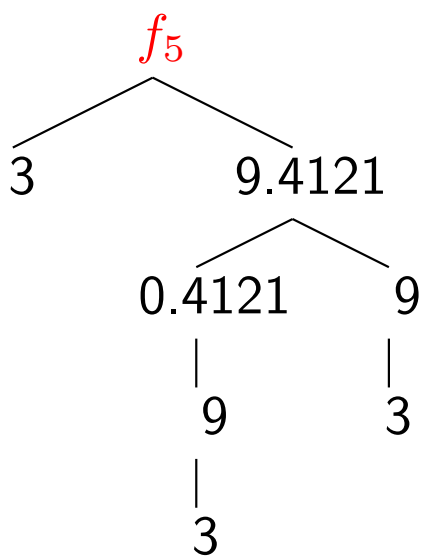


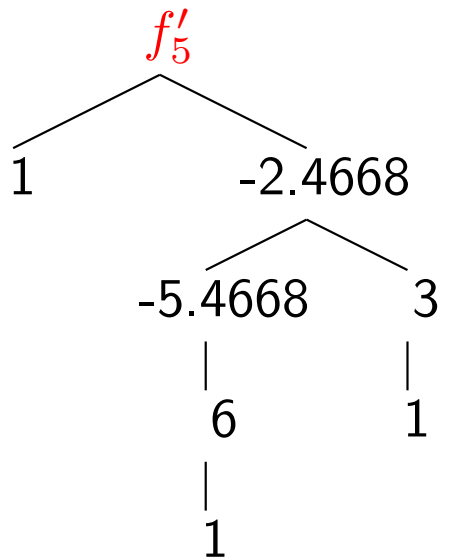


Forward AD example (2)

Example function: $f(x) = x(sin(x^2) + 3x)$. Computing $f(3), f'(3)$.
Sequence of elementary functions:

$$
\begin{aligned}
f_1(u) &= u^2 & f_1'(u) &= 2uu' \\
f_2(u) &= \sin(u) & f_2'(u) &= \cos(u)u' \\
f_3(u) &= 3u & f_3'(u) &= 3u' \\
f_4(u,v) &= u + v & f_4'(u,v) &= u' + v' \\
f_5(u,v) &= u \cdot v & f_5'(u,v) &= u'v + uv'
\end{aligned}
$$

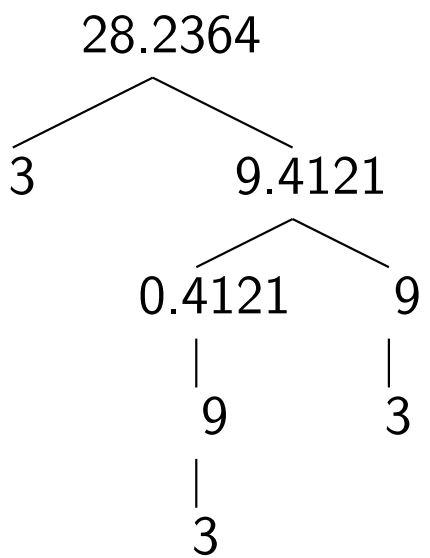


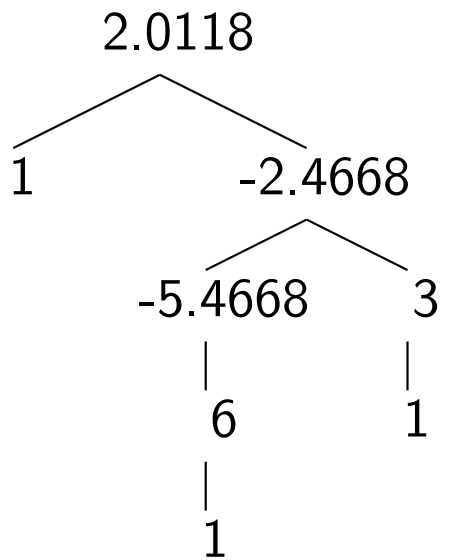


Forward AD example (2)

Example function: $f(x) = x(sin(x^2) + 3x)$. Computing $f(3), f'(3)$.
Sequence of elementary functions:

$$
\begin{aligned}
f_1(u) &= u^2 & f_1'(u) &= 2uu' \\
f_2(u) &= \sin(u) & f_2'(u) &= \cos(u)u' \\
f_3(u) &= 3u & f_3'(u) &= 3u' \\
f_4(u,v) &= u + v & f_4'(u,v) &= u' + v' \\
f_5(u,v) &= u \cdot v & f_5'(u,v) &= u'v + uv'
\end{aligned}
$$





Properties of forward AD

- Easy to implement with operator overloading
- Storage required (scalar): $2\times$ normal (value, derivative).
- Storage required ($f : R^m \to R^n$): $(n+1)\times$ normal (value, derivative vector), unless sparse.

Reverse Mode AD

Recall: Reverse Mode

Carry derivatives with respect to dependent variables (adjoints):

$$(u, \frac{df}{du})$$

We will use the chain rule again, but in the opposite direction:

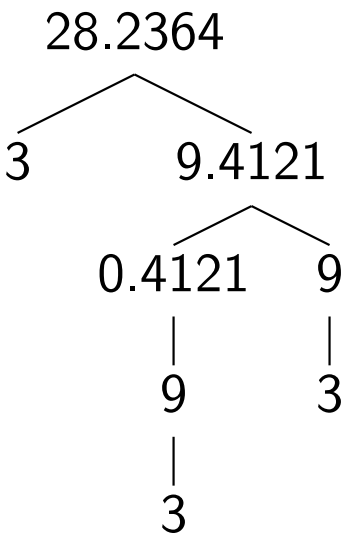
$$\mathrm{adj}(u) = \mathrm{adj}(f_i)\frac{\partial f_i}{\partial u}.$$

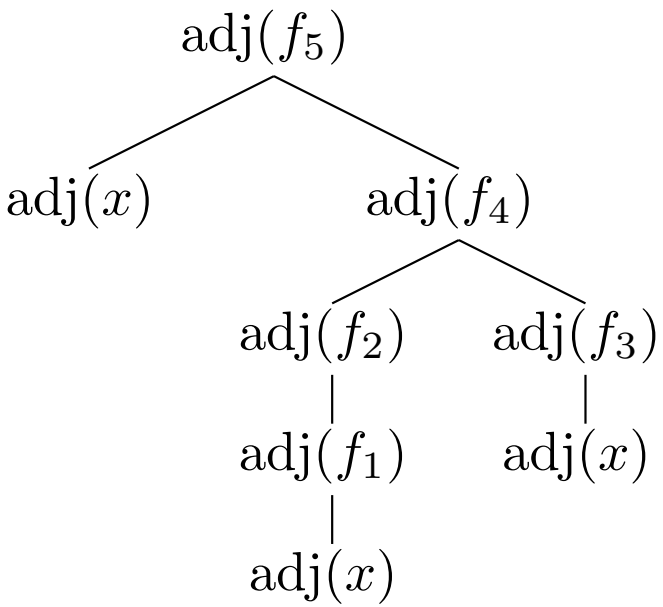
Using $\mathrm{adj}(u)$ to mean the adjoint $\frac{df}{du}$.
(So $\mathrm{adj}(x)$ is our goal.)

Reverse AD example

Example function: $f(x) = x(sin(x^2) + 3x)$. Computing $f(3), f'(3)$.
Sequence of elementary functions:

$$
\begin{aligned}
f_1(u) &= u^2 & \mathrm{adj}(u) &= \mathrm{adj}(f_1) \cdot 2u \\
f_2(u) &= \sin(u) & \mathrm{adj}(u) &= \mathrm{adj}(f_2) \cdot \cos(u) \\
f_3(u) &= 3u & \mathrm{adj}(u) &= \mathrm{adj}(f_3) \cdot 3 \\
f_4(u,v) &= u + v & \mathrm{adj}(u) &= \mathrm{adj}(f_4), & \mathrm{adj}(v) &= \mathrm{adj}(f_4) \\
f_5(u,v) &= u \cdot v & \mathrm{adj}(u) &= \mathrm{adj}(f_5) \cdot v, & \mathrm{adj}(v) &= \mathrm{adj}(f_5) \cdot u
\end{aligned}
$$

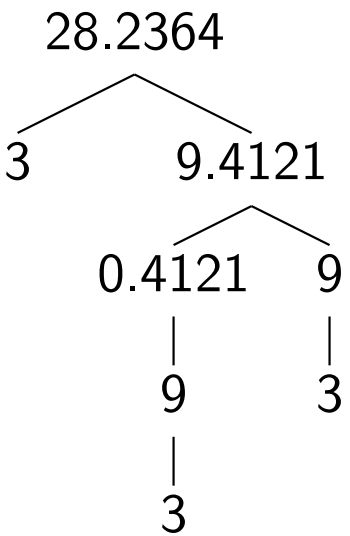


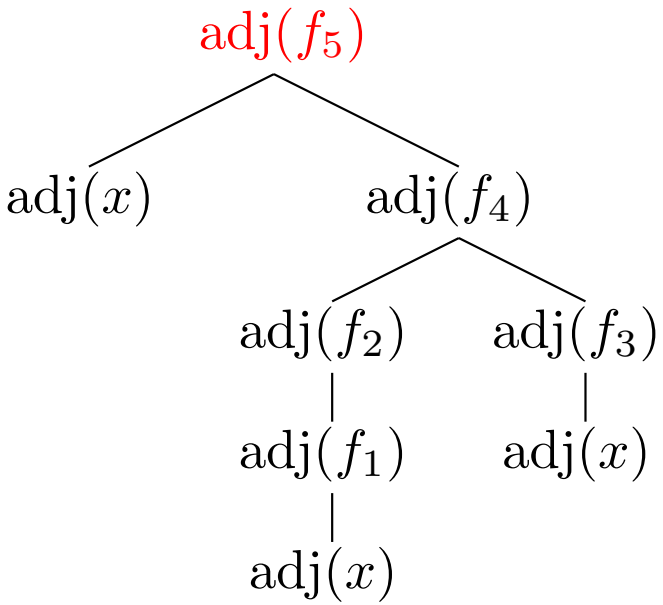


Reverse AD example

Example function: $f(x) = x(sin(x^2) + 3x)$. Computing $f(3), f'(3)$.
Sequence of elementary functions:

$$
\begin{aligned}
f_1(u) &= u^2 & \mathrm{adj}(u) &= \mathrm{adj}(f_1) \cdot 2u \\
f_2(u) &= \sin(u) & \mathrm{adj}(u) &= \mathrm{adj}(f_2) \cdot \cos(u) \\
f_3(u) &= 3u & \mathrm{adj}(u) &= \mathrm{adj}(f_3) \cdot 3 \\
f_4(u,v) &= u + v & \mathrm{adj}(u) &= \mathrm{adj}(f_4), & \mathrm{adj}(v) &= \mathrm{adj}(f_4) \\
f_5(u,v) &= u \cdot v & \mathrm{adj}(u) &= \mathrm{adj}(f_5) \cdot v, & \mathrm{adj}(v) &= \mathrm{adj}(f_5) \cdot u
\end{aligned}
$$

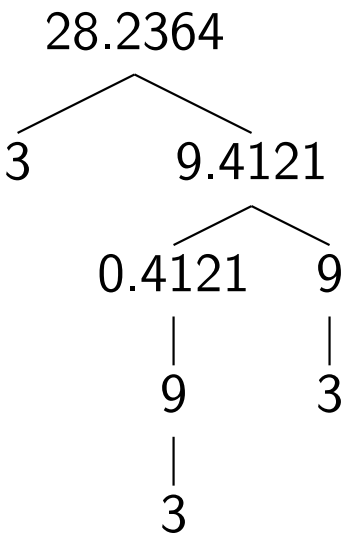


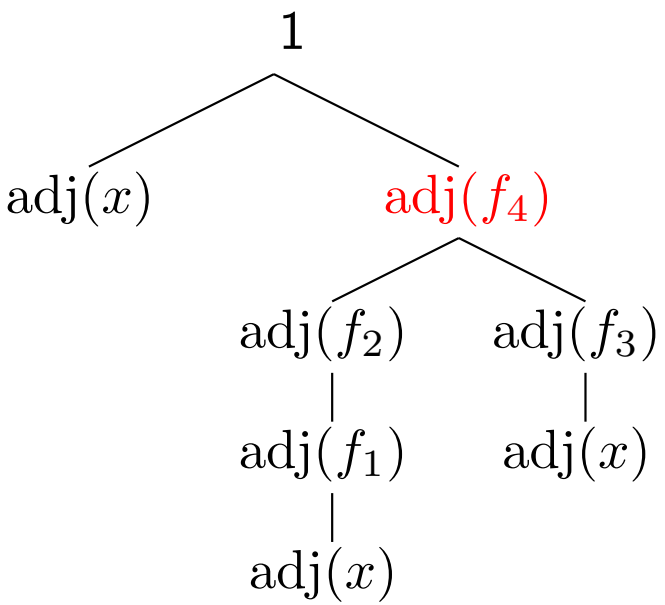


Reverse AD example

Example function: $f(x) = x(sin(x^2) + 3x)$. Computing $f(3), f'(3)$.
Sequence of elementary functions:

$$
\begin{aligned}
f_1(u) &= u^2 & \mathrm{adj}(u) &= \mathrm{adj}(f_1) \cdot 2u \\
f_2(u) &= \sin(u) & \mathrm{adj}(u) &= \mathrm{adj}(f_2) \cdot \cos(u) \\
f_3(u) &= 3u & \mathrm{adj}(u) &= \mathrm{adj}(f_3) \cdot 3 \\
f_4(u, v) &= u + v & \mathrm{adj}(u) &= \mathrm{adj}(f_4), & \mathrm{adj}(v) &= \mathrm{adj}(f_4) \\
f_5(u, v) &= u \cdot v & \mathrm{adj}(u) &= \mathrm{adj}(f_5) \cdot v, & \mathrm{adj}(v) &= \mathrm{adj}(f_5) \cdot u
\end{aligned}
$$

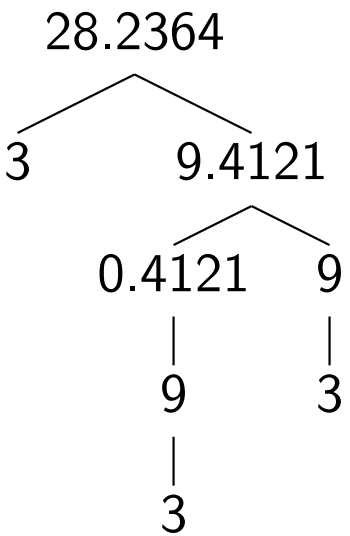


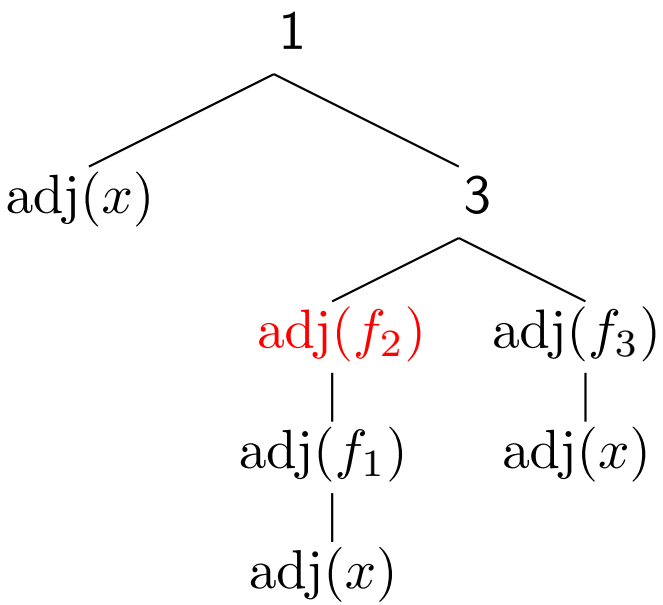


Reverse AD example

Example function: $f(x) = x(sin(x^2) + 3x)$. Computing $f(3), f'(3)$.
Sequence of elementary functions:

$$
\begin{aligned}
f_1(u) &= u^2 & \mathrm{adj}(u) &= \mathrm{adj}(f_1) \cdot 2u \\
f_2(u) &= \sin(u) & \mathrm{adj}(u) &= \mathrm{adj}(f_2) \cdot \cos(u) \\
f_3(u) &= 3u & \mathrm{adj}(u) &= \mathrm{adj}(f_3) \cdot 3 \\
f_4(u,v) &= u + v & \mathrm{adj}(u) &= \mathrm{adj}(f_4), & \mathrm{adj}(v) &= \mathrm{adj}(f_4) \\
f_5(u,v) &= u \cdot v & \mathrm{adj}(u) &= \mathrm{adj}(f_5) \cdot v, & \mathrm{adj}(v) &= \mathrm{adj}(f_5) \cdot u
\end{aligned}
$$

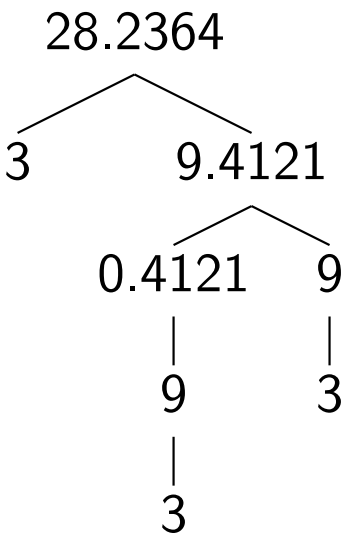


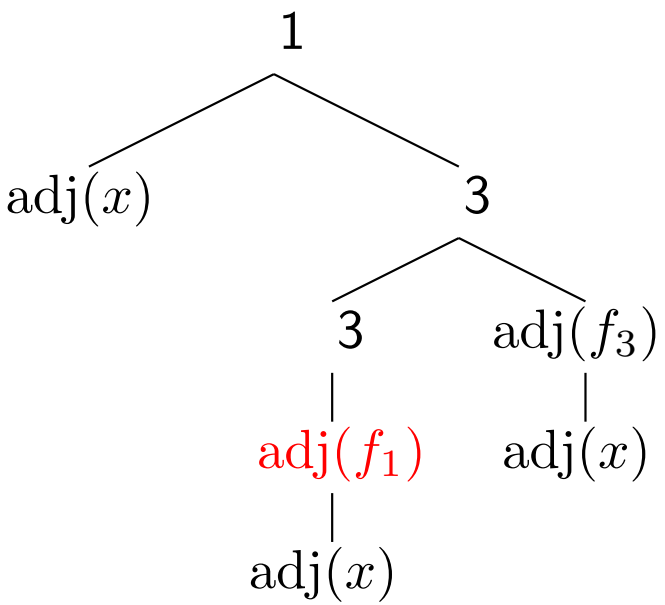


Reverse AD example

Example function: $f(x) = x(sin(x^2) + 3x)$. Computing $f(3), f'(3)$.
Sequence of elementary functions:

$$
\begin{aligned}
f_1(u) &= u^2 & \mathrm{adj}(u) &= \mathrm{adj}(f_1) \cdot 2u \\
f_2(u) &= \sin(u) & \mathrm{adj}(u) &= \mathrm{adj}(f_2) \cdot \cos(u) \\
f_3(u) &= 3u & \mathrm{adj}(u) &= \mathrm{adj}(f_3) \cdot 3 \\
f_4(u, v) &= u + v & \mathrm{adj}(u) &= \mathrm{adj}(f_4), & \mathrm{adj}(v) &= \mathrm{adj}(f_4) \\
f_5(u, v) &= u \cdot v & \mathrm{adj}(u) &= \mathrm{adj}(f_5) \cdot v, & \mathrm{adj}(v) &= \mathrm{adj}(f_5) \cdot u
\end{aligned}
$$

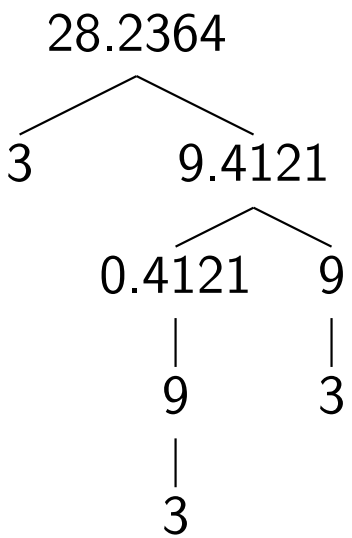


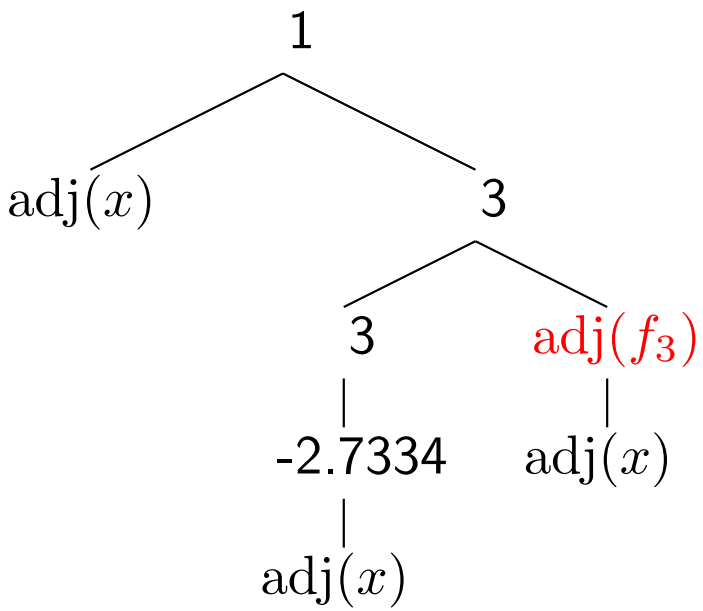


Reverse AD example

Example function: $f(x) = x(sin(x^2) + 3x)$. Computing $f(3), f'(3)$.
Sequence of elementary functions:

$$
\begin{aligned}
f_1(u) &= u^2 & \mathrm{adj}(u) &= \mathrm{adj}(f_1) \cdot 2u \\
f_2(u) &= \sin(u) & \mathrm{adj}(u) &= \mathrm{adj}(f_2) \cdot \cos(u) \\
f_3(u) &= 3u & \mathrm{adj}(u) &= \mathrm{adj}(f_3) \cdot 3 \\
f_4(u,v) &= u + v & \mathrm{adj}(u) &= \mathrm{adj}(f_4), & \mathrm{adj}(v) &= \mathrm{adj}(f_4) \\
f_5(u,v) &= u \cdot v & \mathrm{adj}(u) &= \mathrm{adj}(f_5) \cdot v, & \mathrm{adj}(v) &= \mathrm{adj}(f_5) \cdot u
\end{aligned}
$$

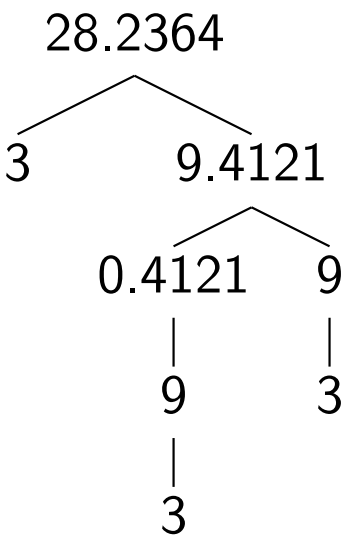


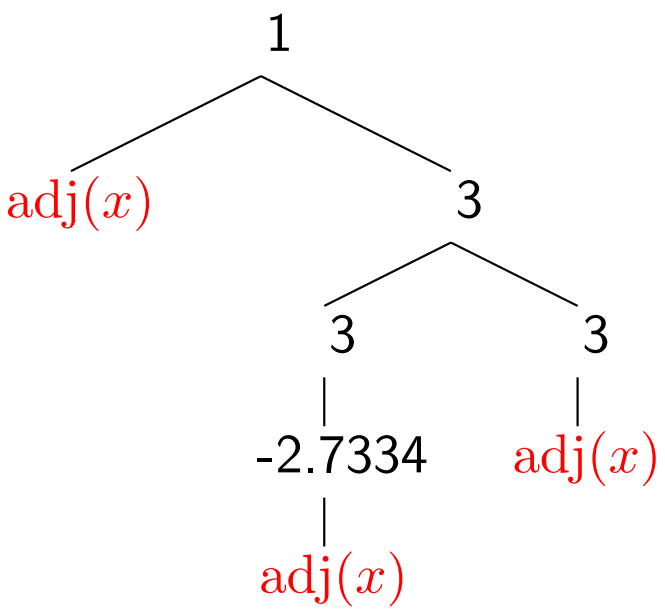


Reverse AD example

Example function: $f(x) = x(sin(x^2) + 3x)$. Computing $f(3), f'(3)$.
Sequence of elementary functions:

$$
\begin{aligned}
f_1(u) &= u^2 & \mathrm{adj}(u) &= \mathrm{adj}(f_1) \cdot 2u \\
f_2(u) &= \sin(u) & \mathrm{adj}(u) &= \mathrm{adj}(f_2) \cdot \cos(u) \\
f_3(u) &= 3u & \mathrm{adj}(u) &= \mathrm{adj}(f_3) \cdot 3 \\
f_4(u, v) &= u + v & \mathrm{adj}(u) &= \mathrm{adj}(f_4), & \mathrm{adj}(v) &= \mathrm{adj}(f_4) \\
f_5(u, v) &= u \cdot v & \mathrm{adj}(u) &= \mathrm{adj}(f_5) \cdot v, & \mathrm{adj}(v) &= \mathrm{adj}(f_5) \cdot u
\end{aligned}
$$

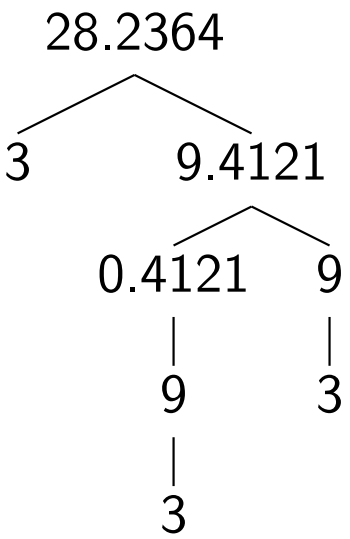


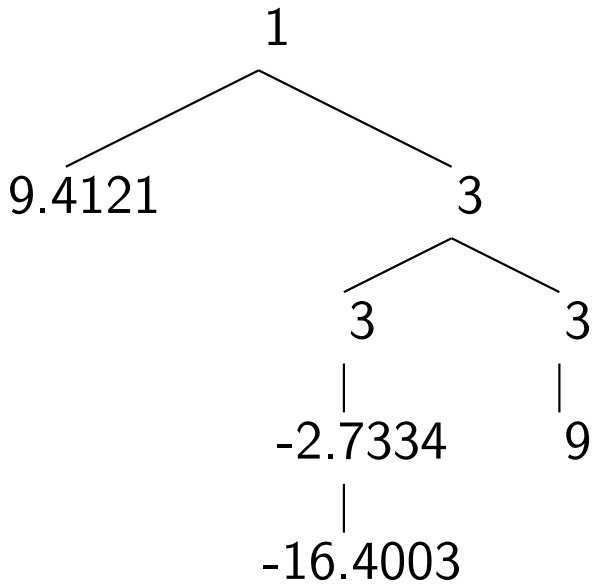


Reverse AD example

Example function: $f(x) = x(sin(x^2) + 3x)$. Computing $f(3), f'(3)$.
Sequence of elementary functions:

$$
\begin{aligned}
f_1(u) &= u^2 && \mathrm{adj}(u) = \mathrm{adj}(f_1) \cdot 2u \\
f_2(u) &= \sin(u) && \mathrm{adj}(u) = \mathrm{adj}(f_2) \cdot \cos(u) \\
f_3(u) &= 3u && \mathrm{adj}(u) = \mathrm{adj}(f_3) \cdot 3 \\
f_4(u, v) &= u + v && \mathrm{adj}(u) = \mathrm{adj}(f_4), \quad \mathrm{adj}(v) = \mathrm{adj}(f_4) \\
f_5(u, v) &= u \cdot v && \mathrm{adj}(u) = \mathrm{adj}(f_5) \cdot v, \quad \mathrm{adj}(v) = \mathrm{adj}(f_5) \cdot u
\end{aligned}
$$

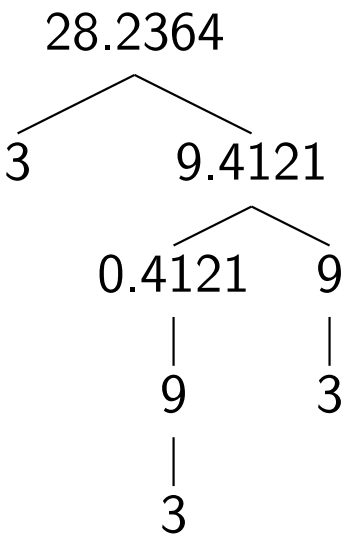


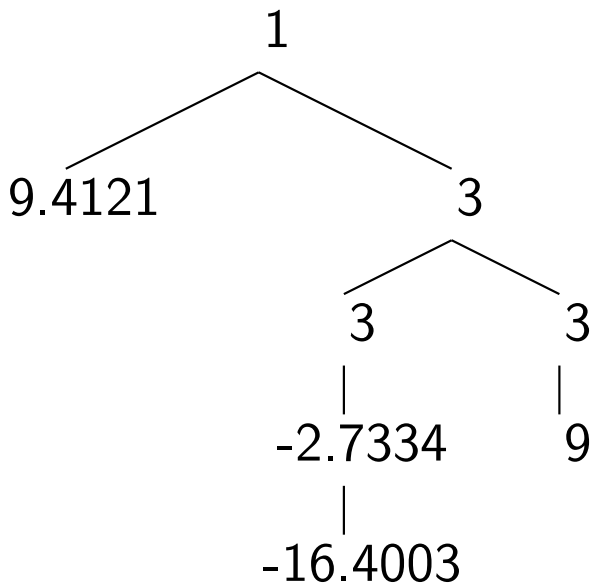


Reverse AD example

Example function: $f(x) = x(sin(x^2) + 3x)$. Computing $f(3), f'(3)$.
Sequence of elementary functions:

$$
\begin{aligned}
f_1(u) &= u^2 & \mathrm{adj}(u) &= \mathrm{adj}(f_1) \cdot 2u \\
f_2(u) &= \sin(u) & \mathrm{adj}(u) &= \mathrm{adj}(f_2) \cdot \cos(u) \\
f_3(u) &= 3u & \mathrm{adj}(u) &= \mathrm{adj}(f_3) \cdot 3 \\
f_4(u, v) &= u + v & \mathrm{adj}(u) &= \mathrm{adj}(f_4), & \mathrm{adj}(v) &= \mathrm{adj}(f_4) \\
f_5(u, v) &= u \cdot v & \mathrm{adj}(u) &= \mathrm{adj}(f_5) \cdot v, & \mathrm{adj}(v) &= \mathrm{adj}(f_5) \cdot u
\end{aligned}
$$





Must sum contributions:
$f'(3) = 9.4121 - 16.4003 + 9 = 2.0118.$

Automatic Differentiation: OPM implementations

AutoDiffBlock (legacy)

- class implementing *forward AD*
- deals with *vectors of values* at a time
- derivatives are *sparse matrices*
- implemented with operator overloading
- based on Eigen library for basic types and operands
- helper library provides discrete div, grad etc.

Evaluation (new effort)

- class implementing *forward AD*
- deals with *a single scalar value* at a time
- derivatives are *compile-time-size vectors*
- implemented with operator overloading
- discrete div, grad must be implemented “manually”