

# Reservoir Modelling Using MATLAB - The MATLAB Reservoir Simulation Toolbox (MRST)

Knut-Andreas Lie

SINTEF Digital, Oslo, Norway

MATLAB Energy Conference, 17–18 November 2020

# MATLAB Reservoir Simulation Toolbox (MRST)

Transforming research on  
reservoir modelling

Unique prototyping platform:

- Standard data formats
- Data structures/library routines
- Fully unstructured grids
- Rapid prototyping:
  - Differentiation operators
  - Automatic differentiation
  - Object-oriented framework
  - State functions
- Industry-standard simulation

```
% Three-phase template model
fluid = initSimpleADIFluid('mu', [1, 5, 0]*centi*po
    'rho', [1000, 700, 0]*kilogram/meter^3'n',

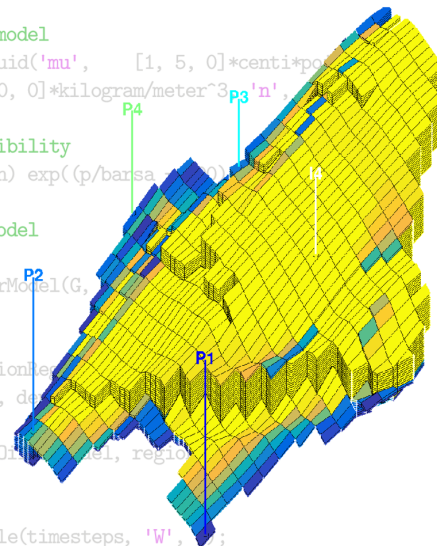
% Constant oil compressibility
fluid.b0 = @(p, varargin) exp((p/barsa

% Construct reservoir model
gravity reset on
model = TwoPhaseOilWaterModel(G,

% Define initial state
region = getInitializationRegion
    'datum_depth', de

state0 = initStateBlackOil

% Define schedule
schedule = simpleSchedule(timesteps, 'W',
```



## MATLAB Reservoir Simulation Toolbox (MRST)

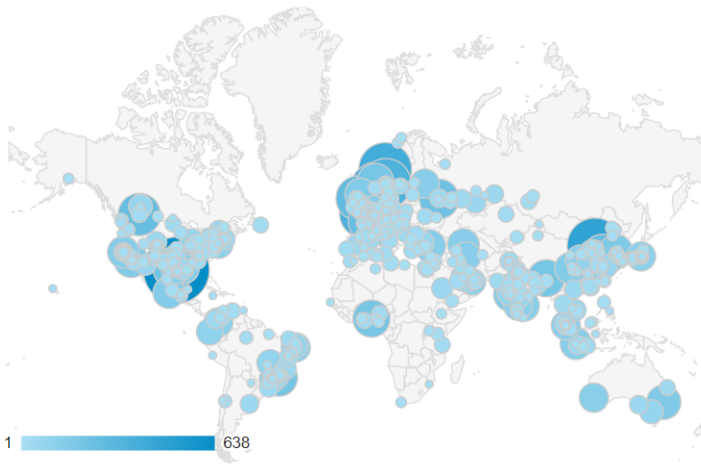
Transforming research on  
reservoir modelling

Large international user base:

- downloads from the whole world
- 123 master theses
- 56 PhD theses
- 226 journal papers (not by us)
- 144 proceedings papers

Numbers are from Google Scholar notifications

Used both by academia and industry



Google Analytics: access pattern for [www.mrst.no](http://www.mrst.no)  
Period: 1 July 2018 to 31 December 2019

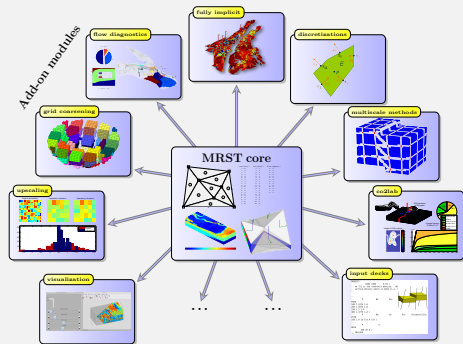
Different development process:

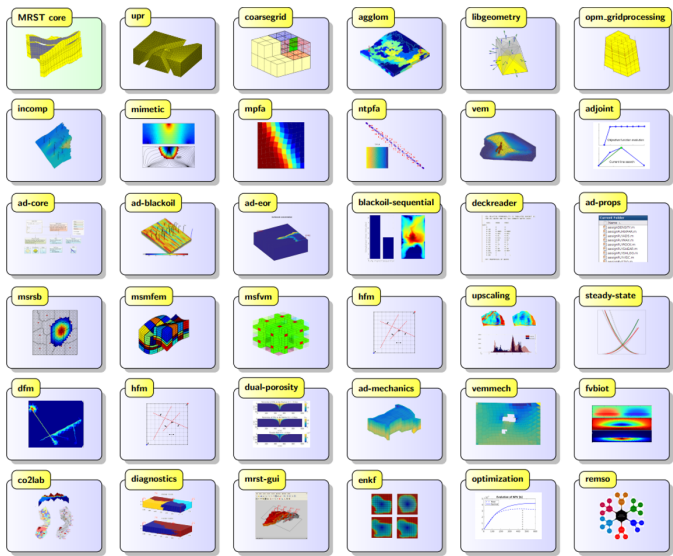
- Use abstractions to express your ideas in a form close to the underlying mathematics
  - Build your program using an interactive environment:
    - try out each operation and build program as you go
  - Dynamic type checking lets you prototype while you test an existing program:
    - run code line by line, inspect and change variables at any point
    - step back and rerun parts of code with changed parameters
    - add new behavior and data members while executing program
- MATLAB is fairly efficient using vectorization, logical indexing, external iterative solvers, etc.
  - Avoids build process, linking libraries, cross-platform problems
  - Builtin mathematical abstractions, numerics, data analysis, visualization, debugging/profiling,
  - Use scripting language as a wrapper when you develop solvers in compiled languages

## Modular design:

- **small core** with mature and well-tested functionality used in *many* programs or modules
- **semi-independent modules** that extend/override core functionality
- in-source documentation like in MATLAB
- all modules must have code examples and/or tutorials
- new development: project  $\rightarrow$  module

This simplifies how we distinguish public and in-house or client-specific functionality





- Grid generation and coarsening
- ECLIPSE input and output
- Upscaling / multiscale solvers
- Consistent discretizations
- Black-oil, EOR, compositional
- Fractures: DFM, EDFM, DPDP
- Geomechanics, geochemistry, geothermal
- Unsaturated media (Richards eq.)
- Multisegment wells (general network)
- CO2 storage laboratory
- Adjoints, optimization, ensembles
- Pre/postprocessing/visualization
- Flow diagnostics
- ...

3000 files, 213 000 code lines

MRST - MATLAB Reservoir Simulation Toolbox

MRST is a free open-source software for reservoir modeling and simulation, developed primarily by the Computational Geosciences group in the Department of Earth and Atmospheric Science at the University of Colorado at Boulder. It also includes third-party modules developed by researchers from several other universities (MIT, University of Glasgow, UCL, and TU/e).

**Basic functionality**

- Basic data management
- Visualization
- Perforation
- Rock data analysis
- Adaptive meshing
- Grid generation

**Distributions and solvers**

- ITIS
- Thyridon/TPFA
- TPFA
- TPFA2
- Adaptive meshing
- TPFA2

**Workflow tools**

- Grid generation
- Flow simulation
- Visualization
- Reservoir simulation
- TPFA2
- TPFA2

website

Welcome to the MRST User Group (1)  
Av MRST-users: The Matlab Reservoir Simulation Toolbox User Group - 6 innlegg - 648 visninger

Could you get the data? (9)  
Av jaeshw\_@gmail.com - 9 innlegg - 58 visninger

Set wells in geology model (and perforation intervals) (3)  
Av Aneceia Vlasovs - 3 innlegg - 7 visninger

EXAMPLE (2)  
Av 丑木成 - 2 innlegg - 18 visninger

Invalid MEX-file in the mac os platform (3)  
Av franko\_@hotmail.com - 3 innlegg - 11 visninger

Rate constraints for Compositional models (4)  
Av xcu\_@yahoo.com - 4 innlegg - 35 visninger

error during interpolation of vaporized oil table (2)  
Av ltu\_@gmail.com - 2 innlegg - 22 visninger

well equation in TwoPhaseWaterGasModel model (2)  
Av kai wang - 2 innlegg - 16 visninger

IncompTPFA generating pressure and flux Nan's..... (3)  
Av Paul Morris - 3 innlegg - 4 visninger

user forum

Access Open access  
Knut-Andreas Lie, SINTEF, Norway

An Introduction to Reservoir Simulation Using MATLAB/GNU Octave

User Guide for the MATLAB Reservoir Simulation Toolbox (MRST)

Publisher: Cambridge University Press  
Online publication date: July 2019  
Print publication year: 2019  
Online ISBN: 9781108591416

DOI: <https://doi.org/10.1017/9781108591416>

textbook

```
>> help computeTrans
Compute transmissibilities.

SYNOPSIS:
T = computeTrans(G, rock)
T = computeTrans(G, rock, 'ga', pf, ... )

PARAMETERS:
G - Grid structure as described by grid_structure.

rock - Rock data structure with valid field 'perm'. The permeability is assumed to be in measured in units of metres squared (m^2). Use function 'darcy' to convert from darcies to m^2, e.g.,

perm = convertFrom(perm, milliDarcy)

If the permeability is provided in units of millidarcies.

The field rock.perm may have ONE column for a scalar permeability in each cell, TWO/THREE columns for a diagonal permeability in each cell (in 2/3 D) and THREE/SIX columns for a symmetric full tensor permeability. In the latter case, each cell gets the permeability tensor

K_i = [ k1 k2 ] in two space dimensions
      [ k2 k3 ]

RETURNS:
T - half-transmissibilities for each local face of each grid. The number of half-transmissibilities equals the number
```

manpages

Basic Flow Solver Tutorial

The purpose of this tutorial is to guide you through the steps to set up and solve a standard two-phase problem using the MATLAB Reservoir Simulation Toolbox (MRST).

For a flow driven by Dirichlet and Neumann boundary conditions. Our geological model will be simple: a single grid with anisotropic, homogeneous permeability.

In this tutorial, we will also discuss:

1. the grid structure,
2. how to specify wells and fluid data,
3. the structure of the data objects used to build solutions,
4. how to execute the solver and how to post-process the results.

Useful links for modeling and interacting with the grid and simulation results:

- [Data objects](#)
- [Basic objects](#)
- [Wells and fluid data](#)
- [Basic simulation objects](#)
- [Basic data objects](#)
- [Basic data](#)

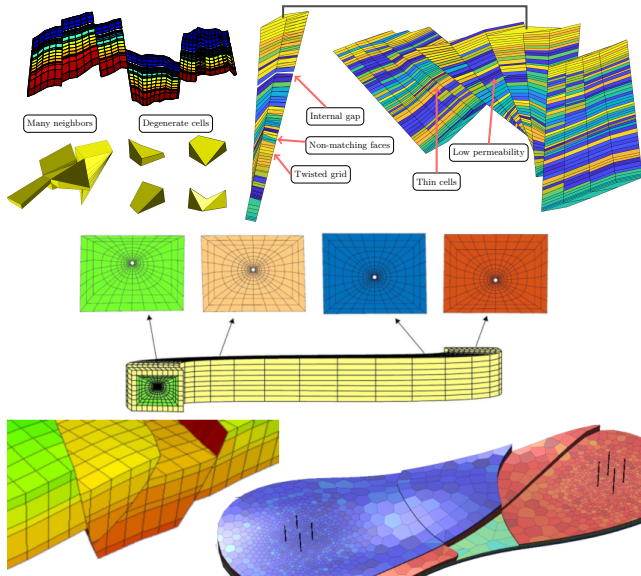
Basic geometry

Geometry is a structure grid of size [20, 20, 20] by default, where each cell has dimensions [1, 1, 1] by [1, 1, 1]. Because our flow solver is applicable for general anisotropic grids, the following is how you would create a grid with anisotropic permeability:

```
(N) = 20; (Y) = 20; (Z) = 20;
C = createGrid(N, Y, Z);
flowGrid(C);
```

tutorial codes

online tutorials



A wide variety of grid formats:

- Cartesian and rectilinear
- Corner-point
- Tetrahedral, prismatic, PEBI
- General polyhedral/polytopal
- Hybrid, cut-cell, or depogrids
- Local refinements . . .

MRST grids are chosen to always be **fully unstructured**

→ can implement algorithms without knowing the specifics of the grid

Also: coarse grids made as static or dynamic partitions of fine grid

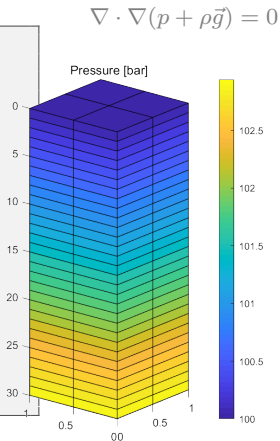


```

%% Define the model
gravity reset on
G      = cartGrid([2, 2, 30], [1, 1, 30]);
G      = computeGeometry(G);
rock.perm = repmat(0.1*darcy(), [G.cells.num, 1]);
fluid   = initSingleFluid();
bc = pside([], G, 'TOP', 1:G.cartDims(1), ...
           1:G.cartDims(2), 100.*barsa());

%% Assemble and solve the linear system
S = computeMimeticIP(G, rock);
sol = solveIncompFlow(initResSol(G, 0.0), ...
                     initWellSol([], 0.0), ...
                     G, S, fluid, 'bc', bc);

%% Plot the face pressures
newplot;
plotFaces(G, 1:G.faces.num, sol.facePressure./barsa);
set(gca, 'ZDir', 'reverse'), title('Pressure [bar]')
view(3), colorbar
    
```

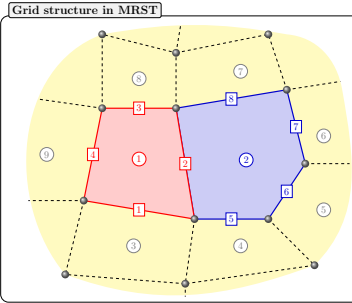
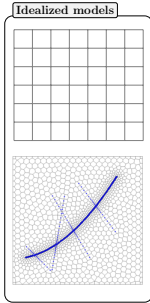


Oldest part of MRST:

- Procedural programming
- Structs for reservoir state, rock parameters, wells, b.c., and source term
- Fluid behavior: struct with function pointers

Advantages:

- **hide specific details** of geomodel and fluid model
- **vectorization**: efficient/compact code
- **unified access** to key parameters

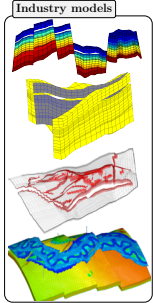


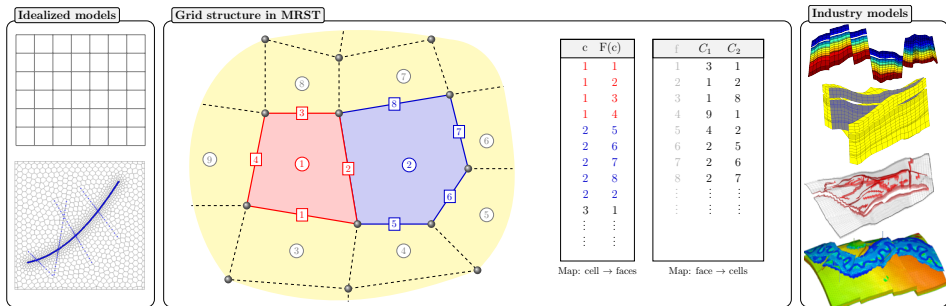
c	F(c)
1	1
1	2
1	3
1	4
2	5
2	6
2	7
2	8
2	2
3	1
⋮	⋮
⋮	⋮

Map: cell → faces

f	C <sub>1</sub>	C <sub>2</sub>
1	3	1
2	1	2
3	1	8
4	9	1
5	4	2
6	2	5
7	2	6
8	2	7
⋮	⋮	⋮
⋮	⋮	⋮

Map: face → cells





For finite volumes, discrete grad operator maps from cell pair  $C_1(f), C_2(f)$  to face  $f$ :

$$\text{grad}(\mathbf{p})[f] = \mathbf{p}[C_2(f)] - \mathbf{p}[C_1(f)],$$

where  $\mathbf{p}[c]$  is a scalar quantity associated with cell  $c$ . Discrete div maps from faces to cells

Both are linear operators and can be represented as sparse matrix multiplications

## Continuous

Incompressible flow:

$$\nabla \cdot (\mathbf{K}\nabla p) + q = 0$$

Compressible flow:

$$\frac{\partial(\phi\rho)}{\partial t} + \nabla \cdot (\rho\mathbf{K}\nabla p) + q = 0$$

## Discrete in MATLAB

Incompressible flow:

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

Compressible flow:

```
eq = (pv(p).*rho(p)-pv(p0).*rho(p0))/dt ...  
+ div(avg(rho(p)).*T.*grad(p))+q;
```

## Continuous

Incompressible flow:

$$\nabla \cdot (\mathbf{K}\nabla p) + q = 0$$

Compressible flow:

$$\frac{\partial(\phi\rho)}{\partial t} + \nabla \cdot (\rho\mathbf{K}\nabla p) + q = 0$$

## Discrete in MATLAB

Incompressible flow:

$$\text{eq} = \text{div}(\mathbf{T} .* \text{grad}(p)) + q;$$

Compressible flow:

$$\text{eq} = (\text{pv}(p) .* \text{rho}(p) - \text{pv}(p0) .* \text{rho}(p0)) / \text{dt} \dots \\ + \text{div}(\text{avg}(\text{rho}(p)) .* \mathbf{T} .* \text{grad}(p)) + q;$$

Discretization of flow models leads to large systems of nonlinear equations. Can be linearized and solved with Newton's method

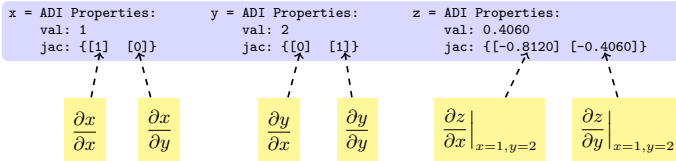
$$\mathbf{F}(\mathbf{u}) = \mathbf{0} \quad \Rightarrow \quad \frac{\partial \mathbf{F}}{\partial \mathbf{u}}(\mathbf{u}^i)(\mathbf{u}^{i+1} - \mathbf{u}^i) = -\mathbf{F}(\mathbf{u}^i)$$

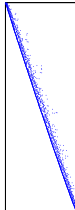
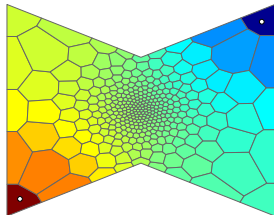
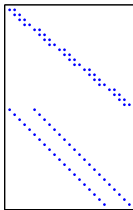
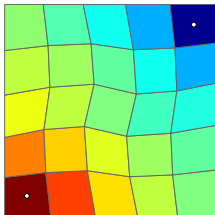
Coding necessary Jacobians is time-consuming and error prone

General idea:

- Any code consists of a limited set of arithmetic operations and elementary functions
- Introduce an extended pair,  $\langle x, 1 \rangle$ , i.e., the value  $x$  and its derivative 1
- Use chain rule and elementary derivative rules to mechanically accumulate derivatives *at specific values of  $x$* 
  - Elementary:  $v = \sin(x) \quad \longrightarrow \quad \langle v \rangle = \langle \sin x, \cos x \rangle$
  - Arithmetic:  $v = fg \quad \longrightarrow \quad \langle v \rangle = \langle fg, fg_x + f_xg \rangle$
  - Chain rule:  $v = \exp(f(x)) \quad \longrightarrow \quad \langle v \rangle = \langle \exp(f(x)), \exp(f(x))f'(x) \rangle$
- Use operator overloading to avoid messing up code

```
[x,y] = initVariablesADI(1,2);
z = 3*exp(-x*y)
```





```

% Make grid
G = twister(cartGrid([8 8]));
G = computeGeometry(G);

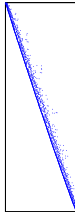
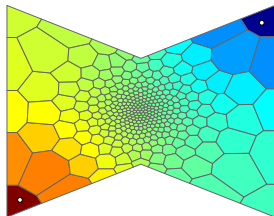
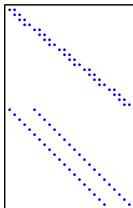
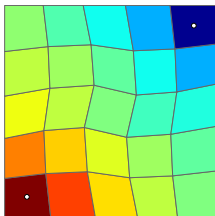
% Set source terms (flow SW -> NE)
q = zeros(G.cells.num,1);
q([1 end]) = [1 -1];

% Unit isotropic permeability
K = ones(G.cells.num,4); K(:,[2 3]) = 0;
    
```

```

% Make grid using external grid generator
pv = [-1 -1; 0 -.5; 1 -1; 1 1; 0 .5; -1 1; -1 -1];
fh = @(p,x) 0.025 + 0.375*sum(p.^2,2);
[p,t] = distmesh2d(@dpoly, fh, 0.025, [-1 -1; 1 1], pv, pv);
G = computeGeometry(pebi(triangleGrid(p, t)));

% Set source terms (flow SW -> NE)
q = zeros(G.cells.num,1);
v = sum(G.cells.centroids,2);
[-,i1]=min(v); [-,i2]=max(v);
q([i1 i2]) = [1 -1];
    
```



```
% Make grid
G = twister(cartGrid([8 8]));
G = computeGeometry(G);

% Set source terms (flow SW -> NE)
q = zeros(G.cells.num,1);
q([1 end]) = [1 -1];

% Unit isotropic permeability
K = ones(G.cells.num,4); K(:,[2 3]) = 0;
```

```
% Make grid using external grid generator
pv = [-1 -1; 0 -.5; 1 -1; 1 1; 0 .5; -1 1; -1 -1];
fh = @(p,x) 0.025 + 0.375*sum(p.^2,2);
[p,t] = distmesh2d(@dpoly, fh, 0.025, [-1 -1; 1 1], pv, pv);
G = computeGeometry(pebi(triangleGrid(p, t)));

% Set source terms (flow SW -> NE)
q = zeros(G.cells.num,1);
v = sum(G.cells.centroids,2);
[-,i1]=min(v); [-,i2]=max(v);
q([i1 i2]) = [1 -1];
```

```
S = setupOperatorsTPFA(G,rock);
p = initVariablesADI(zeros(G.cells.num,1));
eq = S.Div(S.T .* S.Grad(p)) + q;
eq(1) = eq(1) + p(1);
p = -eq.jac{1}\eq.val;
```

```
% Define Div, Grad, etc
% Initialize p as AD variable
% Residual equation: F = Ap + q
% Fixate pressure
% Solve system A
```



```
[p, sW] = initVariablesADI(p0,
[pIx, sIx] = deal(1:nc, (nc+1):(2*
[tol, maxits] = deal(1e-5, 15);
t = 0;
while t < totTime,
    t = t + dt;
    resNorm = 1e99; nit=0;
    [p0, sW0] = deal(value(p), value(
    while (resNorm > tol) && (nit <=
        % one Newton iteration
    end
    if nit > maxits,
        error('Newton solves did not
    end
end
```

```
% Evaluate equations
```

```
[rW, r0, vol] = deal(rhoW(p), rho0(p), pv(p));
```

```
:
```

```
water = (vol.*rW.*sW - vol0.*rW0.*sW0)./dt + div(vW);
```

```
water(inIx) = water(inIx) - inRate.*rhoWS;
```

```
:
```

```
eqs = {oil, water}; % concatenate equations
```

```
eq = cat(eqs{:}); % assemble
```

```
res = eq.val; % residual
```

```
upd = -(eq.jac{1} \ res); % Newton update
```

```
% Update variables
```

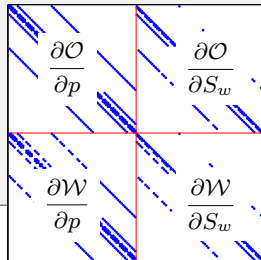
```
p.val = p.val + upd(pIx);
```

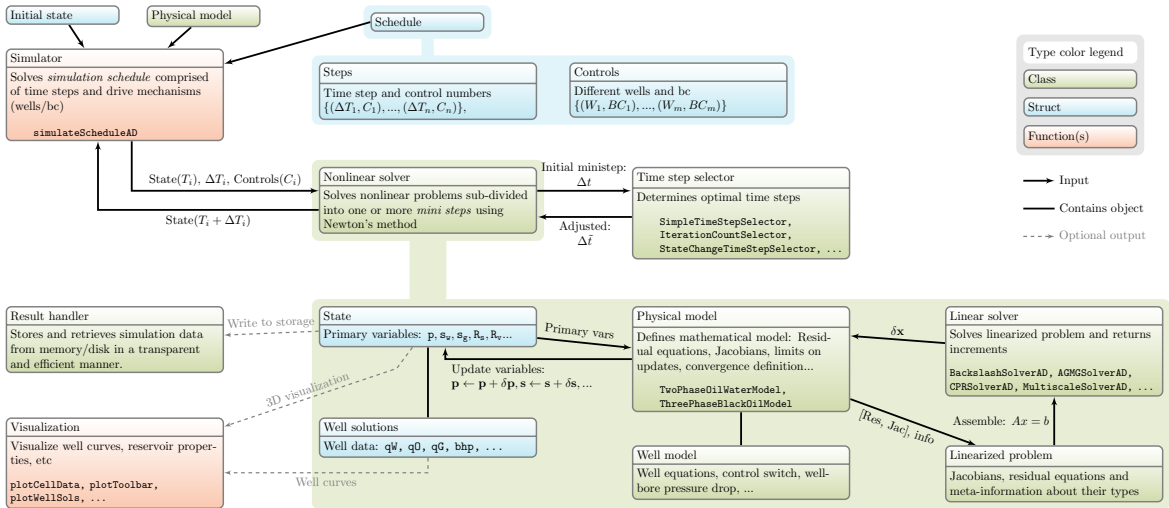
```
sW.val = sW.val + upd(sIx);
```

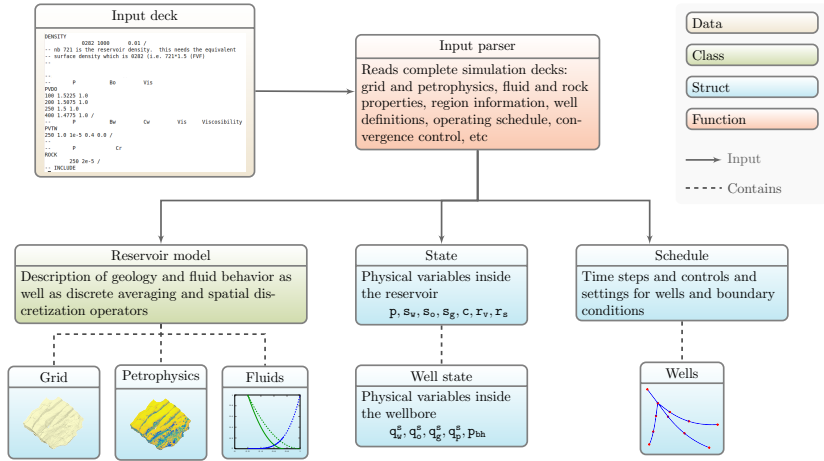
```
sW.val = max( min(sW.val, 1), 0);
```

```
resNorm = norm(res);
```

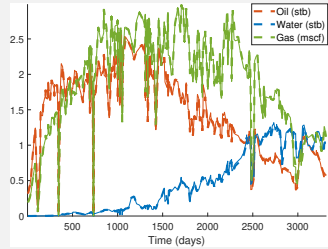
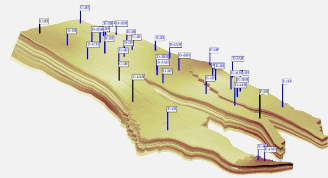
```
nit = nit + 1;
```







## Example:



Field production compared with OPM Flow for the Norne field

It would be convenient to have:

- **Dependency management:** keep track of dependency graph, ensure all input quantities have been evaluated before evaluating a function
- **Generic interfaces:** avoid defining functional dependencies explicitly, e.g.,  $G(S)$ , and  $G(p, S)$ .
- Lazy evaluation with caching
- Enable spatial dependence in parameters while preserving vectorization potential
- Implementation independent of the choice of primary variables

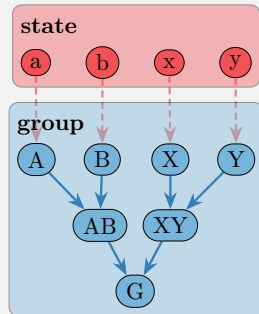
It would be convenient to have:

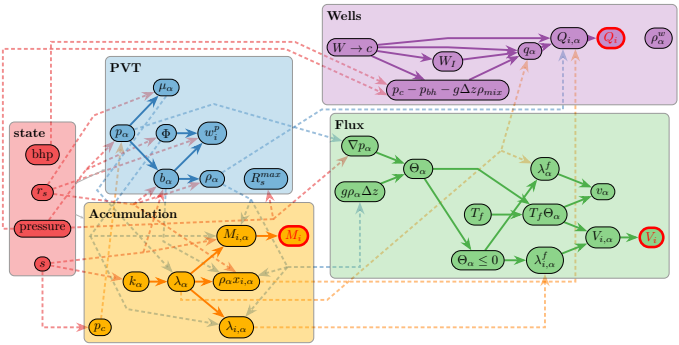
- **Dependency management:** keep track of dependency graph, ensure all input quantities have been evaluated before evaluating a function
- **Generic interfaces:** avoid defining functional dependencies explicitly, e.g.,  $G(S)$ , and  $G(p, S)$ .
- Lazy evaluation with caching
- Enable spatial dependence in parameters while preserving vectorization potential
- Implementation independent of the choice of primary variables

**State function:** any function that is uniquely determined by the contents of the state struct alone

Implemented as class objects, gathered in functional groups

$$G(x, y, a, b) = xy + ab$$





**Example:** State-function diagram for a simple black-oil model. Each entity is a state function that is easy to replace.

Idea: apply this concept to

- flow property evaluation
- PVT calculations
- accumulation, flux, and source terms
- spatial/temporal discretization

Simulator → differentiable graph

Further granularity

- Immiscible components
- Black-oil type components
- Compositional components
- Concentration components

Combined at runtime to form compact models with only necessary unknowns

Total time of a program consists of several parts:

- programming + debugging

- + documenting + testing + executing

MRST is designed to prioritize the first four over the last

Does this mean that MRST is slow and not scalable?

Total time of a program consists of several parts:

programming + debugging  
+ documenting + testing + executing

MRST is designed to prioritize the first four over the last

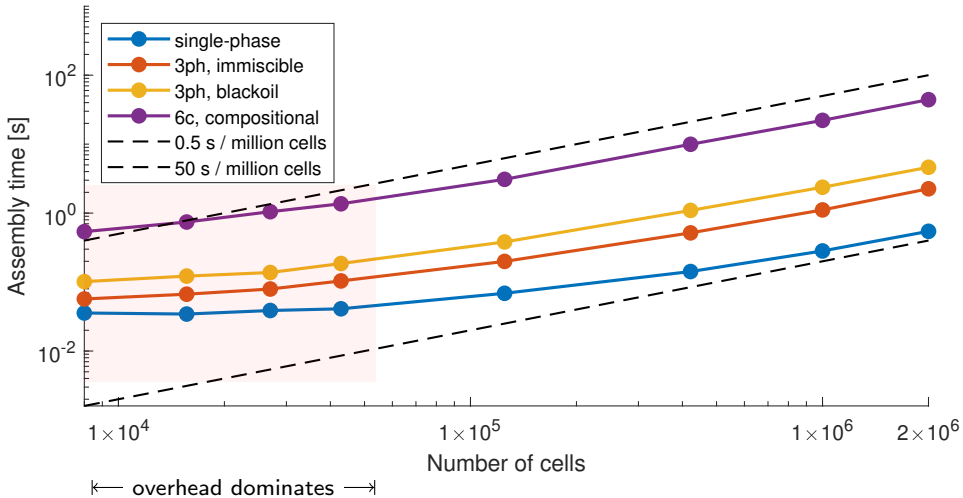
Does this mean that MRST is slow and not scalable?

No, I would say its is surprisingly efficient

Potential concerns:

- MATLAB is interpreted  
cure: JIT, vectorization, logical indexing, pre-allocation, highly-efficient libraries
- Redundant computations  
cure: state functions = dependency graph + computational cache
- Computational overhead  
cure: new auto diff backends
- Scalability/performance  
cure: external high-end iterative solvers





New AD backends: storage optimized wrt access pattern, MEX-accelerated operations

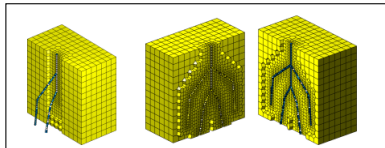
Interface to external linear algebra packages implemented as classes in AD-OO framework

**Example:** compressible three-phase, black-oil problem

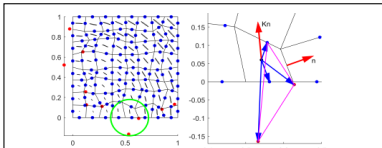
Solver	Req.	8,000 cells	125,000 cells	421,875 cells	1,000,000 cells
LU	–	2.49 s	576.58 s	–	–
CPR*	–	0.90 s	137.30 s	–	–
CPR*	AGMG	0.18 s	3.60 s	13.78 s	43.39 s
CPR*	AMGCL	0.21 s	3.44 s	16.20 s	51.35 s
CPR	AMGCL	0.07 s	0.43 s	3.38 s	10.20 s
CPR	AMGCL <sup>†</sup>	0.05 s	0.86 s	1.97 s	5.60 s
CPR	AMGCL <sup>‡</sup>	0.05 s	0.38 s	1.33 s	3.82 s

\* – in MATLAB, † – block AMGCL (block ILU + AMG/CPR), ‡ – block AMGCL with tweaks

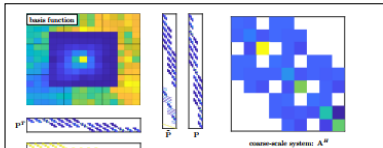
Performance is approaching commercial and compiled academic codes



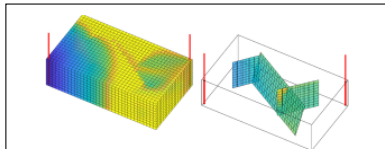
Berge et al.: Constrained Voronoi grids



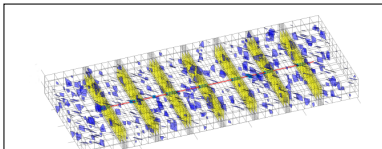
Al Kobaisi & Zhang: nonlinear FVM



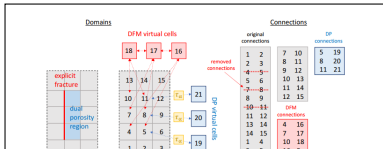
Lie & Møyner: multiscale methods



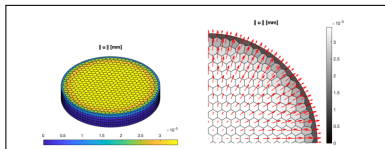
Wong et al.: embedded discrete fractures



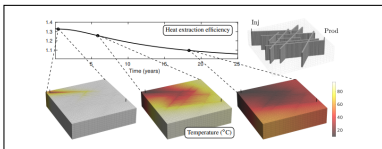
Olorode et al.; fractured unconventional



March et al.: unified framework, fractures



Varela et al.: unsaturated poroelasticity



Collignon et al.: geothermal systems

- Klemetsdal & Lie: discontinuous Galerkin
- Møyner: state functions, AD backends
- Sun et al.: chemical EOR
- Møyner: compositional
- Andersen: coupled flow & geomechanics

Thanks to all my co-developers at SINTEF (Olav Møyner, in particular), our master and PhD students, and our national and international collaborators.

Thanks also to all MRST users who have asked interesting questions that have helped us shape the software

## Funding:

- Research Council of Norway
- SINTEF
- Equinor: gold open access for the MRST textbook
- Chevron, Ecopetrol, Eni, Equinor, ExxonMobil, Shell, SLB, Total, Wintershall DEA, . . .