OPM + Reaktoro

David Landa-Marbán and Tor Harald Sandve

Computational Geosciences and Modelling

dmar@norceresearch.no

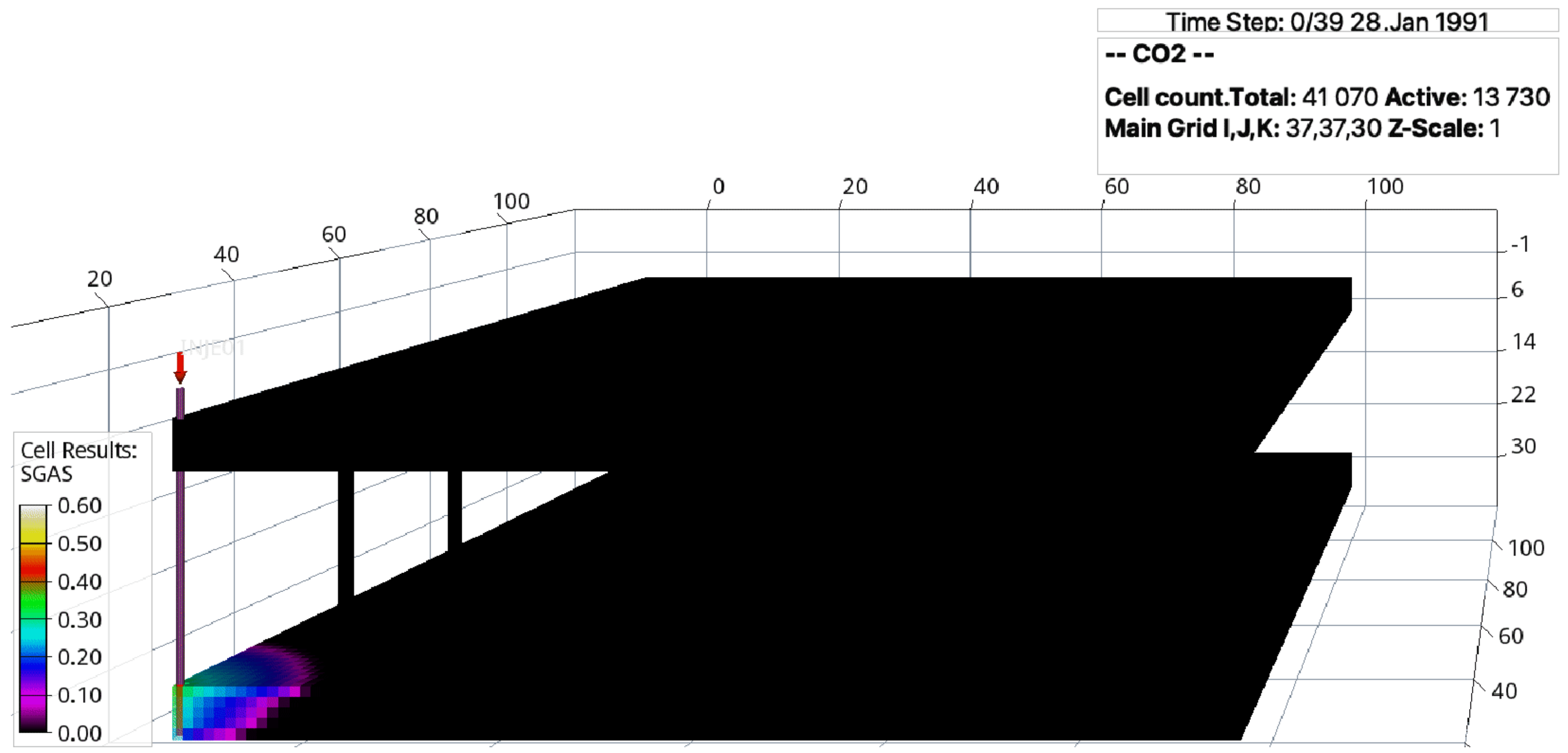
OPM summit, Trondheim - August 30 - 31st, 2022

Outline

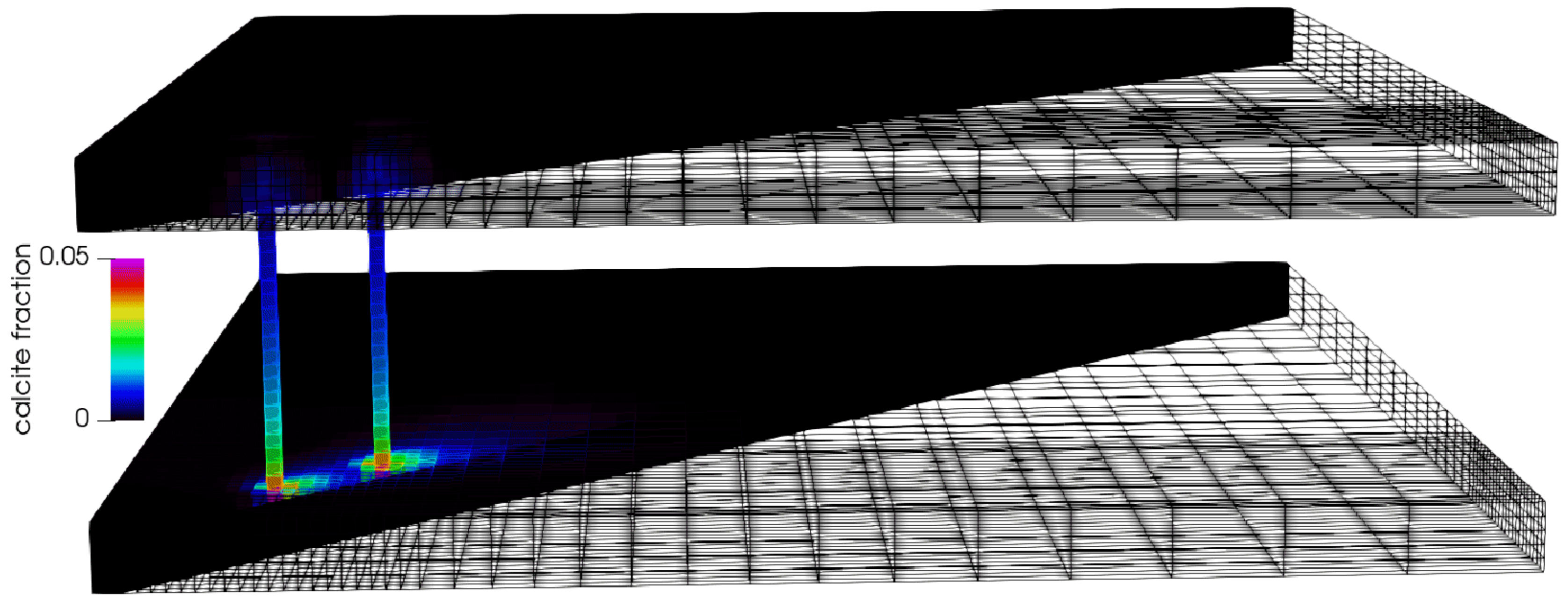
- Past and current OPM related work
- Overview of Reaktoro
- Coupling OPM to Reaktoro: Proof of concept
- Numerical results
- Current work

py-micp: An Open-Source Simulation Workflow for Field-Scale Application of Microbially Induced Calcite Precipitation Technology for Leakage Remediation

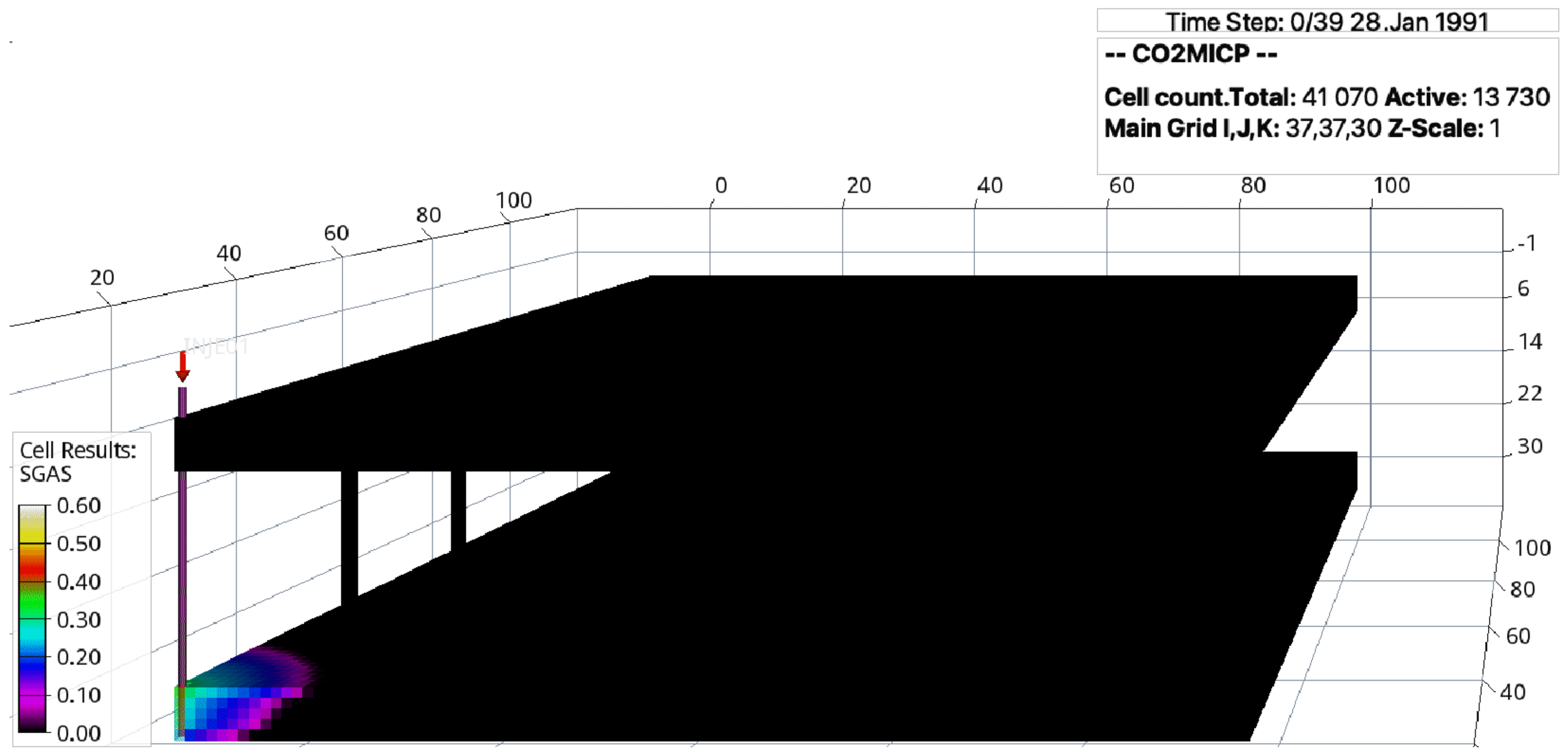
CO2 migration before MICP treatment


Time Step: 0/39 28.Jan 1991
-- CO2 --
Cell count.Total: 41 070 Active: 13 730
Main Grid I,J,K: 37,37,30 Z-Scale: 1
Cell Results:
SGAS
0.60
0.50
0.40
0.30
0.20
0.10
0.00

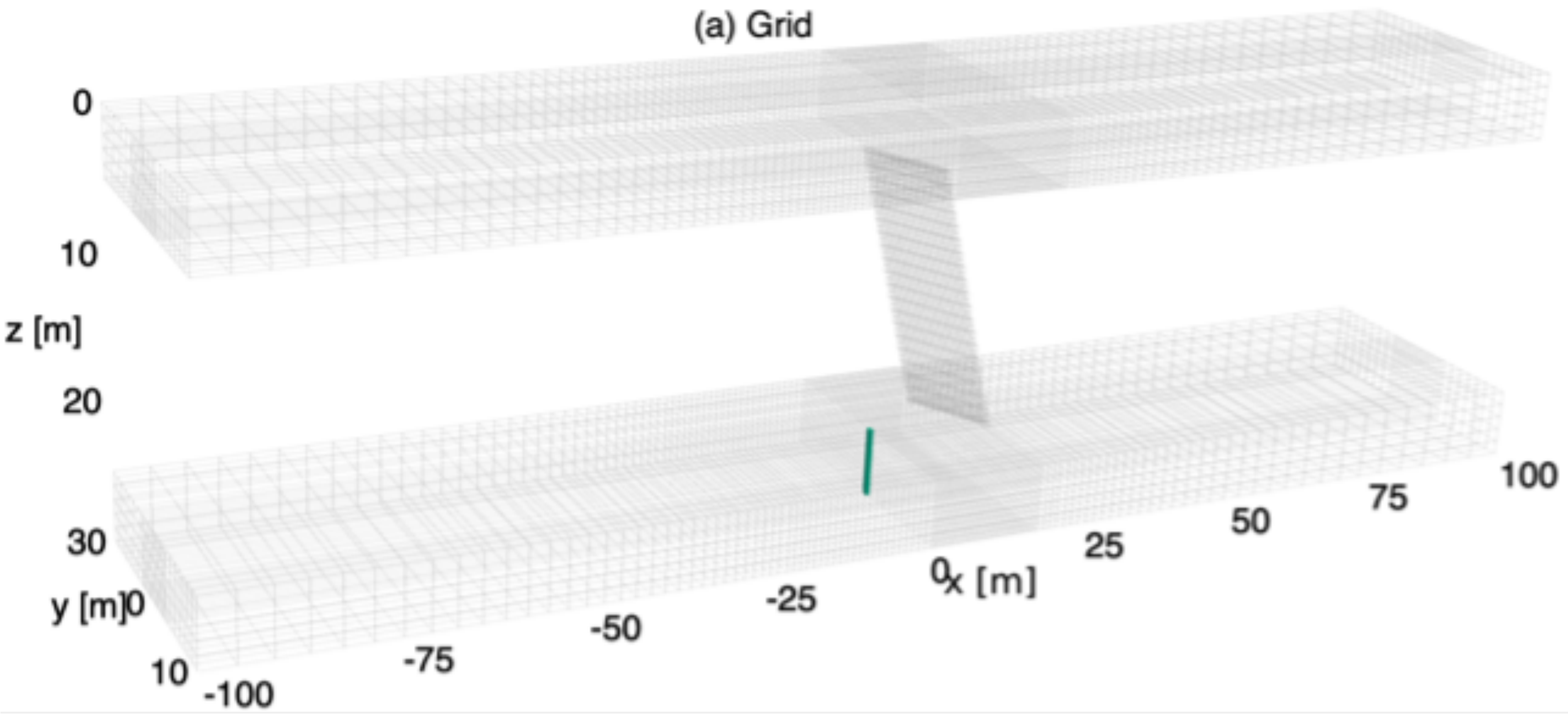
Final calcite distribution after MICP treatment



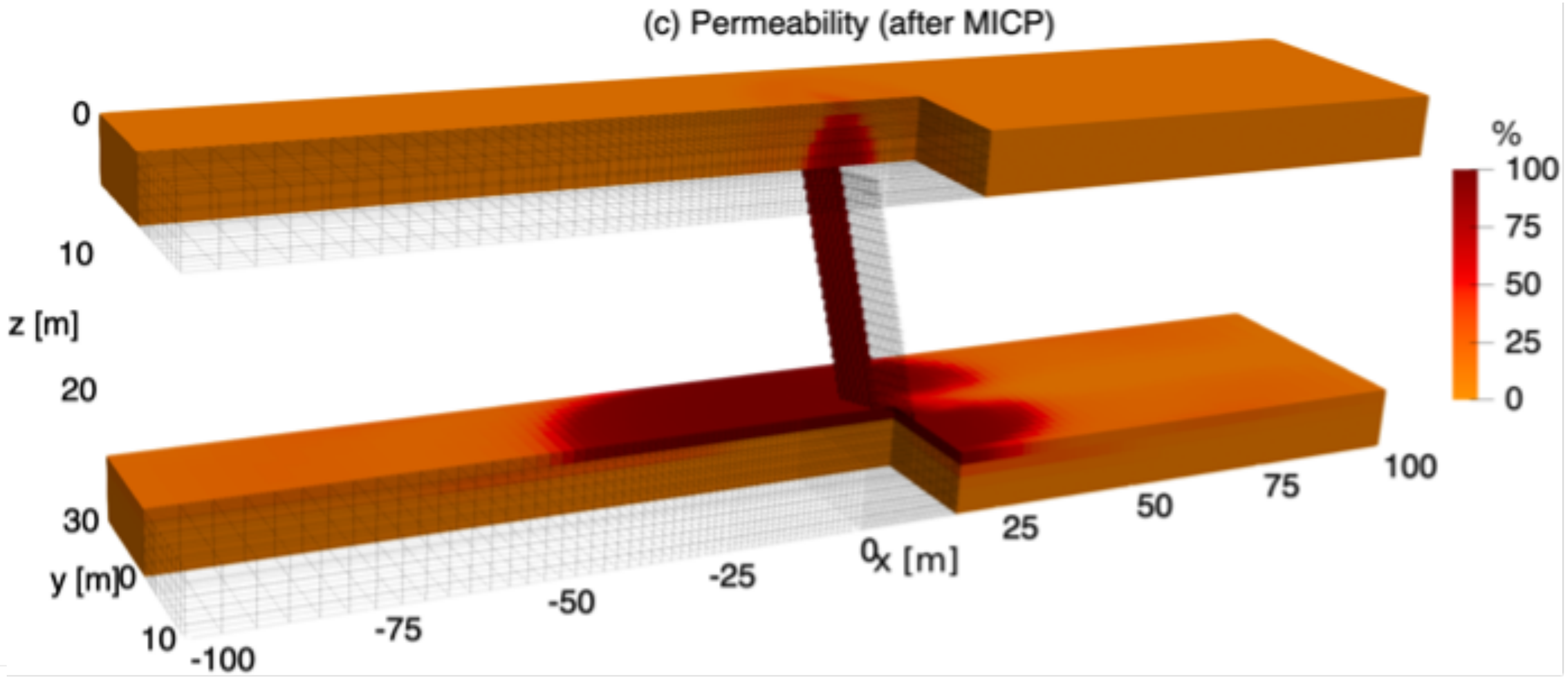
CO2 migration after MICP treatment


Time Step: 0/39 28.Jan 1991
-- CO2MICP --
Cell count.Total: 41 070 Active: 13 730
Main Grid I,J,K: 37,37,30 Z-Scale: 1
Cell Results:
SGAS
0.60
0.50
0.40
0.30
0.20
0.10
0.00

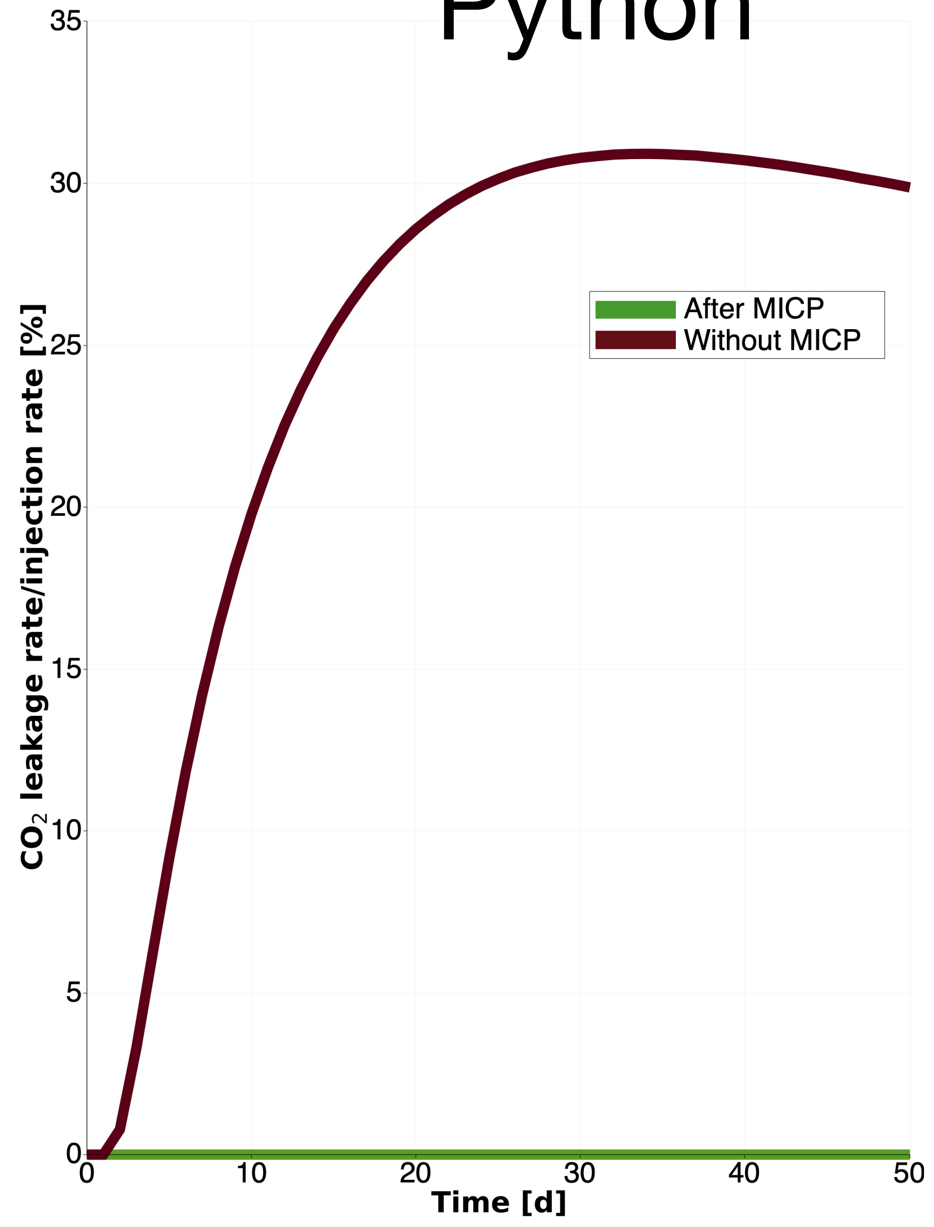
MRST



OPM



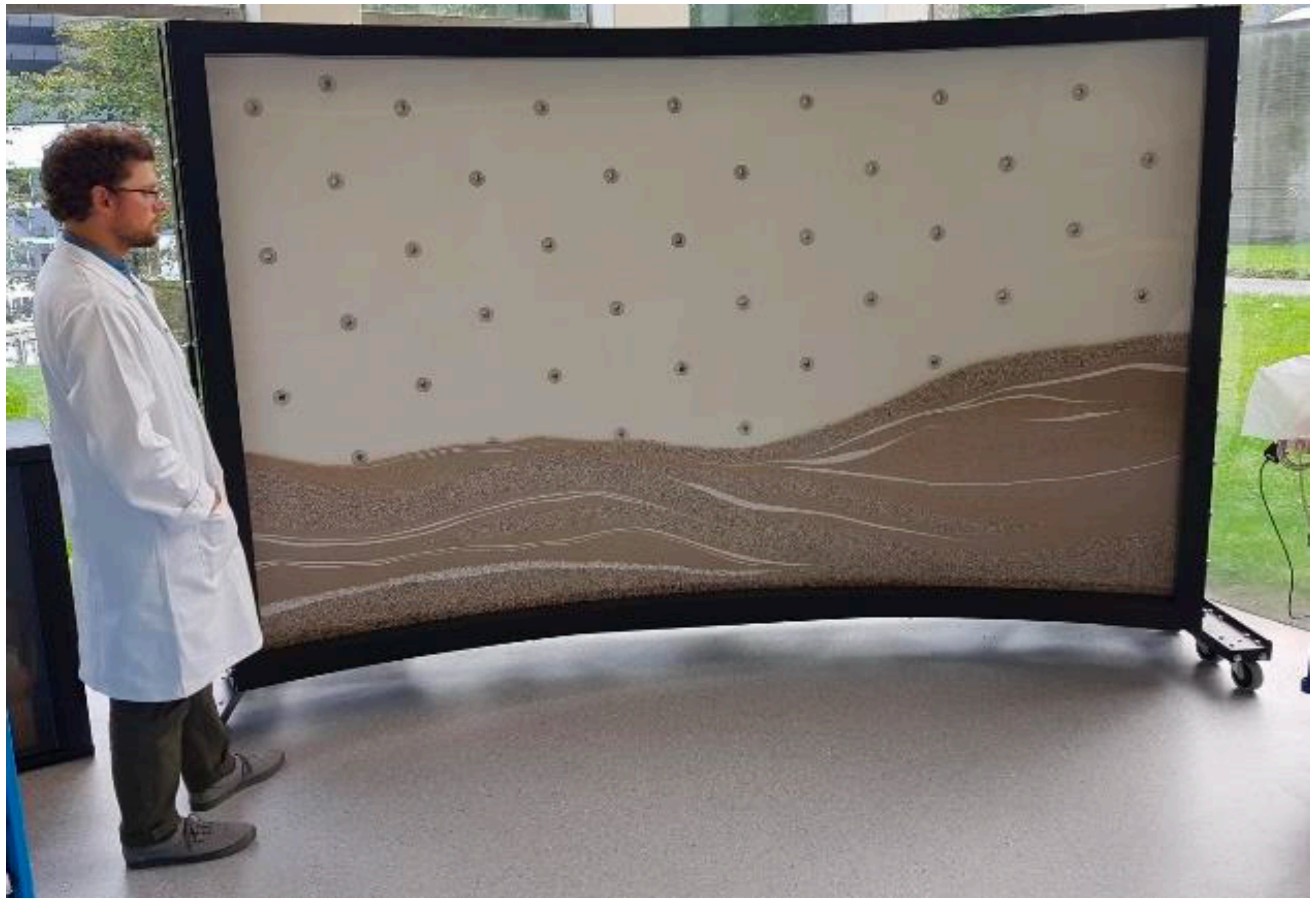
Python

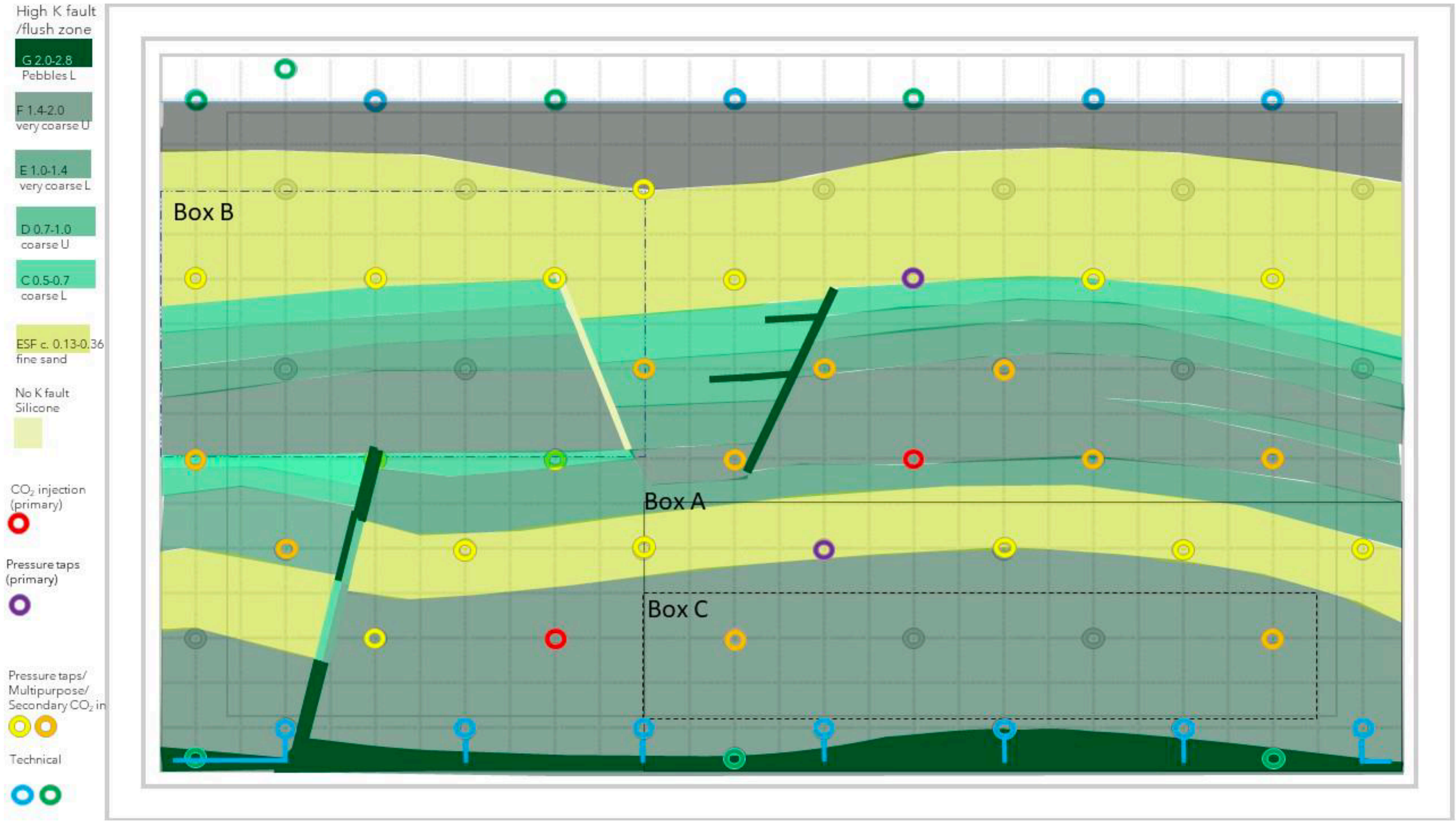


https://github.com/daavid00/py-micp

FluidFlower international benchmark study

J.M. Nordbotten, M. Fernø, B. Flemisch, R. Juanes, M. Jørgensen

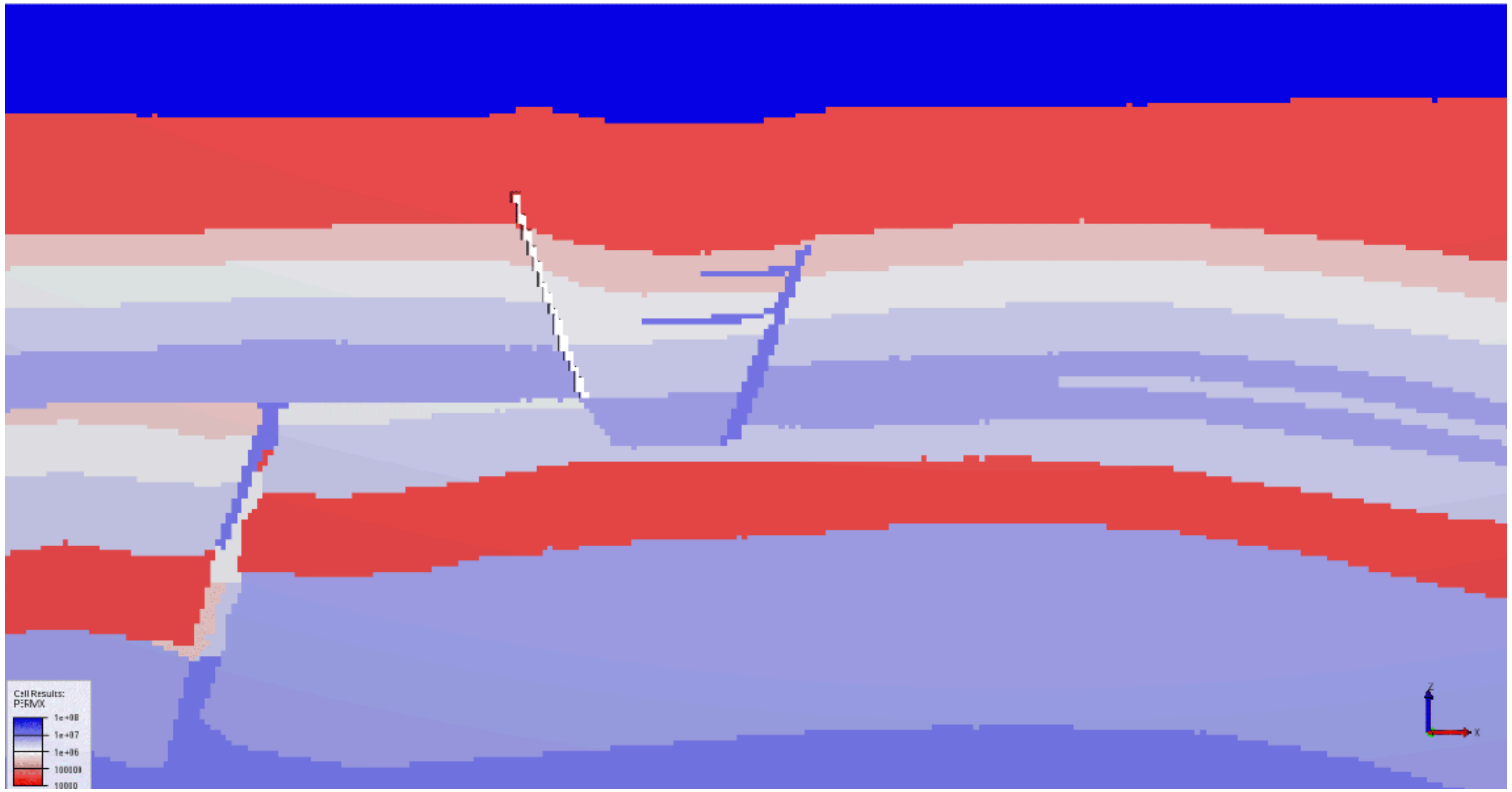




- Objective: To provide a full-physics validation of the state-of-art simulation capabilities within the international porous media community.

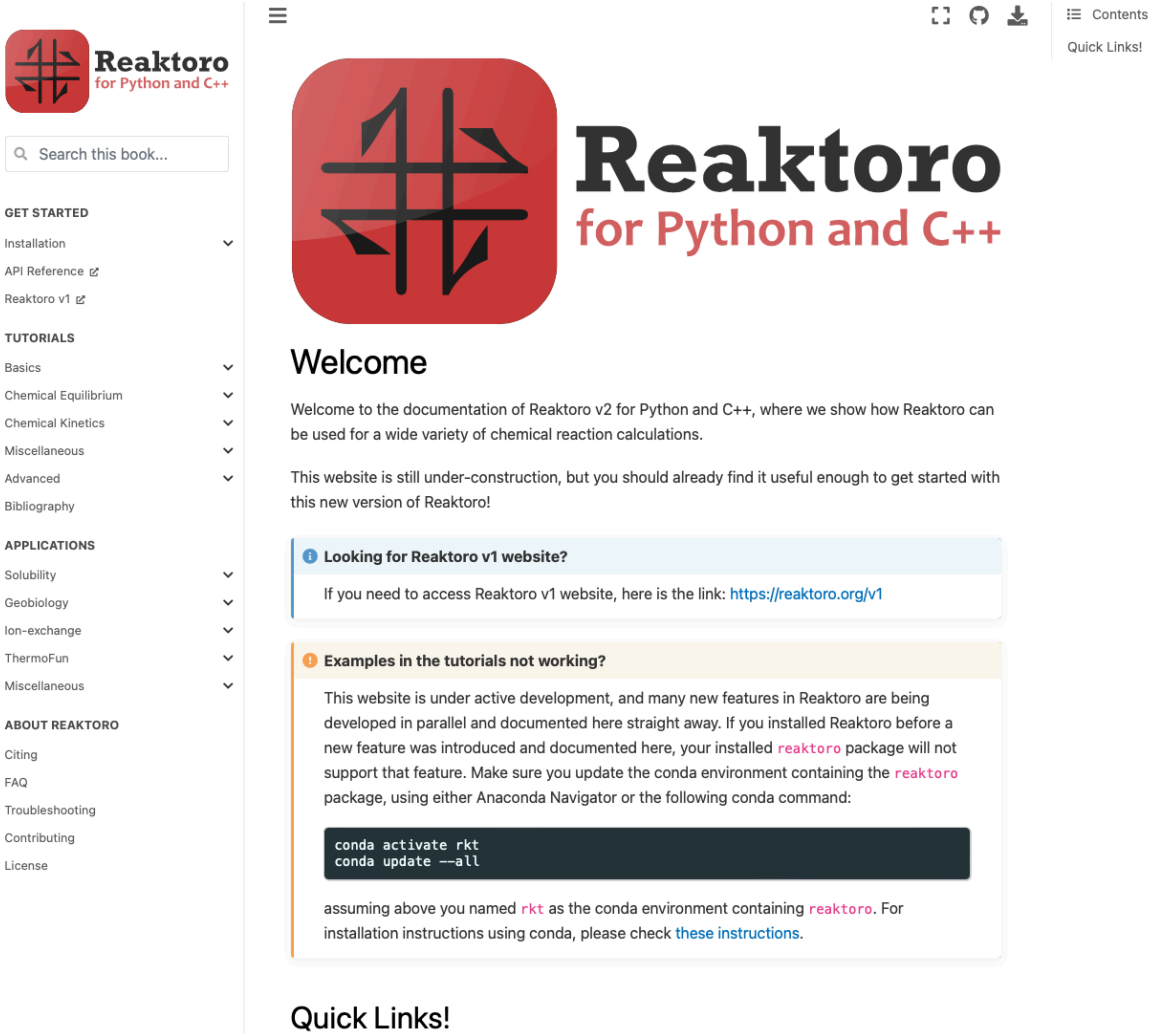
https://zenodo.org/record/6807102#.YwQQAy0RrT8 https://fluidflower.w.uib.no

pyff: An open-source history matching framework for the Fluid Flower Benchmark study



https://github.com/daavid00/pyff

Reaktoro



Welcome

Welcome to the documentation of Reaktoro v2 for Python and C++, where we show how Reaktoro can be used for a wide variety of chemical reaction calculations.

This website is still under-construction, but you should already find it useful enough to get started with this new version of Reaktoro!

Looking for Reaktoro v1 website?

If you need to access Reaktoro v1 website, here is the link: https://reaktoro.org/v1

Examples in the tutorials not working?

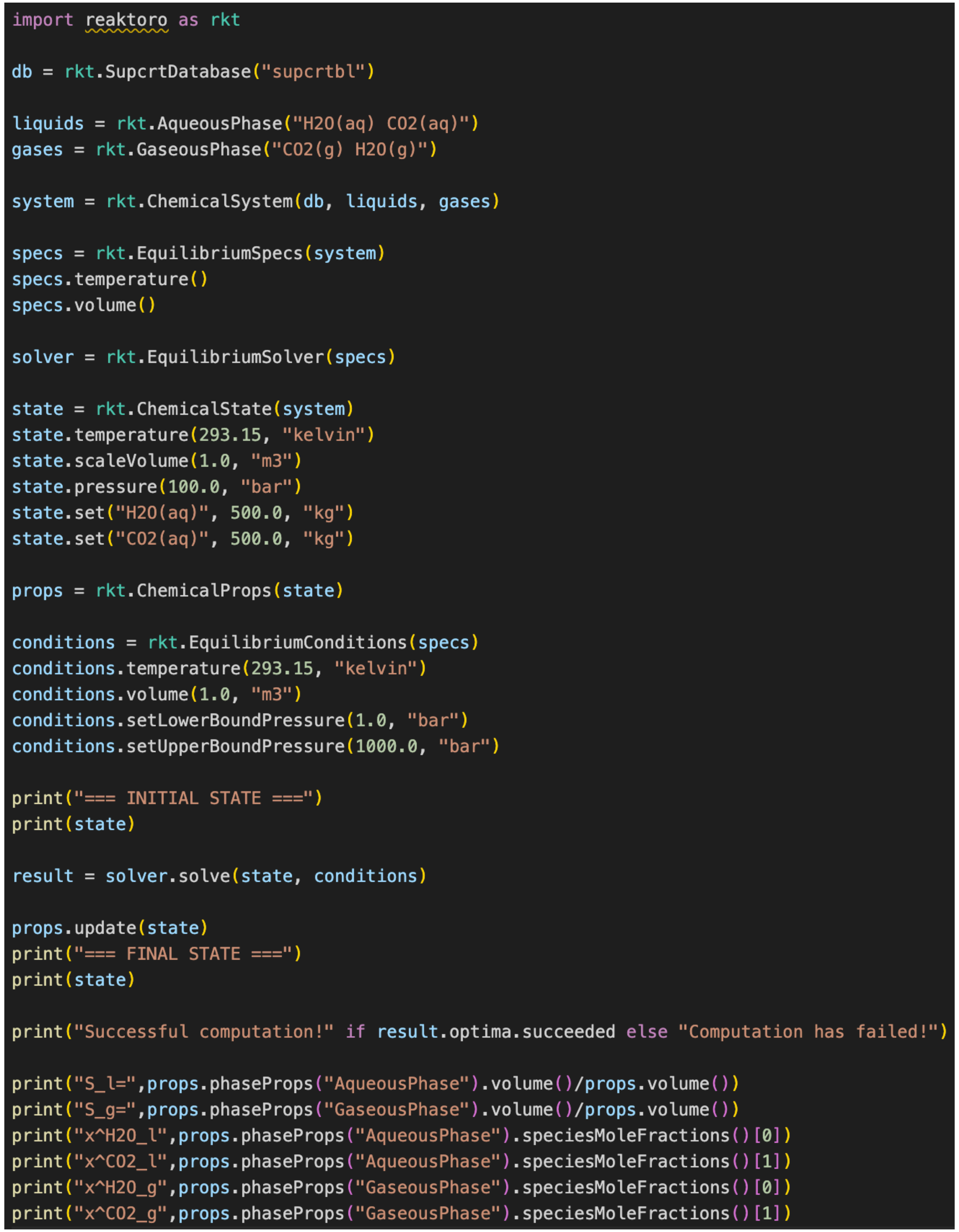
This website is under active development, and many new features in Reaktoro are being developed in parallel and documented here straight away. If you installed Reaktoro before a new feature was introduced and documented here, your installed `reaktoro` package will not support that feature. Make sure you update the conda environment containing the `reaktoro` package, using either Anaconda Navigator or the following conda command:

```
conda activate rkt
conda update --all
```

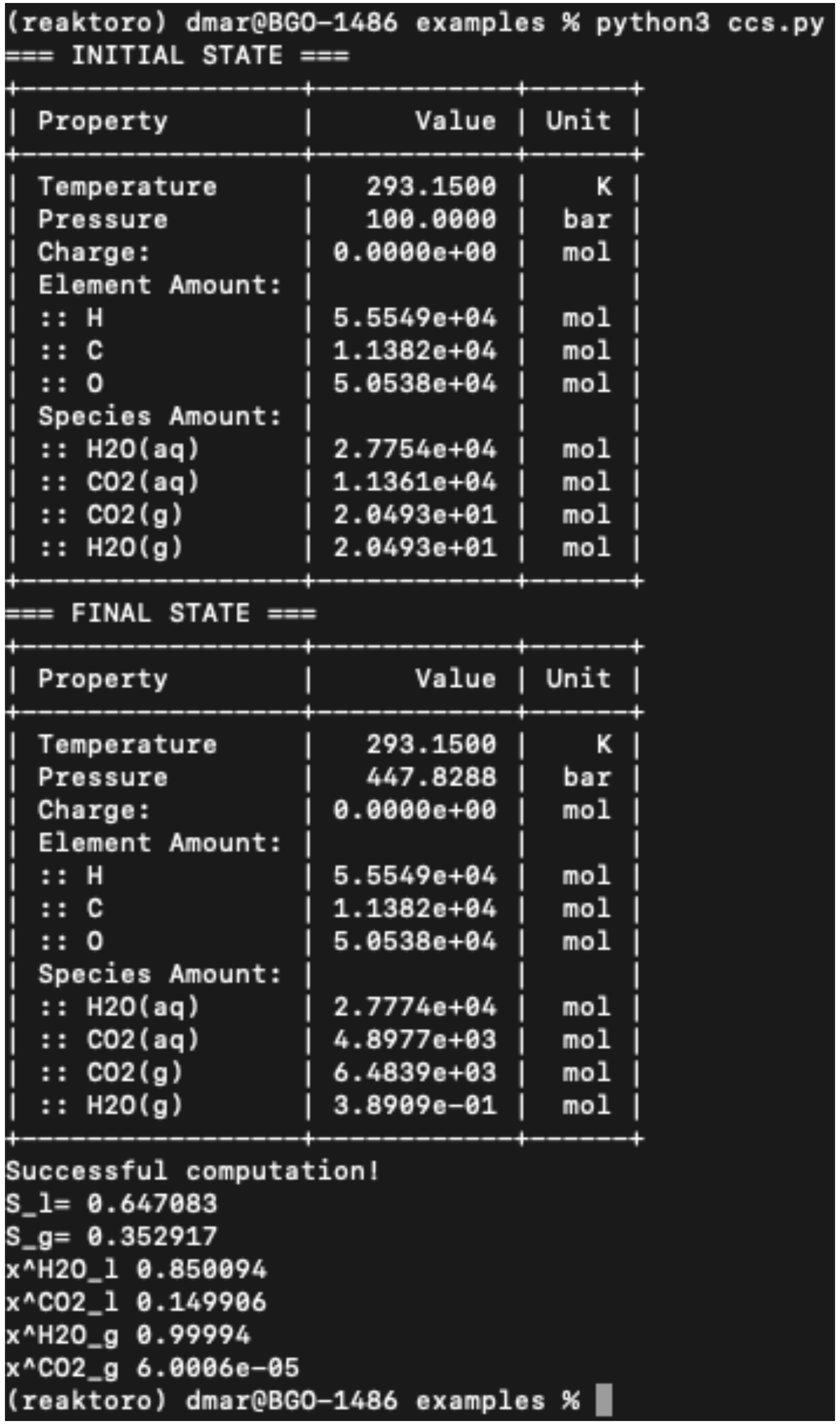
assuming above you named `rkt` as the conda environment containing `reaktoro`. For installation instructions using conda, please check these instructions.

Quick Links!

https://reaktoro.org/intro.html

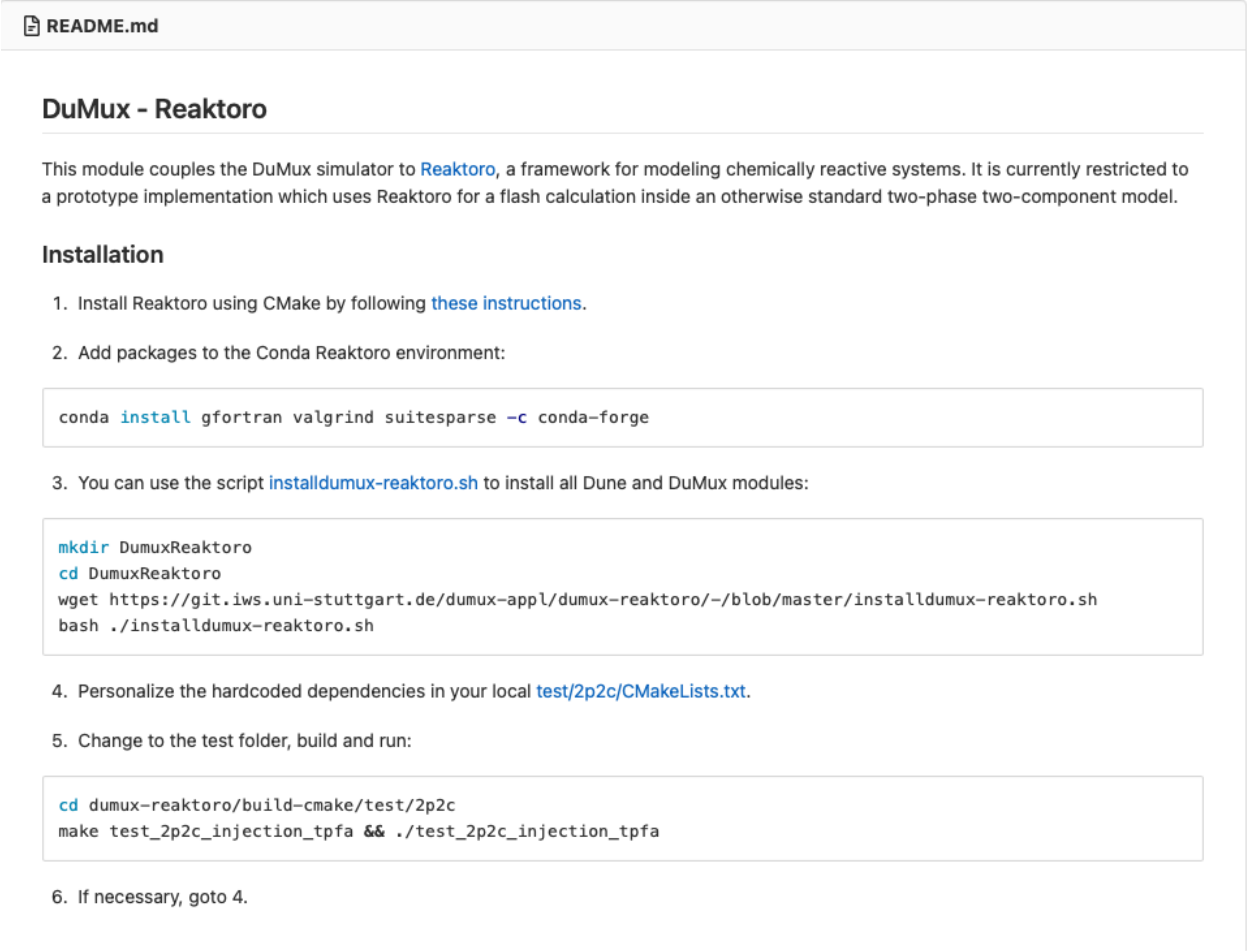


```python
import reaktoro as rkt

db = rkt.SupcrtDatabase("supcrtbl")

liquids = rkt.AqueousPhase("H2O(aq) CO2(aq)")
gases = rkt.GaseousPhase("CO2(g) H2O(g)")

system = rkt.ChemicalSystem(db, liquids, gases)

specs = rkt.EquilibriumSpecs(system)
specs.temperature()
specs.volume()

solver = rkt.EquilibriumSolver(specs)

state = rkt.ChemicalState(system)
state.temperature(293.15, "kelvin")
state.scaleVolume(1.0, "m3")
state.pressure(100.0, "bar")
state.set("H2O(aq)", 500.0, "kg")
state.set("CO2(aq)", 500.0, "kg")

props = rkt.ChemicalProps(state)

conditions = rkt.EquilibriumConditions(specs)
conditions.temperature(293.15, "kelvin")
conditions.volume(1.0, "m3")
conditions.setLowerBoundPressure(1.0, "bar")
conditions.setUpperBoundPressure(1000.0, "bar")

print("=== INITIAL STATE ===")
print(state)

result = solver.solve(state, conditions)

props.update(state)
print("=== FINAL STATE ===")
print(state)

print("Successful computation!" if result.optima.succeeded else "Computation has failed!")

print("S_l=",props.phaseProps("AqueousPhase").volume()/props.volume())
print("S_g=",props.phaseProps("GaseousPhase").volume()/props.volume())
print("x^H2O_l",props.phaseProps("AqueousPhase").speciesMoleFractions()[0])
print("x^CO2_l",props.phaseProps("AqueousPhase").speciesMoleFractions()[1])
print("x^H2O_g",props.phaseProps("GaseousPhase").speciesMoleFractions()[0])
print("x^CO2_g",props.phaseProps("GaseousPhase").speciesMoleFractions()[1])
```



```
(reaktoro) dmar@BGO-1486 examples % python3 ccs.py
=== INITIAL STATE ===
+-----------------+------------+------+
| Property        |      Value | Unit |
+-----------------+------------+------+
| Temperature     |   293.1500 |    K |
| Pressure        |   100.0000 |  bar |
| Charge:         | 0.0000e+00 |  mol |
| Element Amount: |            |      |
| :: H            | 5.5549e+04 |  mol |
| :: C            | 1.1382e+04 |  mol |
| :: O            | 5.0538e+04 |  mol |
| Species Amount: |            |      |
| :: H2O(aq)      | 2.7754e+04 |  mol |
| :: CO2(aq)      | 1.1361e+04 |  mol |
| :: CO2(g)       | 2.0493e+01 |  mol |
| :: H2O(g)       | 2.0493e+01 |  mol |
+-----------------+------------+------+
=== FINAL STATE ===
+-----------------+------------+------+
| Property        |      Value | Unit |
+-----------------+------------+------+
| Temperature     |   293.1500 |    K |
| Pressure        |   447.8288 |  bar |
| Charge:         | 0.0000e+00 |  mol |
| Element Amount: |            |      |
| :: H            | 5.5549e+04 |  mol |
| :: C            | 1.1382e+04 |  mol |
| :: O            | 5.0538e+04 |  mol |
| Species Amount: |            |      |
| :: H2O(aq)      | 2.7774e+04 |  mol |
| :: CO2(aq)      | 4.8977e+03 |  mol |
| :: CO2(g)       | 6.4839e+03 |  mol |
| :: H2O(g)       | 3.8909e-01 |  mol |
+-----------------+------------+------+
Successful computation!
S_l= 0.647083
S_g= 0.352917
x^H2O_l 0.850094
x^CO2_l 0.149906
x^H2O_g 0.99994
x^CO2_g 6.0006e-05
(reaktoro) dmar@BGO-1486 examples %
```

Coupling of Reaktoro and DuMux



README.md

DuMux - Reaktoro

This module couples the DuMux simulator to Reaktoro, a framework for modeling chemically reactive systems. It is currently restricted to a prototype implementation which uses Reaktoro for a flash calculation inside an otherwise standard two-phase two-component model.

Installation

1. Install Reaktoro using CMake by following these instructions.

2. Add packages to the Conda Reaktoro environment:

```
conda install gfortran valgrind suitesparse -c conda-forge
```

3. You can use the script installdumux-reaktoro.sh to install all Dune and DuMux modules:

```
mkdir DumuxReaktoro
cd DumuxReaktoro
wget https://git.iws.uni-stuttgart.de/dumux-appl/dumux-reaktoro/-/blob/master/installdumux-reaktoro.sh
bash ./installdumux-reaktoro.sh
```

4. Personalize the hardcoded dependencies in your local test/2p2c/CMakeLists.txt.

5. Change to the test folder, build and run:

```
cd dumux-reaktoro/build-cmake/test/2p2c
make test_2p2c_injection_tpfa && ./test_2p2c_injection_tpfa
```

6. If necessary, goto 4.

https://git.iws.uni-stuttgart.de/dumux-appl/dumux-reaktoro

Two-phase Two-component Model

- Two mass-balance equations for the components $\kappa \in \{w, a\}$,

$$\sum_{\alpha\in\{l,g\}} \phi \frac{\partial\left(\varrho_\alpha X_\alpha^\kappa S_\alpha\right)}{\partial t} + \boldsymbol{\nabla} \cdot \mathbf{F}^\kappa - \sum_{\alpha\in\{l,g\}} q_\alpha^\kappa = 0,$$

- Mass fluxes of the components are given by

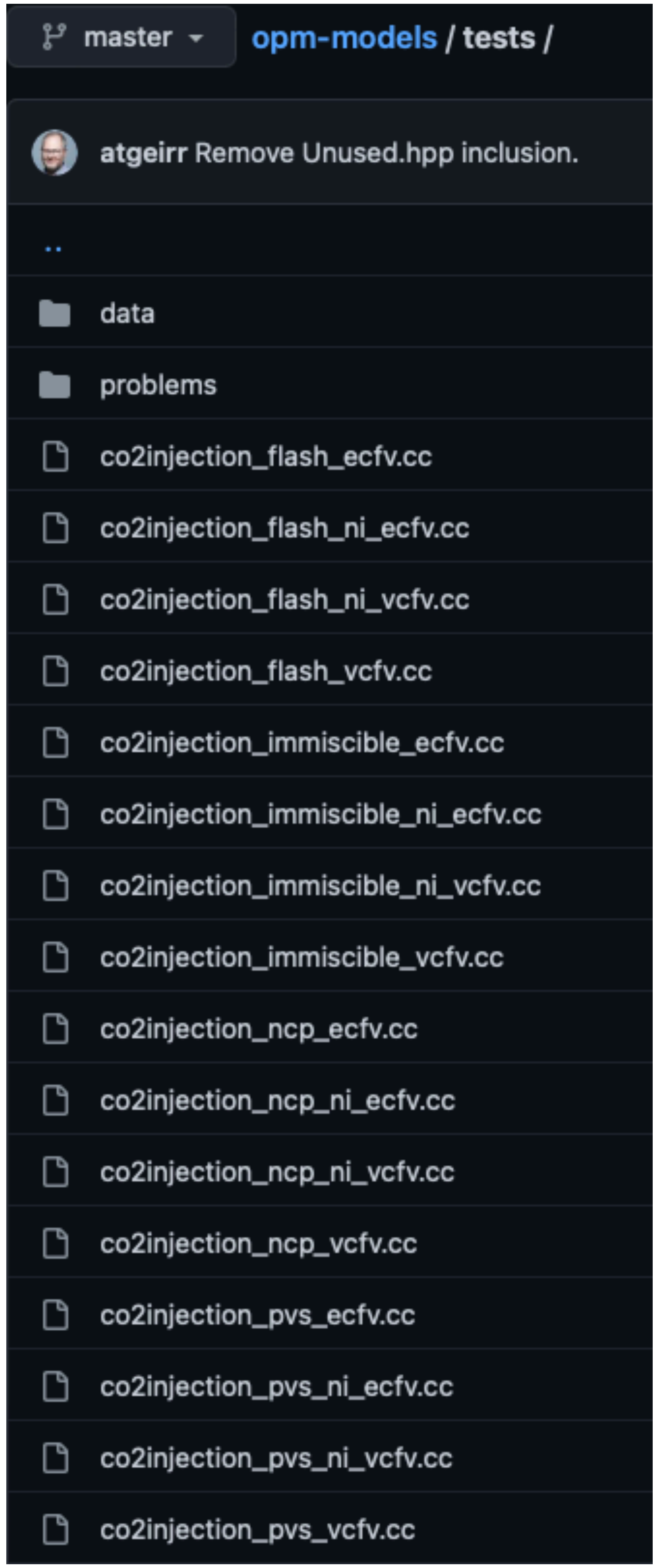
$$\mathbf{F}^\kappa = \sum_{\alpha\in\{l,g\}} \left(\varrho_\alpha \mathbf{v}_\alpha X_\alpha^\kappa - D_{\alpha,\mathrm{pm}}^\kappa \, \varrho_\alpha \boldsymbol{\nabla} X_\alpha^\kappa\right) .$$

- Darcy law for the description of the phase velocities $\mathbf{v}_\alpha$, namely,

$$\mathbf{v}_\alpha = -\frac{k_{r\alpha}}{\mu_\alpha}\mathbf{K}\left(\boldsymbol{\nabla} p_\alpha - \varrho_\alpha \mathbf{g}\right), \qquad \alpha \in \{l, g\}.$$

OPM

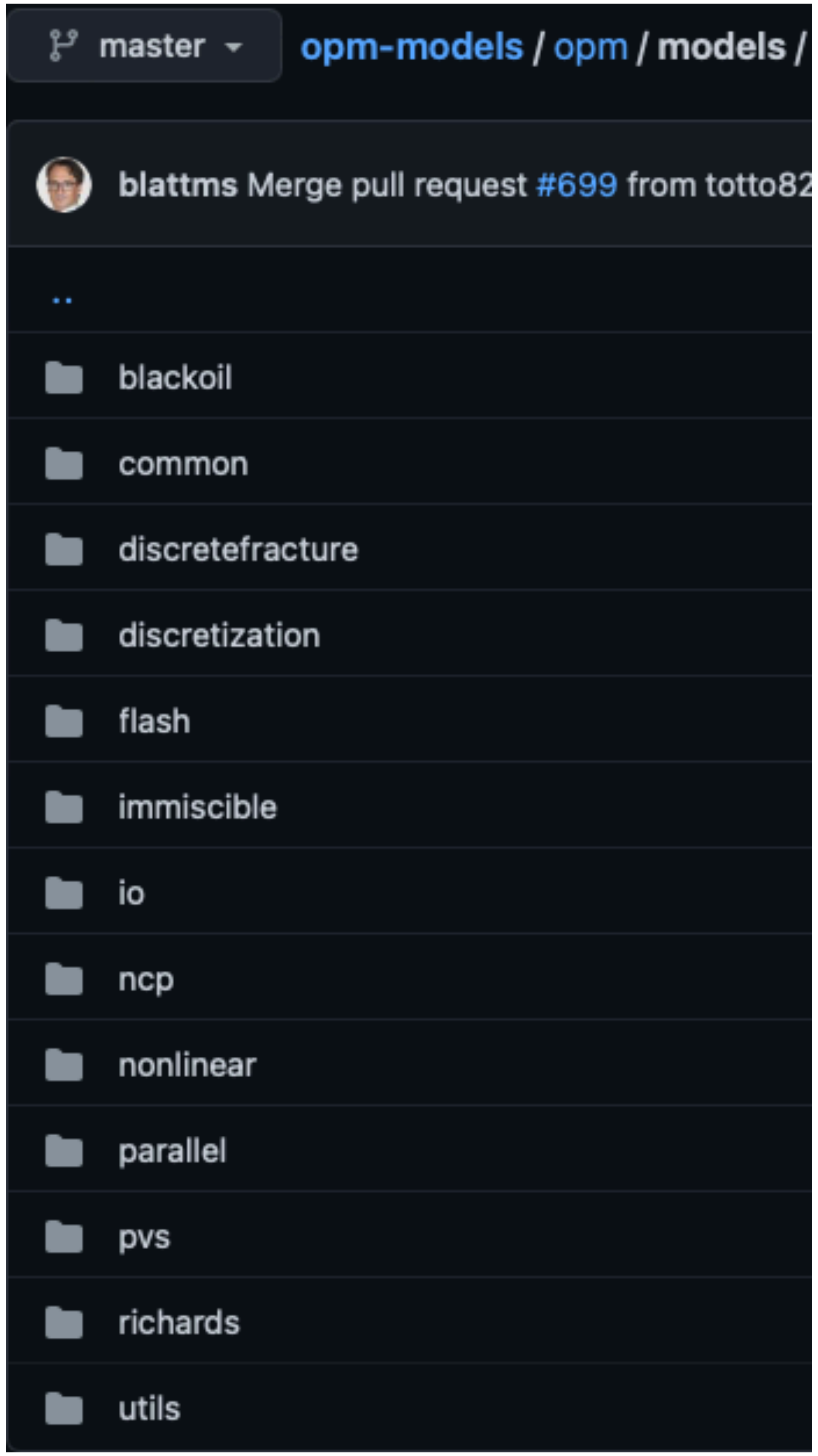
NORCE



master opm-models / tests /

atgeirr Remove Unused.hpp inclusion.

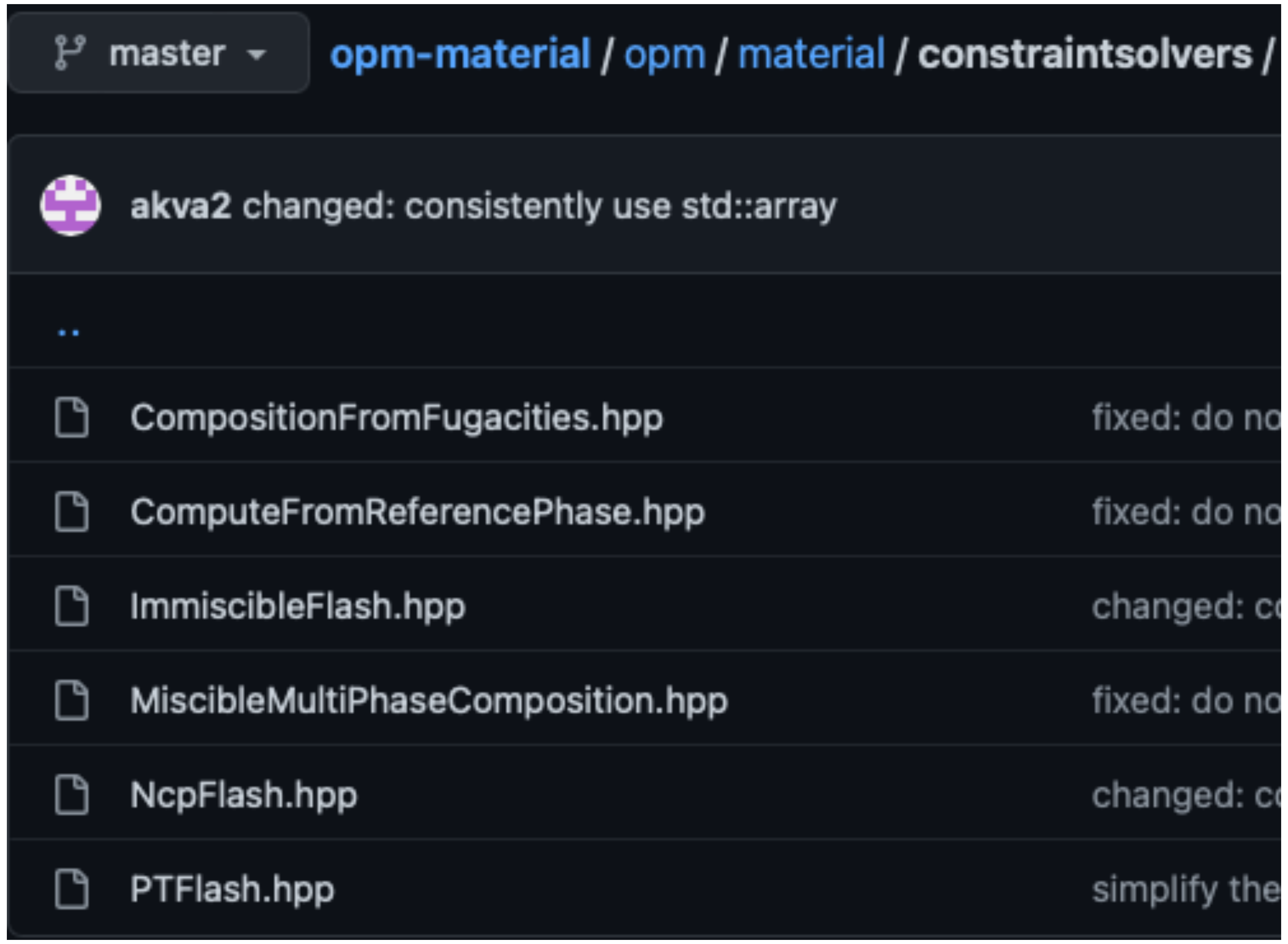
..
data
problems
co2injection_flash_ecfv.cc
co2injection_flash_ni_ecfv.cc
co2injection_flash_ni_vcfv.cc
co2injection_flash_vcfv.cc
co2injection_immiscible_ecfv.cc
co2injection_immiscible_ni_ecfv.cc
co2injection_immiscible_ni_vcfv.cc
co2injection_immiscible_vcfv.cc
co2injection_ncp_ecfv.cc
co2injection_ncp_ni_ecfv.cc
co2injection_ncp_ni_vcfv.cc
co2injection_ncp_vcfv.cc
co2injection_pvs_ecfv.cc
co2injection_pvs_ni_ecfv.cc
co2injection_pvs_ni_vcfv.cc
co2injection_pvs_vcfv.cc



master opm-models / opm / models /

blattms Merge pull request #699 from totto82

..
blackoil
common
discretefracture
discretization
flash
immiscible
io
ncp
nonlinear
parallel
pvs
richards
utils

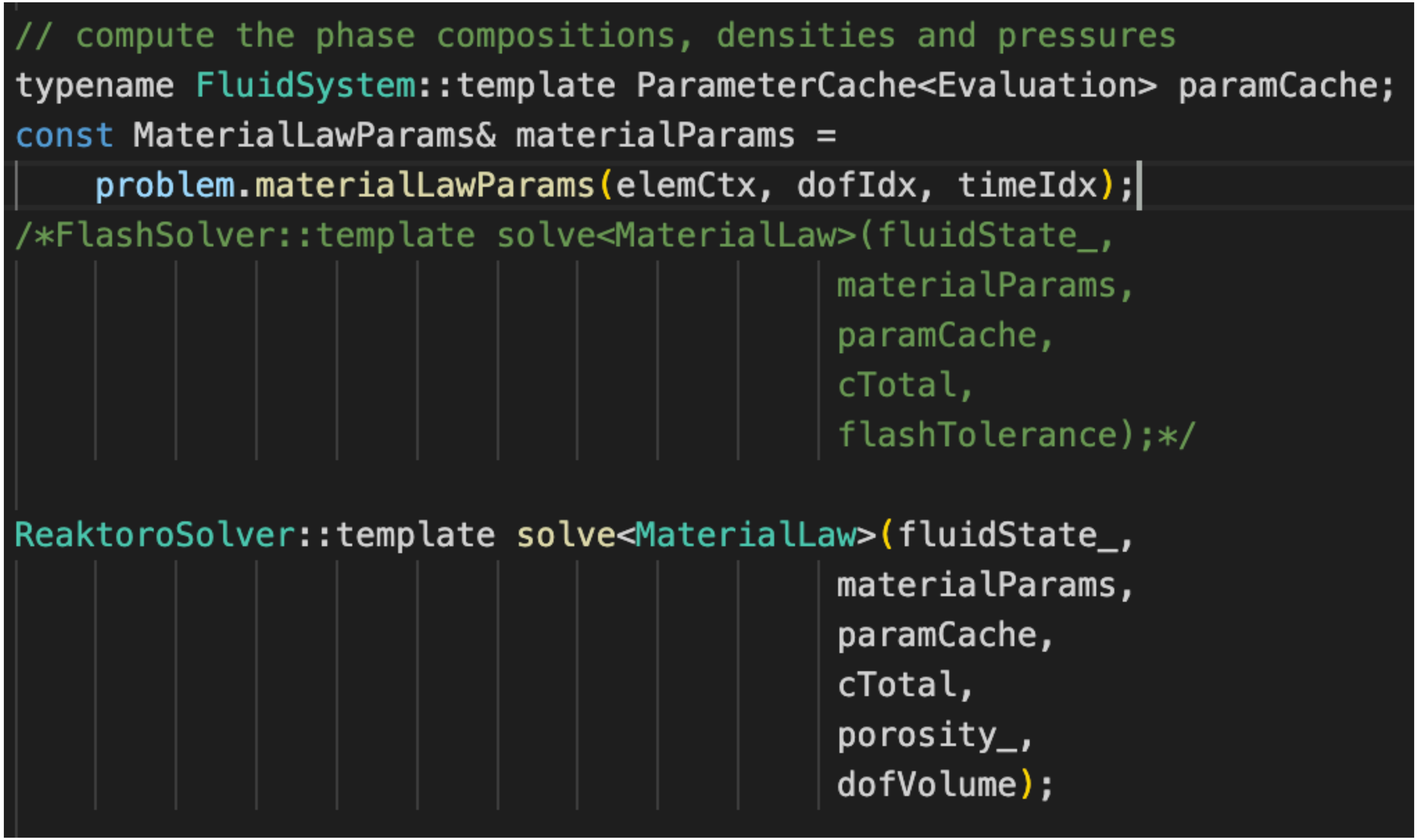


master opm-material / opm / material / constraintsolvers /

akva2 changed: consistently use std::array

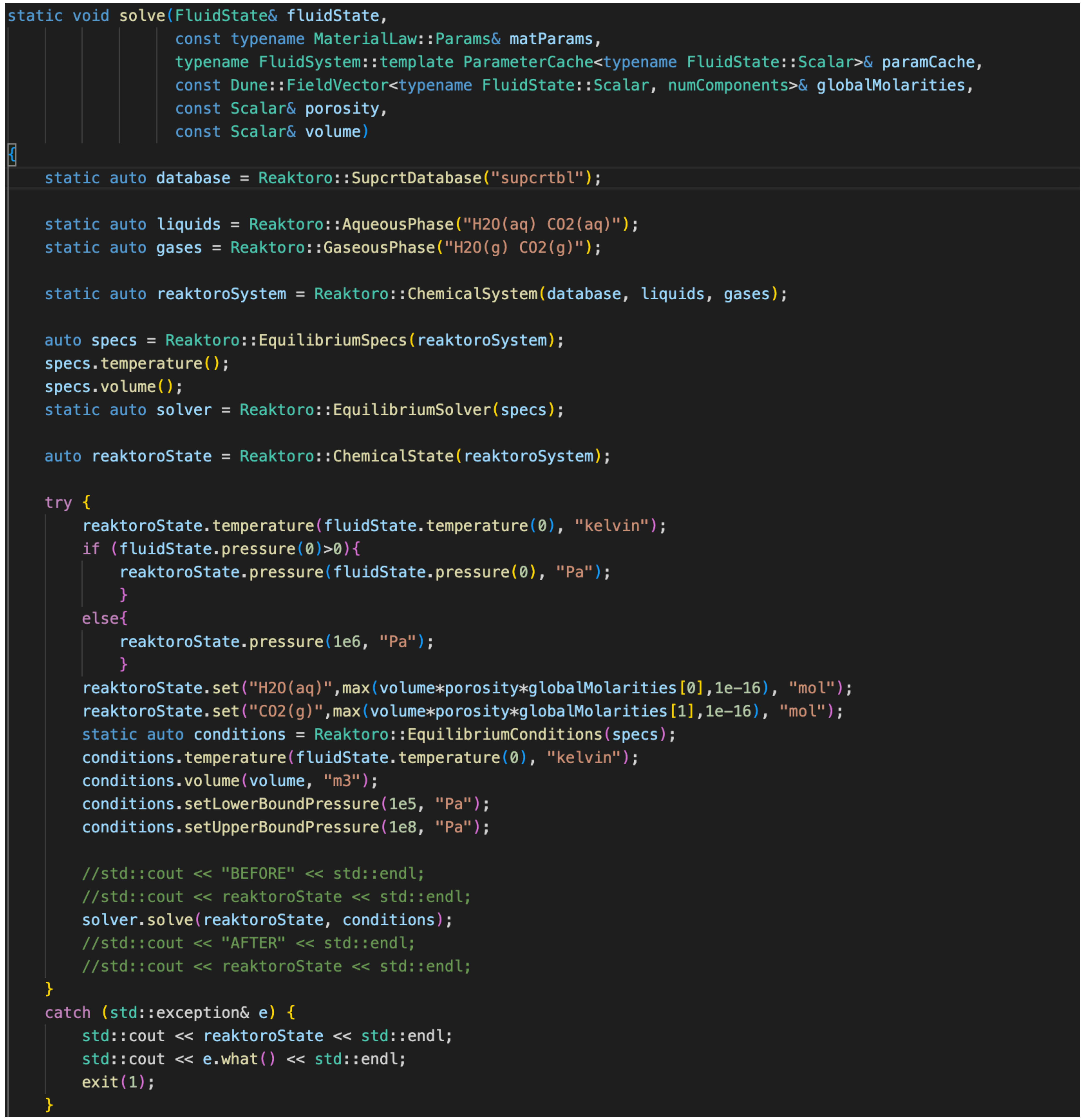
..
CompositionFromFugacities.hpp fixed: do no
ComputeFromReferencePhase.hpp fixed: do no
ImmiscibleFlash.hpp changed: c
MiscibleMultiPhaseComposition.hpp fixed: do no
NcpFlash.hpp changed: c
PTFlash.hpp simplify the

opm-models/opm/models/reaktoro/reaktorointensivequantities.hpp

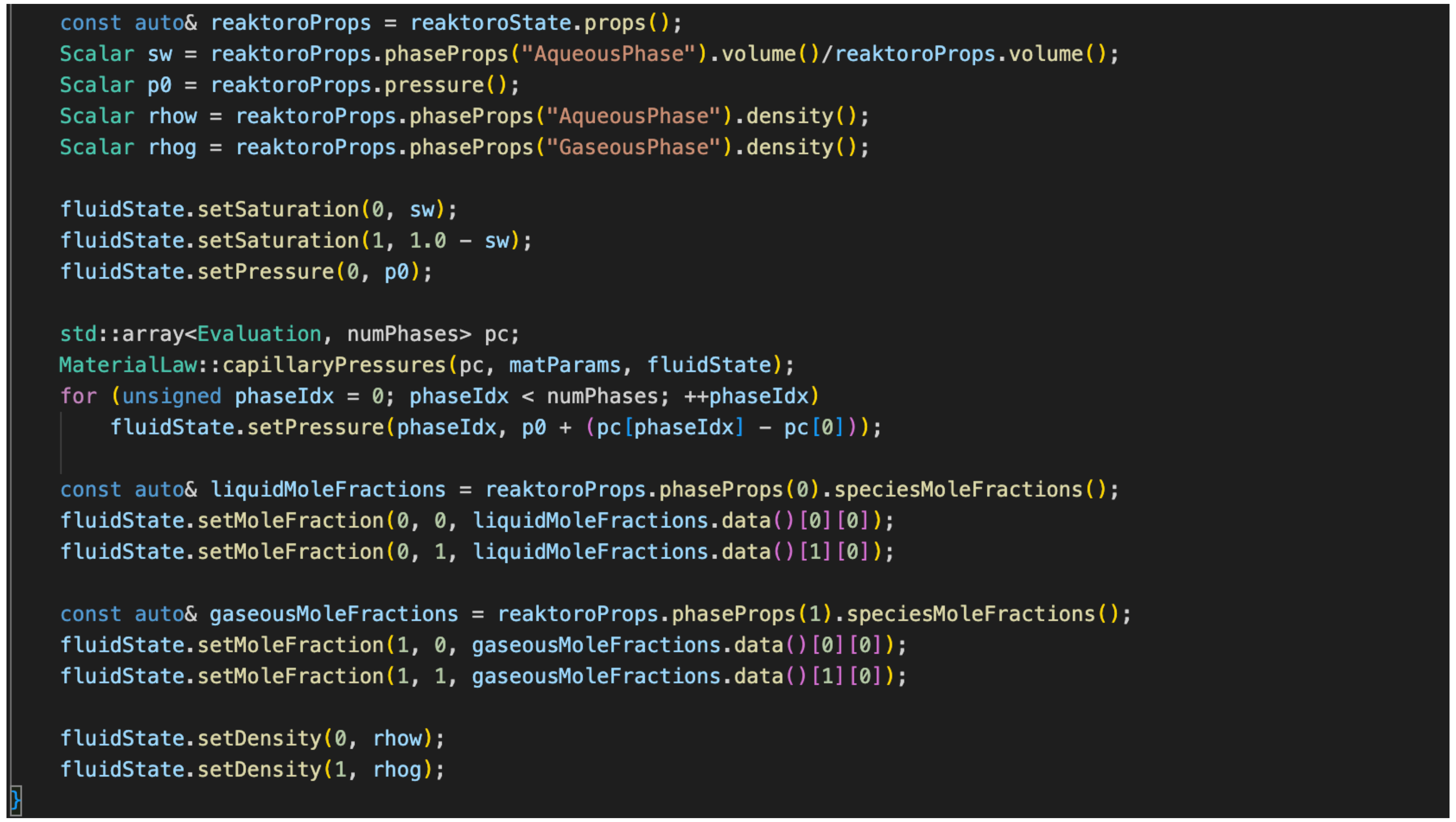


```
// compute the phase compositions, densities and pressures
typename FluidSystem::template ParameterCache<Evaluation> paramCache;
const MaterialLawParams& materialParams =
    problem.materialLawParams(elemCtx, dofIdx, timeIdx);
/*FlashSolver::template solve<MaterialLaw>(fluidState_,
                                         materialParams,
                                         paramCache,
                                         cTotal,
                                         flashTolerance);*/

ReaktoroSolver::template solve<MaterialLaw>(fluidState_,
                                         materialParams,
                                         paramCache,
                                         cTotal,
                                         porosity_,
                                         dofVolume);
```

opm-material/opm/material/constrainsolvers/ OpmReaktoro.hpp



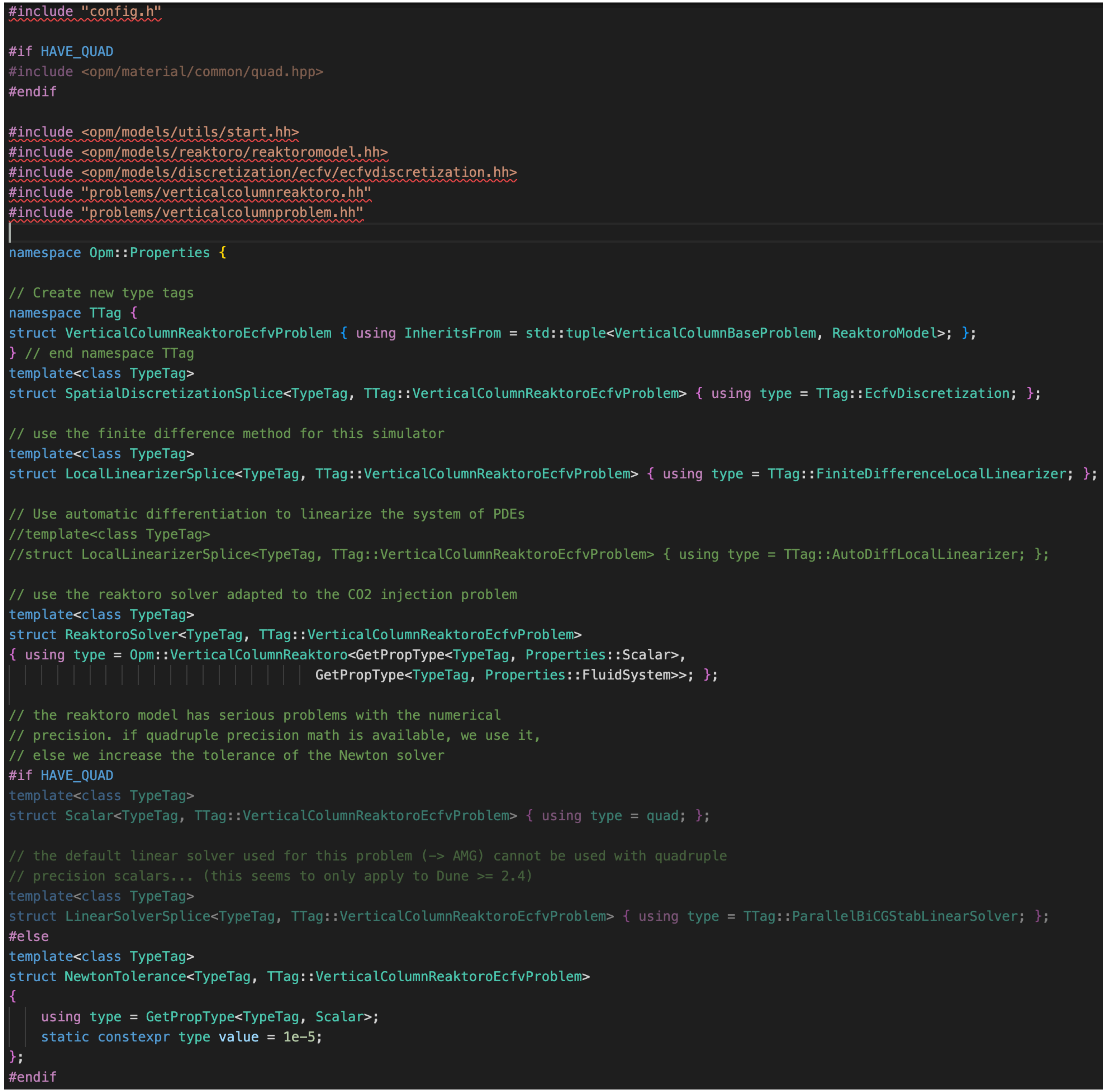
```cpp
static void solve(FluidState& fluidState,
                  const typename MaterialLaw::Params& matParams,
                  typename FluidSystem::template ParameterCache<typename FluidState::Scalar>& paramCache,
                  const Dune::FieldVector<typename FluidState::Scalar, numComponents>& globalMolarities,
                  const Scalar& porosity,
                  const Scalar& volume)
{
    static auto database = Reaktoro::SupcrtDatabase("supcrtbl");

    static auto liquids = Reaktoro::AqueousPhase("H2O(aq) CO2(aq)");
    static auto gases = Reaktoro::GaseousPhase("H2O(g) CO2(g)");

    static auto reaktoroSystem = Reaktoro::ChemicalSystem(database, liquids, gases);

    auto specs = Reaktoro::EquilibriumSpecs(reaktoroSystem);
    specs.temperature();
    specs.volume();
    static auto solver = Reaktoro::EquilibriumSolver(specs);

    auto reaktoroState = Reaktoro::ChemicalState(reaktoroSystem);

    try {
        reaktoroState.temperature(fluidState.temperature(0), "kelvin");
        if (fluidState.pressure(0)>0){
            reaktoroState.pressure(fluidState.pressure(0), "Pa");
            }
        else{
            reaktoroState.pressure(1e6, "Pa");
            }
        reaktoroState.set("H2O(aq)",max(volume*porosity*globalMolarities[0],1e-16), "mol");
        reaktoroState.set("CO2(g)",max(volume*porosity*globalMolarities[1],1e-16), "mol");
        static auto conditions = Reaktoro::EquilibriumConditions(specs);
        conditions.temperature(fluidState.temperature(0), "kelvin");
        conditions.volume(volume, "m3");
        conditions.setLowerBoundPressure(1e5, "Pa");
        conditions.setUpperBoundPressure(1e8, "Pa");

        //std::cout << "BEFORE" << std::endl;
        //std::cout << reaktoroState << std::endl;
        solver.solve(reaktoroState, conditions);
        //std::cout << "AFTER" << std::endl;
        //std::cout << reaktoroState << std::endl;
    }
    catch (std::exception& e) {
        std::cout << reaktoroState << std::endl;
        std::cout << e.what() << std::endl;
        exit(1);
    }
```



```cpp
    const auto& reaktoroProps = reaktoroState.props();
    Scalar sw = reaktoroProps.phaseProps("AqueousPhase").volume()/reaktoroProps.volume();
    Scalar p0 = reaktoroProps.pressure();
    Scalar rhow = reaktoroProps.phaseProps("AqueousPhase").density();
    Scalar rhog = reaktoroProps.phaseProps("GaseousPhase").density();

    fluidState.setSaturation(0, sw);
    fluidState.setSaturation(1, 1.0 - sw);
    fluidState.setPressure(0, p0);

    std::array<Evaluation, numPhases> pc;
    MaterialLaw::capillaryPressures(pc, matParams, fluidState);
    for (unsigned phaseIdx = 0; phaseIdx < numPhases; ++phaseIdx)
        fluidState.setPressure(phaseIdx, p0 + (pc[phaseIdx] - pc[0]));

    const auto& liquidMoleFractions = reaktoroProps.phaseProps(0).speciesMoleFractions();
    fluidState.setMoleFraction(0, 0, liquidMoleFractions.data()[0][0]);
    fluidState.setMoleFraction(0, 1, liquidMoleFractions.data()[1][0]);

    const auto& gaseousMoleFractions = reaktoroProps.phaseProps(1).speciesMoleFractions();
    fluidState.setMoleFraction(1, 0, gaseousMoleFractions.data()[0][0]);
    fluidState.setMoleFraction(1, 1, gaseousMoleFractions.data()[1][0]);

    fluidState.setDensity(0, rhow);
    fluidState.setDensity(1, rhog);
}
```

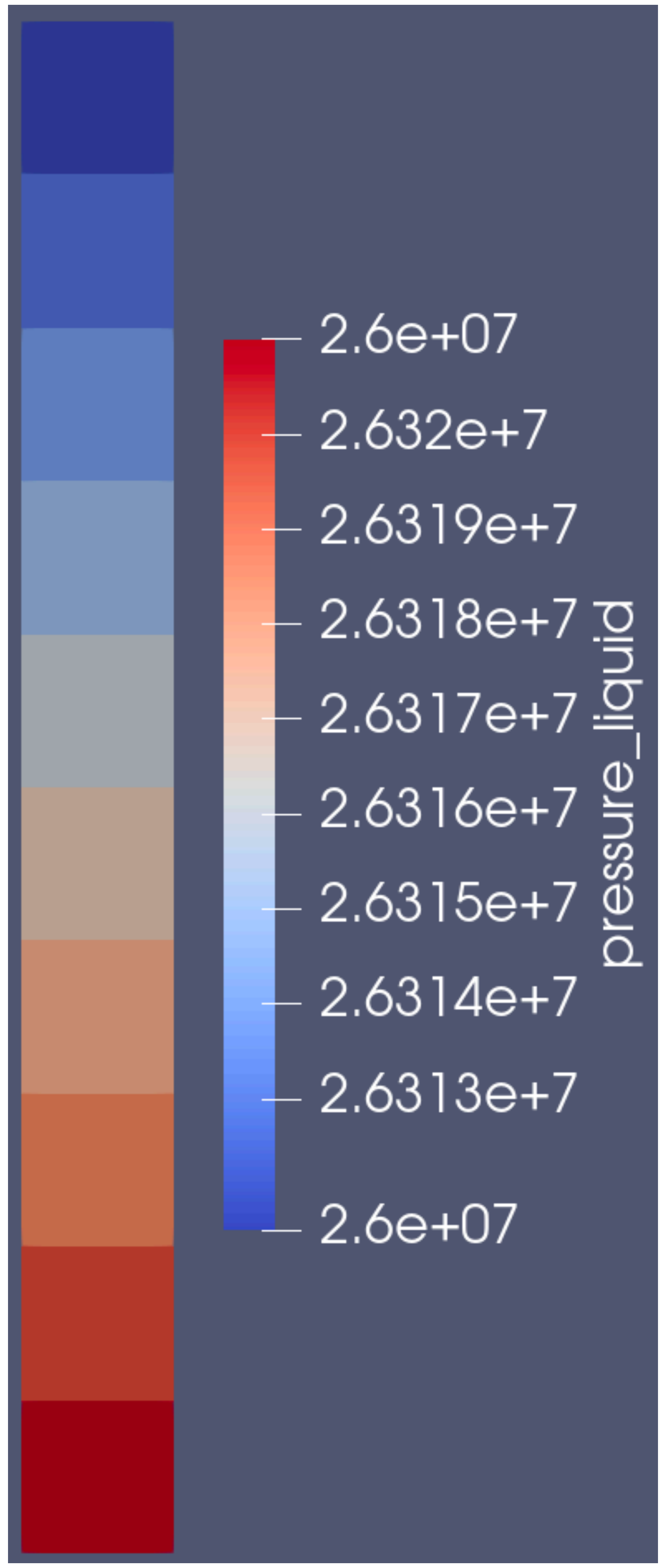
opm-models/tests/vertical_column_reaktoro_ecfv.cc

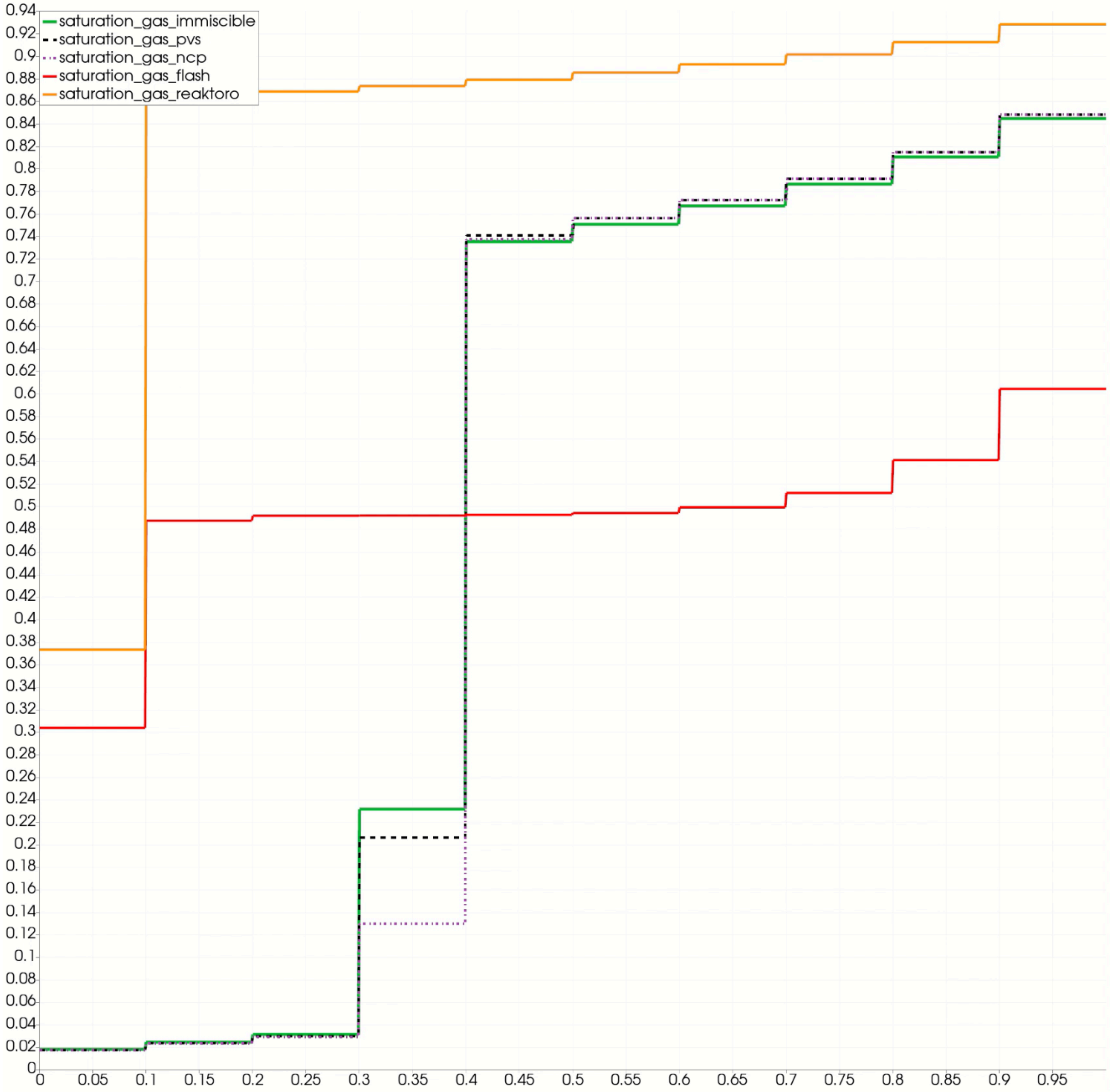
NORCE

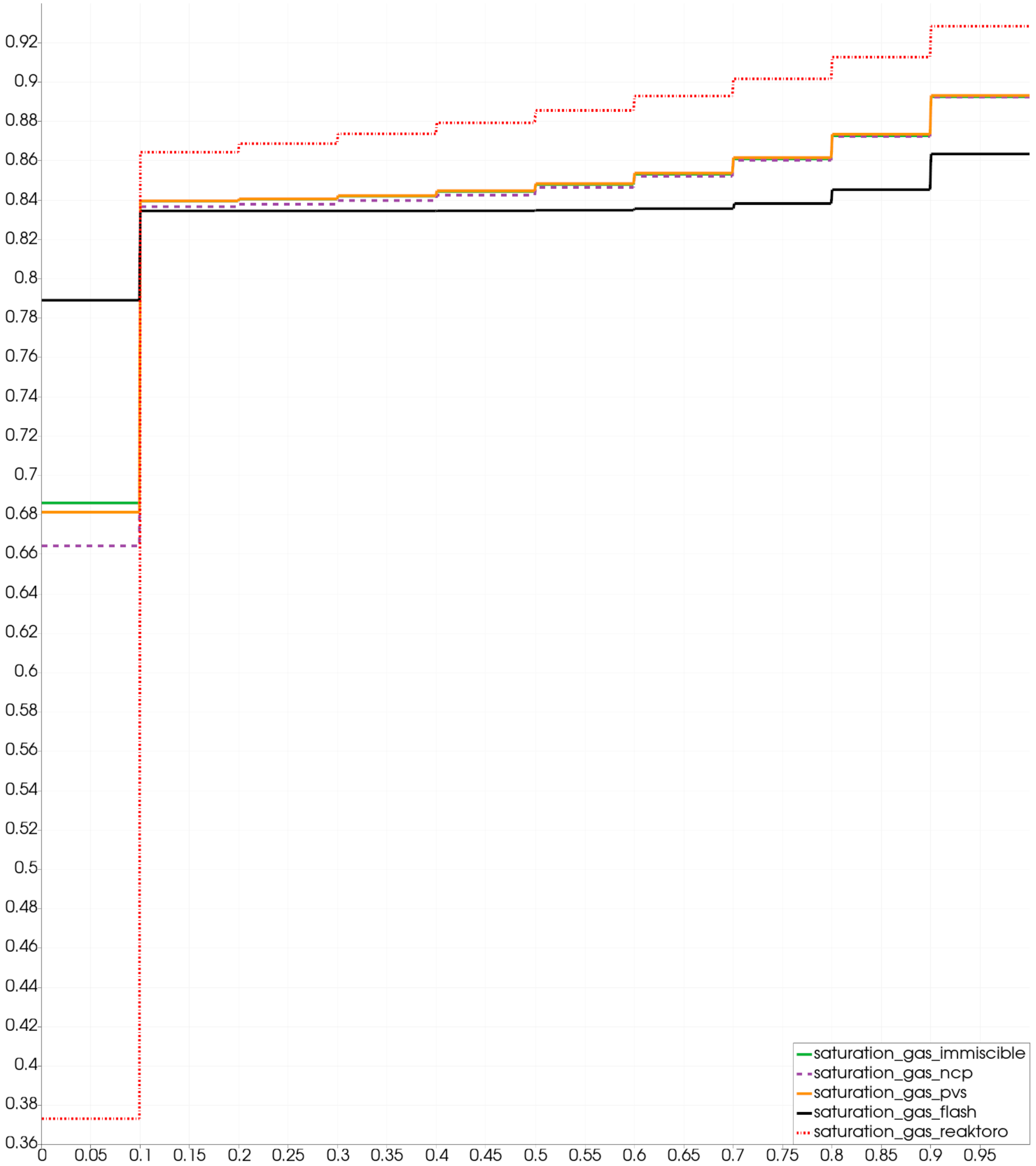


```
#include "config.h"

#if HAVE_QUAD
#include <opm/material/common/quad.hpp>
#endif

#include <opm/models/utils/start.hh>
#include <opm/models/reaktoro/reaktoromodel.hh>
#include <opm/models/discretization/ecfv/ecfvdiscretization.hh>
#include "problems/verticalcolumnreaktoro.hh"
#include "problems/verticalcolumnproblem.hh"

namespace Opm::Properties {

// Create new type tags
namespace TTag {
struct VerticalColumnReaktoroEcfvProblem { using InheritsFrom = std::tuple<VerticalColumnBaseProblem, ReaktoroModel>; };
} // end namespace TTag
template<class TypeTag>
struct SpatialDiscretizationSplice<TypeTag, TTag::VerticalColumnReaktoroEcfvProblem> { using type = TTag::EcfvDiscretization; };

// use the finite difference method for this simulator
template<class TypeTag>
struct LocalLinearizerSplice<TypeTag, TTag::VerticalColumnReaktoroEcfvProblem> { using type = TTag::FiniteDifferenceLocalLinearizer; };

// Use automatic differentiation to linearize the system of PDEs
//template<class TypeTag>
//struct LocalLinearizerSplice<TypeTag, TTag::VerticalColumnReaktoroEcfvProblem> { using type = TTag::AutoDiffLocalLinearizer; };

// use the reaktoro solver adapted to the CO2 injection problem
template<class TypeTag>
struct ReaktoroSolver<TypeTag, TTag::VerticalColumnReaktoroEcfvProblem>
{ using type = Opm::VerticalColumnReaktoro<GetPropType<TypeTag, Properties::Scalar>,
                                      GetPropType<TypeTag, Properties::FluidSystem>>; };

// the reaktoro model has serious problems with the numerical
// precision. if quadruple precision math is available, we use it,
// else we increase the tolerance of the Newton solver
#if HAVE_QUAD
template<class TypeTag>
struct Scalar<TypeTag, TTag::VerticalColumnReaktoroEcfvProblem> { using type = quad; };

// the default linear solver used for this problem (-> AMG) cannot be used with quadruple
// precision scalars... (this seems to only apply to Dune >= 2.4)
template<class TypeTag>
struct LinearSolverSplice<TypeTag, TTag::VerticalColumnReaktoroEcfvProblem> { using type = TTag::ParallelBiCGStabLinearSolver; };
#else
template<class TypeTag>
struct NewtonTolerance<TypeTag, TTag::VerticalColumnReaktoroEcfvProblem>
{
    using type = GetPropType<TypeTag, Scalar>;
    static constexpr type value = 1e-5;
};
#endif
```

Numerical test







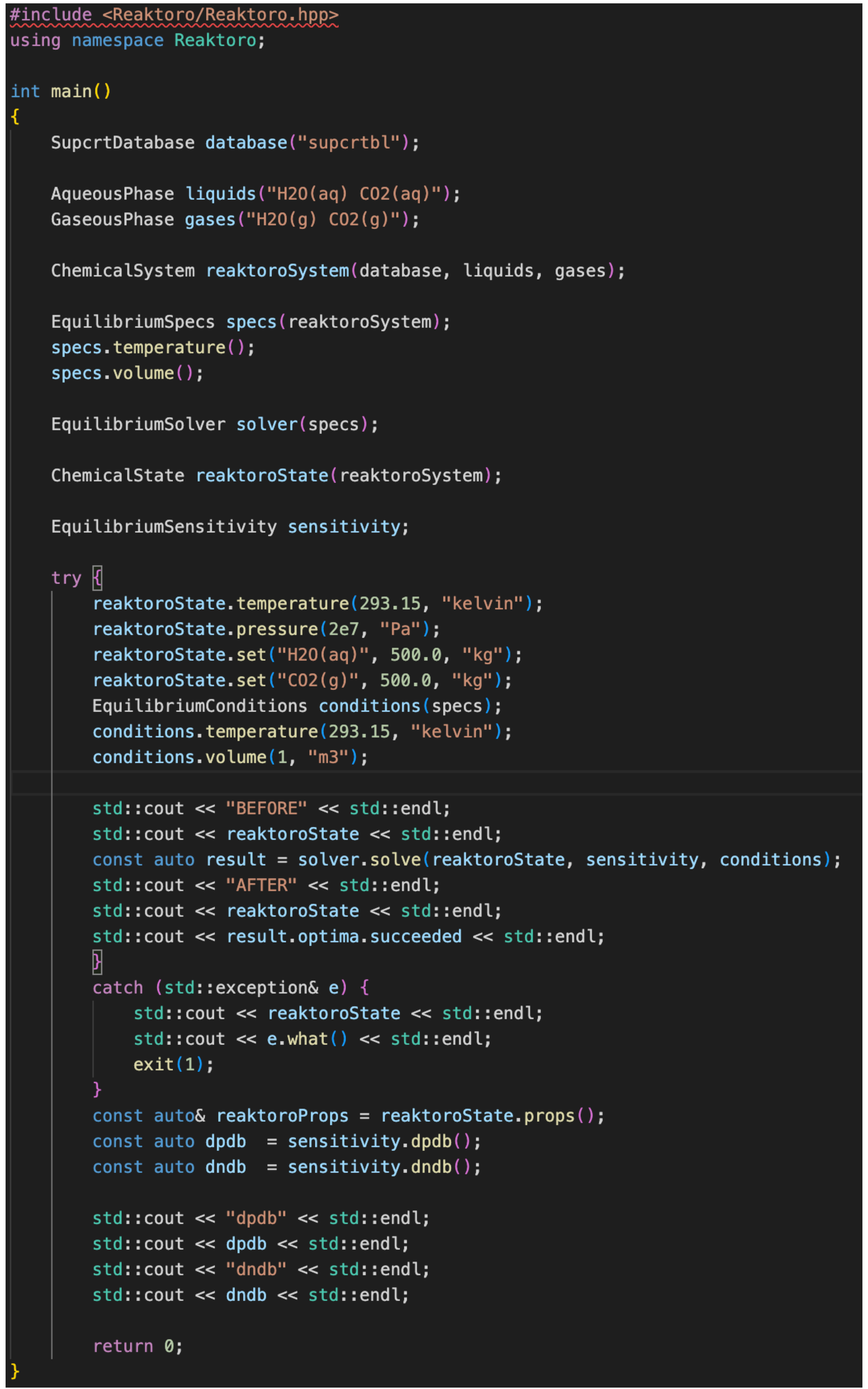
Current work

https://git.iws.uni-stuttgart.de/dumux-appl/dumux-reaktoro

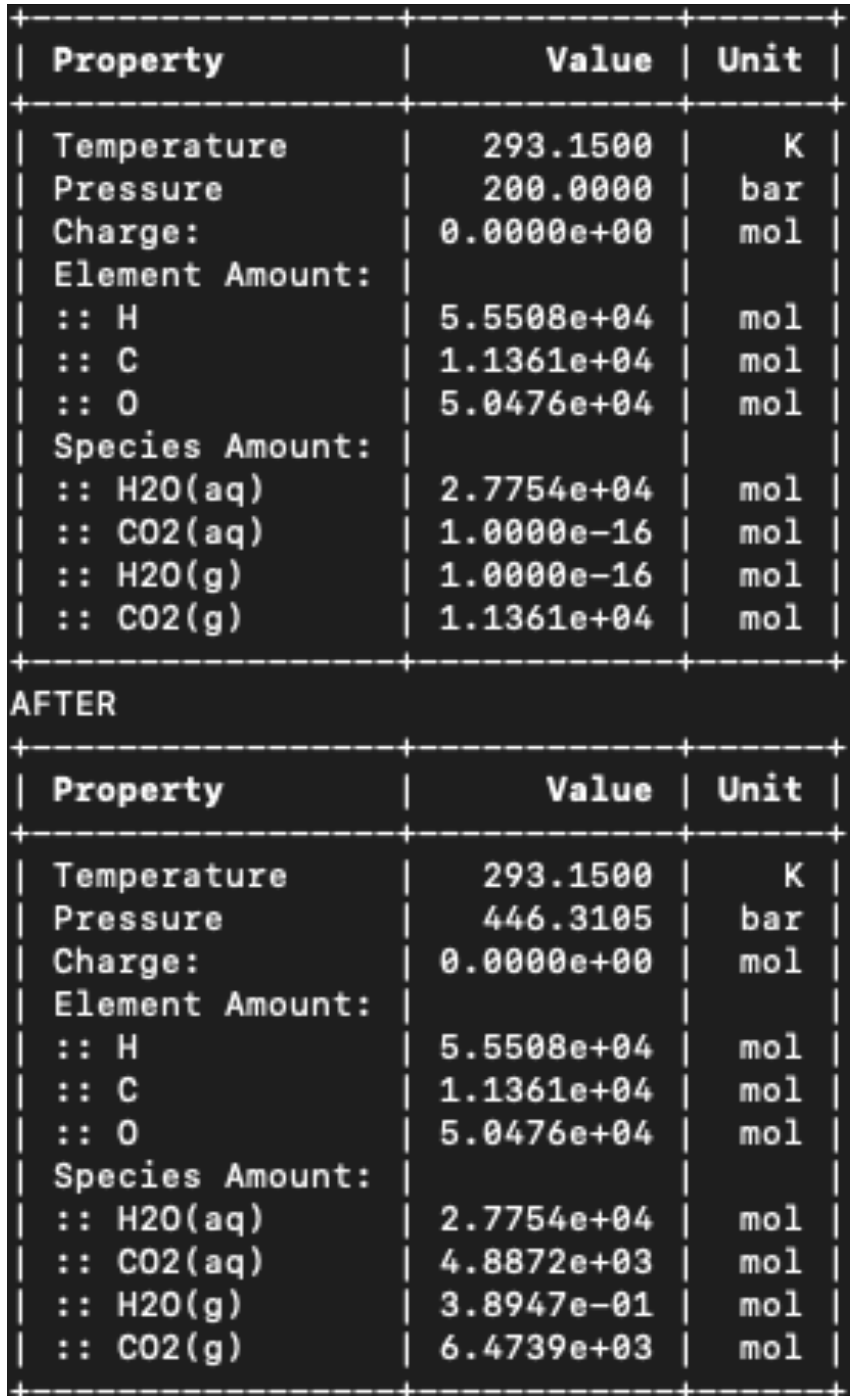
- Write a `FindReaktoro.cmake`.
- Derive a DuMu^x-conforming `FluidState` from `Reaktoro::ChemicalState`.
- Develop a DuMu^x `FluidSystem` using a `Reaktoro::equilibriumSolver`.
- Write a proper module for chemical reactions and reactive transport.

- To use AD instead of FD to linearise the system of PDEs.

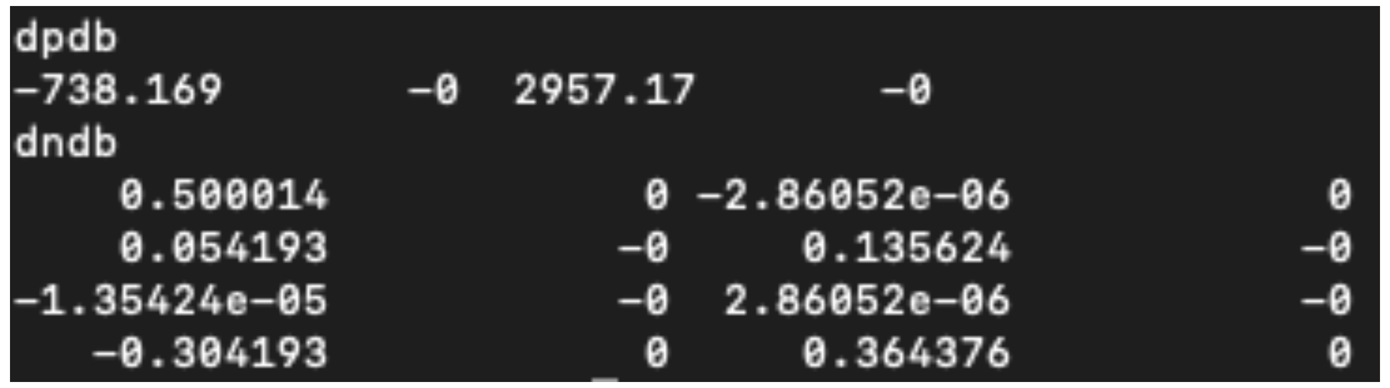
Current work



```cpp
#include <Reaktoro/Reaktoro.hpp>
using namespace Reaktoro;

int main()
{
    SupcrtDatabase database("supcrtbl");

    AqueousPhase liquids("H2O(aq) CO2(aq)");
    GaseousPhase gases("H2O(g) CO2(g)");

    ChemicalSystem reaktoroSystem(database, liquids, gases);

    EquilibriumSpecs specs(reaktoroSystem);
    specs.temperature();
    specs.volume();

    EquilibriumSolver solver(specs);

    ChemicalState reaktoroState(reaktoroSystem);

    EquilibriumSensitivity sensitivity;

    try {
        reaktoroState.temperature(293.15, "kelvin");
        reaktoroState.pressure(2e7, "Pa");
        reaktoroState.set("H2O(aq)", 500.0, "kg");
        reaktoroState.set("CO2(g)", 500.0, "kg");
        EquilibriumConditions conditions(specs);
        conditions.temperature(293.15, "kelvin");
        conditions.volume(1, "m3");

        std::cout << "BEFORE" << std::endl;
        std::cout << reaktoroState << std::endl;
        const auto result = solver.solve(reaktoroState, sensitivity, conditions);
        std::cout << "AFTER" << std::endl;
        std::cout << reaktoroState << std::endl;
        std::cout << result.optima.succeeded << std::endl;
        }
        catch (std::exception& e) {
            std::cout << reaktoroState << std::endl;
            std::cout << e.what() << std::endl;
            exit(1);
        }
        const auto& reaktoroProps = reaktoroState.props();
        const auto dpdb  = sensitivity.dpdb();
        const auto dndb  = sensitivity.dndb();

        std::cout << "dpdb" << std::endl;
        std::cout << dpdb << std::endl;
        std::cout << "dndb" << std::endl;
        std::cout << dndb << std::endl;

        return 0;
}
```



```
+-----------------+------------+------+
| Property        |      Value | Unit |
+-----------------+------------+------+
| Temperature     |   293.1500 |    K |
| Pressure        |   200.0000 |  bar |
| Charge:         | 0.0000e+00 |  mol |
| Element Amount: |            |      |
| :: H            | 5.5508e+04 |  mol |
| :: C            | 1.1361e+04 |  mol |
| :: O            | 5.0476e+04 |  mol |
| Species Amount: |            |      |
| :: H2O(aq)      | 2.7754e+04 |  mol |
| :: CO2(aq)      | 1.0000e-16 |  mol |
| :: H2O(g)       | 1.0000e-16 |  mol |
| :: CO2(g)       | 1.1361e+04 |  mol |
+-----------------+------------+------+
AFTER
+-----------------+------------+------+
| Property        |      Value | Unit |
+-----------------+------------+------+
| Temperature     |   293.1500 |    K |
| Pressure        |   446.3105 |  bar |
| Charge:         | 0.0000e+00 |  mol |
| Element Amount: |            |      |
| :: H            | 5.5508e+04 |  mol |
| :: C            | 1.1361e+04 |  mol |
| :: O            | 5.0476e+04 |  mol |
| Species Amount: |            |      |
| :: H2O(aq)      | 2.7754e+04 |  mol |
| :: CO2(aq)      | 4.8872e+03 |  mol |
| :: H2O(g)       | 3.8947e-01 |  mol |
| :: CO2(g)       | 6.4739e+03 |  mol |
+-----------------+------------+------+
```



```
dpdb
-738.169        -0  2957.17        -0
dndb
    0.500014               0 -2.86052e-06             0
    0.054193              -0     0.135624            -0
-1.35424e-05              -0  2.86052e-06            -0
   -0.304193               0     0.364376             0
```