Finite volume schemes and coupled problems in DuMu\textsuperscript{x}

Timo Koch

https://dumux.org/
https://timokoch.github.io/
DuMu$^x$ is a simulation framework with a focus on finite volume discretization methods, model coupling for multi-physics applications, and flow and transport applications in porous media.

DuMu$^x$ is based on the DUNE framework from which it uses the versatile grid interface, vector and matrix types, geometry and local basis functions, and linear solvers. DuMu$^x$ then provides

- **Finite volume discretizations** (Tpfa, Mpfa, Staggered) and control-volume finite element discretization schemes
- A (thread-parallel) system matrix assembler (coloring) and approximation of the Jacobian matrix by numeric differentiation
- A customizable Newton method implementation including line search and various stopping criteria
- Many pre-implemented models (Darcy-scale porous media flow, Navier-Stokes, Geomechanics, Pore network models, Shallow water equations) and constitutive models
- A multi-domain framework for model coupling suited to couple subproblems with different discretizations/domains/physics/dimensions/... and create monolithic solvers
DuMu$^X$ - DUNE for Multi-\{Phase, Component, Scale, Physics, ...\} flow and transport in porous media
An open-source simulator and research code in modern C++

https://dumux.org/
Finite volumes

Vertex-centered, face-centered, cell-centered, ...

Control volume finite element schemes

Combine FE functions and control volumes

Local mass conservation by construction

Unstructured meshes
Finite volumes / Control volume finite element schemes

Recipe

Nodal basis function (some FE space)

Construct control volumes “around” nodal degrees of freedom

Element-wise assembly
Finite volumes / Control volume finite element schemes

Recipe

Nodal basis function (some FE space)

Construct control volumes “around” nodal degrees of freedom

Element-wise assembly
Finite volumes / Control volume finite element schemes

Recipe

Nodal basis function (some FE space)

Construct control volumes “around” nodal degrees of freedom

Element-wise assembly

SCVs are associated with dofs
Finite volumes / Control volume finite element schemes

Interpretation as Petrov Galerkin FEM

control volumes \( K \in \mathcal{T} \)

\[
C_K(x) := \begin{cases} 
  c_K \in \mathcal{P}^0(K) & x \in K, \\
  0 & x \notin K,
\end{cases}
\]

\[
B_h(\mathcal{T}) = \left\{ q_h \in L^2(\Omega) : q_h = \sum_{K \in \mathcal{T}} C_K \right\}
\]

\[
\int_{\Omega} r q_h \, dx = 0 \quad \forall q_h \in B_h(\mathcal{T}) \iff \int_K r \, dx = 0 \quad \forall K \in \mathcal{T}.
\]
Finite volumes / Control volume finite element schemes

Other trial function spaces
Finite volumes / Control volume finite element schemes

Other trial function spaces
Finite volumes / Control volume finite element schemes

Other trial function spaces

CV

\( P_1^{+B} \)

\( P_1^{(-B)} \)

cubic bubble functions
Finite volumes / Control volume finite element schemes

Other trial function spaces

3D?
Faces within elements?

Non-overlapping CVFE

cubic bubble functions
Finite volumes / Control volume finite element schemes

Other trial function spaces

Overlapping CV

Overlapping CVFE

cubic bubble functions

\[ P_1^{+B} \]

\[ P_1^{(-B)} \]
Finite volumes / Control volume finite element schemes

Other trial function spaces
template<class Context> NumEqVector stokesMomentumFlux(const Context& context) const
{
    const auto& element = context.element();
    const auto& fvGeometry = context.fvGeometry();
    const auto& elemVolVars = context.elemVolVars();
    const auto& scvf = context.scvFace();
    const auto& fluxVarCache = context.elemFluxVarsCache()[scvf];

    // interpolate velocity gradient at scvf
    Tensor gradV(0.0);
    for (const auto& scv : scvs(fvGeometry))
    {
        const auto& volVars = elemVolVars[scv];
        for (int dir = 0; dir < dim; ++dir)
            gradV[dir].axpy(volVars.velocity(dir), fluxVarCache.gradN(scv.indexInElement()));
    }

    const auto mu = context.problem().effectiveViscosity(element, fvGeometry, scvf);
    const auto pressure = context.problem().pressure(element, fvGeometry, scvf);

    // compute -(\mu*(\nabla v + (\nabla v)^T) - p)*n*dA
    NumEqVector flux = mv(gradV + transpose(gradV), scvf.unitOuterNormal());
    flux *= -mu * Extrusion::area(fvGeometry, scvf);
    flux.axpy(pressure * Extrusion::area(fvGeometry, scvf), scvf.unitOuterNormal());

    return flux;
}

Stokes equations

\[
\int_{\partial K} [-2\mu \mathbf{D}(\mathbf{v}_h) + p_h \mathbf{I}] \cdot \mathbf{n} \, dA = \int_K f \, dx, \quad \forall K \in T^v,
\]

\[
\int_{\partial K} \mathbf{v}_h \cdot \mathbf{n} \, dA = \int_K q \, dx, \quad \forall K \in T^p,
\]

\[
D(v) = \frac{1}{2} \left( \nabla v + (\nabla v)^T \right)
\]
Finite volumes / Control volume finite element schemes

Other trial function spaces

Hybrid CVFE

Local conservation

test function space

hierarchical split

\[ P_2 = P_1 + \text{rest} \]
Finite volumes / Control volume finite element schemes

Stable and locally mass- and momentum-conservative control-volume finite-element schemes for the Stokes problem

Martin Schneider*, Timo Koch

https://arxiv.org/abs/2309.00321

Table 3: Convergence study. Boussinesq flow test case (2D) on a Delaunay grid.

| scheme | $h^*$ | $||p - p'||_1$ rate | $k^*$ | $||v_n - w||_1$ rate | $||v_n - w||_{1,y}$ rate | $l_{z}$ |
|--------|-------|----------------------|-------|----------------------|--------------------------|-------|
| $[P_0(1)^3]$ | $h^*$ | 1.0e-01 | 6.51e-01 | - | 6.2e-02 | 4.73e-02 | - | 1.7e+00 | - | 24 |
| $[P_1(1)^3]$ | $h^*$ | 5.7e-02 | 3.06e-01 | 1.34 | 3.4e-02 | 1.28e-02 | 2.21 | 9.4e-01 | 1.03 | 26 |
| $P_1$ | $h^*$ | 3.1e-01 | 1.06e-01 | 1.66 | 1.8e-02 | 3.62e-03 | 1.98 | 4.8e-01 | 1.06 | 26 |
| $h^*$ | $k^*$ | 1.5e-02 | 4.06e-02 | 1.51 | 9.3e-03 | 9.37e-04 | 2.04 | 2.48e-01 | 1.00 | 26 |
| $[P_0(1)^3]$ | $h^*$ | 8.1e-03 | 1.51e-02 | 1.44 | 4.7e-03 | 2.42e-04 | 1.96 | 1.24e-01 | 1.01 | 26 |
| $P_1$ | $h^*$ | 4.0e-03 | 5.48e-03 | 1.48 | 2.4e-03 | 6.10e-05 | 2.00 | 6.25e-02 | 1.00 | 27 |

Control-volume finite element (CVF) discretizations for the Stokes problem

SCV of vertex-centered CV
SCV of element-centered CV
FEM integration region

Timo Koch, University of Oslo
@ Dune Meeting, Dresden 2023
https://dumux.org
**Finite volumes / Cell-centered schemes**

\[ scv = CV \cap \text{Element} \]

**Tpfa**

**Mpfa**

**Staggered**

\[ CV = scv \]
Multidomain
Multidomain
Multidomain “CouplingManager”

/*!
 * \brief returns an iterable container of all indices of degrees of freedom of domain j
 * that couple with / influence the element residual of the given element of domain i
 *
 * \param domainI the domain index of domain i
 * \param elementI the coupled element of domain i
 * \param domainJ the domain index of domain j
 *
 * \note The element residual definition depends on the discretization scheme of domain i
 * box: a container of the residuals of all sub control volumes
 * cc : the residual of the (sub) control volume
 * fem: the residual of the element
 * \note This function has to be implemented by all coupling managers for all combinations of i and j
 */

template<std::size_t i, std::size_t j>
const CouplingStencilType<i, j>& couplingStencil(Dune::index_constant<i> domainI,
                                                  const Element<i>& elementI,
                                                  Dune::index_constant<j> domainJ) const
{

  ...

}

Multidomain “CouplingManager”

```cpp
/*!
 * \brief returns an iterable container of all indices of degrees of freedom of domain j
 * that couple with / influence
 *
 * \param domainI the domain inde.
 * \param elementI the coupled element.
 * \param domainJ the domain inde.
 *
 * \note The element residual deliver a container of the
 *      box: a container of the
 *      cc : the residual of the
 *      fem: the residual of the element
 * \note This function has to be implemented by all coupling managers for
 */

template<std::size_t i, std::size_t j>
const CouplingStencilType<i, j>& couplingStencil(Dune::index_constant<i> d
const Element<i>& element
Dune::index_constant<j> j d

{

    ...

}
```
Multidomain “MultiDomainAssembler”

\[ R(U) = 0 \]

Newton until convergence:

\[ U_{n+1} = U_n - (\partial R / \partial U)^{-1} \cdot R(U_n) \]

Dune::MultiTypeBlockMatrix
Stokes in DuMux multidomain

Prototyping coupled problems

**Stokes equations**

\[
\int_{\partial K} [-2\mu D(v_h) + p_h I] \cdot n \, dA = \int_K f \, dx, \quad \forall K \in \mathcal{T}^v,
\]

\[
\int_{\partial K} v_h \cdot n \, dA = \int_K q \, dx, \quad \forall K \in \mathcal{T}^p,
\]

\[
D(v) = \frac{1}{2} \left( \nabla v + \nabla^T v \right)
\]

**Coupled PDE**

- **Momentum model**
  - \( P_{1}^{+B} \)
  - \( P_{k} \)
  - \( P_{1}^{CR} \)

- **Mass model**
  - \( P_{1} \)
  - \( P_{k-1} \)
  - \( P_{0} \)

**Multiple discretization schemes**
Multidomain “CouplingManager”

Give me coupled variable at integration point

```cpp
auto divU(typename GridGeometry<flowIdx>::LocalView const& fvGeometry,
          typename GridGeometry<flowIdx>::SubControlVolume const& scv) const
{
    const auto& gg = this->problem(mechanicsIdx).gridGeometry();
    const auto elemSol = elementSolution(fvGeometry.element(), curSol(mechanicsIdx), gg);

    const auto gradU = evalGradients(
        fvGeometry.element(),
        fvGeometry.element().geometry(),
        gg, elemSol,
        scv.center()
    );

    double divU = 0.0;
    for (int i = 0; i < gradU.size(); ++i)
        divU += gradU[i][i];
    return divU;
}

auto porePressure(typename GridGeometry<mechanicsIdx>::LocalView const& fvGeometry,
                  typename GridGeometry<mechanicsIdx>::SubControlVolumeFace const& scvf) const
{
    const auto& gg = this->problem(flowIdx).gridGeometry();
    const auto elemSol = elementSolution(fvGeometry.element(), curSol(flowIdx), gg);
    return evalSolution(
        fvGeometry.element(),
        fvGeometry.element().geometry(),
        gg, elemSol,
        scvf.ipGlobal()
    )[0];
}
```