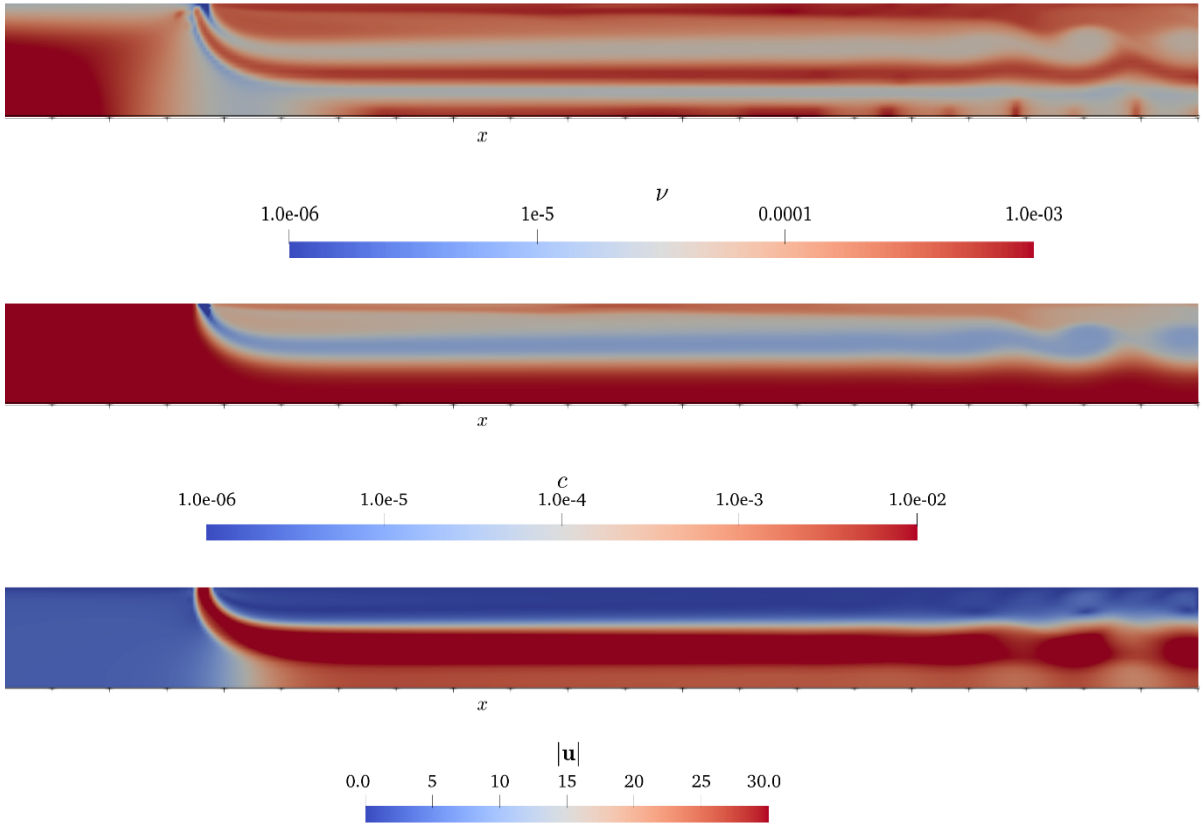
Advanced Viscosity Models in OpenFOAM

Implementation and Stability



Velocity, concentration, and viscosity field in pipe flow with water injection

S. Radl, Angelika Zachl

Institute of Process and Particle Engineering, TU Graz

23 minutes speaking time

Viscosity Model Basics

Graz University of Technology

Non-Newtonian Fluids

Generalized Newtonian Fluid Models

- Power Law Model
- Carreau-Yasuda Model

Standard OpenFOAM
(e.g., OpenFOAM-6/src/transportModels/incompressible)

Linear Viscoelastic Fluid Models

- Maxwell Model
- Jeffreys Model

Nonlinear Viscoelastic Fluid Models

- Giesekus Model
- Phan-Thien & Tanner Model
- FENE-CR Model

Specialized OpenFOAM functionality
(e.g., foam-extend-foam-extend-4.0, fppimenta/rheoTool)

- Solve additional transport equation
- Need stabilization

Viscosity Model Basics

Graz University of Technology

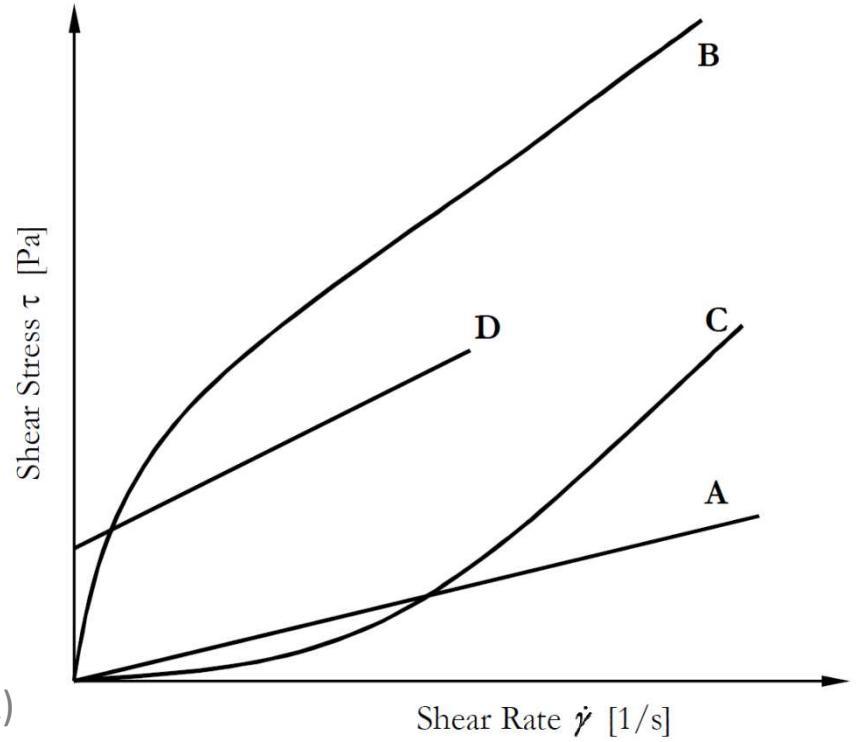
Non-Newtonian Fluids

Generalized Newtonian Fluid Models

- Power Law Model
- Carreau-Yasuda Model

Standard OpenFOAM
(e.g., OpenFOAM-6/src/transportModels/incompressible)

A…Newtonian
B…shear thinning fluid ("pseudoplastic")
C…shear thickening fluid ("dilatant")
D…Bingham plastic



Viscosity Model Basics

Graz University of Technology

Non-Newtonian Fluids

Generalized Newtonian Fluid Models

- Power Law Model
- Carreau-Yasuda Model

Standard OpenFOAM
(e.g., OpenFOAM-6/src/transportModels/incompressible)

$$D = \frac{1}{2} \cdot \left(\nabla \vec{u} + \nabla \vec{u}^{T} \right)$$

$$\dot{\gamma} = \sqrt{2 \cdot tr\left(D^2\right)}$$

Herschel-Bulkley (A+D)
(„yield stress fluids“)

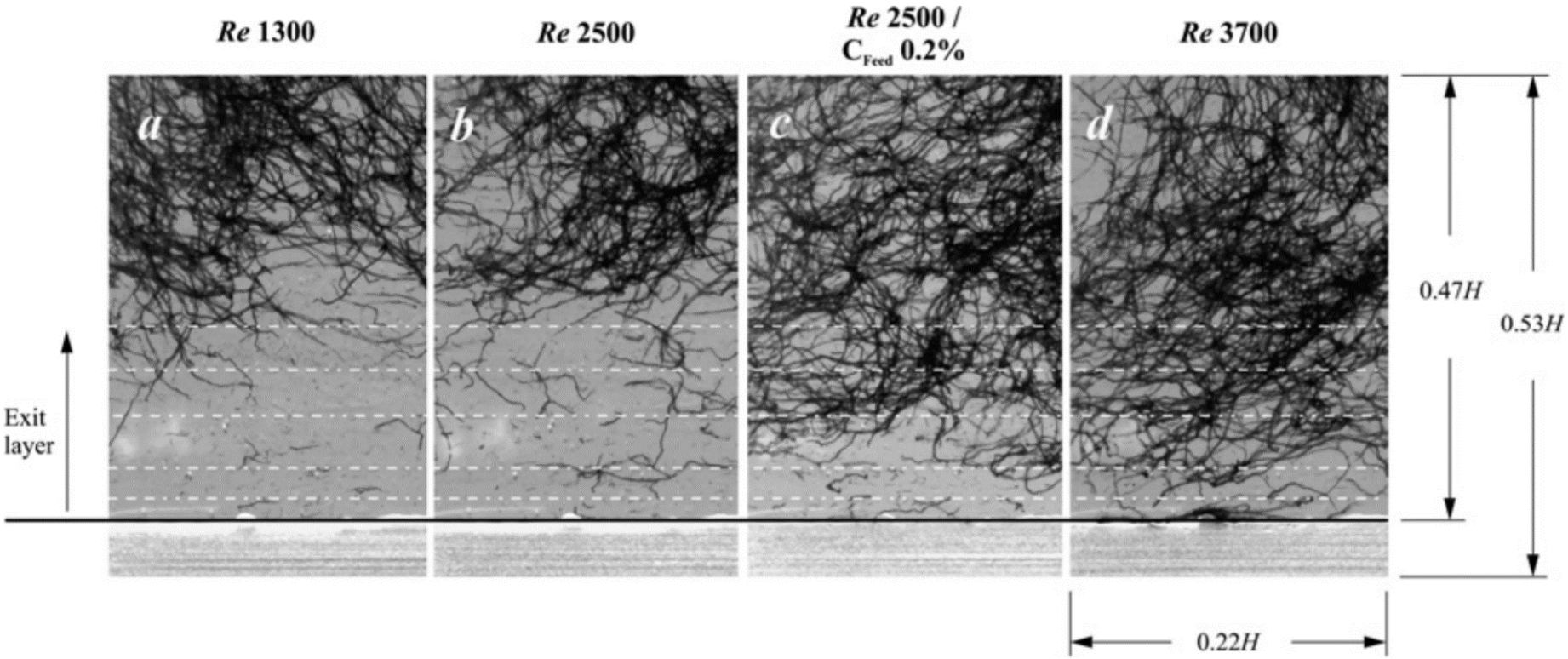
$$\tau\left(\vec{x}\right) = \tau_0 + k\,\dot{\gamma}\left(\vec{x}\right)^n$$

$$\mu\left(\vec{x}\right) = \min\left[\mu_0, \frac{\tau\left(\vec{x}\right)}{\dot{\gamma}\left(\vec{x}\right)}\right]$$

Why Advanced Viscosity Models?

Graz University of Technology

…because fluid viscosity is often a function of a **spatially-variable property** (e.g., the temperature, or the composition)



Redlinger-Pohn et al. (Chem. Eng. Res. Des. 126:54–66, 2017),
Schmid et al. (Nordic Pulp & Paper Research Journal 2019; 34(2): 182–199

Goal & Expected Results

…couple a simple scalar transport model with a **Herschel-Bulkley viscosity model**.

A **concentration and shear rate dependent** rheological model is available in OpenFOAM 5.x

$$\tau\left(\vec{x},c\right)=\tau_0\left(c\right)+k\left(c\right)\dot{\gamma}\left(\vec{x}\right)^{n(c)}$$

$$\mu\left(\vec{x},c\right)=\min\left[\mu_0,\frac{\tau\left(\vec{x},c\right)}{\dot{\gamma}\left(\vec{x}\right)}\right]$$

Prerequisite

…ensure **you know what you are doing**
(https://**wiki.openfoam.com**/Programming)

Phase 1: Add Scalar Transport Solver

…in controlDict, add:

```
scalarTransportSolver_1
{
    functionObjectLibs ("libfuncSolversIPPTFunctionObjects.so");
    type                scalarTransportIPPT;
    Schmidt             1e99;
    SchmidtTurb         0.7;
    schemesField        s;
    solutionField       s;
    field               "c";

    fvOptions
    {
    }
}
```

…a slightly „pimped“ solver

…add initial field for “c”, as well as modify fvSolution and fvSchemes

Phase 1: Add Scalar Transport Solver

…our “pimped” scalar transport solver does the following

```
//*****************************************
// Helper function to calculate local diffusivity
...scalarTransportIPPT::D(phi)
{
    return model.nu()/Schmidt_ + model.nut()/SchmidtTurb_
}

//*****************************************
// Calculate the diffusivity
volScalarField D(this->D(phi));

// Assembe Eqn & Solve
fvScalarMatrix sEqn
        (
            fvm::ddt(rho, s_)
          + fvm::div(phi, s_, divScheme)
          - fvm::laplacian(D, s_, laplacianScheme)
         ==
            fvOptions_(rho, s_)
        );
```

Phase 2: Implement viscosity model

…simply **copy-paste-rename** Herschel-Bulkley model from OpenFOAM src code

… in constructor add **ability to read name of concentration field** (you may want to read additional parameters if you like…)
…also add a bool to **avoid run before first time step**

```
Foam::viscosityModels::HerschelBulkleyIPPT::HerschelBulkleyIPPT
(
    const word& name,
    const dictionary& viscosityProperties,
    const volVectorField& U,
    const surfaceScalarField& phi
)
:
    viscosityModel(name, viscosityProperties, U, phi),
    hasBeenCalled_(false),
...
    concName_(HerschelBulkleyIPPTCoeffs_.lookupOrDefault<word>("concName", "C"))
{
}
```

Phase 2: Implement viscosity model

…implement (**your {possibly} secret**) rheological model. You might want to ensure (i) **correct limiting behavior**, (ii) **calculation of negative viscosities**.

```
if(hasBeenCalled_)
{
    const volScalarField& Conc_ = U_.mesh().lookupObject<volScalarField>(concName_);

    return
    (
        min
        (
        nu0_,
        (
          (0.123 * pow(Conc_,2) + 0.123 * Conc_)    //tau0_(Conc_)
          * dimensionedScalar ("oneStress", dimLength*dimLength/dimTime/dimTime, 1.0)

        +  (0.123 * Conc_ + 0.123)
          *dimensionedScalar ("oneVisc", dimLength*dimLength/dimTime, 1.0) //k_(Conc_)

        *  rtone
        *  pow(tone*sr(),
              0.123+(1.0-0.123)*exp(-Conc_ * 0.123)  //n_(Conc_)
            )
        )
       /(max(sr(), dimensionedScalar ("VSMALL", dimless/dimTime, VSMALL)))
        )
    );
}
```

Hint: …instead of “lookupObject” on could possibly find better alternatives (findObject, getObjectPtr…) [thanks to Mark Olesen for the hint]

Phase 3: Activate model

…after letting OpenFOAM know in which library file your viscosity model resides (in controlDict, add:

```
libs ("libUserViscosityIPPT.so");
```

),

you can simply activate in your transportProperties file:

```
transportModel  HerschelBulkleyIPPT;

HerschelBulkleyIPPTCoeffs
{
    concName    "c";
    k [0 2 -1 0 0 0 0]      0.06522;    //Only needed for first time step!
    n [0 0  0 0 0 0 0]      0.1124;     //Only needed for first time step!
    tau0 [0 2 -2 0 0 0 0]   0.1363;     //Only needed for first time step!
    nu0 [0 2 -1 0 0 0 0]    1e-3;
}
```

Adressing Stability Issues

Graz University of Technology

Goal & Expected Results

…auto-adjust time step depending on flow and fluid properties (i.e., impose **Courant and „viscosity number“ limitation**)

…ensure a **stable simulation (dynamically choose time step)**.

A functionObject is available that set the timestep based on a limiting „**viscosity number**“ (in addition to a predefined Courant number).

Prerequisite

…**again**, ensure **you know what you are doing** (https://wiki.openfoam.com/Programming)

Phase 1: Decide how to limit the time step

…the usual **Courant number-based limiter** should remain to be active

$$Co_{\max} = \frac{\Delta t}{\min\left(\Delta x / U\right)}$$

$$Co_{\max} = \frac{\Delta t}{\min\left(\Delta V / \left[0.5 \sum_{i \in faces} |\phi_i|\right]\right)}$$

…**in addition, a „viscosity number"-based limitation** should kick in in high viscosity regions

$$t^{*,visc} = \frac{\Delta t}{\min\left(\Delta x^2 / \nu_{eff}\right)}$$

Adressing Stability Issues

Graz University of Technology

Phase 2: Implementation as functionObject

…derived from class „fvMeshFunctionObject“, and similar to „setTimeStep“

…allows setting a single value for the time step (or $t^{*,visc}$) **or a table (change time step during start up of simulation)**

```
//- Time step function (single value or table: this is why it is
implemented like this)
autoPtr<Function1<scalar>> timeStepPtr_;
autoPtr<Function1<scalar>> viscDimLessTimePtr_;
```

…takes the **minimum of the Courant number-based time step**, a predefined **minimum time step, as well as the $t^{*,visc}$-based time step**

…uses relaxation to avoid rapid changes of time step

Adressing Stability Issues

Graz University of Technology

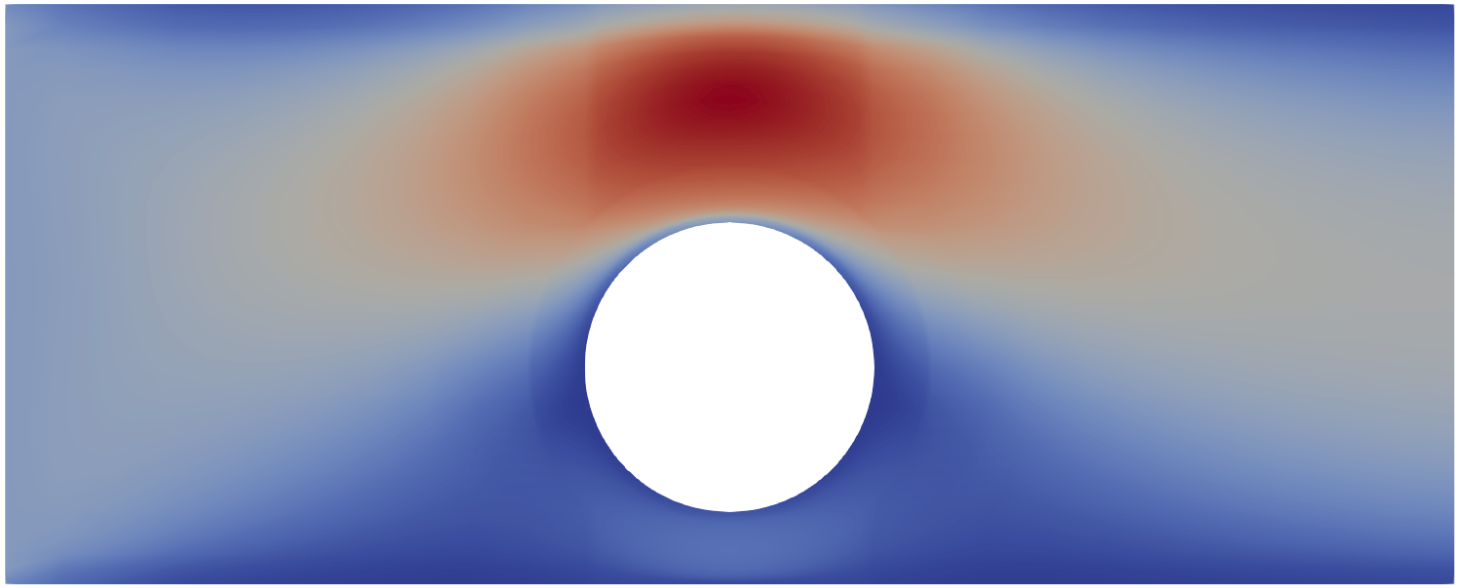
Phase 2: Implementation as functionObject

```
bool Foam::functionObjects::setTimeStepViscAutoFunctionObject::adjustTimeStep()
{
    if(!hasBeenExecuted_) return false;

    scalar targetTimeStep =  min(
                                timeStepPtr_().value(time_.timeOutputValue()),
                                min( minViscTime_*viscDimLessTimePtr_().value(time_.
                                timeOutputValue()),
                                time_.deltaT().value())
                            );

//    Info << "time step before adjustment:   " << currentTimeStep_ << endl;
//    Info << "target time step:              " << targetTimeStep << endl;
    currentTimeStep_ = currentTimeStep_*(1.0-relaxationTimeFactor_)
                     + relaxationTimeFactor_*targetTimeStep;
    const_cast<Time&>(time()).setDeltaT
    (
        currentTimeStep_,
        false
    );

    return true;
}
```

Graz University of Technology

Case 1 - offsetCylinder

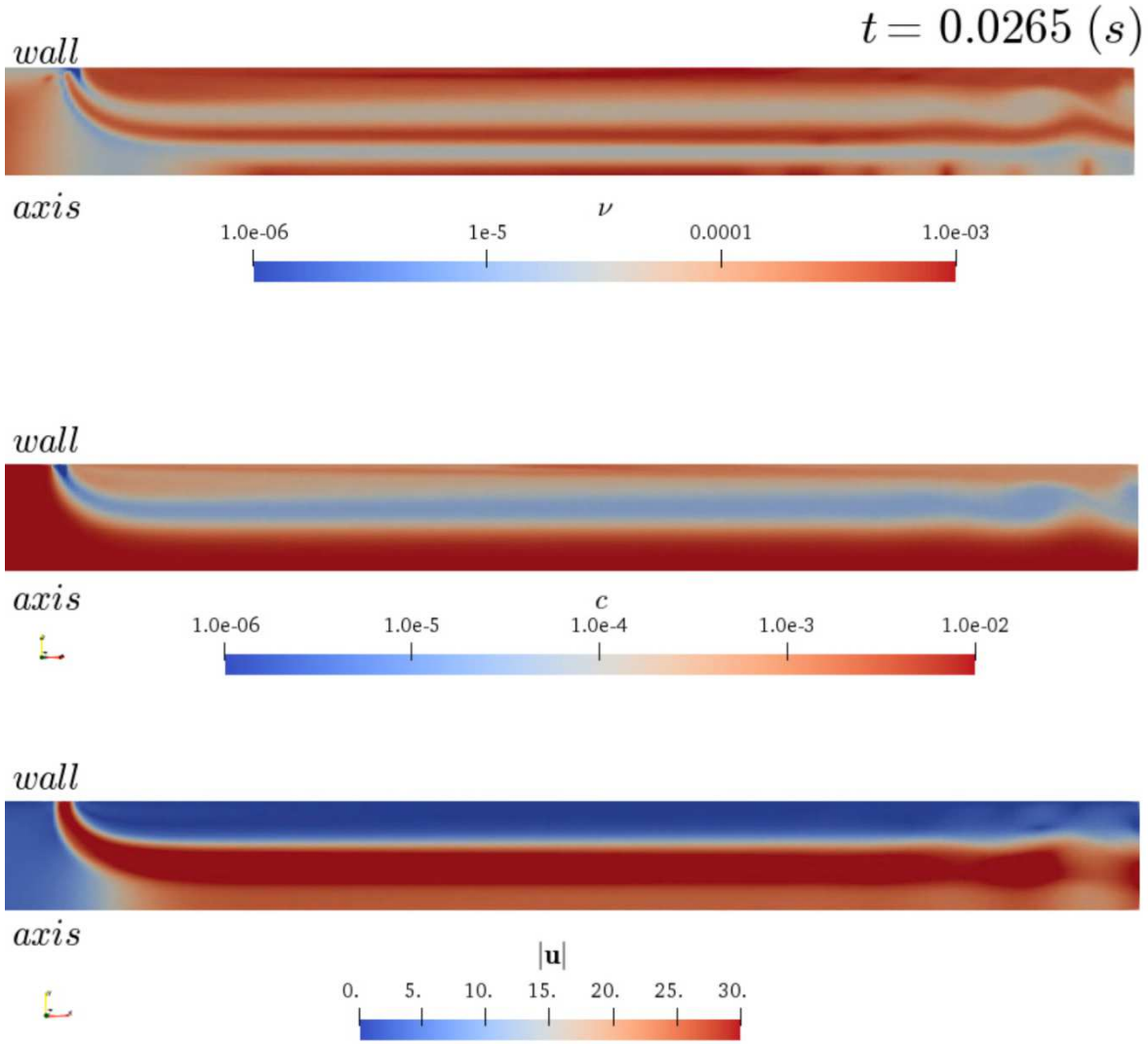
…steady-state flow of a power-law fluid („CrossPowerLaw“) with n=m=1, nu0 = 10, nuInf=0. 1



Graz University of Technology

Case 2 - Pipe Flow with Wall Water Injection

…axisymmetric, plug flow inlet profile, radial water injection



Results for
U_{inj} = 30 [m/s]

Conclusions

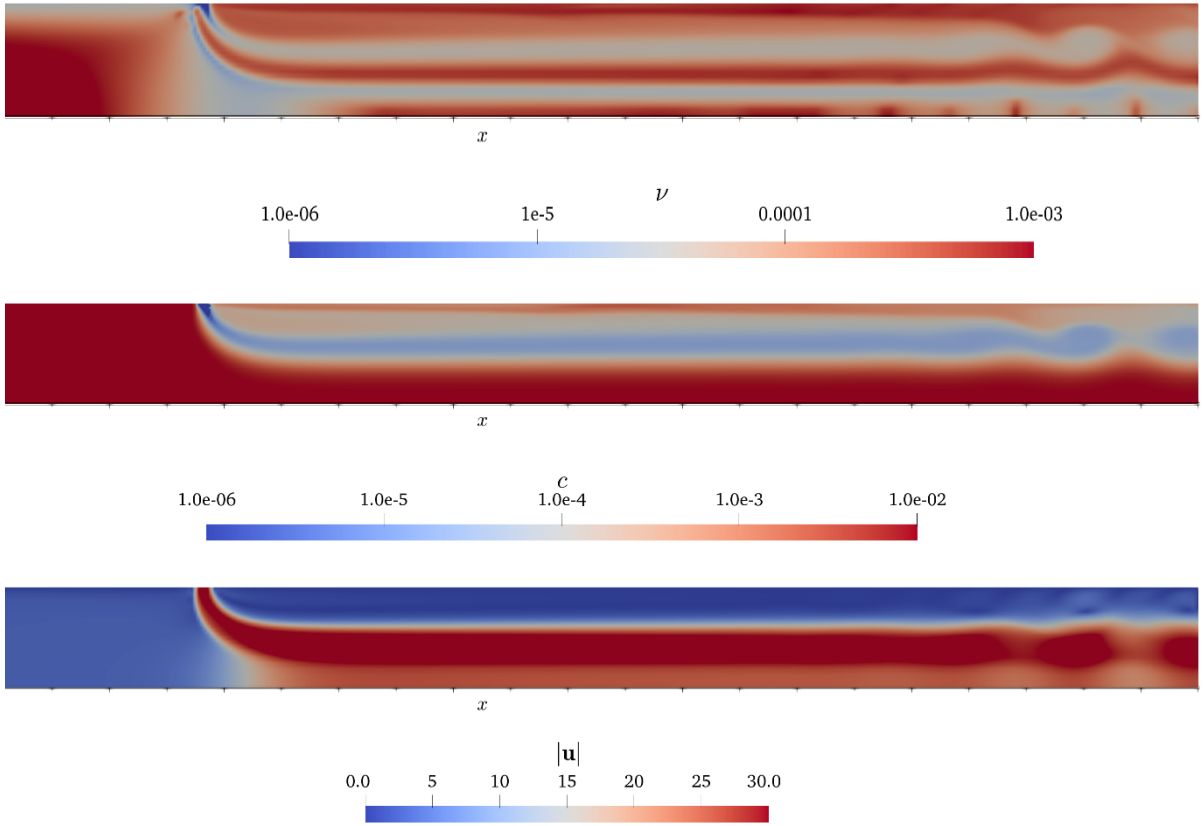
Graz University of Technology

- No new solver - **pimpleFoam** is great!
- Use functionsObjects as long as there is no **complex back coupling**.
- Automatic time stepping control is easy to realize – use ‚**setTimeStep**‘ as a template
- https://**wiki.openfoam.com**/ViscousTimeStep Limitation_by_Stefan_Radl to **access files**

Advanced Viscosity Models in OpenFOAM

Implementation and Stability

S. Radl, Angelika Zachl
Institute of Process and Particle Engineering, TU Graz



Velocity, concentration, and viscosity field in pipe flow with water injection

23 minutes speaking time