CFD with OpenSource software

A course at Chalmers University of Technology

Taught by Håkan Nilsson

Project Work:

Non-Newtonian Models in OpenFOAM

Implementation of a non-Newtonian model

Developed for OpenFOAM-2.3.x

Author:
Naser Hamedi

Peer Reviewed by:
Sandra Busch

December 20, 2014

Table of Contents:

1. Summary

In this tutorial, the rheological models in OpenFOAM are presented in detail. The structure of the non-Newtonian models and a step-by-step guide to show how a new non-Newtonian model can be implemented in OpenFOAM is explained in this report.

In addition, the theoretical background of non-Newtonian models is illustrated. The Casson model as a non-Newtonian model which is widely used to model the blood flow is added to the open source code and tested in a rectangular channel.

2. Rheology and non-Newtonian Models in OpenFOAM

2.1. Rheology

Rheology is the study of such materials in between the fluids and the solids in which the plastic deformation is more dominant than elastic deformation under an applied force. Rheological materials show a combination of elastic, viscous and plastic behavior.

Fig. 1 schematically shows the classification of continuum mechanics. Rheology studies the materials with both solids and fluids properties:

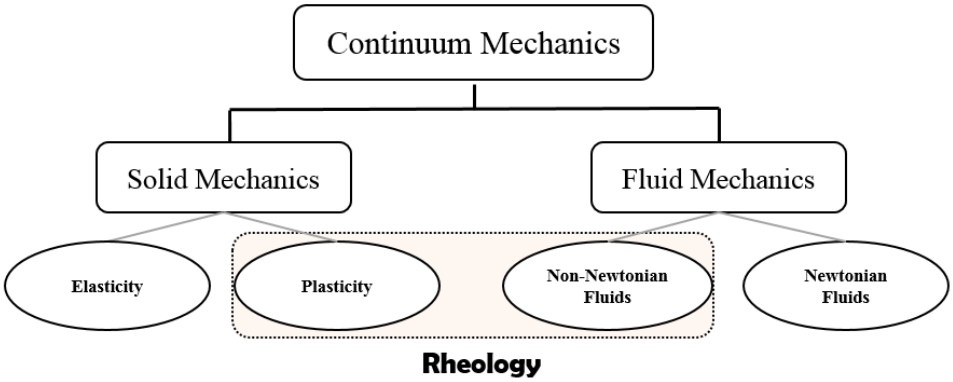

Figure 1: Rheology is study of materials with both solids and fluids properties

The rheological materials usually have complex micro-structure and behavior. Some examples are blood, toothpastes and paints. The fluid materials are mainly classified into two categories, Newtonian and non-Newtonian fluids. In Newtonian fluids, the strain shear rate ($\dot{\gamma}$) is linearly dependent to shear stress (τ) with a constant coefficient that is called viscosity (μ).

$$\tau = \mu\dot{\gamma} \qquad (1)$$

In non-Newtonian fluids, the relation between the shear stress and shear strain rate is different. The viscosity is not constant and depends on the shear rate (Fig. 2) and/or time (Fig. 3). Most of the fluids in nature have non-Newtonian behavior while the small portion belongs to Newtonian fluids. Mainly, the rheology term is usually used for non-Newtonian fluids.

Non-Newtonian fluids viscosity (η) which is not dependent to time, is defined by different models (Fig. 2):

- Shear thickening (dilatant): The viscosity is increased by decreasing the shear strain rate. Some samples are honey and corn starch solution.
- Shear thinning (Pseudoplastic): The fluid in which the viscosity is decreased by increasing the shear rate. Some samples are human blood, yogurt.

- Bingham plastic: The shear stress has linear relationship with the shear rate, but it needs a finite rate of yield stress to flow. Toothpaste is an example.
- Herschel-Buckley: The relationship between shear stress and shear rate is non-linear and the fluid has yield stress before starting to flow. Starch solution 5% is a sample for this kind of fluid.

It should be noted that there is another category of non-Newtonian fluid in which the time has effect on the viscosity (Fig. 3). On the other hand, the apparent viscosity of time-dependent fluids may increase or decrease with time. There are two categories of these fluids, thixotropic and rheopectic fluids. Thixotropic fluids show shear thinning behavior and rheopectic fluids show shear thickening behavior, but they change with the kinematic history of the sample. Some types of honey shows thixotropic behavior under certain conditions. Printer ink is an example of rheopectic fluid.

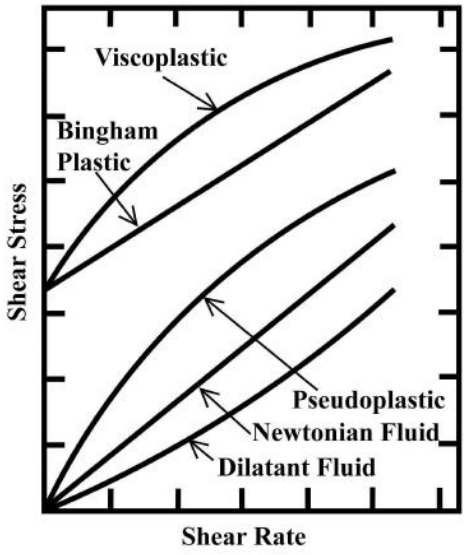

Figure 2. Non-Newtonian models: Independent fluids [1]

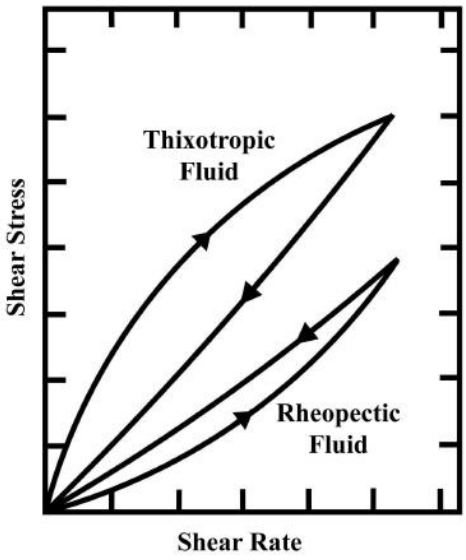

Figure 3. Time dependent fluids [2]

2.2. Non-Newtonian solvers in OpenFOAM

Mainly, there are three solvers in OpenFOAM that solve the flow of non-Newtonian fluids. Table 1 shows these solvers:

Table 1: solvers for non-Newtonian fluids

`SimpleFoam`	Steady state solver for incompressible, turbulent flow of non-Newtonian fluid
`nonNewtonianIcoFoam`	Transient solver for incompressible, laminar flow of non-Newtonian fluid
`PISOFoam`	Transient solver for incompressible, turbulent flow of non-Newtonian fluid

As shown in the table, the difference is that some solvers solve steady state/transient flow and/or laminar/turbulent flow.

non-Newtonian solvers can be found at:

`$FOAM_SOLVERS/incompressible`

2.3. Transport Models

The transport model library in OpenFOAM is classified into two base classes, transport models and viscosity models.

The transport models are located at:

`$FOAM_SRC/transportModels/incompressible`

The viscosity models are located at:

`$FOAM_SRC/transportModels/incompressible/viscosityModels`

There are two implementations of transport model classes, `singlePhaseTransportModel` and `incompressibleTwoPhaseMixer`. It should be noted that the role of the transport models is not transporting the properties. In other words, the viscosity is made accessible by transport models.

2.3.1. `singlePhaseTransportModel`

All of the single phase solvers in OpenFOAM uses `singlePhaseTransportModel` to calculate the viscosity `nu(). nu()` is a public member function and can be accesses in all viscosity models. To specify the viscosity in Newtonian model and the related parameters in non-Newtonian models, the file in `constant/transportProperties` is changed by the user.

2.3.2. `incompressibleTwoPhaseMixer`

The only difference between `incompressibleTwoPhaseMixer` and `singlePhaseTransportModel` is the additional data stored by `incompressibleTwoPhaseMixer`.

2.4. Non-Newtonian models in OpenFOAM

There are four non-Newtonian models implemented in OpenFOAM: `BirdCarreau`, `CrossPowerLaw`, `HerschelBulkley` and `powerLaw`. Table 2, shows the mathematical formulation for non-Newtonian models implemented in OpenFOAM:

Table 2: Mathematical formulation of non-Newtonian models in OpenFOAM.

Non-Newtonian Model	Mathematical Eq.	Coefficient
`powerLaw`	$\eta = K\dot{\gamma}^{n-1}$	K: Consistency index $\dot{\gamma}$: Shear rate n: power law index
`CrossPowerLaw`	$\eta = \frac{\eta_0 - \eta_{inf}}{1+(m\dot{\gamma})^n} + \eta_{inf}$	m: time constant η_0: lower bound viscosity η_{inf}: upper bound viscosity $\dot{\gamma}$: Shear rate
`HerschelBulkley`	$\eta = \tau_y + (k\dot{\gamma})^{n-1}$	τ_y: yield stress k: time constant $\dot{\gamma}$: Shear strain rate
`BirdCarreau`	$\eta = \eta_{inf} + (\eta_0 - \eta_{inf})(1 + (k\dot{\gamma})^{n-1})$	k: time constant η_0: lower bound viscosity η_{inf}: upper bound viscosity $\dot{\gamma}$: Shear rate

`viscosityModels` in OpenFOAM are defined as abstract classes and the above mentioned models as sub-classes. The related function for each non-Newtonian model in OpenFOAM is found in the member function part of the .C files. The respective lines for each model are as follows:

For `powerLaw` model, in powerLaw.C:

```
Foam::tmp<Foam::volScalarField>
Foam::viscosityModels::powerLaw::calcNu() const
{
    return max
    (
        nuMin_,
        min
        (
            nuMax_,
            k_*pow
            (
                max
                (
                    dimensionedScalar("one", dimTime, 1.0)*strainRate(),
                    dimensionedScalar("VSMALL", dimless, VSMALL)
                ),
                n_.value() - scalar(1.0)
            )
        )
    );
}
```

In this model, the viscosity is calculated by selecting the maximum value of minimum value of `nuMax` and `nuMin`. This is a smart way to avoid singularity in calculation of dynamic viscosity. `nuMax` is the maximum value of viscosity in shear thickening fluid in which the power-law index (n) is greater than 1. On the other side, `nuMin` is the minimum value of the viscosity in shear thinning fluid in which the power-law index (n) in less than 1. `k_` is the consistency index in the model.

Below the respective lines for calculation of viscosity in `CrossPowerLaw` model:

```
Foam::tmp<Foam::volScalarField>
Foam::viscosityModels::CrossPowerLaw::calcNu() const
{
    return (nu0_ - nuInf_)/(scalar(1) + pow(m_*strainRate(), n_)) +
nuInf_;
}
```

The calculation of the viscosity for HerschelBulkley, in which the yield stress (`tau0_`) is included in the equation, is shown below. `k_` is the consistency index. The numerical treatment is very similar to the `powerLaw` model implementation.

```
Foam::tmp<Foam::volScalarField>
Foam::viscosityModels::HerschelBulkley::calcNu() const
{
    dimensionedScalar tone("tone", dimTime, 1.0);
    dimensionedScalar rtone("rtone", dimless/dimTime, 1.0);

    tmp<volScalarField> sr(strainRate());
  return
    (
        min
        (
            nu0_,
            (tau0_ + k_*rtone*pow(tone*sr(), n_))
           /(max(sr(), dimensionedScalar ("VSMALL", dimless/dimTime,
VSMALL)))
        )
    );
}
```

`BirdCarreau` in `BirdCarreau.C` is defined as below:

```
Foam::tmp<Foam::volScalarField>
Foam::viscosityModels::BirdCarreau::calcNu() const
{
    return
        nuInf_
      + (nu0_ - nuInf_)
       *pow(scalar(1) + pow(k_*strainRate(), a_), (n_ - 1.0)/a_);
}
```

In the current project, Casson model is added to OpenFOAM.

2.5. Shear strain rate

The shear strain rate ($\dot{\gamma}$), is defined by using I_2, the second invariant matrix which is a scalar in non-Newtonian fluids and is dependent on the rate of the strain tensor, and not dependent on the coordinate system [3]:

$$I_2 = \sum_{i=1}^{3}\sum_{j=1}^{3}\dot{\gamma}_{ij}\dot{\gamma}_{ij}$$

Using above Eq., corresponding shear rate can be calculated by Eq. below.

$$\dot{\gamma} = \sqrt{\frac{1}{2}\sum_{i=1}^{3}\sum_{j=1}^{3}\dot{\gamma}_{ij}\dot{\gamma}_{ij}}$$

It should be noted that the shear strain rate (`strainRate()`) is a member function in `viscosityModel` class which is defined in `viscosityModel.C`:

```
Foam::tmp<Foam::volScalarField> Foam::viscosityModel::strainRate() const
{
    return sqrt(2.0)*mag(symm(fvc::grad(U_)));
}
```

2.6. Casson model

Many models have been developed to describe the blood viscosity. The Casson model is the most widely used one to model the human blood, Eq. below:

$$\sqrt{\eta} = \sqrt{\frac{\tau_y}{\dot{\gamma}}} + \sqrt{m}$$

In the above Eq., η is the viscosity (m^2/s), τ_y is yield stress (m^2/s^2), $\dot{\gamma}$ is shear strain rate (1/s) and m is the consistency index (m^2/s).

3. nonNewtonianIcoFoam

The only difference between `icoFoam` and `nonNewtonianIcoFoam` is the `fluid` (an object) which is defined in `nonNewtonianIcoFoam` to access all member functions and member data of `singlephaseTransportModel. fluid` is defined in createField.H.

The lines regarding the solution of the flow equations in `icoFoam` is as follows:

```
        fvVectorMatrix UEqn
        (
            fvm::ddt(U)
          + fvm::div(phi, U)
          - fvm::laplacian(nu, U)
        );
        solve(UEqn == -fvc::grad(p));
```

The lines regarding the solution of the flow equations in `nonNewtonianIcoFoam` is as follows:

```
        fvVectorMatrix UEqn
        (
            fvm::ddt(U)
          + fvm::div(phi, U)
          - fvm::laplacian(fluid.nu(), U)
        );
        solve(UEqn == -fvc::grad(p));
```

4. Problem definition: Channel Flow

In this tutorial, the flow of the Casson fluid in a rectangular channel is modeled. At first, the grid is imported to OpenFOAM, the boundary conditions are set and the preliminary model for Newtonian

fluid is built. Then, the accuracy of the model is tested against analytical solution. After that, a step by step guide to add the Casson model to the source code is presented and the results are post-processed.

The source code for the whole tutorial is zipped into a folder named ***Casson.tar***. There are several folders in this main folder which are called in different parts of this tutorial. Copy this folder in $FOAM_RUN and unpack it with this command line:

```
tar –xf Casson.tar
```

After entering this command, the folder "Casson" is made with some folders inside.

4.1. Domain

A 2D-channel with dimension of 50×1×0.1 (m^3) was considered as a model for the channel flow (Fig. 4). The length of the channel is considered long enough to have a fully developed velocity profile in the channel in laminar flow.

4.2. Grids

The discretized domain was created in ANSYS ICEM and exported with .msh extension. This extension can be read by ANSYS FLUENT and converted by OpenFOAM. For simplicity the boundary conditions (name and type) are better to be specified in ICEM. The values for each boundary are set later in the related files. The mesh domain had 10 cells per width, 350 cells per length and one cell in depth. Therefore, a coarse mesh was made to have a faster convergence.

The name of the boundaries are INLET, OUTLET and FRONT_AND_BACK.

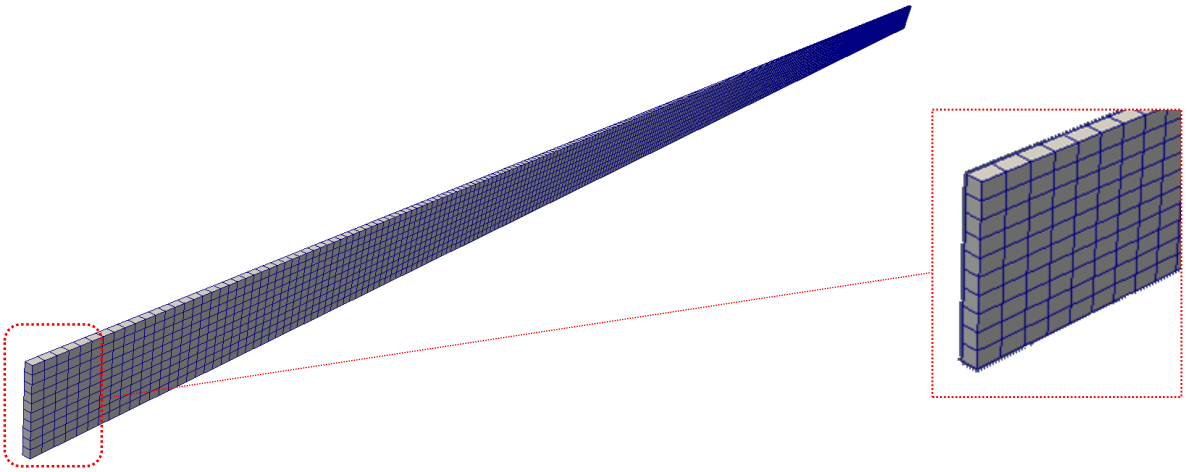

Figure 4. Discretized 2D- channel domain with hexahedral cells

4.3. Importing the mesh

The grid file is located at:

```
$FOAM_RUN/Casson/Grid
```

fluent.msh is the name of the file which was made by ICEM when it is exported in FLUENT format. Before importing the grids from the .msh file, some files should be prepared. controlDict in the system folder can be made as follows:

```
run

mkdir test_grid

cp Grid/fluent.msh test_grid
```

```
cd test_grid

mkdir system //make the system directory which is needed to convert the grids

vi system/controlDict
```

Copy the text in the below box to make the controlDict file which is needed by `fluentMeshToFoam` and save the file.

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     nonNewtonianIcoFoam;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         5;

deltaT          0.0025;

writeControl    runTime;

writeInterval   0.05;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression off;

timeFormat      general;

timePrecision   6;

runTimeModifiable true;

// ************************************************************************* //
```

The command line to convert the meshes into the OpenFOAM format is as follows:

```
fluentMeshToFoam fluent.msh
```

After running this command, some texts are appeared in the terminal window which states that the grid is made successfully. Some information about the number of grids in each patch/boundary is shown too. To view the grids in `ParaFoam` `0` folder should be made in the main root too. Otherwise, the command gives an error. `paraFoam` can skip reading `0` folder if after opening the software, one specify that this directory should not be read.

The `0` folder can be made as follows:

```
mkdir 0
```

```
vi 0/p
```

Paste the texts in the following box into the file and save it:

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      p;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 2 -2 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    INLET
    {
        type            zeroGradient;
    }

    OUTLET
    {
        type            fixedValue;
        value           uniform 0;
    }


    FIXED_WALLS
    {
        type            zeroGradient;
    }


    FRONT_AND_BACK
    {
        type            empty;
    }

}

// ************************************************************************* //
```

To make the boundary conditions in *0/U*:

```
vi 0/U
```

Paste the text in the following box and save the file:

```
dimensions      [0 1 -1 0 0 0 0];

internalField   uniform (0 0 0);

boundaryField
{
    INLET
    {
        type            fixedValue;
        value           uniform (1 0 0);
    }

    OUTLET
    {
        type            zeroGradient;
    }
```

```
    FIXED_WALLS
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }


    FRONT_AND_BACK
    {
        type            empty;
    }

}

// ************************************************************************* //
```

It should be noted that FRONT_AND_BACK boundary condition is changed to "`type wall`" when `fluentMeshToFoam` command is used. Therefore, the type of the boundary condition for this patch should be set to "`type empty`" in:

`constant/polyMesh/boundary`

Another option is using `changeDictionary` utility in OpenFOAM:

`vi system/changeDictionaryDict`

Paste the following lines in the file and save it:

```
FoamFile
{
    version     2.3x;
    format      ascii;
    class       dictionary;
    location    "system";
    object      changeDictionaryDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dictionaryReplacement
{
    boundary
    {
       FRONT_AND_BACK
        {
            type            empty;
        }
    }
}


// ************************************************************************* //
```

`fvScheme` and `fvSolution` in system directory should be added before running the utility:

`vi system/fvSolution`

Paste the following lines into the files:

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
```

```
    location    "system";
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    p
    {
        solver          GAMG;
        tolerance       0;
        relTol          0.1;
        smoother        GaussSeidel;
        nPreSweeps      0;
        nPostSweeps     2;
        cacheAgglomeration true;
        nCellsInCoarsestLevel 10;
        agglomerator    faceAreaPair;
        mergeLevels     1;
    }

    U
    {
        solver          smoothSolver;
        smoother        GaussSeidel; //symGaussSeidel;
        tolerance       1e-05;
        relTol          0;
    }
}

PISO
{
    nCorrectors     4;
    nNonOrthogonalCorrectors 2;
}
```

`vi system/fvScheme`

and paste the text in the below box into the file:

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         Euler;
}

gradSchemes
{
    default         Gauss linear;
    grad(p)         leastSquares;
}

divSchemes
{
    default         none;
    div(phi,U)      Gauss linear;
}
```

```
laplacianSchemes
{
    default         Gauss linear corrected;
}

interpolationSchemes
{
    default         linear;
}

snGradSchemes
{
    default         corrected;
}

fluxRequired
{
    default         no;
    p               ;
}
```

To run the utility:

```
changeDictionary
```

To see the grids in `paraFoam`:

```
paraFoam
```

5. Implementing a new Viscosity Model: Casson

To make a new viscosity model, first a copy of viscosity model e.g. `BirdCarreau` model is made:

```
cd $WM_PROJECT_DIR

cp -r --parents \
src/transportModels/incompressible/viscosityModels/BirdCarreau/ \

     $WM_PROJECT_USER_DIR/

cd $WM_PROJECT_USER_DIR/src/transportModels/incompressible/viscosityModels

mv BirdCarreau Casson

cd Casson

mv BirdCarreau.C Casson.C

mv BirdCarreau.H Casson.H

sed -i s/BirdCarreau/Casson/g Casson.C

sed –i s/BirdCarreau/Casson/g Casson.H
```

We also need Make/files and Make/options:

```
mkdir Make
```

Create Make/files and add Casson model to it:

```
Casson.C
```

```
LIB = $(FOAM_USER_LIBBIN)/libCasson
```

Create Make/options and copy text below into the file:

```
EXE_INC = \
    -I$(LIB_SRC)/transportModels/incompressible/lnInclude \
    -I$(LIB_SRC)/finiteVolume/lnInclude
LIB_LIBS = \
               -lfiniteVolume
```

In Casson.C we add the lines highlighted in blue and comment the lines highlighted in red (or remove them) in `Private Member Functions` part:

```
// * * * * * * * * * * * * * Private Member Functions  * * * * * * * * * * //

Foam::tmp<Foam::volScalarField>
Foam::viscosityModels::Casson::calcNu() const
{
//    return
//        nuInf_
//      + (nu0_ - nuInf_)
//       *pow(scalar(1) + pow(k_*strainRate(), a_), (n_ - 1.0)/a_);
return max
    (
        nuMin_,
        min
        (
            nuMax_,
            pow
            (
                pow(
                          tau0_/max
                    (
                      strainRate(),
                      dimensionedScalar("VSMALL", dimless/dimTime,
VSMALL)
                    ),0.5
                      )
                +pow(m_,0.5)
             ,
                scalar(2.0)
            )
        )
    );
}
```

In `Constructors` part, again comment the red lines and add the blue lines:

```
// * * * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * *
* * //

Foam::viscosityModels::Casson::Casson
(
    const word& name,
    const dictionary& viscosityProperties,
    const volVectorField& U,
```

```
    const surfaceScalarField& phi
)
:
    viscosityModel(name, viscosityProperties, U, phi),
    CassonCoeffs_(viscosityProperties.subDict(typeName + "Coeffs")),
//    nu0_(CassonCoeffs_.lookup("nu0")),
//    nuInf_(CassonCoeffs_.lookup("nuInf")),
//    k_(CassonCoeffs_.lookup("k")),
//    n_(CassonCoeffs_.lookup("n")),
//    a_
//    (
//        CassonCoeffs_.lookupOrDefault
//        (
//            "a",
//            dimensionedScalar("a", dimless, 2)
//        )
//    ),
     m_(CassonCoeffs_.lookup("m")),
     tau0_(CassonCoeffs_.lookup("tau0")),
     nuMin_(CassonCoeffs_.lookup("nuMin")),
     nuMax_(CassonCoeffs_.lookup("nuMax")),

    nu_
    (
        IOobject
        (
            "nu",
            U_.time().timeName(),
            U_.db(),
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        calcNu()
    )
{}
```

In `Member Functions` part, do the same:

```
// * * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * *
* * //

bool Foam::viscosityModels::Casson::read
(
    const dictionary& viscosityProperties
)
{
    viscosityModel::read(viscosityProperties);

    CassonCoeffs_ = viscosityProperties.subDict(typeName + "Coeffs");

//    CassonCoeffs_.lookup("nu0") >> nu0_;
//    CassonCoeffs_.lookup("nuInf") >> nuInf_;
//    CassonCoeffs_.lookup("k") >> k_;
//    CassonCoeffs_.lookup("n") >> n_;
//    a_ = CassonCoeffs_.lookupOrDefault
//    (
//        "a",
//        dimensionedScalar("a", dimless, 2)
//    );
    CassonCoeffs_.lookup("m") >> m_;
```

```
    CassonCoeffs_.lookup("tau0") >> tau0_;
    CassonCoeffs_.lookup("nuMin") >> nuMin_;
    CassonCoeffs_.lookup("nuMax") >> nuMax_;

    return true;
}
```

The key to define the Casson non-Newtonian model is defining the values in which the model function shows singularity (e.g. divided by zero in very low shear strain rate). Therefore we have to define some values for the viscosity in very high value (as `nuMax`) and very low value (as `nuMin`).

In Casson.H, Class Casson Declaration part in the private data section, we add blue lines and comment the red lines (or remove them):

```
    // Private data

        dictionary CassonCoeffs_;

//        dimensionedScalar nu0_;
//        dimensionedScalar nuInf_;
//        dimensionedScalar k_;
//        dimensionedScalar n_;
//        dimensionedScalar a_;
        dimensionedScalar m_;
        dimensionedScalar tau0_;
        dimensionedScalar nuMin_;
        dimensionedScalar nuMax_;

        volScalarField nu_;
```

To compile in the main folder:

```
wmake libso
```

6. Setup the 2D channel flow with Casson model

In this part, we run the channel model that we made in part 4.3 with our new non-Newtonian model. The solver that we use in this part is `nonNewtonianIcoFoam` solver that was explained in part 3.

To do this first we make a copy of `test_grid` directory in the main root and rename it as `test_Casson`:

```
run
```

```
cp -r test_grid test_Casson
```

```
cd test_Casson
```

In `constant` directory, we have to specify `transportProperties`:

```
vi constant/transportProperties
```

In `constant/transportProperties` we insert the text in the below box and we add Casson model parameters to this file.

```
transportModel  Casson;

CassonCoeffs
{
    m             m [ 0 2 -1 0 0 0 0 ] 0.00414;
    tau0          tau0 [0 2 -2 0 0 0 0] 0.0038;
    nuMin         nuMin [0 2 -1 0 0 0 0] 0.0001;
    nuMax         nuMax [0 2 -1 0 0 0 0] 100;
}
```

It should be noted that the related constants are the Casson model coefficients for human blood from [2].

The nonNewtonianIcoFoam solver as a standard solver, should recognize our new non-Newtonian model. Hence, we add the Casson library at the end of `controlDict` file in `system/controlDict`:

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

Other setups are the same as the case which was previously mentioned at the beginning of this report, part 4.3.

To run the solver type:
`nonNewtonianIcoFoam`

The results are shown in Fig. 5 and Fig. 6.

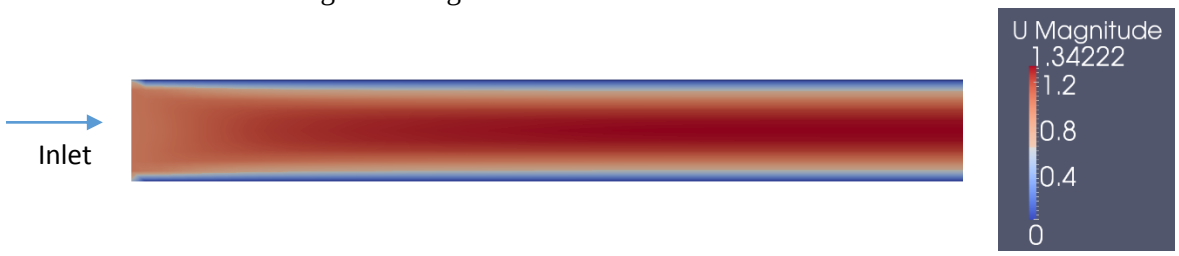

Fig. 5. Velocity distribution in the channel with Casson flow.

The fully-developed velocity profile across the channel for the Casson flow is shown in Fig. 5. This profile is accompanied with a velocity profile with Newtonian flow to compare how the profile is changed in Casson flow. Because of yield stress and shear thinning properties of the flow, the profile is flatter in the centerline in Casson flow while in Newtonian flow it is sharper.
The magnitude of the maximum velocity profile for a Newtonian fluid is $1.5u_{ave}$. As the average velocity in this case is equal to 1 m/s, then the maximum value of velocity profile is u_{max}=1.5 m/s. Our maximum velocity in Newtonian case is very close to this value in Fig. 6. The slight difference belongs to the course mesh that we generated for faster convergence.

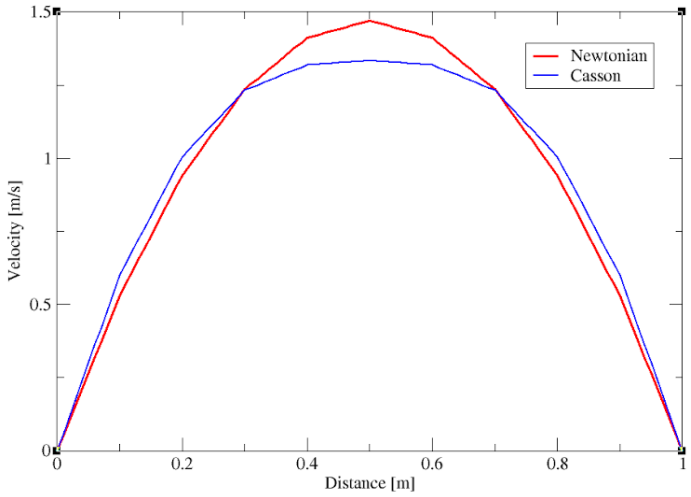

Figure 6. Velocity profile across the channel section in Newtonian and Casson flow

To run the Newtonian case change `constant/transportProperties` according to the below box:

```
transportModel  Newtonian;

nu              nu [ 0 2 -1 0 0 0 0 ] 1;
```

The data is produced by `paraFoam` from the line in the channel section in fully developed area and saved to two different files `velocity_casson.csv velocity_newtonian.csv`.

To plot the lines together in one plot, we can use `xmgrace` by this command:

```
xmgrace velocity_casson.csv velocity_newtonian.csv
```

Other setting e.g. axis labels, legend, etc. can be done in the software.

Reference

[1] Chhabra, R. P. and Richardson, J. F., Non-Newtonian flow in the process industries: Fundamentals and engineering Applications. Butterworth-Heinemann, Oxford, (1999).

[2] Chhabra, Bobbles, drops and particles in non-Newtonian fluids, 2nd edition, Boca Raton, FL: CRC Press, (2006).

[3] Bird, R. B., et. al., Dynamic of polymeric fluids, Vol. 1, 2nd ed., John Wiley and Sons Publication (1987).

[4] W. L. Siauw, E. Y. K Ng, J. Mazumdar, Unstedy stenosis flow prediction: a comparative study of non-Newtonian models with operator splitting scheme, Med. Eng. Phys., 22 (2000) 265-277.