CFD with OpenSource software

A course at Chalmers University of Technology
Taught by Håkan Nilsson

Project work:

Viscoelasticity and Constitutive Relations

Developed for FOAMEXTEND-3.2

Author:
Amith Balasubramanya

Peer reviewed by:
SEBASTIAN Kohlstädt
Håkan Nilsson

January 22, 2016

Learning outcomes

The reader will learn:

- The basics of viscoelasticity
- Constitutive relations in viscoelasticity
- High Weissenberg Number problems(HWNP)
- How HWNP is overcome using stabilizing techniques in OpenFOAM
- The description of viscoelasticFluidFoam solver
- The description of how governing equations are described in OpenFOAM
- How to implement their own constitutive model
- How to switch off DEVSS(Stabilizing scheme) in OpenFOAM
- How to set up a simple case

1 Introduction

This tutorial describes the viscoelasticFluidFoam solver, accompanying constitutive equations, how to edit a constitutive relation and compile it, setting up a simple case and the parameters involved in the same. Some basic theory on high Weissenberg number problems is explained and stability schemes like DEVSS are explained. How this is implemented in foam-extend is also mentioned. A brief theory behind a few constitutive relations and their accompanying equations are explained and the necessary papers are cited.

The primary goal in understanding the viscoelasticFluidFoam solver in this present work is to utilize the solver for solving viscoelastic flows related to biological phenomenon like viscoelastic behaviour of blood, DNA etc. Hence, constitutive relations that help understand biological phenomenon are covered in more detail. Constitutive relations like the Pom-Pom model haven't been applied yet to such flows and is predominantly used for understanding polymer behaviour. The White-Metzner model which is another quasi-linear model and an extension of the Oldroyd-B model isnt covered in this report, however, Larson[7] gives a good idea about it's implementation and relation to Oldroyd-B. The FETA-PTT model isn't covered as linear and exponential PTT models have been explained. High Weissenberg number problems and how they are overcome using the stabilizing scheme DEVSS has been mentioned. Since the report deals with the implementation of constitutive relations in foam-extend, line numbers where the commands and equations are implemented in the code have been mentioned wherever necessary.

2 Theory

This section covers the necessary theory and an introduction to viscoelasticity, governing equations and constitutive relations implemented in foam-extend. The equations described in this section are necessary in understanding how they are implemented in foam-extend.

2.1 Viscoelasticity

Most metals exhibit linear elastic behavior when they are subjected to relatively low stresses at room temperature. They undergo plastic deformations at high stress levels. For an elastic material, the relationship between stress and strain can be expressed in a general form as

$$\sigma = \sigma(\epsilon),$$

This equation states that the normal stress (σ) is a function of the normal strain(ϵ). A similar relationship exists for shear stress(τ) and shear strain(γ). There are a different group of materials like plastics, almost all biological materials and metals at high temperatures that exhibit gradual deformation and recovery when subjected to loading and unloading. The response of such materials is dependent upon how quickly the load is applied or removed, the extent of deformation being dependent upon the rate at which the deformation-causing loads are applied. This time-dependent material behavior is called **viscoelasticity**. A viscoelastic material is one that possesses both fluid and solid properties. The relationship between stress and strain for a viscoelastic material is given by

$$\sigma = \sigma(\epsilon, \dot{\epsilon}).$$

This equation suggests that the stress is not only a function of strain(ϵ), but it is also a function of the strain-rate($\dot{\epsilon}$) and the strain rate is given mathematically as $\dot{\epsilon} = \frac{d\epsilon}{dt}$. Viscoelasticity can be represented by springs and dashpots. The spring is analogous to the elastic behaviour and dashpot is analogous to the viscous behaviour. A simple representation is given in the figure below

2.2 Governing equations

The starting point for the flow analysis of incompressible viscoelastic fluids is the continuity equation and momentum equations . However, our aim is to solve the problem of a flow of non- Newtonian

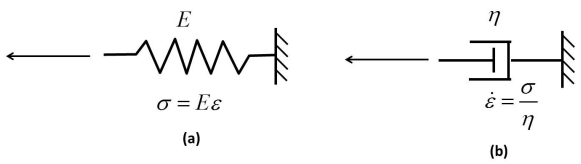

Figure 1: A Simple Spring-Dashpot representation of Viscoelasticity [13]

fluid. This raises some additional problems, since more complex constitutive equations must be solved simultaneously along with the equations of conservation of mass and momentum.
The governing equations for incompressible viscoelastic fluids are the continuity and momentum equations, given by

$$\nabla.u = 0,$$

$$\frac{\partial \rho u}{\partial t} + \nabla.(\rho uu) = -\nabla.p + \nabla.\tau,$$

The split stress tensor approach is incorporated where the stress tensor is split up as a Newtonian component and a non-Newtonian component as

$$\tau = \tau_s + \tau_p,$$

where τ_s which is the solvent or newtonian contribution, defined as

$$\tau_s = 2\eta_s D,$$

and η_s is the Solvent Viscosity and D is the deformation rate tensor, which is given by

$$D = \frac{1}{2}(\bigtriangledown.u + |\bigtriangledown.u|^T),$$

If we substitute this in the momentum equation, we get:

$$\frac{\partial u}{\partial t} + \nabla.(\rho uu) - \eta_s \nabla.(\nabla u) = -\nabla.p + \nabla.\tau_p,$$

τ_p is defined for the multimode form as

$$\sum_{K=1}^{n} \tau_{pk} = 1$$

Here τ_{pk} is a symmetric tensor which is obtained as the sum of the contributions of the individual relaxation modes. The multimode formulation will be used for all models . The use of this formulation makes it possible to obtain more realistic results that are consistent with experimental data. The majority of viscoelastic materials are composed of molecular structures of different sizes (polydisperse) and therefore have different relaxation times. Some important definitions to know before we delve into the constitutive relations are first given
The upper convected derivative is given as:

$$\overset{\bigtriangledown}{\tau_{pk}} = \frac{D\tau_{pk}}{Dt} - [\nabla.u.\tau_{pk}] - [\nabla.u.\tau_{pk}]^T,$$

where $\frac{D\tau_{pk}}{Dt}$ is the material derivative of the extra stress tensor defined as:

$$\frac{D\tau_{pk}}{Dt} = \frac{\partial \tau_{pk}}{\partial t} + u \cdot \ \nabla \tau_{pk}$$

The Gordon-Schowalter derivative of the polymer stress tensor are given by:

$$\overset{\square}{\tau_{pk}} = \frac{D\tau_{pk}}{Dt} - [\nabla u^T.\tau_{pk}] - [\tau_{pk}.\nabla u] + \eta_k(\tau_{pk}.D + D.\tau_{pk})$$

2.3 Constitutive Relations

Constitutive relations are required to calculate the polymeric stress tensor or the non-newtonian component. The equations for the constitutive relations and some accompanying theory is mentioned in this section.

2.3.1 Linear Maxwell

The first equation developed to describe viscoelastic behaviour was proposed by Maxwell [5] where viscoelastic fluids were described as a series of spring and dashpot where the spring is analogous to the elastic behaviour of the fluid and the dashpot the viscous behaviour as already discussed. The equation for the polymeric stress tensor at each relaxation mode is given by

$$\tau_{pk} + \lambda_k \frac{\partial \tau_{pk}}{\partial t} = 2\eta_{pk} D, \tag{1}$$

where λ_k is the relaxation time, η_{pk} is the viscosity at zero-shear rate. More information regarding the linear Maxwell model is given by Maxwell [5]. Bodnar et al [14] gives a good insight into the derivation of the Maxwell model from the Johnson-Segalman Equation.

2.3.2 Oldroyd-B

Oldroyd-B is one of the variants of the Johnson-Segalman model [14]. It is a simple quasi-linear model proposed by Oldroyd [4]. This model is derived from the kinetic theory of concentrated polymer solutions and polymer melts as described by Bird et al [3]. The constants in the equation have the same meaning as described linear maxwell model described previously. This model produces constant values of the shear viscosity relative to strain rate. It also estimates the first normal stress difference (N1) as a quadratic function of shear rate and a second difference normal stress (N2) is zero . The Oldroyd-B model can represent well certain types of fluids that exhibit optimal elasticity. One observation important to note in the Oldroyd-B is that it doesn't take into account the shear-thinning of the fluid. The equation for the Oldroyd-B model is given by

$$\tau_{pk} + \lambda_k . \overset{\triangledown}{\tau} = 2.\eta_{pk}.D, \tag{2}$$

where $\overset{\triangledown}{\tau}$ is the upper convected derivative.

2.3.3 Giesekus

The model proposed by Giesekus [15] results in an equation similar to the ones seen before, but it contains non-linear terms which is given by the product of the polymeric stress tensors. The equation is given by

$$\tau_{pk} + \lambda_k . \overset{\triangledown}{\tau} + \alpha_k . \frac{\lambda_k}{\eta_{pk}} (\tau_{pk} . \tau_{pk}) = 2.\eta_{pk}.D. \tag{3}$$

The constant α_k is called the mobility factor of the relaxation mode k and is associated with the anisotropic Brownian motion or anisotropic hydrodynamic drag [2]. According to Bird et al[3], this model does not produce good results in free shear flows but does a better job predicting shear flows in general compared to the Oldroyd-B model. We observe from the equation that when $\alpha_k = 0$, the model reduces to the Oldroyd-B model. Also, if $0 < \alpha_k < 2$, then $N2 = 0$.

2.3.4 Leonov

Leonov is a special case of Giesekus when $\alpha_k = \frac{1}{2}$. The equation is given by

$$\tau_{pk} + \lambda_k . \overset{\triangledown}{\tau} + \frac{1}{2} \frac{\lambda_k}{\eta_{pk}} (\tau_{pk} . \tau_{pk}) = 2.\eta_{pk}.D, \tag{4}$$

2.3.5 FENE (Finitely Extensible Non-Linear Elastic)

One underlying problem with the linear spring model is that there is no restriction on the deformation of the macromolecules. So, if we have a finite extension restricted by the parameter L^2 , it gives rise to the FENE models first proposed by warner [8]. L^2 is the dimensionless extensibility of the molecules. It is interesting to observe that when $L^2 \rightarrow 0$, the model reduces to the Oldroyd-B model. There are two models of this type:

FENE-P(Finitely Extensible Non-Linear Elastic-Peterlin) proposed by Bird et al [3] is given by

$$(1 + \frac{\frac{3}{1-3/L_k^2} + \frac{\lambda_k}{\eta_{pk}} tr(\tau_{pk})}{L_k^2})\tau_{pk} + \lambda_k \overset{\triangledown}{\tau_{pk}} = 2(\frac{1}{1-3/L_k^2})\eta_{pk}.D, \tag{5}$$

FENE-CR (Finitely Extensible Non-Linear Elastic-Chilcott and Rallison) proposed by Chilcott and Rallison [9].

$$(\frac{L_k^2 \frac{\lambda_k}{\eta_{pk}} tr(\tau_{pk})}{L_k^2 - 3})\tau_{pk} + \lambda_k \overset{\triangledown}{\tau_{pk}} = 2((\frac{L_k^2 \frac{\lambda_k}{\eta_{pk}} tr(\tau_{pk})}{L_k^2 - 3}))\eta_{pk} D \tag{6}$$

2.3.6 PTT (Phan-Thein-Tanner)

A widely used model in numerical simulations of viscoelastic fluids is the PTT model[6] derived from the network theory of concentrated solutions and polymers[3]. There are two ways to write the PTT model.

LPTT

$$(1 + \frac{\epsilon_{pk}\tau_{pk}}{\eta_{pk}} tr(\tau_{pk}))\tau_{pk} + \lambda_k \overset{\square}{\tau_{pk}} = 2\eta_{pk} D, \tag{7}$$

EPTT

$$\exp(\frac{\epsilon_{pk}\tau_{pk}}{\eta_{pk}} tr(\tau_{pk}))\tau_{pk} + \lambda_k \overset{\square}{\tau_{pk}} = 2\eta_{pk} D, \tag{8}$$

$\mathrm{tr}(\tau_{pk})$ takes into account the elastic energy of the network, $\overset{\square}{\tau}_{pk}$ is the Gordon-Schowalter derivative, ϵ_{pk} is a non-linear parameter that is called the extensibility parameter. Opposition to stretch is greater, lower the ϵ_{pk}. The numerical solution is more easily achieved when the fluids have a higher ϵ_{pk}. However ϵ_{pk} is usually less than 1. The parameter $\hat{I}_k$ relates normal stress differences , and it generally has a value close to 0.2 .

2.3.7 Pom-Pom Models

The Pom-Pom model was proposed by McLeish and Larson [16]. With this model, a consistent non-linear behavior is achieved at the same time for shear and elongation. It is based on the reptation theory and in a simplified topology for molecules. The equations for this are given by

$$\tau_{pk} = \frac{\eta_{pk}}{\lambda_{OB_k}}(3 \wedge_{pk}^2 S_{pk} - \delta), \tag{9}$$

$$\overset{\triangledown}{S_{pk}} + 2[D : S_{pk}]S_{pk} + \frac{1}{\lambda_k}[S_{pk} - \frac{1}{3}\delta], \tag{10}$$

$$\frac{D(\wedge_{pk})}{Dt} = \wedge_{pk}[D : S_{pk}] + \frac{1}{\lambda_{sk}}[\wedge_{pk} - 1], \tag{11}$$

where S is the orientation tensor and δ is the unit tensor. λ_{OB_k} is the relaxation time of the backbone tube orientation, which is obtained from the linear relaxation spectrum determined by dynamic measurements. The backbone tube stretch $\wedge_{pk}$ is defined as the length of the backbone tube divided by the length at the equilibrium. The Pom-Pom model however is disadvantageous in a few situations [1] It tends to present a discontinuous solution at high-shear rates, does not

predict the second normal stress difference N2 which gives stability to the system. Because of these disadvantages , various formulations based on the Pom-Pom model were proposed. These maintain the original features of model and try to overcome the disadvantages:

Extended Pom-Pom Model(XPP-SE)

$$\overset{\triangledown}{\tau_{pk}} + \lambda(\tau)^{-1}.\tau_{pk} = \frac{2\eta_{pk}D}{\lambda_{OB_k}} \tag{12}$$

$$\lambda(\tau)^{-1} = \frac{1}{\lambda_{OB_k}}[\frac{\alpha_k\lambda_{OB_k}}{\eta_{pk}}\tau_{pk} + f(\tau)^{-1}\delta + \frac{\lambda_{OB_k}}{\eta_{pk}}(f(\tau)^{-1})\tau_{pk}^{-1} \tag{13}$$

Extended Pom-Pom Model(XPP-DE)

$$\overset{\triangledown}{S_{pk}} + 2[D : S_{pk}]S_{pk} + \frac{1}{\lambda_{OB_k}} \wedge_{pk}^{2} [3\alpha_k \wedge_{pk}^{4} S_{pk}.S_{pk} + (1 - \alpha_k - 3\alpha_k \wedge_{pk}^{4} I_{S.S})S_{pk} - \frac{1-\alpha_k}{3}\delta] = 0 \tag{14}$$

$$\frac{D(\wedge_{pk})}{Dt} = \wedge_{pk}[D : S_{pk}] + \frac{1}{\lambda_{sk}}[\wedge_{pk} - 1]; \tag{15}$$

$$\overset{\triangledown}{S_{pk}} + 2[D : S_{pk}]S_{pk} + \frac{1}{\lambda_k}[S_{pk} - \frac{1}{3}\delta] \tag{16}$$

Double Convected Pom-Pom(DCPP)

$$[(1 - \frac{\xi_k}{2})\overset{\triangledown}{S_{pk}} + \frac{\xi_k}{2}\overset{\triangle}{S_{pk}}] + (1 - \xi_k)[2D : S_{pk}]S_{pk} + \frac{1}{\lambda_{OB_k}\wedge_{pk}^{2}}[S_{pk} - \frac{\delta}{3}] = 0 \tag{17}$$

$$\frac{D(\wedge_{pk})}{Dt} = \wedge_{pk}[D : S_{pk}] + \frac{1}{\lambda_{sk}}[\wedge_{pk} - 1]; \tag{18}$$

$$\tau_{pk} = \frac{\eta_{pk}}{(1 - \xi_k)\lambda_{OB_k}}(3 \wedge_{pk}^{2} S_{pk} - \delta) \tag{19}$$

2.4 High Weissenberg number problem and stability

The Weissenberg number(W_e) is a dimensionless number that is the ratio of the elastic forces to the Viscous forces. Mathematically it is given as $W_e = \lambda\dot{\gamma}$, where, λ is the relaxation time and $\dot{\gamma}$ is the shear-rate. Another dimensionless number which is of importance is the Deborah number (D_e). The Deborah number is defined as the ratio of the relaxation time characterizing the time it takes for a material to adjust to the applied stresses or deformations, and the characteristic time scale of an experiment (or a computer simulation) probing the response of the material. Mathematically it is given as $D_e = \frac{u}{h}$.

While the Weissenberg number is similar to the Deborah number and is often confused with it in technical literature, they have different physical interpretations. The Weissenberg number indicates the degree of anisotropy or orientation generated by the deformation, and is appropriate to describe flows with a constant stretch history, such as simple shear. In contrast, the Deborah number should be used to describe flows with a non-constant stretch history, and physically represents the rate at which elastic energy is stored or released [12].

To understand the High-W_e problem, consider the Oldroyd-B model represented by the Figure 2. Higher the W_e higher the elastic effects as the relaxation time increases, so the material takes higher time to reach back to it's original state. The only term that can balance the exponential blowup is the convection term as seen in the figure. DEVSS is a method that is used to tackle high W_e. In this method an additional diffusion term is introduced on each side of the equation for momentum conservation . The final equation takes the form:

$$\frac{\partial u}{\partial t} + \nabla.(\rho uu) - (\eta_s + \kappa)\nabla.(\nabla u) = -\nabla p + \nabla.\tau_p - \kappa\nabla.(\nabla.u), \tag{20}$$

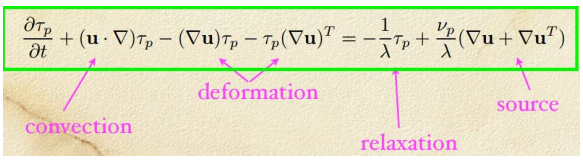

Figure 2: Oldroyd-B equation while explaining High We [11]

Here, κ is a positive number that is related to the parameters of the constitutive model. According to Jovani [1], $k = \eta_p$ is said to be a good choice. We shall show later where this is implemented in foam-extend. It can be seen that the equation for DEVSS is identical to the original equation of motion. The left side of the equation is solved implicitly and the right side the equation is solved explicitly. This method is sometimes found in literature as BSD (Both Sides Diffusion). More about this is explained by Fortin et all [17]

3 viscoelasticFluidFoam Solver

In this section, the implementation of the viscoelasticfluidFoam solver of Foam-extend 3.2 is explained in detail. The .C file of the main solver and each constitutive relations are explained in detail.

3.1 Explaining the solver

The main solver can be accessed in `$FOAM_SOLVERS/viscoelastic/viscoelasticFluidFoam$` The viscoelasticFluidFoam.C is attached in the appendix. The line numbers for viscoelasticFluidFoam.C have been mentioned in Appendix A. The momentum equation is described from line 71 to 80 and is as follows

```
tmp<fvVectorMatrix> UEqn
    (
        fvm::ddt(U)
        + fvm::div(phi, U)
        - visco.divTau(U)
        );

        UEqn().relax();

        solve(UEqn() == -fvc::grad(p))
```

The visco.divTau(U) at line 75 has all the terms related to viscous and elastic portions of the shear-stress and its contents are taken from the file that contains the constitutive models and will be discussed later. The command `UEqn (). relax()` in line 78 makes an explicit relaxation according to a user-supplied value . By equating the negative UEqn with the pressure gradient which is discretized explicitly, the conservation equation is then resolved with the command solve `( UEqn ( ) == - fvc :: grad (p ) )` in line 80 and this is the first(momentum predictor) step of the algorithm which is coupling the pressure and velocity in the PISO algorithm.

Subsequently we have to update the boundary conditions for the pressure with `p.boundaryField() .updateCoeffs()` at line 82. Now the command `rUA= 1.0/UEqn ().()` at line 83 calculates the reciprocal of the diagonal of the coefficient matrix. The `p.relax()` command performs an explicit relaxation of the pressure. `U =- rUA*fvc :: grad(p)` at line 115 applies the correction of velocity. This calculation step of pressure and velocity correction corresponds to step 2 of the algorithm. Finally the `U.correctBoundaryConditions()` command corrects the boundary conditions for velocity at line 116.

The `visco.correct()` at line 118 is responsible for resolving the constitutive equation and also update the τ values for the momentum conservation equation which is step 3 of the algorithm. The

`correct()` function contains information relating to the viscoelastic model and will be discussed later . To achieve a transient solution of better accuracy, steps 1,2 and 3 can be repeated a few times (step 4 of the algorithm).

Equation 20 is the final form of momentum equation that is implemented in foam-extend and lines 71 to 80 represent the first 2 terms of the equation. The remaining terms of the equation are implemented by calling `visco.correct()` at line 118.

3.2 Algorithm

The solver can be summarised based on the algorithm shown below. There are 4 steps in the algorithm for viscoelasticFluidFoam[2]. They are as follows:

1. With the given initial fields of velocity u, stress τ and pressure p , the explicit calculations of the pressure gradient and stress divergence are carried out, and, subsequently, the momentum equation is solved implicitly for each component of the velocity vector, computing a new velocity field estimate u^*.

2. With the new velocity values u^*, the new pressure field p^* is estimated and, subsequently, the correction of velocity is carried out, leading to a new velocity field u^{**} which satisfies the continuity equation. In this step either SIMPLE or PISO algorithm can be used to obtain p^* and u^{**}, with the more accurate PISO being the best option for transient flows.

3. With the corrected velocity field u^{**} the new estimate τ^* for the stress tensor field is calculated by solving the specified constitutive equation.

4. Steps 1,2 and 3 may be repeated recursively within each time step in order to generate more accurate solutions in transient flows. For this, u,p and τ are updated with u^{**}, p^* and τ^*, respectively.

3.3 Constitutive relations

The constitutive relations are located in `$FOAM_SRC/transportModels/viscoelastic/viscoelasticLaws.` The following constitutive relations are found in foam-extend

```
DCPP  FENE-CR  Feta-PTT  Leonov  Maxwell    Oldroyd-B  viscoelasticLaw  XPP_DE
EPTT  FENE-P   Giesekus  LPTT    multiMode  S_MDCPP    WhiteMetzner     XPP_SE
```

The code listing for `Oldroyd-B.C` has been shown in Appendix B with line numbers. The code of the constitutive relations are usually the same except the equation of the polymeric stress tensor at each relaxation mode. The function `divTau( volVectorField & U )` at line 71 is crucial as it is intended to return the value of $\nabla\tau$, where Ï in this case shows the corresponding portions of shear stress of the solvent, the polymeric component and also in this case , the terms relating to DEVSS formulation. The variable `etaPEff` at line 73 stores the value that will be used by DEVSS formulation i.e κ. As mentioned in theory, $\eta_{pk} = \kappa$ is found to be a good value and is hence used. `"div(tau)"` shows how to refer to these terms of the equation in fvSchemes. If we look at the following line of code in `.C` file from line 75 to 80,

```
return
    (
        fvc::div(tau_/rho_, "div(tau)")
      - fvc::laplacian(etaPEff/rho_, U, "laplacian(etaPEff,U)")
      + fvm::laplacian( (etaPEff + etaS_)/rho_, U, "laplacian(etaPEff+etaS,U)")
    );
```

It represents the following equation:

$$\nabla.\frac{\tau_p}{\rho} - \frac{\eta_p}{\rho}\nabla.(\nabla U) + \frac{\eta_s + \eta_p}{\rho}\nabla.(\nabla U) \tag{21}$$

this are the portions corresponding to the divergence of calculated polymer shear stress explicitly using the fvc class , plus the portion corresponding to the Laplacian of the velocity multiplied by η_{pk}. The dominant laplacian guarantess stability through implicit treatment [2]. The shear stress data is calculated using the viscoelastic model. They are passed to the momentum conservation equation for this function . Correct() in line 84 where the constitutive model is defined (Oldroyd-B in the Appendix), is responsible for the remaining terms in equation 20 hence completing the momentum equation. To conclude, equation 21 is the final DEVSS form of the equation used in foam-extend.

Let us look at how the constitutive relations are applied for each constitutive model in Foam Extend.

3.3.1 Linear Maxwell

The equation for Linear Maxwell is from line 93 to 99 in Maxwell.C. The equation is 1 in section 3. This is from foam-extend 3.1. In foam-extend 3.2, the Oldroyd-B equation seems to have been written instead of the Maxwell equation. This might be a bug.

```
// Stress transport equation
    fvSymmTensorMatrix tauEqn
    (
        fvm::ddt(tau_)
     ==
        etaP_/lambda_*twoD
      - fvm::Sp( 1/lambda_, tau_ )
    );
```

The tabular column below explains how each term in the equation is explained in foam-extend.

Mathematical Operator	OpenFOAM implementation
$\frac{\partial \tau_{pk}}{\partial t}$	`fvm::ddt(tau_)`
$2\eta_{pk}D$	`etaP_*twoD`
$\frac{1}{\lambda_k}\tau_{pk}$	`fvm::Sp(1/lambda_,tau_)`

Table 1: Maxwell mathematical operator and code implementation

3.3.2 Oldroyd-B

The equation for Oldroyd-B is from line 96 to 104 in the file `OldroydB.C` and it refers to equation 2.

```
 // Stress transport equation
    fvSymmTensorMatrix tauEqn
    (
        fvm::ddt(tau_)
      + fvm::div(phi(), tau_)
     ==
        etaP_/lambda_*twoD
      + twoSymm(C)
      - fvm::Sp(1/lambda_, tau_)
    );
```

The tabular column below explains how each term in the equation is explained in foam-extend.

Mathematical Operator	OpenFOAM implementation
$\frac{D\tau_{pk}}{Dt}$	`fvm::ddt(tau_)+fvm::div(phi(),tau_)`
$2\eta_{pk}D$	`etaP_*twoD`
$[\tau.\nabla U] + [\tau.\nabla U]^T$	`twoSymm(C)`
$\frac{1}{\lambda_k}\tau_{pk}$	`fvm::Sp(1/lambda_,tau_)`

Table 2: Oldroyd-B mathematical operator and code implementation

3.3.3 Giesekus

The equation for Giesekus is from line 97 to 106 in the file `Giesekus.C` and it refers to equation 3

```
// Stress transport equation
   fvSymmTensorMatrix tauEqn
   (
       fvm::ddt(tau_)
     + fvm::div(phi(), tau_)
    ==
       etaP_/lambda_*twoD
     + twoSymm(C)
     - (alpha_/etaP_)*symm(tau_ & tau_)
     - fvm::Sp(1/lambda_, tau_)
   );
```

The tabular column below explains how each term in the equation is explained in foam-extend.

Mathematical Operator	OpenFOAM implementation
$\frac{D\tau_{pk}}{Dt}$	`fvm::ddt(tau_)+fvm::div(phi(),tau_)`
$2\eta_{pk}D$	`etaP_*twoD`
$[\tau.\nabla U] + [\tau.\nabla U]^T$	`twoSymm(c)`
$\frac{\alpha_k}{\eta_{pk}}(\tau_{pk}.\tau_{pk})$	`(alpha_/etaP_)*symm(tau_&tau_)`
$\frac{1}{\lambda_k}\tau_{pk}$	`fvm::Sp(1/lambda_,tau_)`

Table 3: Giesekus mathematical operator and code implementation

3.3.4 FENE-P

The equation for FENE-P is from line 97 to 109 in the file `FENE_P.C` and it refers to equation 5

```
// Stress transport equation
   fvSymmTensorMatrix tauEqn
   (
       fvm::ddt(tau_)
     + fvm::div(phi(), tau_)
    ==
       (1/lambda_/(1 - 3/L2_))*etaP_*twoD
     + twoSymm(C)
     - fvm::Sp
       (
           1/lambda_ + (3/lambda_/(1 - 3/L2_) + tr(tau_)/etaP_)/(L2_),
           tau_
       )
   );
```

The tabular column below explains how each term in the equation is explained in foam-extend.

Mathematical Operator	OpenFOAM implementation
$\frac{D\tau_{pk}}{Dt}$	`fvm::ddt(tau_)+fvm::div(phi(),tau_)`
$2(\frac{\frac{1}{\lambda_k}}{1-3/L_k^2})\eta_{pk}D$	`(1/lambda_/(1 - 3/L2_))*etaP_*twoD`
$[\tau.\nabla U] + [\tau.\nabla U]^T$	`twoSymm(c)`
$(\frac{1}{\lambda_k} + \frac{\frac{3}{1-3/L_k^2}+\frac{\lambda_k}{\eta_{pk}}tr(\tau_{pk})}{L_k^2})\tau_{pk}$	`fvm::Sp(1/lambda_ + (3/lambda_/(1 - 3/L2_) + tr(tau_)/etaP_)/(L2_),tau_)`

Table 4: FENE-P mathematical operator and code implementation

3.3.5 FENE-CR

The equation for FENE-CR is from line 96 to 108 in `FENE_CR.C` and it refers to equation 6.

```
// Stress transport equation
   fvSymmTensorMatrix tauEqn
   (
       fvm::ddt(tau_)
     + fvm::div(phi(), tau_)
    ==
       ((L2_ / lambda_ + tr(tau_)/etaP_)/(L2_ - 3.0))*etaP_*twoD
     + twoSymm(C)
     - fvm::Sp((L2_/lambda_ + tr(tau_)/etaP_)/(L2_ - 3), tau_)
   );
```

The tabular column below explains how each term in the equation is explained in foam-extend

Mathematical Operator	OpenFOAM implementation
$\frac{D\tau_{pk}}{Dt}$	`fvm::ddt(tau_)+fvm::div(phi(),tau_)`
$2((\frac{L_k^2\frac{\lambda_k}{\eta_{pk}}tr(\tau_{pk})}{L_k^2-3}))\eta_{pk}D$	`((L2_ / lambda_ + tr(tau_)/etaP_)/(L2_ - 3.0))*etaP_*twoD`
$[\tau.\nabla U] + [\tau.\nabla U]^T$	`twoSymm(c)`
$\frac{L_k^2\frac{\lambda_k}{\eta_{pk}}tr(\tau_{pk})}{L_k^2-3})\tau_{pk}$	`fvm::Sp((L2_/lambda_ + tr(tau_)/etaP_)/(L2_ - 3), tau_)`

Table 5: FENE-CR mathematical operator and code implementation

3.3.6 LPTT

The equation for LPTT is from line 98 to 107 in the file `LPTT.C` and it refers to equation 7

```
// Stress transport equation
   fvSymmTensorMatrix tauEqn
   (
       fvm::ddt(tau_)
     + fvm::div(phi(), tau_)
    ==
       etaP_/lambda_*twoD
     + twoSymm(C)
     - zeta_*symm(tau_ & twoD)
     - fvm::Sp(epsilon_/etaP_*tr(tau_) + 1/lambda_, tau_)
   );
```

The tabular column below explains how each term in the equation is explained in foam-extend.

Mathematical Operator	OpenFOAM implementation
$\frac{D\tau_{pk}}{Dt}$	`fvm::ddt(tau_)+fvm::div(phi(),tau_)`
$2\eta_{pk}D$	`etaP_/lambda_*twoD`
$[\tau.\nabla U] + [\tau.\nabla U]^T$	`twoSymm(c)`
$\zeta_k(\tau_{pk}.D + D.\tau_{pk})$	`zeta_*symm(tau_ & twoD)`
$(\frac{1}{\lambda_k} + \frac{\epsilon_{pk}\tau_{pk}}{\eta_{pk}}tr(\tau_{pk}))\tau_{pk}$	`fvm::Sp(epsilon_/etaP_*tr(tau_) + 1/lambda_, tau_)`

Table 6: LPTT mathematical operator and code implementation

3.3.7 EPTT

The equation for EPTT is from line 96 to 109 in the file `EPTT.C` and it refers to equation 8.

```
// Stress transport equation
   fvSymmTensorMatrix tauEqn
   (
       fvm::ddt(tau_)
     + fvm::div(phi(), tau_)
    ==
       etaP_/lambda_*twoD
     + twoSymm(C)
     - zeta_*symm(tau_ & twoD)
     - fvm::Sp
       (
           (1/lambda_)*Foam::exp(epsilon_*lambda_/etaP_*tr(tau_)),
           tau_
       )
   );
```

The tabular column below explains how each term in the equation is explained in foam-extend.

Mathematical Operator	OpenFOAM implementation
$\frac{D\tau_{pk}}{Dt}$	`fvm::ddt(tau_)+fvm::div(phi(),tau_)`
$2\eta_{pk}D$	`etaP_/lambda_*twoD`
$[\tau.\nabla U] + [\tau.\nabla U]^T$	`twoSymm(C)`
$\zeta_k(\tau_{pk}.D + D.\tau_{pk})$	`zeta_*symm(tau_ & twoD)`
$\exp(\frac{\epsilon_{pk}\tau_{pk}}{\eta_{pk}}tr(\tau_{pk}))\tau_{pk}$	`fvm::Sp((1/lambda_)*Foam::exp(epsilon_*lambda_/etaP_*tr(tau_)),tau_)`

Table 7: EPTT mathematical operator and code implementation

3.4 Implementing your own constitutive relations

In this sub-section, we will explain how to implement our own constitutive relation. A simple case of Lower Convected Maxwell (LCM) has been taken into account here. The equation is given by

$$\tau_{pk} + \lambda_k \overset{\triangle}{\tau_{pk}} = 2\eta_{pk}D, \tag{22}$$

where, $\overset{\triangle}{\tau_{pk}}$ is the Lower Convected Derivate given by:

$$\overset{\triangle}{\tau_{pk}} = \frac{D\tau_{pk}}{Dt} + [\nabla u.\tau_{pk}] + [\tau_{pk}.\nabla u^T], \tag{23}$$

Now, we need to copy Oldroyd-B and make a copy of it and create a directory structure. This is done by

```
cd $WM_PROJECT_DIR
cp -r --parents src/transportModels/viscoelastic/viscoelasticLaws/Oldroyd_B $WM_PROJECT_USER_DIR
cd $WM_PROJECT_USER_DIR/src/transportModels/viscoelastic/viscoelasticLaws

Now rename Oldroyd-B as LCM
mv Oldroyd_B LCM
```

We need to create a make directory $|\texttt{cd}WM_PROJECT_USER_DIR/src/transportModels/viscoelastic|\texttt{mkdir}Make|First,$

```
echo "viscoelasticLaws/LCM/LCM.C
LIB= \$(FOAM_USER_LIBBIN)/libmyviscoelasticModels" > Make/files

Next, to create Make/options by pasting the following lines in the terminal
echo "EXE_INC = \\
     -I\$(LIB_SRC)/finiteVolume/lnInclude \\
     -I\$(LIB_SRC)/transportModels/viscoelastic/lnInclude
LIB_LIBS =" > Make/options
```

Now, we need to rename the .C and .H files in our constitutive relation

```
cd viscoelasticLaws/LCM
mv Oldroyd_B.C LCM.C
mv Oldroyd_B.H LCM.H
rm Oldroyd_B.dep
```

In LCM.C and LCM.H, change all occurances of `Oldroyd_B` to LCM so that we have a new class name:

```
sed -i s/Oldroyd_B/LCM/g LCM.C
sed -i s/Oldroyd_B/LCM/g LCM.H
```

Now, we edit the .C file of the new constitutive relation according to equation 22. If we go to LCM.C, from line 96 to 104, replace the equation as follows

```
fvSymmTensorMatrix tauEqn
    (
        fvm::ddt(tau_)
      + fvm::div(phi(), tau_)
     ==
        etaP_/lambda_*twoD
      - twoSymm(C)
      - fvm::Sp(1/lambda_, tau_)
    );
```

NOTE: Now, while solving unsteady problems, DEVSS tends to cause some artifical diffusion and hence can be switched off. To switch off DEVSS, we need to go to the `.C` file and substitute etaPeff=0 and compile the library.

The final step is to compile using:

```
cd ..
cd ..
wmake libso
```

4 Setting up a case

As mentioned in the introduction, the intention of understanding the solver is to setup simulations to understand viscoelastic behaviour of certain biological materials. Though the geometry used is a test-case for polymeric materials, the same geomtery is utilized as a start-up to understand the nuances of the solver. The viscoelastic properties chosen are similar to blood.

Copy the Oldroyd-B tutorial to the local run directory

```
cp -r $FOAM_TUTORIALS/viscoelastic/viscoelasticFluidFoam/Oldroyd_B  $FOAM_RUN
cd $FOAM_RUN/Oldroyd_B
```

4.1 Geometry and Mesh

The geometry used is a 4:1 planar contraction shown in the figure below which is the standard test geometry for polymeric fluids. The diameter upstream is $2H = 0.0254$ m and downstream is $2h = 0.00064$m.

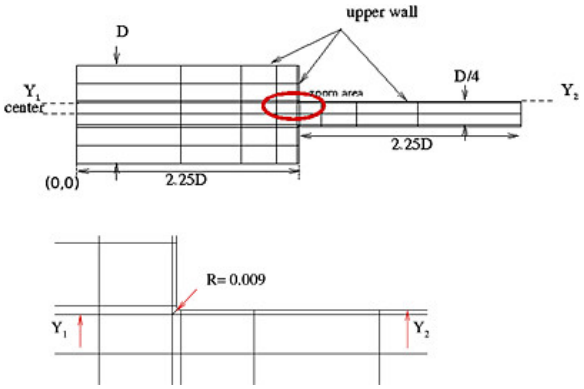

Figure 3: Geometry
[11]

The mesh is created by typing **blockMesh** in the terminal window.

4.2 Initial and Boundary Conditions

Initial Conditions The fluid in the original tutorial is `5.0 wt.% solution of polyisobutylene in tetradecane` which has the following initial conditions $\eta_s = 0.002$ `Pa-S`, $\eta_p = 1.424$ `Pa-S`, $\rho = 803.87097 \frac{kg}{m^3}$ and $\lambda = 0.06$ s

These properties can be accessed at `constant/viscoelasticProperties` This needs to be changed according to the properties of blood like fluid as shown below

```
  rheology
{
    type            Oldroyd-B;
    rho             rho [1 -3 0 0 0 0 0] 1050;
    etaS            etaS [1 -1 -1 0 0 0 0]0.0067;
    etaP            etaP [1 -1 -1 0 0 0 0]0.0017;
    lambda          lambda [0 0 1 0 0 0 0] 0.008;

}
```

Reynold's number is calculated by $R_e = \frac{2\rho u h}{\eta_0}$, where u is the average velocity at the downstream section, $\eta_0 = (\eta_s + \eta_p)$ and Deborah number is calculated by $D_e = \frac{u}{h}$, Where λ is the relaxation time. These are imporant dimensionless numbers in the simulation.

Boundary Conditions

No-Slip at the walls, symmetry at the centerline, Neumann condition at the exit and a prescribed velocity inlet boundary condition is used. The boundary and initial conditions for velocity is changed in `0` directory. The Velocity is calculated using the Reynold's number. In the tutorial, the velocity at the inlet is $0.03875 m/s$. It is changed as shown

```
boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform (0.58 0 0);
    }
    fixedWalls
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    outlet
    {
        type            zeroGradient;
    }
}
```

The boundary conditions for p and tau is the same as the tutorial. Homogeneous Neumann condition is used at the inlet and fixed walls and a constant value at the outlet for pressure.

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

internalField   uniform 0;

boundaryField
{
    inlet
    {
        type            zeroGradient;
    }
    fixedWalls
    {
        type            zeroGradient;
    }
    outlet
    {
        type            fixedValue;
        value           uniform 0;
    }
    simetry
    {
        type            symmetryPlane;
    }
    frontAndBack
    {
        type            empty;
    }
}
```

The Boundary condition for tau is same as that of the tutorial.

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

internalField   uniform (0 0 0 0 0 0);

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform (0 0 0 0 0 0);
    }
    fixedWalls
    {
        type            zeroGradient;
    }
    outlet
    {
        type            zeroGradient;
    }
    simetry
    {
        type            symmetryPlane;
    }
    frontAndBack
    {
        type            empty;
    }
}
```

N1 and N2 are the normal stress differences. As observed, it is shown as calculated which means, after each time step, they are calculated using the **stressDifferences** utility.

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

internalField   uniform 0;

boundaryField
{
    inlet
    {
        type            calculated;
        value           uniform 0;
    }
    fixedWalls
    {
        type            calculated;
        value           uniform 0;
    }
    outlet
    {
        type            calculated;
        value           uniform 0;
```

```
    }
    simetry
    {
        type            symmetryPlane;
    }
    frontAndBack
    {
        type            empty;
    }
}
```

4.3 Setup and Solution

Setup The setup for simulation is in `system/controlDict` and is shown below

```
application     viscoelasticFluidFoam;

startFrom       startTime;

startTime       0.0;

stopAt          endTime;

endTime         1;

deltaT          1e-3;

writeControl    adjustableRunTime;

writeInterval   0.1;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression uncompressed;

timeFormat      general;

timePrecision   6;

graphFormat     raw;

runTimeModifiable yes;

adjustTimeStep on;

maxCo          0.8;

maxDeltaT      0.001;
```

NOTE : If we have to run the simulation for LCM model, we need to add `libs ("myviscoelsticModels.so")` at the end of the`system/controlDict` dictionry in the `systems` folder.

Solution

- In order to run the simulation, we need to type `viscoelasticFluidFoam` in the terminal window.
- After the simulation runs for 1 second, we need to type `stressDifferences` in order to calculate N1 and N2 for each time interval.
- After calculating the Stress Differences, components of shear stress are calculated using `stressSymmComponents`

4.4 Post Processing

The results are visualized using paraView by typing `paraFoam` in the terminal window.

- The Velocity at 1 second is shown below

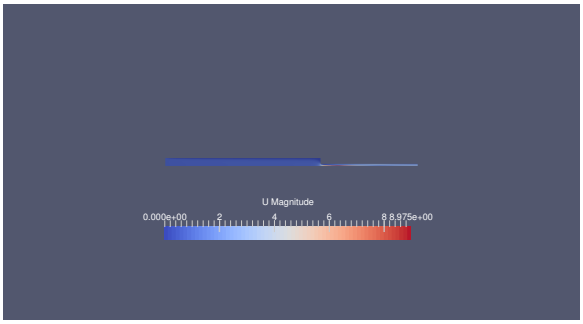

Figure 4: Velocity at 1s

- τ_{xx} at 1 second is shown below

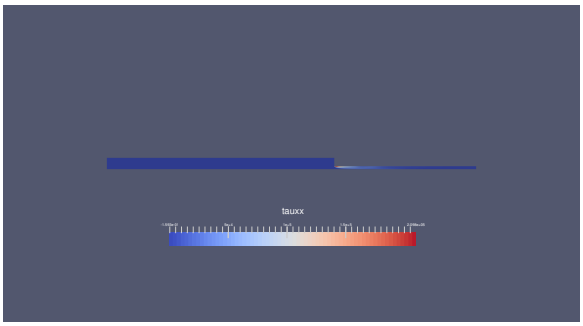

Figure 5: τ_{xx} at 1s

- τ_{xy} at 1 second is shown below
- τ_{yy} at 1 second is shown below

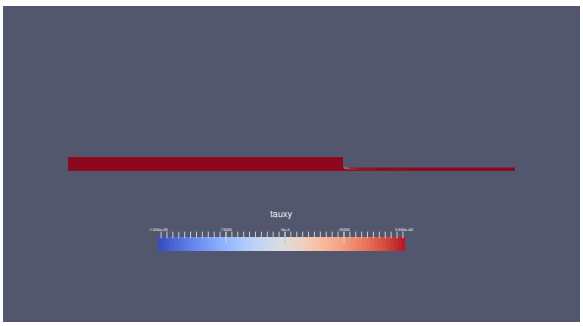

Figure 6: τ_{xy} at 1s

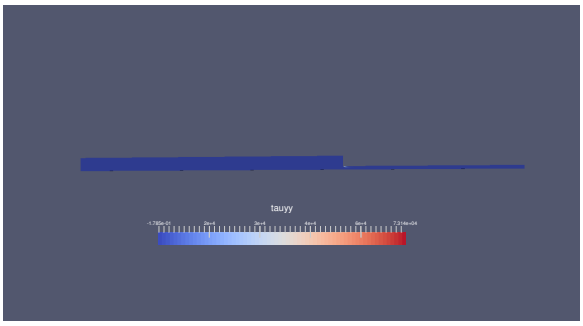

Figure 7: τ_{yy} at 1s

5 Future Work

- Simulate with geometries like Y-bifurcations, T-bifurcations, those that resemble arterial geometries.
- Oldroyd-B though stable doesn't take into account Shear-thinning. A generalized Oldroyd-Model[18] should be implemented in foam-extend
- Once 2D geometries are successful, a 3D model should be simulated.
- Introduce a clot and try simulating.

Study Questions

- How is the high Weissenberg number problem overcome in OpenFOAM?
- What is the value of κ for DEVSS used in OpenFOAM?
- If we need to run the constitutive relation we have created, what line of code should be added and where?
- What are the steps to implement your own constitutive relation?
- What is the purpose of the function `div(tau)`

References

[1] Jovani L Favero. *Simulacao de escoamentos viscoelÃ¡sticos: desenvolvimento de uma metodologia de anÃ¡lise utilizando o software openfoam e equaÃ§Ãµesconstitutivasdiferenciais.*

[2] J. L. Favero1, A. R. Secchi, N. S. M. Cardozo,1, H. Jasak *Viscoelastic flow analysis using the software OpenFOAM and differential constitutive equations.* `Fuel and Energy Abstracts 165(23):1625 $-$ 1636Â· December 2010. DOI:10.1016/j.jnnfm.2010.08.010`

[3] BIRD, R. B.; ARMSTRONG, R. C.; HASSAGER, O. *Dynamics of Polymeric Liquids. 2nd.ed.* New York : John Wiley Sons, Inc., 1987

[4] OLDROYD, J. *Proc. Roy. Soc..* A200 , p. 523 −554 , 1950

[5] MAXWELL, J *J. Phil. Trans. Roy. Soc..* A157, p. 49 p.88, 1867

[6] THIEN, N. P.; TANNER, R. I. *A new constitutive equation derived from network theory.* Journal of Non-Newtonian Fluid Mechanics, v2, n4, p 353 p365, 1977. ISSN 03770257

[7] LARSON, R. G. *Constitutive equations for polymer melts and solutions.* Polymer Engineering, The University of Akron, Akron, OH 44325: Butterworths,1988. 364 p. Boston.

[8] WARNER, H *Ind. Eng. Chem. Fundam.* v. 11, p. 379 p.387, 1972

[9] CHILCOTT, M. D.; RALLISON, J. M. *Creeping flow of dilute polymer solutions past cylinders and spheres.* Journal of Non-Newtonian Fluid Mechanics, v. 29, p. 381 − 432, 1988. ISSN 0377 − 0257.

[10] VERBEETEN,W.M.H.; PETERS,G.W.M.; BAAIJENS, F. P.T. *Differential constitutive equations for polymer melts: The extended pom pom model.* Journal of Rheology, SOR, v. 45, n. 4, p. 823 p.843,2001.—

[11] High Weissenberg Number Problem
`http://www.ma.huji.ac.il/ razk/iWeb/My`$_{S}ite/Research_{f}iles/Visco1.pdf$

[12] Weissenberg Number
`https://en.wikipedia.org/wiki/Weissenberg`$_{n}umber$

[13] *mechanical properties of biological tissues* Ch.15

[14] T. Bodnar, M. Pires and J. Janela *Blood flow simulation using traceless variant of Johnson-Segalman viscoelastic model.*

[15] GIESEKUS, H. *A simple constitutive equation for polymer fluids based on the concept of deformation-dependent tensorial mobility.* Journal of Non Newtonian Fluid Mechanics, v. 11, n. 1-2, p. 69 p109, 1982. ISSN 0377-0257

[16] MCLEISH, T. C. B.; LARSON, R. G. *Molecular constitutive equations for a class of branched polymers: The pom-pom polymer*. Journal of Rheology, SOR, v. 42, n. 1, p.81 p110, 1998

[17] GUÃNETTE, R.; FORTIN, M. *A new mixed finite element method for computing viscoelastic flows*. Journal of Non-Newtonian Fluid Mechanics, v.60, n.1, p.27 p.52,1995. ISSN 03770257

[18] M. Anand , J. Kwack ,A. Masud *A new generalized Oldroyd-B model for blood flow in complex geometries*. International Journal of Engineering Science 72 (2013) 78 88

Appendix A

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | foam-extend: Open Source CFD
   \\    /   O peration     | Version:     3.2
    \\  /    A nd           | Web:         http://www.foam-extend.org
     \\/     M anipulation  | For copyright notice see file Copyright
-------------------------------------------------------------------------------
License
    This file is part of foam-extend.

    foam-extend is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by the
    Free Software Foundation, either version 3 of the License, or (at your
    option) any later version.

    foam-extend is distributed in the hope that it will be useful, but
    WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
    General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with foam-extend.  If not, see <http://www.gnu.org/licenses/>.

Application
    viscoelasticFluidFoam

Description
    Transient solver for incompressible, laminar flow of viscoelastic fluids.

Author
    Jovani L. Favero and Hrvoje Jasak. All rights reserved

\*---------------------------------------------------------------------------*/

#include "fvCFD.H"
#include "viscoelasticModel.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

int main(int argc, char *argv[])
{

#   include "setRootCase.H"

#   include "createTime.H"
#   include "createMesh.H"
#   include "createFields.H"
#   include "initContinuityErrs.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

    Info<< "\nStarting time loop\n" << endl;
```

```
53
54      while (runTime.run())
55      {
56
57  #       include "readPISOControls.H"
58  #       include "readTimeControls.H"
59  #       include "CourantNo.H"
60  #       include "setDeltaT.H"
61
62          runTime++;
63
64          Info<< "Time = " << runTime.timeName() << nl << endl;
65
66          // Pressure-velocity SIMPLE corrector loop
67          for (int corr = 0; corr < nCorr; corr++)
68          {
69              // Momentum predictor
70
71              tmp<fvVectorMatrix> UEqn
72              (
73                  fvm::ddt(U)
74                + fvm::div(phi, U)
75                - visco.divTau(U)
76              );
77
78              UEqn().relax();
79
80              solve(UEqn() == -fvc::grad(p));
81
82              p.boundaryField().updateCoeffs();
83              volScalarField rUA = 1.0/UEqn().A();
84              U = rUA*UEqn().H();
85              UEqn.clear();
86              phi = fvc::interpolate(U) & mesh.Sf();
87              adjustPhi(phi, U, p);
88
89              // Store pressure for under-relaxation
90              p.storePrevIter();
91
92              // Non-orthogonal pressure corrector loop
93              for (int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)
94              {
95                  fvScalarMatrix pEqn
96                  (
97                      fvm::laplacian(rUA, p) == fvc::div(phi)
98                  );
99
100                 pEqn.setReference(pRefCell, pRefValue);
101                 pEqn.solve();
102
103                 if (nonOrth == nNonOrthCorr)
104                 {
105                     phi -= pEqn.flux();
106                 }
```

```
107             }
108
109 #           include "continuityErrs.H"
110
111             // Explicitly relax pressure for momentum corrector
112             p.relax();
113
114             // Momentum corrector
115             U -= rUA*fvc::grad(p);
116             U.correctBoundaryConditions();
117
118             visco.correct();
119         }
120
121         runTime.write();
122
123         Info<< "ExecutionTime = "
124             << runTime.elapsedCpuTime()
125             << " s\n\n" << endl;
126     }
127
128     Info<< "End\n" << endl;
129
130     return(0);
131 }
132
133
134 // ************************************************************************* //
135
```

Appendix B

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | foam-extend: Open Source CFD
   \\    /   O peration     | Version:     3.2
    \\  /    A nd           | Web:         http://www.foam-extend.org
     \\/     M anipulation  | For copyright notice see file Copyright
-------------------------------------------------------------------------------
License
    This file is part of foam-extend.

    foam-extend is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by the
    Free Software Foundation, either version 3 of the License, or (at your
    option) any later version.

    foam-extend is distributed in the hope that it will be useful, but
    WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
    General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with foam-extend.  If not, see <http://www.gnu.org/licenses/>.

\*---------------------------------------------------------------------------*/

#include "Oldroyd_B.H"
#include "addToRunTimeSelectionTable.H"

// * * * * * * * * * * * * * * * Static Data Members * * * * * * * * * * * * * * * //

namespace Foam
{
    defineTypeNameAndDebug(Oldroyd_B, 0);
    addToRunTimeSelectionTable(viscoelasticLaw, Oldroyd_B, dictionary);
}


// * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * * * //

Foam::Oldroyd_B::Oldroyd_B
(
    const word& name,
    const volVectorField& U,
    const surfaceScalarField& phi,
    const dictionary& dict
)
:
    viscoelasticLaw(name, U, phi),
    tau_
    (
        IOobject
        (
```

```
53              "tau" + name,
54              U.time().timeName(),
55              U.mesh(),
56              IOobject::MUST_READ,
57              IOobject::AUTO_WRITE
58          ),
59          U.mesh()
60      ),
61      rho_(dict.lookup("rho")),
62      etaS_(dict.lookup("etaS")),
63      etaP_(dict.lookup("etaP")),
64      lambda_(dict.lookup("lambda"))
65  {}
66
67
68  // * * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * * * //
69
70  Foam::tmp<Foam::fvVectorMatrix>
71  Foam::Oldroyd_B::divTau(volVectorField& U) const
72  {
73      dimensionedScalar etaPEff = etaP_;
74
75      return
76      (
77          fvc::div(tau_/rho_, "div(tau)")
78        - fvc::laplacian(etaPEff/rho_, U, "laplacian(etaPEff,U)")
79        + fvm::laplacian( (etaPEff + etaS_)/rho_, U, "laplacian(etaPEff+etaS,U)")
80      );
81  }
82
83
84  void Foam::Oldroyd_B::correct()
85  {
86      // Velocity gradient tensor
87      volTensorField L = fvc::grad(U());
88
89      // Convected derivate term
90      volTensorField C = tau_ & L;
91
92      // Twice the rate of deformation tensor
93      volSymmTensorField twoD = twoSymm(L);
94
95      // Stress transport equation
96      fvSymmTensorMatrix tauEqn
97      (
98          fvm::ddt(tau_)
99        + fvm::div(phi(), tau_)
100       ==
101          etaP_/lambda_*twoD
102       + twoSymm(C)
103       - fvm::Sp(1/lambda_, tau_)
104     );
105
106     tauEqn.relax();
```

```
107     tauEqn.solve();
108 }
109
110
111 // ************************************************************************* //
112
```